Probabilistic Automata for Architecture-Based Reliability Assessment

Ivo Krka, Leana Golubchik, Nenad Medvidovic
Computer Science Department
University of Southern California
Los Angeles, CA 90089-0781, USA
{krka,leana,neno}@usc.edu

ABSTRACT

Non-functional properties, such as reliability and performance, should be assessed as early as possible in a system's life cycle for cost effectiveness reasons. Hence, several software architecture-based reliability assessment techniques have been proposed. These techniques quantitatively analyze a system's or a component's behavior, which is typically represented using probabilistic generative automata. However, we demonstrate that generative automata do not appropriately capture information available in an operational profile. Furthermore, we overview other existing probabilistic automata formalisms; we identify their features and shortcomings when capturing an operational profile. As a way to circumvent the identified deficiencies, we introduce probabilistic component interface protocols, a new probabilistic automata formalism that supports intuitive and direct mapping of an operational profile. Finally, we discuss how to derive the analysis-oriented generative models from the probabilistic component interface protocols.

Categories and Subject Descriptors

D.2 [**Software Engineering**]: Requirements/Specifications; D.2 [**Software Engineering**]: Software/Program Verification

General Terms

Reliability

Keywords

probabilistic models, operational profile, reliability

1. INTRODUCTION

Software systems are developed to achieve a set of system users' goals. Although the functionality of a software system (i.e., what a system does) is important for its adoption, the critical prerequisite for a system's success is the exhibited non-functional quality (i.e., how a system does). The non-functional quality is typically measured in terms of the different non-functional properties (NFPs) such as performance, security, usability, and reliability. Due to their criticality, NFPs should not be considered as an afterthought during system development. Instead, NFPs should be assessed as early and as continuously as possible throughout a system's life cycle. A system's architectural design should ideally guarantee, or at least provide support for, the desirable NFP levels. Thus, researchers have proposed techniques for reliability assessment [6, 8] based on a system's software architecture. In this paper, we discuss problematic assumptions made by existing architecture-based reliability assessment techniques regarding the modeling granularity and format of the system usage information.

The currently adopted process for architecture-based reliability assessment is depicted at the top of Figure 1. Quantitative methods for reliability assessment depend on the availability of system usage information — i.e., a system's operational profile [12]. The operational profile information is combined with the non-probabilistic behavior models (e.g., automata-based specifications) in order to obtain probabilistic models suitable for analysis. The constructed models are then analyzed using the existing reliability analysis techniques [6, 8].

The existing architecture-based reliability analysis techniques have several shortcomings (for an overview see [10]). In this effort, we study the obstacles that arise when deriving probabilistic annotations from an operational profile (*Annotation* in Figure 1) in the form imposed by the analysis techniques (*Probabilistic Models* in Figure 1).

Different types of challenges arise for system- and component-level analysis techniques. The system-level techniques (e.g., [2]) typically model a system with states, which represent components, and probabilistic transitions that aggregate inter-component transfer-of-control. For example, Figure 3 depicts a system-level probabilistic model of a mobile robotics application *RoboArch* [10], which consists of four software components. However, modern software is rarely sequential; software components run concurrently, and the exact meaning of inter-component transition probabilities in such a setting is not intuitive and possibly misleading. In the *RoboArch* system, for example, an engineer might not possess an understanding of the transfer-of-control probability between *GUI* and *Follower* when these two components are running concurrently.

Representative challenges for component-level techniques that require an annotated occurrence probability for each transition of a component-level automaton (i.e., creating a generative probabilistic automaton [3]) are: (1) determin-

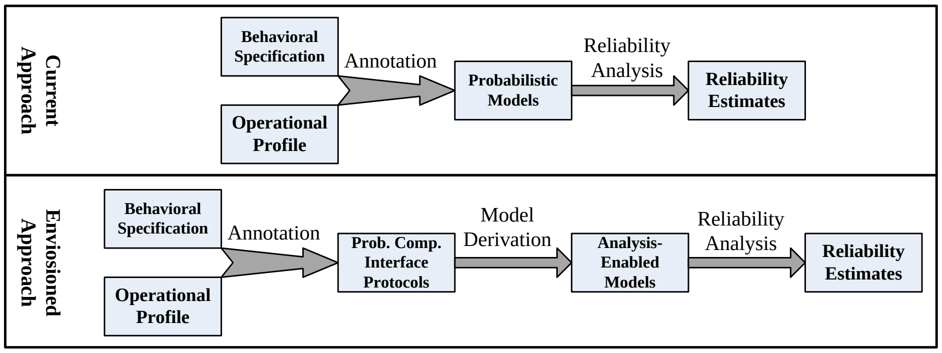

Figure 1: The currently followed reliability assessment process and our envisioned process.

ing probabilities of transitions that correspond to locally uncontrolled events, and (2) assuring consistency between transition probabilities across dependent components. For example, a non-probabilistic automaton for *RoboArch*'s *Follower* (depicted in Figure 5) has to be fully annotated with generative transition probabilities in order to facilitate reliability analysis (e.g., [1]). *Follower*'s functionality is to follow a line on the floor, and to accept *GUI* commands that change its movement. Hence, the probability values of transitions on *correctAngle* and *end*, which are controlled by *GUI*, would have to be defined in both *Follower* and *GUI* models. This results in a challenge of having to maintain consistent probability ratio of *correctAngle* and *end* transitions across *Follower*'s states, as well as across *Follower* and *GUI* models.

