Carnegie Mellon University
Research Showcase @ CMU

Institute for Software Research | School of Computer Science

9-2014

Impact Models for Architecture-Based Self-Adaptive Systems

Javier Camara
Carnegie Mellon University

Antonia Lopes
Universidade de Lisboa

David Garlan
Carnegie Mellon University

Bradley Schmerl
Carnegie Mellon University

Follow this and additional works at: http://repository.cmu.edu/isr

 Part of the Software Engineering Commons

Published In
Lecture Notes in Computer Science, 8997, 89-107.

This Conference Proceeding is brought to you for free and open access by the School of Computer Science at Research Showcase @ CMU. It has been accepted for inclusion in Institute for Software Research by an authorized administrator of Research Showcase @ CMU. For more information, please contact research-showcase@andrew.cmu.edu.

Impact Models for Architecture-Based Self-Adaptive Systems

Javier Cámara[1], Antónia Lopes[2], David Garlan[1], and Bradley Schmerl[1]

[1]Institute for Software Research, Carnegie Mellon University, Pittsburgh, PA 15213, USA
{jcmoreno,garlan,schmerl}@cs.cmu.edu
[2]Dep. of Informatics, Faculty of Sciences, University of Lisbon, Portugal
mal@di.fc.ul.pt

Abstract. Self-adaptive systems have the ability to adapt their behavior to dynamic operation conditions. In reaction to changes in the environment, these systems determine the appropriate corrective actions based in part on information about which action will have the best impact on the system. Existing models used to describe the impact of adaptations are either unable to capture the underlying uncertainty and variability of such dynamic environments, or are not compositional and described at a level of abstraction too low to scale in terms of specification effort required for non-trivial systems. In this paper, we address these shortcomings by describing an approach to the specification of impact models based on architectural system descriptions, which at the same time allows us to represent both variability and uncertainty in the outcome of adaptations, hence improving the selection of the best corrective action. The core of our approach is an impact model language equipped with a formal semantics defined in terms of Discrete Time Markov Chains. To validate our approach, we show how employing our language can improve the accuracy of predictions used for decision-making in the Rainbow framework for architecture-based self-adaptation.

1 Introduction

Self-adaptive systems have the ability to autonomously change their behavior in response to changes in their operating conditions, thus preserving the capability of meeting certain requirements. For instance, to provide timely response to service requests, a news website with self-adaptive capabilities can react to high response latencies by activating more servers, or reducing the fidelity of contents being served [6,12].

Deciding which adaptations should be carried out in response to changes in the execution environment requires that systems embody knowledge about themselves. Knowledge about the impact of adaptation choices on system's properties is particularly important when the decision process involves comparing alternative adaptations at runtime, as is often the case [11,5,17,14].

The effectiveness of the enacted changes, which affects the system's ability to meet its requirements, strongly depends on the accuracy of the analytical models that are used for decision making. Exact models, if attainable at all, tend to be quite complex and costly to obtain. As argued in [8], an alternative is to attend to the uncertainty underlying the knowledge models in the decision process.

However, existing models used to describe the impact of adaptations are either unable to capture the underlying uncertainty and variability of such dynamic execution environments, or are not compositional and described at a level of abstraction too low to scale in terms of specification effort required for non-trivial systems.

In this paper, we address the specification of probabilistic impact models for architecture-based self-adaptive systems that support the representation of: (i) uncertainty in the outcome of adaptation actions (e.g., the activation of a server can fail with some given probability), and (ii) context variability (e.g., the impact on response time of activating a single server will progressively reduce with a growing number of active servers). The core of our approach is a declarative specification language for expressing complex probabilistic constraints over state transitions that is equipped with a formal semantics defined in terms of Discrete Time Markov Chains (DTMC). This language provides the means for expressing impact models in a flexible and compact way.

We illustrate how the proposed impact models can be used in the context of the Rainbow framework [11] for architecture-based self-adaptation, and quantify the benefits of using probabilistic impact models instead of constant impact vectors [5].

The rest of the paper is organized as follows. Section 2 describes related work. Section 3 provides a formal account of the concepts required to define impact models. Section 4 presents the syntax and semantics of a new specification language of probabilistic impact models and Section 5 shows how our impact models can be used in the context of Rainbow for adaptation strategy selection. Next, experimental results that quantify the benefits of using probabilistic impact models instead of impact vectors are presented in Section 6. Finally, Section 7 presents some conclusions and future work.

2 Related Work

Environment domain models are a key element used by adaptive systems to determine their behavior [1,18]. These models capture the knowledge that the system has about itself and its environment by describing how system and environment respond to adaptation actions. Approaches to self-adaptation can be divided into two categories, depending on the way in which environment domain models are built.

A first category takes a systematic approach to modeling the impact of individual adaptation actions, which can be composed to reason about system behavior under adaptation. An example is the approach presented in [5], developed around Stitch, a language that enables the specification of adaptation strategies composed of individual adaptation actions. The impact of these action is specified in terms of constant impact vectors which describe how the execution of adaptation actions affects system quality attributes. The same type of impact models is used in several approaches to optimization of service compositions, such as the approach presented in [14]. Adaptation actions in this approach target service composition instances and the optimal criteria relies on impact models that are defined per adaptation action and system property as constant functions. Slightly more expressive impact models are considered in the approach presented [17], which targets component-based systems where impact models are defined per adaptation action and key performance indicators (KPIs), as functions over a given set of KPIs. These approaches address the specification of environment domain mod-

els in a compositional way and at a very high level of abstraction, thus facilitating specification and promoting reuse. However, they severely limit the ability to represent environment domain knowledge in a realistic way, since they are unable to model uncertainty and provide limited support to capture variability.

The second category consists of approaches that consider the behavior of the system and its environment modeled in a monolithic way in terms of more powerful models defined at a lower level of abstraction [2,1,9]. For example, in the approach presented in [2], DTMCs are used to model, for each system configuration, the future state of a system and its environment if that configuration is used. These models are expressive enough to model variability and the uncertainty underlying adaptation outcomes. The main drawback here is that these models being defined at the level of system configurations are system specific. For instance, in the case of a news website that can react to high response latencies by activating more servers, a DTMC that models the effect of activating one server in a system that can use up to 4 servers is completely different from another that models the same on a system that can use up to 10 servers, although the effect of activating one server does not depend on the maximum number of servers. Moreover, these models are both difficult and cumbersome to write. The specification of a DTMC tends to be a non-trivial task, even using description languages such as the one built into the probabilistic model checker PRISM [13].

The approach described in this paper aims at striking a balance between the ease of specification and reusability found in compositional approaches, and the expressive power of monolithic approaches that use probabilistic models. We present a language for the specification of impact models, which is: (i) more intuitive than describing DTMCs in other probabilistic approaches, since it is based on architectural descriptions and therefore raises the level of abstraction, (ii) able to capture both variability and probabilistic outcomes of adaptation actions, and (iii) scalable in terms of specification effort, since developers can focus on smaller units of conceptualization (i.e., architectural properties) and reason about them individually.

3 Modeling Adaptation

We address the modeling of impact in the context of architecture-based approaches to self-adaptation, that take the architectural style of the managed system as a basis for the system adaptation. The aim is to support the specification of impact models for families of systems that share the same architectural style. The semantics of such specifications assigns, for each system in the family, an impact model (a DTMC).

In this section, we provide a formal account of the concepts required to define impact models, namely architectural style and system state. We start by introducing the running example used in the rest of the paper.

3.1 Running example

Znn.com [3] is a case study portraying a representative scenario for the application of self-adaptation in software systems which has been extensively used to assess different

research in self-adaptive systems. Znn.com is able to reproduce the typical infrastructure for a news website, and has a three-tier architecture consisting of a set of servers that provide contents from backend databases to clients via front-end presentation logic (Fig. 1). The system uses a load balancer to balance requests across a pool of replicated servers, the size of which can be adjusted according to service demand. A set of clients makes stateless requests, and the servers deliver the requested contents.

The main objective for Znn.com is to provide content to customers within a reasonable response time, while keeping the cost of the server pool within a certain operating budget. It is considered that from time to time, due to highly popular events, Znn.com experiences spikes in requests that it cannot serve adequately, even at maximum pool size. To prevent loss of customers, the system can provide minimal textual content during such peak times, to avoid not providing service to some of its customers. Concretely, there are two main quality objectives for the self-adaptation of the system: (i) performance, which depends on request response time, server load, and network bandwidth, and (ii) cost, which is associated with the number of active servers.

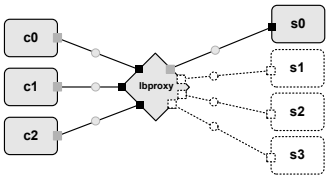


Fig. 1. Znn.com architecture

3.2 Architectural Style

As discussed in [4], when the architectural style of the managed system is taken as a basis for the system adaptation, it has to define not only the class of models to which the managed system architecture belongs but also to determine the operators representing available configuration changes on systems in that style and prescribes what aspects of a system and its execution context need to be monitored.

An architectural style defines a vocabulary of component and connector types that can be used in instances of that style and the properties of each of these types. In the context of self-adaptive systems, it is essential to distinguish between managed and monitored properties. *Managed properties* correspond to properties that are uniquely under system control. Their values can be defined at startup and changed subsequently by the control layer to regulate the system. *Monitored properties* correspond to properties of the managed system or its execution context that need to be monitored and made available to the control layer. While the properties of the execution context are not under the system control (e.g., available bandwidth), monitored properties also include those that the system aims to control (e.g., response time).

