# A Taxonomy of Uncertainty for Dynamically Adaptive Systems

Andres J. Ramirez, Adam C. Jensen, and Betty H. C. Cheng
*Michigan State University*
*3115 Engineering Building*
*East Lansing, MI 48824*
{*ramir105, acj, chengb*}*@cse.msu.edu*

***Abstract*—Self-reconfiguration enables a dynamically adaptive system (DAS) to satisfy requirements even as detrimental system and environmental conditions arise. A DAS, especially one intertwined with physical elements, must increasingly reason about and cope with unpredictable events in its execution environment. Unfortunately, it is often infeasible for a human to exhaustively explore, anticipate, or resolve all possible system and environmental conditions that a DAS will encounter as it executes. While uncertainty can be difficult to define, its effects can hinder the adaptation capabilities of a DAS. The concept of uncertainty has been extensively explored by other scientific disciplines, such as economics, physics, and psychology. As such, the software engineering DAS community can benefit from leveraging, reusing, and refining such knowledge for developing a DAS. By synthesizing uncertainty concepts from other disciplines, this paper revisits the concept of uncertainty from the perspective of a DAS, proposes a taxonomy of potential sources of uncertainty at the requirements, design, and execution phases, and identifies existing techniques for mitigating specific types of uncertainty. This paper also introduces a template for describing different types of uncertainty, including fields such as source, occurrence, impact, and mitigating strategies. We use this template to describe each type of uncertainty and illustrate the uncertainty source in terms of an example DAS application from the intelligent vehicle systems (IVS) domain.**

***Keywords*-Dynamically adaptive systems; uncertainty; taxonomy; requirements engineering; design; run-time**

## I. Introduction

Self-reconfiguration enables a dynamically-adaptive system (DAS) to satisfy requirements even as detrimental system and environmental conditions arise [1]. A DAS achieves this objective by measuring properties about itself and its execution environment, analyzing this monitoring information to detect conditions that can obstruct the satisfaction of requirements, and, if necessary, deciding how to safely reconfigure itself. As DASes grow in complexity, they must increasingly reason about and cope with events that can arise from unpredictable environments. While such *uncertainty* can be difficult to define, its effects can limit the adaptation capabilities of a DAS. This paper revisits the concept of uncertainty from the perspective of a DAS, introduces a taxonomy of common design- and run-time sources of uncertainty, and identifies a preliminary set of promising techniques that can mitigate different types of uncertainty.

It may be infeasible to exhaustively anticipate and resolve all possible system and environmental conditions that a DAS will encounter. The shared boundary [2] between a software system and its execution environment exposes a DAS to a myriad of possible inputs that can obstruct requirements. Recently, the software engineering research community introduced various techniques for documenting, reasoning about, and resolving the threat that uncertainty poses to the adaptation capabilities of a DAS [3]–[14]. While a step in the right direction, these techniques tend to focus on resolving disparate aspects and concerns of uncertainty. For instance, while Welsh *et al.* [13] explored uncertainty from the perspective of what assumptions are embedded upon a soft goal's contribution link, Jensen *et al.* [9], [10] explored uncertainty from the perspective of what latent or unknown properties a software system might also exhibit in addition to its known requirements. These disparate views within the software engineering DAS community highlight the lack of consensus regarding the definition and different types, sources, and effects of uncertainty.

The concept of uncertainty has been extensively explored by other scientific disciplines, such as physics, economics, and psychology. The software engineering DAS community can benefit from reusing and refining such knowledge for the purpose of developing a DAS. Establishing a common vocabulary and taxonomy of uncertainty from the perspective of a DAS should enable this community to leverage existing results from other disciplines and facilitate a more coherent dialogue among researchers. Moreover, a common vocabulary and taxonomy of uncertainty should also facilitate the analysis, comparison, and perhaps integration of different techniques and approaches for dealing with uncertainty in a DAS. Ideally, such information should be organized into a catalogue, using descriptions similar to those used for design patterns [15], to raise awareness about how uncertainty affects a DAS and guide DAS developers towards existing techniques that can handle specific sources of uncertainty.

This paper presents the concept of uncertainty from the perspective of a DAS, proposes a taxonomy of potential sources of uncertainty at the requirements, design, and execution phases, and identifies existing techniques for mitigating specific types of uncertainty. First, we explore how other science disciplines define, represent, analyze, and

978-1-4673-1787-0/12/$31.00 © 2012 IEEE

SEAMS 2012, Zürich, Switzerland

resolve sources of uncertainty. We then combine, generalize, and recast these definitions to suit the needs, constraints, and properties of DASes. Based on this definition, we then propose a taxonomy of sources of uncertainty by identifying the most common sources of uncertainty documented in published work on adaptive systems. For each source of uncertainty, we present the context in which it occurs, its consequences, and known mitigation strategies for resolving the uncertainty.

This paper also introduces a template to facilitate the organization and reuse of the proposed taxonomy of uncertainty. The proposed template, loosely based on the template used by Gamma *et al.* [15] to catalogue their collection of object-oriented design patterns, includes fields such as classification, context, consequences, mitigation strategies, and related sources of uncertainty, among others. We use this template to describe each type of uncertainty and illustrate the uncertainty source in terms of an example DAS application from the intelligent vehicle systems (IVS) domain. The remainder of this paper is organized as follows. Section II provides background information on the IVS application. Section III introduces our definition, template, and taxonomy of uncertainty from the perspective of a DAS. Section IV overviews current techniques for handling uncertainty in a DAS. Lastly, Section V summarizes and discusses our findings and presents future directions.

## II. Background

An IVS provides adaptive cruise control, lane keeping, and collision avoidance features. These tasks require an IVS to balance requirements of passenger safety and comfort, reasonable energy usage, low error rate, and safe interaction with other vehicles. Based on input from industrial collaborators, an IVS has the following invariant (**R1** – **R2**) and non-invariant (**R3** – **R7**) functional requirements:

**R1:** The IVS shall maintain the vehicle within lane boundaries.
**R2:** The IVS shall maintain a safe minimum distance between itself and other obstacles on the road.
**R3:** The IVS shall maintain a speed equal to a target speed.
**R4:** If an obstacle in its path is detected, then the IVS shall maintain a speed equal to the target vehicle's speed.
**R5:** If the obstacle moves out of range, then the IVS shall resume the previously set cruise control speed.
**R6:** The adaptive cruise control module shall disengage and alert the driver if it is incapable of satisfying **R2**.
**R7:** The IVS shall provide automated steering correction to maintain the vehicle in the center of the lane.

