

Quality Prediction of Service Compositions through Probabilistic Model Checking

Stefano Gallotti, Carlo Ghezzi, Raffaella Mirandola, and Giordano Tamburrelli

Politecnico di Milano

DeepSE Group–Dipartimento di Elettronica e Informazione

Piazza Leonardo Da Vinci, 32 – 20133 Milano, Italy

{gallotti,ghezzi,mirandola,tamburrelli}@elet.polimi.it

Abstract. The problem of composing services to deliver integrated business solutions has been widely studied in the last years. Besides addressing functional requirements, services compositions should also provide agreed service levels. Our goal is to support model-based analysis of service compositions, with a focus on the assessment of non-functional quality attributes, namely performance and reliability. We propose a model-driven approach, which automatically transforms a design model of service composition into an analysis model, which then feeds a probabilistic model checker for quality prediction. To bring this approach to fruition, we developed a prototype tool called **ATOP**, and we demonstrate its use on a simple case study.

1 Introduction

Service-Oriented Architectures (SOAs) provide a new paradigm for the creation of business applications. This paradigm enforces decentralized developments and distributed systems compositions: new added-value services may be created by composing independently developed services. Web services are an increasingly important and practical instance of SOAs, supported by standards and by specific technology. Typically, services can be composed in an *orchestrated* manner by using a workflow language, like the Business Process Execution Language (BPEL) [4].

We argue that SOAs can benefit from the Model Driven Development (MDD) [6] paradigm. In essence, this means that models are built to support software engineers in reasoning at the software architecture level. As a satisfactory solution is built at the model level, transformation steps (possibly automated) derive the final, platform-specific implementation. In the case of SOAs, model-level reasoning should support the early QoS assessment of a service composition. The composition may be assessed at design time, before a concrete binding from the workflow to the externally invoked services is established. The assessment is thus performed on the abstract workflow. It is requested, however, that a specification of the external services in terms of their functional and non-functional attributes is available. The actual binding from the abstract workflow to concrete services may then be established dynamically at run time, provided that the selected

concrete services fulfill their specification. This may be enforced by a suitable QoS-driven binding mechanism.

The use of models extends beyond the initial development of an application. Models may be used to support evolution of the software architecture. They can also be useful to devise suitable reconfiguration strategies for the dynamic contexts where the application will be deployed. Once the application is running, model-based reasoning may be used to predict the impact of different reconfigurations in a changing context, driving in this way the reconfiguration process.

In this perspective, hereafter we tackle the following two issues: i) which kind of model is suitable for quality analysis of service-based applications; ii) how we can support the construction of such a model.

Concerning the first issue, we build on past work on architectural reasoning and analysis of quality aspects through model checking [10,11], and in particular on the probabilistic model checker PRISM [34], which was used for a preliminary assessment in [23]. This choice is motivated both by the encouraging results we achieved, which demonstrated the applicability of these techniques to a wide set of systems, and by the existence of tools implementing these techniques.

Concerning the second issue, we leverage on the aforementioned MDD paradigm, to transform a high-level description to executable code. To perform analysis of non-functional quality attributes at the model level, we propose a model transformation step that takes as input a "design-oriented" model of the software system (plus some additional information related to the non-functional attribute of interest) and generates an "analysis-oriented" model, that lends itself to the application of an analysis methodology [26]. Specifically, we provide an integrated framework that, starting from an high level description of the service composition, given in terms of activity diagrams, automatically derives stochastic models that can be solved using the different features of the PRISM model checker. The provided methodology and tool are called ATOP, which stands for *from Activity diagrams TO Prism models*. Besides, we try to overcome a weakness of PRISM. The ATOP tool, in fact, includes the possibility to perform some kind of parametric analysis that at present is not fully supported in PRISM.

This paper is organized as follows. Section 2 presents the basic concepts of probabilistic model checking and PRISM. Section 3 illustrates the proposed MDD approach, while Section 4 provides the details of the approach for early quality assessment of service compositions. Section 5 describes the tool implementing our methodology and Section 6 describes how the proposed approach can be applied to a case study. Section 7 briefly surveys related work and Section 8 presents the conclusions.

2 Background

In this section we shortly review the basic concepts of the probabilistic model checking approach and the PRISM tool.

Probabilistic Model Checking is an automatic procedure for establishing if a desired property holds in a probabilistic system model. Conventional model

checkers start from a description of a model and a specification (using a state-transition system and a formula in some temporal logic, respectively) and return a boolean value, indicating whether or not the model satisfies the specification. In the case of probabilistic model checking, the models are probabilistic (typically, variants of Markov chains) and they add a probability to the transitions between states. In this way it is possible to calculate the likelihood of the occurrence of certain events during the execution of the system. This, in turn, allows quantitative analysis about the system, in addition to the qualitative statements made by conventional model checking. Probabilities are modeled via probabilistic operators that extend conventional (timed or untimed) temporal logic.

