Fractionated Software for Networked Cyber-Physical Systems: Research Directions and Long-Term Vision

Mark-Oliver Stehr, Carolyn Talcott, John Rushby, Pat Lincoln,
Minyoung Kim, Steven Cheung, and Andy Poggio

SRI International
{stehr,clt,rushby,lincoln,mkim,cheung,poggio}@csl.sri.com

Abstract. An emerging generation of mission-critical systems employs distributed, dynamically reconfigurable open architectures. These systems may include a variety of devices that sense and affect their environment and the configuration of the system itself. We call such systems *Networked Cyber-Physical Systems* (NCPS). NCPS can provide complex, situation-aware, and often critical services in applications such as distributed sensing and surveillance, crisis response, self-assembling structures or systems, networked satellite and unmanned vehicle missions, or distributed critical infrastructure monitoring and control.
In this paper we lay out research directions centered around a new paradigm for the design of NCPS based on a notion of software fractionation that we are currently exploring which can serve as the basis for a new generation of runtime assurance techniques. The idea of software fractionation is inspired by and complementary to hardware fractionation — the basis for the fractionated satellites of DARPA's F6 program. *Fractionated software* has the potential of leading to software that is more robust, leveraging both diversity and redundancy. It raises the level of abstraction at which assurance techniques are applied. We specifically propose research in just-in-time verification and validation techniques, which are agile — adapting to changing situations and requirements, and efficient — focusing on properties of immediate concern in the context of locally reachable states, thus largely avoiding the state space explosion problem. We propose an underlying reflective architecture that maintains models of itself, the environment, and the mission that is key for adaptation, verification, and validation.

1 Introduction and Motivation

The increasing availability of systems and devices that can sense and affect their environment in different ways and with different levels of sophistication is the starting point for development of a new generation of *Networked Cyber-Physical Systems* (NCPS). Such systems provide complex, situation-aware, and often safety- or mission-critical services. Examples include traffic control (air and ground), medical systems, smart power grids, flexible manufacturing systems, automated laboratories, microclimate control in buildings, structural monitoring

and control, self-assembling structures or systems, unmanned vehicles (including autonomous robots and UAVs), networked satellite missions (including future fractionated designs), deep space exploration vehicles, instrumented spaces for surveillance and emergency response, and ad hoc combat teams (on the ground and airborne). Especially interesting and challenging examples are complex heterogeneous networked systems with humans and (automomous) agents in the loop, such as vehicular networks, mobile social networks, or the global network of financial markets.

A number of special-purpose solutions exist for different aspects of NCPS. However, general principles and tools for building robust, effective NCPS applications/services using individual cyber-physical devices as building blocks are missing. Furthermore, the verification and validation of NCPS is notoriously difficult and conventional techniques are too expensive, which is a serious problem because the capabilities and the flexibility of NCPS are urgently needed for today's complex mission-critical applications. Factoring out the minimal functionality common to NCPS is a first step toward making verification feasible, because the cost of verification can be amortized over many instantiations of the common framework. This is far from enough, however, because mission-specific properties and performance metrics will require verification, too, and the mission-specific software will typically be much more complex than the minimal framework. Furthermore, conventional verification cannot enable rapid deployment at acceptable cost.

We propose to tackle this problem by considering the notion of software fractionation, which is directly inspired by hardware fractionation, specifically the idea of fractionated satellites [9] that is the basis for DARPA's F6[1] program. We believe that software fractionation has the potential of leading to software that is more robust and can be designed to be verified at reasonable cost by raising the level of abstraction at which verification is applied. We will argue, however, that verification in the conventional sense is not a sensible solution for the flexible, dynamically reconfigurable, mission-critical NCPS, which will lead us to propose new research opportunities in a hardly explored direction of runtime assurance.

Challenges and Opportunities in Networked Cyber-Physical Systems Many challenges exist in the context of NCPS. They have a wide range of assurance requirements, operate in a distributed environment, and unlike pure sensor networks they can perform physical actions and are usually characterized by (distributed) control loops through which the environment provides essential feedback. There is a large overlap between NCPS and wireless sensor networks augmented with actuators, also known as sensor/actor networks [4, 19], but it should be noted that, in NCPS, node and communication capabilities can vary significantly. For instance, in addition to resource-constrained embedded sensor/actuator nodes, devices carried by humans (e.g., PDAs), energy-rich nodes attached to vehicles

[1] Future, Fast, Flexible, Formation-Flying, Fractionated Spacecraft united by Information eXchange

(e.g., laptops), resource-constrained UAVs, solar-powered satellites of different sizes (including pico-satellites such as Cubesats), as well as nodes with continuous Internet connectivity (e.g., ground stations and computationally powerful grid nodes) can all be part of the same NCPS.

In addition to the real-time, resource-limited, reactive aspects of traditional embedded systems, an NCPS must embody a situation awareness that reflects the overall distributed system and its environment. Local situation awareness of a network node is not sufficient. Each node must maintain a model of its local, directly observable situation together with models about the rest of the network. Models must also account for uncertainty, partial knowledge, and bad or stale information. Furthermore, different nodes may have different degrees of awareness according to their capabilities. Asynchronous actions must achieve a desired overall coherent effect. An NCPS needs to be open in the sense that nodes may come and go. In fact, a system may assemble 'on the fly' for a given purpose. Mission-critical systems may be scaled up or down depending on mission requirements.

An advantage of multiple distributed nodes is that resources can be pooled and limitations can be partially overcome by cooperation. To realize the potential benefits of pooling resources (energy, CPU cycles, memory, bandwidth, sensors/actuators) it is necessary for the different processes/layers on each node to adapt resource usage (setting parameters, choosing policies) to achieve system-wide objectives, not just local goals.

From Networked to Fractionated Cyber-Physical Systems The networked structure of NCPS normally arises as a by-product of their required capabilities (e.g., the need to perform distributed sensing) and is usually seen as an inconvenience for engineering, a challenge for verification, and even a hazard to the operation of the system. In this paper, we propose to view distribution as an opportunity (and in some sense as a necessity) rather than an obstacle for building high-assurance systems. In fact, we propose what seems to be counterintuitive — namely, to even further increase the degree of distribution and nondeterminism by moving toward systems that are fractionated by design not only in terms of their hardware but also at the software level. Hardware and software *fractional elements* or *fracments*, as we call them, are very different from traditional components, in that they do not have to correctly perform a well-defined function. Instead, reliable functionality is achieved by a group of such fracments interacting in an opportunistic fashion.

The idea of achieving robustness through diversity and redundancy seems to be a fundamental underlying principle of biological systems. The natural exposure to faults has not only enabled evolution as a mechanism for progress in many dimensions, but has been turned into an advantage by favoring more robust designs. For instance, the human immune system is an example of an effective NCPS. Characteristics of the immune system include robustness, generic and adaptive responses to events, distributed knowledge, diversity, authentication and integrity checking mechanisms, adaptive control, autonomous operation, and heterogeneous actuators. It is a system with continual deployment

of novel entities, intermittent connectivity, exchange of information among heterogeneous entities, such as the nervous and metabolic system components, and uninterrupted operation. There is dynamic optimization, for example, in the crucial balance between quick generic action and deliberate, aggressive specialized actions. The global behavior of the system emerges from predominantly local actions and asynchronous propagation of information.

Diversity and redundancy have also been successfully employed for risk reduction in finance, although the recent financial crisis shows that alone they are not sufficient to prevent systemic failures. Hardware and software fault tolerance is another area where these concepts have been exploited, but their use is mostly coarse grained, with limited degrees of diversity and redundancy, and applied to specific components or subsystems rather than used as an overall design principle. In fault-tolerant or disruption-tolerant networking, the loss of nodes (or connectivity) can be overcome, but a natural question is whether software can be designed so that this tolerance emerges as a special case of more fine-grained general design principles.

Why is this related to verification and validation? The simple answer is that an inherently fault-tolerant architecture raises the level of abstraction to a point where verification and validation becomes interesting and worthwhile. We propose to steer away from low-level code verification and to focus the verification effort mainly on system properties. In our view, code verification is too expensive for what it provides — namely, local correctness properties against detailed and possibly incorrect/incomplete specifications that are based on many assumptions about the environment and the underlying hardware and software. For instance, in challenging environments where failures in processors, memory, networking, sensors, firmware, and drivers are common, the benefit of maximum assurance for just one aspect — namely, the code — is economically questionable. To obtain a precise understanding of the benefits and trade-offs, an economic theory of high assurance design (and possibly beyond) would be needed, for instance along the lines of Rushby's suggested *science of certification* [69] taking into account possible trade-offs between confidence and degree of correctness [8]. A key idea elaborated in [70] is that at some level of abstraction formal methods are able to provide a notion of *possible perfection* enabling compositional arguments about system reliability.

Fractionated software represents a potential paradigm shift, but the high level of abstraction enabled by fractionated design is where the real challenges start. Conventional verification techniques will not be suitable for the mission-driven dynamically reconfigurable cyber-physical systems that we envision in the future. System requirements and configuration are usually not known at design time, which requires us to shift most of the verification activities to the time when sufficient information is available. Typically, this will be after the deployment, that is, at system runtime.

From Design-Time to Runtime Assurance To our knowledge the provocative possibility of just-in-time certification of cyber-physical systems was first raised in [69]. In fact, just-in-time certification is one step beyond just-in-time verifica-

tion in the sense that an explicit certificate is generated at runtime as evidence for system correctness. In general, it may not be necessary to generate an explicit certificate, but the core idea of just-in-time certification — namely, the application of design-time formal methods at runtime — is an opportunity that we suggest exploring systematically. Hence, a few key arguments from the abovementioned paper are worthwhile to summarize. Standards-based certification as it is mostly practiced today in the United States (using a standard such as DO-178B for airborne software) does not provide a clear link between the required artifact and the system requirements. The choice of methods has to rely on extensive expert knowledge and experience, which means that the application to novel circumstances is nearly impossible, making it a barrier to innovation. Usually, the conservative design practices that are required (e.g. limitations on scheduling and memory management) are at odds with innovative architectures such as those needed for today's flexible mission-critical systems. Future systems exist in many configurations, are reconfigurable, and undergo evolution during their lifetime. The number of possible configurations can be enormous (e.g., 50000 lines of XML for an airplane). Since the final configuration is determined after the design and most configurations are never used, just-in-time certification would be a perfectly adequate solution.

We suggest going one step further by looking at systems, like fractionated spacecrafts, that are dynamically reconfigurable and extensible so that the consequent generalization of this idea is to view verification as an ongoing process during the entire lifetime of the system that can be carried out by the system itself. The need for rapid instantiation and deployment of a system for a new possibly unanticipated mission (e.g., within hours) dictates that the verification process must be automated and needs to be executable under critical time and resource constraints. Yet another argument for runtime methods is simply the expectation that future systems will be highly flexible and possibly universal to capture the diversity of possible missions, and the requirements can rarely be stated at design time. A related issue that is often neglected is the validation of specifications, to answer the question if the specification, which will be typically derived from the mission objective, is sensible and captures the intentions. In line with the previous arguments, the most essential validation tasks should also be performed just-in-time — namely, whenever the system interacts with the operator and is tasked with a new mission. The result of a failed validation might mean that the system must be scaled up (e.g., extended) or the objectives need to be scaled down. Clearly, modifications of a mission and changes of the system need to be revalidated, which is why just-in-time validation needs to be, like verification, an ongoing process, which in a similar way takes advantage of (partial) knowledge about the current system configuration.

