MODELING AND ANALYSIS OF INFORMATION SYSTEMS, VOL. 28, NO. 1, 2021
journal homepage: www.mais-journal.ru

COMPUTING METHODOLOGIES AND APPLICATIONS

Online Testing of Dynamic Reconfigurations w.r.t. Adaptation Policies

F. Dadeau[1], J. Gros[1], O. Kouchnarenko[1]

DOI: 10.18255/1818-1015-2021-1-52-73

[1]University Bourgogne Franche-Comté, CNRS, FEMTO-ST Institute, 15B avenue des Montboucons, 25030 Besançon, Cedex, France.

MSC2020: 93A30, 68Q60
Research article
Full text in Russian

Received March 3, 2021
After revision March 10, 2021
Accepted March 12, 2021

Self-adaptation of complex systems is a very active domain of research with numerous application domains. Component systems are designed as sets of components that may reconfigure themselves according to adaptation policies, which describe needs for reconfiguration. In this context, an adaptation policy is designed as a set of rules that indicate, for a given set of configurations, which reconfiguration operations can be triggered, with fuzzy values representing their utility. The adaptation policy has to be faithfully implemented by the system, especially w.r.t. the utility occurring in the rules, which are generally specified for optimizing some extra-functional properties (e.g. minimizing resource consumption).

In order to validate adaptive systems' behaviour, this paper presents a model-based testing approach, which aims to generate large test suites in order to measure the occurrences of reconfigurations and compare them to their utility values specified in the adaptation rules. This process is based on a usage model of the system used to stimulate the system and provoke reconfigurations. As the system may reconfigure dynamically, this online test generator observes the system responses and evolution in order to decide the next appropriate test step to perform. As a result, the relative frequencies of the reconfigurations can be measured in order to determine whether the adaptation policy is faithfully implemented. To illustrate the approach the paper reports on experiments on the case study of platoons of autonomous vehicles.

Keywords: component system; adaptation policy; online testing; usage model

INFORMATION ABOUT THE AUTHORS

Frederic Dadeau	orcid.org/0000-0003-0794-5819. E-mail: frederic.dadeau@univ-fcomte.fr Associate professor, Ph.D in Computer Science.
Jean-Philippe Gros	orcid.org/0000-0002-5159-9442. E-mail: jean-philippe.gros@univ-fcomte.fr Ph.D. candidate.
Olga Kouchnarenko correspondence author	orcid.org/0000-0003-1482-9015. E-mail: olga.kouchnarenko@univ-fcomte.fr Professor, Ph.D. in Computer Science.

Funding: RFBR, project No 17-07-01566.

For citation: F. Dadeau, J. Gros, and O. Kouchnarenko, "Online Testing of Dynamic Reconfigurations w.r.t. Adaptation Policies", *Modeling and analysis of information systems*, vol. 28, no. 1, pp. 52-73, 2021.

МОДЕЛИРОВАНИЕ И АНАЛИЗ ИНФОРМАЦИОННЫХ СИСТЕМ, ТОМ 28, № 1, 2021
сайт журнала: www.mais-journal.ru

COMPUTING METHODOLOGIES AND APPLICATIONS

Онлайн тестирование динамических реконфигураций по отношению к политикам адаптации

Ф. Дадо[1], Ж. Гро[1], О. Б. Кушнаренко[1]

DOI: 10.18255/1818-1015-2021-1-52-73

[1]Университет Бургундия Франш-Комтэ, НЦНИ, Институт ФЕМТО-СТ, 15Б Проспект Монбукон, 25030 Безансон ГПС, Франция.

УДК 519.7
Научная статья
Полный текст на русском языке

Получена 3 марта 2021 г.
После доработки 10 марта 2021 г.
Принята к публикации 12 марта 2021 г.

Самоадаптация сложных систем является активной областью теоретических и прикладных исследований, имеющей чрезвычайно широкий спектр применения.

Компонентно-ориентированные адаптивные системы разрабатываются на базе компонент, которые могут перенастроиться в соответствии с политиками адаптации, описывающими потребности в реконфигурировании. В этом контексте политика адаптации представляет собой набор правил, которые указывают для данного множества конфигураций, какие операции реконфигурирования могут быть инициированы, при этом их полезность представлена нечеткими значениями. Правила обычно разрабатываются для оптимизации некоторых нефункциональных свойств, например, минимизации потребления ресурсов, поэтому реализация системы с политиками адаптации должна быть точной, особенно по отношению к описанной в правилах полезности реконфигурирования.

С целью валидации поведения адаптивных систем в этой статье представлен модельно-ориентированный подход к тестированию, который направлен на создание больших наборов тестов для оценки случаев реконфигурирования и сравнения частоты этих случаев со значениями полезности, описанными в правилах адаптации. Этот процесс основан на модели использования системы в ее среде, для стимулирования ее реконфигурирований. Поскольку система может динамически изменять свою архитектуру, этот генератор тестов наблюдает за откликами системы на события и ее изменениями в режиме онлайн, чтобы решить каким будет следующий подходящий шаг теста. В результате относительные частоты реконфигурирований могут быть измерены, чтобы определить, правильно ли реализована политика адаптации. Чтобы проиллюстрировать предложенный подход, статья описывает эксперименты по моделированию поведения колонн автономных машин.

Ключевые слова: компонентно-ориентированные системы; политика адаптации; онлайн тестирование; модель использования

ИНФОРМАЦИЯ ОБ АВТОРАХ

Фредерик Дадо	orcid.org/0000-0003-0794-5819. E-mail: frederic.dadeau@univ-fcomte.fr доцент информатики.
Жан-Филипп Гро	orcid.org/0000-0002-5159-9442. E-mail: jean-philippe.gros@univ-fcomte.fr аспирант.
Ольга Борисовна Кушнаренко автор для корреспонденции	orcid.org/0000-0003-1482-9015. E-mail: olga.kouchnarenko@univ-fcomte.fr профессор информатики, доктор наук.

Финансирование: РФФИ, проект № 17-07-01566.

Для цитирования: F. Dadeau, J. Gros, and O. Kouchnarenko, “Online Testing of Dynamic Reconfigurations w.r.t. Adaptation Policies”, *Modeling and analysis of information systems*, vol. 28, no. 1, pp. 52-73, 2021.

Введение

Контекст и мотивация Политики адаптации используются для управления адаптивными системами, указывая, когда могут быть инициированы операции реконфигурирования. В этой статье политики адаптации формально определены как наборы правил, которые указывают для данного множества состояний системы, также называемых конфигурациями, какие операции реконфигурирования могут быть инициированы. Таким образом, правила адаптации используются для определения корректного поведения системы, описывая для каждого реконфигурирования его область, предварительное условие и полезность. Предварительное условие и область могут содержать временные шаблоны [1], при этом область ограничивает применение реконфигурирования (например, на множестве конфигураций или при наступлении событий). Для указания полезности реконфигурирования, т.е. необходимости его активации для системы, используется нечеткое значение (например, high, medium, low), принятое в нечеткой логике. Например, можно указать, что после входа автономного автомобиля в туннель, если его запас энергии ограничен, то очень полезно отключить его компонент GPS.

Как следствие использования такого необязательного характера реконфигурирования возможны различные реализации каждой политики адаптации при условии, что результирующие исполнения адаптивной системы не нарушают никаких временных свойств. Верификация и валидация адаптивных систем является сложной задачей из-за этих множественных, хотя и корректных реализаций системы, подчиняющейся политикам адаптации с временными свойствами. Эти дополнительные артефакты – временные свойства и политики адаптации, дают возможность убедиться в корректности выполнений системы. Например, слежение за временными свойствами при выполнении системы позволяет удостовериться, что никакого нарушения этих свойств не обнаружено [2]. Однако для политик адаптации вопрос об установлении точности их соблюдения (их валидация) не является в настоящее время достаточно изученным. Это объясняется главным образом тем, что политики адаптации не носят предписывающий характер.

Как правило, методы валидации систем опираются на модели их поведения, используемые в качестве опоры для тестирования. Для адаптивных систем с политиками адаптации, это подразумевало бы необходимость выбора некоторой опорной реализации системы под политиками адаптации и требовало бы от системы того же самого выбора для соблюдения политики, а это слишком ограничено в случае описанных выше политик. В этой статье предложен подход для валидации реализации политик адаптации, который использует только сами политики в качестве опоры. Поскольку политики адаптации описывают потребности реконфигурирования по отношению к среде, в которой выполняется система, вместо модели поведения системы мы предлагаем использовать модель среды ее выполнения. Это позволяет принимать во внимание наступление событий в среде с последующими реконфигурированиями, вызванными этими событиями. Для каждого компонента такая модель среды описывает, какие события, в частности внешние, могут стимулировать реконфигурирования при его использовании. Поэтому она называется также моделью использования, и, как правило, она значительно меньше, чем модель самой системы. В предыдущей работе [3] описан подход к валидации системы по отношению к политике адаптации путем проверки того, что реконфигурирования, которые инициированы во время выполнения, соответствуют реконфигурированиям, разрешенным политикой адаптации.

Новые результаты Данная работа продолжает рассмотрение вопроса о соблюдении полезности правил политик адаптации при исполнении системы. Для валидации предлагается провести статистический анализ частот реконфигурирований. Для того, чтобы иметь значительное количество выполнений с осуществлением реконфигурирований, в статье предложено автоматически генерировать множество тестов. Тесты получены случайным исследованием моделей среды компонентов, описывающих, как стимулировать адаптивную компонентно-ориентированную систему,

в частности, посредством внешних событий. Поскольку реконфигурирования могут происходить без управления со стороны специалиста по тестированию, процесс формирования теста происходит в режиме онлайн: предлагаемый нами генератор тестов наблюдает реконфигурирования системы в ответ на события в ее среде и вычисляет следующий этап теста, который должен быть выполнен.

