The Journal of Systems and Software 86 (2013) 624–648

Contents lists available at SciVerse ScienceDirect

The Journal of Systems and Software

journal homepage: www.elsevier.com/locate/jss

Optimization of adaptation plans for a service-oriented architecture with cost, reliability, availability and performance tradeoff

Pasqualina Potena

Dipartimento di Ingegneria, Università degli Studi di Bergamo, Viale Marconi, 5, Dalmine (BG) 24044, Italy

ARTICLE INFO

Article history:
Received 31 October 2011
Received in revised form 18 October 2012
Accepted 19 October 2012
Available online 20 December 2012

Keywords:
Selecting adaptation plans
Software cost
Software reliability
Software performance
Optimization model

ABSTRACT

A service-based system may require adaptation for several reasons, such as service evolution (e.g., a new version may be available), hardware volatility (e.g., network quality changes), and varying user demands and new requirements (e.g., a new functionality or a different level of quality of service). Therefore, it is suitable to dynamically adapt a service-based system in an automated manner. However, service adaptations often do not consider software quality attributes and, if they do, they relay on a single attribute in isolation. In this paper, we present an optimization model, which aims to minimize the adaptation costs of a Service-Oriented Architecture (SOA), in correspondence with a certain change scenario (i.e., a set of new requirements) under reliability, availability and performance tradeoff. The model predicts the quality of the new SOA obtained by changing both its structure and behavior. Specifically, it suggests how to replace existing services with available instances and/or adding new services, and how to remove or introduce interaction(s) between existing services and/or new services. We show how our model works on a smartphone mobile application example, and through the sensitivity analysis we highlight its potential to drive architectural decisions.

1. Introduction

Service-based systems (SBSs) are playing an increasingly important role in application domains, ranging from research and healthcare to defence and aerospace. SBSs may require dynamic adaptation for several reasons, such as service evolution (e.g., a new version may be available), hardware volatility (e.g., network quality changes), varying user demands and new requirements (e.g., a new functionality or a different level of quality of service). For example, in the pervasive computing domain (Di Nitto et al., 2008), an application should be context-aware self-adaptive, i.e., the context, capturing information about the environment, should trigger certain adaptations, such as the switch from a Wifi network to a GSM one.

Adaptation decisions should not relay on a single attribute in isolation (which is often done), but they should be based on the right tradeoff among the system software qualities. In fact, the adaptation should be driven by the analysis to characterize the system qualities (e.g., performance and reliability) behavior of the software application. This analysis takes into consideration the behavior of the "components" and the "architecture" of the application. This analysis might be used to answer questions such as: (i) which services/service interactions are critical to the performance and reliability of the application?, (ii) how far are the application performance and reliability influenced by the performance and reliabilities of individual services? and (iii) how could a set of services be assembled to obtain a system, whose failure probability and response time fall under a certain upper bound (if the failure probability and response time of each services is known a priori)? If the software application is to be adapted from a collection of services, then giving an answer to such questions can help to make decisions, such as which services should be picked off the shelf, and which services should be embed into the system.

Several research efforts have been devoted in the last years to the definition of methods and tools able to predict and evaluate the quality of a software system (see, for example, Balsamo et al., 2004; Becker et al., 2004; WOSP, 1998–2007). Usually, their goal is to predict and/or analyze some quality attributes, like performance or reliability, starting from the architectural description of the system, or to select the architecture of the system among a finite set of candidates that better fulfill the required quality.

In this paper, we address the problem of system quality from a different point of view: starting from the description of the system and from a set of new requirements, we devise the set of actions to be accomplished to obtain a new architecture. This is able to both fulfill the new requirements with the minimum cost and guarantee given levels of reliability, availability and performance.

The goal of this paper is to provide a support to the decisions that software architects take for adapting a SOA. We introduce

E-mail addresses: pasqualina.potena@unibg.it, pasqualinapotena@libero.it

0164-1212/$ – see front matter © 2012 Elsevier Inc. All rights reserved.
http://dx.doi.org/10.1016/j.jss.2012.10.929

an optimization model, which minimizes the adaptation costs of a system in correspondence with a certain *change scenario*, while guaranteeing required levels of reliability, availability and performance. A change scenario is a set of new requirements that induce changes in the structural and behavioral architecture of the software system. Specifically, in our model, we consider as possible changes the introduction of new functionalities and the modification of the dynamics of existing functionalities. For each new requirement in a change scenario, we consider the different sets of adaptation actions (called *adaptation plans*) able to guarantee this new requirement. In this way, we obtain a set of decisions that lead to the definition of a new architecture, which minimizes the costs while keeping the reliability, availability and the response time within certain thresholds.

Any combination of adaptation actions may have a considerable impact on the cost, reliability/availability and performance of the software architecture. Therefore, our optimization model aims to quantify such impact in order to suggest the best adaptation plan, which minimizes the costs while satisfying the reliability, availability and performance constraints.

A new architecture is, thus, obtained by modifying both its structure and its behavior. Specifically, in order to modify the software structure, the model replaces existing software services with different available services and/or embeds new software services into the system. With respect to the changes in the system behavior, it modifies the system scenarios (represented, for example, as BPEL processes Andrews et al., 2003) by removing or introducing interactions between existing services and/or between existing and new services.

Our approach is not tied to any particular adaptation phase (i.e., evolution – performed at re-design time – and self-adaptation – performed at run-time) (Bucchiarone et al., 2010) of the software life-cycle. The adoption of the proposed optimization model may require to be specialized in order to capture typical aspects of a certain phase. For example, the SOA self-adaptation regards temporary modification (such as the re-execution of an unavailable service or a substitution of an unsuitable service) and permits to respond to changes in the requirements and/or in the application context. On the contrary, when changes regard critical aspects and should be applied permanently to the system, they should be considered as evolution steps.

This paper extends our previous work (Cortellessa et al., 2010), in which an optimization model suggests how to change the software structure and behavior in order to minimize the maintenance cost while guaranteeing a certain level of software reliability. The new contributions that we present in this paper can be summarized as follows: (i) this paper specializes the work in Cortellessa et al. (2010) to the software service domain; (ii) this paper considers a significant subset of operators expressing the composition of services, whereas in Cortellessa et al. (2010) we only consider the sequential execution of the software; (iii) based on these different kinds of structured activities, a new reliability model is defined; (iv) availability and performance constraints are added to the model. As a consequence of the latter contributions, the model solutions presented in this paper have to properly balance attributes that can be hardly adverse to each other, such as reliability, availability and response time.

In our previous work (Mirandola and Potena, 2010), we have provided with basic premises 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 run-time violation of system quality constraints, or appearing/disappearing of services into the environment. In this paper, we instantiate the optimization model presented in Mirandola and Potena (2010), in which a general formulation has been provided by considering the system adaptation cost, reliability, availability and response time.

The paper is organized as follows: in Section 2 we provide the formulation of the optimization model that represents the core of our approach; in Section 3 we apply the model to an example; in Section 4 we analyze the sensitivity of the model to changes of its parameters; Section 5 introduces related works and discusses the novelty of our contribution; conclusions are presented in Section 6.

2. Optimization model formulation

In this section, we introduce the formulation of our model aimed to find the optimal set of adaptation actions needed to tackle required changes to the software architecture. "Optimal" here denotes actions that incur in a minimum cost while guaranteeing certain levels of reliability, availability and performance.

Definition 1. [Elementary Software Service] Let S be a SOA composed by n elementary software services. Let s_i be the i-th existing elementary service ($1 \leq i \leq n$).

Since our model may support different service application domains, 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.

Definition 2. [External Service] Through the composition of its elementary software services, the system offers services (that we name as "external") to users. Let us assume that the composite service structure of each external service is specified by using BPEL (Andrews et al., 2003). Let us also assume that we allot BPEL specifications for those external services that are not active in the current system implementation, but they can be activated to satisfy new requirements.

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

Let λ_k be the arrival rate of calls for the k-th external service provided by the system. Therefore, the probability that the k-th system external service will be invoked is $\Lambda_k = \lambda_k / \sum_{i=1}^{K} \lambda_i$. It is obvious that for a non-active external service k we have $\lambda_k = 0$. This information can be synthesized from the operational profile (Musa, 1993).[1]

2.1. Modeling a service

As in Cardellini et al. (2007), in this paper, we refer to a significant subset of the whole BPEL definition, focusing on its structured style of modeling (rather than on its graph-based one, thus omitting to consider the use of control links). Specifically, besides the primitive "invoke activity", which specifies the synchronous or asynchronous invocation of a Web service, we consider the following kinds of *structured activities*[2]:

[1] Such assumption might be not realistic in cases where the operational profile may be not (fully) available. However, in such cases the domain knowledge and the information provided by the SOA could be used for estimating it, as suggested for example in Roshandel et al. (2007).

[2] We do not consider all the possible structured activities, but we include a significant subset. In particular, we take into account the workflow patterns 1, 2+3, 4, 10 (for structured cycles only), 13 and 16 reported in Van Der Aalst et al. (2003).

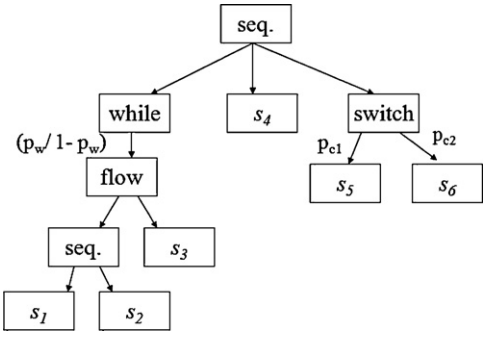

Fig. 1. An example of activity tree.

- *Sequence*: Sequential execution of activities;
- *Switch*: Conditional execution of activities;
- *While*: Repeated execution of activities in a loop;
- *Flow*: Concurrent execution of activities.

Let $T_k = (V_k, E_k)$ a tree representing the k-th BPEL process ($1 \leq k \leq K$). In T_k, the nodes V_k are the activities in the BPEL code, while the edges E_k reflect the relationships among the BPEL process activities. For the sake of simplicity (in the following), we will interchangeably speak of activity i and node i. Therefore, we associate a structured activity with an internal node $i \in V_k$. This last node is labeled $l \in \{seq., switch, while, flow\}$, where the symbol *seq.* denotes (for brevity) the sequential execution. Similarly, we associate a primitive *invoke* activity, which we correspond to an elementary service invocation,[3] with a leaf node $i \in V_k$.

Definition 3. [Probability p_a of Executing Activity a] Let p_a denote the probability of executing activity a in a structured activity (e.g., $p_w/1-p_w$ is the probability to repeat the activities in the *while* statement and p_{ci} is the probability to execute the activity i-th in the *switch* statement). In Cardellini et al. (2007) details on the estimation of the parameter p_a can be found.

An example of activity tree Fig. 1 shows an example of process activity tree for an external service provided by the interaction of six elementary software services. We have associated the short names with the elementary services as illustrated in Fig. 1 (i.e., s_1, $s_2, \ldots, s_6$).

Definition 4. ["Direct Descendant" Node] We will write $i \prec l$ if node i is a descendant of node l. We will say that a node $l \in T_k$ is a *direct descendant* of $l' \in T_k$, denoted by $l \preceq_{dd} l'$, if $l \prec l'$ and for any other node $l'' \in T_k$, $l \prec l'' \prec l'$ implies $l'' \neq flow$, i.e., there is no node labeled *flow* in the path from l to l'.

Definition 5. [Set of Node of Type "Flow" and root node] Let $F_k \subset V_k$ denote the set of nodes corresponding to *flow* activities. Let w represent the root node of the k-th BPEL tree.

2.2. Modeling adaptations to changes

Let *cs* be a *change scenario*, namely a set of m new requirements to be satisfied. Let req_r be the r-th element of this set, with ($1 \leq r \leq m$) that we will call "change requirement". In this paper, we only consider two types of new requirements, that are: (i) introducing a new external service, (ii) modifying the dynamics of an existing external service.

Adaptation actions have to be performed to satisfy new requirements. An *adaptation plan* is a set of actions that address a certain requirement. Obviously, several adaptation plans may be available for each requirement. Since they suggest different adaptation actions, they may differ for adaptation cost and/or for the system reliability, availability and performance attained.

Let AP_r be the set of adaptation plans available for the r-th requirement in *cs*.[4] For example, in order to introduce a new external service, one of the services (i.e., BPEL processes) that are not active must be activated. Namely, it enters the set of BPEL processes that contribute to the costs, reliability, etc. of the whole SOA. To modify the dynamics of an existing external service, certain interactions of the process related to the external service have to be added/removed.

In order to keep the modeling aspects of our approach as simple as possible, we assume that the optimization model gets as input values adaptation plans able to deal with each requirement independent from each other (i.e., changes claimed by a plan do not compromise the application of other plans). To this extent, we assume that plans of different change requirements modify different paths of the k-th BPEL tree (i.e., a path from the root to a leaf is modified only by plans related to a certain requirement). Besides, we assume that the modifications brought in by a plan with respect to an *internal node*, which may be shared by multiple paths, do not compromise the application of the plans related to other requirements. For example, if a plan changes the probability p_a of a *switch* node, all paths including this node cannot be modified by plans related to other change requirements.

In order to keep the modeling aspects of our work as simple as possible, we take into account only software adaptation actions. The model can be enhanced, for example, by envisaging hardware actions, such as the introduction of a new server. Besides, we do not focus on corrective maintenance actions, such as bug fixing or exception handling.

