**Dynamic QoS Management and Optimisation in Service-Based Systems**

<table>
<thead>
<tr>
<th>Journal</th>
<th>Transactions on Software Engineering</th>
</tr>
</thead>
<tbody>
<tr>
<td>Manuscript ID</td>
<td>Draft</td>
</tr>
<tr>
<td>Manuscript Type</td>
<td>Special Issue on Service-Based Systems</td>
</tr>
</tbody>
</table>

http://mc.manuscriptcentral.com/tse-cs
Dynamic QoS Management and Optimisation in Service-Based Systems

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

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-enabled service-based systems achieve high-level QoS requirements specified by their administrators by translating these requirements into probabilistic temporal logic formulae that are then formally and automatically analysed to identify and enforce optimal system configurations in the presence of changes in the system state and workload. The QoSMOS self-adaptation mechanism is capable of supporting both reliability- and performance-related QoS requirements, and can be integrated into newly developed solutions as well as retrofitted to existing, legacy service-based systems. The QoSMOS concepts are validated through experiments performed within a case study involving the development 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 [30], [63].

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 [59], [63]. 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. 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 [37], [80].

Achieving and maintaining well-defined Quality of Service (QoS) properties in a changing environment represent key challenges 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-adapting architectures [12], [75]. This potential explains the significant research effort that has been devoted to the definition and analysis of QoS properties in SBS systems. 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 [90], [5] on the other hand. Among these 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], [75].

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 [44]; (b) model-based QoS evaluation with probabilistic verification techniques provided by the PRISM model checker [77]; (c) monitoring and Bayesian-based parameter adaption of the QoS models exploiting KAMI [36]; and (d) planning and execution of system adaption based on GPAC [18]. The QoSMOS framework supports the practical realisation of adaptive SBS architectures through adding adaptiveness to existing and new SBSs.

Related Approaches. In SBSs, building applications through the composition of available services at runtime is a key point. 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: (i) the considered QoS metrics and QoS Specification languages (QoS Requirement Specification); (ii) the models/algorithms adopted for the QoS metric evaluation (QoS Evaluation Methods); (iii) the type of optimization problem defined and solved and/or the adaptation policies adopted (QoS Optimization or Adaptation Methods); and finally (iv) the validation of the proposed approaches (Validation).

Considering these approaches, we devise some common points of weakness that we try to 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 in service specification languages is a key point for a comprehensive management of QoS in service-based systems. Additionally, this would allow early qualitative and quantitative prediction and analysis of QoS properties of the system. This ability can reduce the overall software cost and risk.

Current examples of specification languages for QoS aspects in the web services domain are: Web Service Level Agreement (WSLA) [58], SLAng [67], the timed Web Service Constraint Language (timed WSCoL) [9] which is close to a real-time temporal logic, the Web Service Management Language (WSML) [81] and the Web Service Offerings Language (WSOL) [87].

In addition, formal QoS specification can be achieved using formalisms like real-time and probabilistic temporal logics [1], [6], [8], [48], [62], [64], timed Life Sequence Charts [49], probabilistic and timed Message Sequence Charts [53], [79], Performance Trees [86], [88] or Probabilistic Timed Behavior Trees [33], [34], [45]. To this end, QoSMOS adopts the probabilistic temporal logics PCTL [48] and CSL [8] because these logics are sufficiently expressive to formulate a variety of QoS requirements [44] whose formal verification can then be carried out using existing probabilistic model checkers. Furthermore, we adopt specification patterns [35], [44], [43], [61] and structured English grammars [44], [61], [88] to improve the usability of these logic-based specification formalisms.

QoS Evaluation Methods: To be useful, QoS evaluation approaches should rely on models representing the systems in an accurate/realistic way, and whose parameters can be adjusted at runtime 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 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 component services. Examples can be found in [39], [74], [71], [82] where, starting from the BPEL business processes modeled by UML activity diagrams or by direct acyclic graphs, performance models based on simple queuing networks [74], [71] or reliability models based on Markov models are derived [39], [82].

In line with these approaches, we argue that a comprehensive predictive quality model—possibly enhanced with run-time adaptation—is needed.

Examples of models that can be used for QoS evaluation are: Markov models, state charts like probabilistic UML state charts [54], [55], queueing networks models [16], [68], stochastic process algebras like PEPA [42] and so on [3]. Towards this end, QoSMOS adopts Markov models as modeling formalisms to determine quantitatively the reliability and performance quality metrics of service-based systems.

To check if a Markov model satisfies its QoS requirements, numerical/symbolic [6], [8], [15], [48] and statistical [89] techniques have been developed, and extensive tool support is available (e.g. PRISM [77]).

QoS Optimisation or Adaptation Methods: Devising QoS-driven adaptation methodologies of SBS 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], [24], [28], [90]). Other papers have instead considered SBS adaptation
### TABLE 1
Overview of related approaches