In addition, the IVS must also balance and satisfice the following non-functional requirements:

**NFR1:** The IVS shall minimize abrupt changes in speed and heading.
**NFR2:** The IVS shall be as energy efficient as possible.

## III. Uncertainty in the Context of a DAS

This section presents a definition and taxonomy of uncertainty from the perspective of a DAS. First, we review definitions of uncertainty proposed by other scientific disciplines, comparing them to the uncertainty that bears upon the design and operation of a DAS. We note other fields, such as robotics and databases have explored the concept of uncertainty, but for this work we are particularly interested in defining uncertainty from the perspective of non-computer science fields. Next, we introduce a template to facilitate the organization and description of how uncertainty affects a DAS. Using this template, we then introduce a taxonomy of common sources of uncertainty in a DAS organized to whether a source of uncertainty arises at the requirements, design, or run-time level.

### A. Defining Uncertainty

Uncertainty is a topic frequently studied in economics [16], usually in the context of risk, expected utility, and rational decision-making. To an economist, uncertainty typically describes a process or situation with a stochastic or probabilistic component. For example, the unpredictable nature of weather (e.g., a hurricane) can introduce uncertainty into the economy by devastating the agricultural output of a nation. Thus, within the field of economics, the concept of uncertainty is interpreted literally: we are simply *not certain* about the occurrence or outcome of an event or decision. Furthermore, uncertainty in economics may also include situations where the *probability* of an event or outcome is neither known nor computable in any objective sense [17].

In contrast, the discipline of classical physics considers uncertainty to be an artifact introduced by imperfect measurements [18]. Every measurement of a physical entity or phenomenon is inherently uncertain either by limitations in the measuring equipment or by the application of *ad hoc* experimental techniques. In general, measuring equipment suffers from limitations in precision and accuracy. A lack of numerical precision or inaccuracy in a physical measurement can lead to *ambiguity*. Even with precise and accurate measurement tools, conducting an experiment with *ad hoc* experimental techniques can also limit the reproducibility of results and thus lead to inconsistent conclusions.

While uncertainty plays a significant role in classical physics, its effects are even more pronounced in the subdiscipline of quantum physics. For instance the uncertainty principle [19] in quantum physics states the existence of a limit upon the precision with which two or more physical properties of the same subject can be simultaneously measured. This principle implies that the act of measuring somehow disturbs other properties of the subject being measured. Both classical and quantum physics often rely on *uncertainty analysis* to estimate the extent to which uncertainty in one measurement affects other measurements and decisions.

Uncertainty is also inherent in everyday psychology and human behavior, most notably in decision-making tasks. In psychology, uncertainty often refers to the ability of human beings to make judgments in the presence of incomplete information [20]. In general, *incomplete* information occurs when a dimension of data, such as the outcome of an action, is unknown at the present time. Despite a lack of sufficient and reliable information, humans are often able to make reasonable judgements based on past experience.

The preceding definitions present distinct yet related concepts of uncertainty. That is, uncertainty is inherent in stochastic processes, the precision and accuracy of measurement tools, and in the complexity of decision-making tasks. A DAS also encounters all of these situations. For instance, a DAS may attempt to satisfy infeasible requirements, interact with unpredictable and adverse environments, inadvertently disturb its environment when using unreliable sensors to measure properties about itself and its environment, and perform decision-making tasks with incomplete and inconsistent information. Based on these definitions and observations, we now propose a definition of uncertainty specific to a DAS:

**Definition of Uncertainty for a DAS.** *Uncertainty is a system state of incomplete or inconsistent knowledge such that it is not possible for a DAS to know which of two or more alternative environmental or system configurations hold at a specific point. This uncertainty can occur due to missing or ambiguous requirements, false assumptions, unpredictable entities or phenomena in the execution environment, and unresolvable conditions caused by incomplete and inconsistent information obtained by potentially imprecise, inaccurate, and unreliable sensors in its monitoring infrastructure.*

This definition of uncertainty for a DAS implicitly distinguishes between sources of uncertainty that occur either at the requirements, design, or execution phases. At each of these phases, uncertainty can be introduced into the DAS either by the DAS itself or its execution environment. Specifically, *system* and *environmental uncertainty* refer to a state of limited knowledge where two or more alternative states of the system or its execution environment, respectively, are possible. Note that sources of uncertainty can crosscut multiple phases. For example, if a source of uncertainty at the requirements level is not resolved before the design phase begins, then that source of uncertainty will necessarily propagate throughout the design of the DAS. Any source of uncertainty that is not resolved before the DAS is deployed must therefore be addressed by the DAS at run time.

### B. Template for Describing Uncertainty

In addition to revisiting the concept of uncertainty from the perspective of an adaptive system, this paper also proposes a taxonomy of potential sources of uncertainty that can affect a DAS. In order to facilitate the organization and reuse of this taxonomy, this paper introduces a template for uniformly describing different types and sources of uncertainty. Ideally, this template, and other instances of it that might follow, will facilitate a more structured and coherent dialogue between researchers in the software engineering DAS community on the challenges that uncertainty poses to adaptive systems.

Figure 1 presents the proposed template for describing uncertainty and briefly describes the intent of each field. The proposed template is loosely based on the template used by Gamma *et al.* [15] for presenting and organizing their catalogue of object-oriented design patterns. In particular, the intent of the **Name**, **Classification**, **Context**, **Sample Illustration**, **Related Sources**, and **Also Known As** fields are similar to those used in Gamma *et al.*'s template. In addition, the proposed template also includes a **Mitigation Strategy** field that lists existing techniques for either resolving or mitigating the respective source of uncertainty. Lastly, the **Impact** field specifies how a source of uncertainty affects the design or execution of a DAS.

