Scalable Computing: Practice and Experience
Volume 12, Number 1, pp. 63–78. http://www.scpe.org
ISSN 1895-1767

A QOS-BASED FRAMEWORK FOR THE ADAPTATION OF SERVICE-BASED SYSTEMS

RAFFAELA MIRANDOLA* AND PASQUALINA POTENA†

Abstract. Since a system may require dynamic adaptation for several reasons (e.g., a new version may be available and a new functionality or a different level of quality of service) it should be possible to dynamically adapt a service-based system in an automated manner. In this paper we give a general overview of the main components of a framework, based on an optimization model, that dynamically adapts a service based system (i.e., both the structural and behavioral software and hardware architecture) while minimizing the adaptation costs and guaranteeing a required level of the system qualities. Adaptation actions can be triggered both by a user request and/or automatically after the runtime violation of system quality constraints, or the appearing/disappearing of services into the environment. In this paper we provide also a deeper discussion of the optimization model that is the core of the framework by providing an example of instantiation of the model together with a first experimentation.

Key words: service-based adaptation; quality of services and non-functional aspects; optimization model.

1. Introduction. Due to the frequent changes required to software after its release, the development processes are rapidly going towards small effort dedicated to the early phases and large effort for the after-deployment phases. Since a system may require dynamic adaptation for several reasons, such service evolution (e.g., a new version may be available), hardware volatility (e.g., network quality changes) and varying users demands requiring new requirements (e.g., a new functionality or a different level of quality of service) it should be possible to dynamically adapt a service-based system in an automated manner. An application should be self-adaptive in order to automatically and autonomously adapt its behavior to respond to changes. For example, in the pervasive computing domain [13], an application should be context-aware self-adaptive, i.e., the context capturing the notion of information about the physical environment should trigger certain changes. The switch from a Wifi network to a GSM one could, for example, be require (survey on context-aware systems can be found in [13, 24, 35, 39]).

In this paper we introduce a framework, based on an optimization model, that dynamically adapts a Service-Based System (SBS) while minimizing the adaptation costs and guaranteeing a required level of the system qualities. Adaptation actions can be triggered both by a user request and by an automatic request raised after the runtime violation of system quality constraints, or the appearing/disappearing of services into the environment. In particular, in this paper we give a general overview of the main components of the framework by providing a deeper discussion of the optimization model that is the core of the framework.

Specifically, to modify the software structure the framework replaces existing software services with different available instances and embeds into the system new software services if necessary. With respect to changes in the system behavior it modifies the system scenarios (expressed, for example, as UML Sequence Diagrams) by removing or introducing interaction(s) between existing services and between these latter and new services. Finally, to modify the hardware architecture it may re-deploy (deploy) existing/new services on the hardware nodes. Furthermore, it may improve (adopt new) hardware resources (e.g., CPU, disk, memory, etc) and modify the interaction between hardware nodes (e.g., introducing a new link). Note that, these hardware improvements can be done only if the service belongs to the application developers or from the provider itself when composing some complex SBSs.

This paper extends the work in [30] by providing a more detailed presentation of the framework. In particular, we provide a deeper discussion for the example of instantiation of the model by providing a first numerical experimentation.

The paper is organized as follows: Section 2 introduces related work and discusses the novelty of our contribution; in Section 3 we introduce our framework; in Section 4 we provide the formulation of the optimization model that represents the core of our framework; in Section 5 we provide an example of instantiation of the model together with a first experimentation; conclusions are presented in Section 6.

2. Related Work. A wide range of approaches and frameworks for adaptation have been proposed (see surveys [4, 14, 21]). Most of them typically adapt a system by (i) service selection (see, for example, [9, 10, 34]

*Politecnico di Milano, Dipartimento di Elettronica e Informazione, Piazza Leonardo da Vinci, 32, Milano, 20133 Italy (mirandola@elet.polimi.it).

†Università degli Studi di Bergamo, Dipartimento di Ingegneria dell'Informazione e Metodi Matematici, Viale Marconi, 5, Dalmine (BG), 24044 Italy (pasqualina.potena@unibg.it).

detailed below), (ii) parametrization (see, for example, [8]) or (iii) exploiting the inherent redundancy of the SOA environment (see, for example, [17]).

Paper [9] presents a runtime selection of services composing a web service while satisfying QoS constraints, maximizing some QoS (e.g., reliability) and minimizing other attributes, such as the price. It leverages both on formulas for estimating QoS attributes with respect to some workflow constructs (i.e., sequence, switch, flow and loop) and genetic algorithms for solving the problem.

In [34] an approach for supporting the service selection driven by QoS attributes is presented. It enhances the classical service scenario (i.e., consumer, provider and registers) by assuming that the service register stores, apart from the information of the services (e.g., QoS data and interfaces), information about services usage and service trees. A service tree is a binary search tree generated by adding a node for each service that could be selected by the customer. Each node is labeled with the service score, which is defined as the weighted sum of the qualities of the service. A service score could be updated, for example, if the service receives a good feedback from other consumers. The algorithm for building the service tree and for updating and selecting (or re-selecting) the best service is presented. In fact, a service could be re-selected, for example, if either the currently selected service is not available or the QoS of the service changes.

In [10] it is presented an approach for the self-adaption of a SOA system in order to meet reliability and availability requirements while dynamically changing the environment. The approach allows adopting simultaneously (for different users, but also for different requests generated by the same user) adaptation actions based on the service selection and architecture selection.

In [19] an approach supporting the impact of the replacement of a service on other services is presented (e.g., how the change of a service provider impact on the other services). It is based on workflow patterns and on the value of changed information [18] for sequence and different parallel pattern scenarios.

Our framework takes advantage of the use of monitoring technique, using approaches like the ones detailed below.

In [31] the VieDAME (Vienna Dynamic Adaptation and Monitoring Environment forWS-BPEL) tool is introduced. It supports the monitoring of BPEL processes according to certain QoS criteria and the replacement of a service with one of its available functional equivalent alternatives. Moreover, it allows the handling of interface mismatches at a SOAP message level.

In [6] an approach for the dynamic monitoring and recovery of BPEL processes is introduced. The monitoring exploits WSCoL (Web Service Constraint Language), whereas the recovering is implemented by introducing WSReL (Web Service Reaction Language) providing a library of atomic actions that can be combined to create reaction strategies, which are activated whenever an anomaly is discovered by the monitoring. An implementation of the approach is proposed using AOP techniques to activate both dynamic monitoring and recovery during process execution.

Lodi et al. [27] describe a QoS-aware middleware that can be included in application servers in order to implement Service Level Agreement (SLA)-driven clustering of servers. The middleware operates by dynamically configuring, monitoring and balancing the load among different servers, in order to provide strong QoS guarantees despite high variability in the request rates.

In order to adapt a system, for example, while changing the requirements (e.g., new functionalities could be claimed or a different level of quality of service), it might be necessary to modify the structure of the service composition, e.g., new services have to be embedded into the system and the interactions between existing services have to be changed for assuring a certain level for the system qualities. An application should automatically and autonomously adapt its behavior (i.e., its service composition) to respond to changes under the guide of the system qualities.

Some frameworks have been introduced for dynamically generating a service composition, e.g., [36] surveys automated web service composition methods. Usually they adapt a system only in a proactive way, after the adaptation request triggered by a user. These frameworks support the service selection with respect to a service composition defined by the user (see, for example, the VRESCo runtime environment [37] where a user can require a service composition satisfying quality constraints) or choose the service composition, which they have generated together with a finite set of other candidates, that better fulfills the required quality (see, for example, [11], where it is presented a framework composing services at run time for answering a user query and the survey [21] where it is defined a service composition middleware model describing the existing service composition middleware in pervasive environments).