As an example, we consider the architectural style of Znn.com. It has one connector type — HttpConnT, and three component types — ClientT, ServerT and ProxyT. For instance, ServerT has two managed properties — isTextualMode:bool and cost:int. The former defines whether web pages are served by a given server in textual or multimedia mode and the latter reflects the cost of an active server per unit time. Since these properties are defined as uniquely under the system's control, the cost of each server included in the system must be defined at deployment time, and whether it will start serving pages in textual or multimedia mode. Additionally, ServerT has two monitored

properties — load:double and isActive:bool. The latter property is defined as monitored, since even if its activation can be controlled, a server may crash anytime.

Formally, architectural signatures are defined as follows.

Definition 1 (Architectural Signature). *An architectural signature Σ consists of a tuple of the form $\langle CompT, ConnT, \Pi^o, \Pi^m \rangle$, where CompT and ConnT are two disjoint sets (the sets of, respectively, component and connector types) and, Π^o and Π^m are functions that assign, to architectural types $\kappa \in CompT \cup ConnT$, mutually disjoint sets whose elements are typed by datatypes in a fixed set $\mathcal{D}$ ($\Pi^o(\kappa)$ and $\Pi^m(\kappa)$ represent, respectively, the managed and monitored properties of the type κ). We abbreviate $\Pi^o(\kappa) \cup \Pi^m(\kappa)$ by $\Pi(\kappa)$ and, for $p \in \Pi(\kappa)$, will use $dtype(\kappa.p)$ to denote its datatype.*

The architectural configurations of systems with architectural signature Σ, hereafter called Σ-system states, are captured in terms of graphs of components and connectors with state. State of architectural elements consist of the values taken by their monitored and managed properties.

We formally define Σ-system states assuming there is a fixed universe $\mathcal{A}_\Sigma$ of architectural elements (components and connectors) for Σ, i.e., a countable set whose elements are typed by elements in $CompT \cup ConnT$. We use $type(c)$ to denote c's type.

Definition 2 (Σ-System State). *A Σ-*system state *s consists of (i) a simple graph $\mathcal{G}$, (ii) a function $type$ that assigns an architectural type to every node of $\mathcal{G}$, (iii) an injective function I^s from the set of nodes of $\mathcal{G}$ to $\mathcal{A}_\Sigma$ such that $type(I^s(c)) = type(c)$ and (iv) a function that assigns a value $[\![c.p]\!]^s$ in the domain of $dtype(\kappa.p)$, to every pair $c.p$ such that c is a node of $\mathcal{G}$, $\kappa = type(c)$ and $p \in \Pi(\kappa)$. We denote by $\mathcal{S}_\Sigma$ (or simply $\mathcal{S}$ when Σ is clear from the context) the set of all Σ-system states.*

An architectural style also defines the ways one can change systems with that style. For instance, the Znn architectural style defines that property isTextualMode of servers can be modified through setLowFidelity and setHighFidelity operators, that set isTextualMode to, true and false, respectively.

Generically, these operators can range from primitive operations, such as changing the value of a property of a given connector type or replacing an implementation of a component type with another, to higher-level operations that exploit restrictions of that style. However, in practice, most approaches to self-adaptation consider only primitive operations. Hence, we focus on architectural styles that define the set of operators provided by the target system for (i) changing the values of the managed properties of its components and connectors and (ii) replacing a component or connector of a given type by another of the same type. Notice that, in these architectural styles, only the last two components of a system state —I^s and $[\![\ ]\!]^s$— can change at runtime. The structure of the system, defined by the graph and $type$ function, does not change.

3.3 Adaptation actions

The adaptation of the managed system is achieved through the execution of adaptation actions defined at design-time. Adaptation actions define actions packaged as applications of one or more operators, with a condition of applicability. In the Znn example we

could, for instance, define an adaptation action switchToTextualMode applicable only if there is at least one active service not serving pages in textual mode, prescribing the application of operator setLowFidelity to all servers in these conditions. A different adaptation action for Znn is enlistServer that is applicable when there is at least one inactive server, and prescribes the application of startServer to one inactive server.

Applicability conditions of adaptation actions are formulas of a constraint language that are evaluated over system states. For illustration purposes, we consider a constraint language inspired by that of Acme [10] [1] in which, for instance, (exists s:ServerT | exists k:HttpConnT | attached(k,s) and s.isActive) holds in a system state iff the state includes at least one active server attached to one http connector.

4 Modeling Impact

Deciding how to best adapt the system when a certain anomaly is detected involves analyzing models describing the effects, in terms of costs and benefits, of the available adaptation actions defined for the system. These models capture the causal relationship between an adaptation action's execution and its impact on the different system properties. Because 100% accurate models are in general not attainable, it is important to have means to address the underlying uncertainty.

In this section, we describe an expressive language to model adaptation action execution, which is able to capture: (i) the context that might influence the outcome of an adaptation action's execution, and (ii) the intrinsic uncertainty that pervades self-adaptive systems. Specifically, this language enables the description of models of the expected impact of each adaptation action on the different system properties. These models are based on DTMCs [15], and enable us to express alternative possible outcomes of the execution of the same adaptation action with some given probability.

4.1 Impact Model Language

The impact model of adaptation actions is defined in terms of probabilistic expressions in a language that allows one to express probabilistic constraints over state transitions (regarded as pairs of before and after system states), incorporating some elements of the PRISM language [13].

The language targets systems whose structure does not change at runtime. For this reason, it is built over a language $\mathcal{E}$ for specifying sets of components and connectors in a system state. For illustration purposes, we use a language in line with the one that we use to express constraints, in which, for instance, (s: ServerT | s.isActive) describes the set of active servers in a system state. To handle data, we assume that the fixed set of datatypes $\mathcal{D}$ is equipped with the relevant operations. We denote by $\mathcal{T}$ the term language used to describe data values and by $\mathcal{T}_d(\Sigma, X)$ the set of terms built over variables in X denoting values of datatype d. Similarly, we use $\mathcal{E}_\kappa(\Sigma, X)$ to denote the set of expressions defined over the variables in X denoting sets of architectural elements of type κ.

[1] Acme is in turn derived from OCL [16], with the addition of functions that relate to architectural structure.

Definition 3 (Probabilistic Expressions). *Let X be a set of variables typed by architectural types in an architectural signature Σ. The set $\mathcal{P}(\Sigma, X)$, of* probabilistic expressions *over variables in X, is defined by the following grammar:*

$$
\begin{array}{ll}
\alpha ::= x.p' = t & \textit{with } x \in X,\ p{\in}\Pi(\kappa),\ t{\in}\mathcal{T}_d(\Sigma, x_\Pi), \\
 & \quad \textit{where } \kappa = type(x) \textit{ and } d = dtype(p) \\
\mid \textbf{forall}\ x : \epsilon \mid \alpha_1 & \\
\mid \textbf{foreach}\ x : \epsilon \mid \alpha_1 & \\
\mid \textbf{foreach}\ x : \epsilon\ \textbf{minus}\ D \mid \alpha_1 & \textit{with } x \notin X,\ \epsilon{\in}\mathcal{E}_\kappa(\Sigma, X), \\
 & \quad \alpha_1{\in}\mathcal{P}(\Sigma, X{\cup}\{x : \kappa\}) \textit{ and } D{\subseteq}X_\kappa \\
\mid \{\alpha_1 \& \ldots \& \alpha_n\} & \\
\mid \{[p_1]\alpha_1 + [p_2]\alpha_2 + \cdots + [p_n]\alpha_n\} & \textit{with, for } 1 \leq i \leq n,\ \alpha_i{\in}\mathcal{P}(\Sigma, X), \\
 & \quad 0 \leq p_i \leq 1 \textit{ and } \Sigma_{i=1}^{n} p_i = 1
\end{array}
$$

where $x_\Pi{=}\{x.p : d \mid p{\in}\Pi(type(x)), d = dtype(p)\}$ and $X_\kappa{=}\{x{\in}X : type(x) = \kappa\}$. $\mathcal{P}(\Sigma)$ is the set of probabilistic expressions without free variables, i.e., $\mathcal{P}(\Sigma, \emptyset)$.

The atomic expression $x.p' = t$ defines the value of the property p in the next state (after the execution of the adaptation action), for every component or connector denoted by x. This value can be defined in terms of the values of the properties of the same element as well as other architectural elements in the system, but the free variables of t are limited to variables representing properties of x. For instance, assuming that s is a variable of type ServerT, we can write s.isActive' = !s.isActive to express that every server denoted by s has its isActive property toggled.

The operator **forall** is used to impose the same constraints over a set of architectural elements of the same type, denoted by a given expression in $\mathcal{E}$. The operator **foreach** is used to define a number of alternative outcomes, all with the same probability. For instance, **foreach** x:ServerT | x.isActive' = true states that all servers have the same probability of having their isActive property set to true. Adding **minus** D to the expression reduces the target to elements not included in the denotation of variables in D.

For instance, **foreach** x:E | **foreach** y:E **minus** x | {x.isActive' = true & y.isActive' = true} where E is (s:ServerT|!s.isActive), expresses that exactly two servers are activated and that all pairs of distinct inactive servers have the same probability of being activated.

A fixed number of constraints over the next state are expressed through conjunction (&). Probabilities that sum to one are assigned to a fixed number of expressions defining constraints over alternative outcomes of the adaptation action execution. Assigning a probability to an expression with $[p]\alpha$ has the effect of world closure: all properties of components and connectors not constrained by α are considered to keep the same value in the next state.

To capture that an adaptation action may have different impacts under different conditions, impact models are defined as sets of guarded probabilistic expressions with mutually exclusive guards (i.e., at most one guard holds in any system state). As before, we abstract from the language used for expressing the guard conditions and assume a fixed language $\mathcal{C}$ of constraints over system states.