The foundational cause of the described challenges is the fact that existing reliability analysis techniques work on generative probabilistic automata. A practice in which an engineer manually combines an operational profile and a non-probabilistic behavioral specification to directly produce an analysis-enabled generative model is tedious, non-intuitive, and error-prone. We posit that the operational profile and non-probabilistic specifications can be more intuitively combined. Specifically, the controlled (output) actions should be handled differently than the actions controlled by other components or the environment. Additionally, only the controlled actions should be modeled in a generative fashion because probabilities of uncontrolled actions innately depend on the behavior of external elements.

To this end, we propose *probabilistic component interface protocol*, which is a novel probabilistic formalism that combines features of probabilistic IO automata [19] and probabilistic interface automata [13]. The probabilistic component interface protocols can be automatically transformed into an analysis-enabled generative. This new envisioned process for architecture-based reliability assessment is depicted at the bottom of Figure 1; *Analysis-Enabled Models* are identical to *Probabilistic Models* in the old process.

The primary contributions of this paper can be summarized as follows:

1. We overview the existing probabilistic automata formalisms. We discuss their features, and their shortcomings when modeling software behavior (Section 2).
2. We critically assess the way existing reliability assessment techniques try to capture an operational profile (Section 3).
3. We reason about an appropriate mapping of a system's operational profile and the available non-probabilistic behavioral specifications (Section 4).
4. We define probabilistic component interface protocols as a way to intuitively and directly merge an operational profile and non-probabilistic models (Section 4).

2. PROBABILISTIC BEHAVIORAL MODELS

Probabilistic behavior of a software system is often modeled and analyzed with discrete-time Markov chains [18], such as M_1 depicted in Figure 2. A DTMC consists of a set of states; each state has corresponding probabilities of transitioning to other states. DTMCs embody a basic way of modeling, and reasoning about probabilistic behavior; further demands, including the need to more faithfully represent software systems, enabled a proliferation of probabilistic automata formalisms [17]. In this section, we first describe the two main paradigms for modeling probabilistic behavior: the *generative* and the *reactive* paradigm. We then present two probabilistic modeling formalisms that differentiate between controlled and uncontrolled actions: the *probabilistic IO automata* [19] and the *probabilistic interface automata* [13]. We are able to demonstrate, with these formalisms, a wide spectrum of approaches to modeling probabilistic behavior. Keeping in mind the subsequent application to reliability analysis, we elucidate the bearing of different modeling formalisms when capturing a software system's operational profile. Additionally, the discussion in this section (1) lays the foundations of the critical assessment of reliability analysis approaches in Section 3, and (2) motivates a novel modeling formalism (Section 4).

2.1 Generative and Reactive Paradigms

Definition 1 (Generative probabilistic automata). *A generative probabilistic automaton M is a 4-tuple (S, s_1, A, T), where S is a set of states, s_1 is an initial state, A is a set of actions, and T is a transition function* $T : S \rightarrow \Pr((A \times S) \cup \{\varnothing\})$.

The notion of generative probabilistic models emerged from the need to model systems that *generate* actions – i.e., systems with locally controlled behavior. Generative models are similar to DTMCs in that states in a generative model have a probabilistic distribution over all of the outgoing transitions. The sum of transition probabilities going out of a state must be 1.0, including the probability of termination. The main distinction compared to basic DTMCs is that generative automata have transitions labeled with actions. An example generative automaton M_2 (depicted in

