Uncertainties in the Modeling of Self-adaptive Systems: a Taxonomy and an Example of Availability Evaluation

Diego Perez-Palacin
Politecnico di Milano
Dipartimento di Elettronica, Informazione e Bioingegneria
Milano, Italy
diego.perez@polimi.it

Raffaela Mirandola
Politecnico di Milano
Dipartimento di Elettronica, Informazione e Bioingegneria
Milano, Italy
raffaela.mirandola@polimi.it

ABSTRACT

The complexity of modern software systems has grown enormously in the past years with users always demanding for new features and better quality of service. Besides, software is often embedded in dynamic contexts, where requirements, environment assumptions, and usage profiles continuously change. As an answer to this need, it has been proposed the usage of self-adaptive systems. Self-adaptation endows a system with the capability to accommodate its execution to different contexts in order to achieve continuous satisfaction of requirements. Often, self-adaptation process also makes use of runtime model evaluations to decide the changes in the system. However, even at runtime, context information that can be managed by the system is not complete or accurate; i.e, it is still subject to some uncertainties. This work motivates the need for the consideration of the concept of uncertainty in the model-based evaluation as a primary actor, classifies the avowed uncertainties of self-adaptive systems, and illustrates examples of how different types of uncertainties are present in the modeling of system characteristics for availability requirement satisfaction.

Categories and Subject Descriptors

D.2.4 [**Software Engineering**]: [Software/Program Verification]; I.6.4 [**Computing Methodologies**]: Simulation and Modeling

Keywords

Uncertainty; Self-adaptive software; Models;

1. INTRODUCTION

Today software is increasingly permeating (safety-)critical areas of daily life, from bank accounting to homeland security, from transport applications to power plant management and health care systems. Currently, there is also a huge increment in the demand of software applications that offer services to their users through mobile devices, which require minimum effort to install, configure and run. In these domains, non-functional properties like performance and availability of software are highly relevant, either to avoid damaging effects that can range from loss of trust on essential services to loss of human life in the critical system domain, or the loss of business and competitiveness in the marketplace for the mobile devices domain. Therefore, software should continuously meet its non-functional requirements.

To allow building software that executes with the appropriate quality, model-based evaluation methods at design time [3, 9] have been proposed as a viable solution. However, design-time analysis cannot always provide accurate results because the information of the environment where the application will be deployed may not be completely known when applications are initially architected; and even worse, such execution environment may change continuously during application lifetime. For example, when developing an application that can be potentially deployed over several platforms with different characteristics, software engineers have only a partial and incomplete knowledge of the external environment in which the application will be deployed. Consider, for example, a mobile device whose availability and reliability is very affected by the environment temperature. With the increase of the temperature, the CPU failure rate grows while the battery life decreases due to the effect of turning on fans, which consume battery. It is evident that if an instance of the application is deployed on this device, its reliability and availability properties will be strictly subject to the environmental temperature. Device constraints and temperature at which it will operate are in most cases uncertain at design time, which entails an uncertainty in the model that is used to evaluate system properties. This is further exacerbated in software that is embedded in dynamic contexts, where requirements, environment assumptions, and usage profiles continuously change. Since these changes in the context happen in a way that is hard to predict when systems are initially built, the outcome of the model analysis at design-time are in these cases subject to even higher uncertainty because assumptions upon which they rely on are not true.

To study these kind of uncertainties new methods emerged during the 1990s. The field of natural science has been a particularly active arena for methodological advancement, see for example [29, 4], and mathematical methods for quantifying uncertainties (e.g., interval analysis, fuzzy methods, probability theory and bayesian analysis [22]) have been developed in this context, starting from [18]. In the computer science field the topic of uncertainty has recently drawn the attention and some discussions and techniques have been presented in [17, 35, 10, 20, 19, 25, 32].

To deal with the lack of complete information and knowledge at design-time, in recent years, industry and academia have increasingly addressed the adaptation concern, particularly with the introduction of autonomic and self-adaptive systems. Self-adaptation endows a system with the capability to adapt itself to the environ-

ICPE'14, March 22–26, 2014, Dublin, Ireland.

http://dx.doi.org/10.1145/2568088.2568095.

ment where it executes. This capability relieves engineers from taking some decisions at design time. These decisions are delayed until runtime and made autonomously by the system, when more information is available, then dealing with the lack of knowledge about the environment at design time. Moreover, self-adaptive systems can perform successive adaptations when environment changes. In this manner, there is a possibility for the system to continuously achieve its functional and non-functional requirements even in dynamic contexts. The structure of the application may change at runtime; for example in terms of its running components and interconnections in order to improve its behavior, correct flaws, or reduce its energy needs.

Self-adaptation process entails several activities [24], some of them requiring planning, analysis and decisions. These activities can be achieved using formal models as suggested in [6] and with a seamless integration of design time and runtime verification. However, even if these models are useful and often the only possible artifact to reason about adaptations, their definition and usage raise some challenges because some uncertainties still remain present at runtime. One of the challenges is that knowledge of the environment is not complete or accurate even at runtime, which entails that the information in the models that are used to govern the adaptation process is subject to uncertainties.

In the literature, there have been proposed methods to deal with some kind of uncertainties that exist in self-adaptive systems. However, even though the works in the literature on modeling uncertainties in computing systems provide a useful approach for concrete types of uncertainty, we have not found a definition or taxonomy for uncertainty in models that can act as a pillar for building research work over it. These definitions for uncertainties in computing models can draw the big-picture that locates each piece of research in the field in its corresponding place; and therefore it will help researches to relate, connect and compare works, merge results and find similar works, ease the learning from these similar works, and push forward the research on uncertainties management. At present, in computing, the most used definitions of uncertainty simply distinguish between natural variability of physical processes (i.e., aleatory or stochastic uncertainty) and the uncertainties in knowledge of these processes (i.e., "epistemic" or state-of-knowledge uncertainty) [10, 20, 19, 25, 32, 14, 17]. Among the works in research fields that hold more maturity than computer science on the study of uncertainties, it is shown that *uncertainties* in models can be seen from other perspectives different from epistemic and aleatory. Learning from them, we have found that some of these different perspectives also exist in the modeling of computing systems; but there is not yet a general enough taxonomy for uncertainties in models in the computing field. A contribution of this paper is, exploiting the work of [33], to give a taxonomy for classifying different types of uncertainties that are present in software models. We then analyze, with respect to self-adaptive systems, the sources of uncertainties and the main approaches existing in the literature to handle them. As a second contribution, we use a concrete example of a self-adaptive system (concretely a system that can analyze its availability and adapt to increase it) with the objective of showing the existence of uncertainties in its managed models systems and how these uncertainties can be managed.

The remainder of the paper is organized as follows. Section 2 describes the existing works on uncertainty in the computer science field. Section 3 discusses the definition of uncertainty and presents a taxonomy describing different dimensions along which uncertainties can be classified. Existing sources of uncertainty and methods to deal with them in the context of self-adaptive systems are presented in Section 4, while examples of their usage are illustrated in Sections 5, 6 and 7. Section 8 concludes the work.

2. RELATED WORKS

In the computer science field the topic of uncertainty has recently drawn the attention and some discussions and techniques have been presented.

For example, works in [11, 14, 12, 13] admit that uncertainties cannot be eliminated in software systems and they propose techniques to manage the existent uncertainty. Specifically, [11, 14, 13] propose techniques to decide the suitable software architecture knowing the presence of uncertainty. They aim at minimizing the impact of uncertainty on architectural decisions. To achieve this goal, they guide how to rank, compare and choose an architectural configuration that maximizes the likelihood of satisfying the system's quality preferences. In [12], authors provide a list of sources of uncertainties that may exist in self-adaptive software systems. They also extend their method to compare the utility of an architecture by including how this utility is expected to vary over time within given constraints. Authors in [34] deal with requirements specification of self-adaptive systems and present a requirement definition language that captures the existing uncertainties. Works in [17, 7] present lists of sources of uncertainties. In particular, [17] explains the changes that uncertainties should entail in the development of software systems and it presents an enumeration of current research challenges to deal with these uncertainty. In [7] three sources of uncertainty specific for self-adaptive systems are identified namely, uncertainty in the identification of a problem in the system, uncertainty in the selection of strategy to adapt the system and solve the problem, and uncertainty in the identification the success or failure of the strategy. Authors integrate the management of these three uncertainties within the Rainbow approach.

