# Dynamic QoS Management and Optimisation in Service-Based Systems

Radu Calinescu, Lars Grunske, Marta Kwiatkowska, Raffaela Mirandola, Giordano Tamburrelli

# Dynamic QoS Management and Optimisation in Service-Based Systems

Radu Calinescu, *Senior Member, IEEE,* Lars Grunske, Marta Kwiatkowska, Raffaela Mirandola, and Giordano Tamburrelli

**Abstract**—Service-based systems that are dynamically composed at run time to provide complex, adaptive functionality are currently one of the main development paradigms in software engineering. However, the Quality of Service (QoS) delivered by these systems remains an important concern, and needs to be managed in an equally adaptive and predictable way. To address this need, we introduce a novel, tool-supported framework for the development of adaptive service-based systems called QoSMOS (QoS Management and Optimisation of Service-based systems). QoSMOS can be used to develop service-based systems that achieve their QoS requirements through dynamically adapting to changes in the system state, environment and workload. QoSMOS service-based systems translate high-level QoS requirements specified by their administrators into probabilistic temporal logic formulae, which are then formally and automatically analysed to identify and enforce optimal system configurations. The QoSMOS *self-adaptation* mechanism can handle reliability- and performance-related QoS requirements, and can be integrated into newly developed solutions or legacy systems. The effectiveness and scalability of the approach are validated using simulations and a set of experiments based on an implementation of an adaptive service-based system for remote medical assistance.

**Index Terms**—Service-Oriented Software Engineering, QoS Management, QoS Optimisation, Adaptive Systems

## 1 Introduction

Service-based systems (SBSs) are playing an increasingly important role in application domains ranging from research and healthcare to defence and aerospace. Built through the dynamic composition of loosely coupled services offered by independent providers, SBSs are operating in environments characterized by continual changes to requirements, state of component services and system usage profiles. In this context, the ability of SBSs to adjust their behaviour in response to such changes through self-adaptation has become a promising research direction [32], [70].

Several approaches to architecting adaptive software systems (i.e., software systems that reconfigure themselves in line with changes in their requirements and/or environment) have already appeared in the literature [70], [66]. These approaches involve the use of intelligent control loops that collect information about the current state of the system, make decisions and then adjust the system as necessary (e.g. [5], [22], [26], [83], [101]). Alternative approaches define self-adaptable architectures that emulate the behaviour of biological systems, where the global, complex behaviour emerges from the cooperation and interaction among distributed, independent components [41], [91].

Achieving and maintaining well-defined Quality of Service (QoS) properties in a changing environment represents a key challenge for self-adapting architectures. Service-based systems are well positioned to address these challenges, as the exploitation of their different composition patterns (orchestration and choreography) can represent an efficient way to achieve self-adaptating architectures [12], [86]. Consider, for example, a highly dynamic system where the set of discoverable services may change over time, either because service providers publish (or withdraw) service descriptions, or because the availability of certain services may vary according to the users location or to the network connectivity. In this settings a more reliable or efficient service might become available and thus self-adaptation may allow its use to improve the overall QoS. A further example is that, because of the increase of the number of users concurrently accessing the system, the response time experienced by a user could become too high. In this case the system should adopt appropriate reconfiguration strategies (such as using more computational resources or changing service providers) to tackle the peaks in the workload.

Therefore, a significant research effort has been devoted to the definition and analysis of QoS properties in SBS systems (e.g. [43], [47], [79], [93]). As illustrated by the overview of related approaches later in this section, typical QoS properties associated with SBSs include operation cost on one hand, and probabilistic quality attributes such as availability, reliability and reputation [101], [5] on the other hand. Among these


- *R. Calinescu is with Aston University, Aston Triangle, Birmingham B4 7ET, UK. E-mail: R.C.Calinescu@aston.ac.uk*
- *L. Grunske is with the Faculty of ICT, Swinburne University of Technology, Australia. E-mail: lgrunske@swin.edu.au*
- *M. Kwiatkowska is with the Oxford University Computing Laboratory, Wolfson Building, Parks Road, Oxford OX1 3QD, UK. E-mail: Marta.Kwiatkowska@comlab.ox.ac.uk*
- *R. Mirandola and G. Tamburrelli are with Politecnico di Milano, Piazza Leonardo da Vinci, 32, Milano, Italy, E-mail: mirandola@elet.polimi.it; tamburrelli@elet.polimi.it*

QoS properties, the management of probabilistic quality attributes is particularly challenging due to problems arising from the environment variability (e.g., changing service workloads and failure rates). Furthermore, QoS management requires self-adaptive SBSs to take into account aspects such as QoS specification, QoS evaluation, QoS optimisation and QoS-based adaptation. Nevertheless, guaranteeing a given level of QoS in these systems is essential for their success in the envisioned "service market", where service providers will compete by offering services with similar functionality but different quality and cost attributes [12], [86].

To deal with the QoS management of SBSs, we define and realise a generic architecture for adaptive SBSs called QoSMOS (QoS Management and Optimisation of Service-based systems). QoSMOS is a tool-supported framework for the QoS management of self-adaptive, service-based systems that combines in a novel way existing techniques and tools developed by our research groups: (a) formal specification of QoS requirements with probabilistic temporal logics and the ProProST specification system [49]; (b) model-based QoS evaluation with probabilistic verification techniques provided by the PRISM model checker [72]; (c) monitoring and Bayesian-based parameter adaptation of the QoS models exploiting KAMI [40]; and (d) planning and execution of system adaptation based on GPAC [20].

The QoSMOS framework supports the practical realisation of adaptive SBS architectures by means of two complementary mechanisms. The first mechanism consists of selecting the services that compose a QoSMOS service-based system dynamically. Given a set of functionally equivalent services for each component of an SBS, QoSMOS selects those services whose reliability, performance and cost guarantee the realisation of the QoS requirements for the system. QoSMOS is capable of dynamically adapting its selection of services to runtime changes in both the service characteristics (e.g., reliability or performance) and the system QoS requirements. When service selection cannot achieve the QoS requirements, a warning is issued to alert the SBS administrator. The second adaptation mechanism employed by QoSMOS consists of adjusting the resources (e.g., the CPU) allocated to individual services within a service-based system dynamically. This mechanism is applied to services hosted and administered internally by the organisation that implements the SBS. Its key benefit is the ability to adapt the resources allocated to SBS component services to the actual workload of the system, thus ensuring that its QoS requirements are satisfied with minimal cost and environmental impact.

**Related Approaches.** One benefit of SBSs is the ability to build applications through composition of available services at run-time. This composition involves several activities, including the definition of an integration schema yielding the target application, the selection of concrete services that offer the required functionality, and the fulfillment of QoS constraints. While services are described and listed in public registries, there is still little support for QoS-based service management. To cover this gap, the research area of QoS Management in SBSs has been very active in the last five years. A publication-time- and problem-domain-sorted summary of recent approaches in the area of QoS-driven service selection, composition and adaptation is given in Table 1.

Specifically, we summarize the approaches according to: (a) the considered QoS metrics and QoS Specification languages (QoS Requirement Specification); (b) the models/algorithms adopted for the QoS metric evaluation (QoS Evaluation Methods); (c) the type of optimization problem defined and solved and/or the adaptation policies adopted (QoS Optimization or Adaptation Methods); and finally (d) the validation of the proposed approaches (Validation).

Considering these approaches, we identify some common points of weakness that we overcome with our QoSMOS approach.

*QoS Requirement Specification:* As illustrated in Table 1, a variety of different QoS requirements are considered in the current approaches. However, QoS specifications are often tackled in an abstract way, by dealing with simple metrics (e.g., by considering the failure rate as a metric to evaluate reliability). In our view, a detailed and formal specification of QoS requirements is required for a comprehensive management of QoS in service-based systems. A concise and unambiguous specification of the QoS requirements enables, among other benefits, a systematic management of SBSs based on the quantitative analysis of their QoS properties.

Current examples of specification languages for QoS aspects in the web services domain are: Web Service Level Agreement (WSLA) [65], SLAng [75], the timed Web Service Constraint Language (timed WSCoL) [9] which is close to a real-time temporal logic, the Web Service Management Language (WSML) [92] and the Web Service Offerings Language (WSOL) [98].

In addition, formal QoS specification can be achieved using formalisms like real-time and probabilistic temporal logics [1], [6], [8], [53], [69], [71], timed Life Sequence Charts [54], probabilistic and timed Message Sequence Charts [90], Performance Trees [97] or Probabilistic/Timed Behavior Trees [36], [37], [50]). To this end, QoSMOS adopts the probabilistic temporal logics PCTL [53] and CSL [8] because these logics are sufficiently expressive to formulate a variety of QoS requirements [49] whose formal verification can then be carried out using existing probabilistic model checkers.

*QoS Evaluation Methods:* To be effective, QoS evaluation approaches should rely on models representing the systems in an accurate/realistic way, and whose parameters can be adjusted at run-time according to measured data. Several approaches reported in Table 1 rely on the definition of simple aggregate QoS functions (like sum, product, max, and average) that can be easily defined and managed. However, due to dependencies between

| Authors& Citation | Problem Domain | QoS Requirements | QoS Evaluation | QoS Optimization or Adaption Method | Validation |
|---|---|---|---|---|---|
| Zeng et al. 2004 [101] | QoS-driven Service Composition/ Selection and (Re-) Planing | Reputation, Execution Time, Availability, Price | Simple Aggregation Functions (eg. Sum, Product, Max, Average) based on unfolded state-based process specifications with branch & loop probabilities | Local & Global Planning (Multi-objective with simple additive weighting, positive & negative QoS Attributes) | Simulation with generated examples |
| Cao et al. 2005 [27] | QoS-driven Service Selection | Cost | Simple Aggregation Function (Sum) for parallel, sequential and branching service composition | Genetic Algorithm | Experiments with generated examples |
| Ardagna and Pernici 2005, 2007 [4], [5] | QoS-driven Service Composition/Selection | same as [101] + Data Quality | Simple Aggregation Functions (eg. Sum, Product, Max, Average) similar to [101] based on unfolded BPEL4WS specifications | Multi-Choice&Dimension Knapsack Problem [4] and Integer Linear Programming [5] (Multi-objective with simple additive weighting, positive and negative QoS Attributes) | Simulation with generated examples |
| Canfora et al. 2005-2008 [24], [25], [26] | QoS-driven Service Binding and (Re-) Binding/Selection | (Execution) Time, Reliability, Availability, Price | Simple Aggregation Functions (eg. Sum, Product, Max, Average) similar to [101] based on an unfolded BPEL4WS specification with branch and loop probabilities | Genetic Algorithms | Experiments based on a Travel Planner case study |
| Cardellini et al. 2007-2009 [29], [30], [28] | QoS-driven Service Composition/ Selection | Response Time, (log)-Availability, Cost | Simple Aggregation Functions (eg. Sum, Product, Max, Average) similar to [101] based on an process activity trees generated from BPEL specifications | Linear Programming (Multi-objective with a suitable objective function (eg. weighted sum), multiple service activations), adaptation through workflow restructuring in [28] | Experiments with generated examples |
| Menascé et al. 2007, 2008, 2010 [80], [81], [83], [82] | QoS-driven Service Composition/ Selection | Average Execution Time (with cost constraints)[80], [82], Throughput [81], [83], [82], Availability [82], and Security [82] | Aggregation Function for the average execution time based on a tree based representation (BPTree) of a BPEL specification [80] and queuing network based performance evaluation [81], [83] | Exact method [80], [81] and hill-climbing inspired heuristic [80]; use of utility functions [81], [82] for the quality dimensions | Experiments with generated examples [80], [82] and based on a Travel Planner [81] & Emergency Response [82] case study |
| Zhang et al. 2007, 2008 [77], [96], [102] | QoS-driven Service Selection | same as [101] | Simple Aggregation Functions | (Quickly Convergent) Population Diversity Handling Genetic Algorithm (DiGA [102] and CoDiGA [77]) with Simulated Annealing | Generated experiments based on the Travel Planner case study |
| Berbner et al. 2006 [13] | QoS-driven Service Composition/ Selection | Availability, Response time, and Throughput | Simple Aggregation Functions (Sum, Product, Min) similar to [101] based on a sequential web service composition | Iterative approach: (1) Mixed Integer Programming + Backtracking, (2) Simulated Annealing or Random Swapping of Services (Mutation) | Simulation and comparison (IP vs. SA) with generated examples |
| Ko et al. 2008 [67] | QoS-driven Service Composition/ Selection | same as [101] + Frequency and Succ. exec. rate | Simple Aggregation Functions sequential, AND-parallel and XOR-parallel web service composition | Tabu-search+ plan generation based on neighborhood search | Simulation based on generated examples |
| Serhani et al. 2005 [7], [94] | QoS-driven Load Balancing/ Service Selection | Response Time and Throughput | Analytic solving of M/M/1 queueing system model [7] | Probabilistic slitting policy | Experiments based on gen. examples |
| Boone at al. 2010 [19] | QoS-driven Load Balancing | Response Time | Solving of M/M/1 queueing systems with Poisson distributed service requests | Simulated Annealing Load Spreading Algorithm (SALSA) | Experiments based on gen. testbeds |
| Marzolla at al. 2007 [79] | QoS-driven workflow management | Response Time | Analytic solving of BCMP queueing systems | Exact methods and bound analysis | Experiments based on gen. examples |
| Sato at al. 2007 [93] | QoS-driven workflow management | Reliability | Analytic solving of Markov Models | Exact solution based on CTMC analysis | Experiments based on gen. examples |
| Guo at al. 2007 [52] | QoS-driven workflow management | Availability | Simple Aggregation Functions sequential, choice, parallel and iterative web service composition | Hill Climbing based selection redundancy mechanisms | Experiments based on gen. examples |
| QoSMOS | Service Selection/ Resource allocation and Service Parametrisation | Performance and Reliability | Analytic solving of Markov Models (DTMC and MDP) | Exact solution based on probabilistic model checking with PRISM experiments | Experiments and simulations based on gen. examples and the *TeleAssistance* case study |

TABLE 1
Overview of related approaches

different services or between services and resources or the operational profiles these aggregation functions could lead to quality estimation that represent optimistic (or pessimistic) bounds rather than a realistic estimation.

In contrast, some other approaches focus on how to determine the QoS attributes of a composite system, given the QoS delivered by its sub-services. Examples can be found in [43], [83], [79], [93] where, starting from the BPEL business processes modeled by UML activity diagrams or by direct acyclic graphs, performance models based on simple queuing networks [83], [79] or reliability models based on Markov models are derived [43], [93].

In line with these approaches, we argue that comprehensive predictive quality evaluation models are needed. Examples of models that can be used for QoS evaluation

are: Markov models, state charts like probabilistic UML State Charts [59], [60], queuing networks models [18], [76], stochastic process algebras like PEPA [46]. Towards this end, QoSMOS adopts Markov models as modeling formalisms to determine quantitatively the reliability and performance quality metrics of service-based systems. However, as an enhancement to the existing approaches, we observe composite systems (e.g. usage profiles, branching and failure probabilities) at runtime and update the quality evaluation models.

To check if a Markov model satisfies its QoS requirements, numerical/symbolic [6], [8], [16], [53] and statistical [100] techniques have been developed, and extensive tool support is available (e.g. PRISM [72]).

*QoS Optimisation or Adaptation Methods:* Devising QoS-driven adaptation methodologies of SBSs is of utmost importance in the envisaged dynamic environment in which SBS operate. Most of the proposed methodologies for QoS-driven adaptation of SBS address this problem as a service selection problem (e.g., [5], [26], [30], [101]). Other papers have instead considered SBS adaptation through workflow restructuring, exploiting the inherent redundancy of SBS (e.g., [31], [52], [55].) In [28] a unified framework is proposed where service selection is integrated with other kinds of workflow restructuring, to achieve a greater flexibility in the adaptation.

According to this last approach, we conclude that the service selection and composition problem is really important for SBS QoS-based adaptation, but we argue also that for a comprehensive approach to QoS Management also optimal resource allocation and parametrization of the services is required.

The QoSMOS framework does not aim to invent new techniques, but includes and integrates optimization techniques and adaptation strategies derived from approaches already present in literature.

*Validation:* An investigation of the validation strategies, shows that several approaches perform experiments based on generated examples or apply a case study based validation. To validate the QoSMOS approach we use a similar validation strategy and perform experiments and simulations based on an implementation of a service-based system for remote medical assistance called *TeleAssistance* [10], [40].

**Contribution.** Based on the review of the related approaches the main contributions of the QoSMOS framework can be summarised as follows:

- In contrast to the simple and informal metrics that are currently used in the related approaches, QoSMOS uses a precise and formal specification of QoS requirements with probabilistic temporal logics;
- QoSMOS uses a tool-supported model-based quality evaluation methodology for probabilistic QoS attributes (i.e., performance, reliability and resource usage) of service-based systems that significantly improves current approaches that use simple aggregation functions for QoS prediction, because we could model quality dependencies on other services and the operational profile;
- QoSMOS utilises techniques and tools for monitoring service-based systems and learning the parameters of their model(s) from the observed behaviour of the system;
- QoSMOS adds self-adaptation (e.g., self-configuration and self-optimisation) capabilities to service-based systems through continuous verification of quantitative properties at run-time derived from high-level, user-specified system goals encoded with multi-objective utility functions. The self-adaptation capabilities include service selection, run-time reconfiguration and resource assignment. Consequently, QoSMOS subsumes most of the existing approaches.