Одним из методов валидации на базе моделей является использование критериев покрытия тестами возможного поведения системы. Они обеспечивают, что случаи тестов дают хорошее представление о возможностях системы, покрывая их. Следуя этому методу, данная работа опирается на результаты [3], где были определены критерии тестового покрытия с акцентом на два артефакта, а именно: (а) политики адаптации и (б) множество временных свойств, которым система должна удовлетворять. Их можно использовать для иллюстрации различных сценариев выполнения. Эти артефакты позволяют обеспечить разнообразность трасс выполнений, которые будут регистрироваться. Таким образом обеспечивается уверенность в результатах анализа частот реконфигурирований. Данная работа содержит следующие новые материалы: (1) *онлайн* процесс формирования тестов, включающий модель среды использования системы, который направлен на создание больших множеств тестовых случаев, удовлетворяющих критериям покрытия, определенным в предыдущей работе; (2) мера встречаемости операций реконфигурирования, определенная на основе частотного анализа и направленная на подтверждение политик адаптации; (3) экспериментальное подтверждение этого процесса на примере автомобилей и колонн автомобилей на дороге.

Данная статья организована следующим образом. Раздел 1 содержит необходимые сведения о компонентно-ориентированных системах и политиках адаптации для них. Процесс генерации тестов, который опирается на модель использования среды системы описан в разделе 2. Основанная на частотном анализе оценка нечетких значений полезности реконфигурирований представлена в разделе 3. Результаты экспериментов по моделированию поведения колонны автономных машин даны в разделе 4. Наконец, библиографические заметки и заключение содержатся в разделе 5.

1. Пример и необходимые определения

1.1. Адаптивные системы на примере компонентно-ориентированной сети VANet

Пример 1. *Начнем с описания примера компонентно-ориентированной системы сети VANet (Vehicular Ad-Hoc Network), который показан на рисунке 1. Сеть VANet состоит из машин, которые либо находятся в одиночном режиме, либо организованы в колонны. Каждая колонна возглавляется лидирующей машиной. Любая машина в одиночном режиме (соло) может попросить присоединиться к колонне или принять решение о создании новой колонны с другой машиной в режиме соло. Каждая машина в колонне может попросить выйти из нее поскольку либо она добралась до места назначения, либо ей необходимо восполнить свои энергические ресурсы. Колонна может поменять лидера, если другая машина имеет большую автономность, либо из-за более удаленного пункта назначения другой машины. В окружении системы могут происходить внешние события. Например, новая машина может выехать на дорогу, или водитель может принять решение покинуть колонну из-за нового места назначения.*

Опираясь на этот пример, введем понятие компонентно-ориентированных систем. Как правило, компоненты – это объекты, которые можно объединять для создания сложных систем. Рассматриваемые компонентно-ориентированные системы являются иерархическими, имеющими два типа компонентов. Примитивные или базовые компоненты предоставляют информацию или услуги, в то время как составные компоненты содержат другие компоненты. Требуемые и предоставляемые интерфейсы являются точками взаимодействия между компонентами. Компонент реализует работу или сервис и поставляет их через предоставляемый интерфейс. Требуемый интерфейс – это интерфейс, необходимый компоненту для работы. Составные компоненты могут делегировать свои интерфейсы внутренним компонентам. Определения компонентов, интерфейсов, переменных, привязок для их совместимой сборки и прочих элементов модели могут быть найдены в [1, 4].

Состояние компонентно-ориентированной системы, также называемое конфигурацией, представляет собой набор вышеупомянутых архитектурных элементов (компонентов, интерфейсов и параметров) вместе с их типами и отношениями, чтобы структурировать и связать их. Реконфигурирование может быть рассмотрено как переход от одной конфигурации к другой.

Компоненты могут быть реализованы независимо друг от друга. Компонент может быть создан и инициирован несколько раз, например, как компоненты колонн и машин.

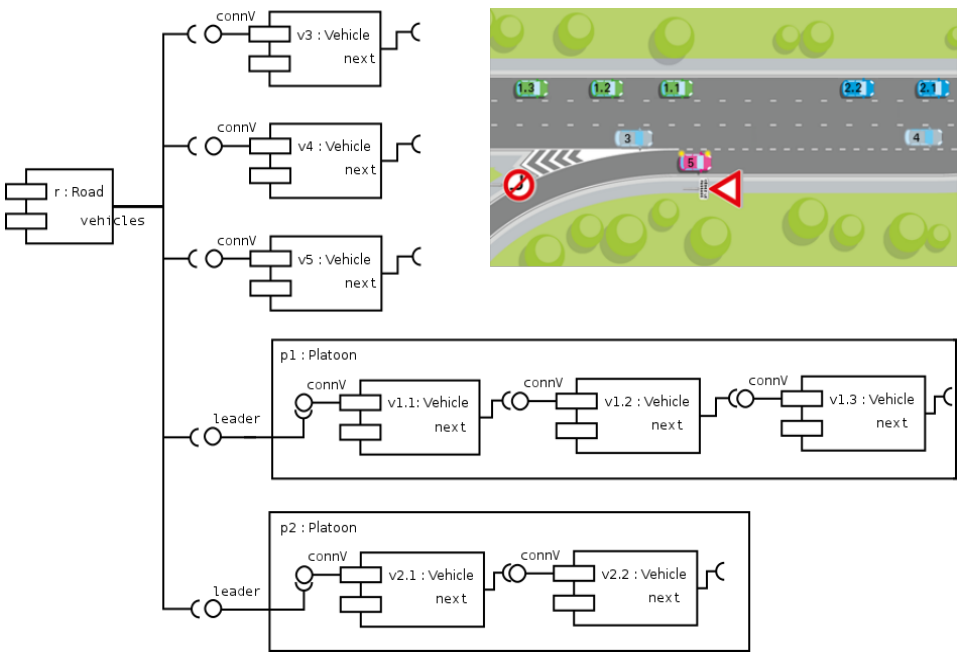

Fig. 1. Component architecture (a) vs. the considered VANet system state (b)

Рис. 1. Компонентная архитектура (a) напротив рассматриваемого состояния сети VANet (b)

Пример 2. *На рисунке 1 слева (a) изображено состояние VANet, соответствующее ситуации на рисунке справа (б). Составной дорожный компонент (Road) содержит компоненты машин (Vehicles), которые могут быть в колоннах (1.X, 2.X) или соло (3, 4, 5). Компоненты машин (Vehicles) соединены друг с другом через интерфейсы, позволяющие обмениваться информацией.*

Политики адаптации для таких систем рассматриваются как артефакты, которые описывают потребности адаптации системы, не являющиеся, однако, обязательными. Например, колонна *может* поменять лидера, когда он не обладает достаточными энергетическими ресурсами, либо когда у другого транспортного средства больше этих ресурсов, чем у лидера, и оно направляется к более удаленному пункту назначения.

1.2. Модели

Пусть $C = \{c, c_1, c_2, \ldots\}$ – множество конфигураций. Определим также множество CP свойств конфигураций, которые являются ограничениями на компоненты и отношения между ними. В частности, эти свойства используются для определения непротиворечивых конфигураций. Например, они используются для описания свойств глобальной инвариантности (например, на компоненты, интерфейсы и их привязки), которым архитектура на базе компонентов должна удовлетворять. Интерпретация $l : C \longrightarrow CP$ дает самую широкую конъюнкцию $cp \in CP$, которая является истинной на $c \in C$, что характеризует текущую конфигурацию наиболее точно.

Определим множество $\mathcal{R}$ операций реконфигурирования, которые делают компонентную архитектуру динамично изменяющейся. Реконфигурирования представляют собой комбинации примитивных операций, таких как инстанциирование/разрушение компонентов, включение/выключение компонентов, привязка/отмена привязки интерфейсов компонентов, запуск/остановка компонентов и т.д. Пусть $\mathcal{R}_{run} = \mathcal{R} \cup \Theta \cup \{run\}$ – множество операций, где $\mathcal{R}$ – конечное множество операций реконфигурирования, Θ – множество операций, запускаемых внешними событиями ($\mathcal{R} \cap \Theta = \emptyset$), и run – название универсального действия, представляющего все текущие операции вне реконфигурирования компонентно-ориентированной системы. Будем считать, что внешние события фиксируются системой и немедленно обрабатываются посредством запуска внутренних методов из Θ, как предложено в [5].

Определение 1 (Маркированная система с переходами (Labelled Transition System (LTS))). *Маркированная система с переходами – это структура $S = \langle C, C^0, \mathcal{R}_{run}, \longrightarrow, l \rangle$, где C множество конфигураций, $C^0 \subseteq C$ – множество начальных конфигураций, $\longrightarrow \subseteq C \times \mathcal{R}_{run} \times C$ – отношение реконфигурирования, и $l : C \longrightarrow CP$ всюду определенная функция интерпретации конфигураций.*

Обозначим $c \xrightarrow{ope} c'$, когда переход $(c, ope, c') \in \longrightarrow$. Переход $c \xrightarrow{ope} c'$ также называется *шагом.*

Определение 2 (Путь). *Для модели S определим путь (реконфигурирования) σ в S как последовательность шагов $c_0 \xrightarrow{ope_0} c_1, c_1 \xrightarrow{ope_1} c_2, ...,$ таких что $\forall i \geq 0.\ (c_i, ope_i, c_{i+1}) \in \longrightarrow$. Для данного пути σ, его трассой является слово $tr(\sigma) = ope_0.ope_1. ... ope_i ...$, составленное из имен операций в σ.*

Пусть c_i или $\sigma(i)$ обозначает i-ю конфигурацию на σ, начальную для i-го шага. Обозначение σ_i используется для суффикса, начинающегося от $\sigma(i)$, и σ_i^j обозначает отрезок пути между $\sigma(i)$ и $\sigma(j)$. Пусть Σ_S обозначает множество путей S, а Σ^f ($\subseteq \Sigma$) – множество конечных путей. Конфигурация c' достижима из c, когда существует путь $\sigma = c_0 \xrightarrow{ope_0} c_1, ... c_{n-1} \xrightarrow{ope_{n-1}} c_n$ в Σ^f такой, что $c = c_0$ и $c' = c_n$ с $n \geq 0$. Выполнение – это путь σ в Σ такой, что $\sigma(0) \in C^0$.