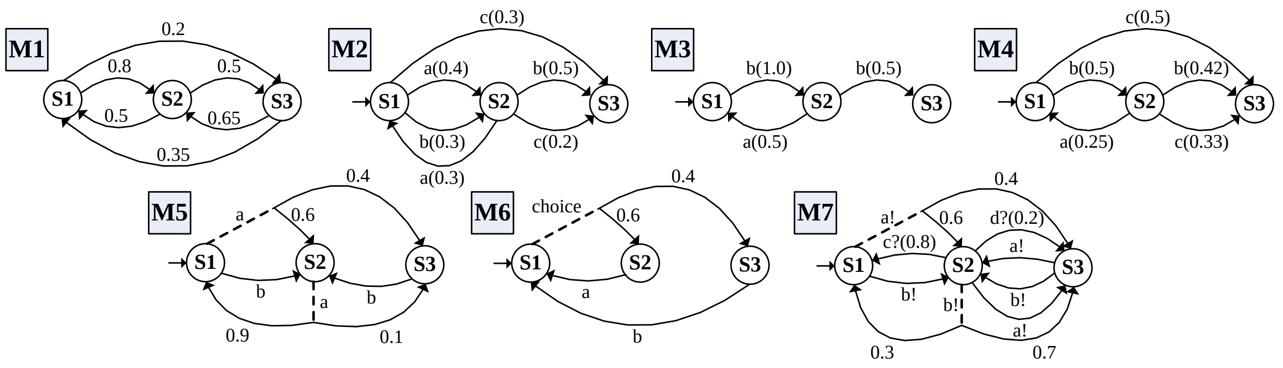

Figure 2: Example probabilistic automata modeled using different formalisms. Input actions for probabilistic IO automata are marked with '!', while output actions are marked with '?' sign.

Figure 2) has transitions annotated with occurrence probabilities, and state s_3 as a terminating state; M_2 also conveys that the transition $s_1 \xrightarrow{a} s_2$ will always be traversed with a 40% chance starting from s_1. When a system is modeled using the generative paradigm, the effects of the environment have to be built into the model. This increases the required modeling effort when analyzing multiple possible environments.

Modern software systems are composed of multiple components that synchronize on shared actions. Hence, the components should ideally be modeled separately, and the system-level model obtained by composing the component models. D'Argenio et al. [3] studied parallel composition in a generative setting. The proposed variants of the composition operator [3] do not preserve the described intuition behind the generative paradigm. We thus posit that the generative paradigm alone is insufficient for modeling individual system components. We illustrate this on an automaton M_4 (Figure 2), which is obtained by composing M_2 and M_3 using CSP-style parallel composition [3]. The composition M_4 does not have a transition on action a from state s_1, although a should occur with probability 0.4 according to M_2. This phenomenon is due to the fact that multiple component-level automata (in this case M_2 and M_3) model control of a shared action (a). This results in a situation where one composed state ($M_2.s_1$) has a shared action (a) enabled, while the other composed state ($M_3.s_1$) does not. To deal with such situations, the generative composition includes non-intuitive probability normalization.

In summary, purely generative models are convenient for modeling behavior of a system for which (1) a complete understanding of the behavior of both the modeled system and its environment is present; and (2) the system controls every modeled action. The reactive paradigm overcomes some of the generative paradigm shortcomings.

Definition 2 (Reactive probabilistic automata). *A reactive probabilistic automaton M is a 4-tuple (S, s_1, A, T), where S is a set of states, s_1 is an initial state, A is a set of actions, and T is a transition function $T : S \rightarrow A \times (\Pr(S \cup \{\varnothing\}))$.*

Reactive probabilistic models, proposed by Larsen [11], were conceived to represent behavior of systems that *react* to external actions. Informally, reactive models convey probabilistic information about the possible successive state after some external action happens. For example, reactive automaton M_5 (Figure 2) has two possible actions in state s_1. When action a occurs, the next state will be s_2 with probability 0.6, or state s_3 otherwise. The model does not ascertain the relative occurrence rate between actions a and b from s_1 — the occurrence of these actions is dependent on the environment. Consequently, the problems with the parallel composition of generative automata do not appear in the case of reactive automata.

The shortcoming of the reactive paradigm is the lack of native support for capturing generative information about the controlled actions. The generative behavior can be described only implicitly; for example, action *choice* simulates generative behavior in model M_6, where a is generated more often than b (Figure 2). However, this way of modeling generative behavior becomes problematic during parallel composition because the standard composition operator [17] does not treat *choice* as a special case. Such problems are resolved by probabilistic IO automata.

2.2 Probabilistic IO Automata and Probabilistic Interface Automata

Definition 3 (Probabilistic IO automata). *A probabilistic IO automaton M is a 5-tuple (S, s_1, A, T, δ), where S is a set of states, s_1 is an initial state, A is a set of actions divided into input and output actions $A = A_{in} \cup A_{out}$, T is a transition function $T : S \rightarrow (\Pr(S))^{A_{in}} \cup \Pr((A_{out} \times S) \cup \{\varnothing\})$, and δ is a delay function such that $\delta : S \rightarrow \mathbb{R}_{\geq 0}$ with $\delta(s) > 0$ if there exists a transition from s labeled with some output action.*

The probabilistic IO automata, proposed by Wu et al. [19], explicitly distinguish between input and output actions. The transitions labeled with output actions are modeled in a generative fashion. Conversely, the transitions on input actions are modeled in a reactive fashion, which is an intuitive choice because input actions are controlled by the environment. Hence, the probabilistic IO automata seem like a natural choice for modeling software components that have both provided and required interfaces. In order to assure the correctness of parallel composition, probabilistic IO automata are input enabled, which means that a reactive transition is defined for each of the input actions in each state. For example, the automaton M_7 (Figure 2) has input actions a and b enabled in each state to assure that M_7 can always accept output actions a and b generated by other automata. Moreover, each state has an assigned delay rate, which is used when composing the generative behavior from multiple automata. Intuitively, a delay rate specifies the rate at which actions are generated from a state.