Definition 4 (Impact Model). *An impact model $\mathcal{I}$ of an adaptation action is a finite set of pairs $\langle \phi, \alpha \rangle$ where ϕ is a constraint in $\mathcal{C}(\Sigma)$ and α is a probabilistic expression in $\mathcal{P}(\Sigma)$ such that all ϕ are mutually exclusive.*

An example of a simple impact model is presented below for the adaptation action switchToTextualMode. For the sake of clarity, we present all examples making use of a concrete syntax that supports the definition of abbreviations and in which guarded expressions are represented as $\phi \rightarrow \alpha$.

```
1 define S=(s:ServerT | !s.isTextualMode and s.isActive) and define k=size(S)
2 define f(x)=x∗(1−k/(2∗(k+1))) and define g(x)=x∗(1−k/(k+1))
3 impactmodel switchToTextualMode
4  k>0 → { [0.8] { forall s:S | s.isTextualMode'=true & forall c:ClientT | c.expRspTime'=f(c.expRspTime) }
5          + [0.2] { forall s:S | s.isTextualMode'=true & forall c:ClientT | c.expRspTime'=g(c.expRspTime) }}
```

Listing 1.1. Impact model for adaptation action switchToTextualMode.

This model expresses the impact of the adaptation action over manipulated properties, where the fact that operator setLowFidelity sets the property isTextualMode to true is represented by s.isTextualMode'=true. Moreover, the model foresees that switchToTextualMode can impact the response time of *all clients* in two ways, both decreasing its value, considering the number of servers that were changed to low fidelity. The more severe reduction of the response time is defined to be the least likely, with probability 0.2. According to this (simplistic) model, the execution of this adaptation action is not expected to affect the remaining properties of servers, clients, or http connectors.

Alternatively, we could specify that switchToTextualMode can impact the response time of *each client* in two ways as follows:

```
1 impactmodel switchToTextualMode
2  k>0 → forall c:ClientT | { [0.8] { forall s:S | s.isTextualMode'=true & c.expRspTime'=f(c.expRspTime) }
3                           + [0.2] { forall s:S | s.isTextualMode'=true & c.expRspTime'=g(c.expRspTime) }}
```

Listing 1.2. Alternative impact model for switchToTextualMode.

While we have considered that the property isTextualMode of servers is subject only to system control, isActive was defined as a monitored property and it was considered that the activation of a server, through the execution of operator startServer, may fail. An impact model for enlistServer that captures this aspect is presented below.

```
1 define m=size(s:ServerT | s.isActive) and define S= (s:ServerT | !s.isActive)
2 define f(x)=x∗(1−((1/log(100∗m,2))∗(m/(2∗m+1)))) and define g(x)=x∗(1−1/log(100∗m,2))
3 impactmodel enlistServer
4  m>0 → { [0.95] { foreach s:S | s.isActive'=true &
5                              { [0.7] forall c:ClientT | c.expRspTime'=f(c.expRspTime)
6                              + [0.3] forall c:ClientT | c.expRspTime'=g(c.expRspTime) }}
7          + [0.05] { forall s:ServerT | s.isActive'=s.isActive & forall c:ClientT | c.expRspTime'=c.expRspTime }}
```

Listing 1.3. Impact model for adaptation action enlistServer.

This impact model implicitly states that the starting of the server is expected to fail with probability 0.05 and foresees that the adaptation action may impact client response time in two ways, both considering the number of servers that were already active.

Moreover, isActive is also defined as a monitored property, since a server could become spontaneously inactive (e.g., due to a server crash). The impact model above does not define any impact of the adaptation action over the property isActive of already active servers, hence the probability of an active server crashing while executing enlistServer is considered to be so small that it can be neglected. Alternatively, we can define the probability of each relevant crash scenario (e.g., for one server, two servers,

etc). For instance, the impact model presented below defines that the probability of exactly one active server crashing while executing enlistServer is 0.001.

```
1 define T=(s:ServerT | s.isActive)
2 impactmodel enlistServer
3 m>0 → { [0.999] { foreach s:S | s.isActive'=true &
4                   { [0.7] forall c:ClientT | c.expRspTime'=f(c.expRspTime) +
5                     [0.3] forall c:ClientT | c.expRspTime'=g(c.expRspTime) } }
6         + [0.001] { foreach s:S | s.isActive'=true & foreach t:T | t.isActive'=false &
7                     forall c:ClientT | c.expRspTime'=c.expRspTime } }
```

Listing 1.4. Alternative impact model for adaptation action enlistServer.

4.2 Impact Model Semantics

The semantics of impact models is formally defined in terms of DTMCs. Since a DTMC has a discrete state space, we have to limit properties of components and connectors to take values in discrete sets and perform quantization.

Quantization. For each property p that takes values in a datatype $d \in \mathcal{D}$ that has a non-countable domain $\mathcal{I}_d$, it is necessary that a countable set $[\mathcal{I}_d]_p$ and a quantization function $Q_p : \mathcal{I}_d \to [\mathcal{I}_d]_p$ be defined. For each property $p : d$ such that $\mathcal{I}_d$ is countable we take $[\mathcal{I}_d]_p = \mathcal{I}_d$ and Q_p as the identity function.

The quantization of the properties of component and connector types can be propagated to the level of system states, defining a discrete set of states $[\mathcal{S}] = \{[s] : s \in \mathcal{S}\}$. In $[s]$, the value of a property p of a component or connector c is obtained by applying the corresponding quantization function to the value it has in s, i.e., $[\![c.p]\!]^{[s]} = Q_p([\![c.p]\!]^s)$.

The semantics of adaptation action impact models is defined in terms of DTMCs over $[\mathcal{S}]$. We start by providing the semantics of the probabilistic expressions used to assemble such models.

The interpretation of a probabilistic expression α over a set of variables X is defined in the context of a system state s and an *interpretation* ρ of X assigning to each variable $x{:}\kappa$, a set of elements in s of type κ. This interpretation, denoted by $[\![\alpha]\!]^s_\rho$, consists of a set Y of properties of component and connectors in s — those which are constrained by α — and a function P defining the probability of a transition between any pair of Y-states. As an example, consider $[\![\text{x.isActive'=true}]\!]^s_\rho$ where x : ServerT, s is a state with servers $z_1, \ldots, z_n$ and $\rho : x \mapsto \{z_1\}$. The expression constrains only the property $isActive$ of z_1, i.e., $Y = \{z_1.isActive\}$ and, hence, in this case, a Y-state is just a truth value for $z_1.isActive$. Its interpretation is that the probability of a transition from any Y-state to $\{z_1.isActive \mapsto true\}$ is 1 and to $\{z_1.isActive \mapsto false\}$ is 0.

Formally, given a set Y of properties, a Y-state s is a function defining the value of each property in $y \in Y$, subject to the corresponding quantization functions. As for system states, we simply write $[\![y]\!]^s$ and use $[\mathcal{S}_Y]$ for referring to the set of all Y-states.

An important operation over probabilistic expressions is world closure through assignment of a probability. As mentioned before, when we write $[p]\alpha$, all properties of components and connectors not constrained by α are considered to keep the same value in the next state. World closure can be captured by the following notion of closure over transition probability matrices:

Definition 5 (Closure). *Let $Y \subseteq Y'$ be two sets of properties. Given a function $P{:}[\mathcal{S}_Y] \times [\mathcal{S}_Y] \to [0,1]$, the closure of P to Y' is the function $P^{Y'}{:}[\mathcal{S}_{Y'}] \times [\mathcal{S}_{Y'}] \to [0,1]$ s.t.*

$$P^{Y'}(s_1, s_2) = \begin{cases} P(s_{1|Y}, s_{2|Y}) & \text{if } \forall y \in Y' \setminus Y,\ [\![y]\!]^{s_2} = [\![y]\!]^{s_1} \\ 0 & \text{otherwise} \end{cases}$$

where $s_{|Y}$ is the Y-state obtained through the restriction of s to the properties in Y.

The closure of P corresponds to extending the probabilities given by P to states with more properties, considering that their values do not change.

Definition 6 (Interpretation of Probabilistic Expressions). *The interpretation $[\![\alpha]\!]^s_\rho$ of $\alpha \in \mathcal{P}(\Sigma, X)$ in a system state s and an interpretation ρ of X is a pair of the form $\langle Y, P{:}[\mathcal{S}_Y] \times [\mathcal{S}_Y] \to [0,1] \rangle$ defined inductively in the structure of α as follows:*

- $[\![x.p' = t]\!]^s_\rho$ = $\langle Y, P \rangle$

  $Y = \{c.p : c \in \rho(x)\}$ *and* $P(s_1, s_2) = \begin{cases} 1 & \text{if } \forall c \in \rho(x), [\![c.p]\!]^{s_2} = [\![t]\!]^{s_1}_{\rho_c} \\ 0 & \text{otherwise} \end{cases}$

  with $\rho_c = \{x.q \mapsto [\![c.q]\!]^{s_1} : q \in \Pi(type(x))\}$

- $[\![$**forall** $x : \epsilon \mid \alpha]\!]^s_\rho = [\![\alpha]\!]^s_{\rho'}$ *with* $\rho' = \rho \oplus x \mapsto [\![\epsilon]\!]^s$

- $[\![$**foreach** $x : \epsilon \mid \alpha]\!]^s_\rho$ = $\langle Y, P \rangle$

  Let $C = [\![\epsilon]\!]^s$. *If* $|C| = 0$, *then* $Y = \emptyset$ *and* P *is the empty function to* $[0,1]$. *Otherwise, let* $\rho_c = \rho \oplus x \mapsto \{c\}$, *for every* $c \in C$, *and* $[\![\alpha]\!]^s_{\rho_c} = \langle Y_c, P_c \rangle$.

