Developing Self-Verifying Service-Based Systems

Radu Calinescu, Kenneth Johnson and Yasmin Rafiq
Department of Computer Science, University of York, UK
Email: {radu.calinescu, kenneth.johnson, yr534}@york.ac.uk

***Abstract*—We present a tool-supported framework for the engineering of service-based systems (SBSs) capable of self-verifying their compliance with developer-specified reliability requirements. These self-verifying systems select their services dynamically by using a combination of *continual quantitative verification* and *online updating* of the verified models. Our framework enables the practical exploitation of recent theoretical advances in the development of self-adaptive SBSs through (a) automating the generation of the software components responsible for model updating, continual verification and service selection; and (b) employing standard SBS development processes.**

I. Introduction

Assembling software systems through the integration of third-party services deployed on remote cloud data-centres and accessed over the Internet can reduce the time, cost and expertise required to develop new applications considerably. There is, however, a downside to the approach: remote third-party services tend to vary in reliability and performance over time. Even selecting services from the most reputable providers is not a guarantee that the resulting *service-based system* (SBS) will comply with its requirements at all times.

Addressing this limitation by replacing underperforming services with functionally equivalent ones "on the fly" has preoccupied the research community for over a decade (e.g., [1, 2, 3, 4]). However, the theoretical results produced by this research have not led to the much-needed improvements in the SBS engineering practice. The tool-supported COntinual VErification (COVE) framework introduced in our paper and freely available from http://www-users.cs.york.ac.uk/~ken/COVE aims to reduce this gap between state-of-the-art research and the current state of practice by automating the exploitation of recent theoretical advances in the engineering of *self-adaptive SBSs*. COVE has several major advantages over the existing approaches to developing self-adaptive SBSs:

1) COVE self-adaptive SBSs are *self-verifying*, i.e., they employ continual formal verification to select the service combination that guarantees the realisation of the SBS reliability requirements with minimal cost. This verification uses an embedded version of the quantitative model checker PRISM [5] to analyse a discrete-time Markov chain (DTMC) model of the SBS workflow.
2) The DTMC model used by COVE is updated online to reflect changes in the service reliability and in the frequency with which the SBS operations are invoked.
3) The continual verification, model updating and service selection capabilities of COVE are fully automated, and are provided by a combination of reusable and automatically generated software components.

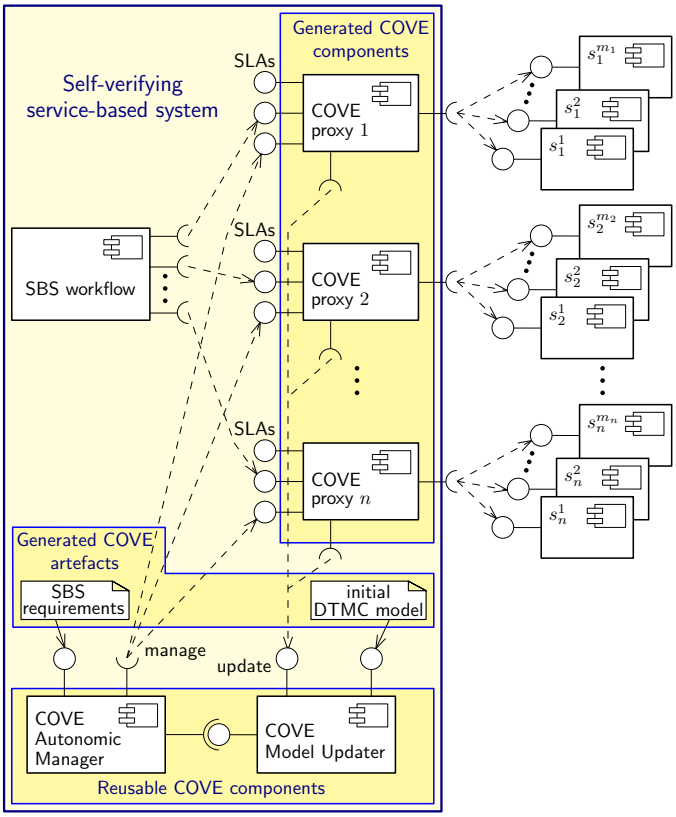

Fig. 1. Architecture of a COVE self-verifying SBS

4) The tool-supported COVE development process resembles the traditional SBS development process, so practitioners can use the framework with little learning effort.