**Organization.** The rest of the paper is organized as follows. In Section 2 we shortly describe the main formalisms used throughout the paper, namely probabilistic temporal logics and Markov Models. Section 3 describes the QoSMOS architecture, while a validation of the proposed framework that shows the effectiveness and scalability is presented in Section 4. Section 5 concludes the paper and points out directions for future works.

## 2 PRELIMINARIES

### 2.1 Formal definition of QoS requirements

The precise specification of QoS requirements or Service Level Agreements (SLAs) is an important aspect for service composition, service selection and optimisation of service-based systems [42]. In QoSMOS, QoS requirements are specified using real-time temporal logics such as MTL (Metric Temporal Logic) [69] and TCTL (Timed Computational Tree Logic) [1], or probabilistic temporal logics such as PCTL (Probabilistic Computation Tree Logic) [53], PCTL* [6], PTCTL (Probabilistic Timed CTL) [71] and CSL (Continuous Stochastic Logic) [8]. The significant benefits of using logic-based requirement specifications include the ability to define these requirements concisely and unambiguously, and to analyse them using rigorous, mathematically-based tools such as model checkers. Furthermore, for logic-based specification-formalism the correct definition of QoS properties is supported with specification patterns [39], [49], [48], [68] and structured English grammars [49], [68], [99].

In this article we focus on PCTL and CSL which are defined as follows [8], [33], [53]:

**Definition (PCTL/CSL Syntax)**. Let $AP$ be a set of atomic propositions and $a \in AP$, $p \in [0,1]$, $t_{PCTL} \in \mathbb{N}$, $t_{CSL1}, t_{CSL2} \in \mathbb{R}^{\geq 0}$ and $\bowtie \in \{\geq, >, <, \leq\}$, then a state-formula $\Phi$ and a path formula $\Psi$ in PCTL are defined by the following grammar:

$\Phi ::= true | a | \Phi \wedge \Phi | \neg\Phi | \mathcal{P}_{\bowtie p}(\Psi)$

$\Psi ::= X\Phi | \Phi U \Phi | \Phi U^{\leq t_{PCTL}} \Phi$

A state-formula $\Phi$ and a path formula $\Psi$ in CSL are defined by the following grammar:

$\Phi ::= true|a|\Phi \wedge \Phi|\neg\Phi|\mathcal{S}_{\bowtie p}(\Phi)|\mathcal{P}_{\bowtie p}(\Psi)$
$\Psi ::= X^{[t_{CSL1},t_{CSL2}]}\Phi|\Phi U^{[t_{CSL1},t_{CSL2}]}\Phi$

The logics distinguish between state and path formulae. The state formulae include the standard logical operators $\wedge$ and $\neg$, which also allow a formulation of other usual logical operators (disjunction ($\vee$), implication ($\Rightarrow$), etc.) and $false$. The main extension of the state formulae, compared to non-probabilistic logics, is to replace the traditional path quantifier $\exists$ and $\forall$ with a probabilistic operator $\mathcal{P}$. This probabilistic operator defines upper or lower bounds on the probability of the system evolution. As an example, the formula $\mathcal{P}_{\geq p}(\Psi)$ is true at a given state if the probability of the future evolution of the system satisfying $\Psi$ is at least $p$. Similarly, the formula $\mathcal{P}_{\leq p}(\Psi)$ is true if the probability that the system fulfills $(\Psi)$ is less than or equal to $p$. The path formulae that can be used with the probabilistic path operator are the "next" formula $X\Phi$, time bounded "until" formula $\Phi_1 U^{\leq t}\Phi_2$ and unbounded "until" formula $\Phi_1 U\Phi_2$. The formula $X\Phi$ holds if $\Phi$ is true in the next state of a path. Intuitively, the time bounded "until" formula $\Phi_1 U^{\leq t}\Phi_2$ requires that $\Phi_1$ holds continuously within a time interval $[0, x)$ where $x \in [0, t]$, and $\Phi_2$ becomes true at time instance $x$. The semantics of the unbounded versions is identical, but the (upper) time bound is set to infinity $t = \infty$. Based on the time bounded and unbounded "until" formula further temporal operators ("eventually" $\Diamond$, "always" $\Box$, and "weak until" $\mathcal{W}$) can be expressed as described in [33], [49]. For example the eventually formula $\mathcal{P}_{\bowtie p}(\Diamond\Phi)$ is semantically equivalent to $\mathcal{P}_{\bowtie p}(trueU\Phi)$. As an additional syntactical feature the logic CSL has been extended in [8] with a steady state operator $\mathcal{S}$ that describes the behavior of the system in the long run. Syntactically, this operator (state formula: $\mathcal{S}_{\bowtie p}(\Psi)$) is used similarly to the probabilistic path operator.

Traditionally, the semantics of the PCTL/CSL is defined with a satisfaction relation $|=$ over the states $S$ and possible paths $Path^{\mathcal{M}}(s)$ that are possible in a state $s \in S$ of a discrete/continuos time probabilistic model $\mathcal{M}$. For details about the formal semantics the reader is referred to [8], [33], [53]. Normally, a PCTL/CSL formula is evaluated starting from the initial state of the probabilistic model $\mathcal{M}$. However, for convenience in tools like PRISM any state and also a set of states can be chosen with a *filter*. Syntactically, a filter is specified as logical expression inside braces $\{\}$ at the end of the PCTL/CSL formula.

### 2.2 Quality evaluation models

Several approaches exist in the literature for the model-based quality analysis and prediction, spanning the use of Petri nets, queueing networks, layered queueing network, stochastic process algebras, Markov processes, fault trees, statistical models and simulation models (see [3] for a recent review and classification of models for software quality analysis).

In this article, we focus on Markov models which are a very general evaluation model that can be used to reason about performance and reliability properties. Furthermore, Markov models include other modelling approaches as special cases, such as queueing networks, Stochastic Petri Nets [78] and Stochastic Process Algebras [34].

Specifically, Markov models are stochastic processes defined as state-transition systems augmented with probabilities. Formally, a stochastic process is a collection of random variables $X(t), t \in T$ all defined on a common sample (probability) space. The $X(t)$ is the state while (time) $t$ is the index that is a member of set $T$ (which can be discrete or continuous). In Markov models [18], states represent possible configurations of the system being modelled. Transitions among states occur at discrete or continuous time-steps and the probability of making transitions is given by exponential probability distributions. The Markov property characterizes these models: it means that, given the present state, future states are independent of the past. In other words, the description of the present state fully captures all the information that could influence the future evolution of the process. The most used Markov models include:

- *Discrete Time Markov Chains* (DTMC), which are the simplest Markovian model where transitions between states happen at discrete time steps;
- *Continuous Time Markov Chains* (CTMC) where the value associated with each outgoing transition from a state is intended not as a probability but as a parameter of an exponential probability distribution (transition rate);
- *Markov Decision Processes* (MDP) [89] that are an extension of DTMCs allowing multiple probabilistic behaviours to be specified as output of a state. These behaviours are selected non-deterministically.

The analytical solution techniques for Markov models differ according to the specific model and to the underlying assumptions (e.g., transient or non-transient states, continuous vs. discrete time, etc.). For example, the evaluation of the stationary probability $\pi_s$ of a DTMC model requires the solution of a linear system whose size is given by the cardinality of the state space $S$. The exact solution of such a system can be obtained only if $S$ is finite or when the matrix of transition probabilities has a specific form. A problem of Markov models, which also similar evaluation models face, is the explosion of the number of states when they are used to model real systems [18]. To tackle this problem tool support (e.g. PRISM [72]) with efficient symbolic representations and state space reduction techniques [64], [73] like partial-order reduction, bisimulation-based lumping and symmetry reduction are required.

## 3 QoSMOS Architecture

This section introduces the generic QoSMOS architecture of an adaptive service-based system, and describes its

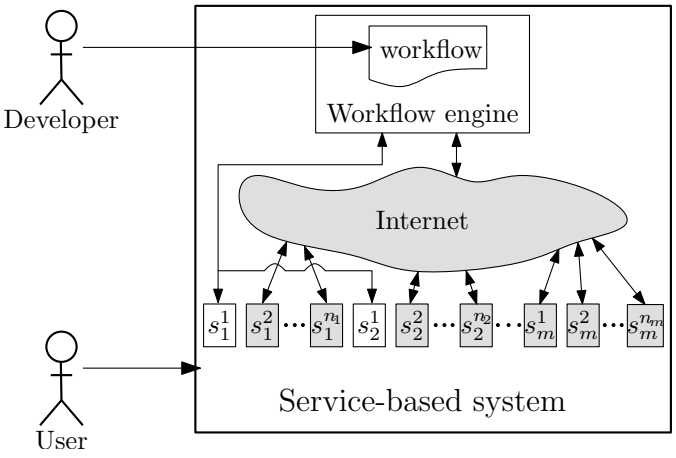



Fig. 1. The architecture of a service-based system comprising a mix of *in-house services* (clear) and *third-party services* (shaded).

realisation using existing tools and components. As QoSMOS extends existing service-based systems with the capability to adapt dynamically, we start by presenting the standard architecture of a service-based system (SBS).

As shown in Figure 1, a typical SBS consists of a composition of web services that are accessed remotely through a software application termed a *workflow engine*. Several services may provide the same functionality, often with different levels of performance and reliability, and at different costs. To capture this characteristic, our diagram depicts $m \geq 1$ sets of *concrete services*: the set $CS_i = \{s_i^1, s_i^2, \ldots, s_i^{n_i}\}$, $1 \leq i \leq m$, comprises $n_i \geq 1$ concrete services that provide the same *abstract service* $as_i$ from a functional viewpoint. The way in which the workflow engine employs some or all of the concrete services in order to provide the functionality required by the SBS *user* is specified in the *workflow* that the engine is executing. This workflow is typically provided by the *developer* of the SBS, and is expressed in a workflow language such as BPEL [62].

SBS users can be humans that access the system through a suitable user interface (not shown in Figure 1) or software components (e.g., other SBSs). In the former scenario, the developer and user roles represented as different entities in Figure 1 are sometimes assumed by the same person. Finally, note that an SBS can employ both services that are run and administered internally by the organisation that implements the SBS (i.e., *in-house services*), and third-party services accessed over the Internet.

*Example:* We will illustrate the concepts introduced so far by presenting a service-based system for remote medical asistance taken from [10], [40]. This *TeleAssistance* (TA) system will be used as a running example throughout the rest of the article, and its associated BPEL workflow is depicted in Figure 2. The TA system incorporates the following abstract services:

- *Alarm Service*, which provides the operation *sendAlarm*;
- *Medical Analysis Service*, which provides the operation *analyzeData*;
- *Drug Service*, which provides the operations *changeDoses* and *changeDrug*.

The TA workflow starts executing as soon as a Patient (PA) enables the home device supplied by the TA provider, and this device invokes the *startAssistance* operation of the workflow. The workflow then enters an infinite loop whose iterations start with a "pick" activity that suspends the execution and waits for one of the following three messages: (1) *vitalParamsMsg*, (2) *pButtonMsg* or (3) *stopMsg*. The first message contains the patient's vital parameters, which are forwarded by the BPEL workflow to the Medical Laboratory service (LAB) by invoking the operation *analyzeData*. The LAB is in charge of analyzing the data, and replies by sending a result value stored in a variable *analysisResult*. A field of the variable contains a value that can be *changeDrug*, *changeDoses* or *sendAlarm*. A *sendAlarm* value triggers the intervention of a First-Aid Squad (FAS) comprising doctors, nurses and paramedics whose task is to visit the patient at home in case of emergency. To alert the squad, the TA workflow invokes the operation *alarm* of the FAS. The message *pButtonMsg* caused by pressing a panic button also generates an alarm sent to the FAS. Finally, the message *stopMsg* indicates that the patient decided to cancel the TA service, deleting each pending invocation to the FAS service.

The workflow in Figure 2 represents the orchestration of $m = 3$ abstract services:

$$as_1 = AlarmService$$
$$as_2 = MedicalAnalysisService$$
$$as_3 = DrugService$$

Different providers could be involved in providing concrete implementations for the abstract services in the TA service-based system. For example, we will consider that the Alarm Service and the Medical Analysis Service are implemented by $n_1 = 3$ and $n_2 = 5$ telecommunication operators, respectively—each such concrete service being provided with different cost, performance and reliability characteristics. Finally, we will consider that a single, in-house implementation of the Drug Service is available (i.e., $n_3 = 1$).

### 3.1 Generic architecture of QoSMOS

As illustrated in Figure 3, QoSMOS augments the standard SBS architecture with a component termed an *autonomic manager*. This component employs the autonomic computing monitor-analyse-plan-execute (MAPE) loop [66], [56] to ensure that the SBS adapts continually in order to achieve a set of high-level, multi-objective QoS requirements specified by its *administrator*. The four stages of the QoSMOS MAPE loop are described below.

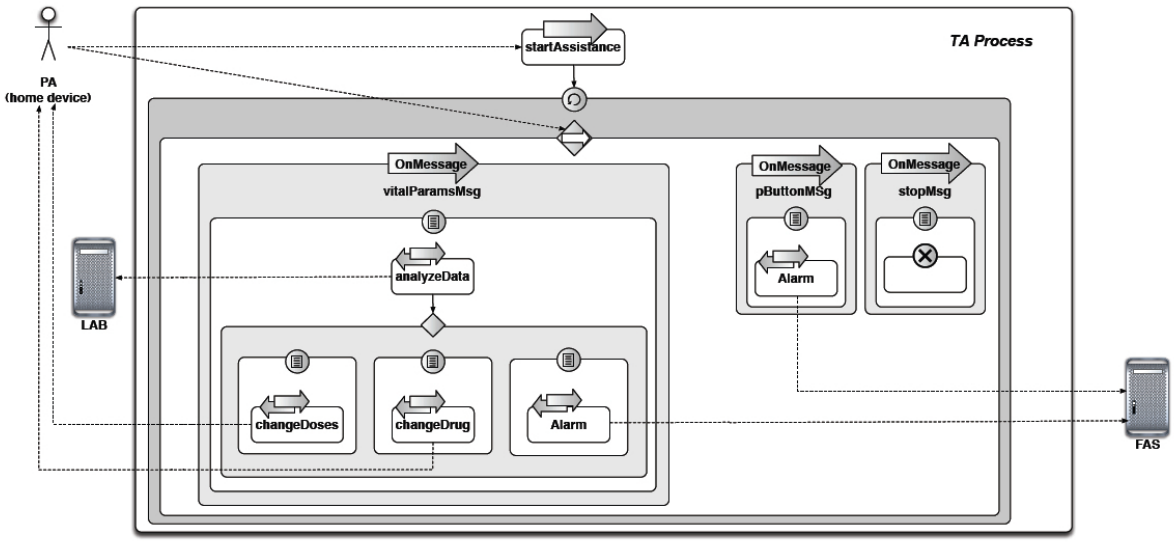



Fig. 2. TeleAssistance BPEL workflow

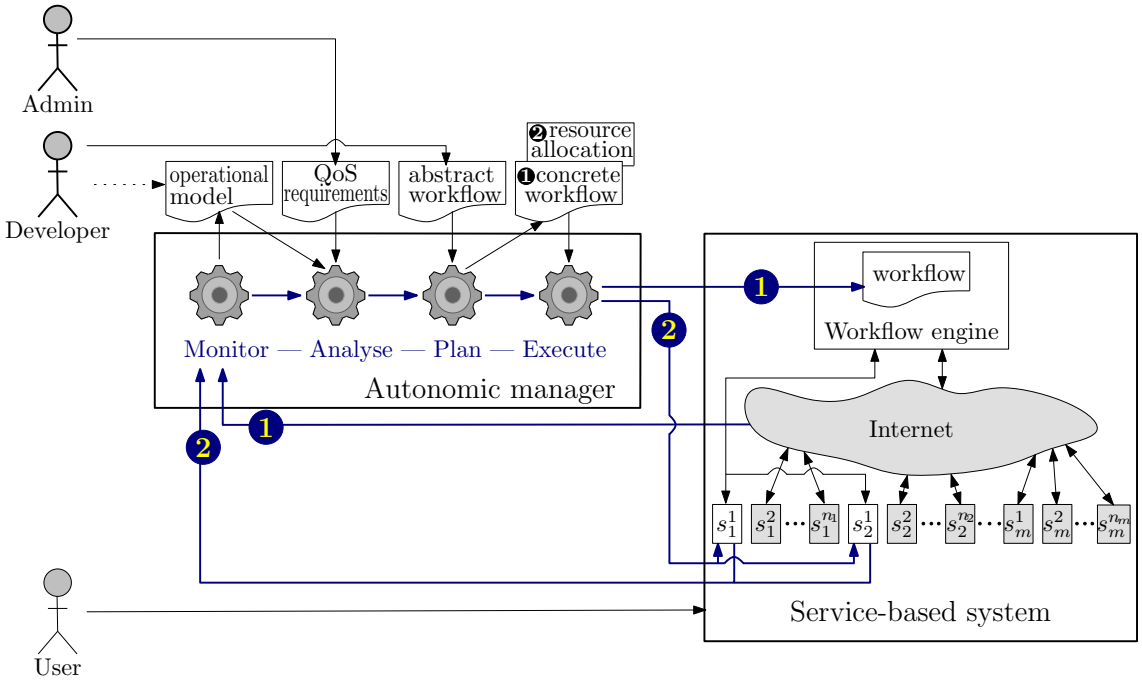



Fig. 3. QoSMOS architecture of an adaptive service-based system. Adaptation can be achieved ❶ through varying the concrete services used by the SBS workflow; and/or ❷ through varying the resources allocated to individual services.