A different set of works proposes specific techniques to deal with parameter uncertainties. Works in [15, 20, 10, 35] cope with prediction of reliability and availability of computer systems in presence of uncertainties. They share the usage of Markovian chains as mathematical model for representing software systems and present formal methods that address the challenge of uncertainties in the parameters of these mathematical models. In [10] the authors describe a Monte Carlo based approach and calculate the number of samples of uncertain parameter values necessary to produce availability results within a confidence interval. In [19, 20] authors use the method of moments for evaluating component-based software reliability under uncertainties. They deal with the presence of uncertainties in both the components estimated reliability and in the operational profile of software. Work in [15] estimates confidence levels of parameters of the software operational profile.

Model-based performance and reliability evaluation of software architectures in presence of uncertainties are tackled in [32, 26, 25]. Their methods are applied at software design time and aim at finding software designs or software component compositions that meet the non-functional requirements. They consider uncertainties in the values of the parameters of their models and propose to model this uncertainty through probability distribution functions. They extract samples of the parameter values and perform Monte-Carlo based simulations.

3. MODEL UNCERTAINTY: TAXONOMY

Several definition of uncertainties can be found in different areas of the scientific literature ranging from the absence of knowledge, to the inadequacy of information or the deficiency of the modeling process [33, 16]. Nevertheless, in computing area, the most

used definitions of uncertainty simply distinguish between natural variability of physical processes (i.e., aleatory or stochastic uncertainty) and the uncertainties in knowledge of these processes (i.e., "epistemic" or state-of-knowledge uncertainty), see for example [26, 10].

Among the work of research fields that hold more maturity than computer science on the study of uncertainties, it is shown that *uncertainties* in models can be seen from other perspectives different from epistemic and aleatory. Learning from them, we have found that some of these different perspectives also exist in the modeling of computing systems; but there is not yet a general enough taxonomy for uncertainties in models in the computing field.

For proposing such a taxonomy for uncertainties in computer systems models, we base on a general definition of *uncertainty* in modeling given in [33] as: "*any deviation from the unachievable ideal of completely deterministic knowledge of the relevant system*". Such deviations can lead to an overall "lack of confidence" in the obtained results based on a judgment that they might be "incomplete, blurred, inaccurate, unreliable, inconclusive, or potentially false" [29].

We present our classification based on the same three categories or dimensions as proposed in [33]. According to these three dimensions, uncertainties are classified regarding: their location, level and nature. In the following paragraphs we explain the meaning of each dimension.

Location.

The *location* of uncertainty refers to the place where the uncertainty manifests itself within the model. An uncertainty can be located in the following parts of a model:

- *Context* uncertainty is an identification of the boundaries of the model; that is uncertainty about the information to be modeled. This uncertainty concerns the completeness of the model with respect to the real world. It refers to the kind of information that should be included in the model and the kind of information that should be abstracted away from it. In Figure 1(a), elements within the dotted line represent a model that includes in its context elements *Service, Hardware* and *MicroHardware*, but it does not allow to represent *CommunicationNode* elements. If *CommunicationNodes* have a strong influence in system behavior, these models will hold a strong uncertainty. In turn, continuous line in Figure 1(a) encloses an example where elements *Service, Hardware* and *CommunicationNode* are in the context of the model, while elements *MicroHardware* are not.

- *Model structural* uncertainty concerns the form of the model itself. This uncertainty refers to how accurately the structure of the model represents the subset of the real world that has to be modeled. Following the example in Figure 1, let us assume that in the real system, due to fault-tolerance in the connections, two additional nodes exist allowing the communication between *Service* B and C. Since the model admits the representation of *CommunicationNode*, the replication of nodes could be represented (e.g., by adding the two *CommunicationNode* in dotted lines in Figure 1(b)). The model in continuous line keeps some uncertainty since its structure could represent better the real word.

- *Input parameters* uncertainty is often identified as parameter uncertainty and it is associated with the actual value of variables given as input to the model and with the methods used to calibrate the model parameters.

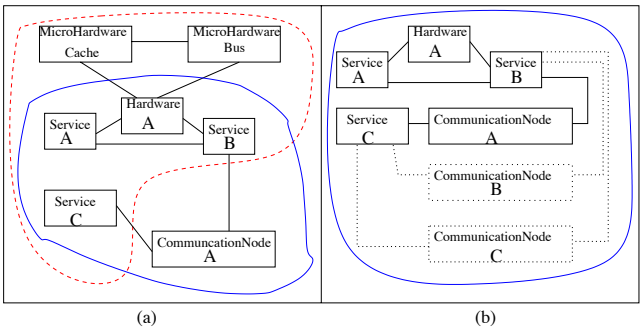

Figure 1: Example of location of uncertainties

The list of possible locations of uncertainties proposed in the literature is large. The rationale we have followed in the above selection is that we filtered out the locations that could hardly exist the models themselves. For example, we have not considered the uncertainties that are located in the solution algorithms of the models; e.g., uncertainty about the correct implementation of the algorithm that analyzes the model and produces the expected performance results of the system.

Doing an analogy to Model-Driven Engineering metamodeling levels, *context* uncertainties are related to the decisions at the metamodel definition (i.e., which kind of information can be included in the model), *structural* uncertainties are related to decisions at the model definition using the meta-model (i.e., the elements that exist in the model and their relations), and *input parameter* uncertainties are related to the values of the attributes of the model objects.

Level.

The *level* of uncertainty is where the uncertainty manifests itself along the spectrum between deterministic knowledge and total ignorance. Usual characterizations of uncertainty levels propose different values in a scale of how much knowledge lacks to achieve the knowledge necessary for studying the system deterministically.

We believe that a classification that differentiated between several amounts of lack of knowledge in a very tailored manner could misguide future research. Hereafter, to avoid classifications that could hamper the progress in the field of uncertainties management by proposing premature biased uncertainty levels, we prefer to classify the level of uncertainty following the more general ranking of *orders of ignorance* proposed in [1]. The five proposed levels of ignorance (here for uncertainty) are:

- 0th order of uncertainty. Lack of uncertainty, i.e., knowledge.

- 1st order of uncertainty. Lack of knowledge. The subject lacks knowledge about something but she is aware of such lack (i.e., *known uncertainty*).

- 2nd order of uncertainty. Lack of knowledge and lack of awareness. The subject does not know that she does not know.

- 3rd order of uncertainty. Lack of process to find out the lack of awareness. The subject does not have any way to move from not knowing that she does not know to, at least, be aware of the existence of the uncertainty.

- 4th order of uncertainty. Meta uncertainty. Uncertainty about orders of uncertainty.

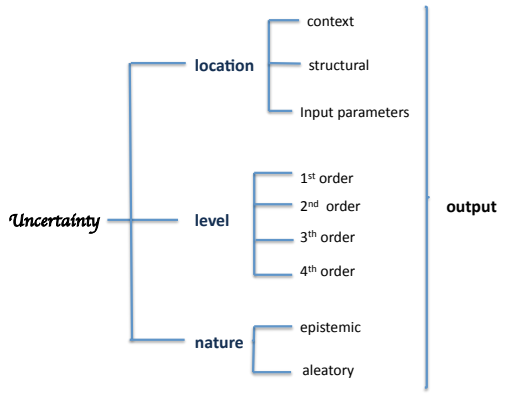

Figure 2: Uncertainty taxonomy

Following this classification, software engineers should build self-adaptive applications having in mind that uncertainties of third and fourth order in their application models should be avoided. The second order of uncertainty, instead, may be unavoidable in some cases (see for example the source of uncertainty called *problem-state identification* in [7]). What is more important for the system is that an uncertainty remains in this level in a transitory manner; i.e., eventually the uncertainty will be recognized and then it will decrease its level to the first. Once the uncertainty is in the first order, it could be used some of the known methods proposed to deal with it. The reduction from the second order to the first one may be assisted, for instance, by providing self-adaptive software both with self-evaluating mechanisms and with monitoring capabilities. In that case, if the results of model-based evaluation do not match with the real monitored characteristics, then the existence of an uncertainty in its models can be recognized (then moving the level of this uncertainty to the first order). Even in presence of these characteristics, uncertainties of second level can be present. Consider, for example, the case in which performing the model-based self-evaluation is time consuming and then it is performed only periodically. In this situation, the system may not continuously be aware of its lack of knowledge, but the uncertainty will be eventually recognized and then it will belong to the second order only for a time-bounded period.