  $Y = \bigcup_{c \in C} Y_c$ *and* $P(s_1, s_2) = \sum_{c \in C} \frac{1}{|C|} \cdot P^Y_c(s_{1|Y_c}, s_{2|Y_c})$

- $[\![$**foreach** $x : \epsilon$ **minus** $D \mid \alpha]\!]^s_\rho$ = $\langle Y, P \rangle$

  Let $C = [\![\epsilon]\!]^s \setminus \rho(D)$. *If* $|C| = 0$, *then* $Y = \emptyset$ *and* P *is the empty function to* $[0,1]$. *Otherwise, let* $\rho_c = \rho \oplus x \mapsto \{c\}$, *for every* $c \in C$, *and* $[\![\alpha]\!]^s_{\rho_c} = \langle Y_c, P_c \rangle$.

  $Y = \bigcup_{c \in C} Y_c$ *and* $P(s_1, s_2) = \sum_{c \in C} \frac{1}{|C|} \cdot P^Y_c(s_{1|Y_c}, s_{2|Y_c})$

- $[\![\{\alpha_1 \& \cdots \& \alpha_n\}]\!]^s_\rho$= $\langle Y, P \rangle$

  Let $[\![\alpha_i]\!]^s_\rho = \langle Y_i, P_i \rangle$, *for* $i = 1, .., n$. *If the sets* $Y_1, \cdots, Y_n$ *are not mutually disjoint, then* $Y = \emptyset$ *and* P *is the empty function to* $[0,1]$. *Otherwise,*

  $$Y = \bigcup_{i=1,..,n} Y_i \ \text{ and } \ P(s_1, s_2) = \prod_{i=1}^{n} P_i(s_{1|Y_i}, s_{2|Y_i})$$

- $[\![\{[p_1]\alpha_1 + \cdots + [p_n]\alpha_n\}]\!]^s_\rho$ = $\langle Y, P \rangle$

  Let $[\![\alpha_i]\!]^s_\rho = \langle Y_i, P_i \rangle$, *for* $i = 1, .., n$.

  $$Y = \bigcup_{i=1,..,n} Y_i \ \text{ and } \ P(s_1, s_2) = \sum_{i=1}^{n} p_i \cdot P^Y_i(s_{1|Y_i}, s_{2|Y_i})$$

Notice that there are some state-dependent semantic restrictions over probabilistic expressions. If, in a given state s, an expression α does not meet these conditions, then α does not impose any restriction in the evolution of the system state (i.e., $Y = \emptyset$).

Proposition 1. *If $[\![\alpha]\!]^s_\rho = \langle Y, P : [\mathcal{S}_Y] \times [\mathcal{S}_Y] \to [0,1] \rangle$, then P is a transition probabilistic matrix, i.e., $\sum_{s_2 \in [\mathcal{S}_Y]} P(s_1, s_2) = 1$, for every $s_1 \in [\mathcal{S}_Y]$.* [2]

The quantization of component and connector properties may invalidate an impact model, by making pairs of constraints that were mutually exclusive, non mutually exclusive anymore. Invalid adaptation action models are inconsistent (i.e., they do not admit any interpretation) and, hence, we limit our attention to valid impact models.

Definition 7 (Semantics of Impact Models). *An impact model $\mathcal{I}$ is valid if for every $s \in \mathcal{S}$, there exists at most one element $\langle \phi, \alpha \rangle \in \mathcal{I}$ such that $[s] \models \phi$. The semantics of a valid impact model $\mathcal{I}$ is the DTMC $\langle [\mathcal{S}], P : [\mathcal{S}] \times [\mathcal{S}] \to [0,1] \rangle$ where P is defined as follows:*

If the graph of $s_1$and s_2 is not the same then $P(s_1, s_2) = 0$,
else if exists $\langle \phi, \alpha \rangle \in \mathcal{I}$ s.t. $s_1 \models \phi$ then $P(s_1, s_2) = P_\alpha^{Y_{s_1}}(s_{1|Y_\alpha}, s_{2|Y_\alpha})$
else if $s_1 \neq s_2$ then $P(s_1, s_2) = 0$ else $P(s_1, s_2) = 1$

where Y_s denotes the set of all properties of components and connectors in a system state s, i.e., $Y_s = \{c.p : d \mid c \text{ is a node in } s, p \in \Pi(type(c)), d = dtype(p)\}$.

As an example, consider an impact model defined by $\langle$size(E)>0, $\alpha\rangle$ with $\alpha =$ (**foreach** x:E | x.isActive' = true) and E=(s:ServerT|!s.isActive). Let s, s_1, s_2 be three system states with servers z_1, z_2, z_3 that only differ in the number of active servers: (i) in s only z_3 is active, (ii) in s_1 only z_2 is inactive and (iii) in s_2 only z_1 is inactive. According to definition above, we have for instance that $P(s, s_1) = P(s, s_2) = \frac{1}{2}$ and $P(s, s') = 0$, for every other system state s' different from s_1 and s_2.

5 Predicting Adaptation Strategy Impact

In this section we show how the proposed impact models can be used when the adaptation of the managed system is achieved through the execution of an adaptation strategy selected from a portfolio of strategies specified in Stitch [5].

5.1 Adaptation Strategies

Strategies are built from tactics, which are Stitch's adaptation actions. Strategies have an applicability condition and a body. The body of a strategy σ is a tree T_σ whose edges $n \to m$ are labelled by a guard condition, a tactic and a success condition. Once at node n, if the guard condition is true, it means that the edge can be taken. When a edge is taken, the corresponding tactic is executed. Upon its termination the success condition is evaluated to determine if the tactic achieved what was expected and node m is reached. Guards include a special symbol $success$ capturing whether the last tactic had succeeded or not.

[2] The proof of this result and an example of application of Def. 6 can be found in the Appendix.

An example of a strategy for Znn is simpleReduceResponseTime presented in Fig. 2. hiLoad, hiLatency and hiRspTime are formulas expressing respectively that the system load, latency and response time is high. hiRspTime, for instance, is defined in terms of the average response time of the clients by:

$$(\text{sum}(c.\text{expRspTime}|c\text{: ClientT})/\text{count}(c\text{: ClientT}) > \text{MAX_RSPTIME})$$

The body of the strategy defines that initially there are three alternatives, depending on the load and latency of the system. If the latency is high, then a possibility defined by the strategy is to just apply the tactic switchToTextualMode. The success condition in this case is hiRspTime that expresses that the average response time is below a given threshold. If the load is high, then a possibility is to apply the tactic enlistServer and, depending on the success of the application of this tactic, either terminate with skip or still try the application of the tactic switchToTextualMode. If neither the latency nor the load is high, the strategy does not offer any remedy.

5.2 Strategy Selection

A particular situation that requires adaptation can typically be addressed in different ways by executing alternative adaptation strategies, many of which may be applicable under the same run time conditions. Different strategies impact quality attributes in different ways; thus there is a need to choose a strategy that will result in the best outcome with respect to achieving the system's desired quality objectives. To enable decision-making for selecting strategies Stitch uses utility functions and preferences, which are sensitive to the context of use and able to consider trade-offs among multiple potentially conflicting objectives. Specifically, the process consists in selecting the strategy that maximizes its expected utility, which entails: (i) defining quality objectives, relating them to specific run-time conditions, and (ii) assessing the expected aggregate utility of every applicable strategy, based on the impact model of its tactics on the system's quality objectives (using utility functions and preferences).

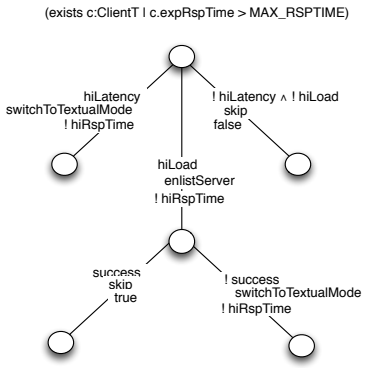


Fig. 2. A strategy for Znn.

Defining Quality Objectives Defining quality objectives requires identifying the concerns for the different stakeholders. In Znn, users are concerned with experiencing service without any disruptions, which can be mapped to specific run-time conditions such as response time. In contrast, the organization is interested in minimizing the cost of operating the infrastructure, which can be mapped to the cost of specific resources used at run-time (e.g., active servers). In short, we identify two quality objectives: maintaining low client response time (R), and cost (C). Table 1 summarizes the utility functions for Znn defined by an explicit set of value pairs (where intermediate points are linearly interpolated). Function U_R maps low response times (up to 100ms) with maximum utility,

whereas values above 2000ms are highly penalized (utility below 0.25), and response times above 4000ms provide no utility. Function $\mathsf{U_C}$ maps a increasing cost (derived from the number of active servers) to lower utility values. Utility preferences capture business preferences over the quality dimensions, assigning a specific weight ($w_{\mathsf{U_R}}$, $w_{\mathsf{U_C}}$) to each one of them. In Znn, preference is given to performance over cost.

$\mathsf{U_R}(w_{\mathsf{U_R}} = 0.6)$				$\mathsf{U_C}(w_{\mathsf{U_C}} = 0.4)$		
0 : 1.00	200 : 0.99	1000 : 0.70	2000 : 0.25	0 : 1.00	2 : 0.90	4 : 0.00
100 : 1.00	500 : 0.90	1500 : 0.50	4000 : 0.00	1 : 1.00	3 : 0.30	

Table 1. Utility functions and preferences for Znn

To compute the utility of a given system state s (denoted as $Util(s)$), we first need to map the values of the different qualities [3] to their corresponding utility values. In a system state with 1250 ms of response time and a cost of 2 usd/hour, based on the utility functions defined in Table 1, we have $[\mathsf{U_R}(1250),\mathsf{U_C}(2)]=[0.625, 0.9]$. Finally, all utilities are combined into a single value, using utility preferences: $0.625*0.6+0.9*0.4=0.735$.