In our view, a service composition should be dynamically generated, other than after a user request, sponta-

neously by the application itself. The spontaneous service composition is typically not supported by the existing frameworks, such as the ones for pervasive environments (see survey [21]) where instead it should be one of the fundamental properties of an application. In [22] a spontaneous service integrator middleware is presented supporting the extension of system functionalities by integrating new services appearing into the environment with existing services. Moreover, it supports the spontaneous service selection while services appear (disappear) into the environment. For the selection activity the authors uses a metric measuring the degree of non-functional QoS similarities between two equivalent syntactic or semantic services.

They do not consider the impact of replacing a service on the quality of the whole system. Moreover, they do not consider the possibility of replacing also the other system elementary services, i.e., the service selection with respect to all services. As opposite, our framework supports the spontaneous adaptation of the hardware and structural and behavioral software architecture if system quality constraints are violated at run time or services appear (disappear) in the environment. Our framework, based on an optimization model, minimizes also the cost of adapting the system while assuring a required level of the system qualities.

The system quality depends, both on the software features, and on the hardware features. A set of software adaptation actions may differ for the system quality achieved after their application while changing the hardware structure. In fact, for example, the replacement of a service with a certain instance may be a good decision for the system reliability, but at the same time it could increase the system response time because of the execution time of the instance, which, in this case, may be decreased by modifying the characteristics of the hardware resources (e.g., CPU, disk, memory, network throughput, etc). Therefore, an application should automatically and autonomously adapt, other than its software behavior, its hardware architecture to respond to changes, whenever this is possible.

Novelty of our approach

With respect to existing approaches, the following major aspects characterize the novelty of the proposed method:

- This is one of the few papers (to the best of our knowledge our previous work is the first [30]) introducing a dynamic self-adaptive framework supporting both the software (including both static and dynamic models) and hardware architecture adaptation using an optimization model minimizing the adaptation costs and assuring a required level of the system qualities.
- Adaptation actions can be triggered both by users, for example, claiming the introduction of new functional (non-functional) requirements or changing already implemented system functionalities (non-functional requirements satisfaction) and spontaneously by the framework itself after the runtime violation of system quality constraints or the appearing/disappearing of services into the environment.
- The proposed approach is general and does not rely on specific architectural style, development process or service application domain. It could be specialized with respect to the application domain chosen. For example, it could be enhanced to support the context-aware property of the pervasive systems exploiting the properties discussed in [4].
- The proposed optimization model is independent from the methodology adopted to represent system architectural models and from the strategy used to generate the adaptation plans for each adaptation requirement. In fact, we assume that each scenario of a functionality offered by the system is represented by an UML Sequence Diagram. However, since our model uses as parameters of a scenario the number of busy periods of an elementary element (see Section 5), a scenario could be represented with whatever notation that permits to describe the system scenarios (e.g., the Message Sequence Charts, MSCs [2], or the UML Collaboration Diagrams).
- Our framework can facilitate the work of a software engineer. In fact, a user does not have to insert as input value architectures satisfying all changes required, but possible sets of (software and hardware) adaptation actions (called *adaptation plans*) for each new requirement (called *adaptation requirement*), which he/she want to implement. When the computation time becomes too big, e.g., while increasing the number of adaptation plans, the adoption of metaheuristic techniques possibly combined with the introductions of dependencies between adaptations plans of different adaptation requirements could allow the reduction of the research space.

3. Framework. Figure 3.1 depicts the main modules of our framework. The framework, based on an optimization model (generated and solved by the *Generator and Evaluator* module), dynamically adapts a service based system (by changing software and hardware features by means of the *Executor* module) while

minimizing the adaptation costs and guaranteeing a required level of the system qualities. Adaptation actions can be triggered both by requests of a user (which can be defined using the *User Requests Manager* module that can also interact with the *Monitor* to know if the system quality constraint are violated by the current runtime system) and/or automatically by the framework itself (after it receives alerts from the *Monitor* using probes for monitoring the system and interacting with services repositories by means of the *Provider Info* module).

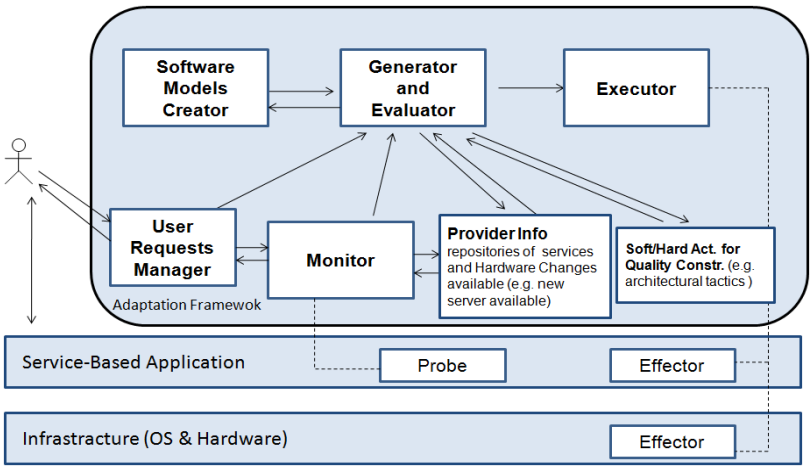

FIG. 3.1. *The Adaptation Framework*

3.1. Software Models Creator. Goal of the Software Models Creator is to generate system models (e.g., component diagram, sequence diagrams and deployment diagram), from the system implementation (by assuring consistency between the architecture and implementation [33]) or by modifying the existing models as suggested by the *Generator and Evaluator* module. The optimization model, generated and solved by the *Generator and Evaluator* module, is independent from the methodology adopted to represent system architectural models and from the strategy used to generate the adaptation plans for each adaptation requirement.

3.2. User Requests Manager. A user can use the User Requests Manager module to make adaptation requests. He/She may claim the introduction of new services offered by the system and modify the dynamics of existing services [1]. Furthermore, he/she may either require a value threshold (e.g., as lower bound) for a new system quality or modify the value threshold for an already required system quality. Before making his/her requests, the user can know (by means of the module *Monitor* interacting with the *User Requests Manager*) if there are requirements violated by the current runtime system. For each adaptation requirement, the *User Requests Manager* allows users to annotate the software architecture diagrams for defining different set of adaptation actions able to guarantee this new requirement.

In order to keep as simple as possible the modeling aspects of our work, we assume that plans of different adaptation requirements are independent from each other, namely the modifications claimed by a plan do not comprise both the satisfaction of the existing functional requirements, and the application of plans for other adaptation requirements. We intend to work for relaxing such a assumption. To this extent, we are investigating how to enhance the framework using the guidelines of the existing tools (like the CoDesign tool [5]) allowing architects geographically distributed to cooperate in order to design a system.

A software adaptation plan may suggest how to change both the structure and the behavior of a system. Specifically, to modify the system structure it suggests how to replace existing software services with different available instances and if the adoption of new software services is necessary. With respect to changes in the system behavior it may suggest how to modify the system scenarios (expressed, for example, as UML Sequence Diagrams) by removing or introducing interaction(s) between existing services and between these latter and new services.

[1] Throughout the paper, the services offered by the system will be named external services.

To define the software adaptation plans, for example, initial solutions based on the use of meta-heuristic techniques can be found in [28]. In fact, in [28] it is suggested how to explore the design space for generating architecture candidates with required performance features. It is basically based on two methods executed in parallel for generating the best architectural candidates from a (some) architecture(s) fulfilling the functional requirements. The first one uses the classic metaheuristic search techniques [7], whereas the second one uses specific knowledge of the performance analysis to specifically solve performance problems in current candidates. As last step, a detailed performance simulation is executed for the best architecture candidates.