The rest of the paper is organised as follows. Sections II describes the architecture of a COVE self-verifying SBS. Section III introduces a telehealth service-based system that is used to illustrate the application of the COVE development approach, which is presented in Section IV. The effectiveness of the framework is evaluated in Section V, and related work is discussed in Section VI. Section VII presents our conclusion.

II. Self-Verifying SBS Architecture

Fig. 1 shows the architecture of a COVE self-verifying SBS comprising $n \geq 1$ operations performed by remote third-party services. The $n > 0$ COVE service proxies in this architecture interface the SBS workflow with sets of remote services such that the i-th SBS operation can be carried out by $m_i \geq 1$ functionally equivalent services. The runtime selection of the service for each SBS operation and, as in the case of traditional

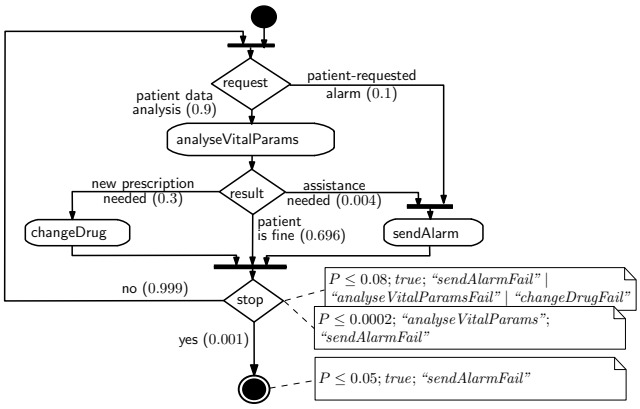

Fig. 2. UML activity diagram of the telehealth SBS. Estimate *a priori* probabilities are associated with the outgoing edges of decision nodes, and comments defining the SBS requirements are associated with relevant nodes.

web service proxies, the interactions with the selected services are handled automatically by the COVE proxies. When an instance of the i-th proxy is created, it is initialised with a sequence of (promised) *service level agreements* (SLAs) $sla_{ij} = (p^0_{ij}, c_{ij})$, $1 \leq j \leq m_i$, where $p^0_{ij} \in [0, 1]$ and $c_{ij} > 0$ represent the provider-supplied probability of success and the cost for an invocation of service s_{ij}, respectively.

The n proxies are also responsible for notifying a *model updater* about each service invocation and its outcome. The COVE model updater starts from an *initial DTMC model* of the SBS workflow, and uses a sophisticated online Bayesian learning technique to adjust the model parameters in line with these proxy notifications. The updated SBS model is then used by an *autonomic manager* that controls the services selected by the n proxies, to ensure that the service combination which satisfies the *SBS requirements* with minimal cost is selected at all times. Accordingly, the COVE proxies, model updater and autonomic manager implement a monitor-analyse-plan-execute (MAPE) autonomic computing loop [6]. Due to space limitations, the theoretical results underpinning the continual verification and online learning employed by the COVE MAPE loop could not be presented in this paper; for a detailed description of these results, please see [7].

A key advantage of our COVE framework is that its model updater and autonomic manager components are SBS-independent and therefore reusable across applications, while the SBS-specific proxies, requirements and initial DTMC model are generated automatically using the COVE software engineering tools described in Section IV.

III. Running Example

We will use a telehealth service-based system taken from [2, 8, 9] as a running example. In this SBS, the vital parameters of a patient are periodically measured by a wearable device and analysed by third-party medical services. The result of the analysis may trigger the invocation of an alarm service (that determines, for instance, the dispatch of an ambulance), may lead to the invocation of a pharmacy service to deliver new medication to the patient, or may confirm that the patient is fine. In addition, the patient can initiate an alarm by using a panic button on the wearable device. The workflow of the telehealth SBS is shown in Fig. 2, and we will consider that it must comply with three reliability requirements:

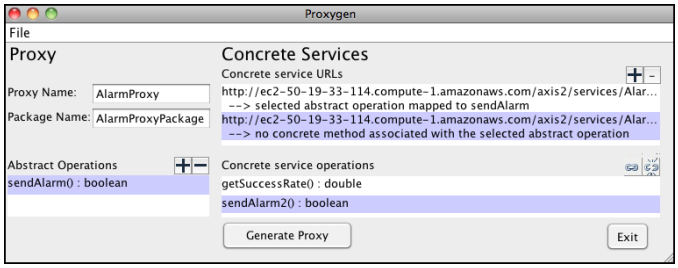

Fig. 3. Proxy generation for the sendAlarm operation from Fig. 2.