Probabilistic modeling is widely used in the field of performance evaluation; for example several algorithmic techniques and tools exist for Markovian models [12]. However, the key point of probabilistic model checking is the ability to combine probabilistic analysis and conventional model checking in a single tool. The first extension of model checking algorithms to probabilistic systems was proposed in the 1980s. However, work on implementation and tools did not begin until recently, when the field of model checking matured [24,25]. Probabilistic model checking draws on conventional model checking, since it relies on reachability analysis of the underlying transition system, but must also entail the calculation of the actual likelihoods through appropriate numerical methods, such as those employed in performance analysis tools [24,25].

PRISM is the model checker we selected to verify our models. *PRISM* [34] is a probabilistic model checker developed at the University of Birmingham. *PRISM* is a tool for the design and analysis of systems that exhibit probabilistic behaviors. It supports three types of probabilistic models: Discrete-Time Markov Chains, Markov Decision Processes, and Continuous-Time Markov Chains ([12]). Models are specified in a simple, high-level modeling language, which is a variant of the Reactive Modules formalism of Alur and Henzinger [3]. Properties are described by the *PRISM* property specification language, which is based on the two probabilistic temporal logics, called Probabilistic Computation Tree Logic (PCTL) [22] and Continuous Stochastic Logic (CSL) [7].

3 The ATOP Methodology

As introduced in Section 1, our approach derives quality predictions for service compositions. Each simple service of the composition is considered as a black-box entity. The process involved in this quality prediction analyzes abstract representations of service compositions to derive models suitable for applying probabilistic model checking techniques. Software architects may exploit this prediction to evaluate and compare different alternatives at design-time.

In Figure 1 we show a UML Activity Diagram (AD) outlining the main steps involved in the application of our methodology, by highlighting also who is in charge of them. Our approach starts from the application workflow specifications

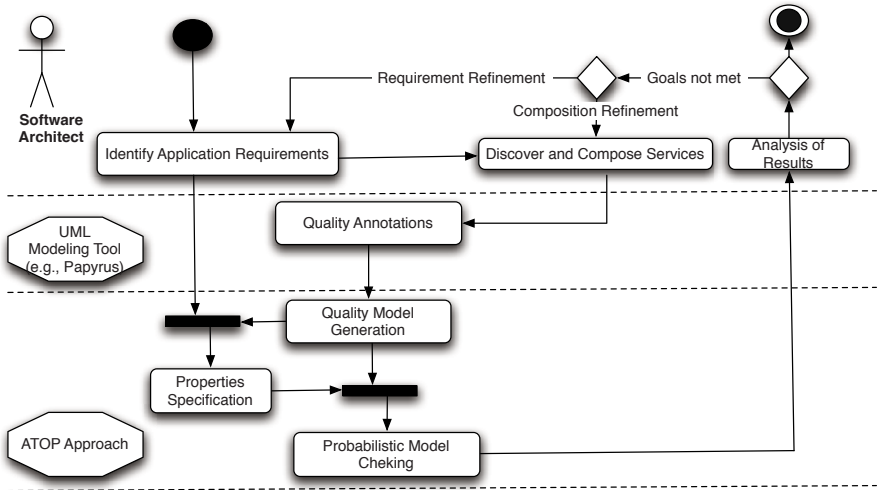


Fig. 1. The ATOP methodology

and derives quality predictions, such as application *Success probability* (an example of this quality prediction is illustrated in Section 6), through the following steps:

Identify application requirements. The software architect describes the application (s)he intends to realize and details its functional and non-functional requirements.

Discover and compose services. The software architect considers the services to compose only through their functional and non-functional annotations and builds the application through a service composition language. More precisely, our approach addresses the design phase of service compositions, which are represented through a UML AD [30]. At the implementation level, software architects exploit techniques like [14] for service discovery and workflow languages like BPEL [4] for service composition representation.

We assume that ADs representing service compositions are generated by using an ad-hoc tool, such as the UML *Papyrus* framework [32]. Our approach considers a subset of UML Activity Diagrams to represent sound service compositions. Supported diagrams are composed by only one *InitialNode* and only one *FinalNode*. Between these two elements there can be a sequence of the following elements: (i) activity, (ii) conditional block, and (iii) concurrent blocks, connected by means of arrows specifying the flow of execution.

Activity models invocation of a service. A conditional block is defined with a *DecisionNode* and a *MergeNode*. The conditional block appears in two different configurations, *If* or *Loop*, depending on the composition topology. Each branch of the decision block can contain a sequence of activities, decision

blocks and concurrent blocks. Concurrent blocks are defined by means of *ForkNodes* and *JoinNodes*. Each outgoing branch can contain the same aforementioned sequence of elements.