1.3. Политики адаптации

Политики адаптации состоят из правил, которые указывают какие операции реконфигурирования с соответствующим уровнем полезности могут быть инициированы для данного множества конфигураций. Операции реконфигурирования в политиках адаптации охраняются событиями, которые могут либо использовать свойства конфигураций, либо включать свойства временной логики. В этом разделе приведены элементы темпоральной логики линейного времени, основанной на временных шаблонах с областями их применения [1]. Эта логика, названная FTPL[1], используется далее в политиках адаптации.

Временные шаблоны FTPL используются для описания свойств над трассами выполнения, на которых ожидаются операции реконфигурирования. Обозначим $Prop_{FTPL}$ множество формул FTPL, соответствующих грамматике FTPL на рис. 2. В дополнение к свойствам конфигурации (*cp*) в *CP* из раздела 1, FTPL содержит внешние события и события из операций реконфигурирования, временные свойства (*temp*) вместе со свойствами трасс (*trace*), интегрированными во временные свойства.

Семантика FTPL была представлена в [4]. Она является классической для свойств конфигураций, например, как в PLTL[2]. Для других временных шаблонов она использует области или рамки применения (до, после, пока) свойств трасс или возникновения событий. Предположим, что внешние события (такие как события в [5]) происходят немедленно и могут рассматриваться как вызовы

[1]FTPL происходит от слияния TPL (Temporal Pattern Language) и "F"для логики первого порядка (First Order Logic) для ограничений на непротиворечивость при сборки компонентов.

[2]PTPL – Propositional Linear Temporal Logic.

<FTPL>	::=	*<temp>* | *<events>* | *cp*
<temp>	::=	**after** *<events>* *<temp>*
	|	**before** *<events>* *<trace>*
	|	*<trace>* **until** *<events>* | *<trace>*
<trace>	::=	**always** *cp* | **eventually** *cp*
	|	*<trace>* ∧ *<trace>*
	|	*<trace>* ∨ *<trace>*
<events>	::=	*<event>*,*<events>*
	|	*<event>*
<event>	::=	*ope* **normal** | *ope* **exceptional**
	|	*ope* **terminates** | *ext*

Fig. 2. FTPL syntax

Рис. 2. Синтаксис FTPL

методов, выполняемых внешними датчиками при обнаружении изменений в окружающей систему среде. Для каждого внешнего события ext, которое может произойти на пути выполнения σ, определим

1) предварительное условие cp_{ext}, представляющее собой формулу логики первого порядка над параметрами, указанными в вызове метода, соответствующего ext, и
2) утверждение $happens_\sigma$ со значением в $\{\top, \bot\}$.

Следуя за семантикой вычисления событий (Event Calculus [6, 7]) определим $happens_\sigma(cp_{ext}, i, j) = \top$, если существует по крайней мере одно вхождение события ext между i-м и j-м состояниями на пути σ такое, что $cp_{ext} = \top$; в противном случае будем считать, что $happens_\sigma(cp_{ext}, i, j) = \bot$.

Определение 3 (Семантика FTPL). *Пусть $\sigma \in \Sigma$. Семантика FTPL определена на $\Sigma \times Prop_{FTPL} \longrightarrow \mathbb{B}_2$ индуктивно по форме формул следующим образом:*

Для свойств конфигураций:		
$\sigma(i) \models cp$	, если	$l(\sigma(i)) \Rightarrow cp$
Для событий:		
$\sigma(i) \models ope$ **normal**	, если	$i > 0 \wedge \sigma(i-1) \neq \sigma(i) \wedge \sigma(i-1) \xrightarrow{ope} \sigma(i) \in \longrightarrow$
$\sigma(i) \models ope$ **exceptional**	, если	$i > 0 \wedge \sigma(i-1) = \sigma(i) \wedge \sigma(i-1) \xrightarrow{ope} \sigma(i) \in \longrightarrow$
$\sigma(i) \models ope$ **terminates**	, если	$\sigma(i) \models ope$ **normal** $\vee\ \sigma(i) \models ope$ **exceptional**
$\sigma(i) \models ext$	, если	$happens_\sigma(cp_{ext}, i, j) = \top$
$\sigma(i) \models event, events$	, если	$\sigma(i) \models event \vee \sigma(i) \models events$
Для свойств трасс:		
$\sigma \models$ **always** cp	, если	$\forall i.(i \geqslant 0 \Rightarrow \sigma(i) \models cp)$
$\sigma \models$ **eventually** cp	, если	$\exists i.(i \geqslant 0 \wedge \sigma(i) \models cp)$
$\sigma \models trace_1 \wedge trace_2$	, если	$\sigma \models trace_1 \wedge \sigma \models trace_2$
$\sigma \models trace_1 \vee trace_2$	, если	$\sigma \models trace_1 \vee \sigma \models trace_2$
Для временных свойств:		
$\sigma \models$ **after** $event\ temp$	, если	$\forall i.(i \geqslant 0 \wedge \sigma(i) \models event \Rightarrow \sigma_i \models temp)$
$\sigma \models$ **before** $event\ trace$	, если	$\forall i.(i > 0 \wedge \sigma(i) \models event \Rightarrow \sigma_0^{i-1} \models trace)$
$\sigma \models trace$ **until** $event$	, если	$\exists i.(i > 0 \wedge \sigma(i) \models event \wedge \sigma_0^{i-1} \models trace)$

Модель реконфигурирования S удовлетворяет свойству $\phi \in Prop_{FTPL}$, что обозначается как $S \models \phi$, *если* $\forall \sigma, (\sigma \in \Sigma_S \wedge \sigma(0) \in C^0 \Rightarrow \sigma \models \phi)$.

Подробная формальная семантика FTPL может быть найдена в [4]. Приведенное выше напоминание FTPL направлено на иллюстрацию временных шаблонов, интегрированных в политики адаптации. Напомним, что эти шаблоны делают политики более выразительными, чем представленные в [8], использующие только свойства состояний и инвариантов над конфигурациями.

Пример 3. *Рассмотрим свойство FTPL для системы VANet:*

$\varphi 1$: ***after*** Join normal ***always*** $Battery \geq 5 \wedge Dist > 2$ ***until*** Quit normal

Это свойство описывает, что после присоединения автомобиля к колонне его автономность и оставшееся расстояние до цели превышают соответственно 5% и 2 км, до тех пор пока он не покинет колонну. Это относится к каждому транспортному средству.

Определим теперь политики адаптации с временными шаблонами, используя как базу определения в [8] и упрощая обозначения из [4].

Определение 4 (Политика адаптации)**.** *Пусть S – маркированная система с переходами LTS с ее реконфигурированиями $\mathcal{R} \cup \Theta$, CP – множество свойств конфигурации, маркирующих ее состояния, и $Ftype$ – конечное множество нечетких типов. Политика адаптации определяется как $A = \langle R_N, R_R \rangle$, где:*

- *$R_N \subseteq \mathcal{R} \cup \Theta$ – непустое конечное множество операций реконфигурирования,*
- *R_R – конечное множество правил адаптации с элементами $(b, g, ir) \in Prop_{FTPL} \times CP \times I$, где $ir \in I \subseteq R_N \times Ft$ – пара, связывающая нечеткое значение полезности f из области $Ft \in Ftype$ с операцией реконфигурирования $ope \in R_N$.*

Отметим, что I является отношением, поскольку инженер может связать различные нечеткие значения с одним реконфигурированием: иногда полезность высокая (`high`), иногда она низкая (`low`). Кроме того, некоторые операции реконфигурирования из $\mathcal{R} \cup \Theta \setminus R_N$ могут не затрагиваться разработанными политиками и не иметь связанных значений полезности.
Каждое правило адаптации в R_R строится с использованием нескольких ключевых слов: **when** b **if** g **then utility of** ope **is** f. FTPL свойство b после ключевого слова `when` определяет временные рамки операции реконфигурирования шаблоном FTPL. В этой части ключевое слово `and` используется для построения сложных шаблонов. Затем после ключевого слова `if` описывается предикат g на конфигурации системы. Он определяет множество конфигураций в пределах временных рамок, где может быть инициировано реконфигурирование. После этого, чтобы привязать полезность к операции реконфигурирования, $ope \in R_N$ ассоциируется с ее полезностью $f \in Ft$ с помощью ключевых слов `then utility of` и `is`. Полезность определяется нечетким значением (например, `high, medium, low`).

when after *Join normal* until *Quit normal* **and** *VehicleId.battery < 33*
if *state = leader* **then utility of** *PassRelay* **is** *high*

when after *Join normal* until *Quit normal* **and** *VehicleId.battery > Leader.battery*
if *state = platooned* **then utility of** *GetRelay* **is** *medium*

Пример 4. *Рассмотрим два правила адаптации из политики для каждого транспортного средства, включающие реконфигурирования PassRelay и GetRelay. Интуитивно вышеуказанные правила применяются ко всем транспортным средствам и используются для определения тех моментов, когда может произойти смена между лидером и другим автомобилем в колонне. В первом случае реконфигурирование PassRelay происходит, когда батарея лидера недостаточно заряжена. Во втором случае реконфигурирование GetRelay инициируется, когда батарея транспортного средства имеет больше ресурсов, чем у лидера. Заметим, что добавление/удаление компонента не является обязательным, а скорее предлагается с некоторым значением полезности (например, из множества $Ft = \{$ high, medium, low $\}$). Следовательно, нет никакой гарантии, что система в конечном итоге выполнит рассматриваемую операцию реконфигурирования.*

Определим теперь влияние политики адаптации на поведение системы S. Для S и конечного множества AP политик адаптации определим маркированную систему с переходами $S \triangleleft AP$, которая обозначает S в присутствии политик адаптации из множества AP.