Assessing the Aggregate Utility of Strategies The expected utility of a strategy σ in a given system state s can be formulated in terms of a tree like that presented in Fig. 3, i.e., a labelled tree with two types of nodes — normal and chance nodes, that alternate in consecutive depth levels of the tree. As with decision trees, chance nodes represent situations in which the choice between the different alternatives is external (i.e., not under the system's control), and is governed according to a given probability distribution function. Normal nodes, as decision nodes of decision trees, represent situations in which the choice between the different alternatives is internal. These nodes reflect situations of non-determinism during the execution of the strategy (that arise when more than one edge can be executed) that we assume are solved by a fair scheduler and, hence, all alternatives have the same probability of being taken. In this way, all edges $\langle n, m\rangle$ of the tree are labelled with a probability p; if n is a normal node then the edge is additionally labelled by a tactic t. For short we write, respectively, $n \overset{p}{\text{——}} m$ and $n \overset{p}{\underset{t}{\text{——}}} m$. Chance nodes are not labelled whereas every normal node n is labelled by a system state.

Formally, this type of labelled trees can be represented as a tuple $\langle N, st, E, l\rangle$ with $N{=}H{\cup}A$, where H and A are the sets of, respectively, chance and normal nodes, st is a function that labels nodes in H with system states, E is the set of edges and l labels edges with a probability and, optionally, a tactic. The tree defined by a strategy σ in a given system state s, which we denote by $T_{\mathcal{I}}(\sigma, s)$, is defined as follows.

Definition 8 **($T_{\mathcal{I}}(\sigma, s)$).** *Let σ be a strategy and s a system state. Given an impact model $\mathcal{I}_t$ for every tactic t used in σ, $T_{\mathcal{I}}(\sigma, s)$ is the labelled tree obtained as follows:*

1. *Start with an empty set of chance nodes and edges and with a single normal node $root(T_\sigma)$ labelled by $[s]$.*

[3] For utility calculation, we assume a representation of system state in terms of qualities. In Znn, we take the average of response time in all clients and the sum of the costs of active servers.

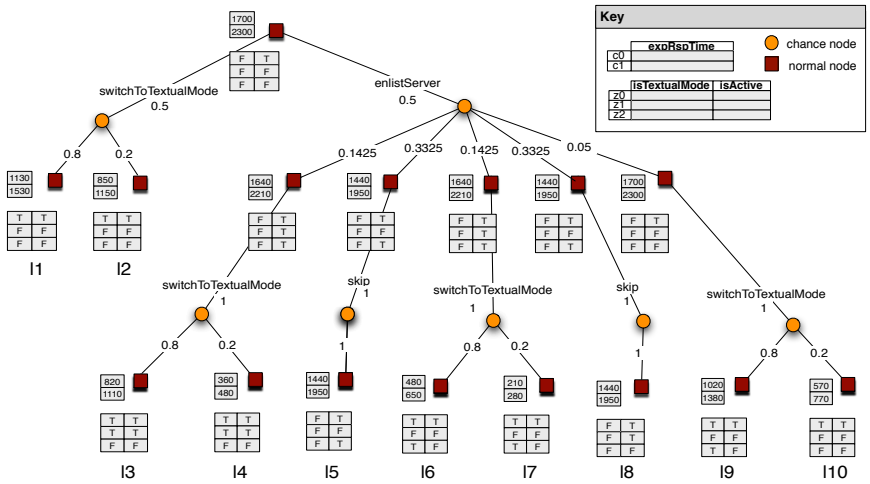


Fig. 3. Tree for simpleReduceResponseTime and a system state with 2000 ms of response time, and a cost of 2 usd/hour.

2. *While there exists a normal node n that has not been considered before:*
   (a) *Find the edges of T_σ that start in n and can be executed in state $st(n)$:*

$$E_n = \{n \xrightarrow{\langle \phi, t, \psi \rangle} m \text{ in } T_\sigma : st^*(n) \vDash \phi\}$$

   where, supposing that k is the parent node of n and ψ' is the success condition of the edge that leads to n in T_σ, $st^*(n)$ is the extension of $st(n)$ with the interpretation of success with true if ψ' holds in $st(k)$ and false, otherwise.
   (b) *For every $n \xrightarrow{\langle \phi, t, \psi \rangle} m \in E_n$:*
      i. *add m to the set of chance nodes and $n \xrightarrow{1/|E|} m$ to the set of edges.*
      ii. *for every state s' such that $p = P_{[\![\mathcal{I}_t)]\!]}(st(n), s') > 0$, add the node $m^{s'}$ labelled by s' to the set of normal nodes and $m \xrightarrow{p} m^{s'}$ to the set of edges.*

Fig. 3 presents the tree $T_{\mathcal{I}}(\text{simpleReduceResponseTime}, s)$ where s is system state with two clients (c_0, c_1) and three servers (z_0, z_1, z_2). Only server z_0 is active in s and is not working in textual mode. The cost assigned to all servers is 1 and the load assigned to z_0, z_1, z_2 is, respectively, 2, 0 and 0. The response time for c_0 is 1700 and for c_1 is 2300. The considered impact models for tactics switchToTextualMode and enlistServer were those presented, respectively, in Listings 1.1 and 1.3. For readability reasons we represented in the figure only the part of the system state that is directly manipulated or affected by the two tactics. Then, using the utility profile, we can calculate the utility of the state associated with each leaf node. The expected utility of the strategy is given by the sum of these utilities weighted by the probability of the path that leads to that node.

Definition 9 (Expected Utility of a Strategy). *Given a set of impact models $\mathcal{I}$, the expected utility of a strategy σ in a system state s is given by*

n	p_n	avg. expRspTime (ms)	cost (usd/hour)	$\mathsf{U_R}(st(n))$	$\mathsf{U_C}(st(n))$	$Util(st(n))$	$p_n \cdot Util(st(n))$
l1	0.4	1330	1	0.585	1	0.751	0.3004
l2	0.1	1000	1	0.75	1	0.85	0.085
l3	0.057	965	2	0.7605	0.9	0.8163	0.046529
l4	0.01425	410	2	0.927	0.9	0.9162	0.013056
l5	0.16625	1695	2	0.4025	0.9	0.6105	0.1014496
l6	0.057	565	1	0.8805	1	0.9283	0.052913
l7	0.01425	245	2	0.9765	1	0.9859	0.014049
l8	0.16625	1695	2	0.4025	0.9	0.6105	0.101496
l9	0.02	1200	1	0.65	1	0.79	0.0158
l10	0.005	670	1	0.849	1	0.9094	0.004245
total	**1**	-	-	-	-	-	**0.734937**

Table 2. Sample calculation of aggregate utility for strategy simpleReduceResponseTime

$$\sum_{n \in leaves(T_{\mathcal{I}}(\sigma,s))} p_n \cdot Util(st(n))$$

where p_n is the product of the probabilities in the path leading from the root to n.

Table 5.2 illustrates the utility calculation for strategy simpleReduceResponseTime that corresponds to the tree shown in Fig. 3. Note that nodes l1 and l2 contribute half of the utility, and that the sum of all p_n assigned to leaf nodes adds to one.

6 Experimental Results

To quantify the benefits of using probabilistic impact models in Rainbow, we considered two alternative models of Znn, one using impact vectors [4] and the other using the proposed impact models. We manually encoded the two models in the language of PRISM [13] and assessed the quality of the models that we are able to specify in each case by quantifying: (i) impact of tactics on the state of the target system, and (ii) impact of strategies on system utility. In addition, both models incorporate an M/M/c queuing model [7], which is able to compute the response time of the system based on the rate of request arrivals to the system, number of active servers, and the service rate (i.e., the time that it takes to service a request, which in this case is directly proportional to the fidelity level). In our experiments, we assume that the response time computed with the queuing model is considered as the actual response time of the system, against which we compare the predictions made using either vectors or probabilistic models.

Using each of the alternative models, we explored a state space $[S] = [1, 9] \times [1, 3]$, which includes the request arrival rate in the interval $[1, 9]$ requests/s, and the number of active servers in the interval $[1, 3]$ (i.e., a valid system configuration can have up to a maximum of 4 active servers). For the sake of clarity, we fixed in our experiments the values of other variables which could have been considered as additional dimensions in our state space (network latency is 0 ms, whereas service rate is fixed at 1 ms).

6.1 Impact of Tactics on System State

To quantify the improvement obtained using vectors with the results of employing probabilistic models, we focused on the enlistServer tactic, for which we encoded two alternative impact descriptions:

[4] The impact of individual adaptation actions is specified in terms of constant impact vectors (called cost/benefit attribute vectors) which describe how the execution of adaptation actions affects system quality attributes [5].

Cost-benefit Attribute Vectors. For the vector-based version of the model, we computed the average impact in response time of adding a server in all points of the explored region of the state space, making use of the M/M/c queuing model, which is the best approximation that can be obtained, given by $(\sum_{s\in[S]} \mathit{MMc}(ar_s, as_s + 1) - \mathit{MMc}(ar_s, as_s))/|[S]|$, where $\mathit{MMc}(a, b)$ returns the response time for request arrival rate a and number of active servers b. Moreover, ar_s and as_s designate the request arrival rate, and number of active servers in state s, respectively.

For our state space, this calculation yielded a reduction of response time of 714 ms. Since the cost is increased in 1 unit, and fidelity is not changed by enlistServer, the vector used in our experiments for the tactic is $[-714, +1, 0]$.

Probabilistic Impact Models. The probabilistic version of the model employed for the experiments is analogous to the one described in Listing 1.3.