<table>
<thead>
<tr>
<th>Authors &amp; Citation</th>
<th>Problem Domain</th>
<th>QoS Requirements</th>
<th>QoS Evaluation</th>
<th>QoS Optimization or Adaption Method</th>
<th>Validation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Zeng et al. 2004 [90]</td>
<td>QoS-driven Service Composition/Selection and (Re-)Planning</td>
<td>Reputation, Execution Time, Availability, Price</td>
<td>Simple Aggregation Functions (eg. Sum, Product, Max, Average) based on unfolded state-based process specifications with branch &amp; loop probabilities</td>
<td>Local &amp; Global Planning with Inter-Programming (Multi-objective with simple additive weighting, positive &amp; negative QoS Attributes)</td>
<td>Simulation with generated examples</td>
</tr>
<tr>
<td>Cao et al. 2005 [25]</td>
<td>QoS-driven Service Selection</td>
<td>Cost</td>
<td>Simple Aggregation Function (Sum) for parallel, sequential and branching service composition</td>
<td>Genetic Algorithm</td>
<td>Experiments with generated examples</td>
</tr>
<tr>
<td>Cemofa et al. 2005-2008 [22], [23], [24]</td>
<td>QoS-driven Service Binding and (Re-)Binding/Selection</td>
<td>(Execution) Time, Reliability, Availability, Price</td>
<td>Simple Aggregation Functions (eg. Sum, Product, Max, Average) similar to [90] based on an unfolded BPEL4WS specification with branch and loop probabilities</td>
<td>Genetic Algorithms</td>
<td>Experiments based on a Travel Planer case study</td>
</tr>
<tr>
<td>Cardellini et al. 2007-2008 [27], [26], [28]</td>
<td>QoS-driven Service Composition/Selection</td>
<td>Response Time, (log)-Availability, Cost</td>
<td>Simple Aggregation Functions (eg. Sum, Product, Max, Average) similar to [90] based on a process activity trees generated from BPEL specifications</td>
<td>Linear Programming (Multi-objective with a suitable objective function (eg. weighted sum), multiple service activations), adaption through workflow restructuring in [26]</td>
<td>Experiments with generated examples</td>
</tr>
<tr>
<td>Menasce et al. 2007, 2008 [72], [73], [74]</td>
<td>QoS-driven Service Composition/Selection</td>
<td>Average Execution Time (with cost constraints)[72] + throughput [73], [74]</td>
<td>Aggregation Function for the average execution time based on an tree based representation (BPTree) of a BPEL specification [72] and queuing network based performance evaluation [73], [74]</td>
<td>Exact method [72], [73] and hill-climbing inspired heuristic [72]; use of utility functions [73] for the quality dimensions</td>
<td>Experiments with generated examples based on the Travel Planer case study</td>
</tr>
<tr>
<td>Zhang et al. 2007, 2008 [69], [81], [89]</td>
<td>QoS-driven Service Selection</td>
<td>same as [90]</td>
<td>Simple Aggregation Functions</td>
<td>(Quickly Convergent) Population Diversity Handling Genetic Algorithm (DiGA [91] and CoDiGA [69]) with Simulated Annealing</td>
<td>Generated experiments based on the Travel Planer case study</td>
</tr>
<tr>
<td>Bertsen et al. 2006 [13]</td>
<td>QoS-driven Service Composition/Selection</td>
<td>Availability, Response time, and Throughput</td>
<td>Simple Aggregation Functions (Sum, Product, Min) similar to [90] based on a sequential web service composition</td>
<td>Iterative approach: (1) Mixed Integer Programming + Backtracking, (2) Simulated Annealing or Random Swapping of Services (Mutation))</td>
<td>Simulation and comparison (IP vs. SA) with generated examples</td>
</tr>
<tr>
<td>Ko et al. 2008 [60]</td>
<td>QoS-driven Service Composition/Selection</td>
<td>same as [90] + Frequency and Succ. exec. rate</td>
<td>Simple Aggregation Functions sequential, AND-parallel and XOR-parallel web service composition</td>
<td>Tabu-search+ plan generation based on neighborhood search</td>
<td>Simulation based on generated examples</td>
</tr>
<tr>
<td>Boonea et al. 2010 [17]</td>
<td>QoS-driven Load Balancing</td>
<td>Response Time</td>
<td>Solving of M/M/1 queueing systems with Poisson distributed service requests</td>
<td>Simulated Annealing Load Scheduling Algorithm (SALSA)</td>
<td>Experiments based on gen. testbeds</td>
</tr>
<tr>
<td>Marzolla et al. 2007 [71]</td>
<td>QoS-driven Workflow Management</td>
<td>Response Time</td>
<td>Analytic solving of BCMP queueing systems</td>
<td>Exact methods and bound analysis</td>
<td>Experiments based on gen. examples</td>
</tr>
<tr>
<td>Sato et al. 2007 [82]</td>
<td>QoS-driven Workflow Management</td>
<td>Reliability</td>
<td>Analytic solving of Markov Models</td>
<td>Exact solution based on CTMC analysis</td>
<td>Experiments based on gen. examples</td>
</tr>
<tr>
<td>Guo et al. 2007 [47]</td>
<td>QoS-driven Workflow Management</td>
<td>Availability</td>
<td>Simple Aggregation Functions sequential, choice, parallel and iterative web service composition</td>
<td>Use of redundancy mechanisms to improve availability</td>
<td>Experiments based on gen. examples</td>
</tr>
</tbody>
</table>

Towards this end, the QoSOMS 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 the different approaches preform experiments based on generated examples or apply a case study based validation. To validate the QoSOMS approach we use a validation based on experiments performed based on the common case study of a service-based system for remote medical assistance called Tele-

---

through workflow restructuring, exploiting the inherent redundancy of SBS (e.g., [29], [47], [50]). In [26] 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.

Validation: An investigation of the validation strategies, shows that the different approaches preform experiments based on generated examples or apply a case study based validation. To validate the QoSOMS approach we use a validation based on experiments performed based on the common case study of a service-based system for remote medical assistance called Tele-
Assistance [10], [36].

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, dependability 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 to other services and to 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 of 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 its validation through a case study is presented in Section 4. Section 5 presents the conclusions and 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 [38]. In QoSMOS temporal logics are used as specification formalism to specify QoS requirements. Examples are real-time temporal logics such a MTL (Metric Temporal Logic) [62] and TCTL (Timed Computational Tree Logic) [1] or probabilistic temporal logics such as PCTL (Probabilistic Computational Tree Logic) [48], PCTL* [6], PTCTL (Probabilistic Timed CTL) [64] and CSL (Continuous Stochastic Logic) [8]. The benefit of a logic-based specification of QoS requirements are the expressiveness and formal semantics. Furthermore, for logic-based specification-formalism the correct definition of QoS properties is supported with specification patterns [35], [44], [43], [61] and structured English grammars [44], [61], [88].

In this article we focus on the PCTL and CSL which is defined as follows [8], [31], [48]:

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

\[
\Phi ::= \text{true} | \neg \Phi | \Phi_1 \land \Phi_2 | \Phi_1 \lor \Phi_2 | \Phi_1 \Rightarrow \Phi_2 | \Phi_1 \Leftrightarrow \Phi_2 | \Phi_1 \forall \Phi_2 | \Phi_1 \exists \Phi_2 | \Phi_1 \ltimes \Phi_2 | \Phi_1 \ldots \Phi_n \ltimes \Phi_{n+1} \\
\Psi ::= X \Phi | X^\omega \Phi | X^{\omega \leq t} \Phi | X^{\omega > t} \Phi | X^{\omega < t} \Phi | X^{\omega \geq t} \Phi |
\]

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

\[
\Phi ::= \text{true} | \neg \Phi | \Phi_1 \land \Phi_2 | \Phi_1 \lor \Phi_2 | \Phi_1 \Rightarrow \Phi_2 | \Phi_1 \Leftrightarrow \Phi_2 | \Phi_1 \ltimes \Phi_2 | \Phi_1 \ldots \Phi_n \ltimes \Phi_{n+1} | \Phi_1 \ltimes \Phi_2 |
\]

The logics distinguish between state and path formulae. The state formulae include the standard logical operators \( \land \) and \( \neg \), which also allow a formulation of other usual logical operators (disjunction (\( \lor \)), implication (\( \Rightarrow \)), etc.) and \( \text{false} \). 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 \( P \). This probabilistic operator defines an upper or a lower bound on the probability of the system evolution. As an example, the formula \( P_\geq t \Psi \) is true at a given time, if the probability that the future evolution of the system which satisfies \( \Psi \) is at least \( t \). Similarly, the formula \( P_{\leq t} \Psi \) is true if the probability that the system fulfills \( \Psi \) is less than or equal to \( t \). The path formulae that can be used with the probabilistic path operator are the “next” formula \( X \Phi \), time bounded “until” formula \( \Phi_1 \ltimes \Phi_2 \) 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 \( \Phi_1 \ltimes \Phi_2 \) requires that the \( \Phi_1 \) 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” \( \& \)) can be expressed as described in [31], [44]. For example the eventually formula \( P_{\geq t} \Phi \) is semantically equivalent to \( P_{\geq t}(true U \Phi) \). As an additional syntactical feature the CSL has been extended in [8] with a steady state operator \( S \) that describes the behavior of the system in the long run. Syntactically this operator (state formula: \( S_{\geq t} \Psi \)) is used similar to the probabilistic path operator.