**Name:** A unique term for identifying a type or source of uncertainty in a DAS.
**Classification:** Organizes a source of uncertainty according to the phase at which it occurs.
**Context:** Defines the source of uncertainty and presents the conditions under which it occurs.
**Impact:** Describes how a source of uncertainty affects the design or execution of a DAS.
**Mitigation Strategies:** Lists existing techniques that can resolve this source of uncertainty.
**Sample Illustration:** Presents an example of how a source of uncertainty occurs and affects a DAS.
**Related Sources:** Additional sources of uncertainty that commonly occur in conjunction.
**Also Known As:** Other terms often used to describe the same source of uncertainty.

Figure 1. Template for describing sources of uncertainty.

As shown in Figure 1, the proposed template does not include fields such as **Structure**, **Participants**, or **Collaboration**. In the template used by Gamma *et al.*, these fields collectively represent the structure and behavior for a reusable *design* that can be instantiated to solve the recurring problem identified by the design pattern. In contrast, we are using the template to describe sources and impacts of uncertainty. As applicable, the proposed template also identifies existing resolution or mitigation techniques under the **Mitigation Strategies** field.

### C. Overview of Taxonomy

Table I provides an overview of the various sources of uncertainty that can affect a DAS that have been identified thus far. This table, and the subsequent discussion of the proposed taxonomy, is organized according to whether a source

of uncertainty arises predominantly at the requirements, design, or run-time levels. This table also identifies candidate resolution techniques previously proposed by the software engineering DAS community to mitigate the corresponding source of uncertainty. Each item in the table is accompanied by a number, and this number is given along with the item when it is referenced in subsequent sections.

### *D. Taxonomy of Uncertainty at the Requirements Level*

In general, the development process for adaptive systems begins with the elicitation of functional and non-functional requirements [21]. Such requirements are often idealized, misunderstood, and incomplete [22]. Figure 2 presents a dependency graph with the various root sources of uncertainty at the requirements level: missing requirements [4], [23], ambiguous requirements [4], [13], [24], and falsifiable assumptions [13], [25]. Each directed arrow in this graph represents a dependency between the two sources of uncertainty. In particular, a missing requirement refers to an incomplete specification that does not cover all the requirements the DAS needs to satisfy. As the name implies, an ambiguous requirement is a specification or satisfaction criteria that can be interpreted in different ways. Both missing requirements and ambiguous requirements can be considered as special instances of inadequate requirements. Lastly, a falsifiable assumption (also known as a falsifiable requirement) is a statement used to support a requirement but may itself become invalid.

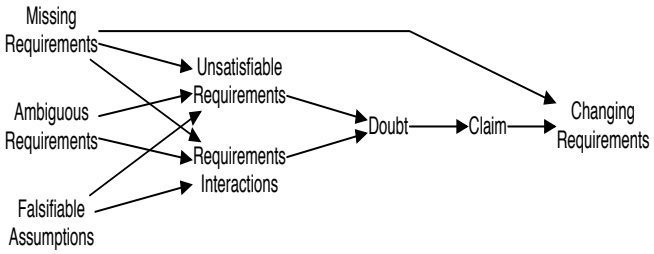


Figure 2. Taxonomy of Uncertainty at the Requirements Level.

These sources of uncertainty at the requirements level can result in unsatisfiable requirements [3], [5], [14], requirements interactions [10], [11], [26], which can then lead to doubts about the requirements and scope of the system, and, in the case of non-functional requirements, unproven contribution links [13], [25]. Naturally, such uncertainty at the requirements level causes requirements engineers and developers to express both concerns and doubts about the set of elicited requirements. Furthermore, this uncertainty can easily propagate to the design and run-time phases of a DAS if a requirements engineer does not document doubts and concerns in the set of requirements. To this end, a requirement may formally state a *claim*, or a subjective rationale, behind an uncertain decision. Though not always possible, uncertainty at the requirements level should be resolved by adding previously missing requirements, refining ambiguous requirements, or showing an assumption always holds true.

We now present one source of uncertainty at the requirements level in template form below.

---

**Name:** Missing Requirements
**Classification:** Requirements
**Context:** A missing requirement occurs when the specification of a DAS is incomplete and does not cover all functional and non-functional requirements that the system is expected to satisfy at run time. This source of uncertainty often arises during the requirements elicitation process.
**Impact:** Missing requirements can lead to unsatisfiable requirements, requirements interactions, and changing requirements. Specifically, it is unlikely that a DAS will of satisfy a requirement for which it was not designed. Similarly, a missing requirement may cause requirements engineers to interpret relationships between identified requirements in different ways, possibly introducing complex requirements interactions. Lastly, once a missing requirement is identified, the specification of the DAS must be changed to incorporate the missing requirement.
**Sample Illustration:** An important, yet missing, requirement can be identified by considering IVS requirement R6 (see Section II). In particular, requirement R6 specifies that the adaptive cruise control module should disengage and alert the driver if it is unable to maintain a minimum safe distance between the IVS and nearby vehicles. The current set of requirements, however, does not specify whether the lane keeping module should continue to operate or disengage in the same situation. As such, it is uncertain how the lane keeping module should behave in this situation as both behaviors are possible.
**Mitigation Strategies:** KAOS Obstacle and Threat Modeling [23] and Partial Goal Satisfaction Framework [27], Loki [11], Marple [9], [10], Requirements Reflection [4].
**Related Sources:** Unsatisfiable Requirements (4), Requirements Interactions (5), Changing Requirements (8).
**Also Known As:** Incomplete requirements [26].

---

### *E. Taxonomy of Uncertainty at the Design Level*

Even under the assumptions that requirements have been completely and unambiguously identified, several sources of uncertainty can arise at the design level. Figure 3 shows a dependency graph of various sources of design-time uncertainty and how they can result in an inadequate design that does not satisfy the intended set of requirements. As this figure illustrates, the two primary sources of design-time uncertainty are *unexplored alternatives* [28] and an *untraceable design* [23]. Unexplored alternatives occur when it is infeasible for developers to consider all possible design alternatives for satisfying a given set of requirements, and an

Table I
TABLE SUMMARIZING TAXONOMY OF UNCERTAINTY IN A DAS.