Quality annotations. Activity diagrams describing the composition are enriched by exploiting UML extensions. Through a UML profile, every ActivityNode is annotated with quality attributes of the selected service. Output arrows from a DecisionNode are annotated with the probability to follow each branch. Our approach exploits a subset of MARTE [31], a UML profile designed for specification of non-functional requirements of software systems and available in the Papyrus framework. We use the support of MARTE to represent a restricted subset of information; the main considered annotations are:

- *Service Reliability*: associated with an ActivityNode. It is a real number between 0 and 1 that represents the reliability of a single service invocation;
- *Service Execution Time*: associated with an ActivityNode. It represents the expected execution time of a service invocation.
- *Service Invocations Attempts*: associated with an ActivityNode. It represents the number of failed invocations necessary to declare a service to be faulty.
- *DecisionNode Output probabilities*: associated with output branches of a DecisionNode: they represent the probability to follow a given branch.
- *Service Degradation Function*: associated with an ActivityNode. It is a domain-related law specifying dependency of service values from the execution context, e.g. size of input parameters. Due to its nature, this law can be inferred from observations or can be obtained from domain experts.

The Service Degradation Function is a peculiarity of the ATOP methodology. In particular, it supports parametric analysis. Indeed, this annotation describes the relation between a service composition input parameter and the quality properties of the basic services involved in the composition. For example, if a degradation function related to a parameter (e.g., input size) is specified for a service, during the model generation step all the values expressed by the other annotations in the service composition are updated by evaluating this degradation function. This mechanism is necessary to express the fact that reliability, execution time, invocation attempts, and branch probabilities are often dependent on specific service composition input parameters. Figure 2 illustrates two examples of non-functional annotations on ADs.

Quality models generation. A service composition, where model and annotations are described as presented before, is automatically translated into a quality model by the ATOP tool. The target quality model must be chosen according to the characteristics of the model and to the properties to be verified via the probabilistic model checker. ATOP considers the following Markovian models: (1) Discrete Time Markov Chains (DTMC),

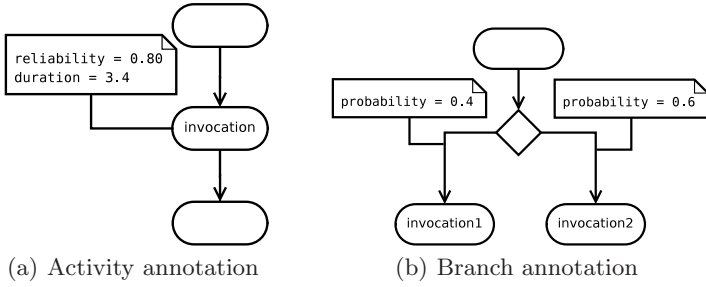


Fig. 2. Annotated Activity Diagrams

(2) Markov Decision Processes (MDP), and (3) Continuous Time Markov Chains (CTMC). A detailed description of this step is given in Section 4.

Properties specification. The set of properties to be verified on the model should be specified according to overall quality requirements. These properties are formulated through logic formulas expressed as CTL logic extensions.

Probabilistic Model Checking. The automatic modeling step generates a model, which is given as input to the probabilistic model checker. The model checker analyzes the received model with respect to the properties specified by the user.

Analysis of Results. The software architect analyzes the output produced by the probabilistic model checker to verify if the service composition matches the quality goals required by the application domain. If these goals are met, the development process continues to produce an implementation; otherwise, alternative compositions are evaluated in order to reach the required goals. A detailed example of the possible analysis is illustrated in Section 6.

4 Quality Modeling of Service Compositions

In this section we present the details of the quality model generation step of the ATOP methodology (illustrated in Figure 3). To this end, we provide a short description of (i) the target transformation models, (ii) the properties specifications and (iii) the translation process.

4.1 Target Models

The translation process is based on the exploration of the AD, starting from the *InitialNode* until the *FinalNode*. Depending on the nature of the model and on the type of analysis to be performed, different Markovian models can be chosen as output of the translation process.

In our framework, the DTMC model is used to model simple service compositions without concurrent branches and without timing information associated

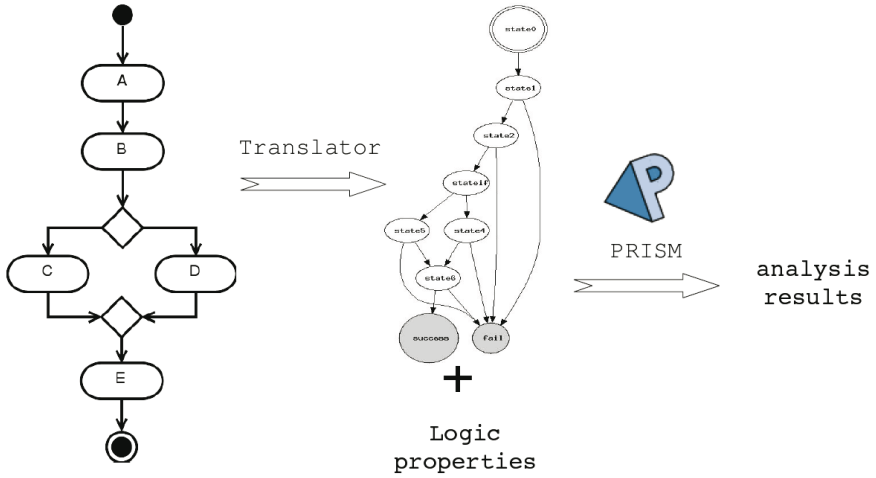


Fig. 3. Service composition analysis

with services. Should service compositions include concurrent sections (where service invocations are executed in parallel), it is necessary to model all the possible interleaved invocations. To this end, it is necessary to use a MDP model, which exploits non-determinism modelling all the paths the system could follow.