Определение 5 (LTS с политиками адаптации)**.** *Ограничение модели S политиками адаптации AP определяется как маркированная система с переходами $S \triangleleft AP = \langle C_{\triangleleft AP}, C^0_{\triangleleft AP}, \mathcal{R}_{run}, \longrightarrow, l \rangle$, где $C_{\triangleleft AP}$ – наименьшее множество конфигураций такое, что если $c \in C$ и $A \in AP$, то $c_{\triangleleft A} \in C_{\triangleleft AP}$, $\mathcal{R}_{run} \cap (\cup_{A \in AP} R_N) \neq$*

$\emptyset, l : \mathcal{C}_{\lhd AP} \longrightarrow CP$ является всюду определенной функцией интерпретации конфигураций, и для каждой операции $ope \in \mathcal{R}_{run}$ отношение реконфигурирования $\longrightarrow \in \mathcal{C}_{\lhd AP} \times \mathcal{R}_{run} \times \mathcal{C}_{\lhd AP}$ определено как наименьшее множество элементов $(c_\lhd, ope, c'_{\lhd A})$, удовлетворяющих следующим правилам:

$$[ACT1] \quad \frac{c \xrightarrow{ope} c'}{c_{\lhd A} \xrightarrow{ope} c'_{\lhd A}} \quad (ope \in \bigcup_{A \in AP} R_N) \wedge b \wedge g$$

$$[ACT2] \quad \frac{c \xrightarrow{ope} c'}{c_{\lhd A} \xrightarrow{ope} c'_{\lhd A}} \quad ope \notin \bigcup_{A \in AP} R_N \ .$$

Это определение означает, что переходы S под AP являются результатом выполнения операций реконфигурирования в соответствии с политиками адаптации (правило *[ACT1]*) или реконфигурирований, незатронутых политиками адаптации (правило *[ACT2]*).

Что касается оценки условия правила *[ACT1]*, а именно его части b, то это может быть сделано децентрализовано, используя прогрессивную семантику, как в [2, 9].

Политики адаптации должны точно соблюдаться системой, особенно по отношению к значениям полезности, встречающимся в правилах, которые обычно определяются для оптимизации некоторых нефункциональных свойств (например, для минимизации потребления ресурсов). С этой целью могут использоваться различные соотношения, например, отношение уточнения [10], симуляции [11], правильного исполнения [12] и т. д. Однако установление этих соотношений для сравнения реализаций или спецификаций в соответствии с политиками адаптации может быть трудной проблемой, которая становится более сложной при рассмотрении событий из окружения системы. Эта проблема является, как правило, неразрешимой для систем с бесконечным числом состояний. Вместо этого в следующем разделе мы предлагаем методологию, основанную на модели среды использования для валидации реализаций адаптивных систем с политиками адаптации.

2. Онлайн-тестирование с моделью использования

В этом разделе сначала обосновывается использование онлайн тестирования, затем описываются модели использования и алгоритм генерации тестов, необходимые для вычисления случаев тестов.

2.1. Онлайн процесс генерации тестов

Адаптивные системы реагируют на внешние события в соответствии с политиками адаптации, которые предоставляют рекомендации для выполнения реконфигурирований системы. Существует множество различных корректных реализаций для системы при заданных политиках адаптации. Назовем частотой реконфигурирований отношение между количеством выполненных реконфигурирований и количеством возможностей их применения. При рассмотрении трасс выполнения и относительных частот реконфигурирований, если операция реконфигурирования с высокой полезностью имеет более низкую относительную частоту, чем операция с низкой полезностью, это означает, что либо реализация системы неправильно учитывает значение полезности, или же необходимо изменить его определение в политике адаптации. Поскольку возможны различные реализации системы при заданных политиках адаптации, иметь единственную модель всех таких реализаций представляется проблематичным: это потребовало бы сделать выбор в отношении поведения системы и заставило бы различные реализации соответствовать этому выбору.

Чтобы избежать описания моделей реализации систем (множество LTS) и их валидации по отношению к спецификации (LTS с политиками адаптации), в данной статье предлагается рассмотреть модель использования тестируемой системы [13]. Таким образом, в нашем случае речь пойдет не о модели адаптивной системы в рамках политик адаптации, а о модели ее среды использования [14], ориентированной на происходящие события, на которые система может реагировать в соответствии

с политиками адаптации. Такие модели обычно меньше моделей, описывающих систему в целом, и инженерам по валидации проще их проектировать вручную.

Напомним, что система может меняться в ответ на внешние события из Θ (также рассматриваемые как контролируемые события, например, посылаемые инженером в целях валидации), ее состояние также изменяется. Это может инициировать внутренние (неконтролируемые) события, которые регистрируются для наблюдения со стороны [3]. Это влияет на генерацию следующих возможных внешних событий, которые будут отправлены в систему, поскольку реконфигурирование может предотвратить происхождение некоторых событий в системе. Таким образом, процесс генерации должен избегать формирования тестов с неактуальными действиями реконфигурирования. Он должен также учитывать внутренние события, которые будут только наблюдаемыми, и соответствующим образом корректировать следующий шаг тестирования. Как следствие, модель среды также принимает во внимание внутренние события, которые соответствуют реконфигурированиям, которые могут произойти в системе.

Чтобы иметь возможность обрабатывать такие ситуации, в предлагаемом подходе онлайн тестирования: (а) каждый шаг теста выполняется непосредственно на тестируемой системе (SUT для System Under Test), (б) изменение тестируемой системы отслеживается алгоритмом генерации тестов и учитывается для генерации следующего шага теста. Предлагаемый процесс показан на рис. 3. Модели использования компонентов исследуются алгоритмом для вычисления следующих шагов теста. Они являются входом для генератора теста (1), который соединен с SUT. В режиме онлайн алгоритм случайным образом выбирает компоненту и вычисляет путем исследования ее модели использования следующий шаг теста (2), а именно событие, которое будет послано тестируемой системе. SUT может отреагировать на событие и выполнить реконфигурирование, которое будет зарегистрировано в трассе выполнения (3). Для вычисления последующего шага генератор тестов также рассматривает обновленную трассу выполнения, которая используется для обновления текущего состояния генератора (4). Параллельно трассы выполнения используются для обнаружения возможного нарушения временных свойств в политиках адаптации, как описано в [2], или для обнаружения несанкционированного реконфигурирования, как описано в [3]. Эти аспекты были уже описаны раньше и не находятся в центре внимания настоящей статьи.

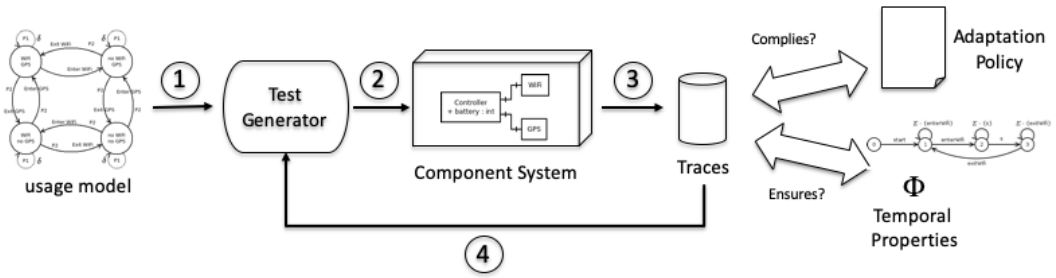

Fig. 3. Online test generation process

Рис. 3. Онлайн процесс генерации тестов

2.2. Тестовые случаи и пути реконфигурирования

Определим тестовые случаи как последовательности контролируемых событий, которые отправляются в тестируемую систему с некоторой периодичностью. В случае адаптивных систем, которые являются гибридными системами, где дискретное и непрерывное время смешиваются, может быть принят дискретизированный подход, подобный описанному в [15]. В этом случае события,

[3] Эти два множества являются непересекающимися, как описано в разд. 1 для операций реконфигурирования и отражено в грамматике FTPL.

используемые для стимулирования системы, посылаются с заданной периодичностью, символизируемой тактами, длительность которых параметризована. Кроме того, введем понятие задержки, обозначенное δ, которая состоит в том, чтобы не предпринимать никаких действий на SUT.

Пример 5 (Тестовый случай для сети VANet). *Для валидации функционирования сети VANet предлагается следующий тестовый случай:*

$$\delta; \delta; V1.join; \delta; ...; \delta; V2.join; \delta; ...V2.quit; ...$$

Он отображает частые вхождения задержки δ, представляющее период времени, в течение которого состояние системы изменяется без какого-либо запроса со стороны окружающей среды, например, в это время уменьшается ресурс аккумуляторных батарей транспортных средств. В этой последовательности join является запросом от автомобиля на вхождение в колонну, а quit отражает выход из колонны.

При выполнении тестового случая на SUT создается трасса реконфигурирования. Затем она может быть проанализирована, чтобы установить, соответствует ли система различным спецификациям, а именно соблюдаются ли политики адаптации и ожидаемые временные свойства.

Пример 6 (Путь реконфигурирования по отношению к внешним событиям). *Рассмотрим две последовательности, показанные на рис. 4: последовательность внешних событий (а именно тестовый случай), состоящий из запросов join и forceQuit в строке верхнего уровня, и путь реконфигурирования (а именно трасса выполнения) системы, сгенерированный в ответ на внешние события, в строке нижнего уровня. Последовательности внешних событий выбираются с одинаковой частотой, а пути реализации сообщают о приеме событий и инициировании реконфигурирований в ответ на эти события.*

В сети VANet могут происходить следующие внешние события. Событие join происходит, когда транспортные средства достаточно близки для слияния в колонну, система может либо принять объединение и вызвать реконфигурирование acceptJoin, либо решить отказаться от запроса refuseJoin. Когда водитель решает выйти из колонны (внешнее событие по отношению к автономной машине), это вызывает событие forceQuit, и система реагирует на него реконфигурированием (quit). Система также может реагировать на внутренние события, например, лидер может поменяться при реконфигурировании (getRelay).

Fig. 4. A reconfiguration path according to external events

Рис. 4. Путь реконфигурирования по отношению к внешним событиям

2.3. Модели использования для онлайн-тестирования

В этом разделе описывается артефакт, который будет использоваться для генерации случаев теста: модели использования компонентов. Такие модели, описанные в [14], в основном направлены на определение различных событий, которые могут происходить в среде системы. Эти события являются контролируемыми и могут быть отправлены в тестируемую систему, на которую она реагирует. Возможно также отправление задержки δ, представляющей отсутствие внешних событий в течение заданного периода времени.

Наиболее распространенным подходом определения таких моделей является использование вероятностных автоматов. В этой статье предложено связать их с использованием компонентов. Интуитивно, каждому состоянию такого автомата соответствует набор событий, которые могут быть инициированы в нем, т.е. на которые компонент способен реагировать. Они будут использоваться генератором теста для создания тестовых случаев.