| | Id | Term | Definition | Mitigation Techniques |
|---|---|---|---|---|
| Requirements | 1 | Missing Requirement | The specification is incomplete and does not cover all requirements. | Requirements Reflection [4], Marple [10], Loki [11], KAOS Obstacle Threat Modeling [22], Partial Goal Satisfaction [27] |
| | 2 | Ambiguous Requirement | The specification or evaluation criteria can be interpreted in different ways. | Requirements Reflection [4], Claims [13,25] |
| | 3 | Falsifiable Assumption | A possibly false statement used to support the validity of a requirement | Claims [13,25] |
| | 4 | Unsatisfiable Requirements | A requirement that cannot be satisfied by the DAS. | FLAGS [3], RELAX [5, 14], Awareness Requirement [35] |
| | 5 | Requirements Interactions | Two or more requirements that inadvertently interfere with each other. | Marple [10], Loki [11] |
| | 6 | Doubt | A concern about a statement or requirement. | Claims [13,25] |
| | 7 | Claim | A subjective rationale that forms the basis of a decision. | Claims [13,25] |
| | 8 | Changing Requirements | Requirements that do not reflect the needs and constraints of the system. | Requirements Reflection [4], Feedback Loop [33] |
| Design | 9 | Unexplored Alternatives | When not all different design options are considered. | Avida-MDE [8], FORMS [31], Partial Models [28], Marple [10] |
| | 10 | Untraceable Design | Irrelevant design decisions from the perspective of requirements. | |
| | 11 | Risk | An exposure to danger or loss. | |
| | 12 | Misinformed Tradeoff Analysis | Design decisions based on misguided and subjective preferences. | |
| | 13 | Inadequate Design | Requirements cannot be fully satisfied or include latent behaviors. | FLAGS [3], RELAX [5, 14], Marple [10], Loki [11] |
| | 14 | Unverified Design | Lack of proof that shows a design satisfies its requirements | Partial Models [28] |
| | 15 | Inadequate Implementation | An implementation that contains errors or faults. | Feedback Loop [33], C2 [36], Rainbow [37] |
| | 16 | Latent Behavior | An unknown behavior that should be disallowed. | Marple [10], Loki [11] |
| Run-Time | 17 | Effector | An adaptation that alters the execution environment in unanticipated ways. | |
| | 18 | Sensor Failure | When a sensor cannot measure or report the value of a property. | Feedback Loop [33], C2 [36], Rainbow [37] |
| | 19 | Sensor Noise | Random and persistent disturbances that reduce the clarity of a signal. | |
| | 20 | Imprecision | A lack of repeatability in a given measurement. | |
| | 21 | Inaccuracy | A divergence between a measured value and its real value. | |
| | 22 | Unpredictable Environment | Events and conditions in the environment that cannot be anticipated. | FLAGS [3], RELAX [5,14], Loki [11], Feedback Loop [33], C2[36], Rainbow [37], Softure [38] |
| | 23 | Ambiguity | A lack of numerical precision and accuracy in a measurement. | |
| | 24 | Non-Specificity | A property whose value is only known within a certain range of values. | |
| | 25 | Inconsistency | Two or more values of the same property that disagree with each other. | |
| | 26 | Incomplete Information | A missing or unknown dimension of data. | Requirements Reflection [4], RELAX [5,14] |

untraceable design occurs design decisions are made without addressing and documenting the specific requirements they are supposed to satisfy.

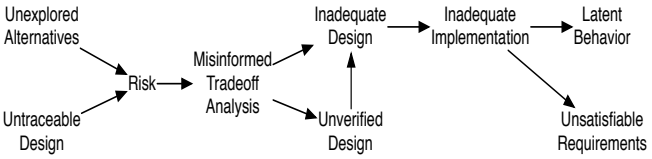


Figure 3. Taxonomy of Uncertainty at the Design Level.

Faced with design-time uncertainty, developers often accept *risk*, or an exposure to danger or loss, depending on the severity of the repercussions if a requirement is violated. Risk is inherently uncertain as it implies not knowing which of two or more possible outcomes will occur for a given event, condition, or decision. While risk crosscuts uncertainty at the requirements and run-time levels, it is particularly problematic at the design level because it can lead to a misinformed tradeoff analysis that provides the rationale for selecting, perhaps incorrectly, one design or development technique over another. An inadequate design often leads to an implementation that can contain errors, due to an unverified design, or comprise an inadequate architecture or set of enabling technologies. Combined, these sources of design-time uncertainty can prevent an implementation from satisfying its intended requirements.

We now present one source of uncertainty at the design level in template form below.

---

**Name:** Unexplored Alternatives
**Classification:** Design
**Context:** An unexplored alternative occurs when developers are either unwilling or unable (i.e., due to constraints upon development time, budget, and resources) to consider all possible designs and tradeoffs for satisfying a given set of requirements.
**Impact:** This source of uncertainty can result in a design that is sub-optimal, does not satisfy all functional and non-functional requirements, and exhibits latent behaviors.
**Sample Illustration:** We reused McUmber and Cheng's [29] adaptive cruise control design since they applied formal model-checking techniques to guarantee it satisfied requirements **R2** through **R6**. Their adaptive cruise control design was based on minimum safe distance standards set by industry. While reusing their design enabled us to reduce implementation efforts, it also implied that we did not explore alternate designs. In particular, we did not consider designs that increased the minimum safe distance between the IVS and another vehicle. Such an alternate design may improve the safety and reliability of the overall IVS by providing additional room for the IVS to maneuver.
**Mitigation Strategies:** Avida-MDE [30], Claims [13], [25], FORMS [31], Partial Models [28].

**Related Sources:** Risk (11), Misinformed Tradeoff Analysis (12), Inadequate Design (13).
**Also Known As:** To be determined.

### F. Taxonomy of Run-Time Uncertainty

Sources of *run-time* uncertainty occur *primarily* from interactions between the DAS and its unpredictable environment. In particular, the execution environment of a DAS can threaten its adaptation capabilities [3]–[5], [11], [13], [14]. This uncertainty arises specifically at the shared boundary between a software system and its environment [2]. Unfortunately, in the case of a DAS, it is often infeasible for a developer to enumerate or anticipate all possible combinations of environmental conditions that the DAS will encounter throughout its lifetime [5], [14]. As a result, the environment adds a dimension of *unpredictability* in the sense that it can introduce events or conditions that a DAS might be unable to interpret or handle because they were unforeseeable at design time.