If the analysis focuses on the time necessary for the system to perform its functionalities, a CTMC model is instead required. By modeling the transition probability as an exponential distribution, each service invocation can be represented as a state whose transition parameter is related to the expected duration of the service execution. Using a parameter λ representing the rate of the exponential distribution and defining it as $1/\text{expected_duration}$ the model approximates the real temporal behavior of the system, giving a time-depending probabilistic result. The system is characterized by an initial transient phase and finally probability values asymptotically stabilize.

4.2 Properties Specifications

We analyze the model by verifying properties specified in temporal logic and evaluated through model checking. Basic properties on a service composition can be the reliability value of the whole complex system (e.g., the probability that starting from the initial state the system eventually reaches the success state), specified in PCTL as

$$P[F(\text{system_state} = \text{success})]$$

where Ff (eventually operator) represents the short form of $true \ U \ f$ (U is the “until” temporal operator).

Similar properties can be evaluated starting from each state of the system

$$\text{system_state} = \text{”a certain service invocation”} \Rightarrow P[F(\text{system_state} = \text{success})]$$

The evaluation of these properties support the discovery of configurations that can be critical for the system. Properties can also be specified to obtain a boolean result. Indeed, we can also express properties like

$$P_{\geq threshold}[F(system_state = success)]$$

whose evaluation yields a boolean value (true if the probability result complies with the threshold bound). Depending on the desired analysis, different logic properties can be formulated over the model and then submitted to the model checker.

4.3 Translation Elements

The translation process from ADs to Markov models is based on the exploration of the original model, starting from the *InitialNode* along the execution path defined by the control flow. In the following we describe how the translation of the main AD elements is performed. The tool implementing the translation is described in Section 5. The *Initial* and *Final* nodes of the AD correspond to the initial and final states of the Markov model.

The *Activity* is the core element of an AD that describes a service invocation and can contain additional information through annotations. An Activity is translated into a node with two outgoing transitions: the success transition and the failure transition. The probabilities associated with the two transitions depend on the annotations of the original diagram; the destination states can be the next state, in case of success, and a retry or a fail state, in case of failure.

Decision and *Merge* nodes in ADs decision blocks can assume the *If* and *Loop* configurations, depending on the topology of the components. In the former case, the translation process creates in the Markovian model a new node representing the *If* node and as many transitions as there are outgoing branches. The exploration continues for each branch over the path, until the MergeNode is reached. Then the exploration resumes from the node following the MergeNode. In the latter case (*Loop*), the exploration is performed over the loopback arc until the MergeNode is reached. Then the exploration continues over the remaining branch exiting the *Loop* node.

Fork and *Join* nodes define a concurrent block. The translation requires the exploration of all the branches exiting the ForkNode, until a JoinNode is reached. Each branch is modeled by an independent sub-diagram. The exploration resumes starting from the node following the JoinNode.

Additional information on the model define non-functional properties. They are used to drive the creation of the probabilistic model, as described hereafter:

- Service Reliability: the value α of reliability is used as the probability of the success transition. The related fail transition has the probability $1 - \alpha$.
- Service Execution Time: in a CTMC model, the rate of the exponential distribution λ is defined as $1/expected_duration$
- Service Invocations Attempts. The service can be invoked a given numbers of time before being declared as failed. Therefore, the fail state is reached only when the last attempt is performed and fails.

The translation process is realized in the ATOP tool described in the next section.

5 The ATOP Tool

As described in Section 3, the ATOP approach takes as input a formal representation of the service composition drawn as a UML AD extended with quality annotations. This representation is exported in the XMI format, elaborated by the ATOP tool and translated into a PRISM model. Note that, although the XMI format is near to be a standard, each tool representation differs for small details, which implies that a single interpreter is not valid in each case. The translator tool builds a graph-based representation obtained by means of a tool-specific interpreter. In this way the translator can be easily extended to support new design tool just adding an extension tailored on the new XMI format. The translation process is based on the recursive exploration of the graph, performing the operations described in Section 4 to generate an output model on the basis of the information provided by the AD. This tool can be executed directly from the command line or through a graphical user interface. The latter offers an aid to select the input and output format options and shows translation results.

In several contexts, service quality needs to be evaluated depending on some execution parameters. At present the PRISM model checker does not offer full support for parametric analysis of the model; the ATOP tool overcomes such limitation generating models whose values are set depending on given parameters. Using ATOP it is possible to specify the execution context parameters (e.g., input size) based on which, if a degradation law is defined for services components, the non-functional values are adjusted. In this way the system behaviour can be automatically analyzed in different contexts. The model is then evaluated by means of automatic analysis techniques, via PRISM.