Overview of this Paper To build systems that satisfy requirements (verify) and perform their intended mission (validate) under a wide range of possible system configurations and with potentially degraded resources, we propose the reflective system architecture depicted in Fig. 1 and outline key research directions. The architecture has three main components: (1) A fractionated software ker-

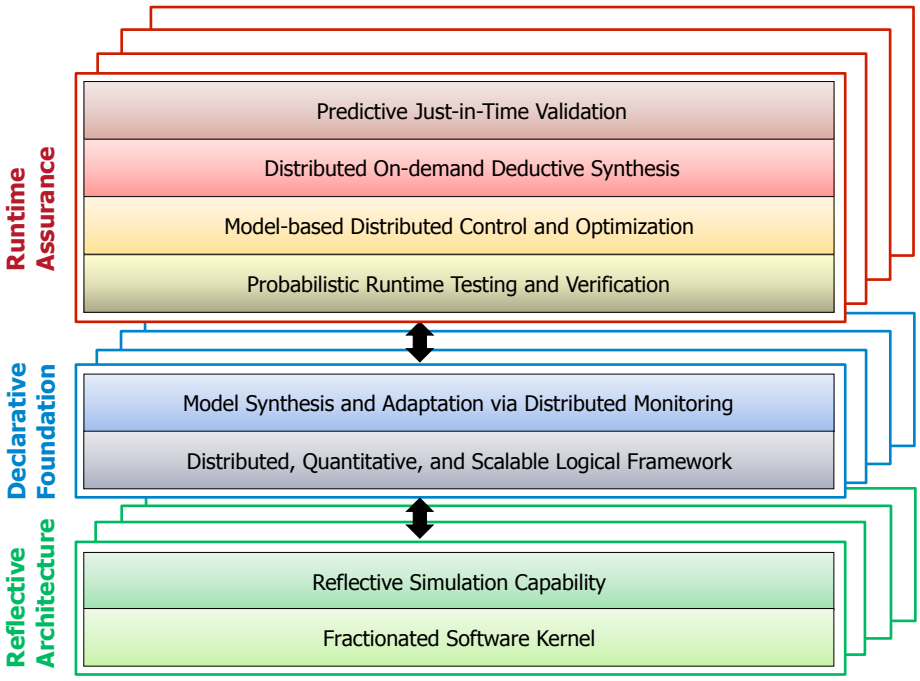

Fig. 1. Stylized System architecture

nel with a reflective simulation capability that is a crucial building block for runtime assurance. (2) A declarative foundation for NCPS in the form of a distributed logical framework that is quantitative and scalable. The logic supports reasoning in the context of system goals and models maintained via distributed monitoring. (3) A new generation of runtime assurance techniques, including novel probabilistic runtime testing and verification methods such as predictive analysis, integration of symbolic and simulation-based techniques, adaptive runtime abstraction, resource- and situation-aware runtime assurance, and learning from the system dynamics. System adaptation through model-based distributed control and optimization is at the core of this set of techniques. We illustrate these ideas using examples centered around fractionated satellite networks, which originally served as an inspiration for the overall approach.

The stylized system architecture illustrates how the different techniques discussed in this paper can work together. For clarity we used a one-dimensional presentation, showing how different levels of runtime assurance can be built on top of each other and ultimately on top of our proposed reflective fractionated software architecture. This is by no means the only way to integrate the different techniques, and not all layers will be equally important or even needed for all NCPS. The architecture clearly distinguishes between the logical framework that provides a declarative view of the NCPS and the runtime assurance layers. Declarative and executable models of the physical world and of the system fragments are maintained and continuously adapted while the system evolves. The

logical framework enables a rich set of possible behaviors of the NCPS, whereas the control and optimization strategy restricts the evolution of the NCPS by exploiting models, invoking runtime assurance techniques, and taking into account the overall system goal, which is represented in the language of the logical framework. On-demand synthesis produces solutions (plans) for complex tasks that require transitioning the system though a series of intermediate goals. Just-in-time validation, finally, is concerned with the validation of mission goals in the context of other applicable policies.

Guided by this architecture, we will address each for the following research directions in a subsequent section, followed by an illustration of these ideas in the domain of fractionated satellites and by a discussion of some related work.

- Software Fractionation
  - provides high level of abstraction from low-level failures
  - leverages diversity, redundancy, distribution, and nondeterminism
  - covers wide spectrum of autonomy and cooperation
- Distributed Logical Foundation for NCPS
  - expresses degrees of satisfaction and uncertainty
  - supports distributed robust dynamic proofs
  - enables adaptive models with multilevel abstraction
- Runtime Assurance with Distributed Declarative Control
  - covers runtime validation, synthesis, verification, testing
  - integrates proof and optimization strategies
  - balances system and assurance goals through agile adaptation

2 A Reflective Architecture for Fractionated Software

To provide a suitable level of abstraction for the new set of runtime verification and related techniques that we propose to explore, we assume that software consists of fracments that are running on top of a fractionated software kernel, a minimalistic framework that enables the fracments to interact. Since a common aspect of all our proposed verification techniques is the capability of the system to analyze its own behavior, we factor out a reflective simulation capability as an intermediate layer directly on top of the kernel.

2.1 Fractionated Software Kernel

Like a biological cell, a software fracment does not have to make sense in isolation, but its interesting properties may emerge only at a higher level of abstraction where multiple fracments interact with each other. The emerging properties are the ones that we are interested in verifying. How they are established is less important as long as we can quantify their probabilities.

By enabling verifiable software, the objectives of the fractionated software kernel are similar to those of a separation kernel [67] and robust partitioning

[68] in integrated modular avionics. The fractionated software kernel, however, is conceptually distributed over many nodes and exploits distribution to achieve a new dimension of decoupling, diversity, and redundancy. Each node could be running a separation kernel, but this may not be necessary in highly fractionated systems where sufficient probabilistic separation guarantees between fracments are provided by their random distribution over the nodes and hence by the distributed nature of the system alone. Like a separation kernel, the fractionated software kernel should be minimalistic so that trust in it can be established once and for all with acceptable cost. The fracments will be required to be self-coordinating so that global coordination, a bottleneck, and a potential point of failure in component-based approaches to fault tolerance, would not be needed.

To exploit distribution, the kernel will also provide minimal but robust networking capabilities so that, disregarding possible networking delays, local fracments can interact with nonlocal fracments just as if they were local. The interaction will be delay tolerant and no upper bounds on delays will be assumed because network disruptions (including intermittent or episodic connectivity) are assumed to be part of the normal operation. To ensure maximal decoupling between the fracments, the interactions will not be direct (unlike remote procedure calls and the message-passing paradigm used for instance in today's service-oriented architectures) but rather enabled by partially ordered knowledge sharing. The idea, which has been successfully used in sensor networks and in disruption-tolerant networking (DTN), specifically in our work [75], is that each node has a local knowledge base, which local fracments can access and which is shared across the network using a peer-to-peer knowledge dissemination protocol. A general framework, which serves as a prototype to experiment with the partially ordered knowledge-sharing model, has been presented in [42] and used as the basis of a distributed logical framework in [74]. This loosely coupled paradigm resembles that of a distributed blackboard (generalizing the well-known blackboard paradigm for multiagent systems) and distributed tuple spaces (e.g., [61]) with the important difference that no global coordination and no consistency guarantees are required. Instead, fracments are engineered to be delay insensitive and tolerant to inconsistent and incomplete knowledge. Similar to the paradigm of content-centric networking [79], the fracments operate at a level of abstraction in what might be called a knowledge-centric approach, where they are not concerned with protocols and message flow (and resulting synchronization problems) but only with the question of how to use the knowledge once it becomes available.

In a fractionated networked system, software fracments solving the same or similar problems will be distributed over the networked nodes with a suitable degree of diversity and redundancy. Various approaches developed for software fault tolerance [78, 54] can be utilized and combined to achieve diversity of fracments, in particular distributed n-way redundancy and n-version design [12]. We additionally propose to use randomization (exploiting both non determinism and concurrency) as an important source of diversity. Individual fracments should be self-checking [54] (in a rigorous sense) but thanks to fracment diversity

and redundancy do not have to be self-correcting. Nevertheless, checkpointing, restart, and recovery block techniques [78, 54] can be used locally at the fracment level to improve robustness, but not at the distributed system level, where inconsistent knowledge is accepted as a normal operating condition. Furthermore, diversity does not have to be confined to the implementation level. Fracments (e.g., parameterized by complexity levels) that can accomplish similar functions with possibly different resource requirements are also desirable and have the advantage of providing not only diversity but also a potentially continuous high-dimensional trade space for system optimization. In spite of each individual fracment being bound to a fixed location, this approach with transparent knowledge sharing leads to location independence of the function provided by the fracments as a group. Note that this approach does not rely on any form of code or agent mobility (such as [61], which comes with its own set of problems, especially in networks that are dynamic and unreliable).

We envision fractionated software to be continuously maintained and evolving at runtime. In fact, complex and expensive future systems (such as fractionated satellites) may not only be deployed incrementally but design and deployment will likely become concurrent incremental activities to enable risk and cost reduction through partial deployment and early testing. Hence, beyond the addition of new nodes, the fractionated software kernel needs to support removal and installation of software fracments on existing nodes without system interruption. Since a new fracment will typically be installed remotely (e.g., from a ground station in the case of a satellite network) and on many nodes, the dissemination of fracments can utilize the same mechanism that is used to disseminate knowledge. Clearly, an asynchronous system cannot be upgraded globally in one step, but with a sufficient amount of diversity and redundancy an incremental distributed upgrade should be possible even without risking the interruption of an ongoing mission.

2.2 Reflective Simulation Capability

To support runtime assurance and related techniques efficiently we envision that a reflective simulation capability will be directly built as a layer on top of the fractionated software kernel. In our context, reflection means that software fracments and their encapsulated hardware are reflected as models that can support reasoning and optimization activities. Models are not constrained to a single level of abstraction. Furthermore, models can be executable and can themselves be viewed as fracments that can be composed to larger models. Executability of models is essential to enable the predictive runtime verification techniques discussed in the next section. Computational reflection is a well-known concept in computer science that has many applications [55]. It has been successfully implemented as part of the Maude system [16]. The importance of runtime reflection as an enabler of traditional (monitoring-based) runtime verification for safety-critical systems has already been recognized [50]. Here, we propose to generalize runtime reflection to open distributed systems, to multiple levels of abstraction,

and to use it systematically as a basis for the implementation of a wide range of runtime assurance techniques.