Nature.

The *nature* of uncertainty refers to whether the uncertainty is due to the imperfection of the acquired knowledge or is due to the inherent variability of the phenomena being described. Our taxonomy uses the classical distinction between:

- *Epistemic* uncertainty due to the lack of enough data to build reliable knowledge, imperfection of the acquired data or imperfection in the process of building the knowledge from the data.
- *Aleatory* uncertainty due to inherent variability of the some parts under consideration or randomness of events.

The presented three-dimension classification is sketched in Figure 2. The aggregated effect of the different uncertainties on the results of the model analysis is the so-called model *output*. The analysis outcome will have an uncertainty that derives from the uncertainties in the information represented in model and how they are handled during analysis. Up to now, there exist methods to manage the presence of uncertainties in the model analysis, but there do not exist methods to completely eliminate them. Therefore, the *output* uncertainty cannot be avoided. Using methods that consider the uncertainties, it is expected that the outcome of an analysis that considers the presence of uncertainties to be closer to the real values of the running system than the outcome of an analysis that do not consider their presence; though this fact cannot even be completely ensured.

4. MODEL UNCERTAINTY AND SELF-ADAPTIVE SYSTEMS

When an application is initially architected, the available information for engineers regarding some important concepts - such as the environment where the application will execute, the usage profile of the application or its requirements - is partial and incomplete. This is reinforced by the facts that all these concepts are prone to change during the application lifetime, and it is impossible to foresee every type of change, the moment in which it will occur and the value to which they will change. Therefore, models that are used to evaluate the application properties at design time hold some uncertainties.

Self-adaptive systems are an effective solution to deal with some aspects of the lack of information and actual knowledge that exist when the application is not running yet (e.g, at design time). By building a self-adaptive system, some of the decisions that should be otherwise made by engineers at design time can be delayed, and they can be made by the system itself at runtime when more information about changing concepts is available. Self-adaptive systems usually keep a model of their "world of interest", and make use of it to plan their adaptations and when to adapt. However, even at runtime, information is not complete or accurate. Therefore, information in the models used to govern the self-adaptation is subject to uncertainty [12].

Research works [17, 12] have reported concrete sources of uncertainties in software systems. Work in [8] also lists sources of uncertainty. Since the concern of [8] is a model-based software reliability evaluation, their presented sources are concentrated on the uncertainties on software architectural models. We have gathered different sources of uncertainty in software systems presented in the literature and we have investigated how these uncertainties can affect the trustworthiness of the information in the models managed by self-adaptive software. Once the effect of each source of information in the model is recognized, we can find a relation between each source of uncertainty and a type of model uncertainty according to the taxonomy presented in Section 3. We present this relation in the next subsection.

To avoid incorrect decisions about system self-adaptations, the software should know that its model contains uncertainties and apply some methods to handle them during the model analysis phase. We discuss in Subsection 4.2 methods for handling model uncertainties presented in the literature.

4.1 Sources of uncertainty

In this subsection we propose a classification of the sources of uncertainties that we have found in the literature regarding the effect they entail in the model managed by the self-adaptive system. We classify them according to the taxonomy presented in Section 3. This will make easier the comparison of the similarities and differences of the effect of different sources of uncertainties. Having them classified according to a taxonomy (instead of using simple lists) will help the selection of general approaches that can deal with a group of uncertainties concurrently, rather than a particular approach at a time for each uncertainty. This relation is shown in

Table 1. Each source of uncertainty is classified according to its *location* and *nature* dimensions of the taxonomy. The *level* dimension of the taxonomy is not shown in the table since each source of uncertainty can be of any level depending on the implemented capabilities in the system that should deal with the uncertainty.

Source of Uncertainty	Classification	
	Location	Nature
Simplifying assumptions [12]	Structural/context	Epistemic
Model drift [12]	Structural	Epistemic
Noise in sensing [12]	Input parameter	Epistemic/ Aleatory
Future parameters value [12]	Input parameter	Epistemic
Human in the loop [12, 17]	Context	Epistemic/ Aleatory
Objectives [12]	Input parameter / context	Epistemic
Decentralization [12]	Context/structural	Epistemic
Execution context/ [12] Mobility [17]	context/ structural/ input parameters	Epistemic
Cyber-phisical system [12] [17]	Context/Structural Input parameter	Epistemic
Automatic learning [17]	Structural Input parameter	Epistemic Aleatory
Rapid evolution [17]	Structural Input parameter	Epistemic
Granularity of models [8]	Context/Structural	Epistemic
Different sources of information [8]	Input parameter	Epistemic/ Aleatory

Table 1: Sources of uncertainty

Due to space reasons, we do not provide a description of each source of uncertainty here. Readers are referred to works referenced in Table 1 for further details regarding the meaning of each source of uncertainty. Nevertheless, for the sake of understandability of the process that was followed for the classification of the sources of uncertainty, we describe in the following some of them as examples. We have selected: *simplifying assumptions*, which is an uncertainty very general in modeling and easily understandable, *future parameter value* and *automatic learning* that are uncertainties more restricted to the domain of self-adaptive systems.

Simplifying assumptions: the model is an abstraction of the reality and some details whose significance is supposed to be minimal are ignored. This uncertainty can be located, for example, in the structure (i.e., *structural* location) of the model if the model language has enough modeling power to represent the lacking concepts but they have been deliberately excluded from the model. It can also be located in the boundaries (i.e., *context* location) of the model if it was decided to exclude some information within the set of type of concerns considered in the model. To illustrate the difference, let us refer again to the examples in Figure 1. If it is possible to model the characteristics of every existing *CommunicationNode* (e.g., routers) between *Services* but it was decided not to represent the replication of *CommunicationNode* between two *Services* (elements in dotted lines in Figure 1(b)), then the results of the model evaluation will be uncertain due to "simplifying assumptions" in the model *structure*. If properties of hardware micro-components such as cache memory or the bus between CPU-memory have an influence in the system properties but the language to create the model does not allow modeling the characteristics of micro-components (continuous line in Figure 1(a)), then the results of the model evaluation will be uncertain due to "simplifying assumptions" in the model *context*. This source is related to a deliberate hiding of information in the model (i.e., *epistemic*) rather than to the random nature of the lacking elements.

Future parameter value: uncertainty in the future world where the system will execute creates uncertainties in the correct actions to take at present. For example, if the self-adaptive application is not in the optimal configuration for its current execution environment, model analysis may produce an advice to change its configuration. However, if the environment conditions are close to a change and the current configuration is the optimal for the future environment, the best behavior may be to resign itself to executing for a short period in the sub-optimal configuration without requiring any adaptation, instead of performing two adaptations in a short time interval. As future changes can involve changes in the model structure or parameters, this source of uncertainty may have *structural* or *input parameter* location. Since this source of uncertainty concerns the future of the environment it shows an *aleatory* nature.

Automatic learning: adaptive applications that include an automatic machine-learning phase usually use statistical processes to create their knowledge about their execution context and most-useful behavior. This machine-learning process can lead to applications with uncertain behavior. The location of this uncertainty may be in the model *structure* or *input parameters* depending on how general are the concepts for which the application has been provided with machine-leaning capabilities. Regarding the nature of the uncertainty, it may depend on the point of view. From the point of view of the application and its models of the world, as long as the origin of this uncertainty is that the machine had to learn using imperfect and limited data, the nature of this uncertainty is *epistemic*. From the point of view of the user, since the information may have passed through a statistical process during its learning to create the models, it produces some randomness in the model information and analysis results, and consequently it can be seen as an *aleatory* uncertainty.