### *3.1.1 Monitoring stage*

The first stage of the MAPE loop involves monitoring either or both of:

1) The performance (e.g., response time) and reliability (e.g., failure rate) of the SBS services. These parameters can be monitored for both in-house and third-party services.
2) The workload of individual concrete services (e.g., their request inter-arrival rates) and the resources allocated to these services (e.g., CPU, memory and bandwidth). Note that this is possible only for in-house services; these characteristics cannot be monitored for third-party services.

This information is used to build and/or to update an *operational model* of the SBS, an initial version of which can be provided by the developer of the service-based system. The model updates can happen periodically or when the monitor identifies significant changes in the parameters of the system. The types of operational models supported by the QoSMOS approach are those

described earlier in Section 2.2, i.e., Markovian models.

*Example:* Each concrete service $s_i^j$, $1 \leq i \leq 3$, $1 \leq j \leq n_i$ from our running example of a TeleAssistance service-based system is characterised by the following parameters:

- $r_i^j \in [0,1]$, the failure rate of the service;
- $c_i^j \geq 0$, the cost associated with each invocation of the service;
- $idem_i^j \in \{\text{true}, \text{false}\}$, a parameter that specifies whether the service is *idempotent*, i.e., can be invoked repeatedly without affecting the outcome of the SBS workflow (but with an increased probability of overall success).

Additionally, the third-party concrete services are characterised by their expected execution time ($t_i^j$); and the in-house concrete service $s_3^1$ by its request inter-arrival rate ($\mu_3^1$) and maximum request service rate ($\lambda_3^1$). The maximum request service rate for this concrete, in-house service represents the request service rate when the service is allocated the maximum amount of CPU resources on the server(s) on which it is running.

Table 2 shows the initial values of these parameters for the TA service-based system; these parameter values are updated in the monitoring stage of the MAPE loop. Notice that the Alarm Service and the Medical Analysis Service are idempotent: for example, the Alarm Service is idempotent since each alarm invocation is associated with an unique identifier. Consequently, issuing the same invocation several times does not produce false alarms because any duplicate requests are ignored. In contrast, the Drug Service is non-idempotent, because of the potential errors that the redundant invocation of its operations might cause.

### 3.1.2 Analysis stage

The operational model from the monitoring stage is then employed to analyse the QoS requirements specified by the SBS administrator. The model is parameterised by the configurable parameters of the SBS, and this analysis step is intended to identify SBS configurations that satisfy the QoS requirements for the system. The analysis step includes a pre-processing step in which the QoS requirements specified by the SBS administrator in a high-level language are converted automatically into formally defined QoS requirements of the form presented in Section 2.1.

*Example:* The high-level requirements for the TA service-based system from our running example comprise reliability- and performance-related requirements. Note that the reliability-related requirements take into account the fact that the average number of alarms associated with a particular patient throughout his or her utilisation of the TA service-based system (i.e., the *lifetime of the system*) is ten; these requirements are described below:

$R_0$ *The probability $P_0$ that at an alarm failure ever occurs during the lifetime of the system is less than $\underline{P}_0 = 0.13$.*

$R_1$ *The probability $P_1$ that a service failure ever occurs during the lifetime of the system is less than $\underline{P}_1 = 0.14$.*

$R_2$ *The probability $P_2$ that a changeDrug or a changeDoses request generates an alarm which fails (i.e., the FAS is not notified) is less than $\underline{P}_2 = 0.007$.*

$R_3$ *Assuming that alarms generated by pButtonMsg have low priority while alarms generated by analyzeData have high priority, it is required that the probability $P_3$ that a high priority alarm fails (i.e., it is not notified to the FAS) is less than $\underline{P}_3 = 0.005$.*

In addition to reliability, we considered the following performance requirements specified by the administrator of the TA system:

$R_4$ *The probability $P_4$ that the number of pending changeDrug requests exceeds 75% of the request queue capacity for the in-house service $DrugService1$ in the long run is less than 0.2.*

$R_5$ *The probability $P_5$ of a changeDrug request being dropped due to the request queue being full during a day of operation is less than 0.05.*

SBS administrators require two types of information in order to specify suitable bounds for the reliability and performance metrics in the QoS requirements. First, they need to know the approximate range of values that the adaptive SBS can achieve for these metrics. Precise information is not necessary because QoSMOS-enabled service-based systems have the ability to notify the administrator if the specified requirements cannot be achieved (this is explained in Section 3.2.4). This information can be obtained by analysing the metrics off-line, based on the initial parameter values from Table 2 or on recent values that the monitoring stage provides for these parameters. Second, the SBS administrators need to have an understanding of the service-level agreements that the SBS users require or are likely to expect from the system. Given these two types of information, SBS administrators can specify QoS requirements that the system can achieve and which its users are likely to deem acceptable. Of course, it is also possible that the user expectations cannot be fulfilled by the SBS, in which case either the system or the user SLA needs to be altered.

### 3.1.3 Planning stage

The planning stage of the QoSMOS MAPE loop uses the results of the analysis stage to build a plan for adapting the configuration of the SBS. The two types of adaptation made possible by the QoSMOS approach and implemented in the execution step of its MAPE loop are described below.

1) Adaptation through changing the workflow implemented by the service-based system. This type of adaptation is possible for all service-based systems considered by the QoSMOS framework, including those that employ third-party services. It requires that the SBS developer provides a workflow that

TABLE 2
Concrete services for the TA service-based system

| Concrete service | Name | Failure rate ($r_i^j$) | Expected execution time† ($t_i^j$) | Maximum request service rate‡ ($\lambda_i^j$) | Request inter-arrival rate‡ ($\mu_i^j$) | Cost ($c_i^j$) | Idempotent ($idem_i$) |
|---|---|---|---|---|---|---|---|
| $s_1^1$ | $AlarmService1$ | 0.03 | 1.1s | – | – | 4.1 | true |
| $s_1^2$ | $AlarmService2$ | 0.04 | 0.9s | – | – | 2.5 | true |
| $s_1^3$ | $AlarmService3$ | 0.008 | 0.3s | – | – | 6.8 | true |
| $s_2^1$ | $MedicalAnalysisService1$ | 0.0006 | 2.2s | – | – | 9.8 | true |
| $s_2^2$ | $MedicalAnalysisService2$ | 0.001 | 2.7s | – | – | 8.9 | true |
| $s_2^3$ | $MedicalAnalysisService3$ | 0.0015 | 3.1s | – | – | 9.3 | true |
| $s_2^4$ | $MedicalAnalysisService4$ | 0.0025 | 2.9s | – | – | 7.3 | true |
| $s_2^5$ | $MedicalAnalysisService5$ | 0.0005 | 2.0s | – | – | 11.9 | true |
| $s_3^1$ | $DrugService1$ | 0.0012 | – | $2.75\text{s}^{-1}$ | $1.2\text{s}^{-1}$ | 0.1 | false |

†Only for third-party concrete services
‡Only for in-house concrete services

is defined in terms of the abstract services needed to implement the intended SBS functionality, i.e., an *abstract workflow*. It is worth emphasising that developing an abstract workflow is identical to developing a concrete workflow, minus the step in which the addresses of the concrete services to use are decided. This last step is carried out at run time, when the analysis results are used to map the abstract services within this original workflow to concrete services—a process that takes place during the planning stage. Note that it is possible to restrict this adaptation to a subset of the workflow services by associating a single concrete service with each abstract service that does not belong to this subset. We actually envisage this as a common use case, and we will illustrate it by means of a number of experiments in Section 4.3. This use case is supported without having to specify in the abstract workflow which services should be considered for runtime adaptation and which services should always be implemented using the same concrete service.

2) Adaptation through modifying the resources allocated to individual services. When internally-administered services are used to implement the SBS, it may be possible to adapt the resources allocated to these services in line with the variation in their workloads and in the QoS requirements for the system. Potential applications of this type of adaptation include: achieving performance-related QoS requirements with minimal cost and environmental impact; and achieving dependence QoS requirements by running services across a variable number of servers for redundancy purposes.

The mapping of abstract to concrete services within the QoSMOS architecture can be performed using one of the *mapping patterns* described below:

- In a *single mapping* (SGL), a concrete service with suitable performance, reliability and cost characteristics is used for the abstract service.
- In *sequential one-to-many mapping* (SEQ), an abstract service is mapped to a sequence of concrete services. When the workflow is executed, these services are used one at a time, starting with the first service in the sequence and carrying on through the sequence until either a non-erroneous response is obtained or all services in the sequence fail to respond successfully. This concretisation of an abstract service is useful for improving the reliability-related QoS of an SBS, but can elongate its response time. Note that the sequence of concrete services for a SEQ mapping pattern may include several instances of the same concrete service, or even a single concrete service to be invoked repeatedly for redundancy purposes.
- Finally, in *parallel one-to-many mapping* (PAR), an abstract service is mapped to a set of concrete services, all of which are called during the execution of the workflow. This ensures that an increase in the reliability-related QoS metrics is obtained without impacting the SBS response time, but potentially at a higher cost.

QoSMOS supports the use of a different mapping pattern ($mp_i \in \{\text{SGL}, \text{SEQ}, \text{PAR}\}$) for each abstract service $as_i$, $1 \leq i \leq m$, that is idempotent. For non-idempotent services, the only mapping pattern that can be used is SGL.

*Example:* Consider again the TA service-based system from our running example. The configurable parameters of this system are:

(a) the mapping patterns $mp_i \in \{\text{SGL, SEQ, PAR}\}$, $1 \leq i \leq m$, used for the three abstract TA services (note that $mp_3 = \text{SGL}$ at all times since the Drug Service is non-idempotent);

(b) the concrete service sequences $\langle s_i^{j_1}, s_i^{j_2}, \ldots, s_i^{j_{S_i}} \rangle$, $1 \leq i \leq m$, used to implement the three abstract TA services, where $S_i \geq 1$ and $\{j_1, j_2, \ldots, j_{S_i}\} \subseteq \{1, 2, \ldots, n_i\}$ (note that $S_i = 1$ for all abstract TA services $as_i$ for which $mp_i = \text{SGL}$);

(c) the $cpu_3^1 \in [0, 1]$ fraction of CPU to be allocated to the in-house TA service $s_3^1$; a CPU fraction of $cpu_3^1 = 1$ corresponds to the maximum amount of CPU that the service can be allocated, and to the

service request rate given by the $\lambda_3^1$ parameter in Table 2 (cf. Example 2 in Section 3.1.1). Note that the service response time varies linearly with the value assigned to this parameter, which can therefore be used to adapt the service behaviour to its request arrival rate and to the system requirements.

The values of all these parameters are decided in the planning stage of each iteration of the MAPE loop for the QoSMOS-enabled TA system. The parameter values corresponding to a feasible configuration for the TA system are presented in Table 6 later in the article.

#### 3.1.4 Execution stage

If a new concrete workflow was derived in the planning stage of the QoSMOS MAPE loop, this workflow is used as a replacement for the one that the workflow engine was previously executing. Given that an increasing number of workflow engines support dynamic workflow modification (e.g., [35] and, more recently, [63]), realising this functionality in QoSMOS involves exploiting the capabilities of such existing engines.

When a new allocation of resources to concrete services was decided during the planning stage, this allocation is enforced during the execution stage of the MAPE loop. Depending on the platform(s) used to run the services affected by the change in resource allocation, this operation may involve modifying the parameters of an application server; starting, stopping or migrating virtual machines; or using dedicated resource management mechanisms. One such mechanism is described in [38], where the resources allocated to individual in-house services are dynamically adjusted by varying the number of servers running these services and the distribution of the service invocations across all these servers. Similarly, the results in [88] represent important steps towards enriching conventional service-oriented architectures with dynamic resource provisioning capabilities.

The purpose of QoSMOS is to provide the infrastructure necessary to exploit such functionality in ways that ensure compliance with the high-level QoS requirements of service-based systems rather than to implement this low-level dynamic resource provisioning that other research groups are working on. Note also that when such functionality is not present explicitly, it can be emulated to a large degree. A standard workflow engine can run multiple workflows, and dynamic workflow modification can be emulated by running a new version of a QoSMOS workflow next to the old version; all new executions of the workflow would use the latest version, with old versions being removed when all their running instances finish execution. Similarly, (coarse-grained) dynamic resource allocation can be achieved by load balancing the requests for a concrete service among a dynamically chosen set of physical servers that run instances of the service.

*Example:* For the TA service-based system from our running example, the execution stage involves:

1) Synthesising the version of the BPEL workflow from Figure 2 that corresponds to the $mp_i$ and $\langle s_i^{j_1}, s_i^{j_2}, \ldots, s_i^{j_{S_i}} \rangle$, $1 \leq i \leq 3$, parameter values decided in the planning stage, and supplying this new concrete workflow to the workflow engine.
2) Reconfiguring the in-house service $s_3^1$ so that a $cpu_3$ fraction of the maximum CPU resources that are available for running this service are actually allocated to it.

For instance, given the TeleAssistance SBS configuration from Table 6, the execution stage synthesises a concrete TA workflow that employs $AlarmService2$, the SEQ combination $\langle MedicalAnalysis1, MedicalAnalysis4 \rangle$ and $DrugService1$; and allocates a CPU fraction of $0.467$ to the in-house service $DrugService1$.

TABLE 3
Tools whose extended versions were integrated to implement the QoSMOS MAPE loop

| MAPE loop stages: | | | |
|---|---|---|---|
| monitoring | analysis | planning | execution |
| KAMI [40] | PRISM [72], [74]; ProProST [49] | PRISM [72], [74]; GPAC [20] | GPAC [20] |

### 3.2 Realisation of the QoSMOS architecture using existing tools

This section describes a practical realisation of the QoSMOS architecture that is built through the integration of extended versions of software tools previously developed by the authors. These tools are listed in Table 3, and their responsibilities and dependencies are depicted in Figure 4. We start with brief descriptions of the tools in Sections 3.2.1–3.2.4, then present their integration into a realisation of the QoSMOS architecture in Section 3.2.5.

#### 3.2.1 KAMI

KAMI is a framework conceived for run-time modeling of SBS systems. It focuses on non-functional models which are typically dependent on (numerical) parameters that are: (1) provided a-priori by domain experts or (2) extracted from other similar systems. At design time, such parameters can be unknown or imprecise. Even if these values are initially correct, they can change during the operating life of the system. Consequently, accurate models must be updated over time. KAMI focuses on keeping non-functional models up to date with the current behavior of the modeled system by updating model parameters. Updated models can then be used to re-check at run-time requirements initially verified at design time to guarantee the correctness of the system.

KAMI starts considering the initial parameters that characterize the model. These values are derived using

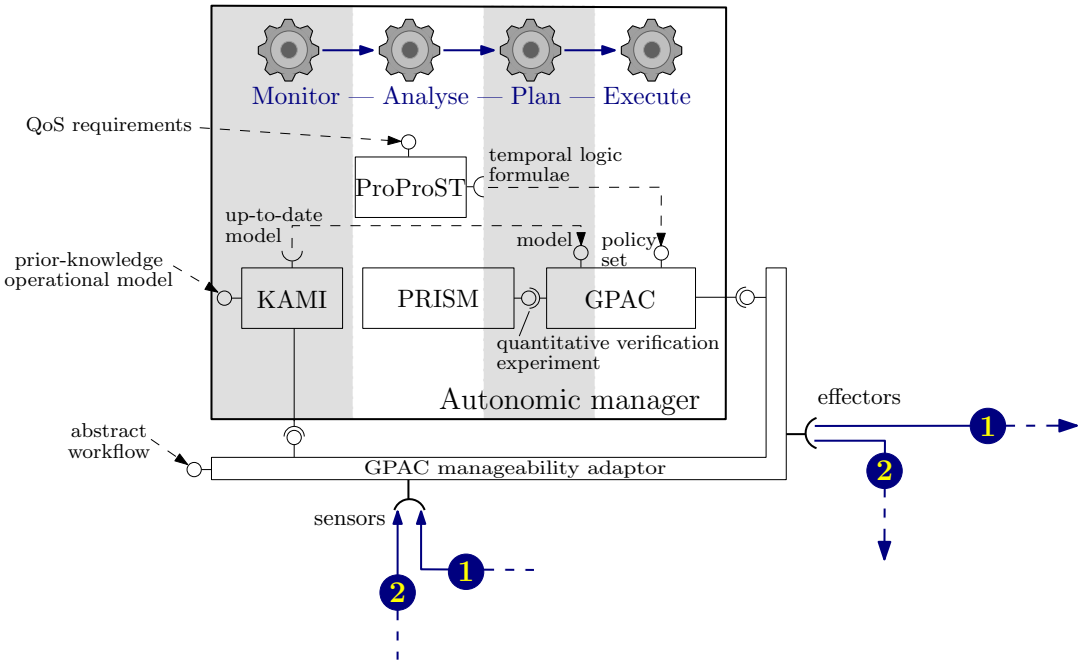



Fig. 4. Realisation of the generic QoSMOS architecture in Figure 3 using existing tools.