To generate software adaptation plans for non-functional requirements several approaches can be adopted, for example, looking for "architecture bad smells", i.e., recurring software designs solutions that negatively impact software quality [15] and/or the application of architectural tactics, i.e., reusable architectural building blocks that provide a generic solution to address issues pertaining to quality attributes [25].

Finally, for each adaptation requirement, the *User Requests Manager* allows users to annotate the hardware architecture diagrams to suggest also hardware architecture changes (such as how to deploy existing/new services and if adopt new hardware nodes or links). For example, if he/she is a service provider, in order to improve the system response time he/she could suggest to try to improve its hardware resources (e.g., CPU, disk, etc.).

3.3. Monitor. Goal of the Monitor is to monitor the system at runtime and raise the violation of non-functional requirements, e.g., the system reliability is under the minimum level required. Moreover, it claims the appearing/disappearing of services in the environment by interacting with service repositories managed by service providers. Such repositories are accessed by interacting with the *Provider Info* module.

For implementing such a module the guidelines of the several approaches presented in literature leveraging on monitoring technique can be used (e.g., [27, 31, 41]).

Moreover, the monitor could continually measure the QoS attributes of the services (see, for example, the monitor used in [37]) or the providers could improve the estimate of the services non-functional property by monitoring them e.g., as suggested in [29], the providers monitor the resources utilization as client applications access them, and dynamically adjust the advertised average service times as better estimates are computed.

3.4. Provider Info. Goal of the Provider Info module is to allow accessing the service repositories managed by service providers (or by the brokers). These latter ones can use such a module to notify possible hardware changes that could be performed (e.g. possible deployments/re-deployments of services, how can be improved hardware resources, such as CPU, disk, etc.). By using such information the framework automatically suggests additional hardware actions by introducing variables and constraints in the optimization model generated and solved by the *Generator and Evaluator* module.

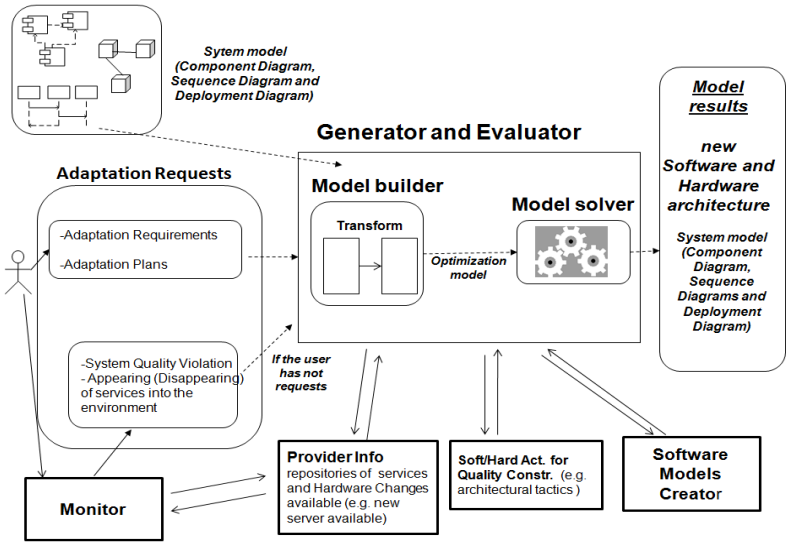

FIG. 3.2. *The Generator and Evaluator module and its environment*

3.5. Generator and Evaluator. Figure 3.2 shows the *Generator and Evaluator* module within its working environment. It is made of two components, which are a *Model builder* and a *Model solver*.

The *Model builder* after receiving either adaptation requests from the user (by means of the *User Requests Manager* module) or alerts from the *Monitor* generates the optimization model in the format accepted from the solver (e.g., LINGO [1]).
The user defines adaptations plans for each new requirement, whereas if the *Model builder* gets alerts from the *Monitor* about the violation of non-functional requirements, it itself generates a set of adaptation plans for each non-fulfilled requirement using the *Software/Hardware Actions for Quality Constraints* module. Such a module is a repository of software and hardware adaptations actions that may be used for improving the system qualities (e.g. general solutions as the architectural tactics or software/hardware actions defined by the system maintainer based on his/her experiences). Instead, if the *Monitor* notifies the appearing (disappearing) of services in the environment, the *Model builder* generates the optimization model for finding the best service selection actions while minimizing the replacement costs and guaranteeing a certain level for the system qualities. Note that the framework could be enhanced for generating adaptation plans also for each non-functional requirement of the system.

The *Model solver* processes the optimization model received from the builder and produces the results, that consists in a set of software and hardware adaptation actions. It suggests how to adapt both the static and dynamic software architecture and the hardware architecture. The combined use with the *Software Models Creator* module generates the new software (hardware) architectural models (e.g. component, sequence and deployment diagrams).

The framework could be enhanced for supporting the service level agreement (re-)negotiation, for example, if there is not a configuration of services able to guarantee the required system quality.

3.6. Executor. Goal of Executor module is to implement the adaptation actions suggested by the module *Generator and Evaluator*, i.e., it implements the system as described by the new software (hardware) diagrams defined by the optimization model. To this extent, existing guidelines of the approaches supporting the dynamic service invocation (e.g. [26]) and the ones for dynamically adapting the system behavior (e.g. the context-oriented programming [20]) could be exploited.

4. Optimization Model Formulation. In this section we introduce the general formulation of the model, that is generated and solved by the *Generator and Evaluator* module, aimed at finding the optimal set of adaptation actions needed to tackle required changes to the software and hardware architecture of a service based system. "Optimal" is here intended as the actions that incur in a minimum cost while guaranteeing a required level of a number Q of quality attributes (such as response time, availability, etc.) for the whole system.

Since our model may support different service application domain, we adopt a general definition of software service: it is a self-contained deployable software module containing data and operations, which provides/requires services to/from other elementary elements. A service instance is a specific implementation of a service.

Through the composition of its n software services, the system offers external services to users. Let us assume that for each external service we dispose of a diagram (e.g. a UML Sequence Diagram (SD)) describing its dynamics in terms of interactions that take place between software services to provide the external service. Let us also assume that we dispose of SDs for external services that are not active in the current system implementation, but they can be activated to satisfy new requirements.

Let s_i be the i-th service ($1 \leq i \leq n$).

Let K be the total number of SDs, i.e., the ones related to active external services plus the ones related to external services that may be introduced.

Let $pexec_k$ be the probability that the k-th system functionality will be invoked. It must hold: $pexec_k \geq 0$ for all $k = 1 \dots K$ and $\sum_{k=1}^{K} pexec_k = 1$. This information can be synthesized from the operational profile [32][2]. It is obvious that for a non-active functionality k we get $pexec_k = 0$.

Getting User Requests - An *user adaptation scenario* is a set of new requirements to be satisfied, which are claimed by the user. The new requirements are: (i) a new functional requirement, i.e., either introducing a new external service or modifying the dynamics of an existing external service; (ii) a new non-functional requirement, i.e., either requiring a value threshold (e.g. as lower bound) for a new system quality or modifying the value threshold for an already required system quality.

[2]Note that such assumption might be not realistic in all cases the operational profile may be not (fully) available. However, in such cases the domain knowledge and the information provided by the software architecture could be used for estimating it, as suggested for example in [38].

In order to introduce a new external service one of the SDs that are not active must be activated, namely it enters the set of SDs that contribute to the costs, reliability, response time etc. of the whole software architecture. In order to modify the dynamics of an existing external service certain interactions of the SD related to the service have to be added/removed. Finally, to introduce (modify) new/existing non-functional requirements certain interactions of the SDs may be added/removed, for example, by following the guidelines of the "architectural tactics" (see Section 3).

Getting Alerts by the Monitor - An alert raised by the *Monitor* can claim: (i) the violation of a non-functional requirement, e.g. the system reliability is under the minimum level required; (ii) the appearing/disappearing of a service in the environment.

