Abstract—We present the theory underpinning the operation of a new tool-supported approach to engineering self-adaptive service-based systems (SBSs), and preliminary results from its evaluation in a telehealth case study. SBSs developed using our approach select their services dynamically, in order to maintain compliance with reliability requirements in the presence of changes in service behaviour. This adaptation is enabled by a new type of web service proxy called an intelligent proxy.

I. INTRODUCTION

The development of new applications through the integration of third-party services accessed over the Internet is rapidly becoming the de facto standard in domains ranging from e-commerce and telehealth to bioinformatics. The process starts with the specification of a workflow that describes the business process underpinning a new application. Concrete services that implement the operations of this workflow are then selected, and standard tools are used to generate web service proxies for the selected services. Finally, the new application is assembled as a service-based system (SBS) through integrating these proxies with code that implements the business workflow.

Notwithstanding the broad adoption of SBSs, there is an implicit expectation that their component services may occasionally fail to operate dependably. This could be due, for instance, to individual services being slow or temporarily unavailable during periods of high workload. When such behaviour occurs infrequently and is short lived, an SBS may still satisfy its high-level requirements. However, when it occurs frequently or for longer periods of time, the high-level SBS requirements are violated. The problem is traditionally resolved through offline corrective maintenance in which the underperforming services are replaced with new services.

The automation of this SBS corrective maintenance process has been an active area of research in recent years. The numerous solutions proposed by this research range from approaches that use intelligent control loops (e.g., [1], [2], [3], [4]) to approaches which emulate the cooperative behaviour of biological systems (e.g., [5], [6]). Although these approaches to developing self-adaptive SBSs are effective in many scenarios, none has been adopted in SBS engineering practice. Our project aims to eliminate this divide between state-of-the-art research and the current state of practice. To achieve this aim, we introduce a tool-supported SBS engineering approach that combines the rigour of formal verification with a development process that SBS developers are familiar with. The main contributions and advantages of our approach include this use of an established development process, the use of formal verification to derive per operation service-level agreements, and the automated synthesis of the intelligent web service proxies underlying the working of our self-adaptive SBSs.

II. RUNNING EXAMPLE

We will use a telehealth service-based system taken from [2], [7] 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. 1, and we will consider that it must comply with three reliability requirements: $R_1$ 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 invocation of the analysis service fails 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 $p_{R_3} = 0.0002$.

III. SBS DEVELOPMENT PROCESS

Like in our previous work from [2], we consider a generic SBS workflow comprising $m \geq 1$ operations that are executed...
by remote third-party services. We further assume that the SBS needs to satisfy \( r \geq 1 \) reliability requirements similar to those from Section I. The development process employed by our approach comprises the three stages described below.

1. **Requirement analysis** — In this stage, the \( r \) requirements are analysed formally, to determine a combination of operation success probabilities \( p_1, p_2, \ldots, p_m \) for which the SBS satisfies the \( r \) SBS requirements. First, the UML activity diagram of the SBS workflow is used to devise a discrete-time Markov chain (DTMC) model \( M = (S, s_0, P) \) of the system, where: \( S \) is the finite set of workflow states associated with the activity diagram nodes and with the failure of each of its \( m \) operations; \( s_0 \in S \) is the initial state (corresponding to the start node of the diagram); and \( P \) is a transition probability matrix. Given two states \( s_x, s_y \in S \), the element \( p_{xy} \) from \( P \) represents the probability that the SBS will transition from state \( s_x \) to state \( s_y \). The probabilities associated with state transitions corresponding to activity edges not leaving from a decision node are all 1.0. Those associated with state transitions leaving from a decision node specify the frequency with which the different next activities are executed. Finally, the probabilities of the transitions from the state associated with the execution of the \( i \)-th SBS operation, \( 1 \leq i \leq m \), to the “successful invocation” and “failed invocation” states for the service that implements this operation are \( p_i \) and \( 1 - p_i \), respectively. The probabilistic model checker PRISM \(^{2} \) is then used to identify combinations of operation success probabilities that satisfy the \( m \) SBS requirements. This involves using PRISM to analyse the circumstances in which model \( M \) satisfies probabilistic temporal logic versions of each of the \( r \) SBS requirements.