Как указано в разделе 2.1, неконтролируемые операции реконфигурирования могут быть зарегистрированы на трассах выполнения. Связанные с ними события также учитываются в модели использования для обнаружения и фиксирования изменения текущего состояния компонента, что необходимо для отправки подходящих событий в тестируемую систему.

Определение 6 (Вероятностный автомат модели использования)**.** *Для каждого компонента C его модель использования определена как детерминированный вероятностный автомат $\mathcal{A}_C = \langle Q, q_0, E_\delta \cup O, F, P \rangle$, где Q – множество состояний, $q_0 \in Q$ – начальное состояние, E_δ – набор контролируемых внешних событий*[4] *вместе с δ для задержки (отсутствия внешних событий), O – множество неконтролируемых событий, состоящее из операций реконфигурирования, наблюдаемых на системе* [5]*, F – отношение перехода $F \in Q \times (E_\delta \cup O) \times Q$, и P – вероятность*[6] *перехода $P : Q \times E_\delta \longrightarrow [0;1]$ такая, что $\forall q \in Q \Longrightarrow \Sigma_{e \in E_\delta} P(q, e) = 1$.*

Каждому компоненту соответствует вероятностный автомат, будь то составной виртуальный компонент высокого уровня (например, дорога в нашем примере), или базовый компонент (например, транспортное средство).

Пример 7 (Модели использования транспортных средств в VANet)**.** *В примере VANet каждое транспортное средство реагирует на три внешних события: enter, обозначающее, что транспортное средство въезжает на дорогу; join, которое обозначает, что транспортное средство запрашивает присоединение к другому транспортному средству или существующей колонне; и forceQuit, которое указывает, что машина покидает колонну из-за вмешательства водителя. Кроме того, имеются три неконтролируемых, но наблюдаемых события относительно автомобилей: leave означает, что транспортное средство покидает дорогу; acceptJoin представляет реконфигурирование колонны для принятия запроса от машины; и quit представляет операцию реконфигурирования для выхода машины из колонны. Модель использования для этого примера показана на рис. 5. Предполагается, что события происходят с определенной инженером вероятностью (число в скобках). На рисунке δ-маркированные переходы представляют задержку в одну единицу времени, а другие переходы с метками представляют либо контролируемые события create, join и forceQuit, либо наблюдаемые события leave, acceptJoin и quit, обозначающие реконфигурирования в системе. Наблюдаемые события представлены пунктирными линиями.*

На основе моделей использования отдельных компонентов можно построить композицию моделей использования компонентов, используя, например, операции инкапсуляции и рафинирования компонентов, композицию расширенных автоматов интерфейсов или иерархические input/output автоматы и т.д. Чтобы избежать такого построения, требующего времени и объемов памяти, можно работать децентрализованно, например, как это сделано в [2, 9] для оценки свойств во время выполнения. В нашем подходе модели использования компонентов обрабатываются децентрализованно.

[4] вызывающие реконфигурирования из Θ, см. раздел 1.2.

[5] реконфигурирования из $\mathcal{R}$, см. раздел 1.2.

[6] Предполагается, что если никакой переход из текущего состояния не отмечен событием, то вероятность этого события равна 0.

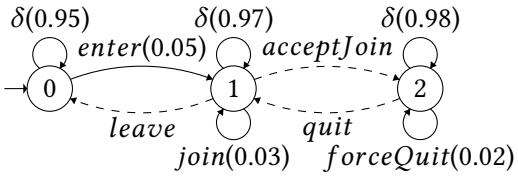

Fig. 5. Usage model of the VANet vehicle

Рис. 5. Модель использования автомобиля в сети VANet

2.4. Генерация тестов на базе моделей использования

Процесс генерации тестов опирается на модели использования компонентов, составляющих SUT, как представлено на рис. 3. Поскольку компоненты могут эволюционировать неконтролируемым образом, онлайн подход также опирается на трассу выполнения системы, регистрирующую происходящие реконфигурирования. Тестовый случай определяется как последовательность событий, полученных при прохождении переходов вероятностных автоматов моделей использования компонентов системы. Пусть $A_\delta(C)$ – модель использования, связанная с компонентом C. Для данной компонентно-ориентированной системы тестовый случай, основанный на множестве моделей использования, представляет собой конечную последовательность событий $C_0^j.e_0^j; C_1^k.e_1^k; ...C_n^l.e_n^l$ (длины $n+1$), в которой на i-м шаге теста запускаемое событие e^j связано с компонентом C^j и имеет вероятность $P_{C^j}(q^j, e^j) > 0$.

Для вычисления множества тестовых случаев, также называемого набором тестов, случайный обход Маркова [16] выполняется по вероятностным моделям использования компонентов, при этом исследование моделей использования компонентов выполняется случайным образом. Предлагаемый алгоритм генерации тестов, представленный алгоритмом 1, повторяет следующие шаги до тех пор, пока не будет достигнута максимальная длина тестовых примеров. Сначала случайным образом выбирается один из компонентов (строка 6). Текущее состояние автомата обновляется по отношению к трассе выполнения системы (строка 7). Этот этап представляет собой учет различных операций реконфигурирования (на базе наблюдаемых событий), которые могли произойти с момента последнего выбора компонента. Затем среди переходов, исходящих из текущего состояния, случайным образом выбирается переход и связанная с ним вероятность (строка 8). Если переход не является задержкой (строка 9), событие передается в тестируемую систему через рассматриваемый компонент (строка 10-12), и текущее состояние обновляется. В конце (строка 14) при учете реального времени выполнения системы, процесс генерации теста ожидает (в зависимости от частоты дискретизации событий), прежде чем перейти к вычислению следующего шага теста.

Этот алгоритм может быть использован для генерации наборов тестов произвольного размера и тестовых случаев произвольной длины. Это позволяет провести оценку того, правильно ли реализованы нечеткие значения, связанные с операциями реконфигурирования в политике адаптации. С этой целью производится анализ относительных частот реконфигурирований, описанный в следующем разделе.

3. Соблюдение политик адаптаций

Нашей целью является оценка соответствия реализации политики адаптации по отношению к полезности реконфигурирования, определенной в спецификации нечеткими значениями. В предыдущей работе [3] представлено использование правил политик адаптации в качестве метрики (критериев) покрытия модели для оценки релевантности набора тестов. В настоящей работе представлено новое дополнительное использование такой метрики покрытия, чтобы оценить, как реализация

системы соблюдает инициирования операций реконфигурирования при применении правил политик адаптации.

Algorithm 1. Algorithm for online test case generation

1: **for all** A_C **do**
2: state(A_C) ⟵ $q_0(A_C)$
3: **end for**
4: $i \longleftarrow 0$
5: **while** i < n **do**
6: $C \longleftarrow$ selectComponent()
7: state(A_C) ⟵ update(A_C, $trace$)
8: $e \longleftarrow$ pick(E_δ, state(A_C))
9: **if** $e \neq \delta$ **then**
10: C.send(e)
11: state(A_C) ⟵ update(A_C, [e])
12: $i \longleftarrow i + 1$
13: **end if**
14: await()
15: **end while**

Algorithm 1. Алгоритм генерации тестов в онлайн

3.1. Покрытие, пригодность и частота правила

Покрытие правила Начнем с определения функции, которая подсчитывает количество выполнений правила на заданном пути реконфигурирования.

Определение 7 (Число выполнений правила)**.** *Пусть σ – путь реконфигурирования, и $actual_{\sigma(i)}$ – фактическая операция реконфигурирования, происходящая в состоянии $\sigma(i)$. Число выполнений правила $r \in R_R$ на σ определено следующим образом:*

$$\#actual^r(\sigma) = \sum\nolimits_i f_a^r(\sigma(i)),$$

где $f_a^r(\sigma(i))$ – характеристическая функция предиката $actual_{\sigma(i)} \in dom(I)$[7], в котором I – отношение, связывающее операции реконфигурирования с нечеткими значениями.

Эта информация может использоваться для оценки того, охватывается ли данное правило тестовым случаем или, в более общем плане, набором тестов. Однако если правило никогда не охватывается, это может происходить по нескольким причинам. Во-первых, тестовые примеры не достигают конфигурации, в которой применимо реконфигурирование. В этом случае набор тестов должен быть уточнен, чтобы охватить правило. Во-вторых, некоторые части правил адаптации могут быть написаны неправильно и содержать слишком строгий или даже недостижимый/недопустимый триггер (свойство b или охранный предикат g в определении 4). Для таких случаев мы предлагаем подсчитать количество раз, когда эти различные части правил удовлетворяются.

Пригодность правила Определим теперь число переключений условия инициирования правила как число конфигураций, в которых свойство запуска правила становится верным.

Определение 8 (Количество инициирований правила)**.** *Пусть σ – путь реконфигурирования, и $trig_{\sigma(i)}$ – множество операций реконфигурирования, которые могут быть инициированы в состоянии $\sigma(i)$, то есть b-триггеры которых истинны в $\sigma(i)$. Число инициирования правила $r \in R_R$ на σ определяется следующим образом: $\#trig^r(\sigma) = \sum_i f_t^r(\sigma(i))$, где $f_t^r(\sigma(i))$ – характеристическая функция предиката*

$$r \in trig_{\sigma(i)} \wedge r \notin trig_{\sigma(i-1)} \wedge actual_{\sigma(i-1)} \notin dom(I).$$

[7] которая равна 1, если предикат истинен, и 0 в противном случае

Поскольку реконфигурирование может происходить в области применения правила, пока свойство b является верным, подсчет количества таких конфигураций не дает информации о реальной ситуации с реконфигурированием. Поэтому для данного пути σ определим число пригодности (число допусков) правила, которое является числом состояний на пути σ, когда правило стало пригодным. Назовем такие правила, чьи триггеры b и охранные предикаты g истинны в конфигурации $\sigma(i)$, приемлемыми в данном состоянии.

