Analysis and Design of Uncertain Cyber-Physical Systems

Alessandro Pinto

Several sources of uncertainty have to be taken into account in the analysis and design of CPS. The set of parameters used in the model of the physical plant of a CPS may be uncertain due, for example, to manufacturing processes that are precise up to some bounded tolerance. Physical quantities are sensed by electronic components that add noise to the sensed signals. Abstraction of the physical world, which is often necessary to limit the complexity of the models used in analysis and at run-time in decision-making, leads to non-determinism. The cyber side of a CPS, which includes both hardware and software components, exposes several types of uncertainty such as failures, latency, and implementation errors.

Design processes and tools allow engineers to minimize the impact of these types of uncertainty, and to deliver systems which can be operated with an acceptable level of risk. Control applications tend to have a stable computational workload as filtering, estimation, and control are implemented by software tasks running the same computation independently of data. Real-time operating systems, can provide hard guarantees on the execution of control functions, and several methods can be used to ensure the worst-case execution time of software. Similarly, time-triggered or priorities-based communication protocols can provide hard bounds on the delay experienced by messages exchanged over a network. Failures can be handled through redundant architectures as is typical in safety critical systems. Finally, protection from cyber-attacks can be achieved through physical isolation, careful control of the supply-chain, and other strategies such as authentication and encryption at the boundary of the system. Furthermore, many cyber-physical systems are employed in the automation of processes for which sound models are available. These models have enabled formal analysis and synthesis which further reduce human errors.

In the past several years, cyber-physical systems have evolved, primarily due to pervasive connectivity, miniaturization, cost-effectiveness of hardware, and advances in the area of Artificial Intelligence. The complexity of the cyber side of

Alessandro Pinto
Raytheon Technologies Research Center, 2855 Telegraph Avenue, Berkeley, CA, e-mail: alessandro.pinto@rtx.com

the CPS is constantly increasing as more functionality is moved to software. Correspondingly, the hardware platforms on which the software runs have become more complex. Another evolution has occurred in the area of communications at all levels of the protocol stack allowing disparate computing platforms to be interconnected. Connectivity is pervasive today leveraging technologies such as WiFi, 4G and 5G, and higher-level protocols for the internet of things such as the Message Queuing Telemetry Transport (MQTT). The ability to connect sensing and actuation nodes to the internet enables deployment over a cloud computing infrastructure where data can be aggregated, and computationally expensive algorithms can be executed with virtually unlimited resources. This trend has driven the development of new applications that control societal-scale systems such as the electric grid, transportation, and logistics. Coupling between the physical world and the cyber world can also occur in unconventional ways such as through social media platforms that shape consumer preferences and even opinions of people, driving their physical acts.

These new class of applications features an environment which is much more complex to model than traditional physical systems due not only to their scale, but also to new sources and types of uncertainty. Consider, for example, the typical case of echo chambers which is attributed to the effect that machine learning algorithms have on the bias of people. Such behavior is not easily predictable because of high uncertainty in the environment (people), which is only approximately represented by machine learning models, but that is inherently due to lack of knowledge. New models and analysis methods are therefore needed to capture different types of uncertainties, and to analyze these new classes of systems.

In this chapter, we start by discussing how cyber-physical systems have evolved from simple controllers to networks of highly autonomous systems. In Section 2, we describe some of the most common sources of uncertainty induced by the platform supporting the software of a CPS, and in Section 3 we review the engineering methods used for analysis and implementation, aiming at reducing the risks due to uncertainty. These two sections show the complexity of dealing with many sources of uncertainty, and how current practice makes use of several models tailored to the analysis of specific classes of uncertainty. In Section 4 we discuss the need for more autonomous systems, the new challenges that they bring in terms of analysis and implementation, and the need for new methods and tools to reason compositionally about aleatoric and epistemic uncertainty. Finally, in Section 5 we present a design methodology and a modeling paradigm that addresses these gaps.

1 The Evolution of Cyber-Physical Systems

In the last decade, we have witnessed an evolution of the architecture and application domains of CPS. Systems that control physical processes are functionally decomposed into a plant and a controller. The plant comprises a set of physical variables that evolve over time according to certain dynamics. The controller observes some of these variables through sensors, tracks their evolution over time,

and computes actions that are translated into physical effects by actuators. All these elements have evolved in complexity, from control systems to the most advanced form of autonomous systems such as self-driving vehicles and teams of Unmanned Aerial Vehicles (UAVs).

In traditional control systems, the implementation platform is an embedded system with analog and digital Input/Output (I/O), a processing unit, local storage, and communication interfaces. The I/O are connected to sensors and actuators, and the estimation and control laws are implemented as a software function that runs periodically according to the time-scale of the physical process to be controlled. In these systems, the model of the plant is typically known. The controller is designed using traditional tools such as stability analysis.

Several control systems can be connected over a network to form a *networked control system*. This type of systems are used in those applications that span larger physical plants such as a car, an aircraft, or a building. Networked control systems are typically federated: many local sensing and actuation points are connected to a controller, which in turn may be connected to a supervisory controller. The functionality of a sensing point is typically signal conditioning, or computation of threshold crossings, while actuation points implement a local controller for a physical device that maintains a given set point. The set point is computed by the supervisory controller according to some pre-defined policies. Such architecture can be found in building controls for instance, where the supervisory controller decides set-points for chillers and variable air volume terminal units depending on occupancy and time of day, while local controllers track the temperature of individual zones and control the mix of cold/hot air. The communication demand over the network that links these systems is low, the medium can be considered reliable, and the messages among systems are not subject to stringent delay or jitter requirements.

As connectivity becames more pervasive, and embedded platforms less power hungry, more capable, and less expensive, local controllers (even if elementary such as switches) could be networked as well. The *internet of things* is a term that was coined at the edge of the current century. Being embedded in physical objects, these new class of CPS are characterized by addressable physical things at larger scale (millions, billions or even tens of trillions of devices [12]). Connectivity gives rise to new opportunities as the massive amount of data collected by these devices can be processed to create models using new machine learning techniques for prediction, maintenance, and optimization. However, connection among nodes can no longer be assumed reliable, and the applications that rely on such connections cannot be safety critical. In these types of systems, new nodes can join the network while others can leave, and nodes need to be able to describe their capabilities.

Moving along the functional axis, CPS have evolved to automate more complex functions that have been traditionally under the responsibility of humans. *Autonomous systems* are a natural evolution of CPS, and autonomy is a feature orthogonal to the evolution trajectory described above. Autonomous systems, in fact, can be localized or distributed, and networked over a range of communication technologies. We can contrast traditional controlled systems shown in Figure 1, and autonomous systems shown in Figure 2. In control systems, the environment in which the system

operates is bounded within an envelope of the physical variables that constitute the state of the system. It is also bounded in terms of the interaction points with the human operator. The inputs to the system are encoded as a finite set of well-defined commands, while the outputs are values of state variables and modes of the systems displayed to the user. The interaction, in fact, is mediated by a user interface that can be implemented by a display and a set of switches or numerical set points. All higher level commands, which could represent a large class, are elaborated by humans that plan for lower-level commands sent to the system. The controlled system senses physical variables and updates the state estimate using accurate models. A control law, also derived using models, maps the belief state into actuation that can be applied with known error bounds. Control laws operate at short time scales which makes predictions fairly accurate. The state is typically encoded as a vector of continuous state variables representing the physical variables in the environment. They are also typically associated with a known probability density function. In some cases, a supervisory control may change the control law in a pre-defined way depending on the command received by the user.

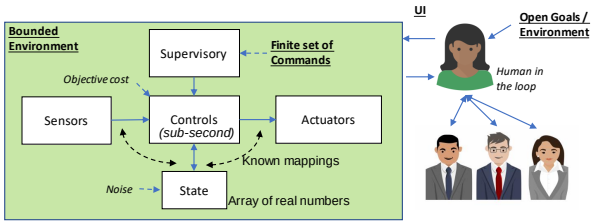

Fig. 1 A traditional controlled systems

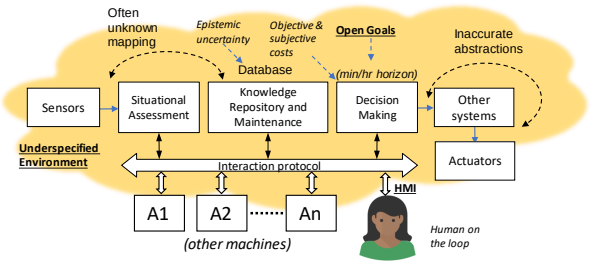

Fig. 2 A multi-agent system featuring autonomous agents with local situational awareness and decision-making capabilities.

The shift of tasks from humans to the autonomous system exposes a set of other challenges. First, the environment and the goals that were assessed and understood by humans, have to be now understood by the system. Both the environment and goals

become hard to bound at design time, resulting in an under-specified environment and an open set of goals. The autonomous system is in charge of using the sensed information to assess the situation. However, the semantic gap between the sensed variables and what they mean in terms of state of the environment is much higher than in control systems. A classic example is the identification and classification of objects from images. In many cases, models that related sensed variables to situations are not known, and are approximated by parametric or non-parametric models learned from data. Goals have to be understood and then decomposed by the system through planning. Plans span a much longer time horizon than control systems and predictions become much more uncertain. Moreover, optimality is judged using subjective measures: cost functions are defined by humans to encode preferences such as choices that are more or less ethical or safe, they are evaluated over an internal world model, and they are projected in the future with high uncertainty. Even the effect of actions is hard to model as it is the result of abstractions that are needed to reduce the complexity of the planning problem. Finally, autonomous systems are also typically networked and interact by negotiating actions and sharing information contributing to a potential explosion of uncertainty in the global state.