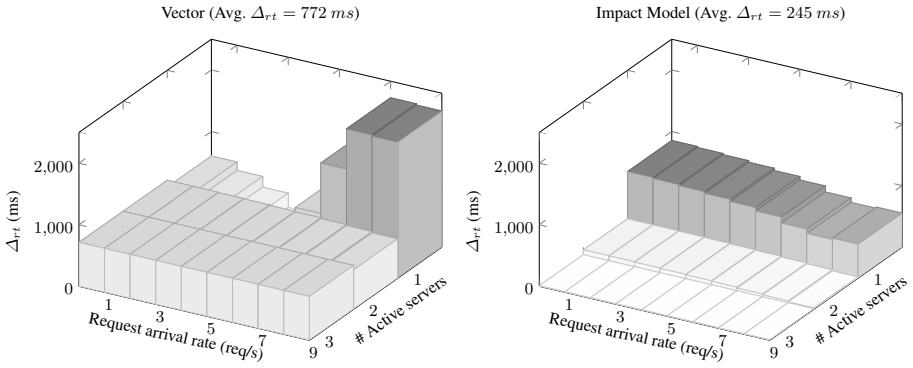


Fig. 4. Deviation in response time impact prediction for tactic enlistServer: cost/benefit vector (left) and probabilistic impact model (right).

Fig. 4 shows the deviation from actual response time impact values (computed using the M/M/c model) for tactic enlistServer. The values computed using the probabilistic impact model (right) are much more accurate, since their deviation is far less prominent than the one presented when computing impact with vectors (average deviation Δ_{rt} in values computed using vectors is $\simeq 315\%$ wrt the values obtained using impact models). Moreover, while the values computed using vectors are not sensitive to context, presenting reduced deviations wrt actual response times only in states that are close to the average (e.g., 1 server, 3-5 requests/s), values obtained with the probabilistic model better approximate actual impact, reflecting the fact that a higher number of active servers noticeably reduces the impact of the tactic on response time.

6.2 Impact of Strategies on System Utility

The use of different models to express the impact of tactics on system state also affects the predictions that concern the expected utility of the system after the execution

of adaptation strategies. To assess how utility prediction is affected by the constructs available to express tactic impact, we included in our PRISM model an encoding of the strategy simpleReduceResponseTime shown in Fig. 2, and computed the expected utility after its execution for each of the states included in $[S]$ for each of the alternatives.

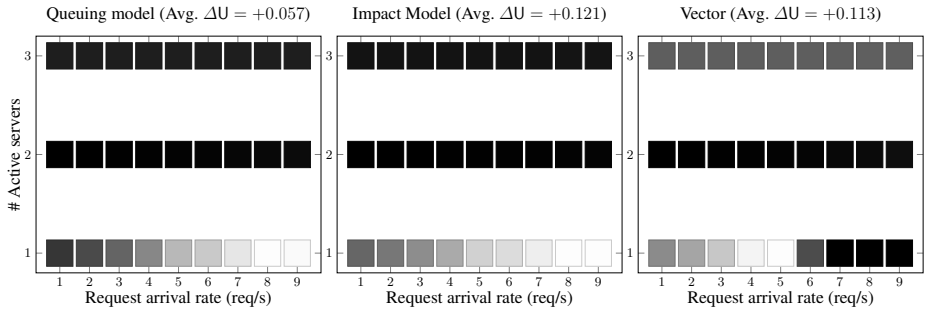


Fig. 5. Utility prediction for the execution of strategy reduceResponseTime based on: queuing model, impact model, and impact vector. Lighter colors represent higher utility improvements.

Fig. 5 shows how the utility values predicted using probabilistic impact models (center) exhibit a similar pattern to the one obtained using the queuing model (left). In contrast, vectors (right) show an entirely different pattern, which only coincides with the one resulting from the queuing model in the area in which the impact of tactics is close to their average impact (i.e., when there are two active servers).

It is worth noticing that the overall average difference in utility ΔU across the state space does not constitute a representative indicator of the accuracy of utility predictions, since positive and negative utility deltas in different states can cancel each other (i.e., the average ΔU yielded by vectors is $+0.113$, which is closer to the one obtained with the queuing model of $+0.057$, even if the absolute value of the deviation in individual states in vectors is greater than in the case of impact models).

7 Conclusions and Future Work

In this paper we addressed the specification of impact models for self-adaptive systems and presented a declarative language that allows one to explicitly represent the uncertainty in the outcome of adaptation actions. The mathematical underpinnings of the language were heavily influenced by the input language of PRISM [13], but its syntax is also based on the language of structural constraints of Acme [10]. The language was shown to have the ability to express sophisticated impact models, providing expressive and compact descriptions. Although there is an upfront investment in learning the notation and specifying these impact models compared to other approaches [17,5], the fact that we can model variability improves reusability (e.g., across systems sharing the same architectural style).

We also showed how the proposed impact models can be used in the context of Rainbow with adaptation strategies defined in the language Stitch [5], and proposed a new method for calculating the utility of a strategy. The benefits of the proposed impact models can be extended to other architecture-based approaches to self-adaptation that rely on impact models for adaptation decision-making such as [17] and [14].

Regarding future work, we plan on extending our declarative language to cater to architectural styles that support structural changes. Moreover, we plan on leveraging and furthering formal analysis of adaptation behavior by encoding impact models described in our language into existing tools. A third research direction aims at further refining our approach to consider time as a first-class entity in impact models.

References

1. R. Calinescu, C. Ghezzi, M. Kwiatkowska, and R. Mirandola. Self-adaptive software needs quantitative verification at runtime. *Commun. ACM*, 55(9):69–77, Sept. 2012.
2. R. Calinescu and M. Z. Kwiatkowska. Using Quantitative Analysis to Implement Autonomic IT Systems. In *ICSE*, 2009.
3. S. Cheng et al. Evaluating the Effectiveness of the Rainbow Self-Adaptive System. In *SEAMS*. IEEE, 2009.
4. S.-W. Cheng et al. Using architectural style as a basis for system self-repair. In *WICSA*, pages 45–59. Kluwer, B.V., 2002.
5. S.-W. Cheng and D. Garlan. Stitch: A language for architecture-based self-adaptation. *Journal of Systems and Software*, 85(12), 2012.
6. S.-W. Cheng, D. Garlan, and B. R. Schmerl. Raide for engineering architecture-based self-adaptive systems. In *ICSE Companion*, pages 435–436. IEEE, 2009.
7. R. Chiulli. *Quantitative Analysis: An Introduction*. Automation and production systems. Taylor & Francis, 1999.
8. N. Esfahani and S. Malek. Uncertainty in self-adaptive software systems. In *Software Engineering for Self-Adaptive Systems 2*, volume 7475 of *LNCS*, pages 214–238. Springer, 2010.
9. A. Filieri, C. Ghezzi, A. Leva, and M. Maggio. Self-adaptive software meets control theory: A preliminary approach supporting reliability requirements. In *ASE*, pages 283–292, 2011.
10. D. Garlan et al. Acme: An architecture description interchange language. In *CASCON*, pages 169–183, 1997.
11. D. Garlan et al. Rainbow: Architecture-Based Self-Adaptation with Reusable Infrastructure. *IEEE Computer*, 37(10), 2004.
12. C. Klein et al. Brownout: Building more robust cloud applications. In *ICSE*, pages 700–711. ACM, 2014.
13. M. Kwiatkowska et al. PRISM 4.0: Verification of probabilistic real-time systems. In *CAV*, volume 6806 of *LNCS*, pages 585–591. Springer, 2011.
14. P. Leitner, W. Hummer, and S. Dustdar. Cost-based optimization of service compositions. *IEEE T. Services Computing*, 6(2):239–251, 2013.
15. J. R. Norris. *Markov chains*. Cambridge series in statistical and probabilistic mathematics. Cambridge University Press, 1998.
16. OMG. Object Constraint Language V2.4. http://www.omg.org/spec/OCL/2.4, 2014.
17. L. Rosa, L. Rodrigues, A. Lopes, M. A. Hiltunen, and R. D. Schlichting. Self-management of adaptable component-based applications. *IEEE Trans. Software Eng.*, 39(3):403–421, 2013.
18. D. Sykes, D. Corapi, J. Magee, J. Kramer, A. Russo, and K. Inoue. Learning revised models for planning in adaptive systems. In *ICSE*, pages 63–71. IEEE, 2013.

Appendix

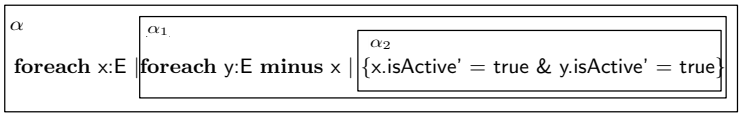
The Semantics of Probabilistic Expressions: an example

An example of probabilistic expression α presented in Section 4 is

foreach x:E | **foreach** y:E **minus** x | {x.isActive' = true & y.isActive' = true}

where E is (s:ServerT|!s.isActive).

Informally, α was said to express that exactly two servers are activated and that all pairs of distinct inactive servers have the same probability of being activated. Formally, according to Definition 6, the evaluation of this expression in a given system state s is given by a set of properties Y of components and connectors in s and a probability function for state transitions $P{:}[\mathcal{S}_Y] \times [\mathcal{S}_Y] \to [0,1]$. Below, we show, step by step, how Y and P are calculated, considering that the system state s has servers z_1, z_2, z_3, z_4, all but z_1 inactive. We conclude the example by showing that the informal meaning assigned to the expression corresponds to $P(s, _)$, i.e., the probability of making a transition from the state where α is being evaluated to each state in $[\mathcal{S}_Y]$.