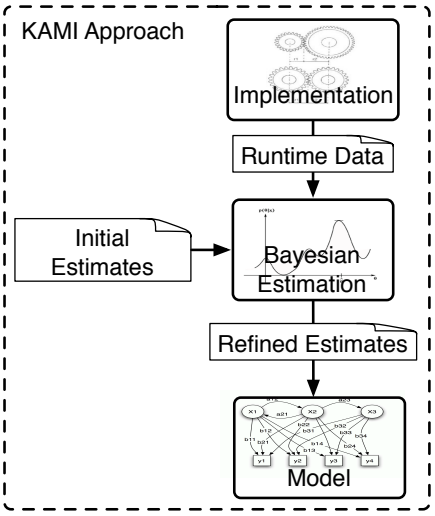



Fig. 5. The KAMI Approach.

estimates of the expected behavior of the system. This initial (imprecise or even incorrect) knowledge is called *"a-priori knowledge"*. At run-time the framework records all the events which occur in the system that are relevant from the modeling point of view. Focusing on DTMCs and CTMCs as, respectively, reliability models and performance models, KAMI relies on run-time data representing transitions among states of the model associated with their execution time. For example, in SBS systems such events are service invocations associated with their response time.

KAMI's input data are represented in a tuple-based textual format (as described in [40]) which is independent from the actual implementation of the system generating data. In the SBS domain several approaches (e.g., [11], [15], [17], [63], [84]) can be adopted to monitor and collect the needed data. In particular, we rely on the approach of Baresi et al. [11] to obtain this run-time information.

By collecting run-time data from running instances of the system, KAMI feeds a Bayesian estimator [14] as defined in [40] in charge of producing new estimates of the model parameters. In this scenario run-time data represent the *"a-posteriori knowledge"* engineers have with respect to the system being modeled. Indeed, KAMI is in charge of mixing appropriately a-priori knowledge with a-posteriori evidences by continuously updating model parameters to achieve increasingly better accuracy. Up-to-date models provide a more accurate representation of the current behavior of the system and allow engineers to automatically check the system requirements while the system is running. The overall approach is illustrated in Figure 5.

The accuracy of the initial values adopted as a-priori knowledge does not affect the effectiveness of estimates of model parameters (i.e., a-posteriori knowledge). In the extreme case, random values might be adopted as a-priori knowledge. In this scenario the KAMI's Bayesian estimation produces the correct estimates even if the convergence of estimates is slower (i.e., it requires more run-time data). In particular, a smoothing parameter tunes the estimation process and convergence speed by adjusting the *trustworthiness* on a-priori knowledge [40].

The current version of KAMI supports DTMCs [40] and Queueing networks [45]. The QoSMOS MAPE loop employs KAMI to perform the automatic model parameter tuning during its monitoring stage (Figure 4).

*Example:* For the TeleAssistance SBS, for instance, the KAMI component of QoSMOS is monitoring the failure rates for all concrete services, so that variations from the design-time predicted values in Table 2 can be taken into account in the adaptation process. The operation of KAMI in the context of the TA system is described in detail in Section 4.2.

### 3.2.2 PRISM

PRISM [72], [74] is an open-source probabilistic model checker developed originally at the University of Birmingham, and currently supported and extended at the University of Oxford. The tool supports the analysis of a growing number of model types, including discrete- and continuous-time Markov chains (DTMCs and CTMCs), Markov decision processes (MDPs), and extensions of these models with costs and rewards.

The models to be analysed are specified in the PRISM modelling language, which is based on the Reactive Modules formalism of Alur and Henzinger [2]. The properties to be established are specified using PCTL (Probabilistic Computation Tree Logic) [53] for DTMCs and MDPs, and CSL (Continuous Stochastic Logic) [8] for CTMCs.

The tool works by first building a symbolic, MTBDD (multi-terminal binary decision diagram) representation of the reachable state space of the analysed model [72]. It then performs the analysis by induction over syntax, being capable of handling both *bound* properties—i.e., deciding whether a probability is above or below a specified threshold; and *quantitative* properties—i.e., calculating the actual probability of an event or the expectation for cost/reward formulas. Particularly important for its integration in the QoSMOS architecture, PRISM supports the concept of *experiments*, which allows the automated analysis of several versions of a parameterised model. We will use this capability within the QoSMOS MAPE loop, to carry out automatically the analysis of a range of possible configurations for a service-based system.

The model checking algorithms employed by PRISM involve a combination of *graph-theoretical algorithms* and *numerical computation*. The first type of algorithms operate on the underlying graph structure of the analysed Markov model, e.g., to determine the reachable states within a model. Numerical computation (typically using iterative methods) is required for the solution of linear equation systems and the calculation of the transient probabilities of Markov chains.

The probabilistic model checker PRISM has been used in a large number of case studies that spawn application domains ranging from communication protocols and security systems to biological systems and dynamic power management. Many of these case studies are presented in detail on the PRISM web site (www.prismmodelchecker.org). An extensive, independent performance analysis of a broad selection of probabilistic model checkers [61] ranked PRISM as the best tool for the quantitative analysis of large models such as the ones encountered in the adaptive service-based systems targeted by our QoSMOS work.

*Example:* Since the QoS requirements for the QoSMOS-enabled TA system include both reliability- and performance-related requirements, two PRISM operational models are used.

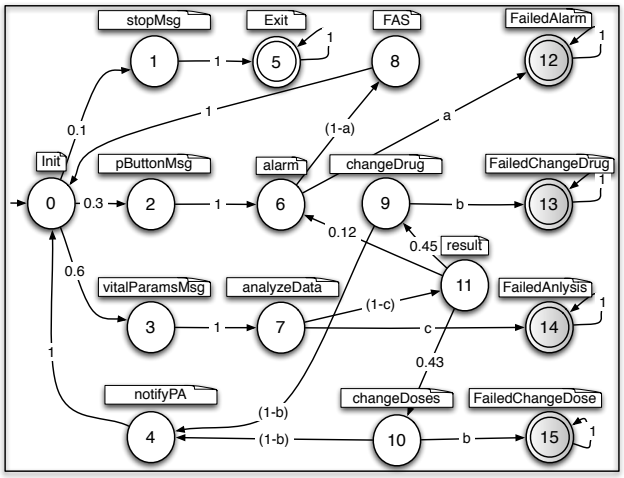



Fig. 6. TeleAssistance DTMC model.

First, the DTMC model depicted in Figure 6 is used for the analysis required to achieve the reliability QoS requirements $R_0$ to $R_3$. This model follows the structure of the BPEL workflow, and assigns probabilities to branches and failure probabilities to service invocations (failures are represented by states highlighted in grey). Our approach relies on initial estimates for transition probabilities that come from domain experts and from monitoring previous versions of the system. Transition probabilities corresponding to service failure rates are unspecified in the DTMC model and represented by the unknown parameters $a$, $b$ and $c$ because they depend on the mapping patterns and concrete services selected by the QoSMOS MAPE loop.

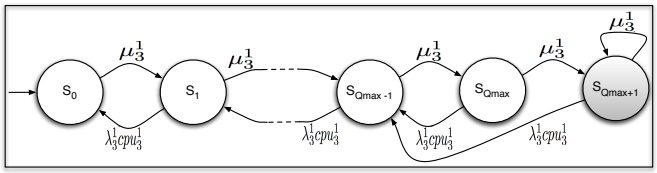



Fig. 7. CTMC model for the in-house concrete service $s_3^1$.

Second, the CTMC model shown in Figure 6 is employed for the analysis supporting the implementation of the performance QoS requirements $R_4$ and $R_5$. This model corresponds to the only in-house service in our SBS, i.e., $s_1^3$ or $DrugService1$. The parameters of this CTMC model are: $Q_{max} > 0$, the size of the request queue for the service; $\mu_3^1 > 0$, the request arrival rate; $cpu_3^1 \in [0, 1]$, the fraction of CPU resources allocated to

the service on the server on which it is run; and $\lambda_3^1 > 0$, the request service rate corresponding to $cpu_3^1 = 1$. Accordingly, states $S_i$, $0 \leq i \leq Q_{max}$, from the CTMC model in Figure 7 correspond to the service request queue containing $i$ requests with no request being dropped; and state $S_{Q_{max}+1}$ corresponds to the queue being full and requests being dropped.

### 3.2.3 ProProST

To ease the formalization of QoS properties as required by the QoSMOS architecture, the idea of specification patterns [39], [68] has been recently investigated for probabilistic logics [49]. The outcome of an investigation of 152 properties from academia and 48 properties from industrial requirements specifications resulted in a pattern-based specification system called ProProST (Probabilistic Property Specification Templates). This specification system contains eight generic patterns that covered a large percentage of the investigated academic (147 out of 152) and industrial properties (46 out of 48). The eight property specification patterns including their instance counts in academia and industry are presented in Table 4. Please note, that seven of the academic properties are composites of the eight patterns that are counted separately.

| Pattern Name | Logical Formulation | Academia | Industrial |
|---|---|---|---|
| Transient State Probability | $\mathcal{P}_{\bowtie p}[\Diamond^{[t,t]}\Phi]$ | 6 | 0 |
| Steady State Probability | $\mathcal{S}_{\bowtie p}[\Phi]$ | 18 | 1 |
| Probabilistic Invariance | $\mathcal{P}_{\bowtie p}[\Box^{\leq t}\Phi]$or $\mathcal{P}_{\overline{\bowtie}(1-p)}[\Diamond^{\leq t}\neg\Phi]$ | 6 | 18 |
| Probabilistic Existence | $\mathcal{P}_{\bowtie p}[\Diamond^{\leq t}\Phi]$or $\mathcal{P}_{\bowtie p}[trueU^{\leq t}\Phi]$ | 57 | 9 |
| Probabilistic Until | $\mathcal{P}_{\bowtie p}[\Phi_1 U^{\leq t}\Phi_2]$ | 30 | 2 |
| Probabilistic Precedence | $\mathcal{P}_{\bowtie p}[\neg\Phi_2 \mathcal{W}\Phi_1]$or $\mathcal{P}_{\overline{\bowtie}(1-p)}[\neg\Phi_1 U (\neg\Phi_1 \wedge \Phi_2)]$ | 1 | 0 |
| Probabilistic Response | $\mathcal{P}_{\geq 1}[\Box(\Phi_1 \Rightarrow \mathcal{P}_{\bowtie p}(\Diamond^{\leq t}\Phi_2))]$ | 18 | 13 |
| Probabilistic Constrained Response | $\mathcal{P}_{\geq 1}[\Box(\Phi_1 \Rightarrow \mathcal{P}_{\bowtie p}(\neg\Phi_2 U^{\leq t}\Phi_3))]$ | 4 | 3 |

TABLE 4
Probabilistic specification patterns

Within the QoSMOS framework the ProProST specification system can be used for the initial translation of QoS requirements into a probabilistic temporal logic or during the system run-time to add new QoS requirements or update existing ones. These two tasks are supported with the ProProST wizard (Figure 8), which helps SBS administrators to select the appropriate pattern and clearly define a QoS requirement in a probabilistic temporal logical formula. The ProProST wizard is based on the structured English grammar that is presented in [49]. As a result, new QoS requirements can be easily specified, or existing QoS requirements could be relaxed or strengthened, by SBS administrators with limited knowledge of probabilistic temporal logics.

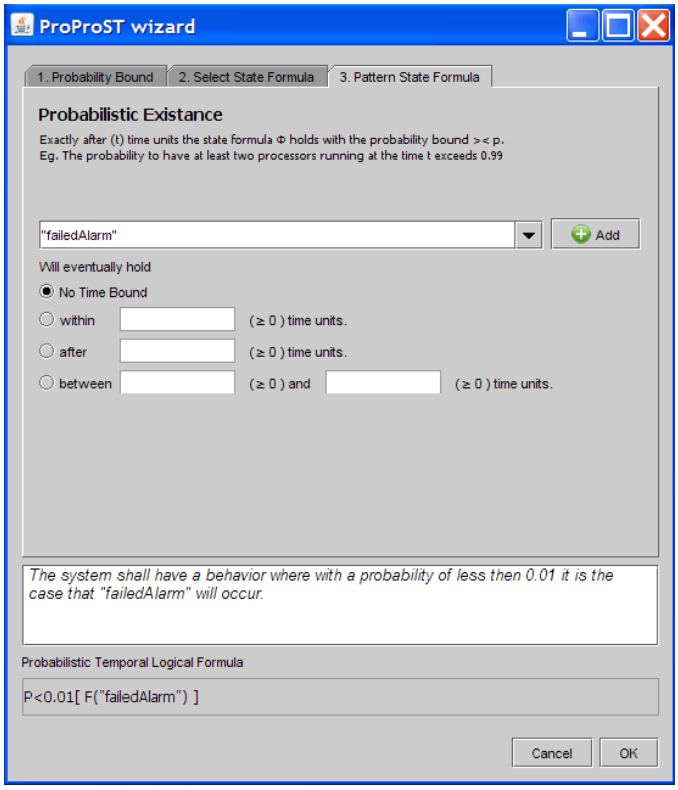



Fig. 8. The ProProST Wizard.

*Example:* The QoS requirements $R_0$–$R_5$ for the TeleAssistance system from our running example (Section 3.1.2) are ProProST pattern instances defined as probabilistic temporal logical formulae based on the labels defined in the operational models from Figures 6 and 7. $R_0$, $R_1$ and $R_5$ are Probabilistic Existence properties, $R_2$ and $R_3$ are filtered Probabilistic Until properties, and $R_4$ is a Steady State property. To identify the specific pattern and to formulate the probabilistic temporal logical formulae, the structured English grammar and the ProProST wizard are used. As an example, the structured English representation for requirement $R_0$ resulting from the use of the wizard is:

```
The system shall have a behavior
where with a probability of less
than 0.13 it is the case that
"failedAlarm" will occur.
```

This sentence corresponds to the temporal logic formula $\mathcal{P}_{\leq 0.13}[\Diamond \text{“}failedAlarm\text{”}]$. As a result of the structured process for the formulation of probabilistic properties, the SBS administrator derived the following temporal logic formulae that correspond to the QoS requirements that the TA service-based system must satisfy:

$$
\begin{aligned}
&R_0 : \mathcal{P}_{\leq 0.13}[\Diamond \text{“}failedAlarm\text{”}] \\
&R_1 : \mathcal{P}_{\leq 0.14}[\Diamond \text{“}failure\text{”}] \\
&R_2 : \mathcal{P}_{\leq 0.007}[(\neg \text{“}stopMsg\text{”} \,\&\, \neg \text{“}pButtonMsg\text{”} \\
&\quad \,\&\, \neg \text{“}FAS\text{”}) U \text{“}failedAlarm\text{”} \\
&\quad \{\text{“}changeDrug\text{”} \mid \text{“}changeDoses\text{”}\}] \\
&R_3 : \mathcal{P}_{\leq 0.005}[(\text{“}alarm\text{”} | \text{“}analyzeData\text{”} | \text{“}result\text{”}) \\
&\quad U \text{“}failedAlarm\text{”} \{\text{“}analyzeData\text{”}\}] \\
&R_4 : \mathcal{S}_{\leq 0.2}[q/Q_{max} > 0.75] \\
&R_5 : \mathcal{P}_{\leq 0.05}[\Diamond^{[0,86400]} \text{“}dropped\text{”}]
\end{aligned}
\tag{1}
$$

### 3.2.4 GPAC

GPAC (General-Purpose Autonomic Computing) is a tool-supported methodology for the model-driven development of self-managing IT systems [20]. The core component of GPAC is a reconfigurable policy engine capable of augmenting existing IT systems with a MAPE autonomic computing loop. The policy engine comprises multiple software components that are reused within the MAPE loop of any such application, and a small number of application-specific software components that are generated automatically at run-time.

The automated code generation techniques employed by GPAC are based on a specification supplied to the policy engine as part of a run-time configuration step [21]. This specification is termed a *GPAC system model*, and describes formally (a) the characteristics of every relevant parameter of the system, including its name, type (i.e., to be `monitored` or `configured` by the MAPE loop) and value domain (e.g., `integer`, `double` or `string`); and (b) the run-time behaviour of the system. The latter element of the GPAC system model corresponds to the operational model from the QoSMOS architecture in Figure 3 and can be specified by means of quality evaluation models such as those described in Section 2.2. The policy engine is implemented as a service-oriented architecture, and employs advanced object-oriented technology capabilities such as reflection-oriented programming [95] and generic programming [44] in its handling of systems whose characteristics are unknown until run-time.

Another key component of GPAC is a tool for the model-driven development of the thin software interfaces (i.e., *manageability adaptors*) that the policy engine uses to monitor and control the parameters of the managed system [21]. This tool was used to speed up the development of autonomic solutions in several application domains [20], and was recently integrated into a GPAC environment for the computer-aided development of autonomic systems [22].

The high-level system goals whose realisation is supported by the GPAC MAPE loop include *multi-objective utility optimisations* in which $N \geq 1$ configurable system parameters $c_1$, $c_2$, …, $c_N$ are dynamically adjusted to maximise the *utility* of the system. These utility optimisations are expressed as

$$
\begin{aligned}
(c'_1, c'_2, \ldots c'_N) = \underset{(x_1, x_2, \ldots, x_N) \in C_1 \times C_2 \times \ldots \times C_N}{\operatorname{argmax}} & \; utility(c_1, \\
& c_2, \ldots, c_N, st_1, st_2, \ldots, st_M, x_1, x_2, \ldots, x_N),
\end{aligned}
\tag{2}
$$

where $c_i$ and $c'_i$, $1 \leq i \leq N$, represent the current and the new values of $i$-th configurable parameter; $C_i$, $1 \leq i \leq N$, is the value domain for this parameter; and $st_1$, $st_2$, …, $st_M$ represent the $M \geq 1$ system parameters monitored but not modified by the MAPE loop. The utility function has the form

$$
\begin{aligned}
utility(c_1, c_2, \ldots, c_N, st_1, st_2, \ldots, st_M, \\
x_1, x_2, \ldots, x_N) = \textstyle\sum_{i=1}^{r} w_i \, objective_i,
\end{aligned}
\tag{3}
$$

where the *weights* $w_i \geq 0$, $1 \leq i \leq r$, are used to express the trade-off among $r \geq 1$ system objectives. Each objective function $objective_i$, $1 \leq i \leq r$, is an analytical expression of (configurable and monitored-only) system parameters specified in the GPAC system model. This can include formally defined QoS requirements such as those presented in Section 2.1.