The preceding discussion suggests that probabilistic IO automata provide a "cleaner" abstraction for probabilistically modeling software components. However, devising a probabilistic IO automaton for reliability analysis requires specifying probabilistic transitions on input actions, even when they are not present in the non-probabilistic counterpart. Furthermore, an engineer working with an automaton has to assign a delay to each state. These laborious tasks should be avoided in order to make formal specifications more accessible to engineers without prior training in probabilistic models. Some of these issues are alleviated with probabilistic interface automata.

Definition 4 (Probabilistic interface automata). *A probabilistic interface automaton M is a 6-tuple $(S, s_1, A_{in}, A_{out}, A_{hid}, T)$, where S is a set of states, s_1 is an initial state, A is a set of actions divided into input, output, and internal (hidden) actions $A = A_{in} \cup A_{out} \cup A_{hid}$, T is a reactive transition function, and M is an interface automaton when the transition probabilities are removed.*

Probabilistic interface automata [13], distinguish between input and output actions, while also explicitly modeling internal actions. The main distinctions between probabilistic interface automata and probabilistic IO automata are that interface automata (1) do not demand input-enabledness; (2) do not utilize the generative paradigm for modeling controlled behavior; and (3) do not have state delays. The parallel composition of probabilistic interface automata is derived from the composition operator of non-probabilistic interface automata [4], with the addition of reactive probabilistic reasoning. In addition, the composition operator supports discovery of illegal states, which is particularly interesting when aiming to build a reliable system. The generative behavior can be simulated with a special *choice* action, as discussed for reactive automata. However, this technique of simulating generative behavior fails during model composition, which is not desirable when reasoning about systems with multiple software components.

3. PROBABILISTIC BEHAVIOR IN RELIABILITY ASSESSMENT TECHNIQUES

Reliability is considered to be a crucial non-functional property, hence the emergence of techniques for analyzing reliability of components and systems based on their architectural specification (surveyed in [6, 8]). The proposed techniques typically analyze DTMCs that capture a component's or a system's probabilistic behavior. The DTMCs are derived directly from generative probabilistic models that are in turn obtained by manually annotating non-probabilistic behavioral models. The probabilistic annotations are assumed to be straightforwardly derivable from an operational profile [12]; in the previous section, we demonstrated that some operational profile information cannot be appropriately captured using the generative paradigm.

The focus in Section 2 was on introduction of general-purpose formalisms for modeling probabilistic behavior. In this section, we take a narrower viewpoint and discuss how probabilistic behavior is modeled in existing architecture-based reliability assessment techniques. In our discussion, we utilize four techniques [1, 2, 13, 16] that are representative of the commonly applied modeling approaches. For each technique, we explain the operational profile information that is assumed to be available by describing the proposed

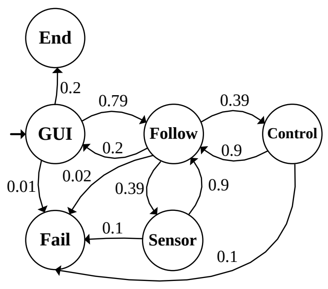

Figure 3: The *RoboArch* system modeled according to Cheung's approach [2].

analysis-enabled models. We challenge the existing assumptions regarding the operational profile, and discuss the difficulties that arise when combining operational profile information commonly found in practice with non-probabilistic models. Notably, a large number of the issues we describe in this section are specific instances of the issues that were already discussed in a more general context in Section 2. The discussion in this section also motivates the need for a new formalism that appropriately captures operational profile information.

3.1 System-Level Techniques

Cheung [2] proposed one of the first techniques for analyzing reliability based on a system's architectural configuration. Several subsequent techniques extend Cheung's technique (e.g., [15]). In Cheung's technique, the utilized architectural model is a program's control graph, where the nodes represent the system processes (or components), and the probabilistic transitions represent transfer-of-control between the processes. Hence, an operational profile is expected to contain quantitative transfer-of-control information. Each modeled process has an assigned reliability value, which is captured with supplementary transitions to a failure state. The probabilistic model obtained in this manner can be mapped to a DTMC. Hence, calculating the system-wide reliability in this setting is equivalent to computing the steady-state probabilities of a DTMC.