3 A Declarative Foundation for Cyber-Physical Systems

A logical framework should serve as a uniform declarative interface to all capabilities of the NCPS. At the same time it should provide a semantically well-founded way to represent, manipulate, and share knowledge across the network. The logical framework should also serve as a basis for abstract models that take the form of logical theories and are continuously adapting to new incoming knowledge resulting from local or nonlocal observations.

3.1 Distributed, Quantitative, and Scalable Logical Framework

Various kinds of knowledge need to be expressed including models, facts, goals, and proofs — i.e., derivations of goals from facts. In NCPS, facts can represent sensor readings at specific locations, and goals can represent queries for information or requests to actors or actuators to perform certain actions. Although there are cases where a goal can be directly satisfied by a single local action, it is typically the case that distributed actions are needed and the more relevant feedback will be conveyed via a feedback loop through the environment. Such indirect feedback can consist of facts (representing observations) from multiple sensors that together can measure the progress toward reaching the original high-level goal. As a consequence, rigid top-down or bottom-up approaches are not sufficient for NCPS. Furthermore, models can have many different flavors ranging from precise physical models to qualitative commonsense models, and can include approximate and partial models of the real world based on observations. Combinations of different flavors are usually needed. For instance, a satellite as part of a network could utilize an approximate model for network connectivity combined with a precise orbit model based on Kepler's laws and knowledge gathered by active exploration (e.g., beaconing for neighbor discovery) and passive observations (e.g., attitude determination).

A Logical View of Cyber-Physical Systems Apart from a few notable exceptions such as cyberlogic [66], it is interesting to note that the distributed nature of today's problems is rarely considered in the design of logical frameworks. For cyber-physical systems it is essential, since carrying out proofs may require cooperation across multiple nodes. In many cases, goals and facts cannot be matched locally. Consider an example of gathering certain information from a particular area under observation (e.g., from a sensor network on the ground that is part of a global network of UAVs and satellites). In an interest-driven routing protocol, such as directed diffusion [38], a node expresses interest for specific data by sending requests into the network. Data matching the interest is then drawn toward the node from which the interest originates. From a logical point of view, a goal, representing an information request, is injected and disseminated through

the network. The goal is a logical formula expressing that the information needs to be of the required kind (content subgoal) and be delivered at the requesting node (delivery subgoal). A fact representing the presence of information at the source will match or satisfy part of the goal — namely, the content subgoal. Now there is an incentive to route the partially satisfied goal with the requested content toward the interested application, since this will incrementally increase the *degree of satisfaction* of the overall goal and eventually complete the distributed proof. In other words, an interest-driven routing and many similar processes can be seen as distributed proofs and optimization strategies that try to bring facts and goals together.

To serve as a formal framework for NCPS, the logic must have the capability to express degrees of satisfaction so that both search and optimization become instances of a generalized notion of deduction. As a starting point, we propose to use a version of first-order logic with equality, real arithmetic, and degrees of confidence. The specific application domain will be reflected in the background theory relative to which the reasoning takes place and can also influence the search, reasoning, and optimization strategies employed at the higher layers (see Section 4). Due to the resource-constrained nature of many cyber-physical systems, trade-offs between expressiveness and efficiency need to be considered and a scalable logic — i.e., a logic with sublanguages and inference systems of adjustable complexity — would be the ideal solution. The language needs to go beyond propositional and Horn clause logic, since a functional sublanguage representing cost and utility functions with discrete and continuous parameters and functional parts of the models will be essential. Furthermore, predicates with discrete and continuous parameters are important to support predicate abstraction [31]. To support functional computation as part of reasoning and optimization strategies, the logic should be equipped with operational semantics — e.g., based on conditional term rewriting similar to that of equational specification languages such as Maude [16], which is key to combining abstract logical models with an efficient notion of execution.

The logical view of NCPS allows us to recast information collection, control, and decision problems as logical problems that are primarily centered around the duality of two kinds of knowledge: facts and goals. Various classes of distributed algorithms can be declaratively expressed using this duality. Proactive, data-driven, or optimistic algorithms are mostly concerned with the establishment of new facts from existing facts, hoping to satisfy the goal but considering it as a secondary aspect. Reactive, demand-driven, or pessimistic algorithms are primarily goal oriented, meaning that during their execution new subgoals are established based on existing goals, which eventually can be directly established using the facts. It is noteworthy, however, that many interesting practical algorithms (e.g., hybrid routing for sensor nets) are a mixture of different paradigms. Hence, in our logical framework, both facts and goals need to be treated on an equal footing together with corresponding forward and backward inference rules.

Toward a Robust Logic of Degree and Uncertainty A logical model is an instance from a fixed model class represented by a common background theory. In most

applications, we are concerned with incomplete information, and the model of the real world is not entirely characterized. Hence, we are almost always concerned with an entire class of models that are consistent with the facts to various degrees. Apart from the natural incompleteness of knowledge due to partial observability, many sources of uncertainty exist in cyber-physical systems, including environmental noise, measurement errors, system perturbations, sensor and actuator delays, and clock drift. Networked systems exhibit further sources of uncertainty caused by delayed, outdated, incomplete, or inconsistent knowledge. Furthermore, uncertainties play a natural role in information fusion and probabilistic algorithms. The consideration of a class of models also allows standard logics to represent certain aspects of uncertainty, but the degree of uncertainty is not explicitly represented. A natural solution would be to use an instance of many-valued logic [30] that is sufficiently constrained to be consistent with common probabilistic [22, 24], stochastic [17], and quantitative interpretations [81]. To enable expression of priorities between goals or their relative importance (e.g. to differentiate between hard and soft constraints), we furthermore need weighted formulas.

In cyber-physical systems, models, facts, and goals are continuously changing. Therefore, it will be essential to design a robust logical framework that can gracefully, incrementally, and efficiently deal with such changes. One possible approach is to maintain proofs explicitly at a suitable level of abstraction — e.g., as partial orders (as opposed to sequential proofs) capturing all dependencies between facts and goals. Proof maintenance will take advantage of the locality of changes and hence can improve the efficiency of automated deduction and constraint solving/optimization. For instance, an explicit partial-order representation of dependencies enables more sophisticated search and optimization strategies, such as conflict-driven backtracking and logical state composition strategies that do not assume centralized control (see Section 4.2).

Depending on the nature of changes, proofs can either remain valid, require local adjustments, or become entirely invalid. Clearly, the former case is preferred, which is why we suggest complementing proof maintenance with a notion of proof robustness that, when used as an optimization criterion, allows us to avoid fragile proofs whenever possible. Proofs can be fragile because they are based on rapidly changing or unstable facts or because they lack redundancy. Consider, for instance, the goals of maintaining network connectivity or sensor coverage. Clearly, proofs representing solutions that rely on stable facts about the neighborhood of a node are preferred. Furthermore, in dynamic environments, proofs can be carried out in a robust way that instead of relying on an individual fact, which could become a single point of failure, relies on an abstraction — e.g., a disjunction of independent facts representing coverage or connectivity via several neighbors that remains invariant under a larger set of network perturbations.

3.2 Model Synthesis and Adaptation via Distributed Monitoring

Models in our approach come in two flavors — namely simulation and logical. Simulation models are executable and represented by a set of software fracments. Such fracments can be direct reflections of implementation fracments, but they can also represent a more abstract version of the implementation. Additional simulation fracments can capture executable models of the environment of the NCPS, which for instance includes node mobility and networking capability. Logical models are represented by a background theory together with facts that further narrow the relevant class of models, — e.g., by fixing or constraining model parameters. Depending on its level of abstraction, a logical model can have an underlying executable model.

In most cases, the models cannot be fully characterized in advance and can change while the system is in operation, which is why model adaptation is an essential ingredient of our architecture. Models (or their characteristic parameters) are shared just like other forms of knowledge, and this process is subject to the limitations of network connectivity and bandwidth, which leads to additional possibilities for delays, incompleteness, and inconsistencies.

Model synthesis and adaptation go hand in hand. Lacking other knowledge, the system can start with a default model (e.g., a single node cluster/constellation in our satellite application), which is incrementally refined during operation of the system. Knowledge can be passively accumulated by observations — e.g., from cyber-physical sensors — while the system is executing its primary function or mission, or it can be actively pursued by exploration, which may require physical actions. Often, combinations of the passive and active modes of model adaptation will be needed for acceptable performance with low exploration overhead. The specific exploration strategy is part of the system strategy so that trade-offs between exploration and exploitation of knowledge can be expressed as part of the overall system goal. The trade-offs between exploration and exploitation (of accumulated knowledge) are well known in the context of reinforcement learning [76], but are more challenging in our context due to constraints imposed by the model and goals, by resource limitations, and by the distributed nature of NCPS. In the case of satellites, even a relatively minor form of physical exploration by means of orbit adjustments can require significant resources (e.g., energy, time) and can be in conflict with the primary goal (e.g., maintenance of network connectivity). Network beaconing or probing (for node discovery or performance estimation) can be seen as another less expensive form of exploration of the environment.

In addition to these trade-offs at the strategy level, there are trade-offs that need to be considered when developing the models themselves. Even logical models can be executable in a sense. In fact, suitable abstract models can take advantage of the fragment of logic that supports symbolic execution, efficient deduction, search, and optimization. In the context of model adaptation, there is another reason why abstract models are often preferable as a basis for system control. In principle, models can try to precisely capture the reality such as mechanical models of motion or the path loss models used for wireless commu-

nication, but the parameter estimation needed to adapt such models to reality can be expensive or infeasible given the amount of data and sensing capabilities of cyber-physical systems. Indeed, in machine learning the notion that precise system identification is necessary for best performance has mostly been rejected [13]. Adaptation of simpler models has the advantage of requiring fewer data points, but predictions will be necessarily less precise. Still simpler models often lead to superior performance, because their lack of precision is compensated for by their robustness under noise and their capability to generalize to new situations.

4 A New Generation of Runtime Assurance Techniques

A major hurdle in traditional system verification is the explosion of possible cases to consider due to lack of knowledge at design time about the particular system state or configuration at runtime. The flexibility offered by dynamic reconfiguration and retasking further exacerbates this problem. Furthermore, the typical NCPS we are interested in should support unanticipated missions which means that even the specification is not known at design time. To tackle these problems, we propose validation, synthesis, and verification techniques that can take place at system runtime when the best possible knowledge about the system state and the mission goals is available. Such runtime assurance techniques can exploit the knowledge about the current system state to focus the verification on what is currently relevant or relevant in the near future. Due to their very focused nature, the potential for state space explosion is significantly reduced and the savings in terms of resources can be substantial.