Another type of autonomic computing policy supported by GPAC is an *action policy* of the form

$$
\begin{aligned}
&\text{if } condition(c_1, c_2, \ldots, c_N, st_1, st_2, \ldots, st_M) \text{ then} \\
&\quad (c'_1, c'_2, \ldots c'_N) = (x_1, x_2, \ldots, x_N)
\end{aligned}
\tag{4}
$$

where $x_i \in C_i$, $1 \leq i \leq N$, represents a predefined value for the $i$-th configurable system parameter. The policy engine is required to enforce these predefined parameter values when $condition$ is satisfied. As described in the next section, our QoSMOS prototype employs this simple policy to alert the SBS administrator when the utility achievable through the optimisation in eq. (2) falls below a given threshold.

In recent work, the GPAC tools and the probabilistic model checker PRISM were used together successfully to develop autonomic systems involving dynamic power management and adaptive allocation of data-center resources [23]. Unlike QoSMOS, this related project required that the individual objectives in (3) were specified as low-level, reward-based PRISM properties; employed a fixed PRISM model to describe the behaviour of the system; and targeted the development of adaptive system in other application domains than service-based systems.

*Example:* Specifying a utility function (3) for the TeleAssistance system involves taking into account its three objectives: achieving the QoS requirements $R_0$–$R_5$ from (1); minimising the cost of third-party concrete services; and minimising the resources used by in-house services. These three objectives are expressed formally as:

$$
\begin{aligned}
&objective_1 = \textstyle\prod_{j=0}^{5} \mathrm{goal}(R_j) \\
&objective_2 = -\textstyle\sum_{i=1}^{3} \sum_{x=1}^{S_i} c_i^{j_x} \\
&objective_3 = -cpu_3^1
\end{aligned}
\tag{5}
$$

where the function $\mathrm{goal} : \{\texttt{false}, \texttt{true}\} \rightarrow \{0, 1\}$ is defined by $\mathrm{goal}(\texttt{false}) = 0$ and $\mathrm{goal}(\texttt{true}) = 1$. Notice that this definition ensures that $objective_1$ has value 1 if all QoS requirements are satisfied, and value 0 otherwise.

An upper bound

$$S \geq S_1, S_2 \tag{6}$$

is placed on the maximum number of concrete services used by a SEQ or PAR mapping pattern in the QoSMOS-enabled TA system, and the weights for the utility function

$$utility = \sum_{i=1}^{3} w_i objective_i \tag{7}$$

are chosen such as to ensure that any configuration for which $objective_1 = 1$ corresponds to a higher system $utility$ than the $utility$ of any other configuration for which $objective_1 = 0$. Given the costs of the concrete service in Table 2, the weights below ensure that $utility < 0$ whenever any of the $R_i$ constraints is not satisfied and $utility > 0$ otherwise:

$$\begin{aligned} w_1 &= 100S \\ w_2 &= 1 \\ w_3 &= 10. \end{aligned} \tag{8}$$

This property can be used to define the $condition$ of an instance of the action policy (4) that raises an alarm notifying the SBS administrator about SLA failures:

$$\text{if } utility < 0 \text{ then } SBS\_admin\_alarm = true. \tag{9}$$

As explained in more detail in Section 4, eqs. (7) and (9) specify the concrete utility function and action policy used by the QoSMOS-enabled TA system, respectively.

#### 3.2.5 Tool integration

In our realisation of the QoSMOS architecture, the tools presented so far are integrated as illustrated in Figure 4. A QoSMOS-based service-based system takes three parameters as input: (a) the QoS requirements; (b) the abstract SBS workflow; and (c) the prior-knowledge model describing the behaviour of the SBS. The way in which each of these parameters is employed by the prototype QoSMOS architecture is described below.

The QoS requirements specified by the system administrator are supplied to the ProProST component of QoSMOS, and converted into temporal logic formulae that are used to define the policy set for a running instance of the GPAC policy engine. Since the policy engine is implemented as a web service, the QoS objectives can be modified at run time by calling the appropriate GPAC web method. Such run-time objective changes are taken into account automatically in the next iteration of the MAPE loop. In this respect, objective changes are treated similarly to changes in the system state, and do not require any downtime. Note that this ability to modify the QoS objectives as and when needed is a characteristic that sets QoSMOS apart from other approaches to building adaptive service-based systems.

The prior-knowledge operational model consists of a set of PRISM models that is supplied to the KAMI component of the prototype. KAMI updates this model continuously based on its monitoring of the service-based system, and ensures that the GPAC policy engine is kept up to date about all these updates of the operational model. The policy engine uses the PRISM models and the QoS requirements obtained from the KAMI and ProProST components to perform a number of PRISM experiments. Each such experiment analyses one of the temporal logic formulae from the SBS objectives for all possible values that can be assigned to the configurable SBS parameters. In the planning stage of the MAPE loop, the GPAC policy engine parses the results of the PRISM experiments and uses them to choose optimal values for the configurable SBS parameters. To achieve this, QoSMOS employs the straightforward exhaustive search algorithms described in our previous work in [23]. Note that this approach works well for the service-based systems targeted by QoSMOS due to the relatively small configuration space that has to be explored by these searches (e.g., the set of concrete services that can be used to implement each abstract service has typically only a few elements).

The abstract workflow is used to configure a GPAC manageability adaptor developed as described in Section 3.2.4. In the execution stage of the MAPE loop, the optimal values chosen for the configurable SBS parameters are applied to the abstract workflow and thus converted by this adaptor into a concrete workflow that is supplied to the SBS workflow engine; and/or into new resource allocations for the services run in-house.

Note that a potential limitation of the approach is the overhead associated with the continuous redeployment of the concrete workflow in case of constant parameter and/or service changes. This limitation is overcome in our QoSMOS implementation by using a hybrid event-driven/periodical workflow adaptation technique. This technique involves performing an iteration of the MAPE loop only if (a) a configurable time interval (i.e., the autonomic manager *period*) has elapsed and (b) the SBS model or the QoS objectives have changed since the previous MAPE-loop iteration. For the experiments described in the article, an autonomic manager period of 2 seconds was successfully used.

## 4 VALIDATION

This part of the article describes a series of experiments that evaluate the effectiveness and scalability of the QoSMOS approach by applying it to the TeleAssistance system used as a running example in the first part of the article. First, we will describe in more detail the implementation of the MAPE loop for the TeleAssistance service-based system in Section 4.1. To establish the effectiveness of the approach, we test how well the QoSMOS-enabled version of the TA system adapts to changes that infringe its QoS requirements in the absence

of adaptation (Section 4.2). To explore the scalability of QoSMOS, we measured its overheads for a range of extensions of the TA service-based system (Section 4.3).

### 4.1 QoSMOS MAPE loop for the TeleAssistance SBS

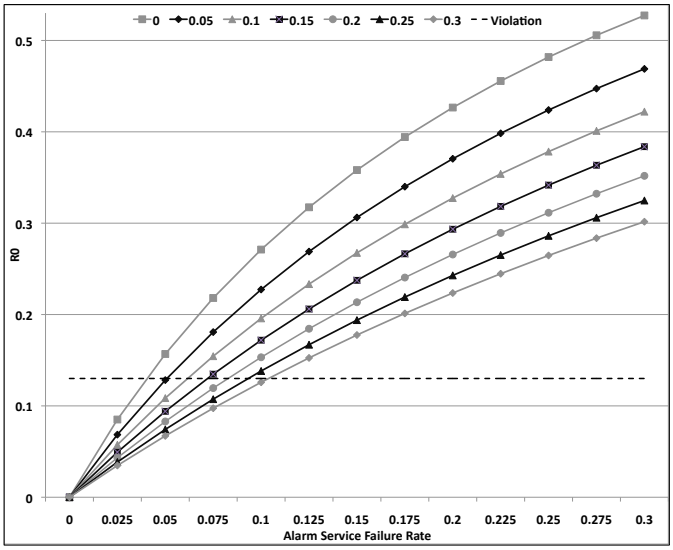
**Monitoring Stage.** In this stage of the MAPE loop, KAMI ensures that any changes in the values of the TA parameters from Table 2 are reflected in the operational model that QoSMOS employs in the subsequent stages of the MAPE loop.

**Analysis Stage.** In this stage, the QoSMOS MAPE loop analyses the PCTL and CSL properties $R_0$–$R_5$ from (1); remember that these properties correspond to the reliability- and performance-related QoS requirements for the TA system. This analysis is performed by running background PRISM experiments using the DTMC and CTMC models in Figures 6 and 7, respectively.

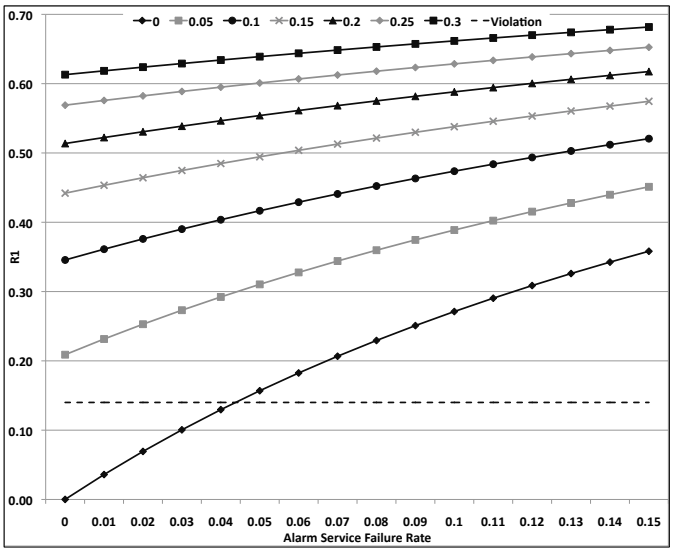
We will first describe the PRISM experiments associated with the analysis of the reliability-related PCTL formulae $R_0$ to $R_3$ for the system. These PRISM experiments consider all possible mapping patterns and service bindings for the SBS workflow, as described in Section 3.2.4. Note that each possible SBS configuration corresponds to certain values for the DTMC parameters $a$, $b$, $c$ from Figure 6, and may or may not satisfy requirements $R_0$ to $R_3$.

Several results from the PRISM experiments performed for requirements $R_0$ and $R_1$ are depicted in Figure 9a and Figure 9b, respectively. To make this graphical representation possible, we fixed a number of configurable SBS parameters, namely $mp_1 = mp_2 = mp_3 = SGL$ (i.e., the "single" mapping pattern was considered for all abstract services in the system) and the concrete service used to implement the MedicalAnalysisService was chosen to be $MedicalAnalysisService1$. The configurable SBS parameters that were varied in the PRISM experiments shown in Figures 9a and 9b are the other concrete services, i.e., AlarmService and DrugService. The horizontal dashed lines in the two graphs show the thresholds which divide valid configurations from invalid ones: the requirements are met for all configurations on or below these lines, and violated for all configurations above the lines.

Another set of results from the PRISM analysis for requirement $R_0$ is presented in Figure 10. This time, the mapping patterns for the three abstract services were chosen to be $mp_1 = SEQ$ and $mp_2 = mp_3 = SGL$, i.e., the sequential one-to-many mapping pattern was considered for the idempotent Alarm Service. Several possible sequences of concrete alarm services (shown in Table 5) were considered, and the failure rate for the concrete service $DrugService1$ was set to the value in Table 2. The graph in Figure 10 depicts the variation of the probability of failure from requirement $R_0$ for different failure rates for the Medical Service and the




(a) R0 Evaluation




(b) R1 Evaluation

Fig. 9. Reliability Evaluation of Requirement R0 and R1

sequences of concrete alarm services from Table 5. Again, the horizontal dashed line partitions the possible SBS configurations considered into valid (those on or below the line) and invalid ones (those above the dashed line). As expected, the use of SEQ one-to-many mappings whose sequences of concrete services contain multiple elements does lead to more valid configurations, albeit at a higher cost.

Note that when the PRISM experiments are run as part of the QoSMOS MAPE loop, none of these parameters is fixed, hence the analysis takes into account all possible values for the configurable parameters of the system (subject to the upper bound (6) placed on the length of the sequences of concrete services used for the SEQ and PAR mapping patterns). The results of these experiments form a multi-dimensional surface in a hy-

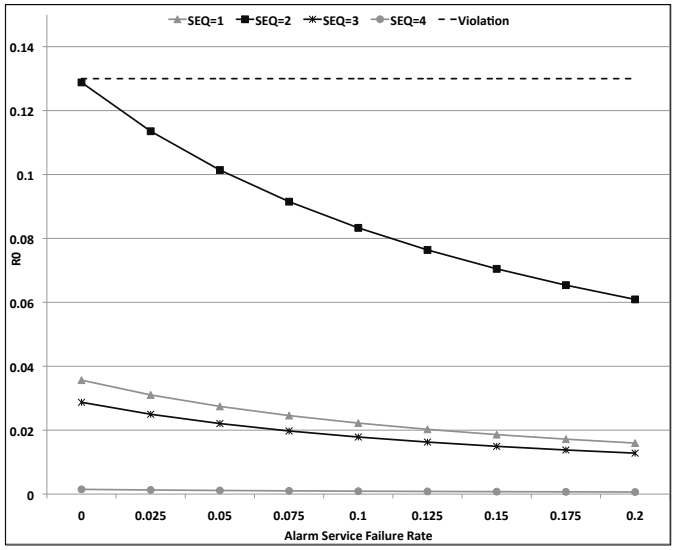



Fig. 10. Reliability Requirement Evaluation comparing different SEQ configurations

TABLE 5
Combination of Alarm Services with different sequential one-to-many mappings

| SEQ Index | Number of Services | Services | Aggregate Failure Rate |
|---|---|---|---|
| 1 | 1 | $s_1^1$ | 0.01 |
| 2 | 1 | $s_1^2$ | 0.04 |
| 3 | 1 | $s_1^3$ | 0.008 |
| 4 | 2 | $s_1^1$, $s_1^2$ | 0.0004 |
| 5 | 2 | $s_1^1$, $s_1^3$ | 0.00008 |
| 6 | 2 | $s_1^2$, $s_1^3$ | 0.00032 |
| 7 | 3 | $s_1^1$, $s_1^2$, $s_1^3$ | 0.0000032 |

perspace whose dimensionality is given by the number of configurable SBS parameters; the graphs in Figures 9 and 10 represent the intersections of these surfaces with hyperplanes obtained by fixing the value of several SBS parameters. Furthermore, all PRISM experiments are run in the background, using the command-line interface of the probabilistic model checker. This ensures that the analysis result is generated in an ASCII format that is easy to parse in the planning stage of the QoSMOS MAPE loop, and eliminates the overheads associated with producing the graphs.

The analysis stage also involves PRISM experiments that analyse the properties $R_4$–$R_5$ from eq. (1), which are associated with the performance-related QoS requirements for the system. Figure 11 shows the result of the PRISM experiments carried out for the analysis of the CSL property $R_4$, with respect to different CPU allocations. The predicted value for the request arrival rate $\mu_3^1$ and the request service rate $\lambda_3^1$ from Table 2 were used for this experiment; the length of the request queue was fixed at $Q_{max} = 100$. The dashed line indicates that a CPU allocation equal to $cpu_3^1 = 0.439$ is the minimum value necessary to meet the requirement.

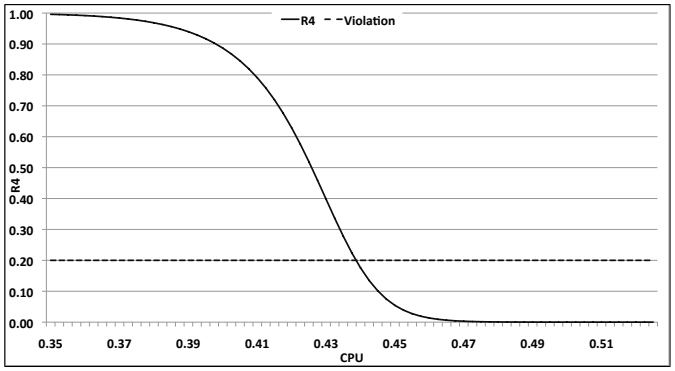



Fig. 11. Evaluation of $R_4$ for the TA system.

Finally, consider requirement $R_5$, again assuming an expected rate of incoming requests $\mu_3^1 = 1.2$ and a $Q_{max}$ value equal to 100. Figure 12 shows the evaluation of requirement $R_5$. In this case, the minimum fraction of CPU required to achieve the requirement is $cpu_3^1 = 0.467$.

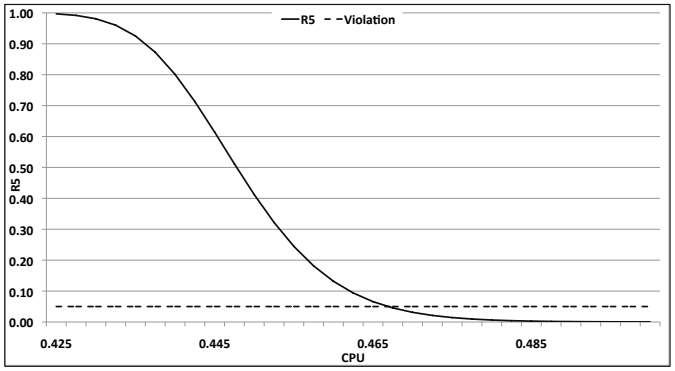



Fig. 12. Evaluation of $R_5$ for the TA system.