A DAS uses sensors in its monitoring infrastructure to measure observable events in its environment. This dependency, captured in Figure 4, implies that a DAS is capable of adapting only in response to those conditions it detects and deems likely to obstruct the satisfaction of a requirement. Nevertheless, an unpredictable environment can introduce uncertain sensor inputs in the form of events and conditions that the DAS is incapable of interpreting. For example, adverse weather conditions, such as dense fog, could alter how the IVS perceives its environment and reacts to it. Since a DAS detects these unanticipated events and conditions through its monitoring infrastructure, run-time uncertainty originates not only from inherent limitations in the precision, accuracy, and reliability of its monitoring infrastructure, but also from environmental inputs it is unable to interpret and reason about.

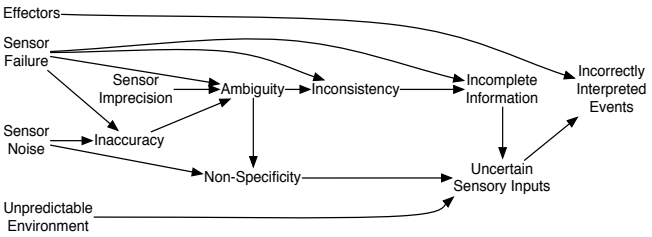


Figure 4. Taxonomy of Run-Time Uncertainty Sources.

As Figure 4 illustrates, the monitoring infrastructure introduces three root sources of run-time uncertainty: sensor imprecision, sensor noise, and sensor failures. Imprecision can adversely affect the ability of a DAS to consistently reproduce a given measurement [32]. Orthogonal to the concept of imprecision, *sensor noise* is a random and persistent disturbance that can reduce the clarity of a given signal. When dealing with large quantities of data, sensor noise can also refer to irrelevant and meaningless data. Lastly, a *sensor failure* occurs when a sensor is unable to either measure or report the value of a given property. As Figure 4 shows, each of these three sources of uncertainty can occur independently since they do not have any incident arrows.

Neither hardware nor software sensors are perfectly precise, accurate, and reliable. As Figure 4 also illustrates, sensor imprecision, inaccuracy, and failure can introduce ambiguity. As previously described, *ambiguity* occurs due to a lack of numerical precision or inaccuracy in a measurement [32]. An imprecise sensor can prevent a DAS from differentiating between two otherwise seemingly equal values. For example, if the precision of a temperature sensor is limited to one degree Celsius, then the values of 37.0°C and 37.9°C would be indistinguishable. [1] An inaccurate sensor, on the other hand, reports values that are considerably different from the real value of what is being measured. Similarly, a failed sensor can either report nonsensical values or fail to report the value of such a measurement. In all three cases, ambiguity limits the ability of a DAS to reliably detect and reconfigure from conditions that warrant adaptation.

Ambiguity can also lead to inconsistency and non-specificity. *Inconsistency* occurs when two or more measurements disagree with each other. Although not always feasible, redundancy may resolve inconsistency in certain situations. For instance, a DAS can select the statistical mode of multiple and independent measurements to determine the most likely value of a property. Without additional information, inconsistency can lead to uncertainty in the form of *incomplete information* and *non-specificity*. Non-specificity refers to the value of a property lying within a certain range of possible values [32]. Unlike inconsistency, to resolve non-specificity, a DAS must selectively discard measurements to converge upon a specific value. Combined, incomplete information and non-specificity leads to uncertain sensory inputs.

As Figure 4 also illustrates, both effectors and uncertain sensory inputs can hinder a DAS's ability to correctly interpret and analyze system and environmental conditions at run time. An effector [33], which is an actuator responsible for enacting reconfiguration instructions, can either fail during an adaptation or inadvertently introduce adverse effects upon the execution environment. In both situations, the DAS may be unable to assess the outcome of the adaptation. Similarly, an uncertain sensory input corresponds to monitoring information that a DAS is not intended to interpret or analyze, and can cause the DAS to incorrectly assess its state.

We now present one of the sources of uncertainty at the run-time level in template form below.

**Name:** Incomplete Information

[1] Though seemingly irrelevant, the difference between these values reflects a normal body temperature and a fever.

**Classification:** Run-Time
**Context:** Incomplete information refers to a missing or unknown dimension of data.
**Impact:** Without additional information, this source of uncertainty can create an unresolvable state in a DAS that hinders it either from detecting when adaptation is necessary, or selecting the most appropriate reconfiguration strategy.
**Sample Illustration:** In a related case study [11], we explored the effects of various system and environmental conditions upon the adaptation capabilities of the IVS. In that example, the monitoring infrastructure of the IVS was unreliable and subject to different types and degrees of sensor noise. Collectively, these conditions introduced uncertainty, in the forms of inconsistencies and non-specificity, into the monitoring data of the IVS such that the IVS was unable to accurately compute its velocity and distance to another vehicle. Since the IVS did not possess additional information to resolve this uncertainty, in certain scenarios, both vehicles collided.
**Mitigation Strategies:** FLAGS [3], RELAX [5], [14], Requirements Reflection [4].
**Related Sources:** Inconsistency (25), Non-Specificity (24), Inadequate Design (13).
**Also Known As:** Unmonitorable [5], [14], known unknown [4].

## IV. Related Work

This section presents related work on managing uncertainty at the requirements, design, and run-time levels. For each of these related works, we identify the source(s) of uncertainty that it can mitigate.

### A. Managing Uncertainty at the Requirements Level

Fuzzy logic has been used as the formal foundation for languages used to represent uncertainty. Whittle *et al.* [14] developed RELAX, a textual requirements language that provides fuzzy logic-based temporal, ordinal, and modal operators to facilitate the specification of sources and impacts of uncertainty in self-adaptive systems. Subsequently, Cheng *et al.* [5] introduced a goal–oriented, model-based process for identifying non-invariant requirements that should be RELAXed due to an unpredictable environment (20). In a similar approach, Baresi *et al.* [3] introduced the FLAGS language to specify, via fuzzy logic function shapes, the satisfaction criteria of adaptation–oriented goals. RELAX and FLAGS, and their corresponding model-based processes, are most suited for identifying, explicitly representing, and reasoning about sources of uncertainty and how they might affect the extent to which a DAS satisfies its requirements.