6 Case Study

In this section, we illustrate our approach through a case study of a service composition. The system resulting from this composition offers a travel management functionality. Starting from travel location, it offers booking services and notifications. The high level composition specification is shown in Figure 4 through an extended AD. The workflow initially performs three service invocations, to identify the requirements of the travel. Then two concurrent task are executed: a parking booking and a process of notification to the user. After execution of both, the service performs a meeting arrangement and subsequently notifies the commitment for the travel. The activities in the diagram represent service invocations, annotated with a reliability value and an execution time expressed in seconds. This information is obtained from the service providers. The diagram offers a view in which single elements are organized to offer a more complex system. The concrete implementation is a BPEL xml file (although any other equivalent orchestration language could be adopted).

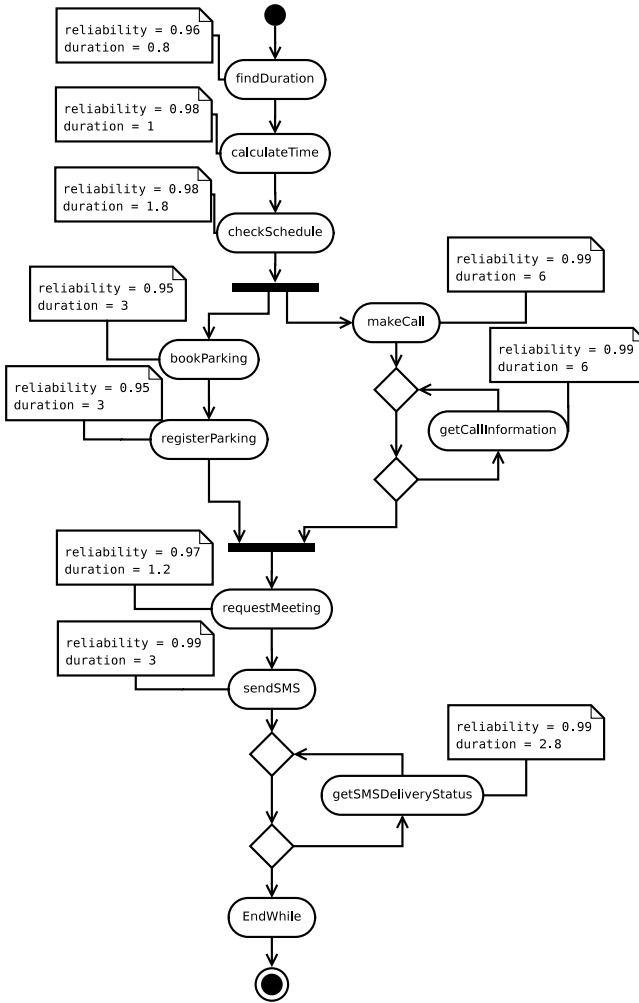


Fig. 4. Travel Management Service Activity

The AD represented by an XMI file can be translated in three different Markovian models, as discussed in Section 4. The first model (DTMC) does not take into account time and cannot model concurrency. The concurrent branches in the AD of Figure 4 are automatically condensed into a single activity¹. DTMC supports the evaluation of the expected reliability value concerning the whole service composition or the expected reliability value concerning part of it (i.e., starting from an inner state, different from the initial state). The property associated with the global reliability of the system is expressed in PCTL as:

¹ This step is carried out by PRISM generating, with equal probabilities, all the possible paths composed by interleaved activities. The paths are then evaluated and the results of the analysis are aggregated in a single node.

$$P[F(\text{system_state} = \text{success})].$$

The probabilistic model checker returns a probability of 0.775, which represents the reliability of the service composition. The same property can be locally evaluated starting from any specific inner state. For example, the table below shows the reliability values obtained starting from two inner states.

state	PCTL formula	result
checkSchedule	$state = \text{checkSchedule} \Rightarrow P[F(\text{system_state} = \text{success})]$	0.824
requestMeeting	$state = \text{requestMeeting} \Rightarrow P[F(\text{system_state} = \text{success})]$	0.950

The second kind of model (MDP), supports the analysis of concurrent executions. More precisely, it is possible to compute the reliability values associated with internal states of concurrent areas of the diagram. In presence of concurrent branches the order in which elements of different branches are interleaved cannot be predicted. By using MDP, it is possible to compute reliability values associated with internal states of concurrent sections of the diagram, which would otherwise be impossible using DTMC. Considering all the possible non deterministic evolutions of the system, the model checker can return the upper and lower bounds of reliability. The table below reports the maximum and minimum reliability values obtained from a concurrent state of the composition.

state	PCTL formula	result
makeCall	$state = \text{makeCall} \Rightarrow Pmax[F(\text{system_state} = \text{success})]$	0.932
makeCall	$state = \text{makeCall} \Rightarrow Pmin[F(\text{system_state} = \text{success})]$	0.841

The third type of model generated is CTMC, which focuses on the time associated with every service invocation. This analysis shows how the reliability of the system changes with respect to the time. This result approximately indicates a time bound for global service execution. The composition illustrated in this case study contains a concurrent block and, consequently, we cannot directly adopt a CTMC model. The analysis is performed by substituting the concurrent block with a synthesis node containing results of probabilistic model checking related to each concurrent branch. The properties checked are similar to the properties obtained with a DTMC model, but with an indication of the time dependency.