2 Sources and types of uncertainty

Several previous work have investigated the sources and types of uncertainty to be considered in the design of CPS. For example, the work in [24] provides a taxonomy of uncertainty, risks or opportunities, mitigation strategies, and outcomes. Uncertainty can arise from lack of knowledge or lack of definition, and it can be either statistically characterized, a known unknown, or an unknown unknown. The risks associated with uncertainty can range from catastrophic to system performance degradation, they can have impact on cost and schedule, or they can present new opportunities such as emergent capabilities. Typical mitigation strategies include margins, redundancy, generality, and upgradability. Finally, the outcomes include reliability, robustness, flexibility, and evolvability. The work in [42] provides a taxonomy of uncertainty introduced either at the requirements, design, or run-time stage of a system lifecycle. Requirements could be incomplete, ambiguous, or unsatisfiable. The design could be inadequate because of unexplored alternatives, or lack of traceability. At run-time, the system may experience failures, unpredictable environments, or inaccurate estimate of the state of the world.

For formal analysis, a mathematical framework is required for representing uncertainty. There are two types of uncertainty that should be considered: aleatoric and epistemic. Aleatoric uncertainty is inherent in the observed phenomenon and cannot be reduced. For example, the measurement of a physical quantity is the combination of the actual value plus an additive noise term that depends on several factors such as thermal noise. In this case, the measurement signal is represented by a stochastic process which can be characterized by its properties such as ergodicity and stationarity, and by its statistics such as mean and autocorrelation, or even by a probability

density function. In other cases, uncertainty is related to the parameters of a system. For example, the mechanical properties of a system which directly contribute to the parameters of the model of a plant might not be exactly known due to the manufacturing process which cannot be accurately controlled. It is, however, far from uncommon to incur in lack of knowledge about uncertainty, and other models might be more appropriate such as the Dempster-Shafer theory of evidence [14, 45] or fuzzy sets [52]. Other methods that directly address epistemic uncertainty will be discussed later in Section 5.

While these representations are useful to develop algorithms for formal analysis of uncertain systems, their application to system design requires additional methods and tools to identify relevant quantities, associated uncertainties, and efficient modeling methodologies. In a model-driven development process, the functional view of the CPS is an architecture which includes the environment, the variables that are sensed, the block diagram of the estimation and control functions, and the actuation signals that are sent back to the environment. This model is then implemented by what we have named the *platform* where sensed variables are mapped to sensors, the estimation and control functions are mapped to software modules, and the actuation signals are mapped to actuators. Links among functional blocks are implemented by communication paths that could be simply variables held in memory, or connections over a network. The semantics of the model determines the execution policy of the software that is supported by an operating system. The platform includes several sources of uncertainty that affect availability and integrity of data, correctness, and precision in actuation. We now list some sources of uncertainty introduced by a platform supporting a CPS application together with models used for analysis.

Failures. All components are subject to failures. A component such as a sensor, an actuator, a processor or any other electronic or mechanical component can be characterized by the probability distribution $F(t) = P(T \leq t)$ of its time to failure T, the probability that the component fails before t [29]. Let $f(t)$ denote the corresponding probability density function. A metric often used in reliability models is the Mean Time To Failure (MTTF) (or Mean Time Between Failures if the component is serviceable) which is the first moment of $f(t)$.

Components may fail in different ways, each having a different impact on system safety and performance. For example, a sensor may stop functioning, or may generate erroneous measurements. These two different failure modes have different consequences and require different identification mechanisms at run-time. The Failure Mode and Effect Analysis (FMEA) [48] methodology is used for this type of analysis.

Fault Tree Analysis [47] is used to determine the probability of critical events to occur as a result of failures. In a fault tree, basic event (the leaves) are composed in different ways all the way up to a top-level event of interest. Starting from the failure statistics of the basic events, quantitative analysis can then be used to compute the probability of the top-level event.

Noise or imprecision. Sensors measure physical quantities and generate an electrical signal. Such signal reflects the physical quantity up to a certain accuracy, and with added noise. Actuators are also imprecise: there is a difference between the electrical

signal and the achieved physical effect. Computation may have to be implemented using finite precision hardware. Finally, the parameters of the plant model may not be precisely known.

Timing. Characteristic of cyber-physical systems is that they control physical processes. This means that the execution of the control algorithm must follow the same tempo of the environment. In other words, execution time is a fundamental contributor to the correctness specification of the control algorithms. The execution time is difficult to predict in general because it is affected by several factors including optimizations done by compilers when generating machine code, delays induced by memory access which varies depending on the location of data and instructions (main memory, or cache), and instruction processing pipelines. Analysis tools aim at estimating the Worst Case Execution Time (WCET) of a task [50]. For networked applications, the communication delay between two tasks depends on the type of protocols used by the network. Predicting communication delays is also often done using worst-case analysis as in Network Calculus [34]. Dedicated techniques can be used if the protocol is time-triggered, or priority-based. For other protocols such as WiFi, probabilistic models based on Markov Chains are also available for analysis (see early work in [10] for example). Given the execution semantics of a set of communicating tasks, these models can be used to predict the end-to-end delay of a control function.

Security. For cyber-physical systems there are both cyber and physical security concerns. Example of cyber-attacks include standard attacks on communication networks. However, more complex attacks may come from the supply chain where counterfeit components containing malicious elements may end up being used in actual products. The physical side of the system can also come under attack either by directly tampering with sensors, or by forcing the environment to go outside the envelope for which the system has been designed. Attacks can be characterized by different metrics including their likelihood (depending on how hard it would be for the attacker to succeed), and cost of their consequences. Attack trees [38], which are similar to fault trees, can be used to analyze the cost and likelihood of an attack (which often involves multiple steps).

Implementation errors. Hardware and software components may include implementation errors. We consider the case where the requirements or specification of the hardware or software component is correct, but the implementation introduces errors. Such errors are not acceptable, and they are not statistically characterized as in the case of failures. Rather, design processes and architectural solutions are used to minimize the risk of errors going undetected in the final implementation.

3 Dealing with uncertainty: A short review

The sources of uncertainty coming from the platform described in the previous section impact the safety and performance of a CPS. However, engineering methods are able to deliver remarkably safe systems. For example, the probability of

loss of control of a commercial aircraft is indeed 10^{-9} per flight hour as shown by the statistics kept updated by The Boeing Company [2]. Design processes and tools have been developed to identify and remove uncertainty towards delivering high-assurance systems. An example of guidelines can be found in aviation where standards such as the ARP4661 [4] and ARP4754 [3] provide considerations on how to analyze, categorize and deal with hazards, and how to derive requirements that, when implemented, minimize the risk of catastrophic events.

Analysis and elimination of uncertainty is not achieved with the use of one system model, but rather with dedicated techniques for different aspects of the design, some of which have been referenced in the previous section. Each model focuses on the effective computation of key performance parameters and on the qualitative or quantitative analysis of uncertainty.

Dealing with failures: guaranteeing availability and integrity. Redundancy is a way to deal with failures. When the measurement of a physical quantity is essential to safe operations, two or three sensors can be used to deal with either permanent failures or the generation of wrong measurements. The outputs of these sensors are the inputs of a voting scheme that decides which measurement to trust. Redundancy is also used for computing platforms where multiple processors execute functionally equivalent software. To avoid common mode failures, processors may also be different, as well as the programming language and the compilers used to generate embedded code, as done in the Boeing 777 primary control system [51]. Redundancy is a basic mechanism also used in communication systems where hosts are linked by multiple paths with independent links, and control bits are added to encoded information to detect and correct errors at the receiving end.

Guaranteeing security. Denying physical access to potential attackers is one possible measure that can be effective in some settings. However, as systems become more connected, and supply chains more global, the attack surface expands and other methods are needed such as the use of secure operating systems [31], and techniques to control the pedigree of hardware components [44]. Physical attacks, require not only increasing the observability of the plant through redundant sensors, but also designing monitors to identify and isolate potential attacks [39].

Guaranteeing timely execution. The total delay from sensors to actuators must be bounded and small enough to follow the time-scale of the physical processes to be controlled. There could be uncertainty in the execution time of software and in the communication delay among hardware and software components. To remove this uncertainty, several techniques can be used both at the hardware and software level. In many cases, time determinacy is favored over performance and many useful advanced implementation methods such as compiler optimization, caching, and dynamic memory allocation are not used. These restrictions are needed to derive a good and reliable estimate of the worst-case computation time of tasks executed on a CPU.