4.2 Methods to manage uncertainty

While research in uncertainty analysis advances, the set of different sources of uncertainties becomes larger. Hopefully, there will be no need of a completely different and particular approach for handling each type of uncertainty in the system. Indeed, it would be much better if more general approaches for managing uncertainties could be reused for different sources of software uncertainties; for example, two uncertainties that at first sight may look very different can be managed using a similar approach.

In order to have a possibility to manage model uncertainties, the first step is to create an application that is eventually able to identify the existence of such uncertainties; i.e., the *level* of uncertainty should be at most in the 2nd order. Once the uncertainty is in the second order, its reduction to the first order one may be assisted, for instance, by providing self-adaptive software both with self-evaluating mechanisms and with monitoring capabilities. Other techniques that have been proposed in the literature that allow the software to realize the existence of uncertainties are the *multiple conceptual model* [28], which proposes the analysis of several models of the same system to realize the existence of uncertainties if their results differ from each other; and *expert elicitation* [28], which manually sets the uncertainty to belong to the 1st order. As previously mentioned, even in presence of methods to reduce the level of uncertainty to the 1st, uncertainties of second level can be temporarily present because the system does not apply these methods are continuously but only periodically.

Once the placement of the uncertainty level on the first order has been achieved, the scientific literature has proposed several methods for handling it. In the following we list the methods we have found for managing uncertainties. In order to make easier its utilization on the appropriate uncertainties, we classify them regarding the type of uncertainty for which they were initially conceived. Table 2 shows such classification. Some of these methods were not proposed in computer science but derived from other research areas. We do not argue against the possible usefulness of the methods across other types of uncertainty; further research is necessary to completely understand whether they can be used.

Since some methods for managing uncertainties were created through continuous refinements or extensions of other general analysis methods, references in Table 2 do not show the origin of the method but a work in which they are applied or their suitability is discussed. Due to space limitations, we do not provide a description of each method. We refer readers again to works referenced in the table for further details regarding the usage of each method. We selected here two of them, which are powerful techniques but not frequently used in computer science or performance evaluation field: *model averaging* and *model discrepancy*.

Model averaging: This approach proposes the generation of several models of the same system. Different modeling languages and model domains can be used. These models are competing and they are all plausible. Each model is assigned with a probability of being the "true" model; i.e., a measure of model adequacy. Every model is analyzed and the outcome of the analysis is calculated as the weighted mean -according to the measure of model adequacy- of the outputs of the models. A self-adaptive system should be provided with a set of methods, one for each type of model, for creating, updating and analyzing the model.

Model discrepancy: This approach assumes that the utilized model is not the "true" model of the system. Instead, it tries to unveil the discrepancy between the model output and the "true" target value. Once the discrepancy is known, it is created a "discrepancy term". For creating the outcome of the analysis, both the model analysis output and the discrepancy term value are considered. Ideally, the discrepancy term is equal to the difference between the output of the model running at its best (i.e., where the input values of the model are all equal to the real values) and the real results (the true target quantity). Calling X the input of the model, $f(X)$ its output and Z the real values; then, the discrepancy term δ_x ideally satisfies $Z = f(X) + \delta_x$. Using statistical techniques and a subset of inputs $X_1, X_2...$ and outputs $Z_1, Z_2, ..$, the discrepancy term δ_X can be estimated parametrically to be used in subsequent analysis.

5. APPLICATION EXAMPLE

In this section we exemplify some of the model uncertainties classified in Section 3. We base our examples in the field of self-adaptive service-oriented systems availability evaluation.

Consider a software application whose functionality is the viewing of video in streaming (real-time events, films, etc.). To meet its mission, it requires services that are offered by third-party service providers over the Internet; e.g., streaming video servers. As there may be multiple providers for each required service, to increase the application's quality, it will be engineered with self-adaptive capabilities in terms of dynamic *service provider selection*; e.g., if it is using a service provider and it becomes unavailable, the application will be able to autonomously bind another different provider. Let us assume that there are N third party-providers, named sp_n such that $n \in [1..N]$. Figure 3 represents a system of this kind. In this example, the software application resides in a mobile device and can connect to the internet using several access media and protocols. Let us assume that there are M access media, named am_m such that $m \in [1..M]$. The application can adapt its behavior for availability reasons (the kind of network that it is using and the third-party server that is requested to execute the service). If the user starts watching a stream and it disrupts, he will be dissatisfied with the application.

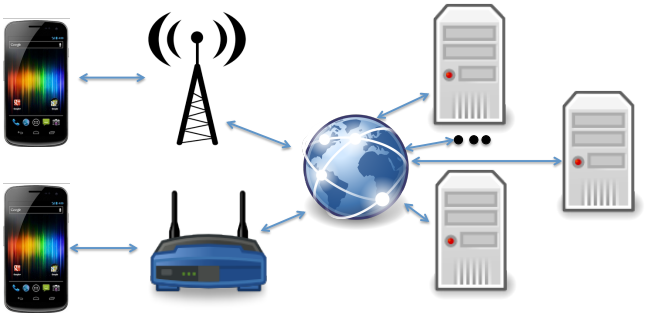

Figure 3: System base case example

To perform the model-based evaluation of the application's availability, the application itself needs the availability models of the world where it executes. In the following sections, we present a modeling study of this application from different perspectives and we highlight the existence of different types of uncertainty among the ones identified in Section 3 in the availability models, and a manner to handle each type of uncertainty. We divide the presentation according to *Location* dimension in the sections below and we concentrate on the dimensions *input parameter* and *model-structure* for the sake of space. Although it is a modeling study, to make realistic examples, we use real data. In particular, due to lack of accessibility to the logs of real deployment of this system, we use the availability logs published in [2, 5, 21, 27, 30] regarding the availability of internet servers. In the examples below we will use data from these logs and we select each time the log containing the more representative data to illustrate the uncertainty we are dealing with.

6. INPUT-PARAMETER UNCERTAINTIES

Nature: aleatory.

To generate the availability model, when the user wants to watch a stream, the application monitors the responsiveness of each element it needs; i.e., the lack of deadlock in the application, the responsiveness of internet access points (via mobile phone antenna or WiFi router) and third-party servers. Using this information, it can fill a block diagram model for availability. Figure 4 represents an instance of this model, where the field `status` of each block is filled with the information monitored.

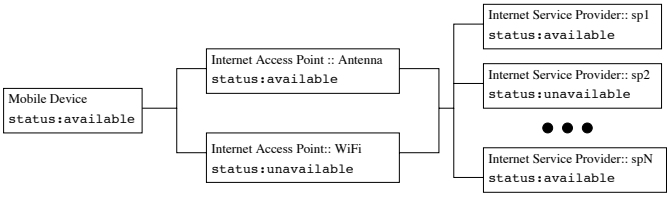

Figure 4: Block diagram model for availability analysis

Location	Nature	Method
Input parameter	Epistemic/ Aleatory	Reliability bound [35], Confidence intervals [35], Probability distributions [35], Fuzzy methods, Range of values, mean and variance, Sensitivity analysis,Sensitivity to information sources [8]
Structural	Epistemic/ Aleatory	Increasing parameter uncertainty to account for structural uncertainty [28]
Context/ Structural	Epistemic	Sensitivity to model granularity [8]
	Epistemic/ Aleatory	Structural uncertainty term [28], Model averaging [23], Model discrepancy [31], Framework for the establishment of plausible models in [28]

Table 2: Methods to manage uncertainty

In this case, since there is a path from the beginning of the diagram until its end that passes through blocks whose `status=available`, the analysis results will inform about the availability of the system, the application will bind an unavailable service provider and it will inform that the stream can be watched.

Problem:

In this case, the application is not aware that there is *uncertainty in the future state* of service providers and access points during the stream viewing. In consequence, the availability analyses ignores that the state of service providers and access points can change in the near future. Thus, the user will be dissatisfied if the system becomes unavailable because he had been informed that the access to the stream was granted.

Since the sample space size in the availability modeling of an element is two (either `status=unavailable` or `status=available`), the outcome of a study that ignores the aleatory nature of availability parameters of required elements is obviously insufficient for informing the user about system availability.

Taming uncertainty:

The availability analysis should consider that some servers may become unavailable, others may become available, mobile phone coverage may be lost or a wifi access point may be gained. In this manner, the application is aware that the state of the required elements during the streaming is not deterministically known; which moves it to the first level of uncertainty and enables the uncertainty management.