Different adaptation actions can be used depending on several factors due mainly to the particular adaptation phase where the model is adopted (e.g., if run-time modifications are claimed, then it is only required the substitution of an unsuitable service without using more sophisticated actions). A user (or "system designer" or "system maintainer") may define adaptation plans for each new requirement at re-design time. On the contrary, for example, in case of self-adaptation (at run-time), if the system gets alerts about the violation of non-functional requirements, it itself can generate adaptation plans[5] (as done by our framework (Mirandola and Potena, 2010) discussed in Section 1).

In this paper, we consider the following adaptation activities:

1. *Introducing new software services*: An adaptation action may suggest to embed into the system one or more new elementary software services.[6] A system maintainer, for example, could picked off the environment a new service in order to apply a recurring software design solution (whose goal is the improvement of one specific design concern of a quality attribute), or introduce a new system functionality. We call *NewS* the set of new available services that can provide different functionalities, whereas $news_h$ represents the h-th service of *NewS*. This action is realized by introducing one or more leaf nodes in the BPEL trees.
2. *Replacing existing service instances with functionally equivalent ones*: An adaptation action may suggest to replace a service with

[3] An elementary service may be associated with more than one leaf node, because a service may be involved in different activities.

[4] In the remainder of the paper, a plan p ($1 \leq p \leq |AP_r|$) is also called ap_{rp}.

[5] Details on the generation of adaptation plans can be found in Section 5. However, for example, in order to address issues pertaining to quality attributes, recurring software design solutions (such as architectural tactics) could be used.

[6] Notice that such type of action has to be associated with another action that indicates how new software services interact with existing ones, therefore it modifies the interactions within certain external services (see type action 3).

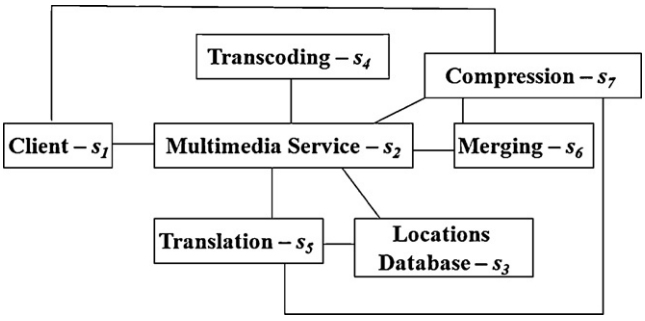

Fig. 2. Software architecture of the smartphone application example.

one of additional instances available for it. 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 the services provided by s_i and require at most all the services required by s_i). However, in Section 2.4 we discuss how to relax such an assumption by introducing integration/adaptation costs. Besides, instances of the same service may differ from each other for cost, response time, reliability and availability characteristics. We call $Inst_i$ the set of instances for s_i, while s_{ij} represents the j-th instance of $Inst_i$.

3. *Modifying the interactions among services in a certain external service*: An adaptation action may suggest to modify the system dynamics by introducing/removing interactions among services within a certain external service. This action is performed by introducing (deleting) leafs and structured nodes in the BPEL tree representing the external service. Besides, a plan may also modify the probability p_a to execute the activity a of an existing structured node of *switch* or *while* type.

It is obvious that any combination of such adaptation actions may have a considerable impact on the cost, performance, reliability and availability of the SOA. Therefore, our optimization model aims to quantify such impact to suggest the best adaptation plan, which still minimizes the costs while satisfying the performance, reliability and availability constraints.

Finally, in order to open the solution space to additional possibilities, our model allows to combine adaptation plans with additional service replacement actions. In other words, in the optimization model we leave to the solver the possibility to choose additional single replacement actions that have not been embedded in any selected plan.

Table 1
Change scenario.

Requirement ID	Requirement specification
req_1	The adaptation of the text and the geographical map to the smartphone should be also performed in a different way
req_2	The video should be also provided in AVI format
req_3	The map should provide additional information, such as a list of the closer hotels or restaurants

2.3. An example of output of the model

The smartphone mobile application: A smartphone user can require the latest news from a service provider (here called *Multimedia Service*). The news include text and topical videos available in MPEG2 format. Besides, we assume that the smartphone can require to the *Multimedia Service* a geographical map showing its locations (Alrifai and Risse, 2009). Fig. 2 illustrates the software architecture of the system. This latter describes the dependencies between two elementary services of the system (i.e., one line ties two services if they interact with each other). As shown in Fig. 2, we have associated the IDs with the services as follows: s_1 to *Client*, s_2 to *Multimedia Service*, s_3 to *Locations Database*, s_4 to *Transcoding*, s_5 to *Translation*, s_6 to *Merging* and s_7 to *Compression*.

It is a client/server system, where the *Client* is connected via wireless network to the *Multimedia Service*. The latter uses other services for sending news to the client. In particular, the service *Transcoding* that adapts the video content to the smartphone format, the service *Compression* that adapts the news content to the wireless link, the service *Translation* that adapts the text to the smartphone format and draws the geographical map, and the service *Merging* that integrates the text with the video stream for the limited smartphone display. In order to provide a map showing the user location, the *Multimedia Service* also interacts with the *Locations Database* to collect information about the localization of cells.

We consider three functionalities of the system, whose BPEL trees are represented in Figs. 3–5, respectively. The first one allows requiring news in textual format, with the second one it is possible to require news with both textual and video content; finally, the third one permits to have a geographical map with the user location. We have derived the BPEL trees from the BPEL specifications of the functionalities (as explained in Section 2.1). Therefore, the leaf nodes of a tree are associated with the elementary services (i.e., *Client*, *Multimedia Service*, *Locations Database*, *Transcoding*, *Translation*, *Merging* and *Compression*), while its

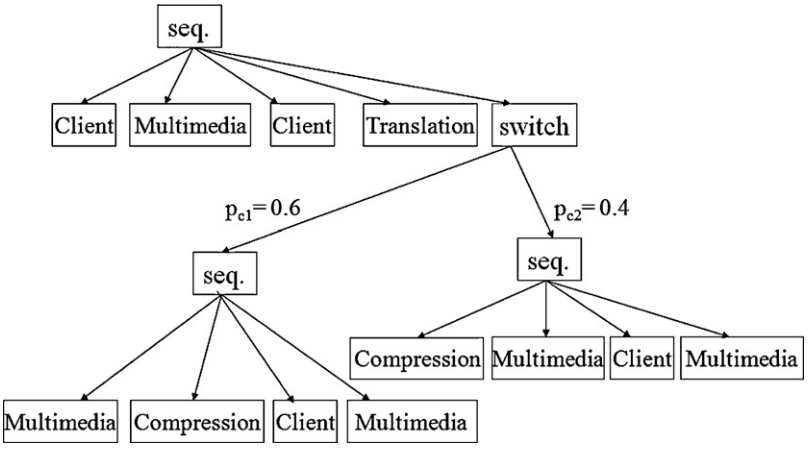

Fig. 3. BPEL tree of the external service "Requiring news in textual format".

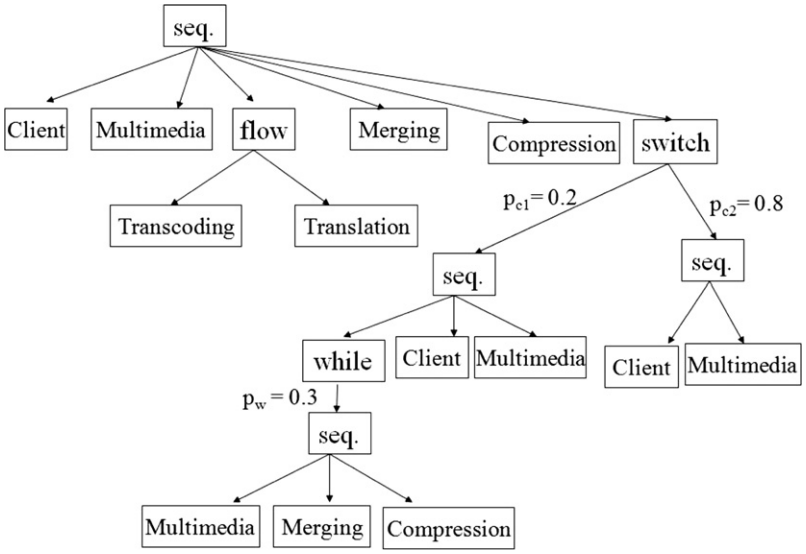

Fig. 4. BPEL tree of the external service "Requiring news with text and video".

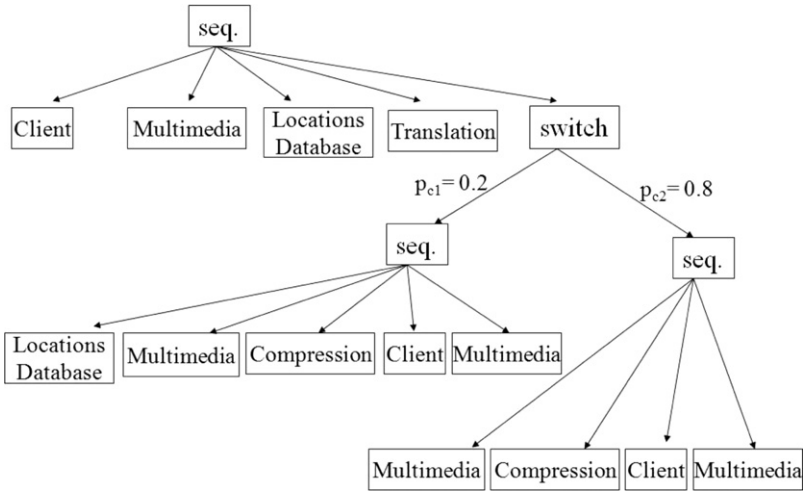

Fig. 5. BPEL tree of the external service "Requiring a map of location".

Table 2
Adaptation plans available to satisfy the requirements of the change scenario.

Requirement ID	Adaptation plan ID	Adaptation plan description
req_1	ap_{11}	Adding a new service $news_1$ that interacts with the service Translation (i.e., s_5)
	ap_{12}	Adding a new service $news_4$ that interacts both with Translation and Client AND Replacing Client (i.e., s_1) with its second instance available
req_2	ap_{21}	Replacing the service Transcoding (i.e., s_4) with its second instance available
	ap_{22}	Adding a new service $news_3$
	ap_{23}	Adding a new service $news_4$
req_3	ap_{31}	Adding a new service $news_2$
	ap_{32}	Adding a new service $news_1$ AND Replacing Locations Database (i.e., s_3) with its first instance available
	ap_{33}	Replacing Multimedia Service (i.e., s_2) with its second instance available

internal nodes are associated with structured activities (labeled $l \in \{seq., switch, while, flow\}$).

An Example of Change Scenario: Table 1 illustrates an example of change scenario. It is composed by three new requirements to

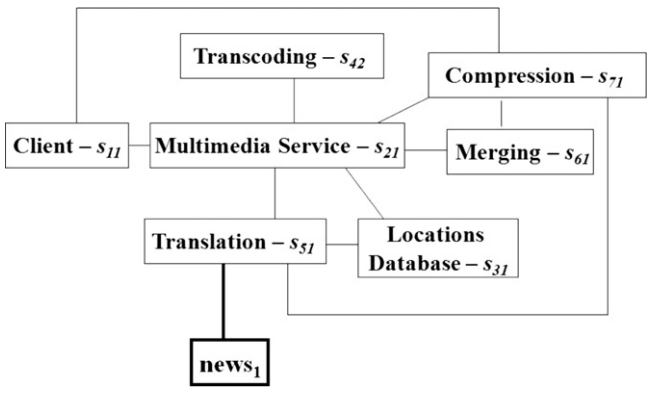

Fig. 6. Static aspect of the system after the application of ap_{11}, ap_{21} and ap_{32}.

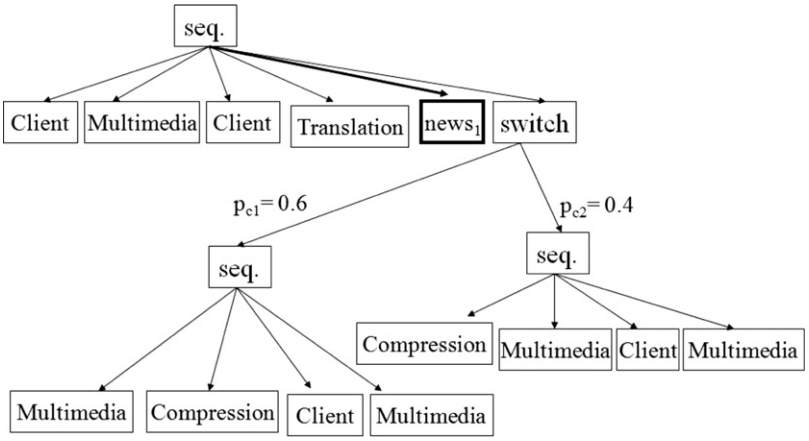

Fig. 7. BPEL tree of Fig. 3 after the application of ap_{11}, ap_{21} and ap_{32}.