In a different approach, Welsh *et al.* [13], [25] introduced Claims (7) to explicitly represent and document the set of assumptions that enable requirements engineer to differentiate and resolve how two seemingly equal designs can affect a soft goal's contribution link. Furthermore, Claims facilitates the evaluation of falsifiable assumptions (3) in subsequent stages of the development process when new information about the system and its environment might be available. Within the proposed taxonomy of uncertainty, Claims are most suited for mitigating sources of uncertainty that occur due to falsifiable assumptions (3) and ambiguous requirements (2) by documenting the rationale for selecting a particular design over another.

Letier and Van Lamsweerde [27] developed a probabilistic framework for reasoning about and evaluating the partial satisfaction of goals and requirements. Each goal is associated with a probability that represents how likely it will be satisfied. Based on these probabilities, a requirements engineer can not only identify goals that frequently become unsatisfied, but also revise goals to handle conditions leading to these violations. Since it is usually difficult to obtain numerical estimates from which to derive probabilities at design time, Epifani *et al.* [34] developed a runtime model-based approach that uses a Bayesian estimator to update the corresponding model with more accurate run time data. These frameworks can be applied to identify falsifiable assumptions (7), unsatisfiable requirements (4), and requirements interactions (5).

The requirements that a DAS must satisfy may not be fully known until run time. Bencomo *et al.* [4] proposed the concept of requirements reflection to enable a DAS to modify and re-prioritize requirements at run time. Their work discusses challenges when dealing with uncertainty from the stochastic nature of physical environments, practical limitations of sensors, and the unpredictability of how an adaptation may affect another agent's behavior and goals. Souza and Mylopolous [35] introduced the concept of an awareness requirement to evaluate the runtime status of other requirements and adapt the system if a requirement is not satisfied. These techniques could be used to address missing requirements (1) and changing requirements (8).

### B. Managing Uncertainty at the Design Level

Certain sources of uncertainty at the requirements level are difficult to identify and resolve before the design phase begins. Weyns *et al.* [31] introduced FORMS, a formal reference model for self-adaptation that builds upon feedback loops and reflection to enable the description and evaluation of alternative design choices for a DAS. In a different approach, Chechik *et al.* [28] presented a model checking–based approach for quantifying the uncertainty and potential tradeoffs inherent in a partial model that comprises certain design decisions that are finalized and others that are changeable due to uncertainty. Both approaches can be applied to address uncertainty from unexplored alternatives (9) at the design level.

Esfahani *et al.* [7] proposed a framework for handling uncertainty in a DAS. Specifically, instead of evaluating adap-

tation decisions on a single value, their framework leverages information from a range of values, some of which may be inherently uncertain. In addition, their framework proposes incorporating uncertainty analysis into the decision-making process of a DAS in order to efficiently address sources of uncertainty at run time. Their framework can be used to mitigate incomplete information (26) and uncertain sensory inputs (18, 19) by incorporating additional, albeit possibly imperfect, information.

Goldsby and Cheng [8] developed Avida-MDE, an approach that generates software system models that represent target systems suitable for dealing with an unpredictable environment (20). Avida-MDE enables developers to identify tradeoffs between generated models and identify the most appropriate target system for a given application. In subsequent work, Goldsby and Cheng [9] introduced Marple, a tool for identifying latent properties for a given behavioral system model. Jensen *et al.* [10] built a toolchain around Marple to identify common model defects, such as syntax errors and unreachable states, and detect structural and behavioral latent properties in embedded system models. These evolutionary computation-based techniques can be applied to address requirements interactions (5) and latent behaviors (16).

In a complementary approach, Ramirez *et al.* [11] developed Loki, an approach for automatically identifying interesting combinations of system and environmental conditions conducive to requirements violations and latent behaviors in a DAS. To achieve this objective, Loki introduces different types and degrees of sensor-based uncertainty and then evaluates how it affects the behavior of a DAS. Loki can be applied to discover falsifiable assumptions (3), unsatisfiable requirements (4), and requirements interactions (5).

### C. Managing Uncertainty at Run Time

Several dimensions of uncertainty at the requirements and design levels can be addressed via run-time adaptation techniques. In particular, Brun *et al.* [33] proposed engineering adaptive systems with feedback loops to deal with changing environments and emerging requirements by incorporating new environmental information into the decision-making process of a DAS that might not have been available at design time. Feedback loops are inherent in the C2 [36] and Rainbow [37] architecture-based adaptation frameworks that enable a DAS to self-reconfigure in response to changing system and environmental conditions. These techniques can specifically address several sources of design-time uncertainty at run time, such as revising a misinformed tradeoff analysis (12) by modifying the architecture of the application to better address non-functional concerns, and correcting an inadequate implementation (15) (e.g., a software bug) by unloading a faulty component and loading a verified one.

In a different approach, Bertolino *et al.* [38] proposed Softure, an approach to assist in the development of generic applications that can be adapted while explicitly addressing dynamic contexts. These frameworks and approaches are primarily concerned with performance-based adaptations that ensure non-functional requirements, such as quality of service constraints, are satisfied. These approaches are most suited for mitigating the effects of changing requirements (8) and unpredictable environment (22).

## V. Conclusions

In this paper we presented a definition and taxonomy of uncertainty within the context of a DAS. The proposed taxonomy describes common sources of uncertainty and their effects upon the requirements, design, and run-time phases of a DAS. For each identified source of uncertainty, we also listed existing techniques for resolving or mitigating that respective type of uncertainty. We illustrated the proposed taxonomy by presenting several examples of how uncertainty affects an intelligent vehicle system. Lastly, to facilitate the organization and reuse of the proposed taxonomy, we also introduced a template for describing sources of uncertainty.

Perhaps the most *manageable* sources of uncertainty originate at the requirements level. Nevertheless, these sources of uncertainty often go unresolved during the design phase and therefore remain present even as the DAS executes. These sources of uncertainty at the requirements level can lead to changing requirements that are inherently difficult to assess, evaluate, and integrate within the existing design and implementation of a DAS. Existing techniques [3], [5], [7], [13], [14], [25], [28] can be applied to not only formally document doubts, concerns, and claims about requirements, but to also reason about the possible impact they may have upon the DAS. Raising the awareness of these sources of uncertainty during requirements engineering may enable developers to address them in subsequent development phases when additional information about the system design and its environment becomes available.