Finally, the SBS developer completes the requirement analysis stage by: (a) selecting one of the acceptable combinations of operation success probabilities that places feasible constraints on the services to be used by the SBS; and (b) assembling the service-level agreements (SLAs) \( sla_i = (p_i, c_i) \), \( 1 \leq i \leq m \), for the \( m \) SBS operations, where \( c_i > 0 \) represents the maximum acceptable cost for the \( i \)-th operation.

2. **Intelligent proxy generation** — In this stage, intelligent web service proxies are generated for the \( m \) SBS operations. For the \( i \)-th operation, \( 1 \leq i \leq m \), the developer first selects \( n_i \geq 1 \) functionally equivalent services that implement the operation, but which may be associated with different levels of reliability and different costs. However, the reliability and cost \( advertised \) for each of these \( n_i \) services must satisfy \( sla_i \). Once the \( n_i \) candidate services have been selected, the developer uses the intelligent-proxy generator (IPGEN) tool developed by our project \(^{3} \) to generate a (Java) intelligent proxy for the \( i \)-th SBS operation. The functionality of IPGEN resembles that of standard web service proxy generators such as wsdl2java and wsdl2php, except that: (a) the proxy class generated by IPGEN is synthesised from the \( n_i \) web service WSDL definitions instead of a single one; and (b) the developer has to select the functionally equivalent methods of the \( n_i \) web services in a preliminary, GUI-supported step of the generation process.

3. **SBS construction** — In this stage, the \( m \) intelligent proxies are integrated with the (Java) code that implements the SBS workflow, in a similar manner to standard web service proxies. Additionally, the intelligent proxy associated with the \( i \)-th SBS operation, \( 1 \leq i \leq m \), comprises special “setSLA” and “setAdvertisedParams” methods that are used to specify the SLA \( sla_i \) for the operation, and the “advertised” reliability and cost for each of its \( n_i \) services, respectively. Once supplied with this information, the \( i \)-th intelligent proxy will select the service invoked for each execution of the \( i \)-th SBS operation dynamically, ensuring that \( sla_i \) is satisfied with minimal cost whenever at least one of the \( n_i \) services is able to do so. The online learning and runtime decision techniques employed by the intelligent proxies are described in the next section.

**Example:** Fig. 2 shows the DTMC model obtained for the SBS workflow from our running example in the requirements analysis stage. The probabilistic temporal logic formulae for requirements \( R_1 = R_3 \) from Section I are \( P_{\leq 0.08}[s_{10} U s_7] \land [s_8] s_9 \), \( P_{\leq 0.05}[F s_7] \), and \( s_3 \Rightarrow P_{\leq 0.0002}[s_{10} U s_7] \). The analysis of these requirements and the selection of a suitable combination of operation success probabilities \( p_1, p_2, p_3 \) for the telehealth SBS are illustrated in Fig. 3.

IV. **OPERATION OF THE INTELLIGENT PROXY**

The intelligent proxy associated with the \( i \)-th operation of an SBS, \( 1 \leq i \leq m \), fulfils three key functions:

1. Continuous online learning of the success probabilities for the \( n_i \) services that provide the SBS operation.

2. Dynamic selection of one of the \( n_i \) services, each time when the operation is executed.

3. Invitation of the selected third-party service.

The actual invocation of the selected service involves using standard web service proxies—as shown in Fig. 4—our IPGEN tool generates a standard proxy for each of the \( n_i \) services, and integrates it into the intelligent proxy. As service invocations are implemented using this standard mechanism, this section focuses on the first two functions of the intelligent proxy.

The intelligent proxy calculates estimates of the success probabilities of the \( n_i \) services using an enhanced variant of the Bayesian learning algorithm we presented in \(^{4} \). Unlike the algorithm from \(^{4} \), its new variant introduced in this paper continues to produce useful estimates during time periods when a service is no longer used, and thus observations of its behaviour are unavailable.\(^{5} \) We will describe the enhanced

---

\(^{1}\)The translation of the \( r \) requirements into probabilistic temporal logic is described in our previous work \(^{2} \).

\(^{2}\)A prototype version of IPGEN is available from [http://www-users.cs.york.ac.uk/~rafiqy/ipgen/](http://www-users.cs.york.ac.uk/~rafiqy/ipgen/).