In what follows we use $\bot$ and $\top$ to represent, respectively, false and true. Through successive applications of Definition 6, according to the structure of α, we have $[\![\alpha]\!]^s_\emptyset = \langle Y, P\rangle$ s.t.:

$[\![E]\!]^s = \{z_2, z_3, z_4\}$, $Y = Y_2 \cup Y_3 \cup Y_4$ and $P = \frac{P_2^Y + P_3^Y + P_4^Y}{3}$
$\langle Y_k, P_k\rangle = [\![\alpha_1]\!]^s_{\rho_k}$, $\rho_k = \{x \mapsto \{z_k\}\}$, for $k = 2, 3, 4$

$[\![E]\!]^s \setminus \rho_2(x) = \{z_3, z_4\}$, $Y_2 = Y_{23} \cup Y_{24}$ and $P_2 = \frac{P_{23}^{Y_2} + P_{24}^{Y_2}}{2}$
$\langle Y_{2i}, P_{2i}\rangle = [\![\alpha_2]\!]^s_{\rho_{2i}}$, $\rho_{2i} = \{x \mapsto \{z_2\}, y \mapsto \{z_i\}\}$, for $i = 3, 4$
$Y_{2i} = Y_{2il} \cup Y_{2ir}$ and $P_{2i} = P_{2il} * P_{2ir}$
$\langle Y_{2il}, P_{2il}\rangle = [\![x.\mathsf{isActive}' = true]\!]^s_{\rho_{2i}}$, $\langle Y_{2ir}, P_{2ir}\rangle = [\![y.\mathsf{isActive}' = true]\!]^s_{\rho_{2i}}$

$[\![E]\!]^s \setminus \rho_3(x) = \{z_2, z_4\}$, $Y_3 = Y_{32} \cup Y_{34}$ and $P_3 = \frac{P_{32}^{Y_3} + P_{34}^{Y_3}}{2}$
$\langle Y_{3i}, P_{3i}\rangle = [\![\alpha_2]\!]^s_{\rho_{32}}$, $\rho_{3i} = \{x \mapsto \{z_3\}, y \mapsto \{z_i\}\}$, for $i = 2, 4$
$Y_{3i} = Y_{3il} \cup Y_{3ir}$ and $P_{3i} = P_{3il} * P_{3ir}$
$\langle Y_{3il}, P_{3il}\rangle = [\![x.\mathsf{isActive}' = true]\!]^s_{\rho_{3i}}$, $\langle Y_{3ir}, P_{3ir}\rangle = [\![y.\mathsf{isActive}' = true]\!]^s_{\rho_{3i}}$

$[\![E]\!]^s \setminus \rho_4(x) = \{z_2, z_4\}$, $Y_4 = Y_{42} \cup Y_{43}$ and $P_4 = \frac{P_{42}^{Y_4} + P_{43}^{Y_4}}{2}$
$\langle Y_{4i}, P_{4i}\rangle = [\![\alpha_2]\!]^s_{\rho_{4i}}$, $\rho_{4i} = \{x \mapsto \{z_4\}, y \mapsto \{z_i\}\}$, for $i = 2, 3$

$Y_{ki} = \{z_k.\mathsf{isActive}, z_i.\mathsf{isActive}\}$, for $k = 2, 3, 4$
$P_{ki}(s_1, s_2) = 1$ if $[\![z_k.\mathsf{isActive}]\!]^{s_2} = [\![z_i.\mathsf{isActive}]\!]^{s_2} = \top$, 0 otherwise

---

$Y_2 = \{z_2.\mathsf{isActive}, z_3.\mathsf{isActive}, z_4.\mathsf{isActive}\}$, $P_2(s_1, s_2) =$
case $[\![z_3.\mathsf{isActive}]\!]^{s_1} = \top$ and $[\![z_4.\mathsf{isActive}]\!]^{s_1} = \bot$:
$= \frac{1}{2}$ if $[\![z_2.\mathsf{isActive}]\!]^{s_2} = [\![z_3.\mathsf{isActive}]\!]^{s_2} = \top$, 0 otherwise
case $[\![z_3.\mathsf{isActive}]\!]^{s_1} = \bot$ and $[\![z_4.\mathsf{isActive}]\!]^{s_1} = \top$:
$= \frac{1}{2}$ if $[\![z_2.\mathsf{isActive}]\!]^{s_2} = [\![z_4.\mathsf{isActive}]\!]^{s_2} = \top$, 0 otherwise
case $[\![z_3.\mathsf{isActive}]\!]^{s_1} = \top$ and $[\![z_4.\mathsf{isActive}]\!]^{s_1} = \top$:
$= 1$ if $[\![z_2.\mathsf{isActive}]\!]^{s_2} = [\![z_3.\mathsf{isActive}]\!]^{s_2} = [\![z_4.\mathsf{isActive}]\!]^{s_2} = \top$, 0 otherwise
case $[\![z_3.\mathsf{isActive}]\!]^{s_1} = \bot$ and $[\![z_4.\mathsf{isActive}]\!]^{s_1} = \bot$:
$= \frac{1}{2}$ if $[\![z_2.\mathsf{isActive}]\!]^{s_2} = \top$ and $[\![z_3.\mathsf{isActive}]\!]^{s_2} = \neg[\![z_4.\mathsf{isActive}]\!]^{s_2}$, 0 otherwise

---

$Y_3 = \{z_2.\mathsf{isActive}, z_3.\mathsf{isActive}, z_4.\mathsf{isActive}\}$ and $P_3(s_1, s_2) =$
case $[\![z_2.\mathsf{isActive}]\!]^{s_1} = \top$ and $[\![z_4.\mathsf{isActive}]\!]^{s_1} = \bot$:
$= \frac{1}{2}$ if $[\![z_3.\mathsf{isActive}]\!]^{s_2} = [\![z_2.\mathsf{isActive}]\!]^{s_2} = \top$, 0 otherwise
case $[\![z_2.\mathsf{isActive}]\!]^{s_1} = \bot$ and $[\![z_4.\mathsf{isActive}]\!]^{s_1} = \top$:
$= \frac{1}{2}$ if $[\![z_3.\mathsf{isActive}]\!]^{s_2} = [\![z_4.\mathsf{isActive}]\!]^{s_2} = \top$, 0 otherwise
case $[\![z_2.\mathsf{isActive}]\!]^{s_1} = \top$ and $[\![z_4.\mathsf{isActive}]\!]^{s_1} = \top$:
$= 1$ if $[\![z_2.\mathsf{isActive}]\!]^{s_2} = [\![z_3.\mathsf{isActive}]\!]^{s_2} = [\![z_4.\mathsf{isActive}]\!]^{s_2} = \top$, 0 otherwise
case $[\![z_2.\mathsf{isActive}]\!]^{s_1} = \bot$ and $[\![z_4.\mathsf{isActive}]\!]^{s_1} = \bot$:
$= \frac{1}{2}$ if $[\![z_3.\mathsf{isActive}]\!]^{s_2} = \top$ and $[\![z_2.\mathsf{isActive}]\!]^{s_2} = \neg[\![z_4.\mathsf{isActive}]\!]^{s_2}$, 0 otherwise

---

$Y_4 = \{z_2.\mathsf{isActive}, z_3.\mathsf{isActive}, z_4.\mathsf{isActive}\}$ and $P_4(s_1, s_2) =$
case $[\![z_2.\mathsf{isActive}]\!]^{s_1} = \top$ and $[\![z_3.\mathsf{isActive}]\!]^{s_1} = \bot$:
$= \frac{1}{2}$ if $[\![z_4.\mathsf{isActive}]\!]^{s_2} = \top$ and $[\![z_2.\mathsf{isActive}]\!]^{s_2} = \top$, 0 otherwise
case $[\![z_2.\mathsf{isActive}]\!]^{s_1} = \bot$ and $[\![z_3.\mathsf{isActive}]\!]^{s_1} = \top$:
$= \frac{1}{2}$ if $[\![z_4.\mathsf{isActive}]\!]^{s_2} = \top$ and $[\![z_3.\mathsf{isActive}]\!]^{s_2} = \top$, 0 otherwise
case $[\![z_2.\mathsf{isActive}]\!]^{s_1} = \top$ and $[\![z_3.\mathsf{isActive}]\!]^{s_1} = \top$:
$= 1$ if $[\![z_4.\mathsf{isActive}]\!]^{s_2} = [\![z_2.\mathsf{isActive}]\!]^{s_2} = [\![z_3.\mathsf{isActive}]\!]^{s_2} = \top$, 0 otherwise
case $[\![z_2.\mathsf{isActive}]\!]^{s_1} = \bot$ and $[\![z_3.\mathsf{isActive}]\!]^{s_1} = \bot$:
$= \frac{1}{2}$ if $[\![z_4.\mathsf{isActive}]\!]^{s_2} = \top$ and $[\![z_2.\mathsf{isActive}]\!]^{s_2} = \neg[\![z_3.\mathsf{isActive}]\!]^{s_2}$, 0 otherwise