Traditionally, the semantics of the PCTL/CSL is defined with a satisfaction relation \( \models \) over the states \( S \) and possible paths \( Path^M(s) \) that are possible in a state \( s \in S \) of a discrete/continuous time probabilistic model \( M \). For details about the formal semantics the reader is referred to [8], [31], [48]. Normally, a PCTL/CSL formula is evaluated starting from the initial state of the probabilistic model \( M \). 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 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 summarized in the following.

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 [16], 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) [78] that are an extension of DTMC allowing multiple probabilistic behaviours to be specified as output of a state, considering these behaviours non-deterministically. MDP is characterized by a discrete set of states representing possible configurations of the system being modeled and transitions between states occur in discrete time-steps, but in each state there is also a non-deterministic choice between several discrete probability distributions over successor states.

The solution of Markovian models aims at determining the system behaviour as the time \( t \) approaches to infinite. It consists of the evaluation of the stationary probability \( \pi_s \) of each state \( s \) of the model. The solution techniques differ according to the specific model and to the underlying assumptions (e.g., transient or nontransient 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 system can be obtained only if \( S \) is finite or when the matrix of transition probabilities has a specific form. The main problem of Markov models is the explosion of the number of states when they are used to model real systems [16]. On the other hand, Markov models are very general since they can include other modelling approaches as special cases, such as queueing networks, Stochastic Petri Nets [70] and Stochastic Process Algebras [32]. Markov models allow estimating both performance and reliability metrics and allow modelling systems without imposing any restriction on the users or resources behaviour. Finally, as a general consideration, the accuracy of Markov models depends on the precision of the state transition probability matrix, which may possibly include, a quite large number of parameters.

3 QoSMOS Architecture

This section introduces the generic QoSMOS architecture of an adaptive service-based system, and describes its 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 dependability, 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 [57].

![Fig. 1. The architecture of a service-based system.](http://mc.manuscriptcentral.com/tse-cs)
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.

### 3.1 Generic architecture of QoSMOS

As illustrated in Figure 2, 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 [59], [51] 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.

#### 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 dependability (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.

#### 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.

#### 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 is defined in terms of the abstract services needed to implement the intended SBS functionality, i.e., an abstract workflow. Based on the analysis results, the abstract services within this workflow are mapped to concrete services during the planning stage.
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 dependability-related 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, dependability 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 dependability-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 dependability-related QoS metrics is obtained without impacting the SBS response time, but potentially
QoSMOS supports the use of a different mapping pattern \( (mp_i \in \{\text{SGL, SEQ, PAR}\}) \) for each abstract service \( as_i, \ 1 \leq i \leq m \), that is idempotent. Idempotent abstract services are services that can be called several times without affecting the outcome of the SBS workflow; examples of such services are presented in Section 4.1. For non-idempotent services, the only mapping pattern that can be used is SGL.

### 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, realising this functionality in QoSMOS is straightforward.

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.

### 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 2.

#### 3.2.1 PRISM

PRISM [65], [66], [77] 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].
properties to be established are specified using PCTL (Probabilistic Computation Tree Logic) [48] 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 [65]. 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 [77]. An extensive, independent performance analysis of a broad selection of probabilistic model checkers [56] 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.

### 3.2.2 ProProST

To ease the formalization of QoS requirements, the idea of specification patterns [35], [61] has been recently investigated for probabilistic logics [44]. 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 3. Please note, that seven of the academic properties are composites of the eight patterns that are counted separately.

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. For these two tasks we have implemented a wizard (cp. Figure 3), which helps QoSMOS Admins to select the appropriate pattern and clearly define a QoS requirement in a probabilistic temporal logical formula. The ProProST wizard is based on the structure English grammar that is presented in [44]. As a result, new QoS requirements can be easily specified or existing QoS requirements could be relaxed or strengthened, by QoSMOS Admins with limited knowledge in probabilistic temporal logics.

#### 3.2.3 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 by 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 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

<table>
<thead>
<tr>
<th>Pattern Name</th>
<th>Logical Formulation</th>
<th>Academic</th>
<th>Industrial</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transient State Probability</td>
<td>$P_{\text{exp}}[\Phi]$</td>
<td>6</td>
<td>0</td>
</tr>
<tr>
<td>Steady State Probability</td>
<td>$S_{\text{exp}}[\Phi]$</td>
<td>18</td>
<td>1</td>
</tr>
<tr>
<td>Probabilistic Invariance</td>
<td>$P_{\text{exp}}[\Phi]$ or $P_{\text{exp}}[\neg \Phi]$</td>
<td>6</td>
<td>18</td>
</tr>
<tr>
<td>Probabilistic Existence</td>
<td>$P_{\text{exp}}[\Phi]$ or $P_{\text{exp}}[\Phi]$</td>
<td>57</td>
<td>9</td>
</tr>
<tr>
<td>Probabilistic Until</td>
<td>$P_{\text{exp}}[\Phi]$</td>
<td>30</td>
<td>2</td>
</tr>
<tr>
<td>Probabilistic Precedence</td>
<td>$\mathcal{P}<em>{\text{exp}}[\Phi]$ or $\mathcal{P}</em>{\text{exp}}[\neg \Phi]$</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Probabilistic Response</td>
<td>$\mathcal{P}_{\text{exp}}[\Phi]$</td>
<td>18</td>
<td>13</td>
</tr>
<tr>
<td>Probabilistic Constrained Response</td>
<td>$\mathcal{P}_{\text{exp}}[\Phi]$</td>
<td>4</td>
<td>3</td>
</tr>
</tbody>
</table>

**TABLE 3**

Probabilistic specification patterns

---

http://mc.manuscriptcentral.com/tse-cs
from the modeling point of view. For example, in SBS systems such event are service invocations and their results. In particular, in the SBS domain several approaches can be adopted to obtain these data (e.g., [11]).