- R_1: The probability that one execution of the workflow ends in a service failure is at most $p_{R_1} = 0.08$.
- R_2: The probability that an alarm failure occurs within $N = 10$ executions of the workflow is at most $p_{R_2} = 0.05$.
- R_3: The probability that an invocation of the analysis service is followed by an alarm failure is at most $p_{R_3} = 0.0002$.

IV. COVE Development Process

The tool-supported COVE development process comprises three stages, each of which is described in detail in this section.

A. Stage 1: Proxy Generation

In this stage, the COVE service proxies in Fig. 1 are generated for the n SBS operations. For the i-th operation, $1 \leq i \leq n$, the developer first selects $m_i \geq 1$ functionally equivalent services that implement the operation, but which may be associated with different levels of reliability and different costs. Once the m_i candidate services have been selected, the developer uses the COVE proxy generator tool to produce a (Java package) proxy for the i-th SBS operation. The functionality of the COVE proxy generator resembles that of standard web service proxy generators such as `WSDL2Java` and `wsdl2php`, except that: (a) the COVE proxy is synthesised from m_i web service WSDL definitions instead of a single one; and (b) the developer needs to specify the functionally equivalent methods of the m_i web services in a preliminary, GUI-supported step of the generation process.

Example 1. Fig. 3 depicts the generation of a COVE proxy for the sendAlarm operation of the telehealth SBS from Fig. 2. Two "concrete" services deployed on Amazon EC2 virtual machines were selected as candidates for executing the "abstract" SBS operation, which is mapped to the sendAlarm method of the first service and to the sendAlarm2 method of the second service. In the general case, a single COVE proxy could map each of several SBS operations to different methods belonging to a subset of the concrete services it relies upon.

B. Stage 2: Initial Model and Requirements Generation

The second stage of the development process involves (a) the construction of the initial DTMC model used to set up the COVE model updater; and (b) the formalisation of the SBS requirements used to configure the COVE autonomic manager. These operations are performed by the COVE *model generator and requirement formalisation tool*, which is implemented as

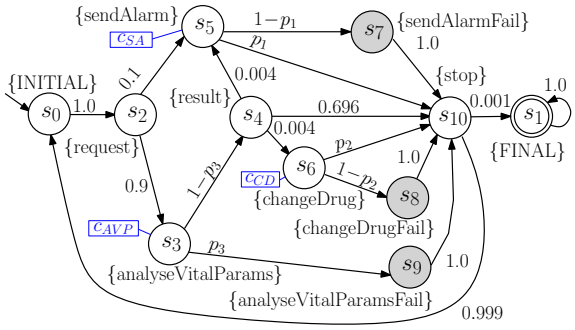

Fig. 4. Initial DTMC model for the SBS workflow from Fig. 2

a Java command-line application. This tool takes as input an annotated SBS activity diagram in the XMI format generated by the Eclipse-based Papyrus graphical editing tool for UML 2 (http://www.eclipse.org/papyrus/), and produces:

- A cost-annotated DTMC model $M = (S, s_0, P, L, c)$ of the SBS, where S is a finite set of states; $s_0 \in S$ is the initial state; P is an $|S| \times |S|$ transition probability matrix; $L : S \to 2^{AP}$ is a labelling function that assigns a set of atomic propositions from AP to each state in S; and $c : S \to \mathbb{R}_+$ is a costing function that associates a cost $c(s) \geq 0$ to each state $s \in S$. For any states $s_i, s_j \in S$, the element p_{ij} from P represents the probability of transitioning to state s_j from state s_i, and $\sum_{s_j \in S} p_{ij} = 1$.
- Probabilistic computational tree logic (PCTL) [10] formalisations of the SBS requirements.

These two SBS artefacts from Fig. 1 are encoded in the PRISM modelling language [5], and are synthesised automatically from the structure and annotations of the SBS activity diagram. Due to space constraints, we could not include the algorithms used for this synthesis in the paper. However, descriptions of these algorithms are available in our technical report [7].