In order to satisfy either the new requirements required by the user or exploit the alerts raised by the *Monitor* some adaptation actions have to be performed. The user defines adaptation plans for each requirement, whereas the framework itself defines adaptation plans when it gets alerts about the violation of non-functional constraints. An adaptation plan is a set of actions modifying the static and dynamic structure of the software architecture and the hardware architecture to (exploit a certain alert) address a certain requirement. Obviously, for each requirement (alert) several adaptation plans may be available. Since they suggest different adaptation actions, they may differ for adaptation cost and/or for the system quality achieved after the application of their actions. Let AP_r be the set of adaptation plans available for the r-th requirement in the user adaptation scenario (the r-th alert raised by the *Monitor*)[3].

We consider the following software adaptation actions:

1. *Introducing new software service*: An adaptation action may suggest to embed into the system one or more new software services [4]. We call $NewS$ the set of new available services (accessed by the *Provider Info* module) that can provide different functionalities, whereas $news_h$ represents the h-th service of $NewS$.
2. *Replacing existing service instances with functionally equivalent ones*: An adaptation action may suggest to replace a software service with one of additional instances available for it (e.g. a web service available in the market). We assume that the additional instances available for the service s_i are functionally compliant with it, i.e., each instance provides at least all services provided by s_i and requires at most all services required by s_i [5]. The instances may differ from s_i for cost and quality attribute (e.g. reliability and response time). We call $Avail_i$ the set of instances available for the s_i, while s_{ij} is the j-th instance of $Avail_i$.
3. *Modifying the interactions between software services in a certain external service*: An adaptation action may suggest to modify the system dynamics by introducing/removing interactions between software services within a certain external service.

The system quality depends on the hardware features other than on the software features. In fact, a set of software adaptation actions may differ for the system quality achieved after the application of their actions while changing the hardware structure (e.g. the system response time may decrease while improving the processing capacity of a hardware node). Since the services are not acquired in terms of their binaries and/or source code, but they are simply used while they run within their own execution environment (that is not necessarily under the control of the system using them), hardware changes can be suggested by the service providers. To this extent, these latter can insert such information by means of the *User Requests Manager* module while requiring adaptation requirements and notify them by means of the *Provider Info* module.

In the following we discuss possible hardware adaptation actions.

- *Defining the deployment of software service on hardware nodes*: An adaptation action may suggest how to re-deploy existing services and/or deploy the new ones on the hardware nodes.
- *Introducing hardware resources*: An adaptation action may suggest to embed into the system one or more new hardware resources, e.g. a new hardware node where to deploy the services[6].
- *Modifying hardware resources*: An adaptation action may suggest to modify the characteristics of the underlying hardware resources (e.g. CPU, disk, memory, network throughput, etc).

[3] In the remainder of the paper a plan $p \in AP_r$ is also called ap_{rp}.

[4] Note that such type of action has to be associated to another action that indicates how this software service interacts with existing services, therefore it modifies the interactions within certain functionalities (see last type of software action).

[5] As we have remarked in [12] such an assumption could be relaxed by introducing integration/adaptation costs.

[6] Note that such type of action has to be associated to another action that indicates how this node interacts with existing nodes (see last type of hardware action).

TABLE 4.1
Example of adaptation plans

Requirement ID	Adaptation Plan ID	Adaptation Plan Description
req_1	ap_{11}	*Replacing s_3 with its first instance AND Replacing s_4 with its second instance*
	ap_{12}	*Adding a new service $news_2$*
req_2	ap_{21}	*Replacing s_2 with its first instance*
	ap_{22}	*Adding a new service $news_1$ AND Replacing s_5 with its first intance*
	ap_{23}	*Adding a new service $news_2$*

- *Modifying the interactions between hardware nodes*: An adaptation action may suggest to introduce/remove connection links between hardware nodes.

It is obvious that any combination of such software and hardware adaptation actions may have a considerable impact on the cost and quality of the system. Therefore, our optimization model aims at quantifying such impact to suggest the best adaptation plans that still minimize the costs while satisfying the quality constraints defined according to the quality properties of interest. These latter predict the quality of the system after the adaptation phase, i.e., after applying the adaptation plans for implementing the new requirements (exploiting the alerts of the monitors). Therefore, for each non-functional property required for the system (i.e., existing properties or new ones required by the user) a constraint is defined verifying the impact of the changes on the system quality.

Finally, in order to open the solution space to additional possibilities, it may suggest additional service replacement actions and hardware actions. In other words, in the optimization model we leave to the solver the possibility to choose additional service replacement actions and hardware actions that have not been embedded in any selected plan.

4.1. An example of Output of the optimization model. Let us assume that a user claims two new requirements for a system composed by five services (i.e., s_1, s_2, s_3, s_4 and s_5). Through the composition of these software services, the system offers external services to users. Table 4.1 describes the user adaptation scenario, where for each new requirement the adaptation plans suggested by the user are reported. For example, req_1 can be managed by either "Replacing s_3 with its first instance AND Replacing s_4 with its second instance" (ap_{11}) or "Adding a new service $news_2$"(ap_{12}). For the sake of simplicity, we have used the natural language for defining the requirements and the adaptation plans.

Let us suppose that to achieve the optimal cost of adaptation and assuring a required level of the system qualities the optimization model suggests to adopt the following plans: adaptation plan ap_{12} for the first requirement and adaptation plan ap_{22} for the second requirement. In this case, the static structure of the software architecture, describing dependencies between two services of the system and the behavior of the system with respect to the external service $\overline{k}$ would change as shown in Figure 4.1. In particular, the model could suggest all the replacements of existing services (i.e., s_{ij}), as well as the adoption of the new services $news_1$ and $news_2$. In addition, the model could also suggest to modify the hardware architecture as shown in Figure 4.2. Note that a new hardware node is introduced and some links are removed or adopted [7].

4.2. Model Variables and Elementary Constraints. The following variables help to select an adaptation plan for each requirement of a user adaptation scenario (for each alert raised by the *Monitor*). For each requirement r of a user adaptation scenario, exactly one plan must be chosen if it is a functional requirement. If it is a non-functional requirement it may be not necessary to select one of the plans suggested for it by the users since for each required quality of the system a constraint, predicting the quality resulting after the adaptation phase, is defined. For example, the reliability for the system is assured above a certain threshold without applying one of the "architectural tactic" suggested by the user. Similarly, for each alert claiming the violation of non-functional constraints it may be not necessary to select one of the plans generated by the framework itself.

[7] In the figures we have marked with bold the modifications brought after the application of the plans and we have put a cross on the interactions that are removed.

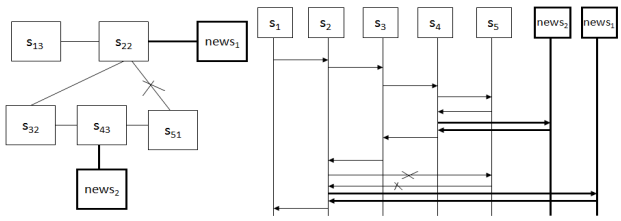

FIG. 4.1. *Resulting static structure and a SD of the system after the application of* ap_{12}, ap_{22}

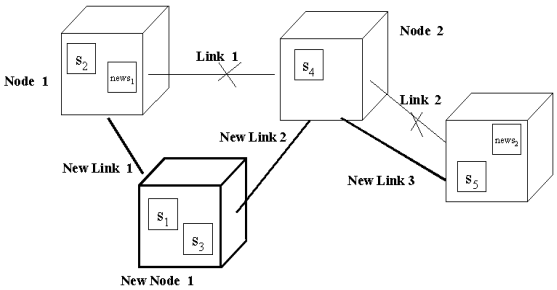

FIG. 4.2. *Resulting Hardware Architecture of the example*

$$y_{rp} = \begin{cases} 1 & \text{if } p \text{ is the plan chosen for requirement } r \\ & \text{(for the alert } r\text{) } (p \in AP_r) \\ 0 & \text{otherwise} \end{cases}$$

The following variables help to select an instance available for the i-th existing service. For each existing service i still used after the adaptation phase, exactly one instance must be selected.

$$x_{ij} = \begin{cases} 1 & \text{if the instance } j \text{ is chosen for the service } i \\ & (j \in Avail_i) \\ 0 & \text{otherwise} \end{cases}$$

If there are no available instances, we assume that $Avail_i$ includes the element i itself. In addition, if a plan is part of the model solution then all instances that it suggests for the existing services have to belong to the solution.