This type of uncertainty is a well-studied case in the literature (e.g., the exponential distribution is broadly used to model the time-to-fail of elements, while the accessibility of an element at a certain moment is usually modeled as a Bernoulli trial within a Bernoulli process) and well covered in availability evaluation research field [35, 10, 20]. The random nature of the availability parameter value of an artifact when its execution is required is usually captured by *probability distributions*.

Once the existence of this uncertainty is recognized, instead of representing in the availability model the status of an element as either `available` or `unavailable`, we model the aleatory uncertainty of each element using a Bernoulli distribution with parameter p: each third-party provider sp_n, accessing media am_m and application app has a parameter $p_{element}$ denoting a probability value, where $element \in \{sp_1 ..., sp_N\} \cup \{am_1, ...am_N\} \cup \{app\}$.

The application needs now a source of information to obtain the $p_{element}$ values. As a straightforward method, the application can record the availability information of each element periodically and create logs with the historical data of elements availability.

Using the data in the selected availability logs, $p_{element}$ values are calculated as the proportion of time that the element is accessible divided by the amount of time covered in the logs. Let us consider, for example, the availability log file `web-sites.avt` in [2]. Specifically, we focus on the availability information of the 7th server in the log for our provider sp_1, which corresponds to the information monitored during 209 days about the availability of mail.yahoo.com, and we obtain that the server was reachable 99.29% of time; therefore we set $p_{sp_1} = 0.9929$. Following a similar procedure, we derived the parameter values of the availability model in Figure 5. The probabilities have been computed using as availability log the file `web-sites.avt` in [2] and servers in lines 2,3,4,5,7,8 corresponding to servers: asia.cnn.com, canberra.yourguide.com.au, digital.library.upenn.edu, games.yahoo.com, mail.yahoo.com and msdn.microsoft.com, respectively.

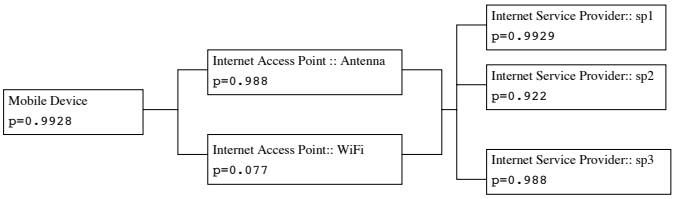

Figure 5: Block diagram model with probabilities for availability analysis

Using standard techniques of availability analysis of series/parallel blocks:

$$SystemAv = appAv \cdot amAv \cdot spAv = 0.98177$$

where $appAv = p_{app} = 0.9928$, $amAv = 1 - \prod_{m=1}^{M}(1 - p_{am_m}) = 0.9889$ and $spAv = 1 - \prod_{n=1}^{N}(1 - p_{sp_n}) = 0.999993$.

Therefore the system knows that, in this configuration, it has a probability of 0.98177 for being available and can allow the user to make an informed decision whether to use the application in this moment or not.

Note that in this example, the modeling of the availability uncertainty through a probability may seem evident. However, this kind of uncertainty is not always modeled when the main outcome of the study is not the availability evaluation but, for instance, performance evaluation. In these cases, the randomness in the availability of components or service providers is not represented and they are represented as "always available".

Nature:epistemic.

In the application example, the p_{sp_n} value characterizing a provider has been calculated as the proportion of time that the provider was available with respect to its lifetime. All data in the provider log has then be used to calculate p_{sp_n} with the underlying assumption that the log contains data regarding the steady-state behavior of uptimes and downtimes.

Problem:

The application in our example collects availability data from entities that run and are engineered independently of the rest of the system. As the application cannot acquire more information about the third-party service providers, these data may be biased.

The real steady-state availability behavior of the service provider, indeed, may be different from the one deduced from the log. For example, consider the case in which some bugs that caused unavailability periods are detected and corrected, in this case the real availability will be higher with respect to the predicted one. On the contrary, if it happens that after the deployment is finished, the service is no longer the protégé and the engineering team and sysadmins change their priorities to other projects, in this case the real availability can decrease.

This uncertainty does not stem from a random nature of changing elements in the service provider but it is due to a lack of knowledge about how to interpret the data that are collected during the monitoring and which data will be used to give value to model parameters. Therefore, this is an epistemic uncertainty in the model parameters.

Taming uncertainty:

This kind of uncertainty has been tackled in the literature conducting sensitivity studies [20], using confidence intervals or distribution functions for the value of parameter p_n [25, 32, 11].

Hereafter, instead of showing what it does mean dealing with this uncertainty considering a first order level, we illustrate how to tackle the second order level. In other words, we illustrate the change from a situation where the application does not know that the availably log does not contain enough information to a situation where the application knows that the information in the log does not represent the service provider's current behavior.

Let us focus the study on one service provider, sp_n. We use as availability log for sp_n the availability data in the 13th line in `web-sites.avt`[1]. The usual availability probability calculation based on the proportion of time the system was available regarding the total amount of time would give us:

$$p_{sp_n} = \frac{17,924,686 seconds}{18,135,257 seconds} = 0.98838$$

To recognize whether the information in the trace represents the steady-state availability behavior of the provider or not, we split up the trace in two halves where each half covers the same amount of time. Then we compare the availability obtained considering the whole trace and the ones related to the two halves. If the log contains the steady-state information, the availability calculated in the three cases above should be similar.

Let us start with the information in the second half of the log. In this case we obtain:

$$p_{sp_n} = \frac{9,061,128.5 seconds}{9,067,628.5 seconds} = 0.99928$$

Therefore, in terms of availability, using only the data in the second half of the log we obtain a probability of downtime of the provider that is $\frac{1-0.98838}{1-0.99928} \simeq 16$ times lower than the one obtained using the complete log.

To strengthen the study, we calculate the availability of other parts of the logs representing the most recent information. For example, if we calculate the availability using the last quarter of the log trace, we obtain

$$p_{sp_n} = \frac{4,532,654.25}{4,533,814.25} = 0.99974$$

representing a system whose downtime seems $\frac{1-0.98838}{1-0.99974} \simeq 45$ times less probable with respect to the information derived analyzing all data in the trace. Besides, we calculate the mean length of each time interval when the system is available and the mean length of each time interval when the system is unavailable. Figure 6 shows this information. Figure 6 (a) shows the mean time interval length that the service is continuously available when using the complete log (x-axis is 100%), the last half of time (x-axis is 50%) and the last quarter of time (x-axis is 25%). In turn, Figure 6 (b) shows the mean time interval length that the service is continuously unavailable (i.e., repair time) for the same log partitions. These figures show that the availability of the service is evolving positively.

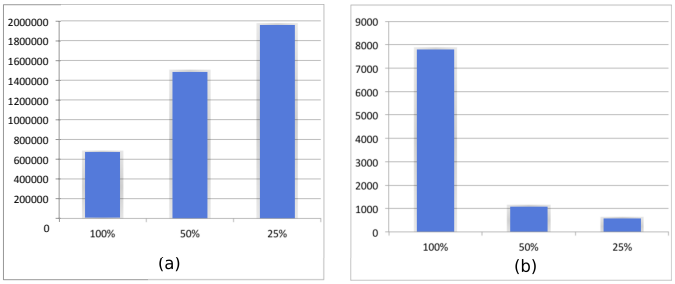

Figure 6: Mean length of time intervals where the service in the 13th line in `web-sites.avt` is continuously: (a) available, (b) unavailable

Observing the differences in the calculated availability depending on the considered data in the log, we can suppose that the log does not show steady state availability information (in the beginning the system is much more unavailable than in the end). Thus, the calculation of the availability probability using the proportion of time that the system has been available since the first moment of existence is not accurate because there is a epistemic uncertainty in the model input parameters [2].

Using this procedure, the existence of the uncertainty can be realized. At this point, an approach among the ones presented in the literature, for instance in [10, 15, 8], can be used.

7. MODEL STRUCTURAL UNCERTAINTIES

Nature: aleatory.