To avoid confusion, we should point out that the techniques we propose as research opportunities are significantly more dynamic than ongoing research in the field of (monitoring-based) runtime verification [51, 1, 11]. This area is not so much concerned with shifting design-time methods to system runtime, but mostly with a much more specific problem — namely, the construction of efficient (possibly distributed) monitors for a given fixed property. The property is known before the system is instrumented for monitoring, and it remains fixed during system runtime. Instead of considering self-verifying systems, the verifier is usually external to the system that is monitored, or more precisely is assumed not to have an impact on its behavior, an assumption that we cannot make for the resource-constrained systems in which we are interested. Furthermore, the properties of interest are expressed in a relatively weak temporal logic and intended to capture only specific aspects (e.g., null-pointer dereferencing or race conditions) of a software system. These state-of-the-art runtime verification techniques can be used offline (at design time) and online (during normal system operation). They have been implemented in various frameworks, such as Pathexplorer [65], Eagle [5], and MaC [47], and have been very successful in finding subtle software bugs. Most current techniques appear to be focusing on the code level. Interesting and notable exceptions are the component-based architecture [6] and AMOEBA-RT [27], which can verify adaptation properties of systems

that transition between different regimes. Unfortunately, these techniques are not sufficiently expressive and dynamic for our purposes, where system properties and goals are very complex and constantly evolving. The approach [29] of using runtime monitoring to correct the global dynamics of systems (UAV swarms in this case) evolving according to the local rules of an artificial physics model is quite close to the spirit of fractionated software, except that the assumption of a global view and global control needs to be relaxed.

4.1 Probabilistic Runtime Testing and Verification

We sketch several new runtime testing and verification techniques that can provide probabilistic assurance that current and future states of the system satisfy the system goals. These can include specific mission goals, intermediate goals, and general or specific policy constraints and performance requirements. We envision runtime testing and verification techniques to be invoked by higher layers. Specifically, the distributed control and optimization strategy can be used to verify the violation or near-violation (e.g., by means of a slightly stronger property) of critical properties at present or future states. The set of future states will typically be bounded by a state- and property-dependent look-ahead horizon so that state space explosion can be kept under control. Given that runtime verification can be invoked repeatedly on the trajectory of the system, a tightly bounded look-ahead horizon is acceptable as long as it is sufficient to take corrective actions if the need arises. Runtime verification needs to be complemented by just-in-time validation (see Section 4.4), which will be applied at longer time scale.

Randomized Symbolic Verification The most basic form of runtime verification without prediction can focus all resources on the present state. It uses the formal model together with the current goal to detect violations or near violations of critical invariants. The verification will typically involve global system properties, but it will be based on local knowledge (about local and remote states and about the state of the environment). As an extension of the basic technique, we propose to take into account the imprecision or lack of knowledge about the global state of a distributed system. To this end, one might consider performing runtime verification on a (symbolically represented) envelope (i.e., set of states) around the system state derived from the best knowledge available. Efficient symbolic SMT (satisfiability modulo theories) solving techniques such as those used in Yices [20] and efficient symbolic computation and deduction by conditional rewriting (modulo theories) as in Maude [59, 16] are mature technologies on which to build on. If the symbolic representation reaches a certain complexity threshold, probabilistic assurance can be obtained by lifting the basic verification techniques to the set of states using sampling techniques. Each sample can be an entire symbolic region (a possibly infinite set of states) so that high coverage can be achieved with a relatively small number of samples. This probabilistic use of symbolic solvers opens a rich set of possibilities that to our knowledge have not

been investigated in the past. Other very promising approaches to the integration of logic and sampling-based analysis techniques are Markov Logic Networks [64] as, for instance, implemented in the Probabilistic Consistency Engine [3] (PCE), which can be used to quantify the probability that a property holds or that the system is in a certain state (e.g., based on partial observations).

Dynamic Runtime Model Checking A complementary direction is to exploit the time dimension by using available computational resources to predict the future evolution of the system up to a certain time horizon, which needs to be short enough to avoid state space explosion. One category of prediction-based assurance methods would use model checking at runtime. Model checking can be directly applied to executable logical models as the Maude LTL model checker [23] demonstrates. Some early work exists on runtime model checking of safety properties for multithreaded programs [86], which incorporates interesting ideas on dynamic partial-order reduction (to further reduce the state space to be explored). There is also work on guiding a model checker based on a runtime analysis of programs [35], and a next consequent step would be to perform both analysis and model checking at runtime. Lifting these ideas from program code to higher levels of abstraction is an opportunity worth exploring. Model checking would be performed locally but based on the continuously adapting models of local and nonlocal behavior. Since the time horizon is limited, bounded-model checking techniques, which can be implemented using efficient SAT solvers, would be of interest as well. Furthermore, statistical model checking techniques [48] that can deal with probabilistic models (e.g., represented as Markov chains) and probabilistic properties could be applied at runtime. A major challenge with all these approaches is to develop efficient runtime algorithms that can scale with the available resources and to explore how model checking can be distributed (some ideas from [72] about distributing formulas can be useful in this context) and take advantage of the fractionated nature of our systems.

Dynamic Directed Monte Carlo Analysis Simulation-based methods for distributed systems are well suited for quantitative analysis, but since they do not provide full coverage, they are inherently probabilistic in nature. Monte Carlo techniques can be used to account for imprecision of models and of the global state. The advantage of using such techniques at runtime rather than design time is the potential for a much more directed application exploiting the additional knowledge available, which directly translates into resource savings and/or precision improvements. Furthermore, by generalizing statistical verification via hypothesis testing, the number of samples can be dynamically adjusted based on the actually required confidence that may be known only at runtime. Black box statistical model checking generalizes hypothesis testing to temporal properties [73, 88], has been extended to quantitative properties [80], and has been applied to verifiable cross-layer adaptation in our own work [44, 43]. In a fractionated architecture, Monte Carlo techniques such as these will naturally scale, and the precision improves (or alternatively the local load will be reduced) with the number of nodes. Several unexplored extensions of this scalable simulation-

based approach will also be of interest. First, the use of the current state as a starting point can be relaxed, by focusing the verification on interesting, critical, or recurring states. Machine learning techniques could acquire such states (and their distribution) during the lifetime of the system. Monte Carlo simulations on such states can then be continuously executed in the background, possibly controlled by resource availability. Monte Carlo simulation can furthermore be biased to explore performance extremes (runtime stress testing), rare (e.g., black swan) events, or high-risk situations (based on runtime risk assessment). Markov-Chain Monte Carlo (MCMC) techniques, which are at the core of PCE [3], are of interest as well due to their capability to efficiently sample from complex joint distributions. By making the temporal dimension explicit in the model, a tool such as PCE can also be used to perform temporal analysis (with a reasonably short look-ahead horizon).

Integrating Formal Methods and Simulation Given that deduction, model checking, and simulation-based techniques each have their own advantages, a natural question is if these approaches could be integrated into a single hybrid runtime verification technique. One possibility is through executable formal methods such as Maude [16], an idea that we explored under the name *formal prototyping* in the context of fault-tolerant middleware [33] and security [28]. Another unexplored possibility is based on a notion of abstraction. An abstraction essentially replaces subsets of system states by representatives so that the complexity of verification is further reduced. Given such an abstraction we can use simulation-based techniques to evaluate the performance of a system in each abstract state (or a relevant subset) by using detailed simulation models. Model checking and similar formal methods can then take place at the abstract level, and verify properties such as if a certain level of performance can always be maintained in certain situations that can be expressed logically. A more intelligent integration might trigger the underlying simulation on the fly only for states that the model checker explores. Furthermore, the number of samples could be determined by the required confidence level, which essentially means that resources are directed to the properties that matter. It should also be noted that model checking can naturally be used as part of a deductive system [71] and hence fits naturally with the higher-layer runtime assurance techniques that we will discuss subsequently.

Adaptive and Probabilistic Runtime Abstraction We have seen that runtime assurance can greatly benefit from knowledge that is not available at design time. Learning the reachable or relevant states of the system is just one dimension to exploit. A second dimension is concerned with the problem that the properties that need to be verified may become available or sufficiently concrete only at runtime. A third orthogonal dimension is to use runtime information to determine and adjust at runtime the level of abstraction where other techniques are applied. For instance, the combination of model checking with abstraction [15] has attracted a lot of attention for the purpose of design-time verification. Finding the right abstraction, however, is not easy, and currently done by repeated model checking with counterexamples-guided refinement. Without being

confined to model checking, a yet unexplored but related idea of violation-driven refinement could be used at runtime to choose a suitable level of abstraction for monitoring properties of interest. Runtime techniques would furthermore enable an alternative and more efficient approach to learning or synthesizing abstractions from observations of real system dynamics. The key idea is that states with similar observable properties at runtime do not have to be distinguished even if there is a theoretical possibility that they behave differently under some condition that the system will never reach. More generally, verification techniques can benefit from a notion of probabilistic runtime abstraction, where the correctness of the abstraction (and corresponding abstract models) is empirically established with quantifiable probability and confidence.

Resource- and Situation-aware Runtime Assurance Based on the environment, available resources, or timing, one may employ different runtime verification strategies. A flexible, fractionated architecture together with model-based distributed control and optimization strategies can facilitate switching among them. The general idea is to focus on features that matter in a particular situation but also partition the resources between the primary function of the system and the different runtime assurance techniques in a way that takes into account the various trade-offs in this space. For example, when a system (e.g., a spacecraft) is launched for the first time, one may allocate more resources for performing runtime monitoring, testing, and verification to ensure that the system functions properly. At a later stage, when the system has run for some time without issues, one may reduce the frequency or depth of runtime assurance to conserve resources or to reallocate them for other purposes. Another example of situation-aware runtime assurance is that the amount of runtime monitoring, testing, and verification (e.g., of security policies) performed may depend on perceived threats (e.g., attacks from adversaries). Based on the threat level, one may reconfigure the system to deploy a more comprehensive monitoring posture and, possibly additional, what may be called *defensive fracments*.

Learning from Failures and Near Failures Failures of software fracments and violations of properties at runtime, even if corrected, can tell us a lot about future risks. Near failures and near violations (whether they are determined with or without prediction) are another possible source of critical states that should not be ignored. Reachable and critical states (and possibly their distribution) can be learned at runtime and analyzed in the background using directed runtime simulation and verification techniques. However, there is another unexplored opportunity here — namely to learn how to recognize similar critical situations and use runtime state avoidance techniques to circumvent them in the future (at least during a critical mission). The distributed learning of such states can happen as a generalization of distributed monitoring, which is needed in a mission-critical system for many reasons, but in particular to guide the continuous maintenance, improvement, and evolution of the system over its lifetime and over future generations. More generally, machine learning (especially statistical learning theory) is an area with a rich set of techniques that can contribute to new runtime

assurance techniques in many ways (e.g., learning of specifications and model-based prediction), but we have touched upon only a few examples due to space limitations.

4.2 Model-based Distributed Control and Optimization

System control and optimization in NCPS is challenging. The control of runtime assurance techniques and the consideration of the trade-offs of potential adaptations and countermeasures must be performed in context of the overall system goal, in which quantitative aspects will typically play an important role. Traditional optimization techniques that strive for optimal solutions based on precise models are not suitable for most NCPS, where models have many dimensions of uncertainty, and optimality in the strict sense is neither desirable nor achievable. What is needed in practice are strategies to find acceptable and robust solutions, sufficient to achieve the goal while taking into account the limitations of the models and available resources.