The generation of all possible configurations and the evaluation of PCTL and CSL properties associated with all QoS requirements are the outputs of the analysis stage, and the inputs for the planning stage of the MAPE loop.

**Planning Stage.** In this stage of the MAPE loop, the GPAC autonomic manager uses the results of the analysis stage to select a system configuration that maximises the $utility$ function from eq. (7). This involves parsing the output of the PRISM experiments described in the previous section into a dictionary data structure. Each entry in this dictionary maps a possible SBS configuration (the "key" for the entry) to the sequence of probabilities from requirements $R_0$ to $R_5$ that the configuration induces (the "value" for the entry). The value of the $utility$ function from (7) is then calculated for each of the dictionary entries, and an entry that maximises this function is selected as the next configuration for the TA system. The algorithm involved is described in detail in [23]. The initial configuration selected for the adaptive TA system (i.e., the one corresponding to the estimated

TABLE 6
Initial Configuration for the TA system

| Abstract Service | Index ($i$) | Mapping Pattern ($mp_i$) | Concrete service(s) | In-house service CPU ($cpu_i^x$) | Aggregate Failure Rate |
|---|---|---|---|---|---|
| Alarm Service | 1 | SGL | $s_1^2$ | – | 0.04 |
| Medical Analysis | 2 | SEQ | $\langle s_2^1, s_2^4 \rangle$ | – | 0.000015 |
| Drug Service | 3 | SGL | $s_3^1$ | $cpu_3^1 = 0.467$ | 0.0012 |

parameter values from Table 2) is shown in Table 6.

**Execution Stage.** If any of the mapping patterns $mp_i$ or the concrete-service sequences $\langle s_i^{j_1}, s_i^{j_2}, \ldots, s_i^{j_{S_i}} \rangle$, $1 \leq i \leq 3$, selected during the planning stage differ from those of the TA workflow executed by the BPEL workflow engine, the concrete workflow corresponding to the new optimal configuration is deployed and starts being used for the adaptive TA system. Similarly, whenever a new value was "planned" for the CPU resources $cpu_3^1$ allocated to the in-house service $DrugService1$, the configuration of the concrete service is adjusted accordingly. Finally, when the $SBS\_admin\_alarm$ configurable parameter from policy (9) was set to $true$, an alarm message is generated and sent to the administrator of the TA system. Depending on the particular realisation of the system, this could take the form of an email, a log entry, an SMS message or a combination thereof.

### 4.2 QoSMOS effectiveness

In this section, we look at how the QoSMOS MAPE loop adapts the configuration of the TA system to reflect the changes in its state and workload. As described so far, the configuration in Table 6 satisfies all the SBS requirements $R_0$ to $R_5$, and maximises the system $utility$ for the anticipated service failure rates and request arrival rates from Table 2. Indeed, in this setting we have $P_0 = 0.129 \leq 0.13$, $P_1 = 0.134 \leq 0.14$, $P_2 = 0.006 \leq 0.012$, $P_3 = 0.0048 \leq 0.005$, $P_4 = 0.0047 \leq 0.2$ and $P_5 = 0.049 \leq 0.05$. However, the characteristics of services evolve over time and can lead to requirement violations. For instance, Figure 13 shows how requirement $R_3$ is violated if the actual failure rate exhibited by $AlarmService2$ increased unexpectedly at run-time. The figure shows the evaluation of requirement $R_3$ with different failure rates of $AlarmService2$: a failure rate equal to $0.05$ instead of the predicted value of $0.04$ violates $R_3$. In fact, requirement $R_3$ is violated for any failure rate larger than or equal to $0.0417$.

Therefore, once the concrete workflow corresponding to the initial configuration is deployed, the KAMI component of QoSMOS collects run-time data concerning the number of successful and failed service invocations and the in-house request inter-arrival times. These data are used to re-compute the failure rates of all concrete services, and the arrival rates for the in-house concrete services. The new parameter values correspond to the *"a posteriori knowledge"* that the autonomic manager has with respect to the system, and are used to bring the operational model underlying the MAPE loop analysis in line with the actual state and workload of the TA system.

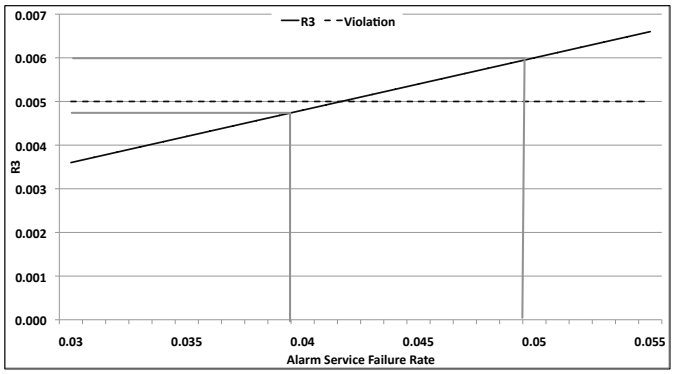



Fig. 13. Evaluation of $R_3$ with different failure rates of the Alarm Service.

Consider again the scenario in which the actual probability of incurring an $AlarmService2$ failure increases from the predicted value $r_1^2 = 0.04$ to $r_1^2 = 0.05$. In this scenario, KAMI considers the number of failures and the total number of invocations, and updates the failure rate associated to $AlarmService2$ by using the Bayesian estimator described in [40]. The "non-adapted behaviour" curve in Figure 14 depicts the result of simulating the behavior of $AlarmService2$ with a Bernoulli distribution with parameter $0.05$. This simulation considers the number of failed invocations collected by the monitoring process, and produces an estimate that is used to recompute the aggregate failure rate. The graph shows the average estimate for the aggregate failure rate of the alarm service depending on the number of run-time data representing invocations to the alarm service over 1000 simulations. The horizontal axis represents the run-time data for the invocations to the alarm service. The vertical axis represents the estimated value for the aggregate failure rate of the alarm service, which starts from the initial value (i.e., $r_2^1 = 0.04$) and gradually converges to the $r_2^1 = 0.05$.

Consider now the scenario in which the QoSMOS analysis is triggered after the $145th$ invocation to the alarm service. Since requirement $R_3$ is violated by any configuration that maps the abstract $AlarmService$ to the concrete service $AlarmService2$, the autonomic manager selects a system configuration in which $AlarmService1$—the least costly concrete alarm service with a failure rate under $0.0417$—is used as part of the TA workflow. Fig-

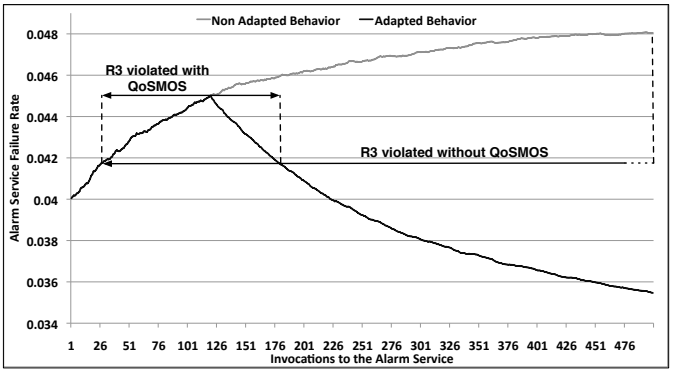



Fig. 14. Average estimate of the alarm service aggregate failure rate corresponding to transition $a$ in the DTMC operational model.

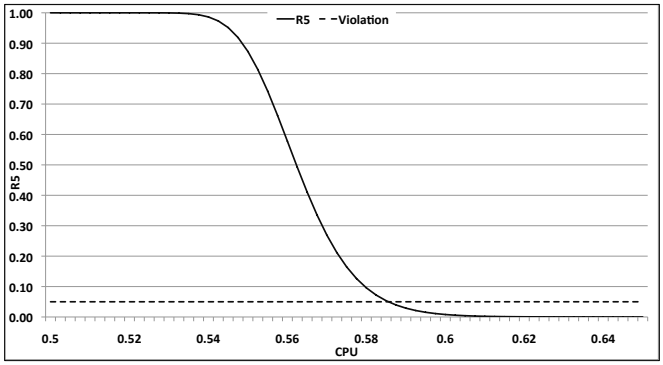



Fig. 15. Analysis of probability for requirement R5 after the request arrival rate increased to $\mu_3^1 = 1.3$.

ure 14 contrasts the system behavior in the presence of QoSMOS adaptation with its behavior in the absence of QoSMOS. In the former case, requirement $R_3$ is violated for a short period of time (i.e., for at most the period of the QoSMOS MAPE loop), whereas in the latter case the requirement is violated at all times when the failure rate of $AlarmService2$ reaches or exceeds $0.0417$.

Likewise, monitoring the in-house concrete services of an SBS can lead to changes in the operational model components underlying the analysis associated with the performance-related QoS requirements of the system. For instance, when the request arrival rate for in-house $DrugService1$ from the TA system changes from $\mu_3^1 = 1.2$ to $\mu_3^1 = 1.3$, the CTMC model in Figure 7 is updated accordingly, and the analysis stage of the MAPE loop performs PRISM experiments that assess the effect of this change. The result of the PRISM analysis for requirement $R5$ is shown in Figure 15. A probability of dropping requests less than $0.001$ is now obtained for $cpu_3^1 \geq 0.581$, so in the planning stage of the MAPE loop the value $cpu_3^1 = 0.581$ will be selected. In the absence of QoSMOS, the TA system violates requirement $R5$ for as long as the request arrival rate increases beyond the initial $\mu_3^1$ estimate from Table 2 (or resource over-provisioning is employed to ensure that the service can cope with the predicted peak demand).

This last analysis shows that, even if selected concrete services exhibit a run-time behavior coherent with data declared in their SLA, requirements can still be violated due to variations of other factors involved in the TA system. For example, a variation in the usage profile of the system can invalidate some of its requirements. Since our approach relies on operational models and on run-time estimation of model parameters, we are able to deal with all these scenario. This capability—unique to the QoSMOS framework—is possible due to the adoption of operational models which consider the overall architecture of the system and its domain.

Finally, note that this example described only one possible option for modeling a service-based system. Indeed, the design of models and the choice of parameters to be analyzed by the autonomic manager are tailored to the requirements of the application under design. For example, in other domains or in different systems, the SBS designer and administrator could be interested in considering configurations based on more complex or different parameters such as the thread pool size for multithreaded applications or the connection pool size for network intensive systems. The main advantage of the QoSMOS approach relies on the use of operational models and on probabilistic and stochastic logics that enable a broad range of applications.

### 4.3 QoSMOS scalability

The main overhead of using the QoSMOS approach to add adaptiveness to a service-based system corresponds to the execution of the PRISM experiments in the analysis stage of the QoSMOS MAPE loop. All other operations performed by the QoSMOS autonomic manager—including the monitoring of the system state and workload, updating the QoSMOS operational model, parsing the results of the PRISM experiments and using these results to plan and enforce a new system configuration—take a negligible fraction of the overall MAPE loop processing time.

For the QoSMOS-enabled TA system in our case study, each full PRISM evaluation of the PCTL and CSL properties associated with the QoS requirements $R_0$ to $R_5$ took between 2–3 milliseconds on a 2.4 GHz Intel Core 2 Duo server with 4 GB of DDR3 RAM at 1067 MHz. Given the number of possible configurations examined and the time spent in the communication steps between the QoSMOS components, the end-to-end execution of the MAPE loop and the adaptation of the SBS configuration to a new system state and workload can be completed in between $2.7$–$3.4$ seconds. Note that this time represents the time required to react to changes in the system objectives, state and/or workload; it does not represent system downtime. Furthermore, this overhead does not need to be accommodated by a production server running one of the SBS components such as the BPEL

workflow engine or one of the in-house concrete services. Instead, the GPAC autonomic manager employed by QoSMOS is itself a service-based system, and can therefore be executed on a separate, management server. In this way, retrofitting adaptiveness to an existing SBS system can be done without modifying the original system or adding overheads to the physical servers that are used to execute its components.

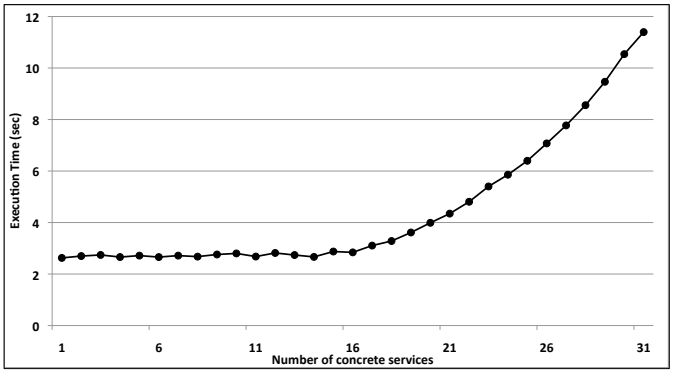
As these encouraging results were obtained for a service-based system comprising only three abstract services and nine associated concrete services, we carried out experiments to assess the scalability of the QoSMOS approach for service-based systems comprising larger numbers of services.

We first considered scenarios involving the original three-service abstract TeleAssistance workflow, and larger sets of concrete services. The increases in the MAPE loop execution time for two such scenarios are presented in Figure 16. Figure 16(a) shows the MAPE loop execution time required for gradually increasing sizes of the set of concrete services implementing the $AlarmService$ (i.e., $CS_1$). The size of the other concrete service sets (i.e., sets $CS_2$ and $CS_3$ implementing the $MedicalAnalysisService$ and the $DrugService$, respectively) were maintained at the values from Table 2. As expected, the MAPE loop execution time grows exponentially due to the background quantitative model checking from the analysis stage. However, the execution time does not exceed five seconds for $CS_1$ sizes of up to 22 concrete services, which is well over the number of concrete alarm services that can be expected for our case study.

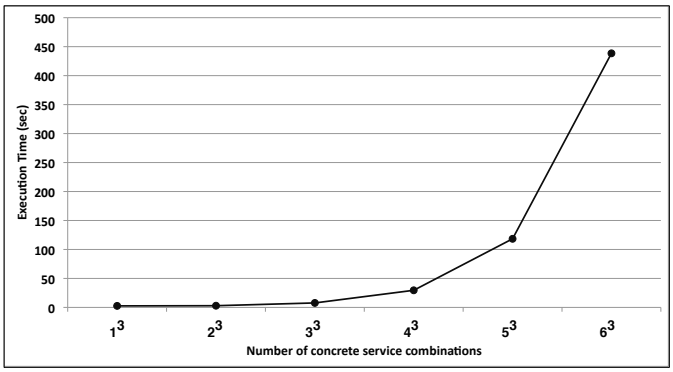
When the sizes of all concrete service sets were increased at the same time, the execution overheads in Figure 16(b) were observed for the QoSMOS MAPE loop. These results suggest that the QoSMOS approach can be used for systems of similar size to the TA SBS with sets of up to four concrete services for each abstract service (MAPE loop execution time under 30 seconds) or even up to five concrete services for each abstract service (MAPE loop execution time under 2 minutes). One way to accommodate larger sets of concrete services is to pre-select and use within the QoSMOS service-based system subsets of three to five concrete services that are most likely to be useful based on criteria such as cost or provider trustworthiness. This pre-selection can be done periodically, either by the SBS administrator or by another instance of the QoSMOS MAPE loop.

One last set of experiments that we present in this section involves examining the scalability of the QoSMOS framework for larger service-based systems. To perform these experiments, we increased the size of the abstract TA workflow by considering that the medical analysis part of the workflow requires the sequential execution of several services, each of which performs one part of the analysis.

In order to choose a realistic range of workflow sizes, we first carried out a study of the SBS development




(a) MAPE loop execution time for different $CS_1$ sizes.




(b) MAPE loop execution time for different values of $CS_1 = CS_2 = CS_3$

Fig. 16. QoSMoS scalability with the number of concrete services.

platform Taverna [57], [87]. Taverna is widely used in the development of scientific workflows in application domains including bioinformatics, chemo-informatics, astronomy and social sciences. Our study focused on the Taverna workflows with the tag 'bioinformatics' and with a download count of 50 or more from the Taverna workflow repository myExperiment [85]. We selected this particular set of workflows because it represents the most used set of workflows from an application domain in which the Taverna platform is used regularly. Out of the 28 workflows in this set, 13 comprise five services or less; seven comprise betwen six and eight services; five comprise ten services; two have 11 services; and one consists of 32 services. We therefore focused our experiments on extensions of the TA abstract workflow of similar size to these Taverna workflows.