Once the computation time is known, schedulability analysis provides the framework for allocating CPU time to tasks. Consider a set of N tasks, each characterized by a period of execution T_i (which in this simplified setting also corresponds to the task deadline) and worst case execution time W_i. A scheduling policy assigns prior-

ities to these tasks so that they don't miss their deadlines. Typical policies include the rate-monotonic static priority assignment, and the earliest deadline first (EDF) dynamic priority assignment schemes. For both, there are simple schedulability tests that can assure that no task will miss their deadline. For example, when the EDF policy is used, a utilization $U = \sum_{i=1}^{N} W_i/T_i$ less than or equal to one guarantees schedulability. In practice, the total utilization is kept much lower than one (a case of margin) to deal with potential uncertainty. Tasks are then scheduled by a Real-Time Operating System (RTOS) which provides hard guarantees on typical services such as context switching (which is instead uncertain for general operating systems).

When communication among tasks occur over a network, the network delay must also be bounded and not uncertain. Communication protocols for cyber-physical systems have been developed to guarantee bounded transmission time. Some protocols allow for the definition of priorities associated with messages, and priority assignment can be used to bound the maximum delay experienced by a message. Other systems, such as the Time Triggered Protocol (TTP) guarantee time determinacy by assigning periodic time slots to the nodes in a network.

Being robust against noise and imprecision. Filtering, and robust control [53] are common techniques to deal with bounded uncertainty about the model of the plant, and to reject several types of disturbances.

Developing bug free software. From the functional specification of a control system to the final implementation, mistakes can be made in several places. The functional specification can be manually implemented in software or, in some cases, encoded into high-level models from which code can be generated. Software runs on a processing unit which is a hardware component that may also contain implementation errors. Similarly, some functions could be implemented in hardware and implementation errors can also be made in this case. To remove this uncertainty, several strategies are put in place in a typical design flow such as code reviews, unit testing, and integration testing (where testing is done in simulation, using Hardware-In-the-Loop, or directly in the field). For high-assurance systems, independence and redundancy are two commonly used strategies: the software development team is kept isolated from the test and evaluation team, the same function is written in different programming languages, compiled using different compilers, and executed at run-time by different processors.

Several steps in the development process can also be automated. For example, Modified Condition / Decision Coverage [25] is a metric that defines how well a set of tests triggers all conditions and decisions in the code. Tools for the generation of tests aiming at maximizing coverage also exist. Finally, formal method tools such as model-checking [27] and abstract interpretations [9] can provide absolute assurance of software correctness.

Final Remarks

Before closing this section, it is worth mentioning modeling, analysis, and synthesis tools that in principle can be used to formally verify properties of a cyber-physical

system under uncertainty, or to guarantee the satisfaction of such properties by construction.

Stochastic hybrid systems [32] is a modeling paradigm for systems that can be characterized by a finite set of discrete locations, and where each location is associated with a noisy dynamical system (a set of stochastic differential equation). Probabilistic transitions among locations are associated with a guard that enables them, and a probabilistic reset map that changes the continuous state variables from the source state to a probability distribution in the target state. Typical analysis problems include reachability, safety, and stability, while design problem include optimal control. In principle, this model could accommodate uncertainties in the availability of sensors, and various delays. However, the failure of a sensor, for example, is a discrete event that requires introducing additional discrete locations whose number increases exponentially with the number of failures, limiting the scalability of these methods.

Correct by construction methods can also be used to automatically explore a design space and find design points that automatically satisfy typical constraints such as maximum communication delay, and other physical constraints on the implementation platform. These methods encode the design problem as a set of constraints whose solution is an actual design. Optimization (or heuristic search) is used to find optimal solutions. The flexibility gained by formulating the design exploration problem as an optimization problem allows for incorporating margins, redundancy, and other preferences into the problem formulation. Examples can be found in our previous work [41, 35, 36] in the context of synthesis of computation and communication platforms for networked embedded systems.

Additional measures can be designed to further reduce the impact of uncertainty on the operations of a cyber-physical system. Clearly, the environment can be designed in such a way to remove uncertainty. For example, warehouse robots such as the ones used by the Kiva system, rely on fiducial markers [17]. The operations of a system can also be restricted so that uncertainty does not lead to safety violations. For example, a flight plan for a commercial flight must guarantee the presence of a suitable alternate airport within a certain time from any point in the plan under a single engine operation [11].

4 Autonomy as a driver of new classes of uncertainty

There is strong *interest in automating tasks that are today done by humans and that are considered dull, dangerous, where machines could perform better, or where humans represent a significant operating cost*. Consider the case of commercial aviation. Two pilots are required for domestic flights, and three or more are required for longer flights. The need for pilots not only increases training and operating costs, but creates scheduling challenges as crews need rest time and cannot fly more than a certain number of hours per month. Moreover, a significant number of incidents are

due to human error. For example, it is estimated that the cost of runway incidents is tens of billion dollars per year [37].

Autonomy is not only a desire, but it is also a need in many application domains. For example, humans cannot match the pace of automation in assembly lines, in detecting and reacting to critical events, or in controlling many devices in parallel. In domains where applications cover a large physical space, individual systems finds themselves far from other systems, and yet they must continue the execution of coordinated plans. While wireless communication technologies have improved, and bandwidth abound, there are still availability challenges that prevent the deployment of centralized architectures. This is for example the case of the envisioned MOSAIC warfare environment [13], and the JADC2 framework [28]. In these environments, communication (which is often contested) is a scarce resource. Similar problems exist even in the case of the Internet-of-Things due to the limited size and power of connected objects, and to the potential for attacks. Thus, nodes in these networks must have a greater level of autonomy in terms of being able to sense and understand their surrounding environment, make decisions that require fast response time locally, predict the global state of the system, and maintain the ability to coordinate within coalitions. *Autonomy, then, is seen as one way to deal with distruption and long, unpredictable delays*.

For these new class of systems, traditional tools to deal with uncertainty, such as the ones mentioned in Section 3 are no longer sufficient. They have been developed to deal with uncertainty that is well-characterized such as noise in measurements, and equipment failures with known failure modes. Moreover, systems, even when distributed, are deployed in closed environments where communication can be implemented through reliable media and timing can be controlled. And finally, system integrators control the interfaces among sub-systems from different suppliers thereby guaranteeing some level of integrity of the information in and out of a system. In the case of autonomous systems, while these types of uncertainties are still present, epistemic uncertainty seems to be predominant. This type of uncertainty, is in principle reducible, albeit at a cost.

The need for understanding the environment demands for complex sensors. An autonomous car, for instance, is equipped with LiDARs, cameras, radars, and ultrasonic sensors. As already mentioned in Section 1, the mapping between the physical quantities measured by these sensors and logical facts useful for decision-making is not known in a form that can be analyzed and implemented. This means that, while locally a system still senses physical quantities such as the time of flight of a laser beam, it needs to be able to deduce high-level facts such as the intent of another actor in the environment. To overcome this problem, approximations of these functions are learned from data [7]. The predictions made by these models are subject to both aleatoric and epistemic uncertainty [26]. Epistemic uncertainty can be reduced by increasing the complexity of the models and the data set used for training, but both have an impact on complexity, and cost and time to deployment. The NVIDIA Drive AGX Pegasus [1], a platform for self-driving cars, is in fact a super-computer, delivering hundreds of tera-operations per second, and consuming hundreds of watts of power. Additional reasoning is required to deduce facts, make predictions about

the environment, and ultimately plan for actions. The complexity of the reasoning and planning algorithms depends on the complexity of the representation and the number of actions that can be taken by the autonomous agent. Reducing such complexity requires abstraction which adds uncertainty as some information must be lost. Moreover, the world in which these systems operate is very dynamic, meaning that the rules according to which the environment behaves may change over time. This is due to other agents in the environment that learn and adapt continuously in response to the behavior exhibited by a system. This is different from the case of a cyber-physical system whose behavior is driven by physical laws that can be considered immutable. Finally, not only communications among agents is imperfect, but systems can be attacked in many ways (both on the cyber and physical side [18]), meaning that the information they receive through the network or even through sensing of the physical world may be corrupted by malicious agent. In principle, these sources of epistemic uncertainty can be reduced by using complex models, complex communication schemes, or even by deploying infrastructure in the environment to simplify perception and situational assessment, but these solutions have an impact on the complexity and cost of the computing and communication platforms.

The additional complexity of autonomous cyber-physical systems brings new challenges to systems engineers. Guaranteeing availability through physical redundancy requires replication of expensive sensors and hardware modules. Falling back on less performing sensors or hardware is a potential solution if the system can be shown safe under the resulting lack of information. The identification of corrupted information requires additional sensors and complex models. The ability to guarantee a timely reaction to deal with contingencies or changes in objectives is not always possible as the running time of some tasks, such as optimization algorithms, is hard to predict and the worst case execution time hard to obtain or too conservative. The software implementation of the functionality of an autonomous system is hard to assure. The programming languages and data structures used in these applications, the heavy reuse of large third-party library, and the mapping over heterogeneous multi-processor platforms, render verification a challenging task. As a consequence, not only bugs may go undetected in the final product, but the code may fail in subtle ways.

The discussion points so far suggest that autonomous cyber-physical systems must be designed to deal with epistemic uncertainty arising from abstraction, approximation, and cost constraints. The design methods should explicitly capture lack of knowledge and its impact on the possible behaviors of a system. It is not necessary for an autonomous agent to have perfect knowledge about the environment, but rather enough knowledge to make the right decisions so that its behaviors achieve desired goals while satisfying given constraints. Given the inherent cost of removing epistemic uncertainty, we are interested in the requirement generation process which drives functional design and implementation.