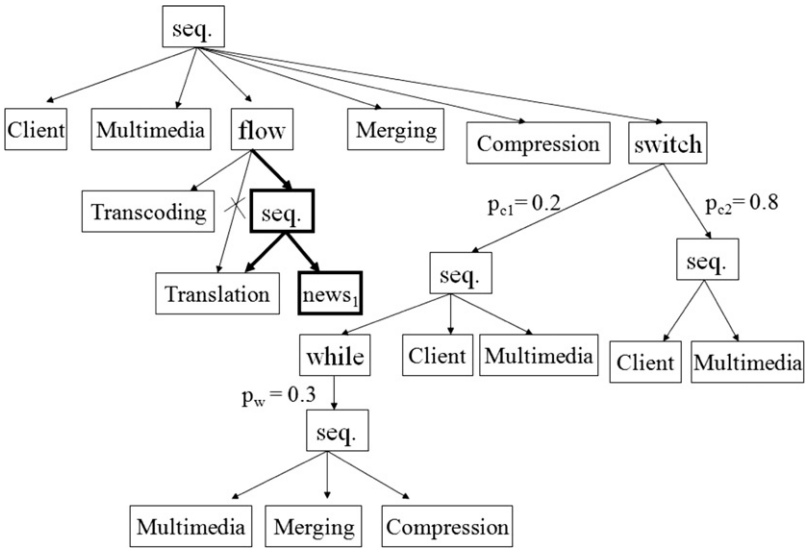

Fig. 8. BPEL tree of Fig. 4 after the application of ap_{11}, ap_{21} and ap_{32}.

be satisfied. For the sake of simplicity, we have used the natural language to define the requirements and the adaptation plans. In Table 2, for each requirement, we provide a set of adaptation plans that can be adopted to satisfy the requirement. For example, req_1 "The adaptation of the text and the geographical map for the smartphone should be also performed in a different way" can be managed by either "Adding a new service $news_1$"(ap_{11}) or "Adding a new service $news_4$ AND Replacing *Client* with its second instance available"(ap_{12}).

An example of output of the model: Let us suppose that to achieve the optimal cost of adaptation, and assure required levels of system reliability and response time, the optimization model suggests to adopt the following plans: adaptation plan ap_{11} for the first requirement, adaptation plan ap_{21} for the second requirement and adaptation plan ap_{32} for the third 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 services would change as shown in Figs. 6–9, respectively. In the figures, we have marked in bold the modifications brought in after the application of the plans and we have put a cross on the interactions that have been removed.

In particular, the model suggests the replacements of existing services (i.e., s_{ij}), as well as the adoption of the new service $news_1$.

2.4. Model parameters

In this section, we describe the parameters of our optimization model.

2.4.1. Parameters of (existing and new) services

The main parameters[7] that we define for an existing elementary service s_i $(1 \leq i \leq n)$ are:

- The paths parameters (P_{ki}, INV_{ki}) in the BPEL tree k.
  P_{ki} represents the number of paths of the k-th BPEL process from the root to the service i.
  INV_{ki} is a vector of P_{ki} elements, where an entry $INV_{ki}(t)$ is equal to the expected number of times that the service i is invoked in

[7] For the sake of readability, other parameters are given and used in Appendix.

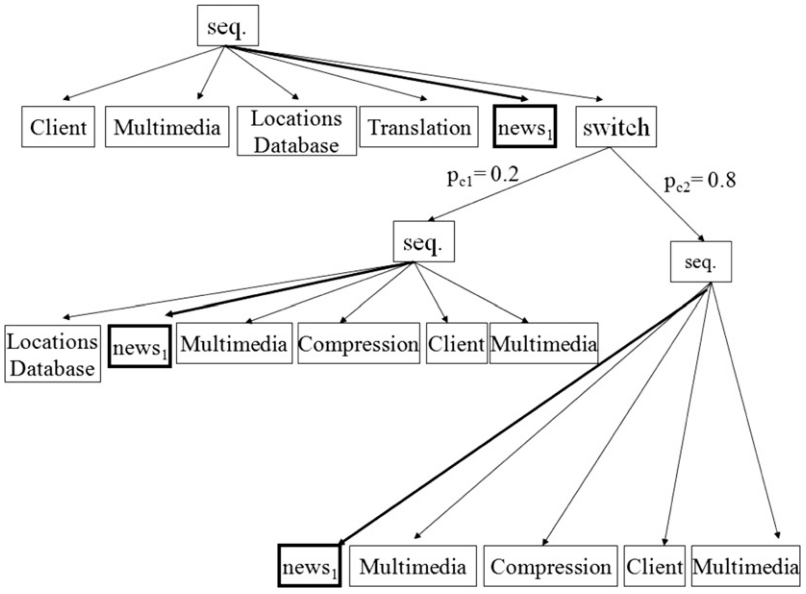

Fig. 9. BPEL tree of Fig. 5 after the application of ap_{11}, ap_{21} and ap_{32}.

the path t. It can be easily estimated by parsing the path t and multiplying the p_a labels by the arcs along the path.

- The cost c_{ij} of its j-th instance $s_{ij} \in Inst_i$ (for example, measured in kiloeuros, ke).

  Although the estimate of c_{ij} is outside the scope of our work, we remark that it may be estimated in different ways. It could be estimated, for example, as a function of the price charged for using the service (see, e.g., Cardellini et al., 2007 for details) and the one for integrating (adapting) the instance j into the system. In Cortellessa et al. (2010) we provide suggestions to calculate c_{ij}.[8]
- The quality attribute q_{ij}, $q \in \{r, a\}$ where r_{ij} (a_{ij}) is the reliability (availability) on demand (Trivedi, 2001) of its j-th instance $s_{ij} \in Inst_i$.

  Suggestions to estimate q_{ij} can be found in Cortellessa et al. (2006), such as a rough upper bound $1/N_{nf}$ of the probability of failure of a software service can be obtained when observed that s_{ij} has been invoked for a N_{nf} number of times with no failures.
- The response time rt_{ij} of its j-th instance $s_{ij} \in Inst_i$, which is the interval of time elapsed from the invocation to the completion of the service s_{ij} (for example, measured in seconds, s). It can be estimated by monitoring the service instance, as well as other characteristics (such as reliability and availability) of s_{ij} can be collected by using the same techniques.

The parameters that we define for a new service $news_h \in NewS$ are:

- the cost $\overline{c}_h$ (for example, measured in kiloeuros);
- the quality attribute $\overline{q}_h$, $\overline{q} \in \{\overline{r}, \overline{a}\}$ where $\overline{r}_h$ ($\overline{a}_h$) is the reliability (availability) on demand;
- the response time $\overline{rt}_h$ (for example, measured in seconds).

These parameters can be estimated by using the same techniques described for existing services.

For the sake of model linearity, as in Zeng et al. (2004), when writing expressions, we will consider the logarithm of the reliability[9] (availability) rather than the reliability (availability) itself. The work in Zeng et al. (2004) – supporting the web service selection – uses the logarithm function in order to obtain linear expressions when composing the reliability (availability) of different services.

Notice that we define the parameters of an existing (new) service as average values of the values of their provided services. The parameters could be refined with respect to the services without essentially changing the overall model structure.

2.4.2. Parameters of adaptation plans

The main parameters[10] that we define for an adaptation plan $ap_{rp} \in AP_r$, with respect to the k-th BPEL tree ($1 \leq k \leq K$), are:

- parameters of existing services $VERS_p$;
- parameters of new services (NS_p, NP_p, $NINV_{kp}$);
- parameters of existing paths (ΔINV_{kp}, Δ_{kp});
- parameters of new paths (ADP_{kp}, $AINV_{kp}$).

For the sake of readability, details of these parameters can be found in Table 3.

2.5. Model variables and elementary constraints

In this section, we describe the variables and the constraints on the variables of our optimization model.

The following variables allow to select an adaptation plan for each requirement of a change scenario.

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

[8] As we remarked in Cortellessa et al. (2010), the introduction of integration/adaptation costs, although increasing the model complexity, would permit to relax our assumption that the instances available for a service are functionally equivalent to each other. Besides, it could be exploited for supporting corrective maintenance activities (e.g., bug fixing or exception handling).

[9] This attribute is called *successful execution rate* in Zeng et al. (2004).

[10] For the sake of readability, other parameters are given and used in Appendix.

Table 3
Parameters of adaptation plans.

Parameter ID	Parameter specification
$VERS_p$	$n \times \max_i \lvert Inst_i \rvert$ matrix, where an element $VERS_p(i,j)$ (also called $[v_{ij}]_p$) is equal to 1 if the plan p suggests to update s_i with its j-th available instance, and it is equal to 0 otherwise. It obviously holds: $[v_{ij}]_p = 0\ \forall j > \lvert Inst_i \rvert$.
NS_p	(Possibly empty) Set of new services to be introduced due to some adaptation actions ($NS_p \subseteq NewS$).
NP_p	$\lvert NS_p \rvert \times K$ matrix, where an element $NP_p(h,k)$ is the number of paths that the plan p suggests to introduce in the k-th BPEL process from the root to the new service $news_h$.
$NINV_{kp}$	$\lvert NS_p \rvert \times \max_h NP_p(h,k)$ matrix, where an element $NINV_{kp}(h,t)$ represents the expected number of times that the service h is invoked in the path t. It obviously holds: $NINV_{kp}(h,t) = 0\ \forall t > NP_p(h,k)$.
ΔINV_{kp}	$n \times \max_i P_{ki}$ matrix, where an element $\Delta INV_{kp}(i,t)$ represents the number of invocations of the service s_i along the t-th path in the BPEL k after the application of the plan p.
Δ_{kp}	Matrix of $n \times \max_i P_{ki}$ elements, where the element $\Delta_{kp}(i,t)$ is equal to 1 if the plan p suggests to change the path t of the service s_i in the BPEL k, and it is equal to 0 otherwise.
ADP_{kp}	Vector of n elements, where an element $ADP_{kp}(i)$ is the number of new paths introduced from the plan p for the service s_i in the BPEL process k.
$AINV_{kp}$	$n \times \max_i ADP_{kp}(i)$ matrix, where an element $AINV_{kp}(i,t)$ represents the number of invocations of the service s_i with respect to the new path t introduced by the plan p. It obviously holds: $AINV_{kp}(i,t) = 0\ \forall t > ADP_{kp}(i)$.

For each requirement r, only one plan must be chosen. The following equation represents this constraint:

$$\sum_{p=1}^{|AP_r|} y_{rp} = 1, \forall r = 1 \ldots m \tag{1}$$

The following variables allow to select an instance available for the i-th existing software service.

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

$Inst_i$ may include the element i itself. If there are no available instances, we assume that the existing service s_i itself belongs to $Inst_i$.

For each existing service i, if it is always in use after the adaptation phase (i.e., the chosen plans do not delete all the leafs representing the service i of the BPEL trees or, at least, an additional path is suggested for the service i after the adaptation phase) only one instance must be selected. The following equation represents this constraint:

$$\sum_{j=1}^{|Inst_i|} x_{ij} = 1 - \prod_{k=1}^{K} \prod_{t=1}^{P_{ki}+1} \sum_{r=1}^{m} \sum_{p=1}^{|AP_r|} y_{rp} D_{krp}(i,t), \forall i = 1 \ldots n \tag{2}$$

D_{krp} is a $n \times \max_i P_{ki}$ matrix, where an element $D_{krp}(i,t)$ is equal to 1 if the adaptation plan p suggests to delete the leaf, representing the service i of the path t in the BPEL k, and it is equal to 0 otherwise. In order to capture if the chosen plans introduce new paths for the service i, we assume that, if the plan p does not suggest to add at least a new path for the service i in the BPEL k, $D_{krp}(i, P_{ki}+1)$ is equal to 1, otherwise is equal to 0.

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 equation represents this constraint:

$$[v_{ij}]_p \cdot y_{rp} \leq x_{ij} \quad \forall r = 1 \ldots m, \forall p = 1 \ldots |AP_r| \quad \forall i = 1 \ldots n, \forall j = 1 \ldots |Inst_i| \tag{3}$$

The following variables allow to select the new software services to be introduced:

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

Similarly, if a plan is part of the model solution, then all the new services that it suggests to introduce have to belong to the solution. The following equation represents this constraint:

$$y_{rp} \leq z_h, \forall r = 1 \ldots m, \forall p = 1 \ldots |AP_r|, \forall h : news_h \in NS_p \tag{4}$$

In order to solve incompatibilities problems among services, for example, due to implementation technology (other than introducing integration/adaptation costs), additional constraints may be added as contingent decisions (Jung and Choi, 1999) (e.g., $x_{12} \leq x_{23}$, which means that if the second instance is selected for the first existing service, then for the second existing service the third instance must be selected).

2.6. Cost objective function (COF)

In this section, we define the objective function of our optimization model.

The objective function to be minimized, as the sum of the costs for all the instances selected for the existing services and the ones that introduce new elementary services, is given by:

$$COF = \sum_{i=1}^{n} \sum_{j=1}^{|Inst_i|} c_{ij} x_{ij} + \sum_{h=1}^{|NewS|} \overline{c_h} z_h \tag{5}$$

This cost model is a light-weighted one, as we work in favor of model solvability. However, it can be replaced by a well-assessed cost model from literature (e.g., COCOMO Boehm, 1981) to increase the result accuracy. This can be done without (essentially) changing the overall model structure, with the side effect of increase solution complexity.

2.7. Reliability (availability) constraint (REL/AV)

In this section, we present the system reliability (availability) constraint and how to compute it as function of the reliability (availability) of the k-th external service provided by the system.

We consider systems that may incur only in crash failures, which immediately and irreversibly compromise the behavior of the whole system.

A minimum threshold R (A) is given to the reliability (availability) on demand (Trivedi, 2001) of the whole system, which can be obtained as a function of the reliability (availability) q_{ij} of the existing services and the reliability (availability) $\overline{q}_h$ of new services. To this end, we have been inspired by the model in Cardellini et al. (2007), which we present and compare with our approach in Section 5.