On the other hand, the fractionated nature of our system offers many advantages including fault tolerance, distributed sensing, coordinated actions, and inherent parallelism for computational processes that should be exploited not only for the primary function of the system but also by the runtime assurance techniques, and in particular the control and optimization strategies. Clearly, a top-down decomposition of the overall goal in a divide-and-conquer fashion is not a viable approach, because solutions may require ad hoc cooperation across layers and across nodes. This and related problems of strictly layered approaches have led to the recent trend of cross-layer design and optimization in networking (and especially sensor networks). Among other sources (see Section 6), our xTune architecture for cross-layer control and optimization based on the runtime application of formal methods has served as an inspiration for the following ideas.

Closely related to the idea of combining runtime verification and model-based control is the area of model predictive control also known as *receding horizon control*. In a discrete setting, it has been successfully applied, for instance, in NASA's Livingstone [83], a kernel for a self-reconfiguring, reactive, and autonomous system. It combines model-based diagnosis and a propositional controller, an idea that has been further generalized to model-based programming in [82], based on a language that can be compiled into hierarchical constraint automata. Another model-based architecture that was the basis for our framework [18] for goal-oriented operation of remote agents [62] is JPL's Mission Data System [21], a unified flight-ground control and data system. Protection against faults and dealing state uncertainty are noteworthy features. None of these architectures, however, were aiming at loosely coupled, highly distributed and fractionated NCPS that lead to many new challenges as we explain below.

Control and Optimization as Logical Strategies Mathematically, the logical framework and its underlying fractionated computing paradigm allow a rich set of conceivable behaviors that need to be constrained to a subset that satisfies the

system objectives. We suggest developing strategies that control and optimize the operation of NCPS based on its declarative representation in the logical framework with the idea that the generated control actions are correct by construction. These strategies will be resource-aware and adaptive. For example, in homogeneous scenarios, our strategies can exploit the parallelism of many nodes so that resource consumption at each node can be low. In heterogeneous cases, they can exploit powerful or energy-rich nodes that perform heavy computations so that low-power nodes can save their resources. Nodes will try to share knowledge and cooperate while communication conditions are good, but if communication is impaired or disrupted, nodes will tend to operate more autonomously. If knowledge including facts, goals, and solutions can be shared using a framework based on partially ordered knowledge sharing such as [42], the location of computations is flexible to a large degree and limited only by the communication and node capabilities. A distributed logical framework that could serve as a basis for this approach is currently being explored in [74] and [46].

Ideally, the strategy exploits parallelism inherent in search and optimization problems, by allowing nodes to sample the search space independently. Unlike numerical approaches, sampling can be done symbolically, by randomly generating new subgoals that represent entire regions of potential solutions in a finite way. The sampling heuristics can be biased by a nonuniform distribution to express locality and preference for solutions that can be reached more easily or with lower cost. In addition, the cost of reaching a solution can be explicitly quantified and constrained by the system goal. The best stable solution will be shared opportunistically across the nodes and is ultimately used to drive the local actions of NCPS. The symbolic sampling strategy explores the search space of potential solutions, but conflicts can arise, possibly after several subsequent reasoning or constraint-refinement (i.e., narrowing down the solution region) steps performed using the logical framework. Conflicts can manifest themselves either as logical inconsistencies or nonacceptable solutions. One possibility to deal with conflicts is by local randomized backtracking driven by the conflict itself, exploiting the dependencies maintained by the underlying logical framework. A randomized approach to search and optimization tends to avoid redundant computations (i.e., the same computation at several nodes) under cooperative conditions, but would not rule out redundant computations that are essential for progress if nodes need to operate autonomously.

In traditional approaches to planning and optimization, the process terminates when an acceptable solution is found and leaves it to lower layers to take the actions to implement and fine-tune the solution. NCPS, however, need to be continuously controlled and optimized. The continuous optimization will consider the most recent known state of the distributed system and hence can quickly adapt to changing facts and goals. Even if actions have been already taken toward the transition into an acceptable solution region, a significantly better solution might emerge either because the solution was not explored until now due to computational resource limitations or because it arises due to new

unexpected conditions, including failures preventing the system from reaching the solution it was aiming at in the first place.

Abstraction as the Key to Robustness and Composability A logical approach to optimization would also enable the composition of (partial) solutions. Rather than aiming at a numerical point solution each node narrows down the goal to one or multiple solution regions represented by logical formulas. If two nodes establish connectivity, the goals will be composed by a logical conjunction resulting in a goal that semantically corresponds to the intersection of solutions acceptable for both nodes. The approach can be generalized to entire groups of nodes that merge due to a network topology modification. There is a natural connection between abstraction, robustness, and composability. Composability is enabled by a suitable level of abstraction that avoids over-constrained point solutions. In other words, solutions are robust enough to accommodate, at least to some degree, the needs of other nodes. The use of an abstract solution region reduces the likelihood of conflicts in the case of composition, but clearly cannot exclude this possibility entirely. Conflicts caused by composition can be treated just like any other conflicts arising during search and optimization.

4.3 Distributed On-demand Deductive Synthesis

Techniques to synthesize software based on a declarative specification have a long tradition in what is sometimes called automated software engineering. NASA's Amphion [53] is one of the well-known projects where automated synthesis has had a large impact in the reduction of labor-intensive software engineering activities. The Amphion system, which is still in use at NASA today, generates scientific programs as a composition of subroutine libraries. Since synthesis is based on deduction in a sound logic, in this case the first-order logic of the SNARK [2] automated theorem prover, the solutions are correct by construction.

As with the other proposed techniques, we propose to shift the synthesis process to system runtime. More specifically, we propose on-demand synthesis whenever a new mission or policy goal requires a solution that cannot be implemented by a single (coordinated) action but requires a certain degree of planning with intermediate goals. The solution would consist of a set of activities or components suitably instantiated, parameterized, and composed to achieve the overall goal. The bigger challenge is, however, to perform the synthesis, like control and optimization, in a process that exploits the loosely coupled fractionated computing paradigm, and furthermore the solution generated by the synthesis should be distributed in the same sense.

To illustrate the logical inferences in a distributed deductive synthesis process, consider a greatly simplified example of intelligent surveillance. Assume that each satellite in a fractionated system is equipped with only one kind of capability, either a high-resolution camera or a motion sensing capability that is implemented on the basis of measurements received from a sensor network on the ground. Assume that predicates $Motion(a, t)$ and $Pattern(a, t)$ are true if

a movement or a particular pattern has been detected in an area a at time t (approximately). Assume furthermore that $Image(I, a, t, t')$ means I is an image of area a taken in the interval $t, ..., t'$, and $Delivered(I, r)$ means that the information I has been delivered at r. Now the following goal is injected at a ground node r:

$$Motion(a, t) \vee Pattern(a, t) \Rightarrow$$

$$\exists I : Image(I, a, t, t + \Delta t) \wedge Delivered(Extract(Abstract(I)), r)$$

It expresses that an image needs to be taken of a specific area a with maximum delay Δt after a motion has been sensed or a visual pattern has been recognized. The image then should be delivered to r after abstraction and feature extraction. After the goal is disseminated in the network, each node tries to solve the goal. Let us now assume that a node above area a generates a fact $Motion(a, t)$ that can be used by another node that is monitoring that area and is equipped with a high-resolution camera to simplify the goal to

$$Image(I, a, t, t + \Delta t) \wedge Delivered(Extract(Abstract(I)), r)$$

so that the only way to make progress is to take an image i to satisfy $Image(i, a, t,$ $t + \Delta t)$ leading to the remaining goal

$$Delivered(Extract(Abstract(i))), r)$$

Let us assume that the abstraction $i' = Abstract(i)$ can be performed immediately after taking the image but feature extraction will be performed at a more powerful node, say at the ground station, because it is computationally expensive. This node will then simplify

$$Delivered(Extract(i'), r)$$

after performing the computation $i'' = Extract(i')$ to $Delivered(i'', r)$, which can be incrementally solved by moving $Delivered(i'', r)$ closer to r, the requesting ground node, where it is finally realized by a delivery action.

A similar but more detailed example of a logical theory for distributed surveillance using a team of mobile robots can found in [74]. In spite of its simplicity, this example exploits three dimensions (computation, abstraction, and communication) of distributed computing, and yields a solution that is synthesized on the fly and correct by construction based on the soundness of the underlying logical framework. It furthermore illustrates the combination of logical inference and partial evaluation and their generalization to the distributed setting in which goals and facts can be bound to actions at different locations in the cyber-physical world. In a more complex example, we might easily imagine that $Motion(a, t)$ and $Pattern(a, t)$ cannot be satisfied using the current distribution of nodes so that some nodes will have to adjust orbits to achieve sufficient coverage of area a. Clearly, this opens a rich trade space of possible solutions, which can be tackled by the combined capabilities of distributed deductive synthesis and the distributed control and optimization strategies discussed previously.

4.4 Predictive Just-in-time Validation

It is well-known (but often forgotten) that the correctness of a system w.r.t. its specification is not sufficient to guarantee that the system operates as expected and is suitable for a given mission. The problem is that a high-level specification of the system goals, even if it is declarative and far less complex than the implementation, is complex enough that it is difficult for humans to judge whether it captures their intent. Inconsistencies (e.g., logical contradictions in the extreme case) or incompleteness (e.g., missing key properties) are very common. Hence, verification needs to be complemented by validation techniques that can increase the users' confidence in the specification. Clearly, another level of verification relative to even higher-level specifications cannot be the answer, because the fundamental problem would be just postponed. Instead, we propose a simulation-based approach, which includes the runtime assurance techniques of all layers and as motivated earlier would be executed just in time, whenever the system goals are modified as a consequence of user interactions. Just-in-time validation has the advantage that the specification can be very specific to a particular mission, eliminating many possible use cases of the system that are simply not relevant. Thanks to its simulation-based nature, the results of the validation will be quantitative rather than simple yes/no answers. Quantities are not limited to probabilities of properties being satisfied but can include expected performance metrics and bounds. Furthermore, counterexamples in terms of property-violating (or just risky) executions can be fed back to the user who then has many options to respond, ranging from adjusting or replanning the mission to reallocating resources or scaling up the system capabilities (e.g. by additional launches in the case of a fractionated satellite mission). Since the simulation is performed by the highest layer, it may include the execution of the embedded runtime assurance techniques, and as a consequence its coverage and capability to detect problems is higher then that of conventional simulation techniques without embedded verification.