\(^{3}\)We assume that service invocations are neither idempotent nor free, so invoking a service just to observe its behaviour is not possible.
learning algorithm by first explaining that the intelligent proxy estimates the success probability $p^k$ of the service it selected for the $k \geq 1$ most recent executions of the $i$-th SBS operation using the formula:
\[ p^k = \frac{c_0}{c_0 + k} p^0 + \frac{k}{c_0 + k} \sum_{l=1}^{k} w_l \sigma^l, \]
(1)
where the prior probability $p^0$ represents the success probability “advertised” by the service provider; $c_0 > 0$ is a smoothing parameter that quantifies the confidence in the accuracy of $p^0$; the $l$-th observation $\sigma^l \in (0, 1]$ is set to 1 if the $l$-th service invocation was successful and 0 otherwise; and, finally, $w_l$ is a weight that reflects the age of the $l$-th observation. Assuming that the $l$-th observation was made at time $t_l$, its associated weight is calculated as $w_l = \alpha^{-(t_k - t_l)}$, where $\alpha > 1$ represents an ageing coefficient. The estimate (1) has two major advantages over other learning techniques [9]. First, the weights $w_l$, $1 \leq l \leq k$, decrease the impact of old observations on the estimate $p^k$, speeding up the detection of service failures significantly, in particular when such failures occur after long periods of normal behaviour. Second, reorganising the terms in (1) allows $p^k$ to be calculated in $O(1)$ time from $p^{k-1}$, which is of great benefit for an online learning algorithm.

At any time, the online estimation technique from (1) is applied to one and only one of the $n_i$ services, namely to the service that was used for the last execution of the $i$-th SBS operation. An extension of the technique is used for the other $n_i - 1$ services:

- A service never used by the intelligent proxy is assumed to operate with its “advertised” success probability $p^0$.
- A service whose use was discontinued because a cheaper service that satisfies $sla$, became available is assumed to return to its “advertised” success probability $p^0$.

A service “discontinued” after $k > 0$ invocations because it ceased to satisfy $sla$, is handled using one of the techniques summarised in Table I and described below.

The “cooling off” technique for handling underperforming services is suitable for scenarios in which a service becomes unavailable suddenly. An intelligent proxy configured to use this technique stops using a service when its success probability $p^k$ from (1) drops below the SLA-specified value $p_t$. The service is reconsidered as (potentially) suitable after a fixed cooling-off time $t_{co} > 0$, which is a parameter of the technique. The “simple” technique is useful in scenarios in which a service is temporarily overloaded, so its probability of success decreases below the threshold specified in the operation SLA, but remains non-zero. When this technique is used for a service deemed unsuitable after the $k$-th invocation because $p^k < p_t$, the probability of successful service invocation at time $t_k > t_k$ is estimated as
\[ p^k = \begin{cases} p^0 \prod_{l=1}^{k} \left( \frac{p^0}{p^0 + \frac{1}{\beta^{t_k - (t_k + t_{co})}} \left( 1 - p^0 \right)} \right), & \text{if } t_k < t_k + t_{co} \\ p^0 \left( \frac{1}{\beta^{t_k - (t_k + t_{co})}} \left( 1 - p^0 \right) \right), & \text{otherwise} \end{cases} \]
(2)
where $\beta > 1$ is a recovery parameter, and $t_{co} > 0$ is a cooling-off time as before. As a result, the time until the service is reconsidered by the intelligent proxy depends on how much the service reliability $p^k$ dropped below the required value $p_t$. This advantage is shared by the “hysteresis” technique, which uses the same approach to estimating $p^k$ but additionally avoids frequent service changes by using a hysteresis parameter $\gamma$.