Figure 17 shows the execution time for the QoSMOS MAPE loop for TA workflow variants comprising up to 13 additional abstract medical services (i.e., up to 16 abstract services in total). In all experiments, we considered that the sets of concrete services for all but the first three abstract services contained a single concrete service, i.e., adaptation was applied only to the original abstract services. The experiments were run for three adaptation scenarios, namely when sets of two, three and four concrete services, respectively, were available

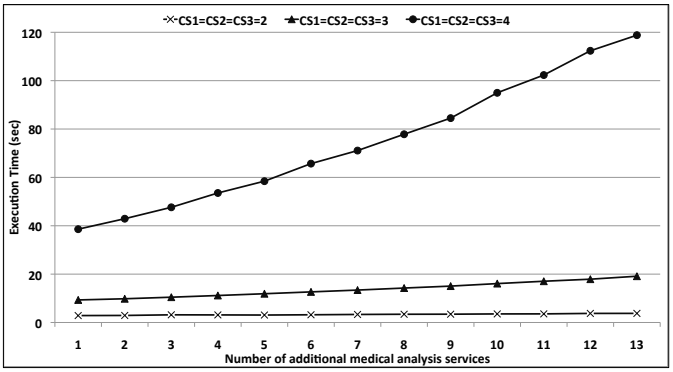



Fig. 17. QoSMoS scalability with the number of additional abstract medical services.

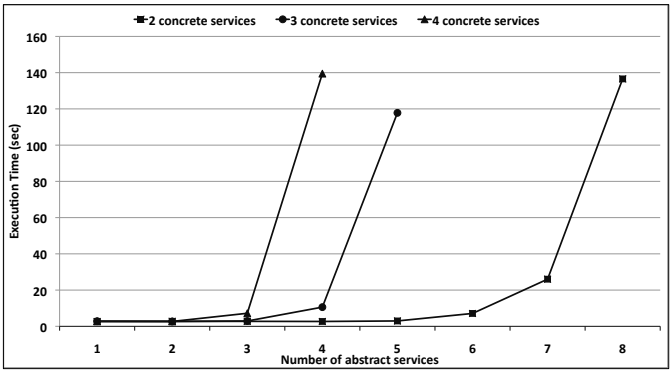



Fig. 18. QoSMoS scalability with the number of abstract and concrete services.

for each abstract service for which QoSMOS adaptation was employed. The results indicate that QoSMOS-based adaptation can be applied to workflows comprising up to 16 abstract services with overheads of under four seconds in the first scenario, under 20 seconds in the second scenario, and up to two minutes in the last scenario.

When more than one concrete service is available for every abstract service within the QoSMOS-based workflow (Figure 18), the workflow sizes for which the QoSMOS MAPE loop completes within 140 seconds are: eight—when $\#CS_i = 2$ for each concrete service set $CS_i$; five—when $\#CS_i = 3$ for all $i$; and four—when $\#CS_i = 4$ for all $i$. Note that these experiments cover over 71% of the Taverna workflows from the study described above (i.e., 20 out of 28 workflows), which we consider a good result for the QoSMOS prototype realisation and this scenario in which the adaptation is applied to every single component of the service-based system. Furthermore, remember that the SBS objectives in our case study consist of no less than six QoS requirements, each of which brings an almost equal contribution to the execution times obtained for the experiments in this section. Adaptation in service-based systems with less complex SLAs can be achieved with significantly lower overheads.

There are several options that we are investigating in our effort to increase the scalability of QoSMOS, including the development of incremental quantitative analysis techniques that build the results of a PRISM experiment from the results generated in the previous analysis stage, the use of intelligent caching and pre-evaluation techniques to bypass most of the analysis stage instances altogether; and the use of a hybrid approach in which a less demanding PRISM experiment is carried out to produce a close-to-optimal configuration and a fast heuristic is then used to refine this configuration. Additionally, a straightforward approach that can bring an immediate, multifold reduction in the quantitative analysis overheads is to run the PRISM experiments for different requirements in parallel, either on a multicore-processor server or on multiple machines.

## 5 Conclusions and Future Work

In this paper we have presented QoSMOS, a tool-supported framework for QoS management of self-adaptive service-based systems. QoSMOS defines and implements an autonomic architecture that combines formal specification of QoS requirements, model-based QoS evaluation, monitoring and parameter adaptation of the QoS models, and planning and execution of system adaptation. The proposed framework has been built through the integration of extended versions of existing tools and components developed by the authors.

Essential strengths of QoSMOS are the use of a precise and formal specification of QoS requirements with probabilistic temporal logics and the definition of a model-based quality evaluation methodology for probabilistic QoS attributes taking into account quality dependencies on other services and on the operational profile. The monitoring phase of QoSMOS and the consequent possible on-line update of the quality models allow discovering requirements violations and triggering adaptation strategies for the SBS. The possible strategies are based on techniques for service selection, run-time reconfiguration and resource assignment to in-house managed services. Furthermore, the quality models in QoSMOS represent the overall system architecture, so it is possible to detect requirement violations generated by different causes and not only related to unexpected behaviors associated to single services of the SBS (e.g., unexpected variations in the usage profile). The validation of the proposed framework has been performed through the application of QoSMOS capabilities to a common case study of a service-based system for remote medical assistance. The results obtained with a high number of numerical experiments and simulations proved the effectiveness of our solution.

On the other hand, we have to also acknowledge some limitations that should be considered when selecting the QoSMOS framework. One limitation of the QoSMOS framework is that, due to the statistical methods behind the monitoring and QoS analysis, it is hard to deal with

models that contain extreme probabilities. As an example, with a Bayesian filter it would require an unfeasible large number of observations to change the value of a transition probability to eg. $10^{-9}h^{-1}$. Additionally, we acknowledge that the quality evaluation with our more realistic model-based QoS models and probabilistic verification can take longer than the quality evaluation with simple aggregation functions. Consequently, there is a trade-off between the improved accuracy of our QoS evaluation compared to the existing approaches and the time needed to obtain these results. For most practical service-based systems where QoSMOS was applied the time efficiency was not a problem. However, when dealing with a workflow with several thousand services and multiple parameters, a very long time could be necessary to get a result of the quality evaluation. Furthermore our approach currently only applies to probabilistically quantifiable and externally observable QoS properties, such as reliability, availability and performance. Due to the underlying techniques for the adaptation and planning procedures, an application to qualitative non-quantifiable QoS properties is currently not possible.

Besides working on the above mentioned limitations our future work will consist in refining the QoSMOS approach by investigating its range of applicability. We plan to enrich the ongoing implementation by: enlarging the set of supported models (e.g., Markov Decision Processes, etc.), integrating black-box monitoring techniques [51], and defining a language aimed at managing multi-model consistency. Additionally, it would be interesting to explore and extend other QoS specification formalisms (such as, for example, ALBERT [10] or probabilistic and timed MSCs [58], [90]) and map them into the ProProST pattern system and the provided structured English grammar.

## ACKNOWLEDGMENT

This work was partly supported by the FP7 European projects CONNECT (FP7 231167) and Q-ImPrESS (FP7 215013), and by the UK Engineering and Physical Sciences Research Council grants EP/F001096/1 and EP/H042644/1.

## REFERENCES

[1] R. Alur, C. Courcoubetis, and D. Dill, "Model-checking in dense real-time," *Information and Computation*, vol. 104, no. 1, pp. 2–34, 1993.

[2] R. Alur and T. A. Henzinger, "Reactive modules," in *Formal Methods in System Design*. IEEE Computer Society Press, 1999, pp. 207–218.

[3] D. Ardagna, C. Ghezzi, and R. Mirandola, "Rethinking the use of models in software architecture," in *4th International Conference on the Quality of Software-Architectures, QoSA 2008*, ser. LNCS, S. Becker, F. Plasil, and R. Reussner, Eds., vol. 5281. Springer, 2008, pp. 1–27.

[4] D. Ardagna and B. Pernici, "Global and local QoS constraints guarantee in web service selection," in *Proc. of the IEEE International Conference on Web Services (ICWS 2005)*. IEEE Computer Society, 2005, pp. 805–806.

[5] D. Ardagna and B. Pernici, "Adaptive service composition in flexible processes," *IEEE Trans. Softw. Eng.*, vol. 33, no. 6, pp. 369–384, 2007.

[6] A. Aziz, V. Singhal, and F. Balarin, "It usually works: The temporal logic of stochastic systems," in *Proc. 7th International Conference on Computer Aided Verification, CAV 95*, ser. LNCS, P. Wolper, Ed., vol. 939. Springer, 1995, pp. 155–165.

[7] E. Badidi, L. Esmahi, and M. A. Serhani, "A queuing model for service selection of multi-classes QoS-aware web services," in *Proccedings of the IEEE International Conference on Web Services (ICWS 2005)*. IEEE Computer Society, 2005, pp. 204–213.

[8] C. Baier, J.-P. Katoen, and H. Hermanns, "Approximate symbolic model checking of continuous-time markov chains," in *Proc. 10th International Conference on Concurrency Theory, CONCUR 99*, ser. LNCS, J. C. M. Baeten and S. Mauw, Eds., vol. 1664. Springer, 1999, pp. 146–161.

[9] L. Baresi, D. Bianculli, C. Ghezzi, S. Guinea, and P. Spoletini, "A timed extension of WSCoL," in *IEEE International Conference on Web Services*, 2007, pp. 663–670.

[10] L. Baresi, D. Bianculli, C. Ghezzi, S. Guinea, and P. Spoletini, "Validation of web service compositions," *IET Software*, vol. 1, no. 6, pp. 219–232, December 2007.

[11] L. Baresi and S. Guinea, "Towards dynamic monitoring of WS-BPEL processes," *Proceedings of the 3rd International Conference on Service Oriented Computing*, 2005.

[12] L. Baresi, E. D. Nitto, and C. Ghezzi, "Toward open-world software: Issue and challenges," *IEEE Computer*, vol. 39, no. 10, pp. 36–43, 2006.

[13] R. Berbner, M. Spahn, N. Repp, O. Heckmann, and R. Steinmetz, "Heuristics for qoS-aware web service composition," in *Proccedings of the IEEE International Conference on Web Services (ICWS 2006)*. IEEE Computer Society, 2006, pp. 72–82.

[14] J. O. Berger, *Statistical Decision Theory and Bayesian Analysis*, 2nd ed. Springer, 1985.

[15] C. Bettini, D. Maggiorini, and D. Riboni, "Distributed context monitoring for the adaptation of continuous services," *World Wide Web*, vol. 10, no. 4, pp. 503–528, 2007.

[16] A. Bianco and L. de Alfaro, "Model checking of probabilistic and nondeterministic systems," in *Proc. of the 15th Conference on Foundations of Software Technology and Theoretical Comp. Science, FSTTCS 95*, ser. LNCS, P. S. Thiagarajan, Ed., vol. 1026. Springer, 1995, pp. 499–513.

[17] D. Bianculli and C. Ghezzi, "Monitoring conversational web services," in *Proceedings of the 2nd Intern. Workshop on Service Oriented Software Engineering, (IW-SOSWE)*. ACM, 2007, pp. 15–21.

[18] G. Bolch, S. Greiner, H. de Meer, and K. S. Trivedi, *Queuing Network and Markov Chains*. John Wiley, 1998.

[19] B. Boonea, S. Van Hoeckea, G. Van Seghbroecka, N. Jonckheereb, V. Jonckersb, F. D. Turcka, C. Develdera, and B. Dhoedta, "SALSA: QoS-aware load balancing for autonomous service brokering," *Journal of Systems and Software*, vol. available online, p. in print, 2010.

[20] R. Calinescu, "General-purpose autonomic computing," in *Autonomic Computing and Networking*, M. K. Denko, L. T. Yang, and Y. Zhang, Eds. Springer, 2009, pp. 3–30.

[21] R. Calinescu, "Reconfigurable service-oriented architecture for autonomic computing," *International Journal On Advances in Intelligent Systems*, vol. 2, no. 1, pp. 38–57, June 2009.

[22] R. Calinescu and M. Kwiatkowska, "CADS*: Computer-aided development of self-* systems," in *Fundamental Approaches to Software Engineering (FASE 2009)*, ser. LNCS, M. Chechik and M. Wirsing, Eds., vol. 5503. Springer, March 2009, pp. 421–424.

[23] R. Calinescu and M. Z. Kwiatkowska, "Using quantitative analysis to implement autonomic IT systems," in *Proceedings of the 31st International Conference on Software Engineering, ICSE 2009*. IEEE Computer Society, 2009, pp. 100–110.

[24] G. Canfora, M. Di Penta, R. Esposito, and M. L. Villani, "An approach for QoS-aware service composition based on genetic algorithms," in *GECCO '05: Proceedings of the 2005 conference on Genetic and evolutionary computation*. New York, NY, USA: ACM, 2005, pp. 1069–1075.

[25] G. Canfora, M. D. Penta, R. Esposito, and M. L. Villani, "Qos-aware replanning of composite web services," in *Proccedings of the IEEE International Conference on Web Services (ICWS 2005)*. IEEE Computer Society, 2005, pp. 121–129.

[26] G. Canfora, M. D. Penta, R. Esposito, and M. L. Villani, "A framework for QoS-aware binding and re-binding of composite web services," *Journal of Systems and Software*, vol. 81, no. 10, pp. 1754–1769, 2008.

[27] L. Cao, J. Cao, and M. Li, "Genetic algorithm utilized in cost-reduction driven web service selection," in *Proceedings of the International Conference on Computational Intelligence and Security, CIS 2005*, ser. LNCS, Y. Hao, J. Liu, Y. Wang, Y. ming Cheung, H. Yin, L. Jiao, J. Ma, and Y.-C. Jiao, Eds., vol. 3802. Springer, 2005, pp. 679–686.

[28] V. Cardellini, E. Casalicchio, V. Grassi, F. Lo Presti, and R. Mirandola, "Qos-driven runtime adaptation of service oriented architectures," in *ESEC/FSE 2009, Proceedings*. ACM, 2009, pp. 131–140.

[29] V. Cardellini, E. Casalicchio, V. Grassi, and R. Mirandola, "A framework for optimal service selection in broker-based architectures with multiple QoS classes," in *Services computing workshops, SCW 2006*. IEEE computer society, 2006, pp. 105–112.

[30] V. Cardellini, E. Casalicchio, V. Grassi, and F. L. Presti, "Scalable service selection for web service composition supporting differentiated qos classes," Dip. di Informatica, Sistemi e Produzione, Universita di Roma Tor Vergata, Tech. Rep. Technical Report RR-07.59, 2007.

[31] G. Chafle, P. Doshi, J. Harney, S. Mittal, and B. Srivastava, "Improved adaptation of web service compositions using value of changed information," in *ICWS*. IEEE Computer Society, 2007, pp. 784–791.

[32] B. H. C. Cheng, H. Giese, P. Inverardi, J. Magee, and R. de Lemos, "08031 – software engineering for self-adaptive systems: A research road map," in *Software Engineering for Self-Adaptive Systems*, ser. Dagstuhl Seminar Proceedings, vol. 08031. IBFI, Schloss Dagstuhl, Germany, 2008.

[33] F. Ciesinski and M. Größer, "On probabilistic computation tree logic," in *Validation of Stochastic Systems - A Guide to Current Research*, ser. LNCS, C. Baier, B. R. Haverkort, H. Hermanns, J.-P. Katoen, and M. Siegle, Eds., vol. 2925. Springer, 2004, pp. 147–188.

[34] A. Clark, S. Gilmore, J. Hillston, and M. Tribastone, "Stochastic process algebras," in *7th Intern. School on Formal Methods, SFM*, ser. LNCS, vol. 4486. Springer, 2007, pp. 132–179.

[35] M. Colombo, E. Di Nitto, and M. Mauri, "Scene: A service composition execution environment supporting dynamic changes disciplined through rules," *LNCS*, vol. 4294, p. 191, 2006.

[36] R. Colvin, L. Grunske, and K. Winter, "Probabilistic timed behavior trees," in *Proceedings of the International Conference on Integrated Formal Methods (IFM 2007)*, ser. LNCS, J. Davies and J. Gibbons, Eds., vol. 4591. Springer-Verlag, 2007, pp. 156–175.

[37] R. Colvin, L. Grunske, and K. Winter, "Timed behavior trees for failure mode and effects analysis of time-critical systems," *Journal of Systems and Software*, vol. 81, no. 12, pp. 2163–2182, 2008.

[38] A. Dan, D. Davis, R. Kearney, A. Keller, R. King, D. Kuebler, H. Ludwig, M. Polan, M. Spreitzer, and A. Youssef, "Web services on demand: WSLA-driven automated management," *IBM Systems Journal*, vol. 43, no. 1, pp. 136–158, 2004.

[39] M. B. Dwyer, G. S. Avrunin, and J. C. Corbett, "Property specification patterns for finite-state verification," in *Proc. 21th International Conference on Software Engineering (ICSE99)*. ACM Press, 1999, pp. 411–420.

[40] I. Epifani, C. Ghezzi, R. Mirandola, and G. Tamburrelli, "Model evolution by run-time parameter adaptation," in *Proc. 31st International Conference on Software Engineering (ICSE09)*. Los Alamitos, CA, USA: IEEE Computer Society, 2009, pp. 111–121.

[41] R. Frei, G. D. M. Serugendo, and J. Barata, "Designing self-organization for evolvable assembly systems," in *Second IEEE Intern. Conf. on Self-Adaptive and Self-Organizing Systems, SASO 2008*. IEEE Computer Society, 2008, pp. 97–106.

[42] S. Frolund and J. Koistinen, "Quality-of-service specification in distributed object systems," *Distributed Systems Engineering Journal*, vol. 5, no. 4, pp. 179–202, Dec. 1998.

[43] S. Gallotti, C. Ghezzi, R. Mirandola, and G. Tamburrelli, "Quality prediction of service compositions through probabilistic model checking," in *Proc. 4th International Conference on the Quality of Software-Architectures, QoSA 2008*, ser. LNCS, S. Becker, F. Plasil, and R. Reussner, Eds., vol. 5281. Springer, 2008, pp. 119–134.