By collecting run-time data from running instances of the system, KAMI feeds a Bayesian estimator [14] as defined in [36] 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. Modified models provide a closer 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 4. KAMI currently supports DTMCs [36] and Queueing networks [41]. We will use the KAMI features within the QoSMOS MAPE loop, to perform the monitoring stage allowing the automatic model parameter tuning.

3.2.4 GPAC

GPAC (General-Purpose Autonomic Computing) is a tool-supported methodology for the model-driven development of self-managing IT systems [18]. The core component of GPAC is a generic autonomic manager capable of augmenting existing IT systems with a MAPE autonomic computing loop. The autonomic manager 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 autonomic manager as part of a run-time configuration step [19]. 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 2 and can be specified by means of quality evaluation models such as those described in Section 2.2. The autonomic manager is implemented as a service-oriented architecture, and employs advanced object-oriented technology capabilities such as reflection-oriented programming [84] and generic programming [40] 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 that the autonomic manager uses to monitor and control the parameters of the managed system [19]. This tool was used to speed up the development of autonomic solutions in several application domains [18], and was recently integrated into a GPAC environment for the computer-aided development of autonomic systems [20].

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 $c_1, c_2, \ldots, c_N$ are dynamically adjusted to maximise the utility of the system. These utility optimi-
the (index of the) concrete service used by each abstract service, and the mapping pattern used for each abstract service.

In our realisation of the QoSMOS architecture, the system objectives from eq. (2) are expressed using instances of ProProST specification patterns from Table 3. As some of these pattern instances correspond to constraints that the service-based system must comply with at all times, the first objective from eq. (2) has the form

$$\text{objective}_1 = \prod_{j=1}^{q} \text{goal}(\text{ProProST}_{-}\text{pattern}_j),$$

where $\text{ProProST}_{-}\text{pattern}_j$, $1 \leq j \leq q$, is an instance of a ProProST pattern from Table 3 that expresses a system constraint (e.g., a lower bound for the probability of an operation succeeding), and the function $\text{goal} : \{\text{false, true}\} \rightarrow \{0, 1\}$ is defined by $\text{goal}($false$) = 0$ and $\text{goal}($true$) = 1$. Notice that this definition ensures that $\text{objective}_1$ has value 1 if all $q \geq 1$ constraints are satisfied, and value 0 otherwise.

The other objectives from eq. (2) are also expressed in terms of instances of ProProST patterns, but define QoS properties that the MAPE loop should optimise (e.g., cost or system response time), subject to the constraints encoded as $\text{objective}_1$ being satisfied. To ensure that the constraints specified by means of $\text{objective}_1$ are given precedence over the QoS properties specified by the other objectives, it is sufficient to choose the weights $w_i$, $1 \leq i \leq r$, from eq. (2) such that

$$w_1 + \min_{(x_1, x_2, \ldots, x_N) \in C_1 \times C_2 \times \ldots \times C_N} \sum_{i=2}^{r} w_i \text{objective}_i > \max_{(x_1, x_2, \ldots, x_N) \in C_1 \times C_2 \times \ldots \times C_N} \sum_{i=2}^{r} w_i \text{objective}_i.$$  

(5)

Indeed, this choice of weights guarantees that any configuration $(x_1, x_2, \ldots, x_N) \in C_1 \times C_2 \times \ldots \times C_N$ for which $\text{objective}_1 = 1$ corresponds to a higher system utility than any other configuration for which $\text{objective}_1 = 0$. Furthermore, when $\text{objective}_1 = 0$ for all possible system configurations, the maximum value for the utility function from eq. (2) is identical to the second term of inequality (5), hence the condition below holds:

$$\text{cond}_1 = \max_{(x_1, x_2, \ldots, x_N) \in C_1 \times C_2 \times \ldots \times C_N} \text{utility}(\ldots) <$$

$$w_1 + \min_{(x_1, x_2, \ldots, x_N) \in C_1 \times C_2 \times \ldots \times C_N} \sum_{i=2}^{r} w_i \text{objective}_i.$$  

(6)

This property can be used as the condition of an instance of the action policy (3), to raise an alarm notifying the SBS administrator about the SLA failure:

$$\text{if } \text{cond}_1 \text{ then SBS\textunderscore admin\textunderscore alarm = true.}$$  

(7)

Possible configurable parameters for a QoSMOS adaptive service-based system (i.e., $c_1, c_2, \ldots, c_N$ from eqs. (1)-(2)) include:

- the mapping pattern used for each abstract service from the SBS workflow;
- the (index of the) concrete service used by each SGL mapping pattern, and the set of indices for the concrete services employed by each SEQ and PAR mapping pattern;
- the resources (e.g., CPU, physical servers or bandwidth capacity) allocated to the in-house services.

Section 4 provides concrete examples of QoSMOS system objectives, and of configurable parameters adjusted by the QoSMOS MAPE loop in order to achieve such objectives.

http://mc.manuscriptcentral.com/tse-cs
3.3.2 SBS objective realisation

SBS objectives specified by the system administrator as described in the previous section are supplied to the GPAC autonomic manager. Note that these objectives can be modified as and when required at run-time, a characteristic that sets QoSOMOS aside from other approaches to building adaptive service-based systems.

As indicated by the generic QoSOMOS architecture in Figure 2, the autonomic manager is also supplied with (a) the abstract SBS workflow and (b) a GPAC system model with the characteristics presented in Section 3.2.4.

Both of these are provided by the developer of the service-based system.

The element of this model that describes the behaviour of the service-based system is represented by a set of PRISM models that correspond to the operational model from the QoSOMOS architecture in Figure 2. The KAMI component of our QoSOMOS prototype ensures that the parameters of the concrete services within the PRISM models are maintained in line with the actual behaviour of these services. Other parameters of the PRISM models represent configurable SBS parameters such as the indices of the concrete services used by the SBS workflow, the number of redundant invocations of the operations of these services, and the amount of resources allocated to in-house concrete services.

The MAPE loop of the adaptive SBS employs the ProProST PRISM extension described in Section 3.2.2 to obtain the various probabilistic temporal logic properties associated with the SBS objectives. It then performs a number of PRISM experiments, each of which analyses one such property for all possible values that can be assigned to the configurable SBS parameters.

In the planning stage of the MAPE loop, the results of the PRISM experiments are parsed and used to choose optimal values for the configurable SBS parameters. To achieve this, QoSOMOS employs the straightforward exhaustive search algorithms described in our previous work in [21]. Note that this approach works well for the service-based systems targeted by QoSOMOS 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).

In the execution stage of the MAPE loop, the optimal values chosen for the configurable SBS parameters are converted into a concrete workflow that is supplied to the SBS workflow engine; and/or into new resource allocations for the services run in-house. The processes involved are depicted in Figure 2.

4 Validation

This section illustrates a case study adopted to validate the QoSOMOS approach. First of all, we introduce the case study and, secondly, we illustrate how the QoSOMOS approach can be successfully applied to it.