V. EXPERIMENTAL RESULTS

We used the development process from Section III to implement a version of our telehealth system that used $n_1 = 2$ sendAlarm services, $n_2 = 3$ analyseVitalParams services, and $n_3 = 2$ changeDrug services. These seven services

| TABLE I. HANDLING SERVICES THAT CEASE TO SATISFY $sla = (p_t, c_i)$. |
|-----------------|-----------------|-----------------|
| Criteria for deciding that: | Estimate for discontinued service |
| (a) service is no longer suitable | (b) unsuitable service is again suitable |
| cooling off | | |
| (a) $p^k < p_t$ | | not required |
| (b) $t_{co}$ time units elapsed since $t_k$ | | |
| simple | | |
| (a) $p^k < p_t$ | | see eq. (2) |
| (b) $p^k \geq p_t$ | | |
| hysteresis | | |
| (a) $p^k < (1 - \gamma)p_t$ | | see eq. (2) |
| (b) $p^k > (1 + \gamma)p_t$ | | |
| where $0 < (1 - \gamma)p_t < p_t < (1 + \gamma)p_t < 1$ | | |

Fig. 4. Intelligent proxy generation and architecture.

Fig. 3. Requirement analysis with PRISM. The three graphs show how the probabilities $p_R^1$, $p_R^2$, and $p_R^3$ from the telehealth SBS requirements in Section III vary for different combinations of operation success rates $(p_1, p_2, p_3)$. The shaded areas contain combinations that violate one of the requirements, and the combination $(p_1, p_2, p_3) = (0.95, 0.94, 0.93)$ satisfies all requirements and was selected to be used in defining the SLAs for the three SBS operations.
were simulated using real Java web services deployed on Amazon EC2 (http://aws.amazon.com/ec2/) “small instance” virtual machines. Individual configuration files were used to specify the variation of the actual probability of successful invocation for each web service, \( p_{\text{Actual},i,j} \), 1 \( \leq i \leq 3 \), 1 \( \leq j \leq n_i \), over the duration of each experiment. A Java implementation of the telehealth workflow from Fig. 1 and the intelligent proxies for its three operations were run on a standard 2.66 GHz Intel Core 2 Duo Macbook Pro computer.

Fig. 5 depicts a typical experiment in which the intelligent proxies select services dynamically, “moving away” from low-cost services whose reliability drops below the value \( p_i \), 1 \( \leq i \leq 3 \), specified in the SLA of the associated operation, but verifying if these services have recovered regularly, and using the “simple” technique from Table I to return to them when they did recover. Fig. 6 details the online learning and dynamic service selection carried out by the analyseVitalParams intelligent proxy during the time interval from the rectangular area labelled ‘A’ in Fig. 5. Early in the scenario depicted in this diagram, the reliabilities \( p_{\text{Actual},2,3} \) and \( p_{\text{Actual},3,3} \) of the two lower-cost analyseVitalParams services drop below the SLA value \( p_3 \). Accordingly, the reliability estimate \( p_{3,3} \) for the lowest-cost service used initially also drops below \( p_3 \) after a period of online learning, and the intelligent proxy starts to use the second least expensive service, analyseVitalParams 2. Learning that this service is also unsuitable is much faster due to the lack of a history of successful invocations, and the most expensive service is eventually used. However, the proxy continues to verify the suitability of the lower cost services regularly, and, shortly after these services recover, switches to using them—first the middle-cost service, and eventually the lowest-cost service.

VI. RELATED WORK

The management and optimisation of SBS properties through dynamic service selection has been the focus of significant research over the past decade. The solutions proposed by this research are overviewed in detail in our previous work [2], and include approaches that use intelligent control loops (e.g., [1], [3], [4]) and approaches that emulate the cooperative behaviour of biological systems (e.g., [5], [6]). However, to the best of our knowledge, none of these existing solutions combines formal requirement analysis with a tool-supported engineering process that resembles established development processes. Another key improvement over our previous work from [2], is the use of an advanced learning technique to estimate the service success probabilities, and its extension to cope with time periods when a service is no longer invoked.

VII. CONCLUSION

We introduced a tool-supported approach to engineering self-adaptive service-based systems, and applied it to the development of a telehealth system. Unlike existing solutions, the new approach combines formal requirement analysis with a development process that today’s practitioners are familiar with, and automates the generation of the intelligent proxies underpinning the operation of its self-adaptive SBSs. Our ongoing work focuses on automating the requirement analysis stage described in Section III, and on integrating the component that implements this analysis into the intelligent proxies.

ACKNOWLEDGMENT

This work was partly supported by the UK Engineering and Physical Sciences Research Council grant EP/H042644/1.

REFERENCES