After the application of a set of adaptation plans, the following values appear in the BPEL tree k: inv_{ki} as the average number of invocations of the existing service i, inv_{kh} as the average number of invocations of the new service h.

Reliability (Availability) of the k-th external service provided by the system.

Let Q_k, $Q \in \{R, A\}$, denote the reliability (availability) of the k-th external service.

Since the (logarithm of the) reliability/availability is additive (Cardoso et al., 2004) and we assume that for each external service provided by the system we allot a BPEL process, the logarithm of the reliability (availability) of the k-th external service can be expressed as follows:

$$\log Q_k = \sum_{i=1}^{n} (inv_{ki} \cdot \sum_{j=1}^{|Inst_i|} x_{ij} \cdot \log q_{ij}) + \sum_{h=1}^{|NewS|} inv_{kh} \cdot \log \overline{q_h} \cdot z_h \quad (6)$$

The parameter inv_{ki} can be expressed as follows:

$$inv_{ki} = \sum_{t=1}^{P_{ki}} \left(\left(\sum_{r=1}^{m} \sum_{p=1}^{|AP_r|} y_{rp} \cdot \Delta_{kp}(i,t) \cdot \Delta INV_{kp}(i,t) \right) + \left((1 - \sum_{r=1}^{m} \sum_{p=1}^{|AP_r|} y_{rp} \cdot \Delta_{kp}(i,t)) \cdot INV_{ki}(t) \right) \right) + \sum_{r=1}^{m} \sum_{p=1}^{|AP_r|} \sum_{\bar{t}=1}^{ADP_{kp}(i)} y_{rp} \cdot AINV_{kp}(i,\bar{t}), \quad 1 \le i \le n \quad (7)$$

inv_{ki} depends, in fact, on how the chosen adaptation plans suggest to modify the paths – having the service i as leaf node – of the BPEL tree k (i.e., on the number of invocations $\Delta INV_{kp}(i, t)$ of the service i along the path t after the application of the plan p [11]). Besides, it depends on the paths $ADP_{kp}(i)$, which the chosen adaptation plans suggest to introduce in the k-th BPEL tree from the root to the service i. Thus, it is also a function of the number of invocations $AINV_{kp}(i, \bar{t})$ of the service i along the $\bar{t}$-th new introduced path.

The parameter inv_{kh} can be expressed as follows:

$$inv_{kh} = \sum_{r=1}^{m} \left(\sum_{p=1}^{|AP_r|} y_{rp} \sum_{t=1}^{NP_p(h,k)} NINV_{kp}(h,t) \right), \quad 1 \le h \le |NewS| \quad (8)$$

inv_{kh} depends, in fact, on the paths $NP_p(h, k)$, which the chosen adaptation plans suggest to introduce in the k-th BPEL tree from the root to the service $news_h$. Thus, it is a function of the number of invocations $NINV_{kp}(h, t)$ of the service $news_h$ along the t-th new introduced path.

In a worst-case scenario, all external services should satisfy the reliability (availability) constraint. Therefore, the latter can be formulated as follows:

$$\min_{k=1\ldots K} (Q_k) \ge Q \quad (9)$$

where $Q \in \{R, A\}$.

In an average-case scenario, the reliability (availability) constraint can be formulated as follows:

$$\sum_{k=1}^{K} \Lambda_k \cdot Q_k \ge Q \quad (10)$$

A finer-grained constraint can be introduced to guarantee a certain level of reliability (availability) $\overline{Q}_k$ to the k-th service provided by the system, i.e., $\Lambda_k Q_k \ge \overline{Q}_k$.

2.8. Performance constraint (RT)

In this section, we present the system performance constraint and how to compute it as function of the average response time Rt_k for a call to the k-th external service provided by the system.

A maximum threshold *Res* is given to the average response time of the software architecture, which can be obtained as a function of the response time rt_{ij} of the existing services and the response time $\overline{rt}_h$ of new services. As shown below, we carry out the computation of Rt_k by recursively computing the average response time of the structured activities of the BPEL tree k. To this end, we have been inspired by the model in Cardellini et al. (2007), which we present and compare with our approach in Section 5.

Response time for a call to the k-th external service provided by the system.

Differently from reliability (availability), the response time metric is additive as long as the composite service does not include *flow* structured activities. In such cases, the response time Rt_k of the BPEL process k can be expressed as follows:

$$Rt_k = \sum_{i=1}^{n} \left(inv_{ki} \cdot \sum_{j=1}^{|Inst_i|} x_{ij} \cdot rt_{ij} \right) + \sum_{h=1}^{|NewS|} inv_{kh} \cdot \overline{rt}_h \cdot z_h \quad (11)$$

In the general case, instead, we need to consider the fact that the response time of a *flow* activity is given by the largest response time among its component activities. Therefore, the response time is not additive and (11) does not hold.

The overall expected response time Rt_k of the k-th BPEL process is given by the following expressions (easily derived by visiting the BPEL tree in postorder and properly aggregating the response time of the child nodes to derive the response time of the parent node):

$$Rt_k = \begin{cases} \max\limits_{l' \in d(w)} Rt_{l'k} & w \in \overline{F}_k \\ \sum\limits_{i=1, i \prec_{dd} w}^{n} inv'_{ki} \cdot \sum\limits_{j=1}^{|Inst_i|} x_{ij} rt_{ij} + \\ + \sum\limits_{h=1, h \prec_{dd} w}^{|NewS|} inv'_{kh} z_h \overline{rt}_h + \\ + \sum\limits_{f \in \overline{F}_k, f \prec_{dd} w} \overline{inv}_{kf} Rt_{fk} & w \notin \overline{F}_k \end{cases} \quad (12)$$

where: (i) $d(w)$ is the set of child nodes of the root w; (ii) $Rt_{l'k}$ is the average response time of the l'-th child of the root w; (iii) $\overline{F}_k$ is the set of nodes of type *flow*. $\overline{F}_k$ includes the existing nodes of type *flow* of the BPEL k, which have not been removed after the adaptation phase, and the additional nodes of type *flow* that the chosen plans suggest to introduce.

The second of the expressions for Rt_k comprises three terms:

- The first term is the sum of the response times of the existing services, which are *direct descendant* of the root w (i.e., they do not appear within a *flow* structured activity).
  inv'_{ki} is the average number of invocations of the service i along the paths (introduced by the chosen plans in the k-th BPEL tree), where the service i is a *direct descendant* of the root. It can be determined from an expression similar to the one used for the parameter inv_{ki} (see Eq. (7)). For the sake of readability, we provide its formulation in Appendix.
- The second term is the sum of the response times of the new services, which are *direct descendant* of the root w.
  inv'_{kh} is the average number of invocations of the new service h along the paths (introduced by the chosen plans in the k-th BPEL tree), where the service h is a *direct descendant* of the root. It can be determined from an expression similar to the one used for

[11] Notice that, since we assume that the set of adaptation actions, able to deal with each requirements, are independent from each other (see Section 2.2), the path t may be modified by only one of the chosen plans.

the parameter inv_{kh} (see Eq. (8)). For the sake of readability, we provide its formulation in Appendix.

- The third term is the sum of the response times of the *flow* activities, which are *direct descendant* of the root w (i.e., flow activities which are not nested within other flow activities).

$\overline{inv}_{kf}$ is the expected number of times node f is visited, whereas Rt_{fk} is the average response time of the node f. For the sake of readability, we provide the estimation of $\overline{inv}_{kf}$ and Rt_{fk} in Appendix.

In a worst-case scenario, all external services provided by the system should satisfy the performance constraint. Therefore, this latter can be formulated as follows:

$$\max_{k=1\ldots K}(Rt_k) \leq Res \tag{13}$$

In an average-case scenario, the response time constraint can be expressed as follows:

$$\sum_{k=1}^{K} \Lambda_k Rt_k \leq Res \tag{14}$$

Again, a finer-grained constraint can be introduced to guarantee that a call to the k-th external service, provided by the system, is satisfied within a certain time Res_k, i.e., $\Lambda_k Rt_k \leq Res_k$.

2.9. Other constraints

For the sake of model formulation, we model queues by introducing constraints on the capacity of elementary services. The following constraint expresses a limit on the capacity of the i-th existing service:

$$x_{ij} \cdot inv_{ki} \cdot \lambda_k \leq 0.9 \times L_{ij} \tag{15}$$

where L_{ij} is the average load of the j-instance available for the i-th existing service.

The total load caused by the arrival rate of the call λ_k, multiplied by the number of invocations of the service i, cannot exceed the processing capacity of the service. Notice that the constraint limits the capacity at $0.9 \times L_{ij}$. The reason is that, as the service load nears its capacity, queuing starts to occur and sharply increases the response time of the service. Moreover, it often increases the processing delay to unacceptable levels. Therefore, the service load is limited to avoid queuing. The exact limit to the load (i.e., $0.9 \times L_{ij}$ in this case) can be adjusted to accommodate the needs of each design project separately.

Similarly, the following constraint expresses a limit on the processing capacity of h-th new service.

$$z_h \cdot inv_{kh} \cdot \lambda_k \leq 0.9 \times \bar{L}_h \tag{16}$$

where $\bar{L}_h$ is the average load of the new service h.

2.10. Model summary and limitations

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

$$min \quad \sum_{i=1}^{n}\sum_{j=1}^{|Inst_i|} c_{ij}x_{ij} + \sum_{h=1}^{|NewS|} \overline{c_h} z_h$$

$$\log Q_k = \sum_{i=1}^{n}\left(inv_{ki} \cdot \sum_{j=1}^{|Inst_i|} x_{ij} \cdot \log q_{ij}\right) + \sum_{h=1}^{|NewS|} inv_{kh} \cdot \log\overline{q_h} \cdot z_h, \forall k = 1\ldots K$$

$$\sum_{k=1}^{K} \Lambda_k \cdot Q_k \geq Q$$

$$Rt_k = \begin{cases} \max\limits_{l' \in d(w)} Rt_{l'k} & w \in \bar{F}_k \\ \sum\limits_{i=1, i \prec_{dd} w}^{n} inv'_{ki} \cdot \sum\limits_{j=1}^{|Inst_i|} x_{ij} rt_{ij} + \\ + \sum\limits_{h=1, h \prec_{dd} w}^{|NewS|} inv'_{kh} z_h \overline{rt}_h + \\ + \sum\limits_{f \in \bar{F}_k, f \prec_{dd} w} \overline{inv}_{kf} Rt_{fk} & w \notin \bar{F}_k \end{cases}$$

$$\sum_{k=1}^{K} \Lambda_k Rt_k \leq Res$$

$$x_{ij} \in \{0,1\}, \forall i = 1\ldots n, \forall j = 1\ldots |Inst_i|$$

$$z_h \in \{0,1\}, \forall h = 1\ldots |NewS|$$

$$y_{rp} \in \{0,1\}, \forall r = 1\ldots m, \forall p = 1\ldots |AP_r|$$

Other Constraints (e.g., (3) and (4))

The objective function (5) under the main reliability, availability (10) and performance (14) constraint, plus the constraints on the model variables, represent our optimization model. The model solution determines the adaptation plan to choose for each change requirement of a change scenario, in order to minimize software adaptation costs under the reliability, availability and performance constraint.

From a practical point of view, different optimization techniques (e.g., branch and bound) can be used to solve the model, depending on several factors, due mainly to its use for evolution (at re-design time) or self-adaptation (at run-time). For example, if permanent modifications are requested or a safe-critical service has to be adapted, precise analysis (resulting often more expensive) must be performed. On the contrary, if run-time modifications are claimed, requiring, for example, only the service selection without using more sophisticated actions, faster approaches must be adopted allowing the run-time adaptation (such as metaheuristic techniques, whose effectiveness and efficiency have been already demonstrated to support the service selection activity at run-time Rosenberg et al., 2010).

For the experimentation we have used the LINGO tool (Online, 2012), which is a non-linear model solver, to produce the results. In cases where the computation time becomes too big, search based techniques, such as metaheuristic techniques (e.g., the tabu-search algorithm) can be used to solve the model.

Summarizing, our model undergoes the following main assumptions:

- We assume that the services communicate by exchanging synchronous messages.

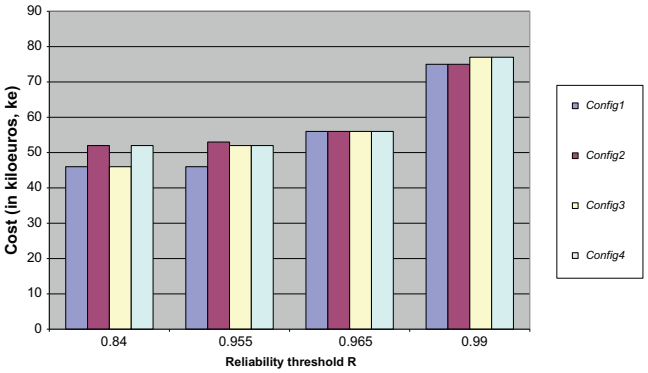

Fig. 10. Objective function values for different cases.