Example 2. Fig. 4 depicts the initial DTMC model produced by the COVE tool for our telehealth SBS. The model comprises a state for each node from the UML activity diagram in Fig.2, and a "Fail" state for each of the three SBS operations. Only the non-zero costs c_{SA}, c_{AVP} and c_{CD} are shown in the diagram, next to the states corresponding to the three SBS operations. These costs and the operation success probabilities p_1, p_2 and p_3 represent model parameters that are updated continually at runtime. Finally, the PCTL formulae synthesised from the annotations in Fig. 2 and corresponding to requirements R_1–R_3 from Section III are:

R_1: $P_{\leq 0.08}[!\mathit{stop}\ U\ \mathit{sendAlarmFail} \mid \mathit{changeDrugFail} \mid \mathit{analyseVitalParamsFail}]$

R_2: $P_{\leq 0.05}[\mathit{true}\ U\ \mathit{sendAlarmFail}]$

R_3: $\mathit{analyseVitalParams} \Rightarrow P_{\leq 0.0002}[!\mathit{stop}\ U\ \mathit{sendAlarmFail}]$

C. Stage 3: SBS Construction

In this stage, the n COVE proxies are integrated with the code that implements the SBS workflow, in a similar manner to standard web service proxies. Additionally, an instance of the COVE model updater and an instance of the COVE autonomic manager are created and initialised (through their constructor parameters) with the initial DTMC model and the array of PCTL requirements from the previous stage, respectively. Standard development tools/integrated development environments can be employed in this stage.

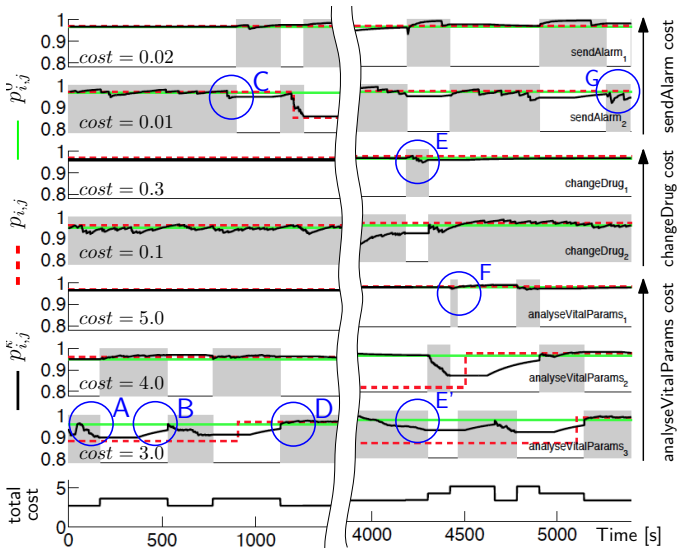

Fig. 5. Automated service selection for the telehealth SBS. The shaded areas denote selected services; the areas labelled 'A', etc. are analysed in the text.

V. Evaluation

We used COVE to implement a self-verifying version of our telehealth SBS with $m_1 = 2$ sendAlarm services, $m_2 = 2$ changeDrug services and $m_3 = 3$ analyseVitalParams services, all of which were implemented as Java web services deployed on Amazon EC2 (http://aws.amazon.com/ec2/) "small instance" virtual machines. Configuration files were used to define the probability of successful invocation for each service, $p_{i,j}$, $1 \leq i \leq 3$, $1 \leq j \leq m_i$, over the duration of each experiment. A Java implementation of the telehealth SBS workflow was integrated with COVE proxies for its three operations, and was run on a 2.66 GHz Intel Core 2 Duo Macbook Pro computer.

Fig. 5 depicts a typical experiment in which the self-verifying SBS selects its services dynamically, over a 1.5-hour wall-clock time period. Low-cost service combinations are preferred when their probabilities of successful completion satisfy all SBS requirements, and are discarded in favour of higher-cost service combinations when this is not the case. These choices are based on the estimate probabilities of success p^k_{ij} calculated by the COVE online learning algorithm, and on the continual verification of the updated SBS model.

Early in the experiment, the lowest-cost service combination is selected since the high *a priori* success probabilities $p^0_{i,j}$ make all service combinations seem suitable. However, when the autonomic manager learns that analysisVitalParams$_3$ is underperforming in the area labelled 'A' in the diagram, it switches to the higher-cost analysisVitalParams$_2$ service. While this higher-cost service is used, the COVE learning algorithm "rebuilds trust" in the temporarily discarded lower-cost service since the observations of frequent failures from

area 'A' are associated weights that decrease over time. Accordingly, the estimate $p^k_{3,3}$ increases slowly towards the prior value $p^0_{3,3}$, so analysisVitalParams$_3$ is selected again in area B, and is used until the SBS realises that it has not yet recovered. In area C, a slight variation in the estimate success probability of sendAlarm$_2$ triggers a potentially unnecessary transition to the more expensive sendAlarm$_1$. As explained in [7], using strict intervals of confidence for the COVE online learning could reduce the occurrence of such "false positives", though eliminating them completely is not possible. In area D, the SBS resumes using analysisVitalParams$_3$, which has now recovered, then in area E–E' it switches to the slightly more expensive changeDrug$_1$ service in order to avoid using a significantly more expensive analysis service. This choice proves unsuccessful, so the most expensive analysis service is eventually selected in area F. Finally, in area G all services have recovered and operate close to their advertised SLAs, so the SBS returns to using the lowest-cost service combination.