5 The key elements of a new modeling paradigm

We first introduce a *functional architecture* that defines what needs to be modeled and helps to drive the process of requirement generation for the different elements of an autonomous cyber-physical system. Recognizing the inherent uncertainty in the goals of autonomous systems, we also seek a framework that facilitates integration of different capabilities and evolution over time. Furthermore, we also seek a framework that enables modular and scalable analysis and design. We will rely on a *compositional modeling framework* where components are captured by their contracts that specify assumptions and guarantees rather than their implementation. Assumptions and guarantee must be specified in a language that enables reasoning about several forms of uncertainty. We will rely on a *modal logic* which enables reasoning about probabilistic and epistemic uncertainty in a formal way. Finally, when components are specified at a level of abstraction that is detailed enough to be implemented, we discuss some promising *uncertainty quantification* methods approaches that can help in verifying that the implementation is compliant with the specified contract.

5.1 High-Level functional architecture

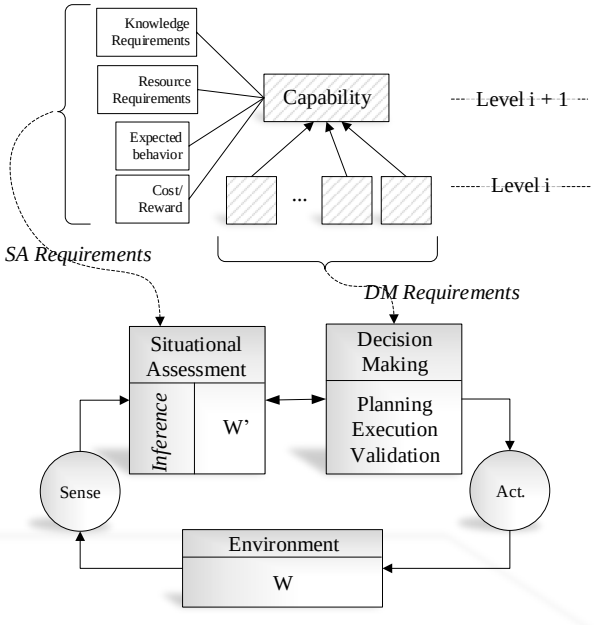

Fig. 3 Overview of the high-level functional architecture of an autonomous systems (bottom), the capability model, and the requirement allocation process (top).

In this section we describe a methodology to drive requirement definition for autonomous systems. We do not address a comprehensive list of requirements such as operational requirements, maintainability, sustainability, security, cost and durability. We also don't address completeness. Rather, the focus of this section is on requirements that are generated from an analysis of the functional architecture of a typical system as shown in Figure 3. A system is a collection of agents, but the abstraction level at which we discuss requirements seats above such internal structure. Independently of the number of agents, an autonomous system has a natural functional decomposition which includes a *situational assessment* function, and a *decision-making* function (see the early work in [16], and our recent work in [40] among many others). The situational awareness function makes sense of inputs to create an understanding of the current state of the world, and to make predictions over a given time horizon. The decision-making function, instead, decides actions to take towards exhibiting certain behaviors and/or bringing about certain states.

At the fundamental level, an autonomous agent (Figure 3) partially senses the environment state $w \in W$, and uses models to create an internal representation $w' \in W'$. The situational assessment function establishes a relation between the two, that qualitatively can be expressed as $R_{SA} \subseteq W \times W'$ (and that will be discussed in more detail in Section 5.2). Each estimate $w' \in W'$ corresponds to potentially many actual situations $w \in W$. Because decision-making has only access to the internal representation of the world, defining requirements on R_{SA} becomes essential. Notice that the estimate w' does not need to be accurate, but only accurate enough to guarantee good decisions. The definition of such requirements, therefore, should start from an analysis of the decision-making process and the types of capabilities that the agents may decide to use.

To define capabilities and decision-making, we find inspiration in the capability approach from economics [43]. A capability represents a potential that an agent has to realize a behavior or to achieve a certain state. A capability can be used only in certain contexts, namely when the agent has enough resources and is confident that the capability will succeed in its intent. For example, the capability of moving without collisions requires access to a map of the environment, and enough power to mechanically move the agent. Decision-making is the process of selecting the capability to use in a given state of the environment to achieve goals (that can be desired behaviors of states of the environment). Capabilities can be structured hierarchically where a group of capabilities at level i can be leveraged together in such a way to deliver a capability at level $i + 1$.

At the lowest level of this hierarchy, we find capabilities that interface directly with the physical world. These capabilities are managed by control agents that provide a set of dependable automation functions which we will call *skills*. Skills have a known set of failure modes that can be modeled and exposed to the next level of the decision hierarchy, and taken into account during the planning process. The timescale at which skills operate is small enough that predictions can be made with a known degree of accuracy. Examples of skills include low-level controls such as the trajectory following capability of an autopilot.

At the higher-levels of the decision hierarchy we find strategic planning. At this level, the representation of the state space tend to be large and unstructured, and time scales tend to become longer. As a result, it is in general harder to provide accurate estimates or projections about the world. Consider the case of detecting stop signs on the road by relying on images from a camera [30]. The function that relates pixel values to the presence of a stop sign is not known. Neural networks can be used to learn a correlation between the domain and range of such function, but it is an approximation that may lead to run-time errors. Other examples of this kind abound such as detecting and avoiding other aircraft in the sky, or predicting the intent of humans.

Decision-making includes planning, execution, and run-time validation. In general, planning results in policies (i.e., when and where to use capabilities) that are executed by the agent, which also uses run-time validation to assess how plan execution is going, and forecast potential issues that may arise in the future. There are two sets of requirements that can be derived from the analysis of the capability model. First, the agent must know if a capability can be used, and if and when it achieves its intended goals – both essential for plan execution, monitoring and validation. These requirements directly apply to situational awareness. Secondly, the agent must be able to deliver a capability at level i using capabilities at level $i + 1$. This second set of requirements directly apply to decision-making which has the freedom to select policies using $i + 1$-th level capabilities. Feasibility in all conditions as expressed by the i-th level capability, as well as optimality while respecting constraints must both be verified. Notice that these requirements are in general hard to verify because it is common to rely on the shape of the cost functions, and on the requirements for using a capability to enforce safety constraints or other metrics. Verification, therefore, must check that under all conditions, and all applicable goals, the computed plan is valid.

In addition to situational assessment and decision-making requirements, several other common requirements may need to be derived and assigned to agents, such as the ability to understand goals, to explain the reasons for a failure to find a plan or to achieve a goal, and to be able to achieve a safe state in any state of the world. Furthermore, validation and verification of a plan at run-time should not be based on the same models that are used in planning which could otherwise lead to common-mode failures. Independence requirements between planning and run-time verification should be considered. In the case of learning agents, several requirements should be included such as the ability to identify good events to learn from, to compute the extent of the revisions of the internal knowledge of the agent, to perform the update, and to safely resume operations. The overall requirement is that the system improves, meaning that its performance improves at least in some environments, while safety guarantees remain unaffected.

Both situational awareness and decision-making functions result in general from the composition of several sub-systems. For example, consider an autonomous vehicle. The situational assessment function will comprise components for dealing with road signs, pedestrians, and vehicle health. Each component will work under some assumptions on the environment such as good visibility. The decision-making

function will comprise components for controlling comfort inside the vehicle, the entertainment system, and the trajectory of the car. These components will only work under assumptions on the health of the mechanical equipment, and the condition of the road. Initial versions of an autonomous system may only include a limited set of capabilities, to be expanded by adding additional functions in later versions. Furthermore, several teams may contribute to the design of a system, with teams perhaps even belonging to different companies. Incremental deployment and multiple suppliers have to be supported by a corresponding incremental analysis and design methodology.

In the next section we introduce a formal framework to support the modeling, analysis, and design approach described in this section.

5.2 Compositional modeling framework and specification language

We will use a contract-based approach [8] to develop our compositional framework, and we will adopt the formalism in [5] which we briefly summarize here. A specification theory is a triple $(\mathcal{S}, \otimes, \leq)$, where $\mathcal{S}$ is a set of specifications, $\otimes : \mathcal{S} \times \mathcal{S} \rightarrow \mathcal{S}$ is a parallel composition operator over specifications, and $\leq \subseteq \mathcal{S} \times \mathcal{S}$ is a reflexive and transitive refinement relation. The refinement relation must be compositional, meaning that given specifications S, S', T and T', whenever $S \leq S'$ and $T \leq T'$, then $S \otimes T \leq S' \otimes T'$. A conjunction operator $\wedge : \mathcal{S} \times \mathcal{S} \rightarrow \mathcal{S}$ is used to combine specifications. The conjunction of two specifications S and T, written $S \wedge T$, if it exists, is the most general specification that realizes both. In some cases, a specification theory can be equipped with a quotient operator $/ \colon \mathcal{S} \times \mathcal{S} \rightarrow \mathcal{S}$. Given two specifications S and T, T/S, if it exists, is the most general specification such that $S \otimes (T/S) \leq T$. Finally, relativized refinement is a ternary relation in $\mathcal{S} \times \mathcal{S} \times \mathcal{S}$ on specifications. Given three specifications S, E, and T, $S \leq_E T$ if and only if for all $E' \leq E$, $S \otimes E' \leq T \otimes E'$. In words, S refines T when both are considered in an environment E' that refines E. It can be shown that relativized refinement is a preorder, meaning that for all specifications S, E, E', and T, $E \leq E'$ *and* $S \leq T \Rightarrow S \leq_{E'} T$.