- From a reliability/availability point of view, we assume that the services are independent. Namely, the failure of a service provokes the failure of the whole system. Such assumption is common to many reliability/availability approaches (see, for example, the survey Immonen and Niemelä, 2008). What is basically neglected under this assumption is the error propagation probability, which in several real domains (such as control systems) is not an issue, because component errors are straightforwardly exposed as system failures. In order to relax such an assumption, an error propagation model must be introduced (e.g., the reliability model for a service-based system, which we have introduced in Cortellessa and Potena, 2007, embedding the error propagation property). Besides, we suppose that a service shows the same reliability (availability and response time) across different invocations, and that the patterns of interactions within each scenario does not change by changing the service instance.
- We assume that the operational profile of the system at run-time is the same used to certify the services. This is a typical assumption of the quality models. However, it can be removed if an estimate of the run-time operational profile is available.
- We do not take into account the fact that sometimes the information provided by the service providers (e.g., web-service provider) is not enough to estimate the model parameters (i.e., cost and reliability). In fact, they could be characterized by a not negligible uncertainty. The propagation of this uncertainty on the objective function and the constraints should be analyzed. Several methods have been introduced in order to estimate the propagation of the uncertainty on the quality system (see, for example, Wattanapongskorna and Coit, 2007).
- We assume that the plans for different change requirements are independent from each other (as explained in Section 2.2). To relax such assumption, additional constraints may be added as contingent decisions (Jung and Choi, 1999), for example, to solve incompatibilities problems among plans due to implementation technology (e.g., $y_{12} \leq y_{23}$, which means that if the second plan is selected for the first requirement, then for the second requirement the third plan must be selected). Moreover, we are working toward supporting the generation of the plans. To this extent, we are investigating how to use the guidelines of existing tools (like CoDesign tool Bang et al., 2010) allowing geographically distributed architects to cooperate in order to design a system.
- We assume that the quality attributes (i.e., reliability, availability and performance) are independent from each other, and for each of them a self-contained analytical expression can be formulated. We express dependencies by only using the tradeoff analysis. This type of approach is more often applied in the state-of-the-art literature (see, for example, Yang et al., 2009). Obviously, this assumption does not always hold, as it depends on the considered quality attributes and the model complexity. If quality attributes have to be correlated (Bass et al., 2001), then additional constraints may be needed, which can be expressed as contingent decisions.
- We assume that for the external services we allot BPEL trees. Even though in a large-scale system the designers may not document all the system functionalities in terms of BPEL processes, this is a typical assumption of the quality models.

3. A running example: smartphone mobile application

In order to show the practical usage of our optimization model, we have applied the optimization model to the example considered in Section 2.3.

3.1. Model parameters

We have considered the change scenario and the adaptation plans illustrated in Tables 1 and 2, respectively, and described in Section 2.3.

We have enhanced the example used in Cortellessa et al. (2010) by taking into account new features of the model, such as (other than the sequential execution) other service composition operators, and we have chosen a slightly different change scenario. Besides, with respect to the experimentation presented in Cortellessa et al. (2010) here we consider the new features of the model. We conduct the experiments to highlight the novelties and potentialities of this paper approach. This is the reason why model parameters have been set (appropriately varied) rather than extensively providing numerical results for the problem scenario.

For the sake of readability, details on parameters can be found in Appendix. The estimation of the parameters entering our model has been partly based on the data test set used in Cortellessa et al. (2010). However, incomplete documentation (due to the enhancement of the model proposed in Cortellessa et al. (2010)) forced to adopt extrapolation techniques to provide additional values. For example, for performance parameters, the confidence intervals – appropriate for the considered scenario – have been obtained by analyzing the scenario descriptions.

3.2. Experimenting the model on the example

In Fig. 10 we report the results obtained from solving the optimization model with two different values of the response time $\overline{rt_1}$ (s) of the new service $news_1$, while varying the reliability $\overline{r_h}$ of the new services $news_1$ and $news_4$. We have used the optimization model under different reliability constraints, as represented in Fig. 10 by each group of four bars that refer to different values of R. Thus, we have varied the reliability bound R in four steps. Table 4 summarizes the values used in the different configurations. In all the configurations, we have fixed Res to 12 s, and the probabilities that the first, second and third external service will be invoked to 0.3, 0.3, 0.4, respectively.

The length of each bar, along the y-axis, represents the optimal value of the objective function of our model, which is the minimum cost of adaptation for the considered change scenario. Hence, each bar corresponds to one solution of the optimization model. This latter for each experiment was solved using the model solver LINGO (as remarked in Section 2.10). The output of a LINGO run, for an alternative model – generated by a configuration of parameters

Table 4
Configurations of the example application.

	$\overline{r_1} = 0.99999, \overline{r_4} = 0.91$	$\overline{r_1} = 0.993, \overline{r_4} = 0.995$
$\overline{rt_1} = 1$	*Config1*	*Config3*
$\overline{rt_1} = 2$	*Config2*	*Config4*

–, determines the adaptation plan to choose for each change requirement.

As an example, the minimum cost to adapt the system in *Config1* is 46 kE, if we intend to guarantee a minimum system reliability equal to 0.84 and that the response time for a call to the system is lesser than (or equal to) 12 s. The optimal solution cost obviously increases in *Config1* (up to 75 kE), while raising the reliability threshold (up to 0.99).

In Table 5 we report the detailed results. Each cell reports the resulting system reliability, the adaptation cost (kE) and the system response time (s) obtained for a certain configuration (row) and a certain reliability threshold (column).

Table 6 summarizes the solution vectors that correspond to the values reported in Table 5. Each cell contains heterogeneous information, sequenced as follows: the choice of available service instances, the possible new services introduced, the adaptation plans suggested for the change requirements.

For example, the entry (*Config1*, $R = 0.84$) in Table 6 means that, in order to achieve the optimal cost of adaptation the following plans have to be adopted: ap_{11} for the first requirement, ap_{21} for the second requirement, and ap_{32} for the third 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.

Let us consider *groups of bars pairwise with the same reliability for the services* $news_1$ and $news_4$ in Fig. 10, or better *Config1* and *Config2*, *Config3* and *Config4*. As expected, while increasing the response time of the new service $news_1$, once fixed the reliability bound R, the adaptation cost almost always increases.

On the other hand, once fixed the response time of the service $news_1$, sometimes it is necessary to take different adaptation actions (even if the adaptation cost does not increase), while decreasing the reliability of the service $news_1$ from 0.99999 to 0.993 and increasing the one of $news_4$ from 0.91 to 0.995.

For example, the model provides two different solutions for the following entries in Table 6: (*Config1*, $R = 0.99$) and (*Config3*, $R = 0.99$). Such solutions differ for the adaptation plan selected for the first change requirement, for the instance selected for the first existing service (i.e., s_1), and the new service to be added. The solution given for *Config1* – cheaper than the other one – would not be a feasible solution for *Config3*, because the system reliability would be equal to 0.989145 and it does not satisfy the reliability constraint.

As another example, the model provides two different solutions for the entries (*Config2*, $R = 0.955$) and (*Config4*, $R = 0.955$) in Table 6. In this case, the adaptation strategies of *Config4* cannot be used for *Config2* because the reliability constraint would not be satisfied. Such solutions differ for the adaptation plans selected for the first and second change requirement, for the instance selected for the existing services s_1 and s_4, and for the new service to be added. The solution in *Config4* suggests to introduce the new service $news_4$. This latter has the same cost of the service $news_1$, which is suggested by the solution in *Config2*. The solution given for *Config4* – cheaper than the other one – would not be a feasible solution for *Config2*, because the system reliability would be equal to 0.854323 and it does not satisfy the reliability constraint.

Let us focus on the case of *varying the response time* of a new service in Fig. 10. Once fixed a reliability threshold, for the same value of the reliability of the new services, the adaptation cost could increase while increasing the response time of a new service. Therefore, different actions could be suggested, for example: once fixed $R = 0.84$, the model provides two different solutions for the entries (*Config 1*, $R = 0.84$) and (*Config 2*, $R = 0.84$) in Table 6.

For the first entry, the model suggests to include the service $news_1$, whereas for the second entry it suggests to add, other than $news_1$, the service $news_4$. The introduction of either $news_1$ or $news_1$ and $news_4$, following the plans suggested by the solutions, would involve different modifications to the structure and behavior of the system. The two solutions differ also for the instances selected for the existing services. For example, the model for the first entry selects s_{11}, whereas it suggests the service s_{12} for the second entry. If s_{11} would represent the *Client* service itself (i.e., if the set $Inst_1$ would also include the *Client* service itself and its cost would represent, for example, the cost that the service would need for some upgrading to be necessarily performed for keeping it), for the first entry the model would suggest to keep the *Client* service, whereas in the second entry the model would suggest to replace it.

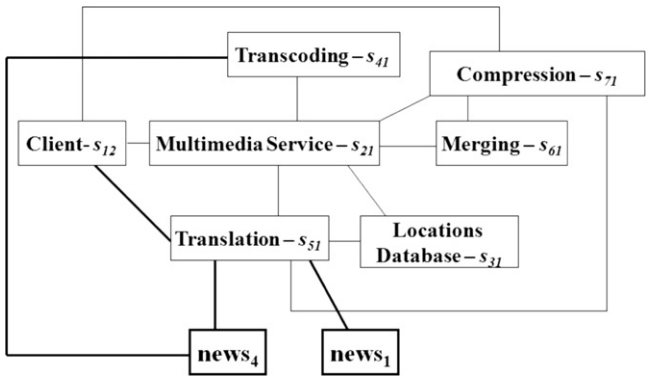

Fig. 11. Static aspect of the system after the application of adaptation actions for (*Config 2*, $R = 0.84$).

This highlights how our model can support a maintainer to choose to either keep or replace an existing service. The solution given for the first entry, which is also cheaper than the other one, is not a good solution for the second entry because the performance constraint would be not satisfied. In fact, the response time of a call to the system would be equal to 12.436 s.

Figs. 6–9 show how the static and dynamic structure of the system change after the application of the adaptation actions suggested by the solution in the entry (*Config 1*, $R = 0.84$) in Table 6.

Figs. 11–14 show similar changes after the application of the adaptation actions suggested by the solution in the entry (*Config 2*, $R = 0.84$) in Table 6.

In the figures, we have marked in bold the modifications brought in after the application of the plans and we have put a cross on the interactions that have been removed.

Finally, once fixed a reliability threshold, the adaptation actions could be not different while varying the response time of the new services. For example, while varying the response time of $news_1$, considering the same pair of reliability values of the new services $news_1$ and $news_4$ (i.e., *Config 1* and *Config 2*, *Config 3* and *Config 4*) and $R = 0.99$, the model suggests exactly the same solution.

4. Model sensitivity analysis

In this section we analyze the model sensitivity to changes in its parameters. For the sake of continuity, we work on the example considered in the previous section.

Our experimentation is meant to give evidence of the model sensitivity to changes in the problem scenario, rather than extensively providing numerical results for a specific scenario. This is the reason why the scenarios we consider show specific values of model parameters. We show through the sensitivity analysis the potential of the optimization model to drive architectural decisions. To this extent, we report significant examples of its usage. Thus, each example is characterized by particular model parameters able to show some peculiar features of the model behavior.

Similarly to Table 6, in Table 7 we report the detailed analysis results that correspond to the values reported in Table 8.

Table 5
System indices for different configurations.

	$R = 0.84$	$R = 0.955$	$R = 0.965$	$R = 0.99$
Config1	*SysRel* = 0.957155 *Cost* = 46 kE *SysRt* = 11.036 s	*SysRel* = 0.957155 *Cost* = 46 kE *SysRt* = 11.036 s	*SysRel* = 0.976514 *Cost* = 56 kE *SysRt* = 10.06 s	*SysRel* = 0.996108 *Cost* = 75 kE *SysRt* = 9.660 s
Config2	*SysRel* = 0.846192 *Cost* = 52 kE *SysRt* = 11.436 s	*SysRel* = 0.966352 *Cost* = 53 kE *SysRt* = 11.636 s	*SysRel* = 0.976514 *Cost* = 56 kE *SysRt* = 11.460 s	*SysRel* = 0.996108 *Cost* = 75 kE *SysRt* = 10.660 s
Config3	*SysRel* = 0.947801 *Cost* = 46 kE *SysRt* = 11.036 s	*SysRel* = 0.959485 *Cost* = 52 kE *SysRt* = 10.636 s	*SysRel* = 0.966971 *Cost* = 56 kE *SysRt* = 10.06 s	*SysRel* = 0.990770 *Cost* = 77 kE *SysRt* = 9.660 s
Config4	*SysRel* = 0.947690 *Cost* = 52 kE *SysRt* = 11.436 s	*SysRel* = 0.959485 *Cost* = 52 kE *SysRt* = 10.636 s	*SysRel* = 0.966971 *Cost* = 56 kE *SysRt* = 11.460 s	*SysRel* = 0.990770 *Cost* = 77 kE *SysRt* = 9.660 s

Table 6
Solution vectors for different configurations.