Predictive just-in-time validation has to cover the entire distributed system as well as all layers of the architecture with a time horizon that covers or is at least representative for the entire mission. Hence, predictive just-in-time validation can be computationally resource intensive and probabilistic simulation-based techniques are preferable. Given that mission validation can be time critical, the parallel nature of probabilistic simulation techniques will be an important advantage. As with all runtime assurance techniques, a fractionated software architecture leaves a lot of freedom regarding where the actual simulation is carried out. In case of a networked satellite mission, it would make sense to utilize a computing grid on the ground to perform a large number of such simulations around an approximation of the current state of the system, which is always available by means of adaptive models. In other words, we continuously maintain a virtual approximation of the real system that is used for just-in-time validation whenever the system needs to be configured for a new mission. Clearly, multiple concurrent overlapping missions by multiple users of the cyber-physical infrastructure are particularly interesting, because the effects of sharing limited

resources will be predicted by the validation process, and the injection of a new mission into the system may be rejected because of resource limitations.

Since predictive just-in-time validation can also be applied at design time (although at a higher computational cost due to the more limited knowledge about the future system state and configuration), it should be general enough to subsume existing validation and performance evaluation techniques, namely discrete-event (network) simulation and hardware-in-the-loop simulation techniques. The current practice is still centered around the use of a variety of simulation tools (such as Mathlab, Qualnet, or the STK satellite modeling toolkit) to capture different aspects of the system under evaluation. However, the diversity of tools and their different levels of abstraction often leave a significant gap between the real system and the model that is evaluated. Keeping the simulation models in sync with the actual code is a labor-intensive and failure-prone task and the confidence that the simulation captures all important aspects is usually based on experience and subjective judgment. In our proposed reflective architecture, simulation models are first-class concepts, so that runtime and design-time validation and evaluation can use the same set of models, which at the lowest level of abstraction can be identical to the actual implementation, thereby reducing the modeling gap.

5 Illustrating Example: Fractionated Satellite Networks

Consider a network of fractionated satellites that has already been deployed in space and needs to be retasked rapidly, i.e., within hours, for surveillance of a particular geographic region during a crisis. To accomplish this, the satellites need to perform coordinated orbit adjustment maneuvers to provide sufficiently good coverage of the areas of interest with a frequency that satisfies the mission requirements. Specifically, we chose a primary goal, such as the collection of information (e.g., images) from a particular area, that can be achieved only by actively morphing and expanding the network topology — e.g., by tethering (stretching the network in a particular direction) possibly with some redundancy to reduce the likelihood and duration of disconnections. Various essential policy and system goals concerning sensor coverage, network connectivity, or energy consumption can be active at the same time in addition to the primary user objective.

Now suppose there exists a (previously) unknown bug in the image processing software that manifests itself when it processes data pertaining to a very small number of (geographical) coordinates. Using runtime verification, the satellite may be able to discover the bug and take corrective actions to avoid the problem. Specifically, based on the current coordinate of the satellite and its trajectory, the runtime verification system discovers that the image processing software fails when it reaches a certain coordinate. The satellite finds several possible solutions to mitigate the problem. First, the satellite may change its trajectory to avoid the problematic coordinate. Second, the satellite may stop functioning temporarily when it reaches the problematic coordinate, while having other satellites to

handle the area it is supposed to cover. Third, the satellite has another implementation of the image processing software module that does not have the bug, and the satellite replaces the faulty software with it. After evaluating the costs and benefits of the options, the satellite chooses the most cost-effective one.

Several variations of this sample mission would lead to more challenging test cases pushing runtime assurance techniques to their limits. Hardware and software fracments could be instrumented to fail continuously with unusually high rates during the mission (simulating a combination of software and hardware faults), and the high-level system objective and performance still needs to be maintained without interruption by agile system adaptation. Also, the dynamic improvement of the capabilities by launching new nodes, as well as simulated network partitioning, merging, perturbations, and the loss of nodes, is a rich source of test cases for system robustness. Other possibilities include considering more complex system goals with partially conflicting multiuser objectives, policies, and corresponding trade-offs. In addition to energy, system goals can involve timing constraints, quantification of QoS and robustness (e.g., of network connectivity), and consideration of risks (e.g., of losing nodes) and options (e.g., flexibility to react to new mission goals). Finally, the resource-adaptive distributed operation in a nonhomogeneous global network — e.g., a combination of small satellites, UAVs, a ground sensor net, a ground station network (with fixed and mobile nodes), and powerful grid nodes in the Internet — would be an ultimate test case in system-of-systems interoperation.

6 Background and Related Work

For an up-to-date overview of our ongoing work on Networked Cyber-Physical Systems and a large body of background literature that is beyond the scope of this paper we refer to [63]. In the following we limit ourselves to a few selected research directions and projects that had a significant influence on our suggested approach.

Delay- and Disruption-Tolerant Networking (DTN) and Sensor Networks DTN [25] enables communication in challenging environments where many NCPS are deployed. Underwater sensor networks [32], wildlife tracking [58], vehicular networks [52], satellite networking [39], and interplanetary deep space networking [10] are just a few examples demonstrating the wide range of applications. By combining network caching and routing on an equal footing, DTN can overcome intermittent connectivity, such as in highly dynamic networks of mobile nodes or in sensor networks that are scheduled for energy efficiency. *Space-Time Adaptive Networking Architecture* (STAN), which we have recently proposed as a small-footprint solution for small satellite networks (such as those based on the Cubesat [77] platform), further improves upon existing DTN architectures. STAN is a true *cross-layer architecture* that leverages adaptive and predictive models for intelligent power-management, caching, and routing. The example used in this paper captures several interesting aspects of DTN and STAN if

applied to fractionated satellite networks. In a limited form, some of the ideas proposed as research opportunities in this paper are present in our earlier work in the context of DTN. For instance, our *reflective routing algorithm* [75] increases the probability of delivery based on a reflective and predictive logical model of the distributed system. Furthermore, a special kind of runtime abstraction, coined *self-organizing abstraction*, of dynamic networks has been used in our recent work to increase performance of disruption-tolerant routing.

Constraint Solving, Optimization, and Distributed Approaches The borderline between constraint satisfaction and optimization has mostly disappeared due to the need to judge the quality of solutions for efficient search. Recent advances in SAT solving also show that logical approaches to SAT solving can be naturally extended to optimization problems such as MaxSAT [49] and MiniMaxSAT [36], which supports weighted clauses. Much progress has been made on moving from propositional logic to more expressive fracments of first-order logic as witnessed by recent SMT solvers such as Yices [20]. Unlike this line of work, which aims at completeness and optimality, our approach aims at sufficiently good results for more expressive fracments of first-order logic and our quantitative extension. In spite of their limited expressiveness, modern SAT/SMT solvers became powerful enough to realize the idea of viewing planning as a satisfiability and optimization problem [41, 34]. Some evidence that higher expressiveness can be very practical with acceptable trade-offs is provided by our work on software-defined radios, in which we developed a policy logic and a constraint-based reasoner for dynamic spectrum access [84].

Some recent research has been conducted on parallelizing SAT solving [7], GridSAT [14] being one implementation. In earlier work, randomized backtracking has been proposed as a mechanism for a parallel Prolog implementation [40]. These parallel approaches are mainly concerned with performance gain, possibly fault tolerance, but do not cope with the inherently distributed nature of the problem, which is crucial in many NCPS.

It seems, however, that distributed algorithms offer this promise. *Distributed constraint satisfaction* (DisCSP) and *optimization* (DisCOP) problems have been investigated in the context of multiagent systems [87]. Some common algorithms are distributed versions of their centralized counterparts, like local annealing [26], distributed hill climbing [56], distributed stochastic search and distributed breakout [89], or ADOPT [60], which performs distributed depth-first backtracking based on a fixed variable ordering. One of the most interesting algorithms is OptAPO [57], which is not simply the adaptation of a centralized algorithm, but is based on a dynamically selected mediator, which internalizes a larger part of the problem and helps to solve conflicts. In spite of their asynchronous nature, all algorithms use classical multimessage protocols. A bigger problem is that the DisCSP/DisCOP assumptions (finite domains, reliable communication) are not satisfied for many NCPS. Nevertheless, there are interesting aspects of DisCSP/DisCOP solutions with potential to generalize. For instance, the idea of *mediation-based cooperation* has served as another source of inspiration for our loosely coupled approach, where every agent internalizes (part of) the problem

and can therefore act as a mediator and disseminate the new solution state. The difference is that in our approach this happens opportunistically (and using an expressive logical framework) rather than as part of a multimessage mediation protocol.

Compositional Cross-layer Optimization It is widely accepted that cross-layer optimization, e.g., involving physical, medium access, and routing layers, is a key technology for resource-efficient networking. The idea of using formal methods at system runtime has recently been applied to *compositional* cross-layer optimization [45] in the context of the xTune framework [85]. In xTune, we have the classical optimization objective of finding suitable parameter settings at each component based on a *utility* function capturing the *effectiveness* of the settings relative to the user and system objectives. For example, utility can be a function of energy consumption, timeliness of operation, quality of service, bandwidth demand, and buffer capacity requirements. In xTune, we achieved cross-layer optimization by constraining the behavior of local optimizers working at all abstraction layers (application, middleware, operating system, hardware architecture) that are connected by a *vertical* composition. Each local optimizer uses the other optimizer's refinement results as its constraints. Thus, the constraint language serves as a common interface among different local optimizers, leading to improvements of solution quality, robustness, and speed of convergence. Compositional optimization through constraint refinement enables a controller to coordinate existing local optimizers, which can accommodate different objectives, by treating them as black boxes. The control and optimization strategies that we discussed in this paper can be seen as a generalization of the compositional constraint-refinement approach to include *horizontal* composition capturing the distributed nature of NCPS.

Constraint-based optimization can be entirely generic or guided by a model of the system to optimize. Aiming at the latter case, we can build on our experience with probabilistic runtime analysis [44] and tuning of abstract cross-layer models [37, 45, 43] specified in the formal modeling framework Maude [59, 16] that is based on the notion of executable specifications. Statistical analysis techniques have also recently been integrated into our cyber-application framework [42]. In this paper, we propose to move from purely local reasoning, statistical analysis, and model checking techniques toward distributed compositional techniques that integrate randomization and symbolic reasoning. Compared with our earlier work [45] the constraint language would become part of an expressive logical framework that can support strategies for distributed cooperative constraint refinement.

7 Conclusion

It is our belief that traditional techniques for the verification and validation of complex distributed software systems are trapped in an unsatisfactory local optimum, and significant progress is possible only by fundamentally rethinking

the way distributed software is designed. Today's distributed software, in the best case, is based on a rigid composition of relatively tightly interacting coarse-grained components. This makes the entire system prone to low-level faults of many different kinds, and the sheer number of possibilities to consider (not only due to faults) makes verification and validation prohibitively expensive. The discrete and nonscalable nature of conventional software makes it furthermore difficult to build trustable distributed systems that are adaptive and dynamically reconfigurable in a flexible manner. The intuition behind fractionated software is to transform software into a more fine-grained, more continuous form (figuratively speaking, more like a flexible fluid than a rigid composition of bricks) that like biological systems leverage diversity and redundancy to achieve a high level of robustness against low-level faults. As a by-product, fractionated software can also be better distributed, scaled, controlled, and optimized especially as part of NCPS that need to interact with the continuous physical world.