Even if complete and unambiguous requirements supported by valid assumptions are available, and the design of the system guarantees to satisfy such requirements, a DAS may still have to interact with unpredictable environments through a monitoring infrastructure that, due to physical and practical limitations, cannot be perfectly precise, accurate, and reliable. Furthermore, the act of adapting itself can introduce uncertainty into the execution environment via the DAS's effectors. As a result, for many application domains, it may be *impossible* to fully resolve all sources of uncertainty at the requirements, design, and execution phases of a DAS. To some degree, other scientific disciplines, such as physics and economics, have embraced this fact and developed both theoretical and practical frameworks for *managing* uncertainty. Recent vision papers have begun to address and incorporate elements of uncertainty analysis at the design [7], [13], [28], [39] and run-time [4], [7], [25] levels. As Table I illustrates with the sparsely populated and

empty cells, techniques are still needed to handle sources of uncertainty at the design- and run-time levels.

Based on the proposed taxonomy, we also observe that existing techniques for mitigating uncertainty in a DAS appear to be complementary. As a result, future directions include integrating these techniques for resolving combinations of uncertainty, as well as further investigations into avoidance and/or mitigation strategies for those types of uncertainty that have not been as extensively studied. Furthermore, this paper is intended only to be a first step towards cataloguing and synthesizing different approaches that deal with uncertainty in a DAS. The interested reader can find an entirely separate treatment on how models at run time can be applied to address uncertainty in an upcoming Dagstuhl Seminar roadmap paper [40]. Lastly, our template and taxonomy can be expanded to include newly identified sources of uncertainty and developed resolution techniques, or specialized to better capture the needs of domain-specific uncertainty.

## ACKNOWLEDGMENTS

This work has been supported in part by NSF grants CCF-0541131, IIP-0700329, CCF-0750787, CCF-0820220, DBI-0939454, CNS-0854931, Army Research Office grant W911NF-08-1-0495, Ford Motor Company. 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 the National Science Foundation, Army, Ford, or other research sponsors.

## REFERENCES

[1] P. K. McKinley, S. M. Sadjadi, E. P. Kasten, and Betty H.C. Cheng, "Composing adaptive software," *Computer*, vol. 37, no. 7, pp. 56–64, 2004.

[2] M. Jackson and P. Zave, "Deriving specifications from requirements: an example," in *Proceedings of the 17th International Conference on Software Engineering*. Seattle, Washington, USA: ACM, April 1995, pp. 15–24.

[3] L. Baresi, L. Pasquale, and P. Spoletini, "Fuzzy goals for requirements-driven adaptation," in *Proceedings of the 18th IEEE International Requirements Engineering Conference*. Sydney, Australia: IEEE, October 2010, pp. 125–134.

[4] N. Bencomo, J. Whittle, P. Sawyer, A. Finkelstein, and E. Letier, "Requirements reflection: Requirements as runtime entities," in *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering*. Cape Town, South Africa: ACM, May 2010, pp. 199–202.

[5] B. H. Cheng, P. Sawyer, N. Bencomo, and J. Whittle, "A goal-based modeling approach to develop requirements of an adaptive system with environmental uncertainty," in *ACM/IEEE International Conference on Model Driven Engineering Languages and Systems (MODELS'09)*, ser. Lecture Notes in Computer Science. Denver, Colorado, USA: Springer-Verlag, October 2009, pp. 468–483.

[6] N. Esfahani, E. Kouroshfar, and S. Malek, "Taming uncertainty in self-adaptive software," in *Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering*, Szeged, Hungary, September 2011, pp. 234–244.

[7] N. Esfahani, "A framework for managing uncertainty in self-adaptive software systems," in *Proceedings of the 26th IEEE/ACM International Conference on Automated Software Engineering*, Lawrence, Kansas, USA, November 2011.

[8] H. J. Goldsby and Betty H.C. Cheng, "Automatically generating behavioral models of adaptive systems to address uncertainty," in *Proceedings of the 11th International Conference on Model Driven Engineering Languages and Systems*. Berlin, Heidelberg: Springer-Verlag, 2008, pp. 568–583.

[9] ——, "Automatically discovering properties that specify the latent behavior of uml models," in *Proceedings of the 13th International Conference on Model Driven Engineering Languages and Systems*. Oslo, Norway: Springer-Verlag, October 2010, pp. 316–330.

[10] A. C. Jensen, B. H. Cheng, H. J. Goldsby, and E. C. Nelson, "A toolchain for the detection of structural and behavioral latent system properties," in *Proceedings of the IEEE International Conference on Model Driven Engineering Languages and Systems (MoDELS '11)*, Wellington, New Zealand, October 2011, pp. 683–698.

[11] A. J. Ramirez, A. C. Jensen, B. H. Cheng, and D. B. Knoester, "Automatically exploring how uncertainty impacts behavior of dynamically adaptive systems," in *To Appear In the Proceedings of the 2011 International Conference on Automatic Software Engineering*, ser. ASE'11, Lawrence, Kansas, USA, November 2011.

[12] P. Sawyer, N. Bencomo, E. Letier, and A. Finkelstein, "Requirements-aware systems: A research agenda for re self-adaptive systems," in *Proceedings of the 18th IEEE International Requirements Engineering Conference*, Sydney, Australia, September 2010, pp. 95–103.

[13] K. Welsh and P. Sawyer, "Understanding the scope of uncertainty in dynamically adaptive systems," in *Proceedings of the Sixteenth International Working Conference on Requirements Engineering: Foundation for Software Quality*, vol. 6182. Essen, Germany: Springer, June 2010, pp. 2–16.

[14] J. Whittle, P. Sawyer, N. Bencomo, Betty H.C. Cheng, and J.-M. Bruel, "RELAX: Incorporating uncertainty into the specification of self-adaptive systems," in *Proceedings of the 17th International Requirements Engineering Conference (RE '09)*. Atlanta, Georgia, USA: IEEE Computer Society, September 2009, pp. 79–88.

[15] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional, 195.

[16] J. J. Laffont, *The Economics of Uncertainty and Information*. The MIT Press, 1989.

[17] I. Gilboa, A. W. Postlewaite, and D. Schmeidler, "Probability and unertainty in economic modeling," *Journal of Economic Perspectives*, vol. 22, no. 3, pp. 173–188, 2008.