	$R = 0.84$	$R = 0.955$	$R = 0.965$	$R = 0.99$
Config1	$[s_{11}, s_{21}, s_{31}, s_{42}, s_{51}, s_{61}, s_{71}]$ $[news_1], [ap_{11}, ap_{21}, ap_{32}]$	$[s_{11}, s_{21}, s_{31}, s_{42}, s_{51}, s_{61}, s_{71}]$ $[news_1], [ap_{11}, ap_{21}, ap_{32}]$	$[s_{11}, s_{21}, s_{31}, s_{42}, s_{51}, s_{62}, s_{71}]$ $[news_1], [ap_{11}, ap_{21}, ap_{32}]$	$[s_{11}, s_{22}, s_{31}, s_{42}, s_{51}, s_{62}, s_{73}]$ $[news_1], [ap_{11}, ap_{21}, ap_{33}]$
Config2	$[s_{12}, s_{21}, s_{31}, s_{41}, s_{51}, s_{61}, s_{71}]$ $[news_1, news_4], [ap_{12}, ap_{23}, ap_{32}]$	$[s_{11}, s_{22}, s_{31}, s_{42}, s_{51}, s_{61}, s_{71}]$ $[news_1], [ap_{11}, ap_{21}, ap_{33}]$	$[s_{11}, s_{21}, s_{31}, s_{42}, s_{51}, s_{62}, s_{71}]$ $[news_1], [ap_{11}, ap_{21}, ap_{32}]$	$[s_{11}, s_{22}, s_{31}, s_{42}, s_{51}, s_{62}, s_{73}]$ $[news_1], [ap_{11}, ap_{21}, ap_{33}]$
Config3	$[s_{11}, s_{21}, s_{31}, s_{42}, s_{51}, s_{61}, s_{71}]$ $[news_1], [ap_{11}, ap_{21}, ap_{32}]$	$[s_{12}, s_{22}, s_{31}, s_{41}, s_{51}, s_{61}, s_{71}]$ $[news_4], [ap_{12}, ap_{23}, ap_{33}]$	$[s_{11}, s_{21}, s_{31}, s_{42}, s_{51}, s_{62}, s_{71}]$ $[news_1], [ap_{11}, ap_{21}, ap_{32}]$	$[s_{12}, s_{22}, s_{31}, s_{42}, s_{51}, s_{62}, s_{73}]$ $[news_4], [ap_{12}, ap_{21}, ap_{33}]$
Config4	$[s_{12}, s_{21}, s_{31}, s_{41}, s_{51}, s_{61}, s_{71}]$ $[news_1, news_4], [ap_{12}, ap_{23}, ap_{32}]$	$[s_{12}, s_{22}, s_{31}, s_{41}, s_{51}, s_{61}, s_{71}]$ $[news_4], [ap_{12}, ap_{23}, ap_{33}]$	$[s_{11}, s_{21}, s_{31}, s_{42}, s_{51}, s_{62}, s_{71}]$ $[news_1], [ap_{11}, ap_{21}, ap_{32}]$	$[s_{12}, s_{22}, s_{31}, s_{42}, s_{51}, s_{62}, s_{73}]$ $[news_4], [ap_{12}, ap_{21}, ap_{33}]$

Table 7
Solution vectors of the model sensitivity analysis.

Case 1	Case 2	Case 3
Sol 1: $[s_{12}, s_{22}, s_{31}, s_{42}, s_{51}, s_{61}, s_{71}]$ $[ap_{12}, ap_{21}, ap_{33}]$	***Sol 2:*** $[s_{11}, s_{22}, s_{31}, s_{42}, s_{51}, s_{61}, s_{71}]$ $[news_1]$ $[ap_{11}, ap_{21}, ap_{33}]$ ***Sol 3:*** $[s_{11}, s_{21}, s_{31}, s_{42}, s_{51}, s_{61}, s_{71}]$ $[news_1, news_2]$ $[ap_{11}, ap_{21}ap_{31}]$	***Sol 4:*** $[s_{11}, s_{21}, s_{31}, s_{42}, s_{51}, s_{61}, s_{71}]$ $[news_2]$ $[news_2], [ap_{21}, ap_{31}]$ ***Sol 5:*** $[s_{11}, s_{22}, s_{31}, s_{42}, s_{51}, s_{62}, s_{71}]$ $[ap_{21}, ap_{33}]$ ***Sol 6:*** $[s_{11}, s_{22}, s_{31}, s_{42}, s_{51}, s_{62}, s_{73}]$ $[ap_{21}, ap_{33}]$

Each cell in Table 7 contains heterogeneous information, sequenced as follows: the choice of available service instances, the possible new services introduced, the adaptation plans suggested for the change requirements. Each cell in Table 8 reports the resulting system reliability, the adaptation cost (kE) and the system response time (in s) obtained for a certain solution returned during the analysis.

Case 1 – Changing the adaptation plans for a requirement

Let us assume that the second adaptation plan available for the first change requirement (i.e., ap_{12}) is *Replacing Client (i.e., s_1) with its second instance available* instead of *Adding a new service $news_4$ that interacts both with Translation and Client AND Replacing Client (i.e., s_1) with its second instance available*.

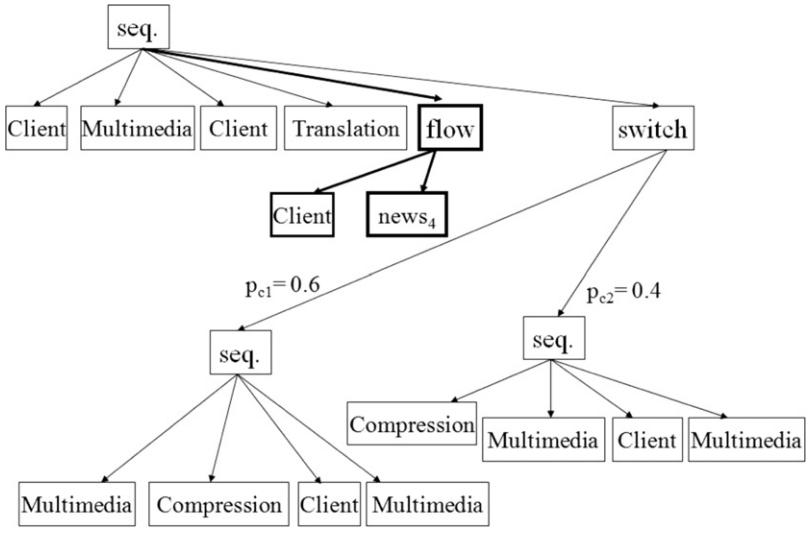

Fig. 12. BPEL tree of Fig. 3 after the application of ap_{12}, ap_{23} and ap_{32}.

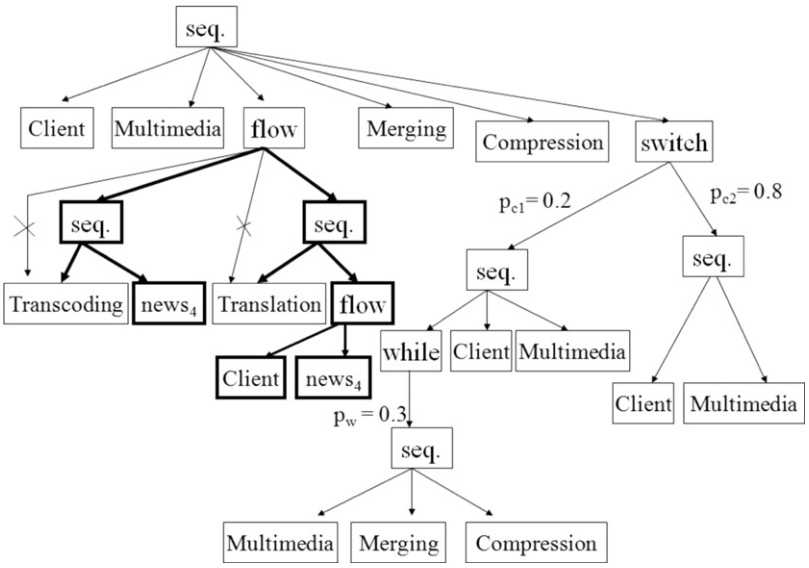

Fig. 13. BPEL tree of Fig. 4 after the application of ap_{12}, ap_{23} and ap_{32}.

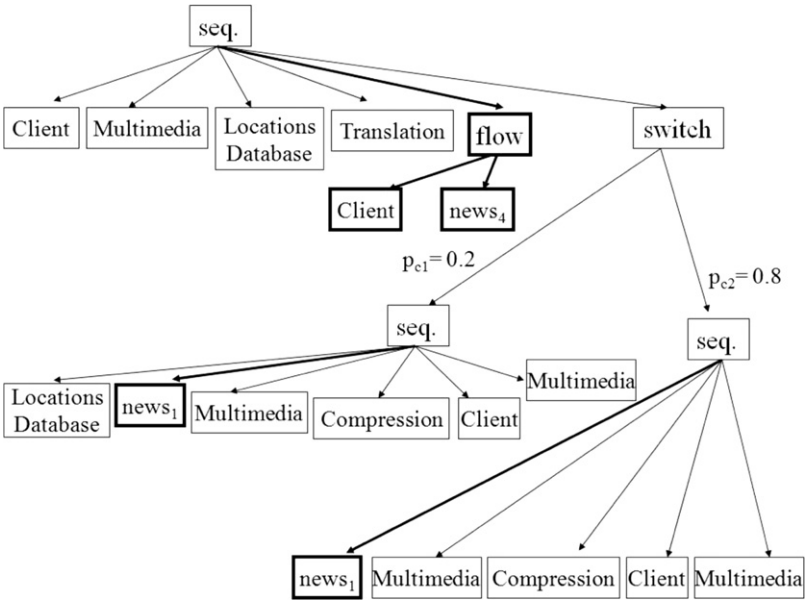

Fig. 14. BPEL tree of Fig. 5 after the application of ap_{12}, ap_{23} and ap_{32}.

For example, starting from the initial values of the parameters, for the entry (*Config1*, $R = 0.965$) we get the solution *Sol 1* in Table 7. The adaptation cost is equal to 48 kE, which is cheaper than the one of the solution shown in Table 6.

In this case, the model does not suggest to add new services; instead, it suggests to maintain the system by only replacing the initial elementary services, in contrast with the solution given in Table 6.

This highlights that in order to maintain a system, sometimes it is sufficient to replace the existing services with available instances, rather than embedding new services. This highlights how our model helps to combine (and, in general, to reason about) the decisions to be taken for each change requirement. As we have shown in this example, the adaptation actions might have to be modified while changing the software features. Therefore, our approach facilitates the work of a system maintainer, who does not have to insert adaptation actions for implementing new requirements, but only possible actions for each change, which are combined together by our model.

Case 2 – Varying reliability and response time of elementary services

Starting from the initial values of the model parameters, let us decrease the reliability of the second instance available for the first existing service (i.e., s_{12}) from 0.9994 to 0.94.

Table 8
System indices of the model sensitivity analysis.

Case 1	Case 2	Case 3
Sol 1: *SysRel* = 0.966584 *Cost* = 48 kE *SysRespTime* = 9.636 s	***Sol 2:*** *SysRel* = 0.937371 *Cost* = 53 kE *SysRespTime* = 10.636 s ***Sol 3:*** *SysRel* = 0.922987 *Cost* = 52 kE *SysRespTime* = 10.935 s	***Sol 4:*** *SysRel* = 0.951229 *Cost* = 45 kE *SysRespTime* = 11.236 s ***Sol 5:*** *SysRel* = 0.985908 *Cost* = 56 kE *SysRespTime* = 8.660 s ***Sol 6:*** *SysRel* = 0.996118 *Cost* = 68 kE *SysRespTime* = 8.660 s

Case 2.1

If we decrease the reliability of the new service $news_1$ from 0.99999 to 0.97 and increase the reliability of $news_2$ from 0.9992 to 0.999998, for the pair ($R = 0.92$, $\overline{rt_1} = 1$ s) we get the solution *Sol 2* in Table 7.

Case 2.2

Starting from *Case 2.1*, if we decrease the response time of the first and second versions available for the sixth service to 1 s (i.e., rt_{61} from 5 s to 1 s and rt_{62} from 2 s to 1 s), for the pair ($R = 0.92$, $\overline{rt_1} = 1$ s) we get the solution *Sol 3* in Table 7.

Figs. 15–18 show how the static and dynamic structure of the system change after the application of the adaptation actions suggested by this solution. In the figures, we have marked in bold the modifications brought in after the application of the plans and we have put a cross on the interactions that have been removed.

Notice that, if the adaptation plan ap_{31} would assume an interaction between the service $news_2$ and *Compression* instead of the one between *Translation* and *Compression*, a dependency between the new services $news_1$ and $news_2$ and between $news_2$ and *Compression* would also be introduced. In Fig. 19, we show how the static structure of the system would change after the application of the adaptation actions. In the figure, we have dotted the additional dependencies brought in after the application of the plans.

The solution given for this example, which is also less expensive than the one (given) for *Case 2.1*, is not a good solution for *Case 2.1*, because the performance constraint would not be satisfied. In fact, the response time of the system would be equal to 12.236 s.

Case 3 – Removing a requirement from a change scenario

Starting from the initial values of the model parameters, let us assume that the change scenario in Table 1 does not include the first requirement req_1.

Case 3.1

For the entry (*Config 1*, $R = 0.84$) we get the solution *Sol 4* in Table 7.

Moving from such a solution, if we apply the first plan ap_{11} for the first requirement, the system reliability decreases from 0.951229 to 0.951220, and the response time increases from 11.236 s to 12.236 s. Therefore, the performance constraint is not satisfied. Besides, the cost increases from 45 kE to 52 kE. Instead, if we apply the second plan ap_{12} for first requirement then the system reliability decreases from 0.951229 to 0.865333 (which is still larger than 0.84) and the system response time increases from 11.236 s to 12.236 s. The cost increases from 45 kE to 54 kE.

Case 3.2

For the pair ($R = 0.97$, $\overline{rt_1} = 1$ s) we get the solution *Sol 5* in Table 7.

Moving from such a solution, if we apply the first plan ap_{11} for the first requirement, we get the following solution: $[s_{11}, s_{21}, s_{31}, s_{42}, s_{51}, s_{62}, s_{71}]$, $[news_1]$, $[ap_{11}, ap_{21}, ap_{32}]$. The cost does not change. The reliability and the performance constraints are satisfied. In fact, the system reliability decreases from 0.985908 to 0.976514, whereas the response time increases from 8.660 s to 10.06 s.