A contract theory is built over a specification theory. Specifically, a contract is a pair $C = (A, G)$ where A and G are specifications called assumption and guarantee, respectively. The environment semantics of a contract is the set of specifications that refines the assumption: $[\![C]\!]_{env} = \{E \in \mathcal{S} | E \leq A\}$. The implementation semantics of a contract is the set of specifications that satisfy the guarantee G under assumption A: $[\![C]\!]_{impl} = \{I \in \mathcal{S} | I \leq_A G\}$. Two contracts are semantically equivalent if their environment and implementation semantics are the same, respectively. Moreover, a contract C is in normal form if for all specification $I \in \mathcal{S}$, $I \leq_A G$ if and only if $I \leq G$, which also means that for a contract $C^{nf} = (A^{nf}, G^{nf})$ in normal form, $[\![C^{nf}]\!]_{impl} = \{I \in \mathcal{S} | I \leq G^{nf}\}$. In some cases, depending on the specification theory, a contract $C = (A, G)$ can be transformed in a semantically equivalent contract $C^{nf} = (A, G^{nf})$ in normal form by weakening its guarantee.

Contracts are related by a refinement relation $\preceq$. A contract $C' = (A', G')$ refines $C = (A, G)$ if and only if $[\![C']\!]_{env} \supseteq [\![C]\!]_{env}$, and $[\![C']\!]_{impl} \subseteq [\![C]\!]_{impl}$. It can be shown that this condition corresponds to $A \leq A'$ and $G' \leq_A G$, and if contracts are in normal form, then the latter can be written as $G' \leq G$.

Contracts can be composed to yield a new contract. Let $C_1 = (A_1, G_1^{nf})$ and $C_2 = (A_2, G_2^{nf})$ be two contracts, then their composition is $C_1 \boxtimes C_2 = (A_1/G_2^{nf} \wedge A_2/G_1^{nf}, G_1^{nf} \otimes G_2^{nf})$. On the other hand, it is sometime easier to prove whether a set of contracts $\{C_1, \ldots, C_n\}$ is a correct decomposition of a contract C. This can be shown [33] to be equivalent to proving that:

$$\bigwedge_{1 \leq i \leq n} G_i^{nf} \leq G^{nf} \tag{1}$$

$$A \wedge \bigwedge_{1 \leq j \neq i \leq n} G_j^{nf} \leq A_i, \qquad \forall i \in [1, n] \tag{2}$$

We now need to define our specification theory. We use a multi-modal first-order language that allows reasoning about knowledge and probabilities and that takes inspiration from [19]. The use of a first order language allows us to also reason about objects, their properties, and their relationships which is convenient for autonomous systems. Let $\Sigma = (C, P, V)$ be a first order signature with the usual set of constant, predicate and variable symbols. We assume that the reader is familiar with the syntax and semantics of a first-order language [22, 21]. The syntax of the specification language that we will use is recursively defined as follows:

$$\phi := P^k(t_1, \ldots, t_k) | \neg\psi | \psi \wedge \psi' | \forall v.\psi | \mathsf{K}_i\psi | \sum_{i=1}^{n} q_i \mathsf{P}_{j_i}\psi_i \leq b$$

where a term t_i is a variable or a constant, $P^k \in P$ is a predicate symbol of arity k, ψ and ψ' are formulas, q_i and b are rational numbers, and $j_i \in [1, m]$. A formula $\mathsf{K}_i\psi$ denotes that agent i knows that ψ, while $\mathsf{P}_i\psi$ denotes the probability that agent i assigns to ψ being true.

The semantics of a formula in this language is the set of Kripke structures that satisfy the formula. A Kripke structure for a multi agent system of n agents is a tuple $M = (W, \{\sim_i\}_{1 \leq i \leq n}, \{P_i\}_{1 \leq i \leq n}, D, I)$ where W is a set of possible worlds, $\sim_i \subseteq W \times W$ is the accessibility relation for the i-th agent, P_i associates to each possible world $w \in W$ a probability space $(W_{i,w}, \mathcal{F}_{i,w}, \mu_{i,w})$, D is a fixed domain of objects, and I is an function that maps each possible world $w \in W$ to a first order interpretation I^w of the symbols in Σ. The satisfaction relation $\models$ that relates a Kripke structure M and a world $w \in W$ to the formulas that are satisfied by the model is defined as follows:

$$(M,w) \models P^k(t_1,...,t_k) \quad iff \quad (I^w(t_1),\dots I^w(t_k)) \in I^w(P^k) \tag{3}$$

$$(M,w) \models \neg\psi \quad iff \quad (M,w) \not\models \psi \tag{4}$$

$$(M,w) \models \psi \wedge \psi' \quad iff \quad (M,w) \models \psi \; and \; (M,w) \models \psi' \tag{5}$$

$$(M,w) \models \forall v.\psi \quad iff \quad \forall o \in D,\ (M,w) \models \psi(v \leftarrow o) \tag{6}$$

$$(M,w) \models \mathsf{K}_i\psi \quad iff \quad \forall w' \in W,\ w \sim_i w' \Rightarrow (M,w') \models \psi \tag{7}$$

$$(M,w) \models \sum_{i=1}^{n} q_i \mathsf{P}_{j_i}\psi_i \leq b \quad iff \quad \sum_{i=1}^{n} q_i \mu_{i,w}(W_{i,w}(\psi_i)) \leq b \tag{8}$$

Where $W_{i,w}(\psi) = \{w \in W_{i,w} | (M,w) \models \psi\}$, namely the set of worlds in $W_{i,w}$ that satisfy ψ. As discussed in [19], we can assume $W_{i,w}$ to be a subset of the set of possible worlds that the agent considers possible in w (consistency). We will also assume that the probability spaces are the same for all agents, i.e. $P_i = P_j$ for all $i, j \in [1..n]$ (objectivity), and that for all i, w and formula ψ, $W_{i,w}(\psi) \in \mathcal{F}_{i,w}$, meaning that all formulas define measurable sets.

We define our specification theory as follows. A specification is a pair $S = (\Sigma, \phi)$ where Σ is a first order signature, and ϕ is a formula in the specification language defined above over the signature Σ. The set of such specifications is denoted $\mathcal{S}_{KP}$. Given two specifications $S_1 = (\Sigma_1, \phi_1)$ and $S_2 = (\Sigma_2, \phi_2)$, their composition is $S_1 \otimes_{KP} S_2 = (\Sigma_1 \cup \Sigma_2, \phi_1 \wedge \phi_2)$. Moreover, $S_1 \leq_{KP} S_2$ if an only if $\Sigma_2 \subseteq \Sigma_1$, and $\phi_1 \models \phi_2$.

Proposition 1 $(\mathcal{S}_{KP}, \otimes_{KP}, \leq_{KP})$ *is a specification theory.*

Proof It is easy to show that the refinement operator is reflexive and transitive. It remains to show that it is compositional. Let $S = (\Sigma_S, \phi_S)$, $S' = (\Sigma'_S, \phi'_S)$, $T = (\Sigma_T, \phi_T)$, and $T' = (\Sigma'_T, \phi'_T)$ be specifications such that $S \leq_{KP} S'$ and $T \leq_{KP} T'$. Then, $\Sigma'_S \subseteq \Sigma_S$, $\phi_S \models \phi'_S$, $\Sigma'_T \subseteq \Sigma_T$, and $\phi_T \models \phi'_T$ hold (premise). Also, we have that $S \otimes_{KP} T = (\Sigma_S \cup \Sigma_T, \phi_S \wedge \phi_T)$, and $S' \otimes_{KP} T' = (\Sigma'_S \cup \Sigma'_T, \phi'_S \wedge \phi'_T)$. From the premise, we can deduce that $\Sigma'_S \cup \Sigma'_T \subseteq \Sigma_S \cup \Sigma_T$, and that $\phi_S \wedge \phi_T \models \phi'_S \wedge \phi'_T$, which implies that $S \otimes_{KP} T \leq_{KP} S' \otimes_{KP} T'$. □

In the rest of this section, we will consider the same universal signature for all specifications. In this case, it is not necessary to explicitly mention the signature of a specification. From this specification theory, we can build a contract theory as already described in the introductory notes to this section.

The last key element that we need to explore is the modeling methodology for autonomous multi-agent systems. The first order universal signature Σ is partitioned into sub-signatures $\Sigma_1, \dots, \Sigma_m$ for each agent, such that (C_i, P_i, V_i) is the signature for agent i. This signature contains the symbols that are used by the internal world model of the agent. The accessibility relation $\sim_i$ of an agent i is defined as follows: $(w, w') \in \sim_i$ if and only if for all $s \in C_i \cup P_i \cup V_i$, $I^w(s) = I^{w'}(s)$. This means that in the two possible worlds w and w', the internal world model of the agent is the same, and therefore the agent cannot distinguish them. It is straightforward to prove that this definition leads to an accessibility relation which is reflexive, symmetric,

and transitive, i.e., it is an equivalence relation. This also means that the fragment of our logic that deals with knowledge is what has been classically known as $S5$ [20].