Определение 9 (Количество пригодности (допусков) правила)**.** *Пусть σ – путь реконфигурирования, а $elig_{\sigma(i)}$ является множеством правил, которые могли бы быть применены в состоянии $\sigma(i)$. Число пригодности правила $r \in R_R$ определяется как $\#elig^r(\sigma) = \sum_i f_e^r(\sigma(i))$, где $f_e^r(\sigma(i))$ – характеристическая функция предиката*

$$r \in elig_{\sigma(i)} \wedge r \notin elig_{\sigma(i-1)} \wedge actual_{\sigma(i-1)} \notin dom(I).$$

Для каждого правила эти метрики помогают определить, какая часть правила не была удовлетворена во время выполнения тестовых случаев. Однако в некоторых случаях возможно, что правило являлось пригодным, но предлагаемая реализация системы намеренно игнорировала его. Для анализа таких случаев в качестве дополнительной метрики мы вычисляем частоту правил адаптации.

Частота правила Для данного набора тестов периодичность или частота активации правила измеряется как количество его активаций, соотнесенное к количеству конфигураций, в которых это правило становилось пригодным.

Определение 10 (Частота правила)**.** *Для данного набора тестов TS, состоящего из тестовых примеров tc, частота правила $r \in R_R$ определена следующим образом:*

$$freq^r(TS) = \frac{\Sigma_{tc \in TS} \#actual^r(exec(tc))}{\Sigma_{tc \in TS} \#elig^r(exec(tc))}.$$

Если правило не является приемлемым ни в одном состоянии, то есть число его допусков равно 0, то его частота также равна 0.

Чтобы получить частоты, имеющие смысл, мы полагаемся на процесс генерации тестов, описанный в разделе 2. С его помощью можно генерировать большие наборы тестов, охватывая при этом правила политик адаптации. Эта метрика полезна для оценки того, как часто активируется интересующее правило. После вычисления эту частоту можно сравнить с нечетким значением, специфицированным в правиле, чтобы обнаружить потенциальное несоответствие в реализации политики адаптации. Примечательно, что, начиная с определенного множества начальных конфигураций и предполагаемого поведения системы, анализ частот позволяет обнаружить правило с высокополезным реконфигурированием (полезность `high`), которое применяется реже, чем правило с низкополезным реконфигурированием (полезность `low`).

3.2. Соблюдение политики адаптации

Интуитивно соответствие реализации по отношению к политике адаптации может быть объяснено с помощью значений полезности, используемых в правилах адаптации. По определению 4, в правиле адаптации $r = (b, g, ir)$ пара $ir = (ope, f)$ присваивает реконфигурированию ope полезность f. Предположим, что нечеткие значения полезности упорядочены. Тогда инженер по валидации может ожидать, что каждое правило с полезностью N будет применяться с большей частотой, чем любое правило с полезностью $N - 1$. Формально, для данной политики адаптации A и данного набора тестов TS, политика адаптации считается точно реализованной, если

$$\forall r, r' \in A, f^r > f^{r'} \Longrightarrow freq^r(TS) > freq^{r'}(TS).$$

Пусть $\sqsubseteq$ обозначает известное отношение погружения слов. Рассматривая множество $E_\delta \cup O$ контролируемых и неконтролируемых событий, после удаления δ из рассматриваемых слов будем использовать отношение $\sqsubseteq$ не учитывая δ, с обозначением $\sqsubseteq_\delta$.

Утверждение 1. *Если политика адаптации A точно реализована, то $\forall tc \in TS$, $\exists \sigma \in \Sigma_{S \lhd A}$, такой что $tc \sqsubseteq_\delta tr(\sigma)$. Более того, $\forall i \geq 0$, конфигурация $\sigma(i)$ достижима посредством выполнения операций реконфигурирования, заданных множеством $elig_{\sigma(i)}$ приемлемых правил адаптации.*

Доказательство Каждый случай теста $tc \in TS$ генерируется с использованием алгоритма 1 и на базе определения 6. По определению 5 для $S \lhd A$ каждое состояние $\sigma(i)$ на пути σ, соответствующем тестовому случаю tc, получено путем применения либо правила *[ACT1]* в случае реконфигурирования по одному из правил политики адаптации, либо правила *[ACT2]* в случае наблюдаемого реконфигурирования. При этом в случае правила *[ACT1]*, начиная с начальных конфигураций, каждое следующее состояние является результатом одной ($actual_{\sigma(i)}$) из операций реконфигурирования, выбранной в соответствии с ее полезностью из множества правил $elig_{\sigma(i)}$, приемлемых в данной конфигурации. Выбор этого реконфигурирования при исполнении системы увеличивает его частоту, что соответствует точной реализации политики адаптации. Достижимость конфигураций на пути σ обеспечивается этой конструкцией. Отношение $tc \sqsubseteq_\delta tr(\sigma)$ погружения слов между случаем теста и трассой пути основано на использовании определений 6, 1 и 5.

4. Экспериментирование

Чтобы подтвердить предложенный в этой статье подход, следующий раздел приводит примеры экспериментов, где были выявлены несоответствия в реализации политик адаптации по сравнению с определенной в них полезностью. Мы начинаем этот раздел с вопросов исследования (RQ), затем опишем эксперименты, их результаты, а также их обоснованность.

[RQ1.] В какой степени наш подход эффективен для генерации большого количества соответствующих тестовых случаев? Целью является оценка способности генераторов тестов производить большие наборы тестов с хорошим охватом правил политик адаптации. Особенно мы хотим гарантировать, что тестовые случаи способны достичь состояний системы, в которых реконфигурирования являются приемлемыми.

[RQ2.] В какой степени анализ частот реконфигурирований позволяет оценить нечеткие значения? Чтобы иметь возможность измерять частоты достаточно точно, мы стремимся проверить, что выполняется значительное количество реконфигурирований.

[RQ3.] В какой степени можно обнаружить подозрительные результаты? Наша общая цель состоит в том, чтобы оценить, способен ли процесс обнаружить несоответствия в реконфигурированиях, которые выполняются с полезностью, заданной нечеткими значениями в политиках адаптаций.

Экспериментальная процедура Для ответа на эти вопросы был разработан следующий эксперимент. Для сети VANet были разработаны различные реализации (на языке Java), которые отличаются друг от друга стратегией выполнения реконфигурирований. Типичные стили реализации взяты нами из ранее опубликованных по этой тематике статей. Например, в [1] авторы преобразовывают нечеткие значения для операций реконфигурирования, используя упорядочение, при этом полезность применения операций гарантируется пороговым значением. Правила политик адаптации, реализованных в Tangram4Fractal [8], рассматриваются в порядке объявления: для применения выбирается правило, которое является первым применяемым в порядке описания в политике адаптации. В [17] авторы классифицируют реконфигурирования в соответствии с определенными приоритетами: если допустимы две операции с одинаковым приоритетом, выбирается первая из них.

Разработанный эксперимент для сети VANet содержит 9 операций реконфигурирования, из которых 3 (GetRelay, PassRelay и Quit) инициируются 8 правилами политики адаптации (R1-R8). Правило

R1 определяет, когда ведущее транспортное средство может передать реле, правило R6 определяет, когда транспортное средство может заменить нынешнего лидера. Другие правила определяют различные случаи, когда машина может выйти из своей колонны из-за исчерпания энергии или достижения пункта назначения. Правила R1-R4 имеют высокую полезность, полезность правил R5-R6 средняя, а правила R7-R8 описываются с низкой полезностью.

В проведенном нами эксперименте транспортные средства могут динамически добавляться на дорогу и удаляться, как только они достигают места назначения. Во время различных экспериментов количество транспортных средств колебалось между 100 и 250. В итоге машины в индивидуальном режиме сопровождались колоннами (от 4 до 25), в каждой из которых было до 8 машин. Такое функционирование обеспечивало обновление транспортных средств на дороге и повышало шансы на достижение конфигураций, в которых возможны реконфигурирования. Заметим, что генерирование приемлемых начальных конфигураций, которые минимизируют количество компонентов, не находилось в центре внимания данного эксперимента.

Процесс генерации тестов, описанный алгоритмом 1, использовался для создания и выполнения случаев теста на различных реализациях. Основываясь на трассах выполнения, были вычислены частоты применения правил в соответствии с формулами, приведенными в разделе 3. Затем эти частоты были сравнены с нечеткими значениями, описанными в правилах политики адаптации. В нашем эксперименте анализ был выполнен на случаях теста, состоящих из 100 000 шагов. Чтобы обеспечить активацию каждого правила адаптации по меньшей мере один раз на рассматриваемом наборе тестов, мы использовали критерии покрытия, определенные в [3].

Измерения частот реконфигурирований на каждом шаге выполнения позволяют нарисовать графики, которые показывают эволюцию частот по мере выполнения. Для каждого правила реконфигурирования эти графики можно использовать для сравнения его частоты с полезностью реконфигурирования, заданной в правиле адаптации. В конце выполнения эксперимента можно сравнить частоты, взятые в совокупности, чтобы проверить фактическое инициирование реконфигурирований. Как представлено на рис. 6, частоты стабилизируются с увеличением числа шагов, показывая, что большое количество генерируемых тестов позволяет выполнить сравнение частот, которое заслуживает доверия (RQ1).

Использование графиков позволяет визуальный анализ, который делает возможным обнаружение подозрительного поведения, например, потенциальную нестабильность на одной из частот (RQ3).

Для оценки способности к обнаружению потенциальных проблем при реализации политики адаптации было разработано шесть реализаций (E1-E6), которые отличаются друг от друга способом выбора реконфигурирования. Первая реализация E1 выбирает реконфигурирования в соответствии с уровнем приоритета. Кроме того, если приемлемое реконфигурирование не было выполнено, его приоритет увеличивается для следующего шага. Результаты выполнения E1 можно найти в левой части рис. 6 и в таблице 1. При рассмотрении рисунка можно заметить, что график реконфигурирования R5 находится под графиками реконфигурирований R7 и R8. Эти графики показывают несоответствие, поскольку R5 имеет средний приоритет и должен иметь более высокую частоту, чем низкоприоритетные реконфигурирования R7 и R8. Затем мы изменили E1, чтобы создать реализацию E2, увеличив уровень приоритета реконфигурирования R5, результаты которого можно найти в правой части рис. 6 и в таблице 1. В результате график R5 стал расположен выше, чем R7 и R8, соответствуя заданным приоритетам. Таким образом, этот пример показывает, что предложенный подход помогает обнаружить несогласованный выбор реализации и затем подтвердить произведенную корректировку (RQ2).