Case 3.3

For the pair ($\overline{rt_1} = 1$ s, $R = 0.99$) we get the solution *Sol 6* in Table 7.

Moving from such a solution, if we apply the first plan ap_{11} for the first requirement, the system reliability decreases from 0.996118 to 0.996108. Thus, the reliability constraint is satisfied. The performance constraint is also satisfied, because the system response time is equal to 9.660 s. The cost is equal to 75 kE. Notice that we get the same solution given in Table 6.

Instead, if we apply the second plan ap_{12} for the first requirement, the system reliability is equal to 0.906132. The reliability constraint is satisfied. The response time is equal to 9.660 s. The cost is equal to 77 kE.

This highlights that it may not be necessary to fully apply a change scenario in order to keep the cost under a certain threshold. On one hand, after introducing a new requirement, the cost of the system may increase (e.g., in *Case 3.3* the cost increases from 68 kE to either 75 kE or 77 kE, whereas in *Case 3.2* the cost does not change). On the other hand, in order to implement all the required changes with a certain budget, it may be necessary to solve a trade-off between the system quality and the cost. For example, in *Case 3.1*, if the cost increases to 52 kE, the reliability (the response time) is lower (larger) than the one achieved without introducing the first requirement.

5. Related work

Several research efforts have been devoted in the last years to the definition of approaches and frameworks for adaptation (see surveys Fox and Clarke, 2009; Ibrahim and Mouël, 2012; André et al., 2010). Most of them typically adapt a system by (i) service selection (see, for example, Canfora et al., 2008; Oh et al., 2008; Cardellini et al., 2007 detailed below), (ii) parameterization (see, e.g., the framework in Calinescu and Kwiatkowska, 2009, based on Markov-chain quantitative analysis, that dynamically adjusts the parameters of an IT System) or (iii) exploiting the inherent redundancy of the SOA environment (e.g., in Guo et al., 2007 it is presented a methodology to select different redundancy mechanisms to improve the reliability experienced by a single request addressed to a composite service).

The work in Canfora et al. (2008) presents a run-time 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 Oh et al. (2008) 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 information about services usage and service trees, as well as the information of the services (e.g., QoS data and interfaces). 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, for example, could be updated 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 also presented. In fact, for example, a service could be re-selected if either the currently selected service is not available or the QoS of the service changes.

The work in Cardellini et al. (2007) presents an optimization model, which supports the service selection for flows of requests, where a flow is generated by multiple requesters all requiring the same QoS level. The objective function is defined as weighed sum of the response time, cost and availability, whereas constraints on the response time and availability are embedded into the model. Based on the QoS composition rules defined in Cardoso et al. (2004), where the same approach is applied to workflows, the computation of the QoS attributes is carried out by recursively computing the QoS of the structured BPEL activities from the QoS of the component activities. Our approach is inspired by this work for the definition of the performance constraint and availability (reliability) constraint. In this paper, we support, other than the service selection, the adoption of new services, and the changing of the system behavior by suggesting how to remove or introduce interaction(s) between existing services and/or new services.

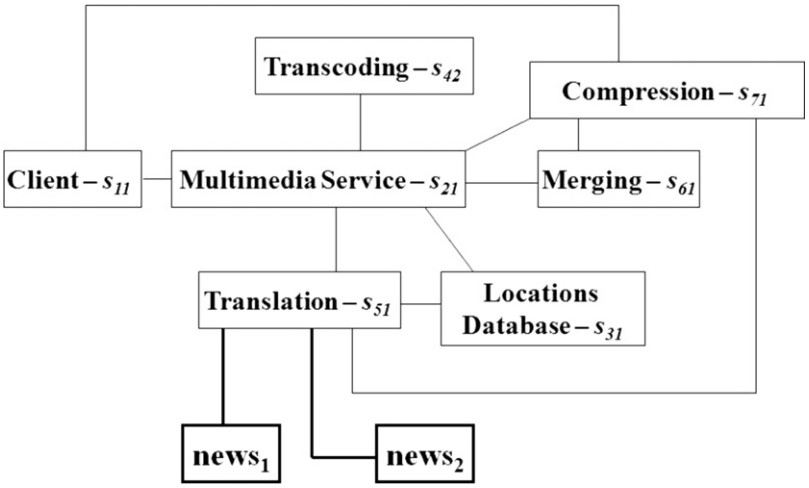

Fig. 15. Static aspect of the system after the application of ap_{11}, ap_{21}, ap_{31}.

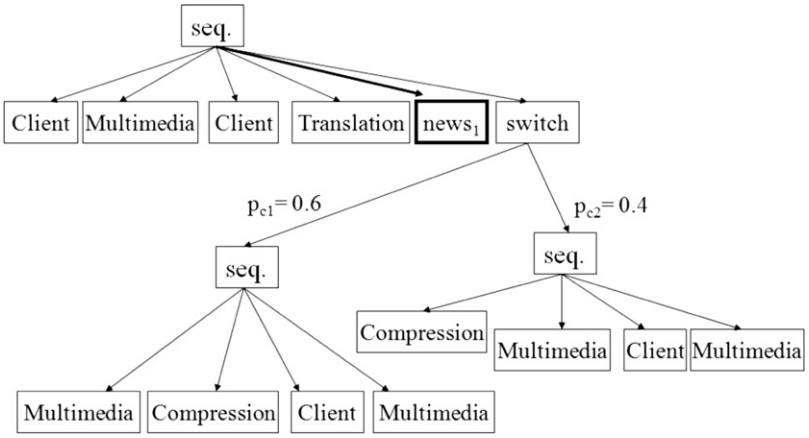

Fig. 16. BPEL tree of Fig. 3 after the application of ap_{11}, ap_{21} and ap_{31}.

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. Typically the adaptation of a service composition is driven by the service selection activity. For example, the approach presented in He et al. (2008) supports the impact of the replacement of a service on other services (e.g., how the change of a service provider impacts on the other services). It is based on workflow patterns and on the value of changed

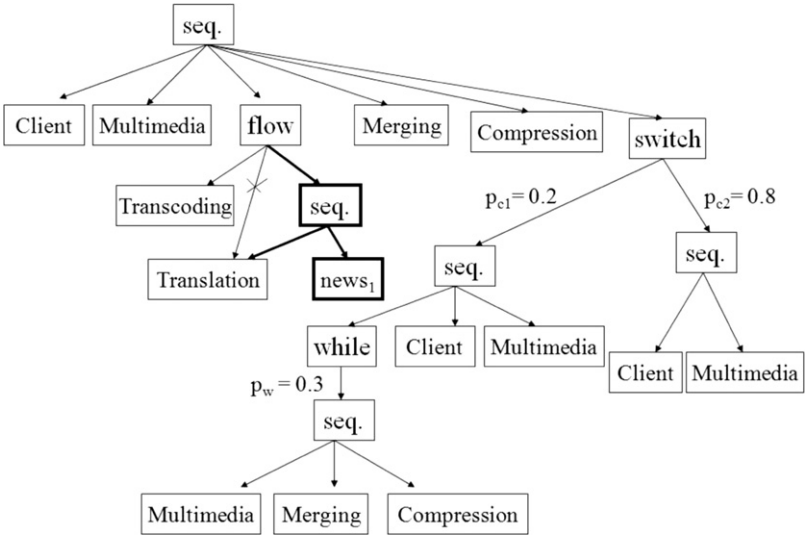

Fig. 17. BPEL tree of Fig. 4 after the application of ap_{11}, ap_{21} and ap_{31}.

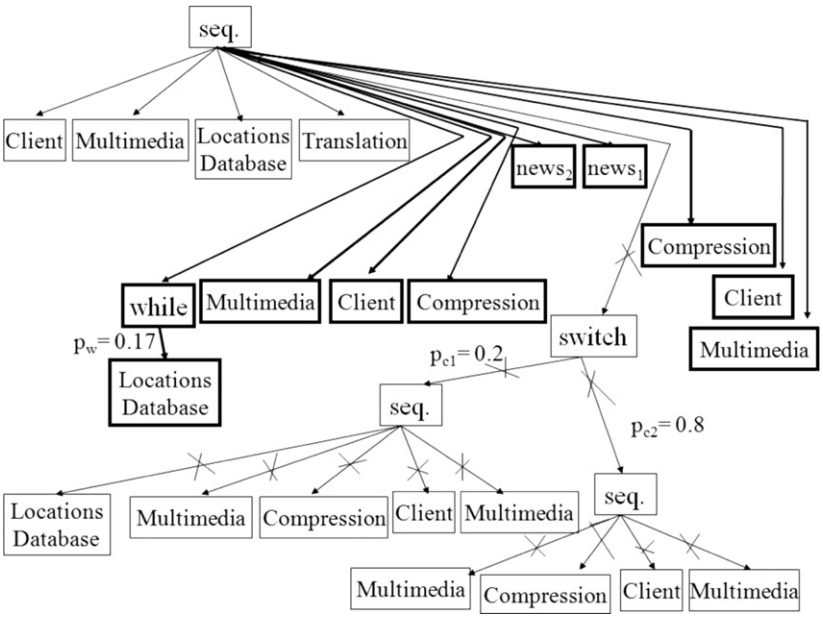

Fig. 18. BPEL tree of Fig. 5 after the application of ap_{11}, ap_{21} and ap_{31}.

information (Harney and Doshi, 2006) for sequence and different parallel pattern scenarios. For example, in order to adapt a system, 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 among existing services have to be changed in order to assure a certain level of the system qualities).

An adaptation approach should change the behavior of a system (i.e., the service composition), based on the right tradeoff among the system software qualities, depending on several factors due mainly to the use for evolution (at re-design time) or self-adaptation (at run time).

Some *frameworks* have been introduced *for dynamically generating a service composition*, e.g., Rao and Su (2004) surveys automated web service composition methods. Usually, they adapt a system only in a proactive way, after the adaptation request has been 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 run-time environment Rosenberg et al., 2009, where a user can require a service composition satisfying quality constraints). Also, they 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, Chiu et al. (2009), where it is presented a framework composing services at run-time for answering a user query, and the survey (Ibrahim and Mouël, 2012), where it is defined a service composition middleware model describing the existing service composition middleware in pervasive environments).

The spontaneous service composition is typically not supported by the existing frameworks, such as the ones for pervasive environments (see survey Ibrahim and Mouël, 2012); instead, it should be one of the fundamental properties of an application. In Ibrahim et al. (2009), a spontaneous service integrator middleware is presented. It allows the extension of system functionalities by integrating new services appearing into the environment with existing services. Moreover, it supports the spontaneous service selection, as services appear (disappear) into the environment. For the selection activity, the authors use 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

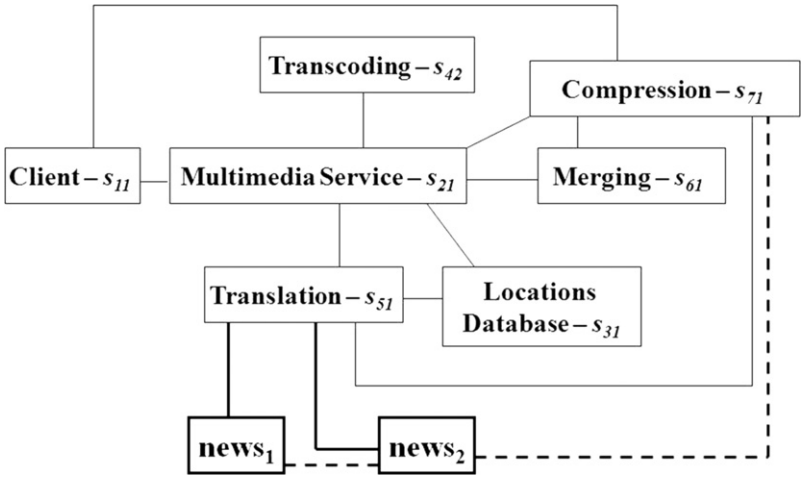

Fig. 19. New Static aspect of the system after the application of ap_{11}, ap_{21}, ap_{31}.

possibility of replacing the other system elementary services (i.e., the service selection with respect to all services). On the contrary, our framework in Mirandola and Potena (2010) (see Section 1) supports the spontaneous adaptation of the hardware, 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.

Other challenges related to the quality impact on software adaptation are represented by the construction of the set of architectural alternatives for each change requirement. For example, in Grunske (2006) a method based on the definition of a multi-objective optimization model is proposed to allow the best architectural decisions to maximize the satisfaction of the quality attributes of the system under certain constraints (such as budget limitations). Different types of methods, mainly based on the use of qualitative approaches, have been proposed (see for example Babar et al., 2004; Kazman et al., 1998), in particular the well-known architecture tradeoff analysis method (Kazman et al., 1998). More quantitative approaches can be found in Martens et al. (2010), where evolutionary techniques based on the use of meta-heuristics are applied starting from an initial solution.

As we remarked in Mirandola and Potena (2010), in order 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 (Garcia et al., 2009) 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 (Kim et al., 2009).

Other approaches for the detection and resolution of mismatches between requirements and existing elementary services/components or among different services/components can also be found in Mohamed et al. (2008) and Tang et al. (2008).