4.1 The TeleAssistance case study

The case study we describe in this article is taken from [10], [36] and involves a service-based system for remote medical assistance. This system is called TeleAssistance (TA), and its associated BPEL workflow is depicted in Figure 5.

The TA system incorporates the following abstract services, orchestrated as described later on:

- **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 the invocation of the FAS service.

The workflow in Figure 5 represents the orchestration of abstract services. Different providers could be involved in providing concrete implementations for the abstract services in the TA service-based system. For example, the Alarm Service could be implemented by several telecommunication operators, with different cost, performance and reliability characteristics. To obtain a concrete workflow for the SBS, each abstract service from the TA workflow in Figure 5 has to be mapped to a concrete service, the number of redundant invocations for the concrete service operations must be established, and the amount of resources allocated to in-house services has to be decided. The remainder of this section describes how these configuration steps are performed within our QoSOMOS application to TA service-based system.

We will start by defining the TeleAssistance SBS formally, using the notation introduced in Section 3. Thus, the TA workflow in our case study comprises $m = 3$
The sets of concrete services implementing the functionalities of these abstract services contain \( n_1 = 3 \), \( n_2 = 5 \) and \( n_3 = 1 \) concrete services, respectively. Each concrete service \( s_{ij}^i \), \( 1 \leq i \leq m, 1 \leq j \leq n_i \), 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;
- \( \text{idemp}^i_j \in \{\text{true}, \text{false}\} \), the parameter that specifies whether the service is idempotent or not—recall from Section 3.1 that idempotent services can be invoked repeatedly without affecting the outcome of the SBS workflow (but with an increased probability of overall success).

Additionally, third-party concrete services are characterised by their expected execution time (\( t_i^j \)); and in-house concrete services by their request inter-arrival rate (\( \mu_i^j \)) and maximum request service rate (\( \lambda_i^j \)). The maximum request service rate for a concrete, in-house service represents the request service rate when the concrete service is allocated the maximum amount of CPU resources on the server(s) on which it is running. As described later in this section, the CPU resources allocated to in-house services are adjusted dynamically in a QoS-enabled SBS, to ensure that the QoS requirements for the system are achieved with minimal resource usage.

Table 4 shows the initial values of these parameters for the TA service-based system. Remember that QoS is updating the values of these parameters continuously, based on its monitoring of the concrete services. Also, 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 a 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.

The configurable parameters of the TA service-based system are:

(a) the mapping patterns \( mp_i \in \{\text{SGL, SEQ, PAR}\} \), \( 1 \leq i \leq m \), used for the \( m = 3 \) abstract services (note that \( mp_3 = \text{SGL} \) at all times since the Drug Service is non-idempotent);

(b) the concrete service sequences \( \langle s_{i1}^j, s_{i2}^j, \ldots, s_{is_i}^j \rangle, \) \( 1 \leq i \leq m \), used to implement the \( m = 3 \) abstract services, where \( S_i \geq 1 \) and \( \{j_1, j_2, \ldots, j_{s_i}\} \subseteq \{1, 2, \ldots, n_i\} \) (note that \( S_1 = 1 \) for all abstract services \( a_s_i \) for which \( mp_i = \text{SGL} \));

(c) the amount of CPU to be allocated to the in-house TA services (i.e., \( cpu_i^j \in [0, 1] \) for the TA case study, whose only in-house service is \( s_{j1}^1 \)).

As described in Section 3.3.2, these configurable parameters are continually updated by the QoS-enabled MAPE loop so that the QoS requirements specified by the TA administrator are achieved at all times. The reliability-related requirements considered by our case study 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 is \( N = 10 \). These requirements are described below:

\( R_0 \) The probability \( P_0 \) that at an alarm failure ever occurs during the whole lifetime of the system is less than \( P_0 = 0.13 \).

\( R_1 \) The probability \( P_1 \) that a service failure ever occurs during the whole lifetime of the system is less then


### Table 4

Concrete services for the TA service-based system

<table>
<thead>
<tr>
<th>Concrete service</th>
<th>Name</th>
<th>Failure rate ($r_i$)</th>
<th>Expected execution time ($t_i$)</th>
<th>Maximum request service rate ($\lambda_i$)</th>
<th>Request inter-arrival rate ($\mu_i$)</th>
<th>Cost ($c_i$)</th>
<th>Idempotent (idem$i$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$s_1$</td>
<td>AlarmService1</td>
<td>0.01</td>
<td>1.1s</td>
<td>–</td>
<td>–</td>
<td>4.1</td>
<td>true</td>
</tr>
<tr>
<td>$s_2$</td>
<td>AlarmService2</td>
<td>0.04</td>
<td>0.9s</td>
<td>–</td>
<td>–</td>
<td>2.5</td>
<td>true</td>
</tr>
<tr>
<td>$s_3$</td>
<td>AlarmService3</td>
<td>0.008</td>
<td>0.3s</td>
<td>–</td>
<td>–</td>
<td>6.8</td>
<td>true</td>
</tr>
<tr>
<td>$s_4$</td>
<td>MedicalAnalysisService1</td>
<td>0.0006</td>
<td>2.2s</td>
<td>–</td>
<td>–</td>
<td>9.8</td>
<td>true</td>
</tr>
<tr>
<td>$s_5$</td>
<td>MedicalAnalysisService2</td>
<td>0.001</td>
<td>2.7s</td>
<td>–</td>
<td>–</td>
<td>8.9</td>
<td>true</td>
</tr>
<tr>
<td>$s_6$</td>
<td>MedicalAnalysisService3</td>
<td>0.0015</td>
<td>3.1s</td>
<td>–</td>
<td>–</td>
<td>9.3</td>
<td>true</td>
</tr>
<tr>
<td>$s_7$</td>
<td>MedicalAnalysisService4</td>
<td>0.0025</td>
<td>2.9s</td>
<td>–</td>
<td>–</td>
<td>7.3</td>
<td>true</td>
</tr>
<tr>
<td>$s_8$</td>
<td>MedicalAnalysisService5</td>
<td>0.0005</td>
<td>2.0s</td>
<td>–</td>
<td>–</td>
<td>11.9</td>
<td>true</td>
</tr>
<tr>
<td>$s_9$</td>
<td>DrugService1</td>
<td>0.0012</td>
<td>–</td>
<td>2.4s$^{-1}$</td>
<td>0.15s$^{-1}$</td>
<td>0.1</td>
<td>false</td>
</tr>
</tbody>
</table>

1) Only for third-party concrete services  
2) Only for in-house concrete services

$P_1 = 0.14$.

$R_2$ 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 $P_2 = 0.012$.

$R_3$ 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 $P_3 = 0.015$.

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 0.001.