To illustrate this approach, consider a single-robot variant of *RoboArch* [10], which is depicted in Figure 3. The *RoboArch* architecture consists of *Controller*, *Sensor*, *Follower*, and *GUI* components. The first three components are deployed onto a physical robot, which can move through a physical environment using several navigation modes (line following, color following, and wall following). An operator issues commands to the robot via a *GUI*. The navigation mode can be dynamically changed by sending messages from *GUI* to *Follower*, while *Follower* sends status updates to *GUI*. Once the robot accomplishes a goal (e.g., reaching a location), *GUI* terminates the application. Several difficulties arise when deriving transition probabilities from an operational profile:

1. the inter-component transfer-of-control probabilities are not intuitive because components have multiple operations (e.g., *GUI* can invoke different *Controller*'s operations depending on the navigation mode);
2. the intuition behind transition probabilities is not intuitive because components run in parallel (e.g., *GUI* and *Follower* are concurrent), while Cheung's model is suitable for describing sequential systems; and

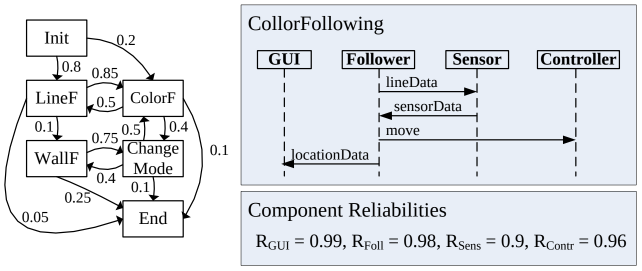

Figure 4: A part of the *RoboArch* system specification required for Rodrigues et al.'s analysis technique [16].

3. the failure probabilities largely depend on the way component operations are used, which is only summarized in Cheung's model (e.g., a line following mode may be more reliable than the color following one).

Some of these shortcomings can be overcome by taking fine-grained component models, composing them together, and then annotating the resulting model with transition probabilities. However, Cheung essentially applies the generative paradigm, and composing component-level models can result in a non-intuitive composition (recall Section 2).

A more intuitive way of mapping a system-level operational profile to non-probabilistic models was proposed by Rodrigues et al. [16]. In this approach, an engineer needs to specify (1) the system behavior in terms of important execution scenarios, (2) the transition probabilities between the different scenarios, and (3) the components' failure probabilities. Figure 4 depicts a part of the required specification for *RoboArch*. The system behavior is described with six scenarios, where an example *CollorFollowing* scenario is depicted on the right-hand side. The system-level generative model conveys that *LineFollowing* is the initial navigation mode in 80% of cases, while *ColorFollowing* is initially used in 20% of cases. Finally, the specification from Figure 4 provides reliability values of system components.

The scenarios and inter-scenario transition probabilities are used to generate component-level generative models. To calculate the system-level reliability, the component-level models are composed into a system-level model using the generative parallel composition operator [3]. However, the generative paradigm can result in non-intuitive models during parallel composition. Moreover, the resulting system-level analysis-enabled model suffers from the state-explosion issues, which limits this techniques to smaller sized systems. Lastly, it is often difficult to comprehensively describe a system's behavior using only scenarios.

3.2 Component-level Techniques

The described system-level techniques are limited by the tradeoff between scalability and the level of detail about component behavior. Including fine-grain component models and composing them into a system-level model can increase the analysis precision, but suffers from scaling issues. Conversely, analyzing a coarse-grain system-level model that abstracts system components to a single reliability value, is problematic because a component's reliability depends on the way it is used. Therefore, Cheung et al. [1] separately analyze the reliability of individual components.

Cheung et al. [1] analyze a generative probabilistic model that is derived from a component's non-probabilistic dynamic behavior model. The probabilistic annotations are derived from an operational profile that is available as a combination of domain knowledge, requirements documents, simulation, and functionally similar components' usage logs. The authors acknowledge the difficulty of obtaining a complete set of annotations for a generative model, and leverage hidden Markov models [14] to determine the unknown probabilities. However, this technique does not consider the consistency of probabilistic annotations across multiple components in a system, which goes back to the general drawbacks of the generative paradigm.

Pavese et al. [13] adopt a converse viewpoint by exclusively considering the reliability impacts of a system's probabilistic environment (the environment is represented with a probabilistic interface automaton). A non-probabilistic system-level model is composed with the environment automaton in order to obtain a model that captures the feasible behavior (the non-feasible behavior is removed during composition). The produced behavioral model is essentially reactive (as opposed to generative); hence, instead of DTMC analysis, reliability estimates are obtained through probabilistic model checking [7]. The main shortcoming of this technique is that it does not consider the often available information about a system's internal behavior (e.g., previous version's usage logs).

4. APPROPRIATELY CAPTURING AN OPERATIONAL PROFILE

In this section, we discuss *what* information about system usage can be available during architectural design, and *how* it should be interpreted in terms of probabilistic transitions. We first consider a system-level approach, which assumes that probabilistic information is available at the level of the whole system, and can be directly mapped to generative transitions. We explain the flaws of this approach, and question its applicability during architectural design. We then enumerate operational profile information that is modular, and can be obtained during architectural design. The enumerated operational profile information cannot be directly mapped to the models expected by the existing reliability assessment techniques (Section 3). More generally, it turns out that none of the existing probabilistic modeling formalisms can successfully capture the identified operational profile information. To this end, we propose a formalism, probabilistic component interface protocols (PCIP), which combines features of probabilistic IO automata and probabilistic interface automata. The new formalism supports intuitive and direct capturing of operational profile information. In addition, PCIPs provide a modular way to model a system's probabilistic behavior – changes to an operational profile are reflected in a small number of places. Therefore, PCIPs have the potential to be used when there is uncertainty in an operational profile or it is constantly changing. Finally, we reason about computationally inexpensive ways to derive analysis-enabled models from PCIPs.