[18] B. N. Taylor and C. E. Kuyatt, *Guidelines for Evaluating and Expressing the Uncertainty of NIST Measurement Results*. Gaithersburg, MD, USA: National Institute of Standards and Technology, 1994.

[19] W. Heisenberg, "Uber den anschaulichen inhalt der quantentheoretischen kinematik und mechanik," *Zeitschrift fur Physik A Hadrons and Nuclei*, vol. 43, no. 3, pp. 172–198, 1927.

[20] A. Tversky and D. Kahneman, "Judgement under uncertainty: Heuristics and biases," *Science*, vol. 185, no. 4157, p. 1124, 1974.

[21] J. Zhang and Betty H.C. Cheng, "Model-based development of dynamically adaptive software," in *Proceedings of the 28th International Conference on Software Engineering*. New York, NY, USA: ACM, 2006, pp. 371–380.

[22] A. van Lamsweerde and E. Letier, "Handling obstacles in goal-oriented requirements engineering," *IEEE Transactions on Software Engineering*, vol. 26, no. 10, pp. 978–1005, October 2000.

[23] A. van Lamsweerde, *Requirements Engineering: From System Goals to UML Models to Software Specifications*. Wiley, March 2009.

[24] K. Welsh and P. Sawyer, "When to adapt? identification of problem domains for adaptive systems," in *Proceedings of the 14th International Conference on Requirements Engineering: Foundations for Software Quality*. Montpellier, France: Springer-Verlag, 2008, pp. 198–203.

[25] K. Welsh, P. Sawyer, and N. Bencomo, "Towards requirements aware systems: Run-time resolution of design-time assumptions," in *Proceedings of the 26th IEEE/ACM International Conference on Automated Software Engineering*. Lawrence, Kansas, USA: IEEE Computer Society, November 2011, pp. 560–563.

[26] R. R. Lutz and I. C. Mikulski, "Requirements discovery during the testing of safety-critical software," in *Proceedings of the 25th International Conference on Software Engineering*. Portland, OR, USA: IEEE Computer Society, 2003, pp. 578–583.

[27] E. Letier and A. van Lamsweerde, "Reasoning about partial goal satisfaction for requirements and design engineering," in *Proceedings of the 12th ACM SIGSOFT International Symposium on Foundations of Software Engineering*. Newport Beach, California: ACM, 2004, pp. 53–62.

[28] O. Wei, A. Gurfinkel, and M. Chechik, "On the consistency, expressiveness, and precision of partial models," *Journal of Information and Computation*, vol. 209, no. 1, pp. 20–47, 2011.

[29] W. E. McUmber and Betty H.C. Cheng, "A general framework for formalizing uml with formal languages," in *ICSE'01: Proc. of the 23rd Intl. Conf. on Soft. Eng.* Washington, DC, USA: IEEE Computer Society Press, 2001, pp. 433–422.

[30] H. J. Goldsby, Betty H.C. Cheng, P. K. McKinley, D. B. Knoester, and C. A. Ofria, "Digital evolution of behavioral models for autonomic systems," in *Proceedings of the Fifth IEEE International Conference on Autonomic Computing*. Chicago, Illinois: IEEE Computer Society, 2008, pp. 87–96.

[31] D. Weyns, S. Malek, and J. Andersson, "Forms: A formal reference model for self-adaptation," in *Proceedings of the 7th International Conference on Autonomic Computing*, Washington, DC, USA, 2010, pp. 205–214.

[32] "Nist/sematech e-handbook of statistical methods," http://www.itl.nist.gov/div898/handbook/apr163, 2006.

[33] Y. Brun, G. M. Serugendo, C. Gacek, H. Giese, H. Kienle, M. Litoiu, H. Muller, M. Pezze, and M. Shaw, "Engineering self-adaptive systems through feedback loops," in *Software Engineering for Self-Adaptive Systems*, Betty H.C. Cheng, R. de Lemos, H. Giese, P. Inverardi, and J. Magee, Eds. Berlin, Heidelberg: Springer-Verlag, 2009, pp. 48–70.

[34] I. Epifani, C. Ghezzi, R. Mirandola, and G. Tamburrelli, "Model evolution by run-time parameter adaptation," in *Proceedings of the 31st International Conference on Software Engineering*. Vancouver, British Columbia, Canada: IEEE Computer Society, May 2009, pp. 111–121.

[35] V. E. S. Souza and J. Mylopoulos, "From awareness requirements to adaptive systems: A control-theoretic approach," in *Proceedings of the Second International Workshop on Requirements at Run Time*. Trento, Italy: IEEE Computer Society, August 2011, pp. 9–15.

[36] P. Oreizy, M. Gorlick, R. N. Taylor, D. Heimbigner, G. Johnson, N. Medvidovic, A. Quilici, D. S. Rosenblum, and A. L. Wolf, "An Architecture-Based Approach to Self-Adaptive Software," *IEEE Intelligent Systems*, vol. 14, no. 3, pp. 54–62, 1999.

[37] D. Garlan, S. W. Cheng, A.-C. Huang, B. Schmerl, and P. Steenkiste, "Rainbow: Architecture-based self-adaptation with reusable infrastructure," *Computer*, vol. 37, no. 10, pp. 46–54, 2004.

[38] A. Bertolino, W. Emmerich, P. Inverardi, and V. Issarny, "Softure: Adaptable, reliable and performing software for the future," in *Proceedings of the Future Research Challenges for Software and Services*, 2006.

[39] P. K. McKinley, Betty H.C. Cheng, A. J. Ramirez, and A. C. Jensen, "Applying evolutionary computation to mitigate uncertainty in dynamically-adaptive, high-assurance middlware," *Journal of Internet Services and Applications Special Issue on the Future of Middleware*, pp. 1–8, Online First, November 2011.

[40] N. Bencomo, A. K. Chopra, S. Clarke, H. Giese, P. Inverardi, V. Issarny, L. Pasquale, A. J. Ramirez, and S. Watzoldt, "Living with uncertainty in the age of runtime models," in *Models@run.time (Dagstuhl Seminar 11481)*, U. Aßmann, N. Bencomo, B. H. C. Cheng, and R. B. France, Eds. Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2012 (in preparation).