### 4.2 QoSMOS at work

#### 4.2.1 Introduction

This section illustrates how the QoS management approach is applied to the TA system. As described in Section 3.1, the QoS-enabled TA system relies on three inputs: the abstract workflow, the operational model and the set of QoS requirements. These three inputs are presented below.

**Abstract SBS workflow.** The abstract workflow is provided by the SBS developer during the SBS development stage. For the TA case study, the workflow is depicted in Figure 5 and described in detail in the previous section.

**Operational Model.** The initial version of the operational model is also provided by the SBS developer. This version is then updated automatically by the QoSMOS MAPE loop through monitoring the actual behavior of the TA components. Because the QoS requirements for the QoS-enabled TA system include both reliability- and performance-related requirements, the TA operational model comprises a DTMC model associated with the reliability requirements $R_0$ to $R_3$, and a CTMC model associated with the performance requirements $R_4$ and $R_5$:

- The DTMC model used to achieve the reliability-related requirements is depicted in Figure 6. 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 gray). 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. 6. TeleAssistance DTMC model.](http://mc.manuscriptcentral.com/tse-cs)
(i.e., \(s_1^j\)), and the CTMC that models the operation of this service is shown in Figure 7. The parameters of this CTMC model are: \(Q_{\text{max}} > 0\), the size of the request queue for the service; \(\mu_1^j > 0\), the request arrival rate; \(\lambda_1^j \in [0, 1]\), the fraction of CPU resources allocated to the service on the server on which it is run; and \(\lambda_1^j > 0\), the request service rate corresponding to \(\text{cpu}_1^j\) = 1. Accordingly, states \(S_i\) \(0 \leq i \leq Q_{\text{max}}\), from the CTMC model in Figure 7 correspond to the service request queue containing \(i\) requests with no request being dropped; and state \(S_{Q_{\text{max}}+1}\) corresponds to the queue being full and requests being dropped.

![Figure 7. CTMC model for the in-house concrete service \(s_1^j\).](image)

The QoSMOS TA system used a fixed-size request queue (i.e., \(Q_{\text{max}} = 10\)), and employed its MAPE loop to monitor \(\mu_1^j\) and \(\lambda_1^j\) for changes from the initial values from Table 4, and to adjust the configurable parameter \(\text{cpu}_1^j\) so that the performance-related QoS requirements \(R_4\) and \(R_5\) were achieved in the presence of variations in the values of these parameters.

**QoS Requirements.** The QoS requirements are provided and updated whenever needed by the SBS administrator. The reliability and performance requirements \(R_0\) to \(R_5\) for the TA case study are ProProST pattern instances defined as probabilistic temporal logical formulae based on the labels defined in the operational models (Figures 6 and 7). \(R_0\), \(R_1\) and \(R_2\) are Probabilistic Existence properties, \(R_3\) and \(R_4\) are filtered Probabilistic Until properties, and \(R_5\) 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 presented in Section 3.2.2 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 logical formula \(P_{\leq 0.13}[\text{"failedAlarm"]}\). As a result of the structured process for the formulation of probabilistic properties, the QoSMOS administrator derived the following temporal logical formulae that correspond to the QoS requirements that the TA service-based system must satisfy:

\[
\begin{align*}
\text{ProProST}_1 & : P_{\leq 0.13}[\text{"failedAlarm"]} \\
\text{ProProST}_2 & : P_{\leq 0.14}[\text{"failure"]} \\
\text{ProProST}_3 & : P_{\leq 0.015}([\text{"stopMsg"} & \text{"vitalParamsMsg"} & \text{"FAS"}]U\text{"failedAlarm"}) \\
\text{ProProST}_4 & : P_{\leq 0.014}([\text{"alarm"]} & \text{"analyzeData"})U\text{"failedAlarm"} \\
\text{ProProST}_5 & : P_{\leq 0.001}([\text{"Failed/trig"]} \& \text{"dropped"})
\end{align*}
\]

The above QoS requirements are employed to encode the constraints-based objective \(\text{objective}_1\) for the QoSMOS-enabled TA SBS, as specified in eq. (4). The two other objectives for the system include the minimisation of the overall cost of running the system, and the minimisation of the amount of CPU allocated to the in-house service (i.e., \(DrugServ1\)):

\[
\begin{align*}
\text{objective}_1 & = \prod_{i=0}^{5} \text{goal}([\text{ProProST}_i]) \\
\text{objective}_2 & = -\sum_{i = 1}^{3} \sum_{x=1}^{s_i} s_i^{x} j_i^{x} - \sum_{i = 1}^{3} c_i^{x} \\
\text{objective}_3 & = -\text{cpu}_1^j
\end{align*}
\]  

(8)

where \((s_1^j, s_2^j, \ldots, s_{Q_{\text{max}}}^j)\), \(S_i \geq 1\) represents the sequence of concrete services used to implement the \(i\)-th abstract service using a SEQ or PAR mapping pattern for idempotent abstract services. An upper bound \(S \geq S_1, S_2\) 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

\[
\text{utility} = \sum_{i=1}^{3} w_i \text{objective}_i 
\]

(9)

to be maximised are chosen such as to ensure that (5) holds for the concrete service cost values in Table 4:

\[
\begin{align*}
w_1 & = 100S \text{ } & w_2 & = 1 \text{ } & w_3 & = 10
\end{align*}
\]  

(10)

This choice of weights ensures that \(\text{utility} < 0\) whenever any of the \(\text{ProProST}_i\) constraints is not satisfied. Therefore, the action policy (7) for our QoSMOS SBS becomes:

\[
\text{if } \text{utility} < 0 \text{ then } \text{SBS}_\text{admin alarm} = \text{true}. 
\]

(11)

### 4.2.2 QoSMOS MAPE loop

**Monitoring Stage.** In this stage of the MAPE loop, the KAMI component of QoSMOS is monitoring all the parameters of the concrete services whose design-time predicted values are shown in Table 4. KAMI ensures that any changes in the values of these parameters are reflected in the operational model that QoSMOS employs in the subsequent stages of the MAPE loop. Before describing how the QoSMOS TA system adapts to such changes in Section 4.2.3, we will present the other stages of the MAPE loop.

**Analysis Stage.** In this stage, the QoSMOS MAPE loop analyses the PCTL and CSL properties associated with

http://mc.manuscriptcentral.com/tse-cs
the reliability- and performance-related QoS requirements for the system, and obtained through the automated translation of the ProProST pattern instances into such properties. The analysis is performed through running background PRISM experiments that employ the DTMC and CTMC components of the operational model for the SBS.