The model in Figure 5 calculates the probability of finding available a service provider as the proportion of time that the provider was available regarding the whole time monitored. A service provider is represented as an element of *Internet Service Provider::spN* in the model structure, and the availability model contains an *Internet Service Provider::spN* element for every service provider that have been registered in the log.

Problem:

Let us now consider the same system and model from a different perspective. As a system that operates in the open-world and uses third-party service providers, new service providers that offer

[1]The monitored website corresponding to the 13th line is vlib.org, and the trace covered almost 210 days

[2]The calculation of the threshold values for which it is assumed that the average system availability is either similar or different to the recent system availability is out of the scope of this example.

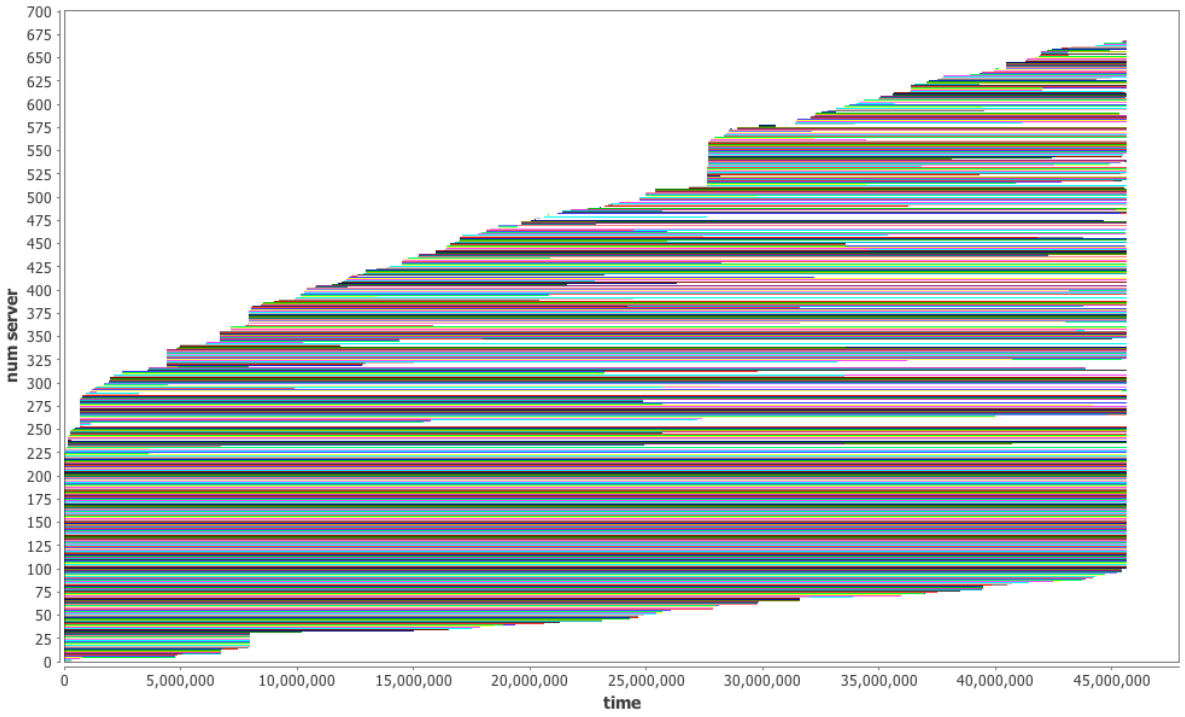

Figure 7: Birth and Death times of servers in PlanetLab ordered chronologically

the same service may appear and existing providers may disappear from the world. Therefore, the number of existing service providers at a given moment may vary over time. The appeareance of a service provider may be represented in the log by a timestamp denoting the time in which it was available by the first time. However, disappearance of servers is not represented in the log. As a consequence, the availability model is not completely accurate because it may consider providers with a calculated availability that do no longer exist.

As a concrete example, let us consider the information in file `pl-app.avt` in [30] as availability log of service providers. This file contains information regarding the availability of 669 servers of PlanetLab platform. Therefore, service provider availability log shows the availability sessions of 669 service providers. Figure 7 shows, for each provider, the first time and the last time its service was available. So, each line in the figure represents the lifetime of a provider. We can see that the first session of availability of some providers was in a point of time days away from the starting of the application (situation in which we could say that these providers did not exist in the beginning but they appeared in the world later). In the same way, we can see that some service providers passed through periods of availability and unavailability until a certain point in time in which they started to be continuously unavailable (situation in which we could say that they had disappeared from the world). This figure shows that 669 different providers have appeared in the world at some point but the average number of existing providers is 384.8.

To illustrate an example of uncertainty in availability modeling, let us consider the value *numServer*=51 in Figure 7 [3]. This provider was available since the first time the application was executed and the last time that this provider was available was 24,895,542 seconds after the application was firstly executed. The current timestamp is 45,573,054. Between the starting point at 0 and 45,573,054, the server was found available for 21,659,147 seconds, which gives it a calculated availability probability $p_{sp_{310}} = 0.4753$. However, since the provider has not been available during the last 45,573,054 - 24,895,542 = 20,677,512 seconds (239 days), it is hardly believable that the provider is just temporarily unavailable. The reasonable option is to consider that the provider does not longer exists, and therefore its representing block *Internet Service Provider::sp51* in the block diagram should be removed from the model structure.

The appearance and disappearance of providers happen many times and cannot be anticipated deterministically because they depend on the decision of third-parties and these are random events from the point of view of the application. From a modeling point of view, since each existing server is represented as an element in the structure of the model, and this element can be present or not, this is an uncertainty of aleatory nature located in the model structure.

Taming uncertainty:

To deal with this uncertainty, assuming that its existence has been recognized and so it belongs to the first order level, several techniques can be applied. In the following we illustrate the application of the model averaging technique.

Let us assume that, from previous experiences, engineers know that observing a continuous unavailability of a provider during the last week would likely mean that this provider disappeared. Following this assumption may entail both pessimistic prediction errors (e.g., providers that suffered a serious downtime of more than one week but they will be available again, the so called false positives), and optimistic ones (e.g., providers that have left the world only a couple of days ago would still be considered as currently existing, the so called false negatives).

[3] This line corresponds to line 310 in file `pl-app.avt` in [30], with `ip=152.3.136.1`

Following the model averaging technique, we create three different availability models. The input data we use in this experiment are the first 14 lines in file `pl-app.avt` in [30]; thus we consider that in the world up to 14 different providers have appeared at some point in time. Figure 8 depicts the birth and death moments of each of these providers.

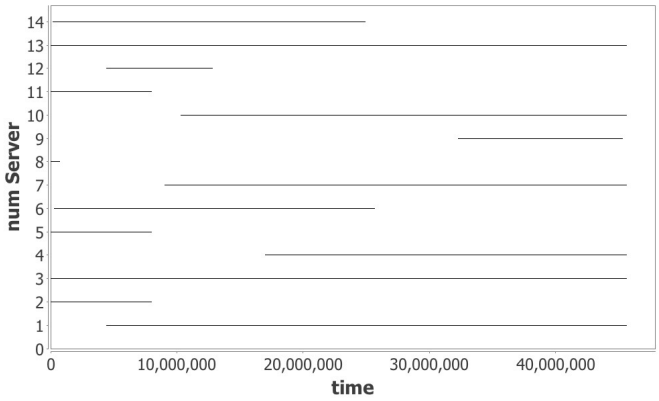

Figure 8: Birth and Death times of PlanetLab servers in the 14 first lines in `pl-app.avt`

The *first availability model, M1* is generated by considering only the providers that have not been continuously unavailable during the last week. That is, since in the used traces the current time is the 45,573,054th second, we take into account servers whose last monitored availability was after the $45,573,054 - secondsInOneWeek = 44,968,254$-th second. This gives us an availability model that includes N=7 service providers in its structure. The calculated availability of service providers ($spAv$) using the standard techniques of availability analysis of parallel blocks gives us an availability of $spAv = 1 - \prod_{n=1}^{N=7}(1 - p_{sp_n}) = 0.9999982$.