4.1 The System-Level Take

Practically available information about a system's operational profile can widely vary in its granularity and level of detail. One extreme is the case in which comprehensive system-level information about operation invocations is available. For example, a simplified system-level model for

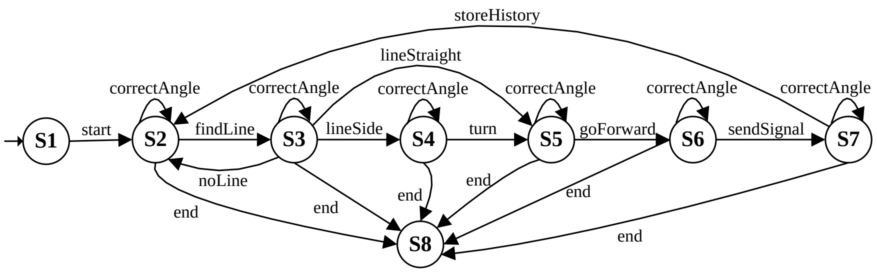

Figure 5: A simplified system-level model of the *RoboArch*'s single-robot variant.

a single-robot *RoboArch* system variant is depicted in Figure 5. However, annotating each transition in a system-level model is feasible only when the annotations are automatically obtainable by monitoring an existing system. The number of states in a system-level model rapidly increases with the system size; this prevents manual annotation of the model thus hampering the applicability of a system-level approach during architectural design. For example, a system-level model of an advanced *RoboArch* variant with three robots that follow one another has over 12,000 states – manually annotating such a model is not a feasible task.

4.2 Probabilistic Component Interface Protocols

An important conclusion from the preceding discussion is that operational profile information will seldom be directly available, during architectural design, at the level of the system as a whole. Contrarily, the design-time operational profile will typically refer to limited and modular aspects of the system (e.g., a particular component). Thus, we define a reasonable set of expected information about the operational profile during architectural design:

1. A probabilistic specification of an environment's generative behavior. For example, such specification can state that a user will invoke *correctAngle* four times more often than *end*. Information about an environment's generative behavior are found in specifications of use-case frequencies, and other requirements documents.
2. A probabilistic specification of the environment's reactive behavior, which can be obtained from domain knowledge. The specification of reactive behavior helps to probabilistically resolve the non-determinism in environment's behavior. For example, a more advanced version of the *RoboArch* system can generate *lost* signal; the environment reacts probabilistically, and starts either the *Abort Mission* or *Panic Mode* sequence.
3. A probabilistic specification of system components' generative behavior. This type of information is obtainable from the requirements and design documents, previous component versions, prototypes, or simulation. For example, a requirement can discover that *Follower* should store its position every other sensing cycle.
4. A probabilistic specification of system components' reactive behavior (similar to 1-3).
5. Finally, to facilitate parallel composition of generative transitions, we expect an engineer to provide a rough estimate of components' delay rates. A component's delay rate is an indicator of that component's performance. For example, a domain expert can ascertain that *Sensor* generates actions 4 times more frequently than *GUI*. Ideally, the delay rates are assigned to each component state, but this can often be too high of an overhead for an engineer (recall Section 2).

The five types of information should be derivable from the different information sources that are available during architectural design [1, 10]. Furthermore, this information is needed to comprehensively, but modularly, describe the probabilistic behavior of a software system — the expected information covers each component's generative and reactive behavior aspects. Additionally, the delay rates are needed to enable modular specification by normalizing probabilities of generative transitions during composition.

With the assumption that component- and environment-level non-probabilistic models are available, the expected operational profile can be intuitively and directly interpreted in terms of probabilistic transitions. However, the formalisms presented in Section 2 cannot be readily used to capture the expected operational profile due to their innate shortcomings. To reiterate, the probabilistic IO automata are input enabled; the probabilistic interface automata do not explicitly model generative behavior. However, the features of these formalisms can be combined into a new formalism, *probabilistic component interface protocols*, which supports an intuitive mapping of an operational profile.