We will first describe the analysis associated with the reliability-related QoS requirements $R_0$ to $R_3$ for the system. The PCTL formulae below were obtained from the translation of the ProProST pattern instances for these requirements:

$$R_0: P \leq 0.13 \left[ \text{true U "failedAlarm"} \right]$$
$$R_1: P \leq 0.14 \left[ \text{true U "failure"} \right]$$
$$R_2: P \leq 0.15 \left[ \left( \neg \text{"stopMsg"} \land \neg \text{"vitalParamsMsg"} \land \neg \text{"FAS"} \right) \U \left\{ \text{"changeDrug"} \mid \text{"changeDoses"} \right\} \right]$$
$$R_3: P \leq 0.014 \left[ \left( \text{"alarm"} \mid \text{"analyzeData"} \right) \U \left\{ \text{"failedAlarm"} \mid \text{"analyzeData"} \right\} \right]$$

Given these PCTL properties and the DTMC model in Figure 6, the analysis involves running PRISM experiments that explore the configuration space of the TA system. The PRISM experiments consider all possible mapping patterns and service bindings for the SBS workflow, as described in Section 3.3.1. 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 8a and Figure 8b, 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 MedicalService was chosen to be $MedicalService$. The configurable SBS parameters that were varied in the PRISM experiments shown in Figures 8a and 8b 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 9. 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 $DrugService$ was set to the value in Table 4. The graph in Figure 9 depicts the variation of the probability of failure from requirement $R_0$ for different failure rates for the Medical Service and the 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. Notice that 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.
TABLE 5
Combination of Alarm Services with different sequential one-to-many mapping

<table>
<thead>
<tr>
<th>SEQ Index</th>
<th>Number of Services</th>
<th>Services</th>
<th>Aggregate Failure Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>$s_1^2$</td>
<td>0.01</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>$s_1^3$</td>
<td>0.04</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>$s_1^4$</td>
<td>0.08</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>$s_1^5, s_1^6$</td>
<td>0.0004</td>
</tr>
<tr>
<td>5</td>
<td>2</td>
<td>$s_1^7, s_1^8$</td>
<td>0.00008</td>
</tr>
<tr>
<td>6</td>
<td>2</td>
<td>$s_1^9, s_1^{10}$</td>
<td>0.000032</td>
</tr>
<tr>
<td>7</td>
<td>3</td>
<td>$s_1^11, s_1^12, s_1^{13}$</td>
<td>0.0000032</td>
</tr>
</tbody>
</table>

the system. The results of these experiments form a multi-dimensional surface in a hyperspace whose dimensionality is given by the number of configurable SBS parameters; the graphs in Figures 8 and 9 represent the intersections of these surfaces with hyperplanes obtained by fixing the value of several SBS parameters. Furthermore, note that 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 QoSMDOS MAPE loop, and eliminates the overheads associated with producing the graphs.

In addition to performing the PRISM experiments described so far, the analysis stage involves PRISM experiments that analyse the CSL properties associated with the performance-related QoS requirements for the system:

1. $R_4$: $s<0.2 \ [ q/Q_{max} > 0.75 ]$
2. $R_5$: $P<0.001 \ [ \text{true \ U \ [0,86400]} \ "dropped" ]$

Figure 10 shows the result of the PRISM experiments carried out for the analysis of the CSL property for requirement $R_4$, with respect to different CPU allocations. The predicted value for the request arrival rate $\mu^1_3$ and the request service rate $\lambda^1_1$ from Table 4 were used for this experiment; the length of the request queue was fixed at $Q_{max} = 10$. The dashed line indicates that a CPU allocation equal to $cpu^1_3 = 0.395$ is the minimum value necessary to meet the requirement.

Finally, consider requirement $R_5$, again assuming an expected rate of incoming requests $\mu^1_3 = 0.15$ and a $Q_{max}$ value equal to 10. Figure 11 shows the evaluation of requirement $R_5$. In this case, the minimum amount of CPU to achieve the performance-related requirement is $cpu^1_5 = 0.305$.

![Fig. 10. Evaluation of $R_4$ for the TA system.](image)

![Fig. 11. Evaluation of $R_5$ for the TA system.](image)

1. An upper limit is placed (by the SBS administrator) on the length of the sequences of concrete services to be considered by the MAPE loop for the SEQ and PAR mapping patterns.
effect of setting this alarm is explained when we describe the execution stage of the MAPE loop below.

**Execution Stage.** If any of the mapping patterns mp_i or the concrete-service sequences \((s_i^1, s_i^2, \ldots, s_i^{3_i})\), \(1 \leq i \leq 3\), 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^j 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 (11) 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.

### 4.2.3 QoS/MOS Adaptiveness

In this section, we look at how the QoS/MOS 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 4. Indeed, in this setting we have \(P_0 = 0.128 \leq 0.13\), \(P_1 = 0.134 \leq 0.14\), \(P_2 = 0.006 \leq 0.012\), \(P_3 = 0.015 \leq 0.015\), \(P_4 = 0.2 \leq 0.2\) and \(P_5 = 0.00097 \leq 0.001\). However, the characteristics of services evolve over time and can lead to requirement violations. For instance, Figure 12 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.

Therefore, once the concrete workflow corresponding to the initial configuration is deployed, the KAMI component of QoS/MOS 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 service failure rates and request arrival rates. The new monitored 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.

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 adopting the Bayesian estimator described in [36]. Figure 13 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 re-compute 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 AlarmService2, which starts from the initial value (i.e., \(r_1^2 = 0.04\)) and gradually converges to the \(r_1^2 = 0.05\).

Notice that a value of the probability equal to 0.045 is the threshold above which requirement R_3 is violated, and that this new value for \(r_1^2\) is estimated after only 9 data points collected through monitoring the service behaviour. Accordingly, the DTMC component of the operational model will reflect this change in the service behaviour after the collection of 9 data points, and the autonomic manager will dynamically modify the system configuration so that AlarmService1 is used as part of the TA workflow.

Likewise, monitoring the in-house concrete services of

<table>
<thead>
<tr>
<th>Abstract Service</th>
<th>Index (i)</th>
<th>Mapping Pattern (mp_i)</th>
<th>Concrete service(s)</th>
<th>In-house service CPU (cpu^j)</th>
<th>Aggregate Failure Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alarm Service</td>
<td>1</td>
<td>SGL</td>
<td>s_i^1</td>
<td></td>
<td>0.04</td>
</tr>
<tr>
<td>Medical Analysis</td>
<td>2</td>
<td>SEQ</td>
<td>(s_j^1, s_j^2)</td>
<td></td>
<td>0.000015</td>
</tr>
<tr>
<td>Drug Service</td>
<td>3</td>
<td>SGL</td>
<td>s_i^3</td>
<td>cpu^j = 0.395</td>
<td>0.0012</td>
</tr>
</tbody>
</table>

**TABLE 6**

Initial Configuration for the TA system

![Fig. 12. Evaluation of R_3 with different failure rates of the Alarm Service](http://mc.manuscriptcentral.com/tse-cs)
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 changed from $\mu_1 = 0.15$ to $\mu_1 = 0.27$, 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 14. A probability of dropping requests less than 0.001 is now obtained for cpu3 ≥ 0.581, so in the planning stage of the MAPE loop, the value cpu3 = 0.581 will be selected.