Equipped with this compositional framework that supports reasoning about knowledge and probability, and with a methodology to model autonomous systems, we can now present an example to show how it can be applied to the requirement analysis and generation phase of a design process.

Example 1 Consider an unmanned aerial platform $u1$ for urban air mobility arriving in proximity of a vertiport $port$ and having to perform a safe landing as shown in Figure 4. The airspace is under the supervision and control of a local operator $ctrl$. The environment env includes different weather conditions. The vehicle is equipped with an on-board localization system that relies on the Global Navigation Satellite System (GNSS). In addition, a ground system also estimates the position of the vehicle which is communicated to $u1$ through a communication link (LNK).

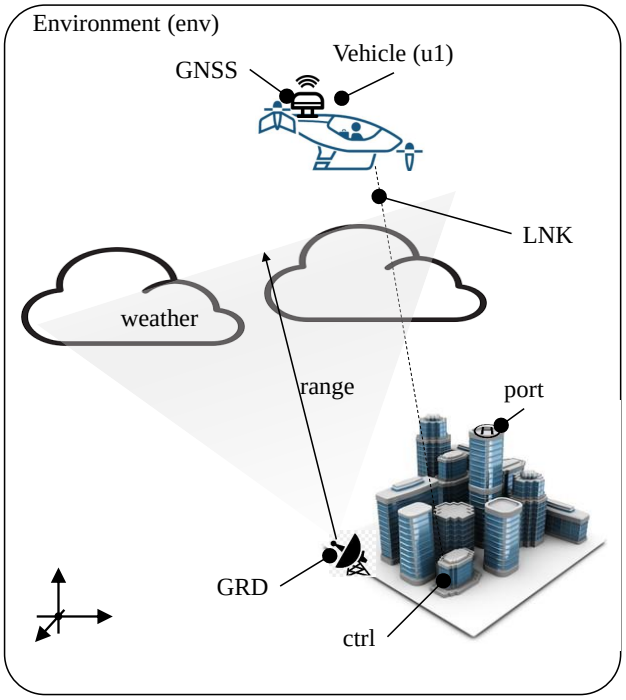

Fig. 4 A scenario representing a vehicle tasked with landing at a local vertiport. The landing task is supported by situational awareness capabilities including a GNSS system on board, a ground support system (GRD), and a communication link between the vehicle and the local controller.

In this example, we use subscripts $u1$, e, and $ctrl$ to denote the UAV, environment, and local operator agents, respectively. Symbols with no subscript have the same interpretation across agents. We also use underlined symbols for constants. Let us focus on a single top-level requirement modeled by contract C_{spec}. In order to model

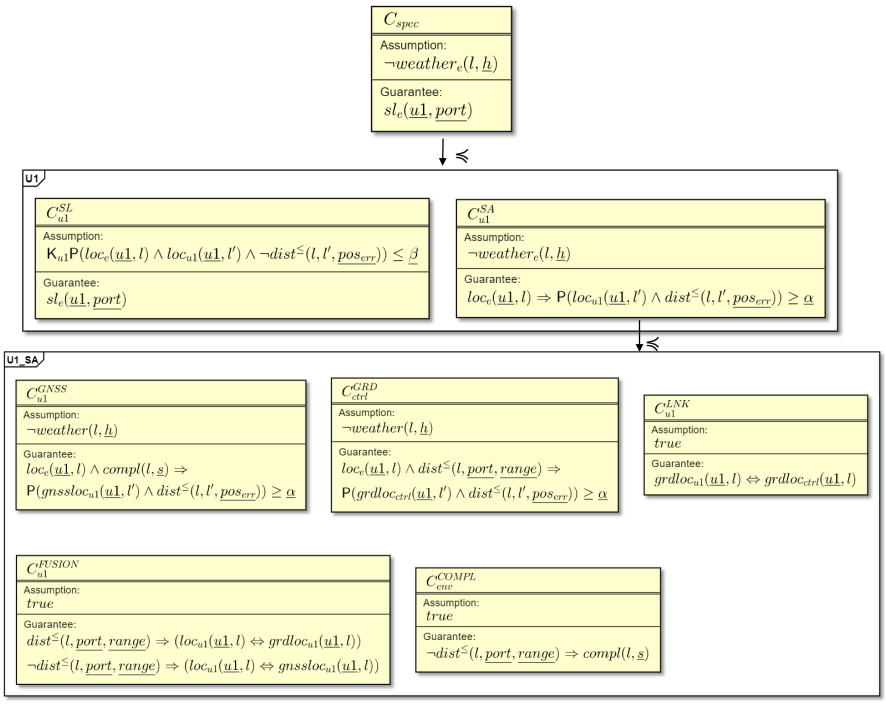

Fig. 5 A compositional model of the scenario in Figure 4. The top-level specification for safe landing in non-harsh weather conditions is refined into a situational awareness contract and a safe landing contract for the vehicle. The situational awareness contract is further refinement into the composition of contracts for the environment, the ground support, the on-board localization system, and the communication link.

such requirement, we introduce a few symbols to reason about weather conditions, and to capture the goal of a safe landing. Let $weather_e$ be a binary predicate such that $weather_e(l, c)$ is true if the weather condition at location l is c. We also introduce a few constants as follows: $\underline{u1}$ denotes the vehicle, $\underline{port}$ denotes the vertiport, and $\underline{s}$, $\underline{m}$, and $\underline{h}$ are three constants used as a discrete representation of environment difficulty (safe, moderate, or harsh, respectively). These levels will be used both for weather conditions and environment level of clutter, modeled by a binary predicate $compl_e$, such that $compl_e(l, c)$ is true if the environment complexity at location l is c. In addition, we assume that both $weather_e$, and $compl_e$ are functional predicates, meaning that for each location l, there exists one and only one difficulty level c such that $weather_e(l, c)$ is true, and one and only one difficult level c' such that $compl_e(l, c')$ is true. We can now state contract C_{spec} as shown in Figure 5: when the weather conditions are not harsh, i.e. $\neg weather_e(l, \underline{h})$, vehicle $u1$ must guarantee a safe landing at the vertiport, i.e. $sl_e(u1, \underline{port})$. Notice that these symbols all belong to the environment as the contract must be satisfied in the real world.

The top-level requirement must be satisfied by vehicle $u1$. According to the methodology described in Section 5, we decompose the specification of the vehicle into two contracts: a safe landing capability C_{u1}^{SL}, and a situational awareness contract C_{u1}^{SA}. The safe landing capability captures the ability to generate plans, to execute them, and to change them as needed during execution. To limit the complexity of this example, we only discuss assurance on the localization of the vehicle. To execute a flight plan correctly, the safe landing capability needs to be confident that its internal location estimate is good enough under all possible conditions. To model this requirement we introduce a functional binary predicate loc_a for each agent a such that $loc_a(o, l)$ is true if object o is in location l according to agent a. We also introduce a ternary predicate $dist^{\leq}$ such that $dist^{\leq}(l, l', d)$ is true if the distance between locations l and l' is less than or equal to d (we are omitting the full axiomatization of such predicate as it is not needed for the purpose of this example). The assumption can then be modeled as $\mathsf{K}_{u1}\mathsf{P}(loc_e(\underline{u1}, l) \wedge loc_{u1}(\underline{u1}, l') \wedge \neg dist^{\leq}(l, l', \underline{pos_{err}})) \leq \underline{\beta}$: the vehicle knows that the probability that the distance between the true location and the estimate is greater than an acceptable position error $\underline{pos_{err}}$ is less than a constant $\underline{\beta}$. Under this assumption, the capability guarantees a safe landing at the vertiport.

The situational awareness contracts C_{u1}^{SA} captures a requirement to maintain good localization with high probability. The assumption is that the weather condition at the generic location l is not harsh. The guarantee is that if the vehicle is at location l in the environment, i.e. $loc_e(u1, l)$ is true, then the probability that the vehicle is in a different location l' according to $u1$, and that l' differs from l more than an acceptable position error $\underline{pos_{err}}$, is less than a constant $\underline{\alpha}$. As described in Section 5, this contract captures constraints on the relation between the external (true) world and the internal estimate of $u1$.

Proposition 2 *If* $1 - \underline{\alpha} \leq \underline{\beta}$*, the set* $\{C_{u1}^{SL}, C_{u1}^{SA}\}$ *refines* C_{spec}.