The *second availability model, M2* is intended to mitigate the uncertainty inaccuracies due to too pessimistic assumption. This is done by using the calculated availability of every provider since the first time it appeared in the world until the current time. The availability of the system is calculated as if none of the providers had never left the world. This gives us an $spAv = 1 - \prod_{n=1}^{N=14}(1 - p_{sp_n}) = 0.9999996214$. Note that although $spAv$ value in this case can seem similar to the previous one, this availability value represents a system almost five times more available than the previous one.

The *third availability model, M3* considers as existing providers only those that are currently available at the 45,573,054-th second. Among the 14 providers, 6 of them are currently available. Therefore the structure of the availability model for the service providers has 6 blocks. We calculate the p_{sp} of each of these T service providers and we use these probabilities to calculate $spAv = 1 - \prod_{n=1}^{N=6}(1 - p_{sp_n}) = 0.99999617$. Note that this availability value represents a system around half time less available than the one obtained with $M1$.

None of the three availability models exactly represents the real situation regarding the number of providers that currently exist in the world as this is an aleatory uncertainty. The first one makes a prediction regarding the moment in which a provider can be assumed as disappeared, the second one makes an over-estimation of existing providers and the third one makes a sub-estimation.

M1	M2	M3	Mave
0.9999982	0.9999996214	0.99999617	0.9999981

Table 3: Availability of service providers $spAv$

Model averaging techniques propose to manage this uncertainty by weighting the availability results of each model in order to produce a new result. In our example, we assume to be quite confident about the one-week existence assumption and we provide a weight of 0.7 to $M1$, 0.15 to $M2$ that over estimates the existence of providers and 0.15 to the model $M3$ that under-estimates it. This gives us a service provider averaged availability of $spAv = 0.7 \cdot 0.9999982 + 0.15 \cdot 0.9999996214 + 0.15 \cdot 0.99999617$, so

$$spAv = 0.9999981$$

Table 3 summarizes the results obtained with the different models.

Nature:epistemic.

The application in our example communicates with service providers through Internet Access Points. In Figure 3 and in the block diagram in Figure 4, two Internet Access Points, a WiFi point and an Antenna working in parallel are represented. This model assumes that the communication across Internet between the application and service providers is a single-hop process transmitted by an Internet Access Point.

Problem:

In the actual system, the communication between the service provider and the application is not a single-hop process routed in isolation by the Internet access point of the application. There are other elements that can fail in the communication, for example, the Internet Access Point of providers, DNS servers or Internet traffic routers in between. We concentrate this example on the Access Point of providers.

When the service is monitored as not available, the reason can stem from an unavailability of its Internet Access Point rather than from the unavailability of the service itself. Monitoring the availability of Internet Access Point of providers is not as trivial as controlling the unresponsiveness of the service, although tools as `traceroute` can identify failures in communication elements. If the access point fails, services may be available but they cannot be reached. Although it is easily noticeable that the lack of the Internet Access Points in the availability block diagram creates an inaccuracy in the model, it is also reasonable to justify that the annotated availability in the service provider block already covers every possible unavailability in the communication elements. Nevertheless, by avoiding the modeling of the Internet Access Points of the service provider, we are also falling into a modeling inaccuracy regarding the independency of the availability of service providers among each other. Next paragraph exemplifies this case.

Nowadays, many companies and service providers over the internet are moving their computing infrastructure to the cloud, let us assume that this is the case of the service providers in our example. In this case, two or more providers can rely on the same cloud provider and belong to the same *availability zone*. Therefore, they share the same Internet Access Point, i.e., the owned by the cloud provider. In our availability model, by using the common formula $spAv = 1 - \prod^{N}(1 - p_{sp_n})$ it was implicitly assumed that each service provider was unavailable independently of others, but in reality there are single motives that can make them be useless at the same time.

The nature of this uncertainty is epistemic because it is due to a lack of knowledge of how the communication between the application and service providers proceeds. It is located in the model structure because the Internet Access Points of service providers are not considered as blocks in our model, even if it would be possible since the type *Internet Access Point* already exists in our availability block diagram.

Taming uncertainty:

The uncertainty related to the lack of knowledge on the model structure is difficult to be identified at the beginning of the execution. It is most likely that the application becomes aware of the uncertainty during its lifetime, when more data about services availability is acquired. Therefore, this uncertainty will start belonging to the second order, and eventually will become of first order. For this reason, we believe it is more convenient to illustrate afresh the taming of the uncertainty of second level, suggesting then more classical methods to deal with it when it is at the first level.

A reasonably easy manner to realize the existence of uncertainty in the modeling of providers reachability is to check the likelihood of concurrent unavailability of services. The application should first recognize that providers are not behaving independently, although at this point it could not be clear which are the actual dependencies. If it is possible to identify the Internet Access Point of providers as the source of dependency, then the uncertainty will be of first order. The system will know that the structure of its model is not completely correct because there are dependencies in the availability behavior of providers but it lacks complete information to include them in the model.

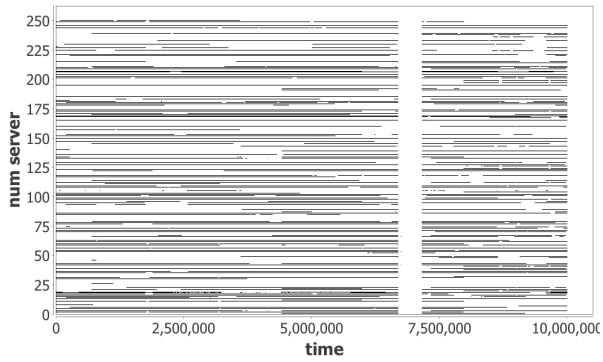

Figure 9: Availability intervals of 250 first servers in `pl-app.avt`

Figure 9 shows the availability intervals of the first 250 servers in file `pl-app.avt` in [30] during the first ten million seconds. Let us assume that this is the availability behavior of $N = 250$ services of our example system. Each y-axis value represents the availability behavior of a service. Horizontal lines are depicted during the periods when the service sp_n was available. It can be seen that availability of the services is not independent because around second 7,000,000 none of the services was reachable (there is a lack of horizontal lines in that period). This fact also happens around seconds 1,769,200 and 5,847,200, although the concurrent unavailability happens during short periods and the Figure does not show it so obviously. Figure 10 shows it more clearly as it depicts the number of services available at each moment between the initial time and the 10,000,000th second. We can see also in this figure that at some times the number of available services decreases abruptly even if it does not reach the zero value (e.g., around 8,675,000th second). This fact reinforces the argument that some event happens in the system that affects the availability of a subset of services concurrently. These abrupt changes in the number of available servers allow the application to suspect that service availability is not independent and that the analysis of its availability block diagram produces too optimistic results, ergo the level of ignorance changes from the second to the first level, and methods described in Section 4.2 can then be applied.

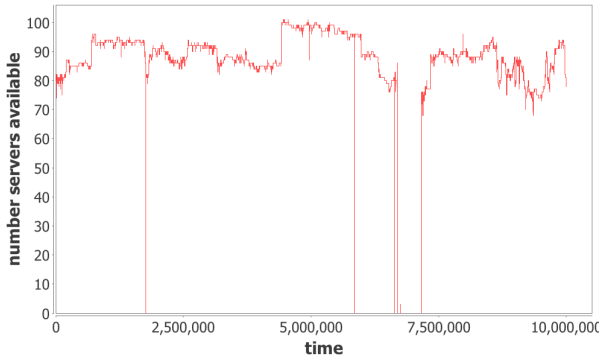

Figure 10: Number of servers available among the 250 first servers in `pl-app.avt` during the firsts 10,000,000 seconds

8. CONCLUSION AND FUTURE WORK

In this paper we have discussed how different types of uncertainties can affect the definition and evaluation of software models. In particular, exploiting works existing in the literature and belonging to the field of natural science, we have proposed here a definition of uncertainty that can be used in computer science research, together with a taxonomy of different types of modeling uncertainties, that considers their location, nature and level, with respect to quality evaluation. Focusing on self-adaptive software systems, we have then analyzed possible sources of uncertainties together with existing methods to reduce their impact in the model evaluation step. We have also shown, using a concrete example together with a set of realistic data logs, what it does mean taking into account the different types of uncertainties and their input on the final availability prediction/evaluation using state-of-the art methods.