The biggest challenge in moving from the traditional coarse-grained to extremely fine-grained concurrency with self-coordination is the overhead associated with the mapping of a large number of small concurrent computation threads and their interaction on today's computer and network architectures. First experimental results with a prototype implementation of our partially ordered knowledge-sharing model have been reported in [42]. This prototype makes use of thread pools, shared memory and multicast capabilities of the network to support a large number of distributed fracments. Using a case study of evolutionary optimization algorithms we evaluated the scalability of our model using a small number of PlanetLab multi-core hosts, but the granularity of concurrency needs to be further decreased to approach the vision of truly fractionated software. At the same time the number of fracments will increase, and technical solutions (ideally at the OS level) need to be developed to more efficiently map a large number of threads to a large number of computing cores connected at various levels (ranging from shared memory to potentially unreliable networking technologies). Considerations of efficient use of caching (in the precense of a large number of threads) as well as low-overhead networking protocols that implement the knowledge-sharing paradigm in a more direct way would be important to explore in the future.

Once a foundation for fractionated software is available, trustable systems can be built by applying suitable verification and validation techniques at the right level of abstraction and at the right time. We have argued that the right level of abstraction for such systems is the macroscopic level of system properties rather than the microscopic code level that is encapsulated in software fracments, which becomes nearly invisible if the degree of diversification and redundancy is sufficiently high. The right time is the system runtime for the flexible mission-critical systems of interest, when the best possible knowledge is available. Hence, we have suggested numerous research opportunities for new runtime assurance techniques that cover the entire spectrum from validation to synthesis, verification, and testing. Different from today's practice, which mostly relies on subjective judgment, confidence in critical properties should be probabilis-

tically quantified, whether empirically or through models, should be explicitly maintained, and needs to flow through the system along with the invocation of assurance techniques at runtime.

An explicit declarative representation not only of the mission objective, policies, intermediate goals, and performance requirements, but also of the NCPS and its models, is a key feature of our suggested approach, because it allows us to use runtime techniques to generate solutions or actions that are correct by construction. We view correctness as just one dimension in a high-dimensional trade space among many other performance metrics, and we accept that it can be achieved at reasonable cost only by a dynamically balanced set of techniques. Hence, it is essential that distributed control and optimization strategies steer the application of runtime assurance techniques as part of the primary system function in a rational way, enabling the system to operate and respond based on available resources, performance goals, and trade-offs. As a illustrating example we have used the mission-driven operation of a fractionated satellite network because it comes with many facets and challenges, especially in terms of fault tolerance and dynamic reconfigurability, that are far beyond the scope of today's verification and validation techniques.

With the idea of fractionated software we are prepared to exploit the growing trend of distributed and parallel hardware, e.g., in the form of large-scale networks of powerful many-core (rather than multicore) processors. With fractionated software we would also be prepared for a possible future where hardware becomes much less reliable — e.g., due to further miniaturization down to the nanoscale or, more speculatively, where reliability is given up completely as a hardware design goal in favor of extreme parallel performance and/or energy efficiency. On the other hand, a much more concrete opportunity can be found in the domain of our proposed case study. For reliability reasons, spacecraft are usually based on previous generation low-performance processors (often radiation hardened), but the combination of fractionated hardware and fractionated software, which does not rely on the reliability of its fracments, would open an entirely new space of exciting possibilities in terms of cost and performance.

Acknowledgments Support from National Science Foundation Grant 0932397 (A Logical Framework for Self-Optimizing Networked Cyber-Physical Systems) and Office of Naval Research Grant N00014-10-1-0365 (Principles and Foundations for Fractionated Networked Cyber-Physical Systems) is gratefully acknowledged. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of NSF or ONR.

References

1. http://runtime-verification.org/.
2. http://www.ai.sri.com/~stickel/snark.html/.
3. *PCE User Guide*, Version 1.0. Technical manual, Computer Science Laboratory, SRI International, July 2009.
4. Ian F. Akyildiz and Ismail H. Kasimoglu. Wireless sensor and actor networks: Research challenges. *Ad Hoc Networks*, 2(4):351–367, 2004.
5. Howard Barringer, Allen Goldberg, Klaus Havelund, and Koushik Sen. Rule-based runtime verification. In *Verification, Model Checking, and Abstract Interpretation, Proc. 5th Int. Conf., VMCAI 2004*, pages 44–57, 2004.
6. Hakim Belhaouari and Frédéric Peschanski. A lightweight container architecture for runtime verification. *Runtime Verification: 8th International Workshop, RV 2008. Selected Papers*, pages 173–187, 2008.
7. Wolfgang Blochinger. Towards robustness in parallel SAT solving. In *Parallel Computing: Current & Future Issues of High-End Computing, Proc. Int. Conf. ParCo 2005*, pages 301–308, 2005.
8. Robin E. Bloomfield, Bev Littlewood, and David Wright. Confidence: Its role in dependability cases for risk assessment. In *37th Annual IEEE/IFIP Int. Conf. Dependable Systems and Networks, DSN 2007*, pages 338–346, 2007.
9. Owen Brown and Paul Eremenko. Fractionated space architectures: A vision for responsive space. In *4th Responsive Space Conf.*, 2006.
10. Scott Burleigh. Interplanetary overlay network: An implementation of the DTN bundle protocol. In *Consumer Communications and Networking Conf.*, 2007.
11. C. Watterson and D. Heffernan. Runtime verification and monitoring of embedded systems. *Software, IET*, 1(5):172–179, October 2007.
12. Liming Chen and Algirdas Avizienis. N-version programming: A fault-tolerance approach to reliability of software operation. *Fault-Tolerant Computing, 1995, 'Highlights from Twenty-Five Years'., Twenty-Fifth International Symposium on*, 1995.
13. Vladimir Cherkassky and Filip M. Mulier. *Learning from Data: Concepts, Theory, and Methods*. Wiley-IEEE Press, 2nd edition, 2007.
14. Wahid Chrabakh and Rich Wolski. GridSAT: A Chaff-based distributed SAT solver for the Grid. In *SC '03: Proc. 2003 ACM/IEEE Conf. Supercomputing*, page 37, Washington, DC, 2003. IEEE Computer Society.
15. Edmund Clarke, Orna Grumberg, Somesh Jha, Yuan Lu, and Helmut Veith. Counterexample-guided abstraction refinement for symbolic model checking. *J. ACM*, 50(5):752–794, 2003.
16. M. Clavel, F. Durán, S. Eker, P. Lincoln, N. Martí-Oliet, Jose Meseguer, and Carolyn Talcott. All about Maude, a high-performance logical framework. *Lecture Notes in Computer Science*, 4350, 2007.
17. James Cussens. Stochastic logic programs: Sampling, inference and applications. In *UAI '00: Proc. 16th Conf. Uncertainty in Artificial Intelligence*, pages 115–122, San Francisco, CA, 2000. Morgan Kaufmann Publishers Inc.
18. G. Denker and C. L. Talcott. A formal framework for goal net analysis. In *Workshop on Verification and Validation of Planning Systems*. AAAI, 2005.
19. Falko Dressler. *Self-Organization in Sensor and Actor Networks*. Wiley, 2008.
20. B. Dutertre and L. de Moura. *The YICES SMT solver*. Tool paper at http://yices.csl.sri.com/tool-paper.pdf, August 2006.

21. D. Dvorak, R. Rasmussen, G. Reeves, and A. Sacks. Software architecture themes in JPL's Mission Data System. In *IEEE Aerospace Conf. USA*, 2000.
22. E. W. Adams. A primer of probability logic. *CSLI Publications*, 1998.
23. Steven Eker, Jose Meseguer, and Ambarish Sridharanarayanan. The Maude LTL model checker and its implementation. In *Model Checking Software: Proc. 10th Intl. SPIN Workshop*, pages 230–234. Springer LNCS, 2003.
24. Ronald Fagin, Joseph Y. Halpern, and Nimrod Megiddo. A logic for reasoning about probabilities. *Information and Computation*, 87:78–128, 1990.
25. Stephen Farrell and V. Cahill. *Delay- and Disruption-Tolerant Networking*. Artech House, Inc., Norwood, MA, 2006.
26. Brian P. Gerkey, Roger Mailler, and Benoit Morisset. Commbots: Distributed control of mobile communication relays. In *Proc. AAAI Workshop on Auction Mechanisms for Robot Coordination (AuctionBots)*, pages 51–57, Boston, MA, July 2006.
27. Heather J. Goldsby, Betty H. Cheng, and Ji Zhang. AMOEBA-RT: run-time verification of adaptive software. In *Models in Software Engineering: Workshops and Symposia at MoDELS 2007, Reports and Revised Selected Papers*, pages 212–224. Springer-Verlag, 2008.
28. Alwyn Goodloe, Carl A. Gunter, and Mark-Oliver Stehr. Formal prototyping in early stages of protocol design. In Catherine Meadows, editor, *Proc. POPL 2005 Workshop on Issues in the Theory of Security, WITS 2005*, pages 67–80, 2005.
29. Diana Gordon, William Spears, Oleg Sokolsky, and Insup Lee. Distributed spatial control, global monitoring and steering of mobile physical agents. In *Proc. IEEE Int. Conf. Information, Intelligence, and Systems*, pages 681–688, 1999.
30. Siegfried Gottwald. *A Treatise on Many-Valued Logics*. Research Studies Press, 2001.
31. Susanne Graf and Hassen Saïdi. Construction of abstract state graphs with PVS. In *CAV '97: Proc. 9th Int. Conf. Computer Aided Verification*, pages 72–83, London, 1997. Springer-Verlag.
32. Zheng Guo, Gioele Colombi, Bing Wang, Jun-Hong Cui, Dario Maggiorini, and Gian Paolo Rossi. Adaptive routing in underwater delay/disruption tolerant sensor networks. In *Fifth IEEE/IFIP Annual Conf. on Wireless On Demand Network Systems and Services (WONS'08)*, 2008.
33. Sebastian Gutierrez-Nolasco, Nalini Venkatasubramanian, Mark-Oliver Stehr, and Carolyn L. Talcott. Towards adaptive secure group communication: Bridging the gap between formal specification and network simulation. In *12th IEEE Pacific Rim International Symposium on Dependable Computing (PRDC 2006), 18-20 December, 2006, University of California, Riverside, USA*, pages 113–120, 2006.
34. H. Kautz. Satplan04: Planning as satisfiability. In *IPC4, ICAPS*, 2004.
35. Klaus Havelund. Using runtime analysis to guide model checking of Java programs. In *Proc. 7th Int. SPIN Workshop on SPIN Model Checking and Software Verification*, pages 245–264. Springer-Verlag, 2000.
36. Federico Heras, Javier Larrosa, and Albert Oliveras. MiniMaxSat: A new weighted Max-SAT solver. In *Int. Conf. Theory and Applications of Satisfiability Testing*, pages 41–55. Addison-Wesley, 2007.
37. http://xtune.ics.uci.edu.
38. Chalermek Intanagonwiwat, Ramesh Govindan, Deborah Estrin, John Heidemann, and Fabio Silva. Directed diffusion for wireless sensor networking. *IEEE/ACM Trans. Netw.*, 11(1):2–16, 2003.