Proof It is sufficient to prove the validity of the following formulas:

$$A_{spec} \wedge (A_{u1}^{SL} \Rightarrow G_{u1}^{SL}) \wedge (A_{u1}^{SA} \Rightarrow G_{u1}^{SA}) \Rightarrow G_{spec}$$
$$A_{spec} \wedge (A_{u1}^{SL} \Rightarrow G_{u1}^{SL}) \Rightarrow A_{u1}^{SA}$$
$$A_{spec} \wedge (A_{u1}^{SA} \Rightarrow G_{u1}^{SA}) \Rightarrow A_{u1}^{SL}$$

The second condition is trivially true becasue $A_{spec} \Leftrightarrow A_{u1}^{SA}$. Becasue $G_{u1}^{SL} \Leftrightarrow G_{spec}$, to show the validity of the first formula it is sufficient to show that $G_{u1}^{SA} \Rightarrow A_{u1}^{SL}$. We first show the following:

$$\Big(loc_e(\underline{u1}, l) \Rightarrow \mathsf{P}(loc_{u1}(\underline{u1}, l') \wedge dist^{\leq}(l, l', \underline{pos_{err}})) \geq \underline{\alpha}\Big) \Rightarrow$$
$$\mathsf{P}(loc_e(\underline{u1}, l) \wedge loc_{u1}(\underline{u1}, l') \wedge \neg dist_{leq}(l, l', \underline{pos_{err}})) \leq 1 - \alpha$$

To show that this is valid, we use the following axiom [19]: $\mathsf{P}(\psi \wedge \neg\psi') + \mathsf{P}(\psi \wedge \psi') = \mathsf{P}(\psi)$. Thus, $\mathsf{P}(loc_{u1}(\underline{u1}, l') \wedge \neg dist^{\leq}(l, l', \underline{pos_{err}})) = \mathsf{P}(loc_{u1}(u1, l')) - \mathsf{P}(loc_{u1}(\underline{u1}, l') \wedge dist^{\leq}(l, l', \underline{pos_{err}})) \leq 1 - \alpha$. It follows that:

$$loc_e(\underline{u1}, l) \Rightarrow \mathsf{P}(loc_{u1}(\underline{u1}, l') \wedge \neg dist^{\leq}(l, l', \underline{pos_{err}})) \leq 1 - \alpha$$

Finally, it is easy to show that $\psi \Rightarrow \mathsf{P}(\psi') \leq \alpha \models \mathsf{P}(\psi \wedge \psi') \leq \alpha$, from which we deduce $\mathsf{P}(loc_e(\underline{u1}, l) \wedge loc_{u1}(\underline{u1}, l') \wedge \neg dist^{\leq}(l, l', \underline{pos_{err}})) \leq 1 - \alpha$. From $1 - \underline{\alpha} \leq \underline{\beta}$, and the knowledge generalization axiom $\psi \Rightarrow \mathsf{K}_i\psi$ [19] we deduce that $\mathsf{K}_{u1}(\mathsf{P}(loc_e(\underline{u1}, l) \wedge loc_{u1}(\underline{u1}, l') \wedge \neg dist^{\leq}(l, l', \underline{pos_{err}})) \leq \beta)$. □

Remark 1 It may seem that the knowledge modality is not really important in the assumption of C_{u1}^{SL}. Notice, however, that the situational awareness contract may induce potentially several possible worlds with different probability spaces relating the external and internal world model. For example, consider a contract C_{u1}^{SAf} that allows for potential failures that degrade the performance of situational awareness. The guarantee of this new contract is $loc_e(\underline{u1}, l) \Rightarrow (failure_e \Rightarrow \mathsf{P}(loc_{u1}(\underline{u1}, l') \wedge dist^{\leq}(l, l', \underline{pos_{err}})) \geq \underline{\gamma}) \wedge (\neg failure_e \Rightarrow \mathsf{P}(loc_{u1}(\underline{u1}, l') \wedge dist^{\leq}(l, l', \underline{pos_{err}})) \geq \underline{\gamma'})$. There are now two possible worlds, one where $failure_e$ is true and one where $\overline{failure_e}$ is false, where the probability associated with a localization error above $\underline{pos_{err}}$ is different. Depending on the value of γ and γ', the safe landing capability may or may not know whether localization is accurate enough.

The next refinement step shows the power of a compositional method. We are going to refine contract C_{u1}^{SA} and check that the refinement is valid, but we would not need to check that C_{spec} is still satisfied. Moreover, we will show how requirements are generated and allocated to different systems in this scenario. To refine C_{u1}^{SA}, we equip $u1$ with a GNSS system modeled by contract C_{u1}^{GNSS}. This system only works in non-harsh weather. It guarantees to provide a position estimate $gnssloc_{u1}$ close to the true position with probability greater than $\underline{\alpha}$, but only in simple environments. Notice that this contract alone, obviously does not refine C_{u1}^{SA}. The GNSS system is complemented by a ground system under the control of the local operator and modeled by contract C_{ctrl}^{GRD}. This system works well even in cluttered environments, but has a limited range equal to $\underline{range}$. It provides a sufficiently accurate position estimate $grdloc_{ctrl}$. To simplify the example, we assume a perfect communication link, and also a perfect fusion algorithm. The communication link contract C_{u1}^{LNK} guarantees that the interpretation of $grdloc_{ctrl}$ (i.e., the location estimate on ground) is the same as the interpretation of $grdloc_{u1}$ (i.e., the received estimate through the communication link) under all environments. The fusion contract C_{u1}^{FUSION} guarantees that the on-board system selects the source of the location estimate depending on the location of the vehicle.

Following the same approach used in the proof of Property 2, it is easy to show that the contract set $\{C_{u1}^{GNSS}, C_{ctrl}^{GRD}, C_{u1}^{LNK}, C_{u1}^{FUSION}\}$ does not refine C_{u1}^{SA}. In particular, consider a location l such that $\neg dist^{\leq}(l, \underline{port}, \underline{range})$, and $compl(l, h)$. In this case, the assumptions of C_{u1}^{GNSS} and C_{ctrl}^{GRD} are not satisfied and therefore their guarantees don't need to hold. Thus, there would be no way for the guarantee of the contract set to entail the guarantee of C_{u1}^{SA}. To address this problem it is sufficient to add a requirement on the environment modeled by contract C_{env}^{COMPL}. This contract guarantees that the environment at a distance greater than $\underline{range}$ from the $\underline{port}$ is not too complex. It can now be shown that the set

$\{C_{u1}^{GNSS}, C_{ctrl}^{GRD}, C_{u1}^{LNK}, C_{u1}^{FUSION}, C_{env}^{COMPL}\}$ indeed refines C_{u1}^{SA}. To check that this is the case, it is sufficient to see that each component sees its assumption satisfied by the assumptions of C_{u1}^{SA}. Furthermore, the guarantees of C_{u1}^{SA} are either satisfied by the GNSS system at distances beyond $\underline{range}$ (given that the environment guarantees $compl(l, \underline{s})$ there), or by the ground system otherwise.

Remarks on analysis tools

To make the methodology presented in this section effective, several tools must also be integrated in the design process. Two key analysis tools are needed: refinement checking, and component verification. Refinement checking consists in verifying that a set of contracts $\{C_1, \ldots, C_n\}$ refines a specification contract C. As mentioned in this section, refinement checking can be reduced to validity checking, or satisfiability checking, in the specification language for knowledge and probability. Several axiomatizations are available for similar languages (see for example [19, 6]) which could be used to develop satisfiability solvers. However, there seems to be a lack of usable implementations.

The component verification problem consists in showing that a component implementation I correctly implements a given contract $C = (A, G)$. The verification methods may depend on the form used to express the implementation. When $I \in \mathcal{S}_{KP}$, then the problem can be reduced again to validity checking. The formula to be shown valid is $I \wedge A \models G$. However, in many cases, the implementation I is directly given as code or a simulation model. Uncertainty quantification tools [46, 49] can be used in these cases. In the case of autonomous systems, specialized advanced tools are also available to find environments in which a specification is not met [15, 23].

6 Concluding remarks

The set of models, methods, and tools for dealing with uncertainty in cyber-physical systems have advanced to a point where engineering can confidently deliver high-assurance systems. Cyber-physical systems have evolved into complex distributed, multi-agent systems that expose new challenges in analysis and design. Traditional techniques for dealing with uncertainty need to be complemented by other tools that can generate tight requirements under aleatoric and epistemic uncertainty. Reasoning compositionally about all these types of uncertainty is necessary to scale verification to large systems, to allow for integration of capabilities from multiple parties, and to enable incremental deployment of autonomous functions with minimal effort.

We presented a methodology and a compositional modeling framework that addresses these new requirements, and we have shown their application to an unmanned aerial system. While this is a promising step, further research is needed in the area of automated reasoning for analysis and verification.