Для продолжения экспериментов были смоделированы и имитированы другие варианты реализации со скорректированным уровнем приоритета, как в E2. Их результаты приведены в таблице 1.

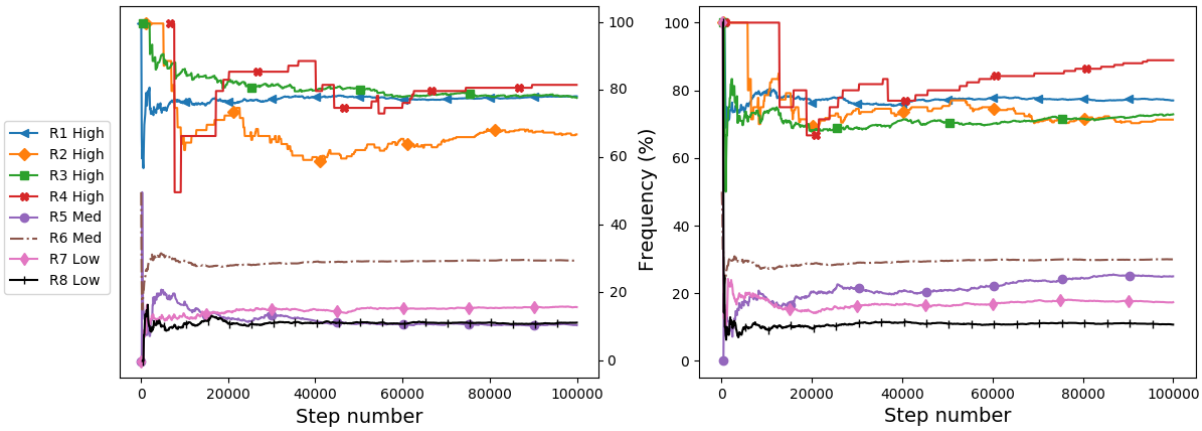

Fig. 6. Frequency analysis of implementations E1 (left) vs. E2 (right)

Рис. 6. Анализ частот реализации E1 (слева) и E2 (справа)

При реализации E3 использовалась гипотеза справедливости: если два реконфигурирования с одинаковым приоритетом являются приемлемыми, то реконфигурирование, которое не было выбрано на данном шаге, будет выбрано в следующий раз, когда сложится такая же ситуация. При моделировании E4 реконфигурирования выбираются на основе их уровня приоритета. Реализация E5 выбирает правило с низкой полезностью в 20% случаев. Наконец, реализация E6 выбирает первое реконфигурирование политики адаптации, которое может быть инициировано.

Результаты моделирования E2 и E3 показывают, что сделанный выбор реализации соответствует указанным нечетким значениям полезности. Реализация E4 показывает приемлемые результаты, но реконфигурирование R4 происходит слишком часто по сравнению с другими реконфигурированиями, имеющими такую же полезность. Реализация E5 является несогласованной, поскольку реконфигурирование R5 среднего приоритета находится на графике ниже реконфигурирований R7 и R8 низкого приоритета. Наконец, моделирование E6 показывает несколько несоответствий в частотах. Эти эксперименты показывают интерес предложенного метода, позволяющего пользователю как подтвердить реализацию системы в присутствии политик адаптации, так и идентифицировать подозрительное поведение. В этом случае последующий анализ позволяет разработать реализацию сложной системы, соответствующую политике адаптации (RQ2).

Table 1. Average results for 100 000-step executions

Таблица 1. Средние результаты для выполнений из 100 000 шагов

	R1(%)	R2(%)	R3(%)	R4(%)	R5(%)	R6(%)	R7(%)	R8(%)
	high	high	high	high	med.	med.	low	low
E1	78.4	67.2	78	82.8	10.6	29.7	16	11.5
E2	77	71.3	72.9	87.9	25.5	30	17.3	10.9
E3	68.7	64.7	70.8	76	23.1	31.4	14.9	9.4
E4	79.2	72.3	74.1	92.9	25	30.2	17.2	11.4
E5	64.4	66	56.6	68.2	17	29.7	19.1	24.4
E6	22.9	14.9	0.7	0	4.2	26.2	10.4	23.3

Обоснованность результатов Первая угроза обоснованности относится к анализу частот, который может опираться на слишком малое количество инициирований правил, чтобы иметь возможность выполнять точное упорядочение реальных значений полезности правил. Заметим в связи с этим, что во время экспериментов каждое реконфигурирование запускалось от нескольких десятков до нескольких тысяч раз в случаях тестов, состоящих из 100 000 шагов. Таким образом, было обеспечено значительное количество допусков для каждого правила, что дает релевантность вы-

числяемым частотам. Вторая угроза обоснованности заключается в том, что эксперименты были проведены только для одной сложной системы, хотя и с многочисленными реализациями. Расширение экспериментов на другие примеры систем является частью будущих работ. Наконец, еще одной угрозой обоснованности является тот факт, что мы сами разработали все реализации, что может вызвать некоторую предвзятость. Тем не менее, как упоминалось выше, выбор сделанных нами реализаций опирается на реальные стратегии применений адаптаций, уже описанные в соответствующей литературе.

5. Библиографические заметки и заключение

5.1. Библиографические заметки

Данная работа продолжает рассмотрение вопроса о самоадаптации сложных систем. Специализированные обзоры [18, 19] подчеркивают, что адаптивные и самоадаптивные системы должны обеспечивать безопасность на разных уровнях, учитывая автоматические реконфигурирования и важность их согласованности. Авторы предлагают обратный контроль поведения системы и изменение системы в случае необходимости. Наш подход с онлайн-тестированием также предлагает инструментирование системы. Однако вместо изменения системы, мы предлагаем информировать инженеров по валидации или пользователей о потенциальных проблемах, поскольку 1) прямой контроль не предполагается при тестировании закрытых систем (black-box testing), и 2) мы рассматриваем, что ошибки могут появляться на фазе разработки, нежели происходить из среды выполнения системы. В статье [20] был обсужден подход к онлайн-тестированию, и авторы выступили за использование онлайн-обнаружения сбоя и диагностики для восстановления до правильного состояния системы после него. В работе [21] авторы подвели итог по использованию моделей для обеспечения уверенности в самоадаптивных системах. Они определили методы, которые опираются на эволюционные алгоритмы для автоматической генерации случаев тестов. Другие подходы [22] использовали цикл MAPE-T для мониторинга применяемости и полезности случаев тестов во время выполнения системы. Наш подход также основан на принципах мониторинга, но не для анализа случаев теста. Нашей целью является проверка реализаций системы в присутствии политик адаптации, поэтому мы генерируем случаи тестов непосредственно в режиме онлайн на основе выполнения системы и модели среды использования системы.

Среди моделей взаимодействия для распределенных систем отметим язык взаимодействия в [23], чья операционная семантика основана на пошаговом выполнении. В отличие от данной работы, позволяющей валидацию трасс выполнения посредством чтения событий одного за другим в офлайн режиме, наш подход к тестированию на базе моделей использования компонентов позволяет онлайн обработку событий. Подход, описанный в [24], также используется офлайн для управления состояниями компонентно-ориентированных систем. Для этого используются политики адаптации с подмножеством временных шаблонов над состояниями. В отличие от нашего подхода политики применяются сразу, позволяя построить множество состояний системы и затем планировать в режиме офлайн реконфигурирования, имеющие целью оптимизацию некоторых нефункциональных требований к системе.

В области систем на основе компонентов онлайн-тестирование использовалось в [25] для обнаружения нарушений свойств системы после реконфигурирования с целью возвращения в нормальное состояние. Несколько статей относятся к встроенным тестам для компонентных систем с акцентом на их архитектуру. В работе [26] использовался онлайн-подход к тестированию на базе моделей, где тестируемая система (SUT) стимулировалась марковским процессом принятия решений (MDP). Состояния модели и поведение системы были связаны, чтобы генерировать действия модели использования, которые приведут к намеченному поведению. В отличие от вышеупомянутых работ,

наш подход основан на извлечении информации из записей поведения системы для генерации соответствующих тестов для SUT на базе модели ее использования.

Правила адаптации и политики адаптации полезны для уменьшения или даже устранения неопределенности, когда несколько разных действий возможны в адаптивной системе. В области динамического дельта-моделирования для адаптивных компонентных систем работа [27] использовала множество правил с приоритетами в качестве технологических маршрутов. В статьях [17, 28] политики адаптации содержат правила со значениями приоритета, тогда как в [29] функции полезности используются для определения целей системы. С одной стороны, наше понятие полезности близко к ним, так как оно позволяет управление приоритетами, чтобы выбрать правило с наибольшей полезностью по отношению к намеченным целям. С другой стороны, вышеупомянутые работы не подтверждали точного выполнения правил. В [30] авторы улучшили реактивность самоадаптивных систем с помощью предиктивных алгоритмов. Наш подход является более общим, поскольку он подтверждает полезность правил реконфигурирования для разнообразных систем в присутствии политик адаптации.

Заключение

В настоящей статье был представлен подход и метод для автоматической генерации тестовых случаев для валидации политик адаптации компонентно-ориентированных систем. Этот подход направлен на получение больших наборов тестов на основе вероятностных моделей среды использования компонентов, чтобы иметь возможность оценивать метрики во время выполнения системы и тем самым сравнивать выполнение операций реконфигурирования по отношению к их формальной спецификации в политиках адаптации. Проведенные эксперименты показали, что этот подход позволяет обнаружить реализации, которые не соблюдают должным образом полезность реконфигурирований. Подчеркнем, что этот подход, применяемый в данной статье к системам компонентов, может быть легко адаптирован к другим типам систем с использованием правил и политик адаптации.