Definition 5 (Prob. Comp. Interface Protocols). *A probabilistic component interface protocol M is a 5-tuple (S, s_1, A, T, δ), where S is a set of states, s_1 is an initial state, A is a set of actions divided into input, internal, and output actions $A = A_{in} \cup A_{hid} \cup A_{out}$, T is a transition function $T : S \to A \times (\Pr(S \cup \{\emptyset\})) \cup \Pr(((A_{out} \cup A_{hid}) \times S) \cup \{\emptyset\})$, and δ is a delay value such that $\delta : S \to \mathbb{R}_{\geq 0}$ with $\delta(s) > 0$ if there exists a transition from s labeled with some output action.*

The proposed formalism has the following features:

1. Each state can have a set of generative transitions with cumulative probability of 1.0 (a feature of probabilistic IO automata). A generative transition can be defined only on internal and output actions in order to prevent unintuitive parallel composition. The generative information present in an operational profile can thus be directly represented.
2. Each state can have a set of reactive transitions labeled with input actions. Each reactive transition has an assigned probabilistic distribution (a feature of probabilistic interface automata). Importantly, the reactive transitions from a state do not have to be defined for each input action (unlike for IO automata), which reduces the burden on an engineer.
3. The states have assigned delay rates (a feature of probabilistic IO automata). However, an engineer is required to provide only a single value per component that is

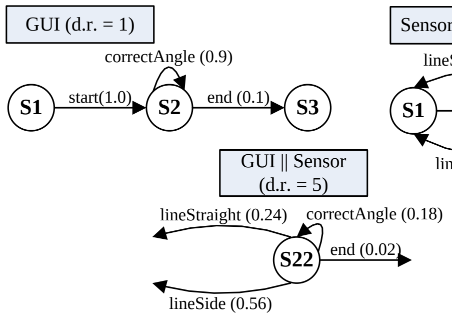

Figure 6: The PCIP specification of *GUI* and *Sensor*, with the excerpt from their parallel composition.

applied for each state. Hence, the corresponding operational profile information can be mapped to a probabilistic component interface protocol.

Figure 6 depicts two *RoboArch* components specified with the PCIP formalism. For example, *GUI* has only output actions, resulting in a model with exclusively generative transitions; *Sensor* has one reactive transition on *findLine*. Both *GUI* and *Sensor* have assigned delay rates; *Sensor*'s delay rate is four times larger than *GUI*'s.

The parallel composition of probabilistic component interface protocols is a combination of the composition operators on probabilistic IO automata, and interface automata. Informally, the composition operator on probabilistic component interface protocols synchronizes on matching input/output action combinations (similar to prob. IO automata). The generative transition probabilities from different components are normalized during composition with respect to the assigned delay rates. For example, the composition of states *GUI.*s_2 and *Sensor.*s_2 is illustrated in Figure 6. During the composition, the generative probabilities for *lineStraight*, *lineSide*, *correctAngle*, and *end* are normalized. Note, however, that the intuition behind *Sensor*'s generative transitions is preserved as the ratio between the transition probabilities on *lineStraight* and *lineSide* remained (0.7/0.3). The delay rate of the composition is the sum of *Sensor* and *GUI* delay rates.

Since probabilistic component interface protocols do not impose input-enabledness, a parallel composition operator must handle inconsistencies between composed model. We are referring to inconsistencies that can occur when composing automata with shared actions. For example, one automaton can generate a shared output action while the synchronizing automaton cannot accept a matching input action. The probabilistic interface automata specification [13] proclaims the inconsistent composite state as *illegal*; the goal is then to create an environment that avoids the illegal state. However, our aim is not to construct an environment that cannot reach an illegal state but to quantitatively analyze erroneous behavior. Hence, the described inconsistency is manifested in the composed model with a generative transition, labeled with the blocked action, to a *special error state*. For example, consider a composition of *Follower.*s_3 (depicted in Figure 7) with s_{22} from *GUI* and *Sensor*'s composition (Figure 6). Note that we do not depict the legal target states of the displayed transitions. *Follower.*s_3 is not *correctAngle*–enabled, and when *GUI* generates *correctAngle*, the system goes to an error state

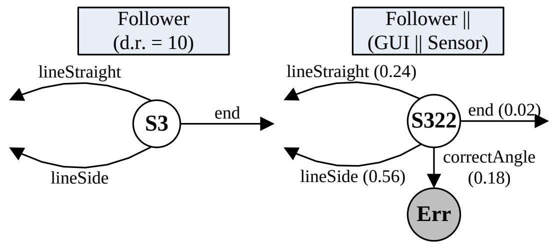

Figure 7: A state in *Follower*'s PCIP specification, and its problematic composition with a state in *GUI* || *Sensor*.

(Figure 7). Intuitively, further composition of an error state with other legal states produces an error state. The error state can have outgoing transitions, and we intend to explore possible PCIP refinements that enable modeling of recovery from an error state.

4.3 Further Considerations

Once an operational profile is captured using a probabilistic component interface protocol, we still need to devise a generative model for reliability analysis (Figure 1). In this section, we hypothesize about a strategy that would help to resolve this problem. The strategy is based on the composition of component-level and environment-level PCIPs into a system-level generative model. The system-level model can be analyzed using traditional DTMC analysis [18], whose complexity is $O(n^3)$ (n is the number of states). However, the resulting model can also be difficult to numerically solve due to state explosion; in such cases DTMC solution might have to be approximated using aggregation/disaggregation approaches [18]. The approximations attempt to partition the model into smaller DTMCs with a lower factor n. The obtained approximations would be used to determine the generative probabilities of reactive transitions in the component-level models; the fully generative component models can be analyzed using existing techniques, and plugged into a system-level model.