The following variables help to select the new software services to be introduced.

$$z_h = \begin{cases} 1 & \text{if the service } h \text{ is chosen } (h \in NewS) \\ 0 & \text{otherwise} \end{cases}$$

Similarly, if a plan is part of the model solution then all new services that it suggests to introduce have to belong to the solution.

Additional constraints, which can be expressed as contingent decisions [23], may be added for claiming incompatibility between services due to problems such as technology and licensing.

Different kinds of variables and constraints can be defined to support hardware actions (e.g. variables to introduce new hardware nodes and links). In the following, for example, we define the variables and discuss constraints to suggest the deployment of the existing/new services on hardware nodes.

The following variables suggest how to deploy the i-th existing/new service on an hardware node. Each existing service i, which is still used after the adaptation phase, must be deployed on exactly one node t. If the new service h is selected, it must be deployed on exactly one node t.

$$S_{it} = \begin{cases} 1 & \text{if the existing/new } i\text{-th service} \\ & \text{is deployed on node } t \ (t \in HN) \\ 0 & \text{otherwise} \end{cases}$$

where HN is the set of hardware nodes. Additional constraints can be defined for existing services that cannot be re-deployed on a different node for legacy reasons (i.e., their deployment does not have to be modified). Also, if a plan is part of the model solution then all the mappings of existing/new services on hardware nodes that it suggests have to belong to the solution. Finally, additional constraints have to be defined to guarantee that a path exists, i.e., a set of links exists, between two (existing and/or new) interacting services.

4.3. Cost Objective Function and Quality Constraints. We want to adapt a system while minimizing the adaptation cost under constraints on a number Q of quality attributes (such as response time, availability, etc.) of the whole system.

Let us suppose that the value of each attribute of any service composing the system after the adaptation phase depends on the value of parameters u, v and w.

Let u denote the maximum number of software architecture observable parameters, such as the number average of busy periods (i.e., the number of invocations of the service within a certain SD, and it can be easily estimated by parsing the diagram and counting the number of activations along its lifeline). Let v represent the maximum number of hardware observable parameters, e.g. the processing capacity of the node hosting the service measured, for example, as the number of instructions per time unit that the resource can execute, under the assumption that all instructions require equal time complete. Finally, let w be the maximum number of parameters expressing the specific features of its implementation (e.g. the reliability of the instance used for replacing an existing service).

Let $\Gamma_{kq} : \Re^u \times \Re^v \times \Re^w \rightarrow \Re$ ($\bar{\Gamma}_{kq} : \Re^u \times \Re^v \times \Re^w \rightarrow \Re$) be the function that predicts the q-th quality attribute ($1 \leq q \leq Q$) of an existing/new service from the u, v and w parameters after the adaptation phase with respect to the k-th external service. For the sake of readability, we introduce here a formulation without correlations among different quality attributes and services interacting between each other (in Section 6 we discuss how relaxing such assumptions).

We can represent the value of the q-th quality attribute of the i-th existing service as a function of the decisional strategy:

$$\theta^q_{ki} = \sum_{j=1}^{|Avail_i|} x_{ij}\Gamma_{kq}(\lambda^q_{i1}, \ldots, \lambda^q_{iu}, \lambda'^q_{i1}, \ldots, \lambda'^q_{iv}, \Lambda^q_{ij1}, \ldots, \Lambda^q_{ijw})$$

θ^q_{ki} is a function of the value of the software architecture observable parameters λ^q_{it}'s denoting, for example, the added/canceled busy periods that the chosen adaptation plans suggest for the service i within the k external service. θ^q_{ki} depends also on the value of the hardware observable parameters λ'^q_{it}'s, denoting, for example, the processing capacity of the hardware node, that is suggested by one of the chosen adaptation plans or that could be one of the additional actions of the model, where the service is deployed after the adaptation phase. Finally, θ^q_{ki} depends also on features Λ^q_{ijt}'s indicating, for example, the probability of failure of its selected instance or the size of the executable file containing this instance after the adaptation phase.

We can represent the value of the q-th quality attribute of the h-th new service as a function of the decisional strategy:

$$\bar{\theta}^q_{kh} = z_h\bar{\Gamma}_{kq}(\lambda^q_{h1}, \ldots, \lambda^q_{hu}, \lambda'^q_{h1}, \ldots, \lambda'^q_{hv}, \bar{\Lambda}^q_{h1}, \ldots, \bar{\Lambda}^q_{hw})$$

$\bar{\theta}^q_{kh}$ is a function of the value of the software architecture observable parameters λ^q_{ht}'s denoting, for example, the number of busy periods that the chosen adaptation plans suggest for the h-th new service within the k-th external service. $\bar{\theta}^q_{kh}$ depends also on the value of the hardware observable parameters λ'^q_{ht}'s, denoting, for example, the processing capacity of the hardware node, that is suggested by one of the chosen adaptation plans or that could be one of the additional actions of the model, where the service is deployed after that the adaptation phase. Finally, $\bar{\theta}^q_{kh}$ depends also on features $\bar{\Lambda}^q_{hw}$'s indicating, for example, the probability of failure of the service.

Let $G_{kq} : \Re^n \times \Re^{|NewS|} \rightarrow \Re$, with $1 \leq q \leq Q$, be the function that computes the q-th quality attribute of the k-th external service as a function of the same attribute of each existing/new service. And let us assume (without loss of generality) that, for each quality attribute, a threshold value Θ^q has been required as a lower bound.

Let *Cost* represent the cost function of the whole system depending on the costs of the existing/new services that we have represented as θ_i^0 ($\bar{\theta}_i^0$). Different cost models could be used to define *Cost*, such as it could be as a function of the cost to replace the existing services, the one to introduce new services and the one for integrating services. For the sake of readability, we introduce here a formulation without correlation between θ_i^0 ($\bar{\theta}_h^0$) and the software (hardware) observable parameters. In fact, for example, θ_i^0 ($\bar{\theta}_h^0$) could be a function of the price charged for each invocation (measured in money per invocation) of the service s_{ij} ($news_h$).

We can summarize the formulation of our optimization model as follows:

$$
\begin{aligned}
&minCost(\theta_1^0,\ldots,\theta_n^0,\bar{\theta}_1^0,\ldots,\bar{\theta}_{|NewS|}^0)\\
&\sum_{k\in K} pexec_k \cdot G_{kq}(\theta_{k1}^q,\ldots,\theta_{kn}^q,\bar{\theta}_{k1}^q,\ldots,\bar{\theta}_{k|NewS|}^q) \geq \Theta^q\\
&\forall q = 1\ldots Q\\
&\text{Other constraints(e.g., equations}\\
&\text{to predict } \theta_{ki}^q\text{'s and } \bar{\theta}_{kh}^q\text{'s)}
\end{aligned}
$$

5. An instantiation of the model. An example of optimization model instantiation can be found in our previous work [12], where an optimization model suggests how to change the (structural and behavioral) software architecture in order to minimize the maintenance cost while guaranteeing a required level of software reliability. In the following we discuss the cost and the reliability function used in [12] in order to show an example of application of the functions *Cost* and G_{kq}.