This research can be extended along several directions. We intend to explore how to generalize methods that have been proven useful for taming a very concrete uncertainty to allow them to tame more uncertainties of the same type. Furthermore, we plan to explore possible dependencies among the different uncertainties and their relationships and impact on different quality attributes. Finally, we intend to analyze the different uncertainties in the context of real-world application scenarios, to assess possible correlation and identify best practice procedures.

Acknowledgments

The work has been partially supported by the FP7 European project Seaclouds. The authors are grateful to Vincenzo Grassi for insightful discussions on this research.

9. REFERENCES

[1] P. G. Armour. The five orders of ignorance. *Commun. ACM*, 43(10):17–20, Oct. 2000.

[2] M. Bakkaloglu, J. J. Wylie, C. Wang, and G. R. Ganger. On correlated failures in survivable storage systems. Technical Report CMU-CS-02-129, Carnegie Mellon University, 2002.

[3] S. Balsamo, A. D. Marco, P. Inverardi, and M. Simeoni. Model-based performance prediction in software development: A survey. *IEEE Trans. Software Engineering*, 30(5):295–310, 2004.

[4] B. Beck and G. van Straten. *Uncertainty and forecasting of water quality*. Springer-Verlag, 1983.

[5] W. J. Bolosky, J. R. Douceur, D. Ely, and M. Theimer. Feasibility of a serverless distributed file system deployed on an existing set of desktop PCs. In *Proc. SIGMETRICS*, 2000.

[6] R. Calinescu, C. Ghezzi, M. Z. Kwiatkowska, and R. Mirandola. Self-adaptive software needs quantitative verification at runtime. *Commun. ACM*, 55(9):69–77, 2012.

[7] S.-W. Cheng and D. Garlan. Handling uncertainty in autonomic systems. In *Proc. of the Int. Workshop on Living with Uncertainties (IWLU'07)*, Atlanta, GA, USA, 2007.

[8] L. Cheung, L. Golubchik, N. Medvidovic, and G. Sukhatme. Identifying and addressing uncertainty in architecture-level software reliability modeling. In *Int. Parallel and Distributed Processing Symposium. IPDPS 2007*, pages 1–6, 2007.

[9] V. Cortellessa, A. D. Marco, and P. Inverardi. *Model-Based Software Performance Analysis*. Springer, 2011.

[10] A. Devaraj, K. Mishra, and K. S. Trivedi. Uncertainty propagation in analytic availability models. In *Proc. of the Symposium on Reliable Distributed Systems*, SRDS '10, pages 121–130, Washington, DC, USA, 2010. IEEE Computer Society.

[11] N. Esfahani, E. Kouroshfar, and S. Malek. Taming uncertainty in self-adaptive software. In *Proc. of ESEC/FSE '11*, pages 234–244, New York, NY, USA, 2011. ACM.

[12] N. Esfahani and S. Malek. Uncertainty in self-adaptive software systems. In *Software Engineering for Self-Adaptive Systems II*, volume 7475 of *LNCS*, pages 214–238. Springer Berlin Heidelberg, 2013.

[13] N. Esfahani, S. Malek, and K. Razavi. Guidearch: guiding the exploration of architectural solution space under uncertainty. In *Proc. of the International Conference on Software Engineering*, ICSE '13, pages 43–52, Piscataway, NJ, USA, 2013. IEEE Press.

[14] N. Esfahani, K. Razavi, and S. Malek. Dealing with uncertainty in early software architecture. In *Proc. International Symposium on the Foundations of Software Engineering*, FSE '12, pages 21:1–21:4, New York, NY, USA, 2012. ACM.

[15] L. Fiondella and S. Gokhale. Software reliability with architectural uncertainties. In *Int. Parallel and Distributed Processing Symposium, IPDPS 2008*, pages 1–5, 2008.

[16] S. Funtowicz and J. Ravetz. *Uncertainty and Quality in Science for Policy*. Springer, 1990.

[17] D. Garlan. Software engineering in an uncertain world. In *Future of Software Engineering Research workshop*, FoSER '10, pages 125–128, New York, NY, USA, 2010. ACM.

[18] C. F. Gauss, C. H. Davis, and M. of America Project. *Theory of the motion of the heavenly bodies moving about the sun in conic sections*. Boston,Little, Brown and company, 1809. http://www.biodiversitylibrary.org/bibliography/19023.

[19] K. Goseva-Popstojanova and S. Kamavaram. Assessing uncertainty in reliability of component-based software systems. In *Proc. of the 14th International Symposium on Software Reliability Engineering*, ISSRE '03, pages 307–, Washington, DC, USA, 2003. IEEE Computer Society.

[20] K. Goseva-Popstojanova and S. Kamavaram. Software reliability estimation under certainty: generalization of the method of moments. In *Proc. of International Symposium on High Assurance Systems Engineering*, pages 209–218, 2004.

[21] S. Guha, N. Daswani, and R. Jain. An Experimental Study of the Skype Peer-to-Peer VoIP System. In *Proc. of the International Workshop on Peer-to-Peer Systems (IPTPS '06)*, Santa Barbara, CA, 2006.

[22] J. C. Helton, J. D. Johnson, W. Oberkampf, and C. J. Sallaberry. Representation of analysis results involving aleatory and epistemic uncertainty. *Int. J. General Systems*, (6):605–646, 2010.

[23] J. B. Kadane and N. A. Lazar. Methods and criteria for model selection. *Journal of the American Statistical Association*, 99(465):279–290, 2004.

[24] J. Kephart and D. Chess. The vision of autonomic computing. *Computer*, 36(1):41–50, 2003.

[25] I. Meedeniya, A. Aleti, and L. Grunske. Architecture-driven reliability optimization with uncertain model parameters. *J. of Systems and Software*, 85(10):2340–2355, Oct. 2012.

[26] I. Meedeniya, I. Moser, A. Aleti, and L. Grunske. Architecture-based reliability evaluation under uncertainty. In *Proc. of the international conference on Quality of Software Architectures*, QoSA'11, pages 85–94, New York, NY, USA, 2011. ACM.

[27] J. Pang, J. Hendricks, A. Akella, B. Maggs, R. D. Prisco, and S. Seshan. Availability, usage, and deployment characteristics of the domain name system. In *Proc. IMC*, 2004.

[28] J. C. Refsgaard, J. P. van der Sluijs, J. Brown, and P. van der Keur. A framework for dealing with uncertainty due to model structure error. *Advances in Water Resources*, 29(11):1586 – 1597, 2006.

[29] J. C. Refsgaard, J. P. van der Sluijs, A. L. Højberg, and P. A. Vanrolleghem. Uncertainty in the environmental modelling process - a framework and guidance. *Environ. Model. Softw.*, 22(11):1543–1556, Nov. 2007.

[30] J. Stribling. Planetlab all pairs ping. http://infospect.planet-lab.org/pings.

[31] M. Strong, J. E. Oakley, and J. Chilcott. Managing structural uncertainty in health economic decision models: a discrepancy approach. *Journal of the Royal Statistical Society: Series C (Applied Statistics)*, 61(1):25–45, 2012.

[32] C. Trubiani, I. Meedeniya, V. Cortellessa, A. Aleti, and L. Grunske. Model-based performance analysis of software architectures under uncertainty. In *Proc. of the international conference on Quality of Software Architectures*, QoSA '13, pages 69–78, New York, NY, USA, 2013. ACM.

[33] W. Walker, P. HarremoŚs, J. Romans, J. van der Sluus, M. van Asselt, P. Janssen, and M. Krauss. Defining uncertainty. a conceptual basis for uncertainty management in model-based decision support. *Integrated Assessment*, 4(1):5–17, 2003.

[34] J. Whittle, P. Sawyer, N. Bencomo, B. H. C. Cheng, and J.-M. Bruel. Relax: A language to address uncertainty in self-adaptive systems requirement. *Requir. Eng.*, 15(2):177–196, June 2010.

[35] L. Yin, M. Smith, and K. Trivedi. Uncertainty analysis in reliability modeling. In *Proc. of Reliability and Maintainability Symposium, 2001*, pages 229–234, 2001.