As a supplement to DTMC approximation, we propose to simulate parts of the system. The PCIPs are amenable to simulation as they precisely capture the manner in which actions are generated and how they are consumed (or not consumed, resulting in transitions to an error state). We expect that architectural simulation tools, like XTEAM [5], can supply the desired simulation capabilities.

The operational profile information that we assume available (Section 4.2) is often just a subset of the available information. The additional information can be checked for consistency on either the system-level model (e.g., occurrence rates of particular scenarios) or component-level models (e.g., occurrence probabilities of uncontrolled actions), depending on the information granularity.

5. CONCLUSIONS

In this paper, we first provided an overview of existing probabilistic automata formalisms, and critically assessed four relevant reliability analysis techniques with particular focus on the way they model probabilistic behavior. Based on our experience, we defined operational profile information that comprehensively describes a system's probabilistic behavior in a modular manner. We then proposed probabilistic component interface protocols as a new formalism suitable for intuitively and directly capturing an operational profile through its support for modeling both generative and reactive behavior. Furthermore, probabilistic component interface protocols facilitate reliability analysis with (1) explicit support for discovery and modeling of error states, and (2) construction of probabilistic transitions to error states. The proposed formalism combines the features of probabilistic IO automata and probabilistic interface automata, while avoiding their shortcomings. Additionally, probabilistic component interface protocols help to avoid the lacking assumptions about an operational profile made by state-of-the-art reliability assessment approaches. As a part of our future work, we intend to formally define different operators and explore properties of probabilistic component interface protocols. We also plan to assess the utility of probabilistic component interface protocols for architecture-based reliability analysis. Additionally, we will explore the feasibility of parameterizing PCIP transitions in order to incorporate information about data parameters in a manner similar to [9].

Acknowledgments

This work was sponsored in part by the National Science Foundation, award numbers 0720612 and 0820170. We also want to thank Leslie Cheung for fruitful discussions about the different topics discussed in this paper.

6. REFERENCES

[1] L. Cheung et al. Early prediction of software component reliability. In *Proc. of ICSE*, 2008.

[2] R. C. Cheung. A user-oriented software reliability model. *IEEE TSE*, 6, 1980.

[3] P. D'Argenio et al. On generative parallel composition. *Electron. Notes Theor. Comp. Sci.*, 22, 1999.

[4] L. De Alfaro and T. Henzinger. Interface automata. In *Proc. of ESEC/FSE*, 2001.

[5] G. Edwards et al. Scenario-Driven Dynamic Analysis of Distributed Architectures. In *Proc. of FASE*, 2007.

[6] S. Gokhale. Architecture-based software reliability analysis: Overview and limitations. *IEEE Trans. on Dependable and Secure Comp.*, 4(1), 2007.

[7] A. Hinton et al. PRISM: A tool for automatic verification of probabilistic systems. In *Proc. of TACAS*, 2006.

[8] A. Immonen and E. Niemelä. Survey of reliability and availability prediction methods from the viewpoint of software architecture. *Soft. and Sys. Modeling*, 7(1), 2008.

[9] H. Koziolek and F. Brosch. Parameter dependencies for component reliability specifications. *Electron. Notes Theor. Comput. Sci.*, 253(1), 2009.

[10] I. Krka et al. A comprehensive exploration of challenges in Architecture-Based reliability estimation. In *Arch. Dependable Sys. VI*. 2009.

[11] K. Larsen and A. Skou. Bisimulation through probabilistic testing. *Information and Computation*, 94, 1991.

[12] J. D. Musa. Operational profiles in software-reliability engineering. *IEEE Software*, 10(2), 1993.

[13] E. Pavese et al. Probabilistic environments in the quantitative analysis of (non-probabilistic) behaviour models. In *Proc. of ESEC/FSE*, 2009.

[14] L. Rabiner. A tutorial on hidden Markov models and selected applications in speech recognition. *Readings in speech recognition*, 53(3), 1990.

[15] R. Reussner et al. Reliability prediction for component-based software architectures. *JSS*, 66(3), 2003.

[16] G. Rodrigues et al. Using scenarios to predict the reliability of concurreny component-based software systems. In *Proc. of FASE*, 2005.

[17] A. Sokolova and E. De Vink. Probabilistic automata: system types, parallel composition and comparison. In *Validation of Stochastic Systems*, 2004.

[18] W. Stewart. *Numerical solution of Markov chains.* CRC, 1991.

[19] S. Wu et al. Composition and behaviors of probabilistic I/O automata. *Theor. Comp. Sci.*, 176(1-2), 1997.