Figure 5 shows the probability value of reaching the success state within time t , computed for t ranging in an interval $[0, 70]$ (seconds). This represents how the probability of success evolves over time after the invocation of the composed service. This value tends in the long run to the value obtained with the DTMC model (0.775), the reliability value of the service.

The CSL specification of this property is

$$P[\text{true } U^{[0,t]} \text{system_state} = \text{success}]$$

and its evaluation is obtained by varying parameter t that represents the time. This example does not model any recovery behavior, each service is invoked just once. The reliability of the whole system could be improved, by increasing the expected execution time, modeling for each service the number of retries for failure invocations.

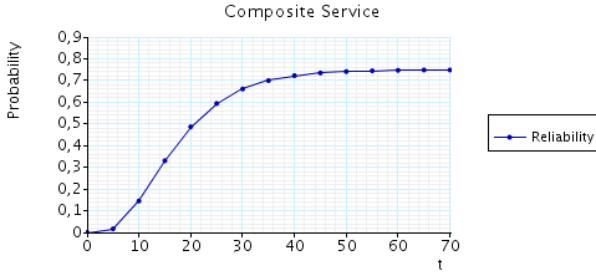


Fig. 5. Success probability evolution

7 Related Work

In the last years, Quality of Service (QoS) has been extensively studied in the context of *traditional software systems*. In particular, there has been a great interest in model transformation methodologies for the generation of analysis-oriented target models (including performance and reliability models) starting from design-oriented source models, possibly augmented with suitable annotations. In particular, several proposals have been presented concerning the direct generation of performance analysis models. Each of these proposals focuses on a particular type of source design-oriented model and a particular type of target analysis-oriented model, with the former spanning UML, Message Sequence Chart, Use Case Maps, formal language as AEmilia, ADL languages as Acme, and the latter spanning Petri nets, queueing networks, layered queueing network, stochastic process algebras, Markov processes (see [8] for a thorough overview of these proposals and [1] for recent proposals on this topic). Some proposals have also been presented for the generation of reliability models. All the proposals we are aware of start from UML models with proper annotations, and generate reliability models such as fault trees, Markov processes and Bayesian models (see [21] for more details). Moreover, some attempts have been made in the literature [2,19,20] to translate UML specifications into models to be solved by probabilistic model checkers. Specifically, in [2] it is proposed a methodology that translates UML statecharts (with annotations for real-time systems) into probabilistic timed automata that are then solved using PRISM. Gilmore et al. in [19,20] propose a method translating UML statecharts into stochastic process algebra (PEPA) models that are then solved using PRISM.

Recently, QoS issues in service selection and composition have obtained great interest in the *service research community*. Different approaches have been followed so far, spanning the use of QoS ontologies [28], the definition of ad-hoc methods in QoS-aware frameworks [33,38], and the application of optimization algorithms [5,13,39].

One of the first works in this area is proposed in [29] where a framework for composed services modeling and QoS evaluation is presented. A composite service is modeled as a directed weighted graph where each node corresponds to a Web Service (WS) and edge weights represent the transition probabilities of

two subsequent tasks. The author shows how to evaluate quality of service of a composed service from basic services characteristics and graph topology.

Some recent proposals face the problem of composition of WSs by implementing genetic algorithms. In Canfora et al. [13] the reduction formulas presented in [16] are adopted, and the problem is also periodically re-optimized in order to take into account WS performance variability. However, only sub-optimal solutions are identified since WSs specified inside execution loops are always assigned to the same Web service implementation. The paper [17] proposes a mechanism that implements an optimizing WS composition combining performance optimization, price optimization, and payload optimization when meeting the requirements of Service-level Agreement (SLA). In [35] the authors propose both an evaluation approach for QoS attributes of WS, which is completely service and provider independent, and a method to analyze WS interactions and extract important QoS information without any knowledge about the service implementation. In [37] the WS composition from a performance viewpoint is studied and measured. These measurements demonstrate that WS composition may reduce the maximal load of a system drastically (i.e., quasi-exponentially with the number of service compositions). In order to mitigate this performance reduction, the author proposes an optimized service composition architecture as a solution.

The works closest to ours concern methods to derive performance related measures of workflow processes [15,27,36]. Cardoso [15] proposes two different metrics to evaluate the control-flow complexity of BPEL web processes before their actual implementation. In [36] a mathematical model based on operations research techniques is proposed to estimate the influence of the execution of orchestrated processes on utilization and throughput of the system. In [27] starting from annotated BPEL and WSDL specifications, performance bounds on response time and throughput are derived. In such a way users are able to assess the efficiency of a BPEL workflow, while service provider(s) can perform sizing studies or estimate performance gains of alternative upgrades to existing systems.