The function *Cost* to be minimized, as the sum of the costs of all the instances selected for the existing services and the ones for introducing new services is given by: $Cost = \sum_{i=1}^{n}\sum_{j=1}^{|Avail_i|} c_{ij}x_{ij} + \sum_{h=1}^{|NewS|} \bar{c_h}z_h$, where $\theta_i^0 = \sum_{j=1}^{|Avail_i|} c_{ij}x_{ij}$ and c_{ij} is the cost of the j-th instance available for the service i; $\bar{\theta}_h^0 = \bar{c_h}z_h$ and $\bar{c_h}$ is the cost to adopt the h-th new service into the system. In [12] we provide suggestions for estimating the parameters c_{ij} and $\bar{c_h}$.

Since we assume that for each external service provided by the system we dispose of a Sequence Diagram, the reliability of the k-th external service G_{kq} can be expressed as follows:

$$G_{kq} = \prod_{i=1}^{n}\Big(\sum_{j=1}^{|Avail_i|} x_{ij}(1-\Lambda_{ij1}^q)^{\lambda_{i1}^q}\Big) \cdot \prod_{h=1}^{|NewS|}(1-\bar{\Lambda}_{h1}^q z_h)^{\lambda_{h1}^q}$$

G_{kq} is function of the software parameter observable λ_{i1}^q equals to the number of busy periods that the existing service i shows in the Sequence Diagram k after the adaptation phase (i.e., after the application of the set of adaptation plans), and $\lambda_{h1}^{\prime q}$ equals to the number of busy periods that the new service h shows in the Sequence Diagram k after the adaptation phase.
The parameter λ_{i1}^q can be expressed as follows:

$$\lambda_{i1}^q = bp_{ki} + \sum_{r=1}^{m}\sum_{p\in AP_r} VBP_p(i,k)\cdot y_{rp}$$

where m is the number of new requirements claimed by the user (alerts raised by the *Monitor* about the violation of non-functional constraints). λ_{i1}^q depends on the added/canceled busy periods $VBP_p(i,k)$ [8] that the chosen adaptation plans for the adaptation requirements suggest for the service i within the k-th external service. bp_{ki} is the number of busy periods that the service i shows in the Sequence Diagram k before the adaptation phase.
The parameter λ_{h1}^q can be expressed as follows:

$$\lambda_{h1}^q = \sum_{r=1}^{m}\sum_{p\in AP_r} BP_p(h,k)\cdot y_{rp}$$

λ_{h1}^q depends on the number of busy periods $BP_p(h,k)$ that the chosen adaptation plans suggest for the h-th

[8] VBP_p is a $n \times K$ matrix, where an element $VBP_p(i,k)$ represents the variation of the number of busy periods of the i-th existing service in the k-th external service. Note that if p suggests to remove interactions then $VBP_p(i,k)$ is a negative number.

TABLE 5.1
Parameters of the available instances for the existing services.

Service ID	Service alternatives	Cost c_{ij}	Prob. of fail. on demand θ_{ij}	Num. of busy in scen. $\bar{k}$ $bp_{\bar{k}i}$
s_1	s_{11}	8	0.008	1
	s_{12}	10	0.0009	
	s_{13}	14	0.00001	
s_2	s_{21}	5	0.004	3
	s_{22}	10	0.0002	
s_3	s_{31}	7	0.008	2
	s_{32}	10	0.0004	
s_4	s_{41}	6	0.009	2
	s_{42}	11	0.008	
	s_{43}	13	0.0001	
s_5	s_{51}	7	0.008	2
	s_{52}	10	0.0001	
	s_{53}	12	0.000003	

TABLE 5.2
Parameters of the new services available.

Service ID	Cost $\bar{c_i}$	Prob. of fail. on demand $\bar{\theta_i}$
$news_1$	7	0.00003
$news_2$	5	0.000002
$news_3$	4	0.00006
$news_4$	7	0.005

new service within the k-th external service.
Finally, G_{kq} depends also on the probability of failure on demand Λ^q_{ij1} of the j-th instance available for the service i and the probability of failure on demand $\bar{\Lambda}^q_{h1}$ of the new service h.

An example of application of the model

In order to show the practical usage of our optimization model we have applied the optimization model to the example considered in Section 4.1. We have solved the optimization model with respect to different configurations of the system for multiple values of the reliability bound R. For the sake of space we do not detail the results of all experiments that we have performed. We have observed several aspects that our model is able to capture and quantify, such as the choice to either keeping or replacing a service and how it helps to combine (and, in general, to reason about) the decisions to be taken for each requirement. A deeper experimentation of the application of an instantiation of the optimization model for a smartphone mobile application can be found in [12].

In the following we report a sample of configuration of the system for the example considered in Section 4.1.

Table 5.1 shows the parameter values of the available instances for the existing services, likewise Table 5.2 does for the new services that can be adopted into the system.

The second column of Table 5.1 lists, for each existing service, the set of alternatives. For each alternative: the buying cost c_{ij} (in KiloEuros, KE) is given in the third column, the probability of failure on demand θ_{ij} is given in the fourth column, the number of busy periods $bp_{\bar{k}i}$ that the service i shows in the scenario $\bar{k}$ is given in the fifth column.

The first column of Table 5.2 lists the new services available. For each new service: the buying cost $\bar{c_i}$ (in KiloEuros, KE) is given in the second column and the probability of failure on demand $\bar{\theta_i}$ is given in the third column.

Tables 5.3 and 5.4 summarize how the adaptation plans available for each requirement suggest to change the system behavior, i.e., to remove(introduce) interaction(s) between existing units and between these latter and new units, with respect to the external service $\bar{k}$.

The first column of Table 5.3 lists the existing services. For each service i: the variation of the number of busy periods $VBP_p(i, \bar{k})$ that the adaptation plans available for the first requirement (i.e., ap_{11} and ap_{12}) suggest for the service i in the scenario $\bar{k}$ is given in the third column; the variation of the number of busy periods $VBP_p(i, \bar{k})$ that the adapation plans available for the second requirement (i.e., ap_{21}, ap_{22} and ap_{23}) suggest for the service i in the scenario $\bar{k}$ is given in the fifth column.

The first column of Table 5.4 lists the name of the new services available. For each service h: the number of

TABLE 5.3
Variation of the number of busy periods of the existing services with respect to the adapation plans.

Service ID	Adaptation Plan ID ap_{1p}	Value for the scenario $\bar{k}$ $VBP_p(i,\bar{k})$	Adaptation Plan ID ap_{2p}	Value for the scenario $\bar{k}$ $VBP_p(i,\bar{k})$
s_1	ap_{11}	0	ap_{21}	0
	ap_{12}	0	ap_{22}	0
			ap_{23}	0
s_2	ap_{11}	0	ap_{21}	0
	ap_{12}	0	ap_{22}	0
			ap_{23}	0
s_3	ap_{11}	0	ap_{21}	0
	ap_{12}	0	ap_{22}	0
			ap_{23}	0
s_4	ap_{11}	-1	ap_{21}	0
	ap_{12}	1	ap_{22}	0
			ap_{23}	0
s_5	ap_{11}	-1	ap_{21}	0
	ap_{12}	0	ap_{22}	-1
			ap_{23}	1

TABLE 5.4
Number of busy periods of the new services with respect to the adaptation plans.