Одним из будущих направлений работы на основе этого подхода является предоставление пользователю средств для проверки того, что политика адаптации, которая правильно реализуется, выполняет такие нефункциональные свойства, как оптимизированное потребление ресурсов и т. д. Эта работа имела целью генерацию случаев теста в виде последовательностей событий. Одним из улучшений может быть автоматическая генерация большого числа имеющих смысл начальных конфигураций, которые могут быть решающими в процессе тестирования для достижения конкретных конфигураций во время выполнения адаптивной системы.

References

[1] J. Dormoy, O. Kouchnarenko, and A. Lanoix, “Using Temporal Logic for Dynamic Reconfigurations of Components”, in *FACS*, ser. LNCS, L. Barbosa and M. Lumpe, Eds., vol. 6921, Springer Berlin Heidelberg, 2012, pp. 200–217, ISBN: 978-3-642-27268-4. DOI: 10.1007/978-3-642-27269-1_12.

[2] O. Kouchnarenko and J.-F. Weber, “Decentralised Evaluation of Temporal Patterns over Component-Based Systems at Runtime”, in *Formal Aspects of Component Software*, I. Lanese and E. Madelaine, Eds., ser. LNCS, vol. 8997, Bertinoro, Italy: Springer, Sep. 2015, pp. 108–126.

[3] F. Dadeau, J.-P. Gros, and O. Kouchnarenko, “Testing Adaptation Policies for Software Components”, *Software Quality Journal*, no. 28, pp. 1347–1378, 2020. DOI: 10.1007/s11219-019-09487-w.

[4] O. Kouchnarenko and J.-F. Weber, “Adapting Component-Based Systems at Runtime via Policies with Temporal Patterns”, in *FACS, 10th Int. Symp. on Formal Aspects of Component Software*, ser. LNCS, J. L. Fiadeiro, Z. Liu, and J. Xue, Eds., vol. 8348, Springer, 2014, pp. 234–253, ISBN: 978-3-319-07601-0. DOI: 10.1007/978-3-319-07602-7_15.

[5] M. Kim, I. Lee, U. Sammapun, J. Shin, and O. Sokolsky, "Monitoring, checking, and steering of real-time systems", *ENTCS*, vol. 70, no. 4, pp. 95–111, 2002. DOI: 10.1016/S1571-0661(04)80579-6.

[6] R. A. Kowalski and M. J. Sergot, "A Logic-based Calculus of Events", *New Gener. Comput.*, vol. 4, no. 1, pp. 67–95, 1986. DOI: 10.1007/BF03037383. [Online]. Available: https://doi.org/10.1007/BF03037383.

[7] R. Miller and M. Shanahan, "The Event Calculus in Classical Logic - Alternative Axiomatisations", *Electron. Trans. Artif. Intell.*, vol. 3, no. A, pp. 77–105, 1999. [Online]. Available: http://www.ep.liu.se/ej/etai/1999/016/.

[8] F. Chauvel, O. Barais, I. Borne, and J.-M. Jézéquel, "Composition of Qualitative Adaptation Policies", in *23rd IEEE/ACM Int. Conf. on Automated Software Engineering (ASE 2008)*, IEEE Computer Society, 2008, pp. 455–458, ISBN: 978-1-4244-2187-9.

[9] A. Bauer and Y. Falcone, "Decentralised LTL monitoring", in *FM 2012: Formal Methods*, ser. LNCS, vol. 7436, Springer, 2012, pp. 85–100.

[10] K. Larsen and B. Thomsen, "A Modal Process Logic", in *LICS'88, 1988*, IEEE Computer Society, 1988, pp. 203–210. DOI: 10.1109/LICS.1988.5119.

[11] R. Milner, *Communication and concurrency*, ser. PHI Series in computer science. Prentice Hall, 1989, ISBN: 978-0-13-115007-2.

[12] B. Jonsson and K. Larsen, "Specification and Refinement of Probabilistic Processes", in *Proc. LICS'91*, IEEE Computer Society, 1991, pp. 266–277, ISBN: 0-8186-2230-X. [Online]. Available: https://ieeexplore.ieee.org/xpl/conhome/360/proceeding.

[13] J. A. Whittaker and M. G. Thomason, "A Markov chain model for statistical software testing", *IEEE Trans. on Software Engineering*, vol. 20, no. 10, pp. 812–824, 1994, ISSN: 0098-5589.

[14] G. H. Walton, J. H. Poore, and C. J. Trammell, "Statistical testing of software based on a usage model", *Software: Practice and Experience*, vol. 25, no. 1, pp. 97–108, 1995.

[15] G. Dupont, Y. Aït Ameur, M. Pantel, and N. Singh, "Proof-Based Approach to Hybrid Systems Development: Dynamic Logic and Event-B", in *Int. Conf. Abstract State Machines, Alloy, B, TLA, VDM, and Z (ABZ 2018)*, M. Butler, A. Raschke, and K. Reichl, Eds., ser. LNCS, vol. 10817, Springer-Verlag, 2018, pp. 155–170.

[16] A. Sinclair, *Algorithms for Random Generation and Counting: A Markov Chain Approach*. Basel, Switzerland, Switzerland: Birkhauser Verlag, 1993, ISBN: 0-8176-3658-7.

[17] D. Romero, C. Quinton, L. Duchien, L. Seinturier, and C. Valdez, "SmartyCo: Managing Cyber-Physical Systems for Smart Environments", in *Software Architecture – 9th European Conference, ECSA 2015*, 2015, pp. 294–302. DOI: 10.1007/978-3-319-23727-5_25.

[18] R. De Lemos, H. Giese, H. A. Müller, M. Shaw, J. Andersson, M. Litoiu, B. Schmerl, G. Tamura, N. M. Villegas, and T. Vogel, "Software engineering for self-adaptive systems: A second research roadmap", in *Software Engineering for Self-Adaptive Systems II*, Springer, 2013, pp. 1–32.

[19] R. De Lemos, D. Garlan, C. Ghezzi, H. Giese, J. Andersson, M. Litoiu, B. Schmerl, D. Weyns, L. Baresi, and N. Bencomo, "Software engineering for self-adaptive systems: Research challenges in the provision of assurances", in *Software Engineering for Self-Adaptive Systems III. Assurances*, Springer, 2017, pp. 3–30.

[20] S. Gupta, A. Ansari, S. Feng, and S. A. Mahlke, "Adaptive online testing for efficient hard fault detection", in *27th Int. Conf. on Computer Design*, 2009, pp. 343–349. DOI: 10.1109/ICCD.2009.5413132.

[21] B. H. C. Cheng, K. I. Eder, M. Gogolla, L. Grunske, M. Litoiu, H. A. Müller, P. Pelliccione, A. Perini, N. A. Qureshi, B. Rumpe, D. Schneider, F. Trollmann, and N. M. Villegas, "Using Models at Runtime to Address Assurance for Self-Adaptive Systems", in *Models@run.time: Foundations, Applications, and Roadmaps*, N. Bencomo, R. France, B. H. C. Cheng, and U. Aßmann, Eds. Cham: Springer International Publishing, 2014, pp. 101–136, ISBN: 978-3-319-08915-7. DOI: 10.1007/978-3-319-08915-7_4.

[22] E. M. Fredericks, A. J. Ramirez, and B. H. C. Cheng, "Towards run-time testing of dynamic adaptive systems", in *Proc. Int. Symp. on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*, May 2013, pp. 169–174. DOI: 10.1109/SEAMS.2013.6595504.

[23] E. Mahe, C. Gaston, and P. L. Gall, "Revisiting Semantics of Interactions for Trace Validity Analysis", in *FASE 2020, Proceedings*, ser. LNCS, vol. 12076, Springer, 2020, pp. 482–501. DOI: 10.1007/978-3-030-45234-6_24.

[24] F. Alvares, E. Rutten, and L. Seinturier, "Behavioural Model-Based Control for Autonomic Software Components", in *IEEE Int. Conf. on Autonomic Computing, ICAC'15*, IEEE Computer Society, 2015, pp. 187–196. DOI: 10.1109/ICAC.2015.31.

[25] M. Greiler, H.-G. Gross, and A. van Deursen, "Evaluation of online testing for services: a case study", in *Proc. Int. Workshop on Principles of Engineering Service-Oriented Systems, PESOS 2010*, 2010, pp. 36–42. DOI: 10.1145/1808885.1808893.

[26] M. Camilli, C. Bellettini, A. Gargantini, and P. Scandurra, "Online Model-Based Testing under Uncertainty", in *29th IEEE International Symposium on Software Reliability Engineering, ISSRE 2018*, 2018, pp. 36–46. DOI: 10.1109/ISSRE.2018.00015.

[27] M. Helvensteijn, "Dynamic delta modeling", in *16th International Software Product Line Conference, SPLC'12*, E. S. de Almeida, C. Schwanninger, and D. Benavides, Eds., ACM, 2012, pp. 127–134, ISBN: 978-1-4503-1095-6. DOI: 10.1145/2364412.2364434. [Online]. Available: http://dl.acm.org/citation.cfm?id=2364412.

[28] F. Trollman, J. Fähndrich, and S. Albayrak, "Hybrid adaptation policies: towards a framework for classification and modelling of different combinations of adaptation policies", in *Proc. Int. Conf. SEAMS@ICSE 2018, Gothenburg, Sweden, May 28-29, 2018*, 2018, pp. 76–86. DOI: 10.1145/3194133.3194137.

[29] J. O. Kephart and W. E. Walsh, "An Artificial Intelligence Perspective on Autonomic Computing Policies", in *5th IEEE International Workshop on Policies for Distributed Systems and Networks (POLICY 2004), 7-9 June 2004, Yorktown Heights, NY, USA*, 2004, pp. 3–12. DOI: 10.1109/POLICY.2004.1309145.

[30] V. Poladian, D. Garlan, M. Shaw, M. Satyanarayanan, B. R. Schmerl, and J. P. Sousa, "Leveraging Resource Prediction for Anticipatory Dynamic Configuration", in *Proc. Int. Conf. on Self-Adaptive and Self-Organizing Systems, SASO 2007*, IEEE Computer Society, 2007, pp. 214–223, ISBN: 0-7695-2906-2. DOI: 10.1109/SASO.2007.35. [Online]. Available: https://ieeexplore.ieee.org/xpl/conhome/4274871/proceeding.