VI. Related Work

The management of SBS properties through dynamic service selection has been the focus of significant research over the past decade. The approaches proposed in [4, 11, 12] use UML activity diagrams or directed acyclic graphs to synthesise simple performance models based on queuing networks [4, 12] or, like COVE, Markovian reliability models [11]. These models are then used to establish the QoS properties of the analysed SBS systems. Unlike these approaches, COVE also uses an adaptive learning technique to update the initial model based on observations of the system behaviour. The QoS-driven selection of services in self-adaptive SBSs is addressed in [1, 3]. All of these approaches lack the adaptive learning capabilities of COVE, and propose theoretical solutions that are hard to replicate in practical SBSs. In addition, approaches such as [1, 3] involve the optimisation of the service selection on a per request basis. These approaches require perfect knowledge of the service QoS capabilities, and are ineffective in the scenarios targeted by COVE, where the characteristics of services need to be learnt from observations of their behaviour.

VII. Conclusion

We introduced the COVE tool-supported framework for the engineering of self-adaptive service-based systems. Unlike existing solutions, COVE combines continual verification (to guarantee reliability requirement compliance) with the automated generation of the components that deliver this verification (to reduce development effort), and with a development process that practitioners are familiar with (to ease adoption).

The effectiveness of the COVE was demonstrated in a case study from the telehealth application domain. In [7], we evaluated its scalability for three SBSs used by related projects, showing its applicability to practical SBSs with small to medium numbers of operations. Extending this to larger systems requires the exploitation of recently emerged incremental verification techniques [13, 14, 15], an area we are currently exploring, alongside the possibility to extend COVE with the ability to handle additional types of requirements such as performance and energy usage.

Acknowledgments This work was partly supported by the UK EPSRC Grant EP/H042644/1. The authors are grateful to James Holt for his work on the COVE proxy generator.

References

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

[2] R. Calinescu, L. Grunske, M. Kwiatkowska, R. Mirandola, and G. Tamburrelli, "Dynamic QoS management and optimization in service-based systems," *IEEE Trans. Softw. Eng.*, vol. 37, pp. 387–409, 2011.

[3] G. Canfora *et al.*, "A framework for QoS-aware binding and re-binding of composite web services," *J. Systems & Software*, vol. 81, no. 10, pp. 1754–1769, 2008.

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

[5] M. Kwiatkowska, G. Norman, and D. Parker, "PRISM 4.0: Verification of probabilistic real-time systems," in *CAV'11*. Springer, 2011, pp. 585–591.

[6] J. Kephart and D. Chess, "The vision of autonomic computing," *Computer*, pp. 41–50, 2003.

[7] R. Calinescu, K. Johnson, and Y. Rafiq, "Using continual verification to automate service selection in service-based systems," Dept. of Computer Science, University of York, Tech. Rep. YCS-2013-484, 2013, http://www.cs.york.ac.uk/ftpdir/reports/2013/YCS/482/YCS-2013-484.pdf.

[8] R. Calinescu, C. Ghezzi, M. Kwiatkowska, and R. Mirandola, "Self-adaptive software needs quantitative verification at runtime," *Comm. ACM*, vol. 55, no. 9, pp. 69–77, Sep 2012.

[9] I. Epifani, C. Ghezzi, R. Mirandola, and G. Tamburrelli, "Model evolution by run-time adaptation," in *ICSE'09*, 2009, pp. 111–121.

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

[11] S. Gallotti, C. Ghezzi, R. Mirandola, and G. Tamburrelli, "Quality prediction of service compositions through probabilistic model checking," in *QoSA'08*, pp. 119–134.

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

[13] A. Filieri, C. Ghezzi, and G. Tamburrelli, "Run-time efficient probabilistic model checking," in *ICSE'11*. IEEE Computer Society, 2011, pp. 341–350.

[14] K. Johnson, R. Calinescu, and S. Kikuchi, "An incremental verification framework for component-based software systems," in *CBSE'13*, 2013, pp. 33–42.

[15] M. Kwiatkowska, G. Norman, D. Parker, and H. Qu, "Assume-guarantee verification for probabilistic systems," in *TACAS'10*. Springer, 2010, pp. 23–37.