References

1. NVIDIA DRIVE AGX Developer Kit. URL https://developer.nvidia.com/drive/drive-agx
2. Statistical summary of commercial jet airplane accidents: Worldwide operations 1959–2019. Aviation Safety, Boeing Commercial Airplanes, Seattle, WA (2019). URL https://www.boeing.com/resources/boeingdotcom/company/about_bca/pdf/statsum.pdf
3. ARP4754A, S.: Guidelines for development of civil aircraft and systems. SAE International (2010)
4. ARP4761, S.: Guidelines and methods for conducting the safety assessment process on civil airborne systems and equipment sae international 12 (1996)
5. Bauer, S.S., David, A., Hennicker, R., Larsen, K.G., Legay, A., Nyman, U., Wąsowski, A.: Moving from specifications to contracts in component-based design. In: International Conference on Fundamental Approaches to Software Engineering, pp. 43–58. Springer (2012)
6. Belardinelli, F., Lomuscio, A.: Interactions between knowledge and time in a first-order logic for multi-agent systems: completeness results. Journal of Artificial Intelligence Research **45**, 1–45 (2012)
7. Bengio, Y., Goodfellow, I., Courville, A.: Deep learning, vol. 1
8. Benveniste, A., Caillaud, B., Nickovic, D., Passerone, R., Raclet, J.B., Reinkemeier, P., Sangiovanni-Vincentelli, A., Damm, W., Henzinger, T.A., Larsen, K.G., others: Contracts for System Design. Foundations and Trends® in Electronic Design Automation **12**(2-3), 124–400 (2018). Publisher: Now Publishers, Inc.
9. Bertrane, J., Cousot, P., Cousot, R., Feret, J., Mauborgne, L., Miné, A., Rival, X.: Static Analysis and Verification of Aerospace Software by Abstract Interpretation. Foundations and Trends in Programming Languages **2**(2-3), 171–291 (2015)
10. Bianchi, G.: Performance analysis of the IEEE 802.11 distributed coordination function. IEEE Journal on selected areas in communications **18**(3), 535–547 (2000). Publisher: IEEE
11. Chiles, P.: Etops redefined. AeroSafety World **2**(3), 88–92 (2007)
12. Dahlqvist, F., Patel, M., Rajko, A., Shulman, J.: Growing opportunities in the internet of things. McKinsey, July (2019)
13. Defense Advanced Research Projects Agency (DARPA): DARPA Tiles Together a Vision of Mosaic Warfare URL https://www.darpa.mil/work-with-us/darpa-tiles-together-a-vision-of-mosiac-warfare
14. Dempster, A., others: Upper and Lower Probabilities Induced by a Multivalued Mapping. Annals of Mathematical Statistics **38**(2), 325–339 (1967). Publisher: Institute of Mathematical Statistics
15. Dreossi, T., Donzé, A., Seshia, S.A.: Compositional falsification of cyber-physical systems with machine learning components. Journal of Automated Reasoning **63**(4), 1031–1053 (2019). Publisher: Springer
16. Endsley, M.R.: Toward a theory of situation awareness in dynamic systems. Human factors **37**(1), 32–64 (1995)
17. Enright, J.J., Wurman, P.R.: Optimization and coordinated autonomy in mobile fulfillment systems. In: Workshops at the twenty-fifth AAAI conference on artificial intelligence (2011)
18. Eykholt, K., Evtimov, I., Fernandes, E., Li, B., Rahmati, A., Xiao, C., Prakash, A., Kohno, T., Song, D.: Robust physical-world attacks on deep learning visual classification. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 1625–1634 (2018)
19. Fagin, R., Halpern, J.Y.: Reasoning about knowledge and probability. Journal of the ACM (JACM) **41**(2), 340–367 (1994)
20. Fagin, R., Moses, Y., Halpern, J.Y., Vardi, M.Y.: Reasoning about knowledge. MIT press (2003)
21. Fitting, M.: First-order logic and automated theorem proving. Springer Science & Business Media (2012)
22. Fitting, M., Mendelsohn, R.L.: First-order modal logic, vol. 277. Springer Science & Business Media (2012)

23. Fremont, D.J., Chiu, J., Margineantu, D.D., Osipychev, D., Seshia, S.A.: Formal analysis and redesign of a neural network-based aircraft taxiing system with VerifAI. In: International Conference on Computer Aided Verification, pp. 122–134. Springer (2020)
24. Hastings, D., McManus, H.: A framework for understanding uncertainty and its mitigation and exploitation in complex systems. In: 2004 Engineering Systems Symposium, pp. 29–31 (2004)
25. Hayhurst, K., Veerhusen, D.S., Chilenski, J.J., Rierson, L.K.: A Practical Tutorial on Modified Condition/Decision Coverage. NASA Report, NASA/TM-2001-210876 (2001)
26. Hüllermeier, E., Waegeman, W.: Aleatoric and epistemic uncertainty in machine learning: An introduction to concepts and methods. Machine Learning **110**(3), 457–506 (2021). Publisher: Springer
27. Jhala, R., Majumdar, R.: Software model checking. ACM Computing Surveys (CSUR) **41**(4), 1–54 (2009). Publisher: ACM New York, NY, USA
28. Jim Garamone: Joint All-Domain Command, Control Framework Belongs to Warfighters URL `https://www.defense.gov/Explore/News/Article/Article/2427998/joint-all-domain-command-control-framework-belongs-to-warfighters/`
29. Kapur, K.C., Pecht, M.: Reliability engineering, vol. 86. John Wiley & Sons (2014)
30. Karpathy, A.: Ai for full-self driving at tesla. `https://youtu.be/hx7BXih7zx8?t=513` (2020)
31. Klein, G., Elphinstone, K., Heiser, G., Andronick, J., Cock, D., Derrin, P., Elkaduwe, D., Engelhardt, K., Kolanski, R., Norrish, M., others: seL4: Formal verification of an OS kernel. In: Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles, pp. 207–220 (2009)
32. Lavaei, A., Soudjani, S., Abate, A., Zamani, M.: Automated verification and synthesis of stochastic hybrid systems: A survey. arXiv preprint arXiv:2101.07491 (2021)
33. Le, T.T.H., Passerone, R., Fahrenberg, U., Legay, A.: Contract-based requirement modularization via synthesis of correct decompositions. ACM Transactions on Embedded Computing Systems (TECS) **15**(2), 1–26 (2016). Publisher: ACM New York, NY, USA
34. Le Boudec, J.Y., Thiran, P.: Network calculus: a theory of deterministic queuing systems for the internet, vol. 2050. Springer Science & Business Media (2001)
35. Leonardi, F., Pinto, A., Carloni, L.P.: A case study in distributed deployment of embedded software for camera networks. In: 2009 Design, Automation & Test in Europe Conference & Exhibition, pp. 1006–1011. IEEE (2009)
36. Leonardi, F., Pinto, A., Carloni, L.P.: Synthesis of distributed execution platforms for cyber-physical systems with applications to high-performance buildings. In: 2011 IEEE/ACM Second International Conference on Cyber-Physical Systems, pp. 215–224. IEEE (2011)
37. Maurizio Anichini: Solutions to the High Cost of Aircraft Ground Damage: While Paper (2017)
38. Mauw, S., Oostdijk, M.: Foundations of attack trees. In: International Conference on Information Security and Cryptology, pp. 186–198. Springer (2005)
39. Pasqualetti, F., Dörfler, F., Bullo, F.: Attack detection and identification in cyber-physical systems. IEEE transactions on automatic control **58**(11), 2715–2729 (2013). Publisher: IEEE
40. Pinto, A.: An open and modular architecture for autonomous and intelligent systems. In: 2019 IEEE International Conference on Embedded Software and Systems (ICESS), pp. 1–8. IEEE (2019)
41. Pinto, A., Carloni, L.P., Sangiovanni-Vincentelli, A.L.: A communication synthesis infrastructure for heterogeneous networked control systems and its application to building automation and control. In: Proceedings of the 7th ACM & IEEE international conference on Embedded software, pp. 21–29 (2007)
42. Ramirez, A.J., Jensen, A.C., Cheng, B.H.: A taxonomy of uncertainty for dynamically adaptive systems. In: 2012 7th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS), pp. 99–108. IEEE (2012)
43. Robeyns, I., Byskov, M.F.: The Capability Approach. `https://plato.stanford.edu/archives/win2016/entries/capability-approach/` (2016)
44. Rostami, M., Koushanfar, F., Karri, R.: A primer on hardware security: Models, methods, and metrics. Proceedings of the IEEE **102**(8), 1283–1295 (2014). Publisher: IEEE

45. Shafer, G.: A mathematical theory of evidence, vol. 42. Princeton university press (1976)
46. Smith, R.C.: Uncertainty quantification: theory, implementation, and applications, vol. 12. Siam (2013)
47. Stamatelatos, M., Vesely, W., Dugan, J., Fragola, J., Minarick, J., Railsback, J.: Fault tree handbook with aerospace applications (2002)
48. Stamatis, D.H.: Failure mode and effect analysis: FMEA from theory to execution. Quality Press (2003)
49. Swiler, L.P., Paez, T.L., Mayes, R.L.: Epistemic uncertainty quantification tutorial. In: Proceedings of the 27th International Modal Analysis Conference (2009)
50. Wilhelm, R., Engblom, J., Ermedahl, A., Holsti, N., Thesing, S., Whalley, D., Bernat, G., Ferdinand, C., Heckmann, R., Mitra, T., others: The worst-case execution-time problem—overview of methods and survey of tools. ACM Transactions on Embedded Computing Systems (TECS) **7**(3), 1–53 (2008). Publisher: ACM New York, NY, USA
51. Yeh, Y.: Safety critical avionics for the 777 primary flight controls system. In: 20th DASC. 20th Digital Avionics Systems Conference (Cat. No. 01CH37219), vol. 1, pp. 1C2–1. IEEE (2001)
52. Zadeh, L.: Fuzzy sets. Information Control **8**(3), 338–353
53. Zhou, K., Doyle, J.C.: Essentials of robust control, vol. 104. Prentice hall Upper Saddle River, NJ (1998)