Service ID	Adaptation Plan ID ap_{1p}	Value for the scen. $\bar{k}$ $BP_p(h,\bar{k})$	Adaptation Plan ID ap_{2p}	Value for the scen. $\bar{k}$ $BP_p(h,\bar{k})$
$news_1$	ap_{11}	0	ap_{21}	0
	ap_{12}	0	ap_{22}	1
			ap_{23}	0
$news_2$	ap_{11}	0	ap_{21}	0
	ap_{12}	1	ap_{22}	0
			ap_{23}	1
$news_3$	ap_{11}	0	ap_{21}	0
	ap_{12}	0	ap_{22}	0
			ap_{23}	0
$news_4$	ap_{11}	0	ap_{21}	0
	ap_{12}	0	ap_{22}	0
			ap_{23}	0

busy periods $BP_p(h,\bar{k})$ that the adaptation plans available for the first requirement (i.e., ap_{11} and ap_{12}) suggest for the service h in the scenario $\bar{k}$ is given in the third column; the number of busy periods $BP_p(h,\bar{k})$ that the adaptation plans available for the second requirement (i.e., ap_{21}, ap_{22} and ap_{23}) suggest for the service h in the scenario $\bar{k}$ is given in the fifth column.

We have solved the optimization model by varying the reliability bound R. The results highlight, in general, that the adaptation cost increases when higher reliability thresholds have to be guaranteed.

For example, if $R = 0.97$ the model provides the following solution [s_{12}, s_{22}, s_{31}, s_{42}, s_{51}] [$news_1$], [ap_{11}, ap_{22}]. This means that, in order to achieve the optimal cost of adaptation the following plans have to be adopted: the adaptation plan ap_{11} for the first requirement and the adaptation plan ap_{22} for the second requirement. In addition, all the replacements of existing services (i.e., s_{ij}) are specified, as well as the adoption of the new service $news_1$ is claimed. The adaptation cost is equal to 52 KE, whereas the system reliability is equal to 0.974698.

If we would increase $R = 0.98$ the model provides the following solution [s_{12}, s_{21}, s_{32}, s_{43}, s_{52}] [$news_2$], [ap_{12}, ap_{21}]. In order to satisfy the reliability constraint for R = 0.98 a different solution, which is also more expensive, is suggested. In fact, the adaptation cost is equal to 53 KE, whereas the system reliability is equal to 0.985874. The two solutions differ also for the instances selected for the existing services and the adaptation plans suggested for the requirements. If $R = 0.97$ the model suggests to include $news_1$, whereas if $R = 0.98$ the new service $news_2$. The introduction of either $news_1$ or $news_2$, following the plans suggested by the solutions, would involve different modifications on the structure and the behavior of the system. For example, by applying the solution for $R = 0.97$ the existing service s_5 would not be used any more after the adaptation phase, whereas by applying the solution for $R = 0.98$ the service s_5 would be replaced by its second available instance (i.e., s_{52}).

Figures 5.1 and 5.2, respectively, show how the static and dynamic structure of the system with respect to the external service $\bar{k}$, respectively, change after the application of the adaptation actions suggested by the solution for $R = 0.97$ and $R = 0.98$. In the figures we have marked as bold the modifications brought after the

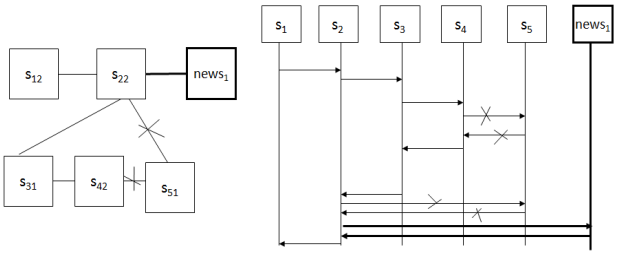

FIG. 5.1. *Resulting static structure and a SD of the system after the application of* ap_{11}, ap_{22}

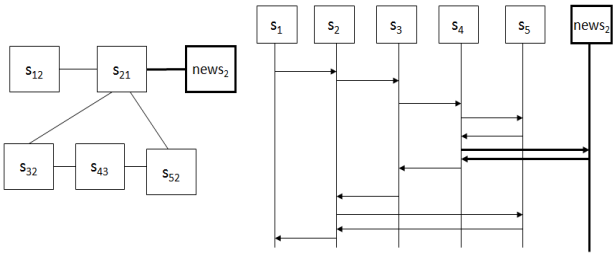

FIG. 5.2. *Resulting static structure and a SD of the system after the application of* ap_{12}, ap_{21}

application of the plans and we have put a cross on the interactions (i.e., messages in the Sequence Diagram and dependencies between services) that are removed.

Finally, if $R = 0.99$ the model provides the following solution [s_{13}, s_{22}, s_{32}, s_{43}, s_{51}] [$news_1$, $news_2$], [ap_{12}, ap_{22}]. The adaptation cost is equal to 66 KE, whereas the system reliability is equal to 0.990273. It is necessary to increase the cost from 53 KE to 66 KE in order to satisfy the reliability constraint. Note that in this case the model suggest to add both the new service $news_1$ and the new service $news_2$, whereas for $R = 0.97$ and $R = 0.98$, $news_1$ and $news_2$, respectively. The introduction of both $news_1$ or $news_2$, following the plans suggested by the solutions, would involve different modifications to the structure and behavior of the system. Figure 4.1 shows how the static and dynamic structure of the system with respect to the external service $\bar{k}$ change after the application of the adaptation actions suggested by such a solution.

Note that one of two requirement req_r claimed by the user could claim to modify the system in order to modify (introduce) one or more non-functional requirements at the same time, e.g., to improve the response time and the availability of the system. Therefore the model can be used to evaluate the tradeoff among the system reliability and other non-functional requirements (e.g. response time and availability).

6. Conclusion. We have presented a framework, based on an optimization model, that dynamically adapts both the software and hardware features of a service based system while minimizing the adaptation costs and guaranteeing a required level of the system qualities. Adaptation actions can be triggered both by a user request and/or automatically after the runtime violation of system quality constraints, or the appearing/disappearing of services into the environment. In addition, we have provided an example of instantiation together with a numerical experimentation. Experiments show that the use of a the proposed method is very helpful to make decisions during the activity of system adaptation.

We are investigating several future directions, for example, we intend to specialize our framework by enhancing it for guaranteeing specific properties of a service application domain, such as the context-aware property of the pervasive systems. In particular, we are implementing a prototype to apply our approach on realistic examples. We want to instantiate the optimization model by considering different system qualities (e.g. software reliability, availability, and response time) by enhancing our reliability model. In this direction, we also intend to take into account dependencies between different quality attributes and dependencies between services. To this extent, additional constraints may be needed, which can be expressed as *contingent decisions* [23] and, for example, the error propagation property [3] for expressing dependencies between failures of services interacting between each other could be embedded into the reliability model.

Since the adaptation of a single service may impact on the other services the propagation of the changes along the service composition should be analyzed. We also intend to leverage on several methods that have been introduced for estimating the propagation of the uncertainty of the model input parameters on the quality

system (e.g. [40])

In order to solve possible problems due, for example, to the model solution too large computation time we intend to investigate the use of the meta-heuristic techniques (e.g. the tabu-search algorithm) to improve the overall model complexity and scalability combined with the simulation techniques [16].

Other interesting research directions we intend to investigate concern the introduction/evaluation of risk factors associated to user adaptation requirements and the evaluation of dependencies among different requirements. Moreover, we intend to introduce dependencies between adaptation plans, by leveraging, for example, the contingent decisions.

REFERENCES

[1] [Online]. Available: www.lindo.com.

[2] Itu-T Recommendation Z.120 Message Sequence Charts (msc99). In *Technical report, ITU Telecommunication Standardization Sector*, 1996.

[3] W. Abdelmoez, D. M. Nassar, M. Shereshevsky, N. Gradetsky, R. Gunnalan, H. H. Ammar, B. Yu, and A. Mili. Error Propagation In Software Architectures. *Software Metrics, IEEE International Symposium on*, pages 384–393, 2004.