A related approach to verifying service-oriented architectures is described in [9]. This work deals with new added-value services obtained by composing existing services through workflows described in the BPEL language. The workflows are verified at design time via model checking. The properties against which designs are checked are then transformed into run-time monitored assertions to support run-time verification. Properties are expressed in a temporal logic language (ALBERT), which can express both functional and non-functional properties, such as expected response time. Properties are of two kinds: Assumed Assertions (AAs), which state the expected behavior of the external services invoked by the workflow, and Guaranteed Assertions (GAs), which state the properties the workflow is expected to ensure to its users. Model checking is used to prove that GAs are satisfied, assuming that AAs hold. The approach explored in this work, however, does not support any kind of probabilistic reasoning. It is based on the use of the Bogor model checker [18].

With respect to existing work, our approach represents a first attempt at a design methodology (and supporting tools) through which QoS properties may be specified and formally analyzed for service compositions.

8 Conclusions

In this paper we have presented ATOP: a design methodology (and supporting tools) through which QoS properties may be specified and formally analyzed for service compositions. In particular, QoS reasoning is based on probabilistic modelling, which is crucial for performance and reliability prediction. Our approach is supported by a tool that translates a high-level design description into a form that is amenable to probabilistic model checking using PRISM. Our initial assessment of the approach has been quite encouraging, and a further development of case studies will be part of our future activities.

Our future work will also focus on systematizing the feedback loop from run-time observations of the performance attributes of a running composite service back to the design environment. If the running system is found to behave inconsistently with respect to the design model, it would be desirable to re-calibrate the design model with more accurate data drawn from run-time gathered information, and possibly derive an improved implementation. This of course requires that models should be kept alive at run time. It also requires ways to perform analyses at run time in an efficient and light-weight manner, possibly incrementally. To the best of our knowledge, this is still an unexplored research path.

Acknowledgments

This work has been partially supported by Project Q-ImPrESS (FP7-215013) funded under the European Union's Seventh Framework Programme (FP7).

References

1. Wosp : Proceedings of the international workshop on software and performance, (1998-2007)
2. Addouche, N., Antoine, C., Montmain, J.: Combining extended uml models and formal methods to analyze real-time systems. In: Winther, R., Gran, B.A., Dahll, G. (eds.) SAFECOMP 2005. LNCS, vol. 3688. pp. 24–36. Springer, Heidelberg (2005)
3. Alur, R., Henzinger, T.A.: Reactive modules. *Formal Methods in System Design: An International Journal* 15(1), 7–48 (1999)
4. Alves, A.: et al. Web service business process execution language version 2.0. Committee Draft, 17 (May 2006)
5. Ardagna, D., Pernici, B.: Global and Local QoS Guarantee in Web Service Selection. In: Proc. of Business Process Management Workshops, pp. 32–46 (2005)
6. Atkinson, C., Kuhne, T.: Model-driven development: A metamodeling foundation. *IEEE Software* 20(5), 36–41 (2003)