Finally, we can deduce that:
$Y = \{z_2.\mathsf{isActive}, z_3.\mathsf{isActive}, z_4.\mathsf{isActive}\}$ and $P(s_1, s_2) =$
case $[\![z_2.\mathsf{isActive}]\!]^{s_1} = \top$ and $[\![z_3.\mathsf{isActive}]\!]^{s_1} = \top$ and $[\![z_4.\mathsf{isActive}]\!]^{s_1} = \top$:
$= \frac{1}{2}$ if $[\![z_2.\mathsf{isActive}]\!]^{s_2} = [\![z_3.\mathsf{isActive}]\!]^{s_2} = [\![z_4.\mathsf{isActive}]\!]^{s_2} = \top$, 0 otherwise
case $[\![z_2.\mathsf{isActive}]\!]^{s_1} = \top$ and $[\![z_3.\mathsf{isActive}]\!]^{s_1} = \top$ and $[\![z_4.\mathsf{isActive}]\!]^{s_1} = \bot$:
$= \frac{1}{3}$ if $[\![z_2.\mathsf{isActive}]\!]^{s_2} = [\![z_3.\mathsf{isActive}]\!]^{s_2} = \top$ and $[\![z_4.\mathsf{isActive}]\!]^{s_2} = \bot$
$= \frac{2}{3}$ if $[\![z_2.\mathsf{isActive}]\!]^{s_2} = [\![z_3.\mathsf{isActive}]\!]^{s_2} = [\![z_4.\mathsf{isActive}]\!]^{s_1} = \top$, 0 otherwise
case $[\![z_2.\mathsf{isActive}]\!]^{s_1} = \top$ and $[\![z_3.\mathsf{isActive}]\!]^{s_1} = \bot$ and $[\![z_4.\mathsf{isActive}]\!]^{s_1} = \top$:
$= \frac{1}{3}$ if $[\![z_2.\mathsf{isActive}]\!]^{s_2} = [\![z_4.\mathsf{isActive}]\!]^{s_2} = \top$ and $[\![z_3.\mathsf{isActive}]\!]^{s_2} = \bot$
$= \frac{2}{3}$ if $[\![z_2.\mathsf{isActive}]\!]^{s_2} = [\![z_3.\mathsf{isActive}]\!]^{s_2} = [\![z_4.\mathsf{isActive}]\!]^{s_2} = \top$, 0 otherwise
case $[\![z_2.\mathsf{isActive}]\!]^{s_1} = \bot$ and $[\![z_3.\mathsf{isActive}]\!]^{s_1} = \top$ and $[\![z_4.\mathsf{isActive}]\!]^{s_1} = \top$:
$= \frac{1}{3}$ if $[\![z_3.\mathsf{isActive}]\!]^{s_2} = [\![z_4.\mathsf{isActive}]\!]^{s_2} = \top$ and $[\![z_2.\mathsf{isActive}]\!]^{s_2} = \bot$
$= \frac{2}{3}$ if $[\![z_2.\mathsf{isActive}]\!]^{s_2} = [\![z_3.\mathsf{isActive}]\!]^{s_2} = [\![z_4.\mathsf{isActive}]\!]^{s_2} = \top$, 0 otherwise
case $[\![z_2.\mathsf{isActive}]\!]^{s_1} = \top$ and $[\![z_3.\mathsf{isActive}]\!]^{s_1} = \bot$ and $[\![z_4.\mathsf{isActive}]\!]^{s_1} = \bot$:
$= \frac{1}{3}$ if $[\![z_2.\mathsf{isActive}]\!]^{s_2} = \top$ and ($[\![z_3.\mathsf{isActive}]\!]^{s_2} = \top$ or $[\![z_4.\mathsf{isActive}]\!]^{s_2} = \top$)
$= 0$ otherwise

case $[\![z_2.\mathsf{isActive}]\!]^{s_1}{=}\bot$ and $[\![z_3.\mathsf{isActive}]\!]^{s_1}{=}\top$ and $[\![z_4.\mathsf{isActive}]\!]^{s_1}{=}\bot$:
$= \frac{1}{3}$ if $[\![z_2 3.\mathsf{isActive}]\!]^{s_2}{=}\top$ and ($[\![z_2.\mathsf{isActive}]\!]^{s_2}{=}\top$ or $[\![z_4.\mathsf{isActive}]\!]^{s_2}{=}\top$)
$= 0$ otherwise
case $[\![z_2.\mathsf{isActive}]\!]^{s_1}{=}\bot$ and $[\![z_3.\mathsf{isActive}]\!]^{s_1}{=}\bot$ and $[\![z_4.\mathsf{isActive}]\!]^{s_1}{=}\top$:
$= \frac{1}{3}$ if $[\![z_4.\mathsf{isActive}]\!]^{s_2}{=}\top$ and ($[\![z_2.\mathsf{isActive}]\!]^{s_2}{=}\top$ or $[\![z_3.\mathsf{isActive}]\!]^{s_2}{=}\top$)
$= 0$ otherwise
case $[\![z_2.\mathsf{isActive}]\!]^{s_1}{=}\bot$ and $[\![z_3.\mathsf{isActive}]\!]^{s_1}{=}\bot$ and $[\![z_4.\mathsf{isActive}]\!]^{s_1}{=}\bot$:
$= \frac{1}{3}$ if $\{[\![z_2.\mathsf{isActive}]\!]^{s_2}, [\![z_3.\mathsf{isActive}]\!]^{s_2}, [\![z_4.\mathsf{isActive}]\!]^{s_2}\} = \{\top, \top, \bot\}$
$= 0$ otherwise

Notice that the probability of making a transition from the state s (where the expression was evaluated) to each state in $[\mathcal{S}_Y]$ is defined by the last case above (server z_1 is active and z_2, z_3 and z_4 are inactive). We can see that the probability of making a transition to any state where z_1 remains active and exactly two servers among z_2, z_3, z_4 are activated is $\frac{1}{3}$. Moreover, the probability of making a transition that involves activating more or less than two servers or deactivating z_1 is 0. This is consistent with the informal meaning assigned to the expression: exactly two servers are activated and that all pairs of distinct inactive servers have the same probability of being activated.

Proof of Proposition 1

If $[\![\alpha]\!]^s_\rho = \langle Y, P : [\mathcal{S}_Y] \times [\mathcal{S}_Y] \to [0,1]\rangle$, then P is a transition probabilistic matrix, i.e., $\sum_{s_2 \in [\mathcal{S}_Y]} P(s_1, s_2) = 1$, for every $s_1 \in [\mathcal{S}_Y]$.

Proof. The proof proceeds by induction in the structure of expressions α:

case $(x.p' = t)$: In this case the result follows from the fact that the term t uniquely defines the values of all properties in Y in the next state. That is, for every s_1 in $[\mathcal{S}_Y]$, there is a single state s such that $[\![y]\!]^s = [\![t]\!]^{s_1}_{\rho_c}$, for every $y \in Y$. Hence, the probability of transition to s is 1 and is 0 for any other state different from s, and hence, the sum $P(s_1, s_2)$ for all states s_2 is 1.

case (**forall** $x : \epsilon \mid \alpha$) : In this case the result follows immediately from the hypothesis of induction.

case (**foreach** $x : \epsilon \mid \alpha$) : Let $C{=}[\![\epsilon]\!]^s$. If $|C| = 0$, then $Y = \emptyset$. This implies that $[\mathcal{S}_Y] = \emptyset$ and, hence, the result is vacuosly true. Otherwise, $Y = \bigcup_{c \in C} Y_c$ and

$$\sum_{s_2 \in [\mathcal{S}_Y]} P(s_1, s_2) = \sum_{s_2 \in [\mathcal{S}_Y]} \sum_{c \in C} \frac{1}{|C|} \cdot P_c^Y(s_{1|Y_c}, s_{2|Y_c})$$

$$= \sum_{s_2 \in [\mathcal{S}_Y]} \sum_{c \in C} \frac{1}{|C|} \cdot P_c(s_{1|Y_c}, s_{2|Y_c}) \cdot (s_1 =_{Y \setminus Y_c} s_2)$$

$$= \sum_{c \in C} \frac{1}{|C|} \cdot \sum_{s_2 \in [\mathcal{S}_Y]} P_c(s_{1|Y_c}, s_{2|Y_c}) \cdot (s_1 =_{Y \setminus Y_c} s_2)$$

$$= \sum_{c \in C} \frac{1}{|C|} \cdot \sum_{s_2^c \in [\mathcal{S}_{Y_c}]} P_c(s_{1|Y_c}, s_2^c)$$

where $(s_1 {=_{Y'}} s_2)$ denotes 1 if $[\![y]\!]^{s_2} = [\![y]\!]^{s_1}$, for every $y \in Y'$, and 0 otherwise. Using the hypothesis of induction for each $[\![\alpha]\!]^s_{\rho_c}$, with $c \in C$, we reach the result.

case (**foreach** $x : \epsilon$ **minus** $D \mid \alpha$) : Similar to the previous case.

case $\{\alpha_1 \& \ldots \& \alpha_n\}$: If $n = 1$, then the result follows immediately from the hypothesis of induction. Without loss of generality, we will prove for the case $n = 2$. If Y_1, Y_2 are not disjoint, as before, the result is vacuosly true. Otherwise, we have

$$\sum_{s_2 \in [\mathcal{S}_Y]} P(s_1, s_2) = \sum_{s_2 \in [\mathcal{S}_Y]} P_1(s_{1|Y_1}, s_{2|Y_1}) \cdot P_2(s_{1|Y_2}, s_{2|Y_2})$$

$$= \sum_{s_2^1 \in [\mathcal{S}_{Y_1}]} (P_1(s_{1|Y_1}, s_2^1) \cdot \sum_{s_2^2 \in [\mathcal{S}_{Y_2}]} P_2(s_{1|Y_2}, s_2^2))$$

The last equality holds because, since Y_1, Y_2 are disjoint, $[\mathcal{S}_Y]$ is isomorphic to $[\mathcal{S}_{Y_1}] \times [\mathcal{S}_{Y_2}]$. Using the hypothesis of induction for each $[\![\alpha_i]\!]^s_\rho$, we reach the result.

case $\{[p_1]\alpha_1 + \cdots + [p_n]\alpha_n\}$: In this case $Y = \bigcup_{i=1,..,n} Y_i$ and

$$\sum_{s_2 \in [\mathcal{S}_Y]} P(s_1, s_2) = \sum_{s_2 \in [\mathcal{S}_Y]} \sum_{i=1}^{n} p_i \cdot P_i^Y(s_{1|Y_i}, s_{2|Y_i})$$

$$= \sum_{s_2 \in [\mathcal{S}_Y]} \sum_{i=1}^{n} p_i \cdot P_i(s_{1|Y_i}, s_{2|Y_i}) \cdot (s_1 =_{Y \setminus Y_i} s_2)$$

$$= \sum_{i=1}^{n} p_i \cdot \sum_{s_2^i \in [\mathcal{S}_{Y_i}]} P_i(s_{1|Y_i}, s_2^i)$$

Using the hypothesis of induction for each $[\![\alpha_i]\!]^s_\rho$ and the fact that $\sum_{i=1}^n p_i = 1$, we reach the result.