Finally, note that this section described only one possible option for modeling the TA 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.

4.2.4 QoSMOS effectiveness and overheads

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 0.002–0.003 seconds on a 2.4 GHz Intel Core 2 Duo server with 4 GB of DDR3 RAM at 1067 MHz. As a result, 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 well under one second (most of which is spent in the communication steps between the QoSMOS components). Furthermore, notice that this small 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 adaptive capabilities 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.

While it is true that these encouraging results were obtained for a service-based system comprising only three abstract services and nine associated concrete services, our study of the widely used SBS development platform Taverna [52], [76] indicates that these numbers are not atypical for today’s service-based systems. Indeed, numerous Taverna workflows consisting of similar or only slightly larger numbers of services have been successfully used by researchers from application domains as
diverse as bioinformatics, chemistry, social sciences and arts.

Nevertheless, the successful adoption of service-based systems in recent years is certain to lead to an increase in the size and complexity of SBS workflows in the future. Undeniably, this is going to increase the overheads associated with the comprehensive quantitative analysis performed within the QoSMOS MAPE loop significantly. There are several options that we are investigating in our effort to ensure that QoSMOS will be capable to support this scenario, 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.

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 based 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 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.

**References**

Adnan Aziz, Vigyan Singhal, and Felice Balarin. It usually works:
Radu Calinescu and Marta Z. Kwiatkowska. Using quantitative
Allan Clark, Stephen Gilmore, Jane Hillston, and Mirco Tribas-
Stefano Gallotti, Carlo Ghezzi, Raffaela Mirandola, and Giordano
Regina Frei, Giovanna Di Marzo Serugendo, and José Barata.
Gerardo Canfora, Massimiliano Di Penta, Raffaele Esposito, and
Luciano Baresi, Domenico Bianculli, Carlo Ghezzi, Sam Guinea,
and Paola Spolletini. A timed extension of WSCOL. In IEEE

Luciano Baresi, Domenico Bianculli, Carlo Ghezzi, Sam Guinea,
and Paola Spolletini. Validation of web service compositions. IET

Luciano Baresi and Sam Guinea. Towards dynamic monitoring of
WS-BPEL processes. Proceedings of the 3rd International Conference
on Service Oriented Computing, 2005.

Luciano Baresi, Elisabetta Di Nitto, and Carlo Ghezzi. Toward
open-world software: Issue and challenges. IEEE Computer,

Rainer Berbner, Michael Spahn, Nicolas Repp, Oliver Heckmann,
Svend Frolund and Jari Koistinen. Quality-of-service specification
and analysis to implement autonomic IT systems. In Marsha Chechik and

Jonckheere, Viviane Jonckers, Filip De Turck, Chris Develder,
and Bart Dhoedt. SALSA: QoS-aware load balancing for au-
tonomous service brokering. Journal of Systems and Software,
available online in print, 2010.

Radu Calinescu. General-purpose autonomic computing. In
Mieso K. Denko, Laurence Tia-Nuoy Yang, and Yan Zhang, editors,

Radu Calinescu. Reconfigurable service-oriented architecture for
autonomic computing. International Journal On Advances in

Radu Calinescu and Marta Kwiatkowska. CADS*: Computer-
aided development of self-* systems. In Marsha Chechik and
Martin Wirsing, editors, Fundamental Approaches to Software En-
gineering (FASE 2009), volume 5503 of Lecture Notes in Computer

Radu Calinescu and Marta Z. Kwiatkowska. Using quantitative
analysis to implement autonomic IT systems. In Proceedings of
the 31st International Conference on Software Engineering, ICSE 2009,

Gerardo Canfora, Massimiliano Di Penta, Raffaele Esposito, and
Maria Luisa Villani. An approach for “qos-aware” service com-

Gerardo Canfora, Massimiliano Di Penta, Raffaele Esposito, and
Maria Luisa Villani. Qos-aware replanning of composite web
services. In Proceedings of the IEEE International Conference on

Gerardo Canfora, Massimiliano Di Penta, Raffaele Esposito, and
Maria Luisa Villani. A framework for qos-aware binding and re-
binding of composite web services. Journal of Systems and Software,

Lei Cao, Jian Cao, and Minglu Li. Genetic algorithm utilized in
cost-reduction driven web service selection. In Yue Hao, Jiming
Liu, and Jiaojiao Yang, editors, Proceedings of the International Conference on the International Conference on Computational Intelligence and Security,

Valeria Cardellini, Emiliano Casalicchio, Vincenzo Grassi,
Francesco Lo Presti, and Raffaella Miranda. Qos-driven runtime

Valeria Cardellini, Emiliano Casalicchio, Vincenzo Grassi, and
Raffaella Miranda. A framework for optimal service selection in

Valeria Cardellini, Emiliano Casalicchio, Vincenzo Grassi, and
Francesco Lo Presti. Scalable service selection for web service
E Produzione, Universita di Roma Tor Vergata, 2007.

Girish Chafle, Prashant Doshi, John Harney, Sumit Mittal, and
Biplav Srivastava. Improved adaptation of web service composi-
tions using value of changed information. In ICWS, pages 784–


Frank Ciesinski and Marcus Großer. On probabilistic computation
logic. In Christel Baier, Boudewijn R. Haverkort, Holger Her-

Allan Clark, Stephen Gilmore, Jane Hillston, and Mirco Tribas-


Robert Colvin, Lars Grunse, and Kirsten Winter. Timed behavior


Elia Zanfani, Carlo Ghezzi, Raffaella Miranda, and Giordano Tamburrelli. Model evolution by run-time parameter adapta-


Stefano Gallotti, Carlo Ghezzi, Raffaella Miranda, and Giordano
Tamburrelli. Quality prediction of service compositions through
probabilistic model checking. In Steffen Becker, Franktisek Plasil,

Ronald Garcia, Jaakko Jarvi, Andrew Lumsdaine, Jeremy Siek,
and Srinivas Namjoshi. The PEGASE workbench: A tool to support a process algebra-based approach to performance
modelling. In Günter Haring and Gabriele Kotsis, editors, Pro-

Stefano Gallotti, Stephen Gilmore, and Vincenzo Grassi. The PEGASE workbench: A tool to support a process algebra-based approach to performance modelling. In Günter Haring and Gabriele Kotsis, editors, Pro-

Stefano Gallotti, Stephen Gilmore, and Vincenzo Grassi. The PEGASE workbench: A tool to support a process algebra-based approach to performance modelling. In Günter Haring and Gabriele Kotsis, editors, Pro-