7. Aziz, A., Sanwal, K., Singhal, V., Brayton, R.K.: Verifying continuous time markov chains. In: Alur, R., Henzinger, T.A. (eds.) CAV 1996. LNCS, vol. 1102, pp. 269–276. Springer, Heidelberg (1996)
8. Balsamo, S., Di Marco, A., Inverardi, P., Simeoni, M.: Model-based performance prediction in software development: A survey. *IEEE Trans. Software Eng.* 30(5), 295–310 (2004)
9. Baresi, L., Bianculli, D., Ghezzi, C., Guinea, S., Spoletini, P.: Validation of web service compositions. *IET Software* 1(6), 219–232 (2007)
10. Baresi, L., Gerosa, G., Ghezzi, C., Mottola, L.: Playing with time in publish-subscribe using a domain-specific model checker. In: SAVCBS 2007: Proceedings of the 2007 conference on Specification and verification of component-based systems, pp. 55–62. ACM, New York (2007)
11. Baresi, L., Ghezzi, C., Mottola, L.: On accurate automatic verification of publish-subscribe architectures. In: ICSE 2007: Proceedings of the 29th International Conference on Software Engineering, Washington, DC, USA, pp. 199–208. IEEE Computer Society, Los Alamitos (2007)
12. Bolch, G., Greiner, S., de Meer, H., Trivedi, K.: *Queuing Network and Markov Chains*. John Wiley, Chichester (1998)
13. Canfora, G., Di Penta, M., Esposito, R., Villani, M.L.: An Approach for QoS-aware Service Composition Based on Genetic Algorithms. In: Proc. of Genetic and Computation Conf. Washington, DC, pp. 1069–1075 (June 2005)
14. Cardellini, V., Casalicchio, E., Grassi, V., Mirandola, R.: A framework for optimal service selection in broker-based architectures with multiple QoS classes. In: Services computing workshops, SCW 2006, pp. 105–112. IEEE Computer Society, Los Alamitos (2006)
15. Cardoso, J.: Complexity analysis of BPEL web processes. *Software Process: Improvement and Practice* 12(1), 35–49 (2007)
16. Cardoso, J., Sheth, A.P., Miller, J.A., Arnold, J., Kochut, K.: Quality of service for workflows and web service processes. *J. Web Sem.* 1(3), 281–308 (2004)
17. Dong, W.L., YU., H.: Optimizing web service composition based on qos negotiation. *EDOCW* 0, 46 (2006)
18. Dwyer, M.B., Hatcliff, J., Hoosier, M., Robby,: Building your own software model checker using the bogor extensible model checking framework. In: Etessami, K., Rajamani, S.K. (eds.) CAV 2005. LNCS, vol. 3576, pp. 148–152. Springer, Heidelberg (2005)
19. Gilmore, S., Haenel, V., Kloul, L., Maidl, M.: Choreographing security and performance analysis for web services. In: Bravetti, M., Kloul, L., Zavattaro, G. (eds.) EPEW/WS-EM 2005. LNCS, vol. 3670. pp. 200–214. Springer, Heidelberg (2005)
20. Gilmore, S., Kloul, L.: A unified tool for performance modelling and prediction. *Reliability Engineering and System Safety* 89, 17–32 (2005)
21. Grassi, V., Mirandola, R., Sabetta, A.: Filling the gap between design and performance/reliability models of component-based systems: A model-driven approach. *Journal of Systems and Software* 80(4), 528–558 (2007)
22. Hansson, H., Jonsson, B.: A logic for reasoning about time and reliability. *Formal Aspects of Computing* 6(5), 512–535 (1994)
23. He, F., Baresi, L., Ghezzi, C., Spoletini, P.: Formal analysis of publish-subscribe systems by probabilistic timed automata. In: Derrick, J., Vain, J. (eds.) FORTE 2007. LNCS, vol. 4574, pp. 247–262. Springer, Heidelberg (2007)
24. Kwiatkowska, M.: Quantitative verification: Models, techniques and tools. In: Proc. 6th joint meeting of the European Software Engineering Conference and the ACM

- SIGSOFT Symposium on the Foundations of Software Engineering (ESEC/FSE), pp. 449–458. ACM Press, New York (2007)
25. Kwiatkowska, M., Norman, G., Parker, D.: Stochastic model checking. In: Bernardo, M., Hillston, J. (eds.) SFM 2007. LNCS, vol. 4486, pp. 220–270. Springer, Heidelberg (2007)
 26. Di Marco, A., Mirandola, R.: Model transformation in software performance engineering. In: Hofmeister, C., Crnković, I., Reussner, R. (eds.) QoSA 2006. LNCS, vol. 4214, pp. 95–110. Springer, Heidelberg (2006)
 27. Marzolla, M., Mirandola, R.: Performance prediction of web service workflows. In: Overhage, S., Szyperski, C.A., Reussner, R., Stafford, J.A. (eds.) QoSA 2007. LNCS, vol. 4880. Springer, Heidelberg (2008)
 28. Maximilien, E.M., Singh, M.P.: A Framework and Ontology for Dynamic Web Services Selection. *IEEE Internet Computing* 8(5), 84–93 (2004)
 29. Menasce, D.A.: QoS Issues in Web Services. *IEEE Internet Computing* 6(6), 72–75 (2002)
 30. Object Management Group. UML 2.0 superstructure specification (2002)
 31. Object Management Group OMG. UML Profile for Modeling and Analysis of Real-Time and Embedded Systems. ptc/07-08-04 (2007)
 32. Papyrus UML, <http://www.papyrusuml.org/>
 33. Patel, C., Supekar, K., Lee, Y.: A QoS Oriented Framework for Adaptive Management of Web Service Based Workflows. In: Mařík, V., Štěpánková, O., Retschitzegger, W. (eds.) DEXA 2003. LNCS, vol. 2736, pp. 826–835. Springer, Heidelberg (2003)
 34. PRISM, Probabilistic Model Checker, <http://www.prismmodelchecker.org/>
 35. Rosenberg, F., Platzner, C., Dustdar, S.: Bootstrapping performance and dependability attributes of web services. *ICWS 0*, 205–212 (2006)
 36. Rud, D., Schmietendorf, A., Dumke, R.: Performance modeling of WS-BPEL-based web service compositions. *SCW 0*, 140–147 (2006)
 37. Schmid, H.A.: Service congestion: The problem, and an optimized service composition architecture as a solution. *ICWS 0*, 505–514 (2006)
 38. Yu, T., Lin, K.J.: A Broker-Based Framework for QoS-Aware Web Service Composition. In: Proc. of 2005 IEEE Int'l Conf. on e-Technology, e-Commerce and e-Service (March 2005)
 39. Zeng, L., Benatallah, B., Ngu, A.H.H., Dumas, M., Kalagnanam, J., Chang, H.: QoS-Aware Middleware for Web Services Composition. *IEEE Trans. Softw. Eng.* 30(5), 311–327 (2004)