39. Will Ivancic, Wes Eddy, Lloyd Wood, Dave Stewart, Chris Jackson, James Northam, and Alex da Silva Curiel. Delay/disruption-tolerant network testing using a LEO satellite. In *Eighth Annual NASA Earth Science Technology Conf.*, 2008.
40. V. K. Janakiram, D. P. Agrawal, and R. Mehrotra. A randomized parallel backtracking algorithm. *IEEE Trans. Comput.*, 37(12):1665–1676, 1988.
41. Henry Kautz and Bart Selman. Pushing the envelope: Planning, propositional logic, and stochastic search. In Howard Shrobe and Ted Senator, editors, *Proc. Thirteenth National Conf. Artificial Intelligence and the Eighth Innovative Applications of Artificial Intelligence Conf.*, pages 1194–1201. AAAI Press, 1996.
42. Minyoung Kim, Mark-Oliver Stehr, Jinwoo Kim, and Soonhoi Ha. An application framework for loosely coupled networked cyber-physical systems. In *Proc. 8th IEEE Intl. Conf. on Embedded and Ubiquitous Computing (EUC'10)*, 2010.
43. Minyoung Kim, Mark-Oliver Stehr, Carolyn Talcott, Nikil Dutt, and Nalini Venkatasubramanian. Combining formal verification with observed system execution behavior to tune system parameters. In *5th Int. Conf. on Formal Modelling and Analysis of Timed Systems (FORMATS'07)*, volume 4763 of *LNCS*, pages 257–273, 2007.
44. Minyoung Kim, Mark-Oliver Stehr, Carolyn Talcott, Nikil Dutt, and Nalini Venkatasubramanian. A probabilistic formal analysis approach to cross-layer optimization in distributed embedded systems. In *9th IFIP Int. Conf. on Formal Methods for Open Object-based Distributed Systems (FMOODS'07)*, volume 4468 of *LNCS*, pages 285–300, 2007.
45. Minyoung Kim, Mark-Oliver Stehr, Carolyn Talcott, Nikil Dutt, and Nalini Venkatasubramanian. Constraint refinement for online verifiable cross-layer system adaptation. In *DATE '08: Proc. Design, Automation and Test in Europe Conference and Exposition*, 2008.
46. Minyoung Kim, Carolyn L. Talcott, and Mark-Oliver Stehr. A distributed logic for networked cyber-physical systems. In *To appear in Proc. Intl. Conf. on Fundamentals of Software Engineering (FSEN'11)*, *LNCS*, 2011.
47. Moonzoo Kim, Mahesh Viswanathan, Sampath Kannan, Insup Lee, and Oleg Sokolsky. Java-mac: A run-time assurance approach for Java programs. *Form. Methods Syst. Des.*, 24(2):129–155, 2004.
48. Marta Kwiatkowska, Gethin Norman, and David Parker. Probabilistic symbolic model checking with PRISM: A hybrid approach. *Int. J. Softw. Tools Technol. Transf.*, 6(2):128–142, 2004.
49. Javier Larrosa, Federico Heras, and Simon de Givry. A logical approach to efficient max-sat solving. *Artif. Intell.*, 172(2-3):204–233, 2008.
50. Martin Leucker. Checking and enforcing safety: Runtime verification and runtime reflection. *ERCIM News*, (75):35–36, October 2008.
51. Martin Leucker and Christian Schallhart. A brief account of runtime verification. *Logic and Algebraic Programming*, 78(5):293–303, May/June 2009.
52. Xu Li, Wei Shu, Minglu Li, Hongyu Huang, and Min-You Wu. DTN routing in vehicular sensor networks. In *Global Telecommunications Conf., 2008. IEEE GLOBECOM 2008i*, pages 1–5.
53. Michael R. Lowry, Andrew Philpot, Thomas Pressburger, and Ian Underwood. A formal approach to domain-oriented software design environments. In *KBSE*, pages 48–57, 1994.
54. Michael R. Lyu, editor. *Software Fault Tolerance*. John Wiley and Sons, Inc., 1995.
55. Pattie Maes. Concepts and experiments in computational reflection. *SIGPLAN Not.*, 22(12):147–155, 1987.

56. Roger Mailler. Using prior knowledge to improve distributed hill climbing. In *IAT '06: Proc. IEEE/WIC/ACM Int. Conf. Intelligent Agent Technology*, pages 514–521, Washington, DC, 2006. IEEE Computer Society.
57. Roger Mailler and Victor Lesser. Solving distributed constraint optimization problems using cooperative mediation. In *AAMAS '04: Proc. Third Int. Joint Conf. Autonomous Agents and Multiagent Systems*, pages 438–445, Washington, DC, 2004. IEEE Computer Society.
58. Margaret Martonosi. ZebraNet and beyond: Applications and systems support for mobile, dynamic networks. In *CASES '08: Proc. 2008 Int. Conf. Compilers, Architectures and Synthesis for Embedded Systems*, pages 21–21, New York, NY, 2008. ACM.
59. Maude System. http://maude.csl.sri.com.
60. Pragnesh Jay Modi, Milind Tambe, and Makoto Yokoo. Adopt: Asynchronous distributed constraint optimization with quality guarantees. *Artificial Intelligence*, 161:149–180, 2005.
61. Amy L. Murphy, Gian Pietro Picco, and Gruia-Catalin Roman. Lime: A coordination model and middleware supporting mobility of hosts and agents. *ACM Trans. Softw. Eng. Methodol.*, 15(3):279–328, 2006.
62. N. Muscetolla, P. Pandurang, B. Pell, and B. Williams. Remote Agent: To Boldly Go Where No AI System Has Gone Before. *Artificial Intelligence*, 103(1–2):5–48, 1998.
63. Networked Cyber-Physcial Systems at SRI. http://ncps.csl.sri.com.
64. Matthew Richardson and Pedro Domingos. Markov logic networks. *Machine Learning*, 62:107–136, Feb. 2006.
65. Grigore Rosu and Klaus Havelund. Monitoring Java programs with Java PathExplorer. In *Proc. Runtime Verification (RV)*, pages 97–114. Elsevier, 2001.
66. Harald Rueß and Natarajan Shankar. Introducing Cyberlogic. 2003.
67. John Rushby. The design and verification of secure systems. In *Eighth ACM Symposium on Operating System Principles (SOSP)*, pages 12–21, Asilomar, CA, Dec. 1981. (ACM *Operating Systems Review*, vol. 15, no. 5).
68. John Rushby. *Partitioning for Avionics Architectures: Requirements, Mechanisms, and Assurance*. NASA Contractor Report CR-1999-209347, NASA Langley Research Center, June 1999. Also to be issued by the FAA as DOT/FAA/AR-99/58 available at http://www.tc.faa.gov/its/worldpac/techrpt/ar99-58.pdf.
69. John Rushby. Just-in-time certification. In *12th IEEE Int. Conf. Engineering of Complex Computer Systems (ICECCS)*, pages 15–24, Auckland, New Zealand, July 2007. IEEE Computer Society. Available at http://www.csl.sri.com/ rushby/abstracts/iceccs07.
70. John Rushby. Software verication and system assurance (invited paper). *SEFM*, 2009.
71. Hassen Saïdi and Natarajan Shankar. Abstract and model check while you prove. In Nicolas Halbwachs and Doron Peled, editors, *Computer-Aided Verification (CAV'99)*, number 1633 in Lecture Notes in Computer Science, pages 443–454, Trento, Italy, July 1999. Springer-Verlag.
72. Koushik Sen, Abhay Vardhan, Gul Agha, and Grigore Rosu. Efficient decentralized monitoring of safety in distributed systems. In *26th Int. Conf. Software Engineering (ICSE'04)*, pages 418–427, 2004.
73. Koushik Sen, Mahesh Viswanathan, and Gul Agha. Statistical model checking of black-box probabilistic systems. In *16th Conf. Computer Aided Verification (CAV)*, volume 3114 of *LNCS*, pages 202–215. Springer, 2004.

74. Mark-Oliver Stehr, Minyoung Kim, and Carolyn L. Talcott. Toward distributed declarative control of networked cyber-physical systems. In Z. Yu, R. Liscano, G. Chen, D. Zhang, and X. Zhou, editors, *Proc. 7th Int. Conf., Ubiquitous Intelligence and Computing (UIC'10)*, volume 6406, pages 397–413, 2010.
75. Mark-Oliver Stehr and Carolyn Talcott. Planning and learning algorithms for routing in disruption-tolerant networks. In *Proc. IEEE Military Communications Conference (MILCOM'08)*, 2008.
76. Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An introduction*. MIT Press, 1998.
77. S. Toorian, K. Diaz, and S. Lee. The CubeSet approach to space access. In *Aerospace Conf., IEEE*, 2008.
78. Wilfredo Torres-Pomales. *Software Fault Tolerance: A Tutorial*. Technical report, NASA, October 2000.
79. V. Jacobson, D.K. Smetters, J.D. Thornton, M.F. Plass, N. Briggs, R. Braynard. Networking named content. In *Fifth ACM Int. Conf. Emerging Networking EXperiments and Technologies (CoNEXT 2009)*, 2009.
80. VeStA Tool. http://osl.cs.uiuc.edu/ ksen/vesta2.
81. Guojun Wang and Hongjun Zhou. Quantitative logic. *Inf. Sci.*, 179(3):226–247, 2009.
82. Brian C. Williams, Michel Ingham, Seung H. Chung, and Paul H. Elliott. Model-based programming of intelligent embedded systems and robotic space explorers. *Proc. IEEE*, 91(3):212–237, January 2003.
83. Brian C. Williams and P. Pandurang Nayak. A model-based approach to reactive self-configuring systems. In *Proc. AAAI-96*, pages 971–978, 1996.
84. XG Reasoner. http://www.springerlink.com/content/25021851k303tlu0.
85. xTune Framework. http://xtune.ics.uci.edu.
86. Yu Yang, Xiaofang Chen, Ganesh Gopalakrishnan, and Robert M. Kirby. *Runtime Model Checking of Multithreaded C/C++ Programs*. Technical report, University of Utah, March 2007.
87. Makoto Yokoo. *Distributed constraint satisfaction: Foundations of cooperation in multi-agent systems*. Springer-Verlag, London, UK, 2001.
88. Håkan L. S. Younes and Reid G. Simmons. Statistical probabilistic model checking with a focus on time-bounded properties. *Inf. Comput.*, 204(9):1368–1409, 2006.
89. Weixiong Zhang, Guandong Wang, Zhao Xing, and Lars Wittenburg. Distributed stochastic search and distributed breakout: Properties, comparison and applications to constraint optimization problems in sensor networks. *Artif. Intell.*, 161(1-2):55–87, 2005.