[4] F. André, E. Daubert, and G. Gauvrit. Towards a Generic Context-Aware Framework for Self-Adaptation of Service-Oriented Architectures. In *Proc. of the Fifth International Conference on Internet and Web Applications and Services*, 2010.

[5] J. Bang, D. Popescu, G. Edwards, N. Medvidovic, N. Kulkarni, G. Rama, and S. Padmanabhuni. Codesign: a highly extensible collaborative software modeling framework. In *ICSE '10: Proc. of the 32nd ACM/IEEE International Conference on Software Engineering*, pages 243–246, 2010.

[6] L. Baresi and S. Guinea. A dynamic and reactive approach to the supervision of bpel processes. In *ISEC '08: Proc. of the 1st conference on India software engineering conference*, pages 39–48. ACM, 2008.

[7] C. Blum and A. Roli. Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM Comput. Surv.*, 35(3):268–308, 2003.

[8] R. Calinescu and M. Kwiatkowska. Using quantitative analysis to implement autonomic IT systems. In *ICSE*, pages 100–110, 2009.

[9] G. Canfora, M. D. Penta, R. Esposito, and M. Villani. A framework for QoS-aware binding and re-binding of composite web services. *Journal of Systems and Software*, 81(10):1754–1769, 2008.

[10] V. Cardellini, E. Casalicchio, V. Grassi, F. L. Presti, and R. Mirandola. Towards Self-adaptation for Dependable Service-Oriented Systems. In *Architecting Dependable Systems VI*, pages 24–48, 2009.

[11] D. Chiu, S. Deshpande, G. Agrawal, and R. Li. A Dynamic Approach toward QoS-Aware Service Workflow Composition. In *ICWS*, pages 655–662, 2009.

[12] V. Cortellessa, R. Mirandola, and P. Potena. Selecting Optimal Maintenance Plans based on Cost/Reliability Tradeoffs for Software Subject to Structural and Behavioral Changes. In *Proc. of the 14th European Conference on Software Maintenance and Reengineering (CSMR 2010)*.

[13] E. Di Nitto, C. Ghezzi, A. Metzger, M. Papazoglou, and K. Pohl. A journey to highly dynamic, self-adaptive service-based applications. *Autom. Softw. Eng.*, 15(3-4):313–341, 2008.

[14] J. Fox and S. Clarke. Exploring approaches to dynamic adaptation. In *MAI '09: Proceedings of the 3rd International DiscCoTec Workshop on Middleware-Application Interaction*, pages 19–24, 2009.

[15] J. Garcia, D. Popescu, G. Edwards, and N. Medvidovic. Toward a Catalogue of Architectural Bad Smells. In *QoSA*, pages 146–162, 2009.

[16] F. Glover, J. Kelly, and M. Laguna. New advances for wedding optimization and simulation. volume 1, pages 255–260, 1999.

[17] H. Guo, J. Huai, H. Li, T. Deng, Y. Li, and Z. Du. ANGEL: Optimal Configuration for High Available Service Composition. *Web Services, IEEE International Conference on*, pages 280–287, 2007.

[18] J. Harney and P. Doshi. Adaptive web processes using value of changed information. In *In International Conference on Service-Oriented Computing (ICSOC)*, pages 179–190, 2006.

[19] Q. He, J. Yan, H. Jin, and Y. Yang. Adaptation of Web Service Composition Based on Workflow Patterns. In *ICSOC*, 2008.

[20] R. Hirschfeld, P. Costanza, and O. Nierstrasz. Context-oriented Programming. *Journal of Object Technology*, 7(3):125–151, 2008.

[21] N. Ibrahim and F. L. Mouël. A Survey on Service Composition Middleware in Pervasive Environments. *CoRR*, abs/0909.2183, 2009.

[22] N. Ibrahim, F. L. Mouël, and S. Frénot. MySIM: a spontaneous service integration middleware for pervasive environments. In *ICPS '09: Proceedings of the 2009 international conference on Pervasive services*, pages 1–10, 2009.

[23] H.-W. Jung and B. Choi. Optimization models for quality and cost of modular software systems. *European Journal of Operational Research*, 112(3):613 – 619, 1999.

[24] G. Kapitsaki, G. Prezerakos, N. Tselikas, and I. Venieris. Context-aware service engineering: A survey. *Journal of Systems and Software*, 82(8):1285–1297, 2009.

[25] S. Kim, D. Kim, L. Lu, and S. Park. Quality-driven architecture development using architectural tactics. *Journal of Systems and Software*, 82(8):1211–1231, 2009.

[26] P. Leitner, F. Rosenberg, and S. Dustdar. Daios: Efficient Dynamic Web Service Invocation. *IEEE Internet Computing*, 13:72–80, 2009.

[27] G. Lodi, F. Panzieri, D. Rossi, and E. Turrini. SLA-Driven Clustering of QoS-Aware Application Servers. *IEEE Trans. Software Eng.*, 33(3):186–197, 2007.
[28] A. Martens and H. Koziolek. Performance-oriented Design Space Exploration. In *Proc. of the 13th Int. Workshop on Component Oriented Programming (WCOP'08)*, 2008.
[29] M. Marzolla and R. Mirandola. Performance Prediction of Web Service Workflows. In *Proc. of third International Conference on Quality of Software Architectures, QoSA 2007*, volume 4880 of *Lecture Notes in Computer Science*, pages 127–144, 2007.
[30] R. Mirandola and P. Potena. Self-adaptation of service based systems based on cost/quality attributes tradeoffs. *Symbolic and Numeric Algorithms for Scientific Computing, International Symposium on*, pages 493–501, 2010.
[31] O. Moser, F. Rosenberg, and S. Dustdar. Non-intrusive monitoring and service adaptation for WS-BPEL. In *WWW*, pages 815–824, 2008.
[32] J. Musa. Operational profiles in software-reliability engineering. *Software, IEEE*, 10(2):14–32, Mar 1993.
[33] E. Nistor, J. Erenkrantz, S. Hendrickson, and A. van der Hoek. ArchEvol: versioning architectural-implementation relationships. In *SCM '05: Proceedings of the 12th international workshop on Software configuration management*, pages 99–111, 2005.
[34] M. Oh, J. Baik, S. Kang, and H. Choi. An Efficient Approach for QoS-Aware Service Selection Based on a Tree-Based Algorithm. In *ACIS-ICIS*, pages 605–610, 2008.
[35] M. Papazoglou, P. Traverso, S. Dustdar, and F. Leymann. Service-Oriented Computing: State of the Art and Research Challenges. *IEEE Computer*, 40(11):38–45, 2007.
[36] J. Rao and X. Su. A Survey of Automated Web Service Composition Methods. In *SWSWPC*, pages 43–54, 2004.
[37] F. Rosenberg, P. Celikovic, A. Michlmayr, P. Leitner, and S. Dustdar. An End-to-End Approach for QoS-Aware Service Composition. In *EDOC*, pages 151–160, 2009.
[38] R. Roshandel, N. Medvidovic, and L. Golubchik. A Bayesian Model for Predicting Reliability of Software Systems at the Architectural Level. In *QoSA*, pages 108–126, 2007.
[39] H. Truong and S. Dustdar. A survey on context-aware web service systems. *International Journal of Web Information Systems (IJWIS)*, 5(1):5–31, 2009.
[40] N. Wattanapongskorna and D. Coit. Fault-tolerant embedded system design and optimization considering reliability estimation uncertainty. *Reliability Engineering and System Safety*, 92:395–407, 2007.
[41] J. Zhu, C. Guo, Q. Yin, J. Bo, and Q. Wu. A Runtime-Monitoring-Based Dependable Software Construction Method. In *ICYCS*, pages 1093–1100, 2008.

Edited by: Dana Petcu and Alex Galis
Received: March 1, 2011
Accepted: March 31, 2011