[44] R. Garcia, J. Jarvi, A. Lumsdaine, J. Siek, and J. Willcock, "A comparative study of language support for generic programming," *ACM SIGPLAN Notices*, vol. 38, no. 11, pp. 115–134, November 2003.

[45] C. Ghezzi and G. Tamburrelli, "Predicting performance properties for open systems with KAMI," in *QoSA '09: Proceedings of the 5th International Conference on the Quality of Software Architectures*. Berlin, Heidelberg: Springer-Verlag, 2009, pp. 70–85.

[46] S. Gilmore and J. Hillston, "The PEPA workbench: A tool to support a process algebra-based approach to performance modelling," in *Proceedings of the 7th International Conference on Computer Performance Evaluation, Modeling Techniques and Tools*, ser. LNCS, G. Haring and G. Kotsis, Eds., vol. 794. Springer, 1994, pp. 353–368.

[47] V. Grassi, "Architecture-based reliability prediction for service-oriented computing," in *Workshop on Architecting Dependable Systems, WADS*, ser. LNCS, vol. 3549. Springer, 2004, pp. 279–299.

[48] V. Gruhn and R. Laue, "Patterns for timed property specifications," *Electr. Notes Theor. Comput. Sci*, vol. 153, no. 2, pp. 117–133, 2006.

[49] L. Grunske, "Specification patterns for probabilistic quality properties," in *30th International Conference on Software Engineering (ICSE 2008)*, Robby, Ed. ACM, 2008, pp. 31–40.

[50] L. Grunske, K. Winter, and R. Colvin, "Timed behavior trees and their application to verifying real-time systems," in *Australian Software Engineering Conference*. IEEE Computer Society, 2007, pp. 211–222.

[51] L. Grunske and P. Zhang, "Monitoring probabilistic properties," in *Proceedings of the 7th joint meeting of the European Software Engineering Conference and the ACM SIGSOFT International Symposium on Foundations of Software Engineering, 2009*, H. van Vliet and V. Issarny, Eds. ACM, 2009, pp. 183–192.

[52] H. Guo, J. Huai, H. Li, T. Deng, Y. Li, and Z. Du, "ANGEL: Optimal Configuration for High Available Service Composition," in *IEEE International Conference on Web Services (ICWS 2007)*. IEEE Computer Society, 2007, pp. 280–287.

[53] H. Hansson and B. Jonsson, "A logic for reasoning about time and reliability," *Formal Aspects of Computing*, vol. 6, no. 5, pp. 512–535, 1994.

[54] D. Harel and R. Marelly, "Playing with time: On the specification and execution of time-enriched LSCs," in *10th International Workshop on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS 2002)*. IEEE Computer Society, 2002, pp. 193–202.

[55] J. Harney and P. Doshi, "Speeding up adaptation of web service compositions using expiration times," in *World Wide Web (WWW)*. ACM, 2007, pp. 1023–1032.

[56] M. C. Huebscher and J. A. McCann, "A survey of autonomic computing—degrees, models, and applications," *ACM Comput. Surv.*, vol. 40, no. 3, pp. 1–28, 2008.

[57] D. Hull, K. Wolstencroft, R. Stevens, C. Goble, M. Pocock, P. Li, and T. Oinn, "Taverna: a tool for building and running workflows of services." *Nucleic Acids Research*, vol. 34, no. Web Server issue, pp. 729–732, July 2006.

[58] ITU-TS, "ITU-TS recommendation Z.120: Message sequence chart 1999 (MSC99)," ITU-TS, Geneva, Tech. Rep., 1999.

[59] D. N. Jansen, H. Hermanns, and J.-P. Katoen, "A probabilistic extension of UML statecharts," in *Proceedings of 7th International Symposium on Formal Techniques in Real-Time and Fault-Tolerant Systems, FTRTFT 2002*, ser. LNCS, W. Damm and E.-R. Olderog, Eds., vol. 2469. Springer, 2002, pp. 355–374.

[60] D. N. Jansen, H. Hermanns, and J.-P. Katoen, "A QoS-oriented extension of UML statecharts," in *Proceedings of 6th International Conference - The Unified Modeling Language. Model Languages and Applications. UML 2003*, ser. LNCS, P. Stevens, J. Whittle, and G. Booch, Eds., vol. 2863. Springer, 2003, pp. 76–91.

[61] D. N. Jansen, J.-P. Katoen, M. Oldenkamp, M. Stoelinga, and I. Zapreev, "How fast and fat is your probabilistic model checker? An experimental comparison," in *Hardware and Software: Verification and Testing*, ser. LNCS, K. Yorav, Ed., vol. 4899. Springer, 2008, pp. 69–85.

[62] M. B. Juric, B. Mathew, and P. Sarang, *Business Process Execution Language for Web Services*. Packt Publishing, 2004.

[63] D. Karastoyanova and F. Leymann, "BPEL'n'Aspects: Adapting Service Orchestration Logic," in *Proceedings of the 2009 IEEE International Conference on Web Services*. IEEE Computer Society, 2009, pp. 222–229.

[64] J.-P. Katoen, T. Kemna, I. S. Zapreev, and D. N. Jansen, "Bisimulation minimisation mostly speeds up probabilistic model checking," in *Tools and Algorithms for the Construction and Analysis of Systems TACAS 2007, Proceedings*, ser. LNCS, O. Grumberg and M. Huth, Eds., vol. 4424. Springer, 2007, pp. 87–101.

[65] A. Keller and H. Ludwig, "The WSLA framework: Specifying and monitoring service level agreements for web services," *Journal of Network and Systems Management*, vol. 11, no. 1, 2003.

[66] J. O. Kephart and D. M. Chess, "The vision of autonomic computing," *IEEE Computer Journal*, vol. 36, no. 1, pp. 41–50, January 2003.

[67] J. M. Ko, C. O. Kim, and I.-H. Kwon, "Quality-of-service oriented web service composition algorithm and planning architecture," *Journal of Systems and Software*, vol. 81, no. 11, pp. 2079–2090, 2008.

[68] S. Konrad and B. H. C. Cheng, "Real-time specification patterns," in *27th International Conference on Software Engineering (ICSE 05)*, G.-C. Roman, W. G. Griswold, and B. Nuseibeh, Eds. ACM Press, 2005, pp. 372–381.

[69] R. Koymans, "Specifying real-time properties with metric temporal logic," *Real-Time Systems*, vol. 2, no. 4, pp. 255–299, 1990.

[70] J. Kramer and J. Magee, "Self-managed systems: an architectural challenge," in *Future of Software Engineering*, 2007, pp. 259–268.

[71] M. Z. Kwiatkowska, G. Norman, D. Parker, and J. Sproston, "Performance analysis of probabilistic timed automata using digital clocks," *Formal Methods in System Design*, vol. 29, no. 1, pp. 33–78, 2006.

[72] M. Z. Kwiatkowska, G. Norman, and D. Parker, "Probabilistic symbolic model checking with PRISM: A hybrid approach," *Int. Journal on Software Tools for Technology Transfer(STTT)*, vol. 6, no. 2, pp. 128–142, Aug. 2004.

[73] M. Z. Kwiatkowska, G. Norman, and D. Parker, "Symmetry reduction for probabilistic model checking," in *Computer Aided Verification, 18th International Conference, CAV 2006, Proceedings*, ser. LNCS, T. Ball and R. B. Jones, Eds., vol. 4144. Springer, 2006, pp. 234–248.

[74] M. Z. Kwiatkowska, G. Norman, J. Sproston, and F. Wang, "Symbolic model checking for probabilistic timed automata," *Inf. Comput*, vol. 205, no. 7, pp. 1027–1077, 2007.

[75] D. D. Lamanna, J. Skene, and W. Emmerich, "Slang: A language for defining service level agreements," in *Proceedings of the 9th IEEE International Workshop on Future Trends of Distributed Computing Systems (FTDCS 2003)*. IEEE Computer Society, 2003, pp. 100–107.

[76] E. D. Lazowska, J. Zahorjan, G. S. Graham, and K. C. Sevcik, *Quantitative System Performance: Computer System Analysis Using Queueig Network Models*. Prentice-Hall, 1984.

[77] Y. Ma and C. Zhang, "Quick convergence of genetic algorithm for QoS-driven web service selection," *Computer Networks*, vol. 52, no. 5, pp. 1093–1104, 2008.

[78] M. A. Marsan, "Stochastic petri nets: an elementary introduction," in *Advances in Petri Nets*. Berlin - Heidelberg - New York: Springer, June 1989, pp. 1–29.

[79] M. Marzolla and R. Mirandola, "Performance prediction of web service workflows," in *International Conference on Quality of Software Architectures, QoSA 2007*, ser. LNCS, vol. 4880. Springer, 2007, pp. 127–144.

[80] D. A. Menascé, E. Casalicchio, and V. Dubey, "A heuristic approach to optimal service selection in service oriented architectures," in *Proceedings of the 7th International Workshop on Software and Performance, WOSP 2008*, A. Avritzer, E. J. Weyuker, and C. M. Woodside, Eds. ACM, 2008, pp. 13–24.

[81] D. A. Menascé and V. Dubey, "Utility-based qos brokering in service oriented architectures," in *Proceedings of the International Conference on Web Services ICWS 2007*, 2007, pp. 422–430.

[82] D. A. Menascé, J. M. Ewing, H. Gomaa, S. Malek, and J. P. Sousa, "A framework for utility-based service oriented design in SASSY," in *1st joint WOSP/SIPEW international conference on Performance engineering (WOSP/SIPEW'10)*. ACM, 2010, pp. 27–36.

[83] D. A. Menascé, H. Ruan, and H. Gomaa, "Qos management in service-oriented architectures," *Perform. Eval.*, vol. 64, no. 7-8, pp. 646–663, 2007.

[84] O. Moser, F. Rosenberg, and S. Dustdar, "Non-intrusive monitoring and service adaptation for WS-BPEL," in *World Wide Web (WWW 2008)*. ACM, 2008, pp. 815–824.

[85] "myExperiment web site," www.myexperiment.org (repository snapshot on 20 April 2010).

[86] E. D. Nitto, C. Ghezzi, A. Metzger, M. P. Papazoglou, and K. Pohl, "A journey to highly dynamic, self-adaptive service-based applications," *Autom. Softw. Eng.*, vol. 15, no. 3-4, pp. 313–341, 2008.

[87] T. Oinn, M. Greenwood, M. Addis, N. Alpdemir, J. Ferris, K. Glover, C. Goble, A. Goderis, D. Hull, D. Marvin, P. Li, P. Lord, M. Pocock, M. Senger, R. Stevens, A. Wipat, and C. Wroe, "Taverna: lessons in creating a workflow environment for the life sciences," *Concurrency and Computation: Practice and Experience*, vol. 18, no. 10, pp. 1067–1100, August 2006.

[88] M. P. Papazoglou and W.-J. Heuvel, "Service oriented architectures: approaches, technologies and research issues," *The VLDB Journal*, vol. 16, no. 3, pp. 389–415, 2007.

[89] M. L. Puterman, *Markov Decision Processes*. Wiley, 1994.

[90] G. N. Rodrigues, D. S. Rosenblum, and S. Uchitel, "Using scenarios to predict the reliability of concurrent component-based software systems," in *Fundamental Approaches to Software Engineering, 8th Intern. Conference, FASE 2005, Proceedings*, ser. LNCS, M. Cerioli, Ed., vol. 3442. Springer, 2005, pp. 111–126.

[91] F. Saffre, R. Tateson, J. Halloy, M. Shackleton, and J.-L. Deneubourg, "Aggregation dynamics in overlay networks and their implications for self-organized distributed applications," *The Computer Journal*, 2008.

[92] A. Sahai, V. Machiraju, M. Sayal, A. P. A. van Moorsel, and F. Casati, "Automated SLA monitoring for web services," in *Proceedings of the 13th IFIP/IEEE International Workshop on Distributed Systems: Operations and Management, DSOM 2002*, ser. LNCS, M. Feridun, P. G. Kropf, and G. Babin, Eds., vol. 2506. Springer, 2002, pp. 28–41.

[93] N. Sato and K. S. Trivedi, "Stochastic modeling of composite web services for closed-form analysis of their performance and reliability bottlenecks," in *ICSOC*, ser. LNCS, vol. 4749. Springer, 2007, pp. 107–118.

[94] M. A. Serhani, R. Dssouli, A. Hafid, and H. A. Sahraoui, "A qoS broker based architecture for efficient web services selection," in *Proccedings of the IEEE International Conference on Web Services (ICWS 2005)*. IEEE Computer Society, 2005, pp. 113–120.

[95] J. M. Sobel and D. P. Friedman, "An introduction to reflection-oriented programming," in *Proc. of Reflection96*, April 1996.

[96] S. Su, C. Zhang, and J. Chen, "An improved genetic algorithm for web services selection," in *Proceedings of the 7th IFIP WG 6.1 International Conference on Distributed Applications and Interoperable Systems, DAIS 2007*, ser. LNCS, J. Indulska and K. Raymond, Eds., vol. 4531. Springer, 2007, pp. 284–295.

[97] T. Suto, J. T. Bradley, and W. J. Knottenbelt, "Performance trees: A new approach to quantitative performance specification," in *14th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS 2006)*. IEEE Computer Society, 2006, pp. 303–313.

[98] V. Tosic, B. Pagurek, and K. Patel, "WSOL - A language for the formal specification of classes of service for web services," in *Proceedings of the International Conference on Web Services, ICWS 2003*, L.-J. Zhang, Ed. CSREA Press, 2003, pp. 375–381.

[99] L. Wang, N. J. Dingle, and W. J. Knottenbelt, "Natural language specification of performance trees," in *Proceedings of the 5th European Performance Engineering Workshop, EPEW 2008*, ser. LNCS, N. Thomas and C. Juiz, Eds., vol. 5261, 2008, pp. 141–151.

[100] H. L. S. Younes and R. G. Simmons, "Statistical probabilistic model checking with a focus on time-bounded properties," *Inf. Comput.*, vol. 204, no. 9, pp. 1368–1409, 2006.

[101] L. Zeng, B. Benatallah, A. H. H. Ngu, M. Dumas, J. Kalagnanam, and H. Chang, "QoS-aware middleware for web services composition," *IEEE Trans. Software Eng*, vol. 30, no. 5, pp. 311–327, 2004.

[102] C. Zhang, S. Su, and J. Chen, "DiGA: Population diversity handling genetic algorithm for qoS-aware web services selection," *Computer Communications*, vol. 30, no. 5, pp. 1082–1090, 2007.

**Radu Calinescu** is a Lecturer in Computing at Aston University, UK. Prior to this, he was a part-time Lecturer on the Software Engineering Programme and a Senior Researcher on the Formal Verification research theme at the University of Oxford. He holds a DPhil in Computation from the University of Oxford, and was awarded a British Computer Society Distinguished Dissertation Award. He has over ten years of academic and industrial research experience in developing complex software systems in areas including adaptive systems, model-driven architectures and information systems for cancer research. He has chaired or has been on the program committees of multiple international conferences on autonomic, adaptive and complex systems. He is a Senior Member of the IEEE and a member of the Editorial Advisory Board for the Journal on Advances in Intelligent Systems.

**Lars Grunske** is currently lecturer at the Swinburne University of Technology, Australia. He received his PhD degree in computer science from the University of Potsdam, Germany, (Hasso-Plattner-Institute for Software Systems Engineering), in 2004 and he was Boeing Post-doctoral Research Fellow at the University of Queensland, Australia, from 2004-2007. He has active research interests in the areas of modelling and verification of systems and software. His main focus is on automated analysis, mainly probabilistic and timed model checking and model-based dependability evaluation of complex software intensive systems.

**Marta Kwiatkowska** is Professor of Computing Systems and Fellow of Trinity College, University of Oxford. Her research is concerned with modelling and automated verification methods for complex systems, and particularly probabilistic and/or real-time systems. The PRISM model checker (www.prismmodelchecker.org) developed in her group is the leading software tool in the area and is widely used for research and teaching. Applications of probabilistic model checking have spanned communication and security protocols, nanotechnology designs, power management and systems biology. Kwiatkowska's research is currently supported by £4.3m of grant funding, including the recently awarded ERC Advanced Grant VERIWARE "From software verification to everyware verification".

Marta Kwiatkowska was invited speaker at LICS, ESEC/FSE and ETAPS/FASE. She serves on editorial boards of IEE TSE, Science of Computer Programming and Royal Society's Philosophical Transactions A. She is a member of the Steering Committee of the International Conference on Quantitative Evaluation of SysTems (QEST) and lead organiser of the Royal Society Discussion Meeting "From computers to ubiquitous computing, by 2020".

**Raffaela Mirandola** is an Assistant Professor at Dipartimento di Elettronica e Informazione at Politecnico di Milano. Raffaela's research interests are in the areas of performance and reliability modeling and analysis of software/hardware systems with special emphasis on the automatic generation of performance models for component-based systems and service-based systems, model driven engineering applied to embedded systems, empirical software engineering and software process analysis. She has published over 70 journal and conference articles on these topics. She served and is currently serving on the program committees of conferences in these research areas, and she is a member of the Editorial Board of the Journal of System and Software.

**Giordano Tamburrelli** is currently a PhD student at the Politecnico di Milano, Italy. He received an MSc degree in Computer Science from the University of Illinois at Chicago, US, and an MSc degree in Computer Science Engineering from the Politecnico di Milano, in a joint degree program. He has active research interests in the areas of run-time modeling and verification of systems and software. His main focus is on automated model-based analysis, and mainly on non-functional requirements of complex software systems and service-based architectures.