With respect to the state-of-art, the following major aspects characterize the novelty of the proposed method:

- This is the first paper (to the best of our knowledge) that supports the adaptation of a SOA (including both static and dynamic models) by using an optimization model that minimizes the adaptation costs with reliability, availability and performance guarantees.
- The proposed approach does not rely on a specific architectural style, development process or service domain. In the literature several interesting approaches have been supporting the choice of the best architectural style. They usually leverage on the fact that an architectural style guarantees quality attributes at a certain level, but they do not take into account the qualities of single elementary elements (see, for example, Davidsson et al., 2005; Bhattacharya and Perry, 2007). Instead, we suggest how to define a new architecture for the system (i.e., how to change both its structure and behavior) by estimating its quality (i.e., the reliability, availability and response time) on the basis of non-functional properties of single services.
- Our optimization model is independent from the methodology adopted to represent the scenarios (new requirements and adaptation plans) and from the strategy used to generate the adaptation plans for each change requirement.
- In this paper, we have not focused on a particular adaptation phase of the software life-cycle, but it could be instantiated. This adoption may require to specialize (appropriately modify) the model in order to capture typical aspects of the specific phase.
- Finally, our approach can facilitate the work of a maintainer. In fact, in our model it is not necessary to define architectures satisfying all the changes required (as typically done by the state-of-the-art methods), but only possible solutions for each change requirement. Any combination of adaptation actions may have a considerable impact on the cost, reliability/availability and performance of the software architecture. Therefore, our optimization model aims at quantifying such impact to suggest the best adaptation plan, which minimizes the costs while satisfying the reliability, availability and performance constraints. 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 introduction of dependencies among adaptation plans of different change requirements could allow to reduce the research space.

6. Conclusions

In this paper, we have defined an optimization model which supports the choice of the best set of adaptation actions to address certain additional system requirements. The solution of such model provides the set of actions that minimize the adaptation cost while guaranteeing required levels of software reliability, availability, and that a call to the system is satisfied within a certain time.

We have shown how our model works on a smartphone mobile application example, and through the sensitivity analysis we have highlighted its potential to drive architectural decisions.

Other interesting research directions we intend to investigate concern the introduction/evaluation of risk factors associated with change requirements, the evaluation of dependencies among different requirements, the extension of our optimization model to other quality constraints (e.g., security constraint) and the introduction of dependency among services.

As a future work, we intend to improve the special formulation of our model by working on the reliability (availability) expression. In fact, with regard to reliability (availability), we have considered only crash failures, or better failures that immediately and irreversibly compromise the behavior of the whole system. Error propagation may enter the model in order to take into account the probability that an error can be masked across services invocations.

It was out of the scope of this paper to deal with combined formulation of hardware and software aspects, but we want to work on this aspect. We are planning to enhance our optimization model in order to support the adaptation of the hardware architecture by introducing hardware adaptation actions (e.g., the introduction/modification of CPU, disk, memory, network throughput, etc.).

We are implementing a prototype of our adaptation framework (Mirandola and Potena, 2010) (explained in Section 1) by embedding the proposed optimization model into the framework. We intend to specialize the framework (and the model) for supporting particular aspects of an application domain due mainly to its use for evolution (at re-design time) or self-adaptation (at run-time), such as the adaptation of elementary services at run-time.

In particular, in order to support the automatic model generation and solution, we are implementing two main components of the framework, which are a *model builder* and a *model solver*.

The model builder will automatically generate the optimization model by allowing to insert the input parameters of the model, as well as the definition of the adaptation plans will be supported.

The model solver will process the optimization model received from the builder and produce the results. The optimization model solver that we have adopted in this paper is LINGO (see Section 2.10). Currently, we are designing different resolution methodologies (e.g., metaheuristic techniques) in order to allow the maintainers to compare their results or use one of the approaches depending on the system features (e.g., number of services).

Obviously, the framework needs to be validated on larger sized (possibly industrial) case studies. In order to support an

automated application of the framework to large-sized problems, we are implementing an XML-based tool – integrating the model builder and the model solver – that allows to annotate UML diagrams (representing the SOA) with cost, reliability, availability and response time information and that automatically generates and solves the optimization model. The tool will allow to specify ranges for model parameters. Alternative system configurations could be automatically generated by sampling parameters in the given ranges, and solved.

Acknowledgements

I would like to thank the anonymous referees for their comments that helped to substantially improve the quality of the paper. I also am grateful to Vittorio Cortellessa and Raffaela Mirandola for the interesting discussions and collaboration that have inspired this work.

Appendix A. Parameter estimation of the performance constraint in Section 2.8

In this section, we show how to determine the value of the parameters in the second of the expressions for the response time of the k-th external service Rt_k (see Eq. (12)).

A.1. Estimation of the parameter inv'_{ki}

The parameter inv'_{ki} in the first term can be expressed as follows:

$$inv'_{ki} = \sum_{t=1}^{P_{ki}} \sum_{r=1}^{m} \left(\sum_{p=1}^{|AP_r|} y_{rp} \cdot \Delta_{kp}(i,t) \cdot \Delta INV_{kp}(i,t) \cdot \Delta dd_{kp}(i,t) \right) + \left(\left(1 - \sum_{r=1}^{m} \sum_{p=1}^{|AP_r|} y_{rp} \cdot \Delta_{kp}(i,t) \right) \cdot INV_{ki}(t) \cdot dd_{ki}(t) \right) + \sum_{r=1}^{m} \sum_{p=1}^{|AP_r|} \sum_{\bar{t}=1}^{ADP_{kp}(i)} y_{rp} \cdot AINV_{kp}(i,\bar{t}) \cdot Add_{kp}(i,\bar{t}),\ 1 \le i \le n$$

where:

- Δdd_{kp} is a matrix of $n \times \max_i P_{ki}$ elements, where the element $\Delta dd_{kp}(i,t)$ is equal to 1 if after the application of the adaptation plan p the service i is a *direct descendant* of the root of the BPEL k with respect to the path t, and it is equal to 0 otherwise;
- dd_{ki} is a vector of P_{ki} elements, where an entry $dd_{ki}(t)$ is equal to 1 if the service i is a *direct descendant* of the root of the BPEL k with respect to the path t, and it is equal to 0 otherwise;
- Add_{kp} is a $n \times \max_i ADP_{kp}(i)$ matrix, where the element $Add_{kp}(i,t)$ is equal to 1 if the service i is a *direct descendant* of the root of the BPEL k with respect to the new path t, and it is equal to 0 otherwise.

inv'_{ki} depends, in fact, on how the chosen adaptation plans suggest to modify the paths of the BPEL tree k (i.e., on the number of invocations $\Delta INV_{kp}(i,t)$ of the service i along the path t after the application of the plan p) – having the service i as leaf node and *direct descendant* of the root.

Besides, it also depends on the paths $ADP_{kp}(i)$ – having the service i as leaf node and *direct descendant* of the root, which the chosen plans suggest to introduce in the k-th BPEL. Thus, it is also a function of the number of invocations $AINV_{kp}(i,\bar{t})$ of the service i along the $\bar{t}$-th new introduced path.

A.2. Estimation of the parameter inv'_{kh}

The parameter inv'_{kh} in the second term can be expressed as follows:

$$inv'_{kh} = \sum_{r=1}^{m} \left(\sum_{p=1}^{|AP_r|} y_{rp} \left(\sum_{t=1}^{NP_p(h,k)} NINV_{kp}(h,t) \cdot Ndd_{kp}(h,t) \right) \right),\quad 1 \le h \le |NewS|$$

where Ndd_{kp} is a $|NS_p| \times \max_h NP_p(h,k)$ matrix, where the element $Ndd_{kp}(h,t)$ is equal to 1 if the service h is a *direct descendant* of the root of the BPEL k with respect to the path t, and it is equal to 0 otherwise. inv'_{kh} depends, in fact, on the paths $NP_p(h,k)$ – having the service $news_h$ as *direct descendant* of the root, which the chosen adaptation plans suggest to introduce in the k-th BPEL. Thus, it is a function of the number of invocations $NINV_{kp}(h,t)$ of the service $news_h$ along the t-th new introduced path.

A.3. Estimation of parameters $\overline{inv}_{kf}$ and Rt_{fk} in the third term.

Case 1: *Let f be an existing "flow" node.*

Parameters of an existing "flow" node "direct descendant" of the root w:

- the number of invocations ($FINV_k$);
- the response time (FRT_k).

$FINV_k$ is a vector of $|f \in F_k, f \prec_{dd} w|$ elements, where an element $FINV_k(f)$ is equal to the expected number of times the node f is invoked, and it can be easily estimated by parsing the path from the root to the node and multiplying the p_a labels by the arcs along the path.

FRT_k is a matrix of $|f \in F_k, f \prec_{dd} w| \times \max_f |d(f)|$ elements, where $d(f)$ is the set of child nodes of f. An element $FRT_k(f,c)$ is equal to the response time of the child node c. In Appendix A.3.1 we show how to estimate it.

Parameters of an adaptation plan $ap_{rp} \in AP_r$. Table A.9 summarizes the parameters of the adaptation plans for the existing *flow* nodes *direct descendant* of the root.

Table A.9
Parameters of adaptation plans for the existing nodes of type *flow*.

Parameter ID	Parameter specification
ΔF_p	$K \times \max_k \lvert f \in F_k, f \prec_{dd} w \rvert$ matrix, where an element $\Delta F_p(k,f)$ is equal to 1 if the plan p suggests to change the number of invocations of the *flow* node f, and it is equal to 0 otherwise.
$\Delta FINV_p$	$K \times \max_k \lvert f \in F_k, f \prec_{dd} w \rvert$ matrix, where an element $\Delta FINV_p(k,f)$ represents the number of invocations of the *flow* node f after the application of the plan p.
ΔC_{kp}	Matrix of $\lvert f \in F_k, f \prec_{dd} w \rvert \times \max_f \lvert d(f) \rvert$ elements, where $d(f)$ is the set of child nodes of the *flow* node f. An element $\Delta C_{kp}(f,c)$ is equal to 1 if the plan p suggests to change the child node c of f, and it is equal to 0 otherwise.
ΔCRT_{kp}	Matrix of $\lvert f \in F_k, f \prec_{dd} w \rvert \times \max_f \lvert d(f) \rvert$ elements, where an element $\Delta CRT_{kp}(f,c)$ represents the response time of the child node c of f after the application of the node p. In Appendix A.3.1 we show how to estimate it.
$ACRT_{kp}$	Matrix of $\lvert f \in F_k, f \prec_{dd} w \rvert \times \max_f \lvert \bar{d}(f) \rvert$ elements, where $\bar{d}(f)$ is the set of additional child nodes of the node f that the plan p suggests to introduce. An element $ACRT_{kp}(f,c)$ represents the response time of the child node c. In Appendix A.3.1 we show how to estimate it.

Table B.10
Parameters of the available instances for the existing services.

Service ID	Service name	Service altern.	Cost c_{ij}	Reliability r_{ij}	Resp. time rt_{ij}	Num. of inv. first scen. $INV_{1i}(t)$	Direct Disc. $dd_{1i}(t)$	Num. of inv. second scen. $INV_{2i}(t)$	Direct Disc. $dd_{2i}(t)$	Num. of inv. third scen. $INV_{3i}(t)$	Direct Disc. $dd_{3i}(t)$
s_1	*Client*	s_{11}	7	0.9993	1	1	1	1	1	1	1
		s_{12}	9	0.9994	1	1	1	0.2	1	0.2	1
		s_{13}	12	0.99999	1	0.6	1	0.8	1	0.8	1
						0.4	1				
s_2	*Multimedia*	s_{21}	5	0.996	1	1	1	1	1	1	1
	Service	s_{22}	12	0.9995	1	0.6	1	0.084	1	0.2	1
						0.6	1	0.2	1	0.2	1
						0.4	1	0.8	1	0.8	1
						0.4	1			0.8	1
s_3	*Locations Database*	s_{31}	8	0.99985	2					1	1
										0.2	1
s_4	*Transcoding*	s_{41}	7	0.999	1			1	0		
		s_{42}	10	0.99985	1						
		s_{43}	12	0.99999	1						
s_5	*Translation*	s_{51}	4	0.9993	1	1	1	1	0	1	1
		s_{52}	5	0.9998	1						
		s_{53}	6	0.99998	1						
s_6	*Merging*	s_{61}	2	0.94	5			1	1		
		s_{62}	12	0.9997	2			0.084	1		
		s_{63}	14	0.9999	1						
s_7	*Compression*	s_{71}	3	0.99	1	0.6	1	1	1	0.2	1
		s_{72}	9	0.992	1	0.4	1	0.084	1	0.8	1
		s_{73}	15	0.999999	1						

Estimation of $\overline{inv}_{kf}$ and Rt_{fk}. The parameter $\overline{inv}_{kf}$ can be expressed as follows:

$$\overline{inv}_{kf} = \sum_{r=1}^{m}\left(\sum_{p=1}^{|AP_r|} y_{rp}\cdot\Delta F_p(k,f)\cdot\Delta FINV_p(k,f)\right) + \left((1-\sum_{r=1}^{m}(\sum_{p=1}^{|AP_r|} y_{rp}\cdot\Delta F_p(k,f)))\cdot FINV_k(f)\right)$$

$\overline{inv}_{kf}$ depends, in fact, on how the chosen adaptation plans modify the node f. If a plan p suggests to modify f (i.e., if $\Delta F_p(k,f)$ is equal to 1), then $\overline{inv}_{kf}$ depends on the number of invocations $\Delta FINV_p(k,f)$ of the node f after the application of the path p, otherwise (i.e., if the chosen adaptation plans do not modify f) it is a function of the number $FINV_k(f)$ of times that f is invoked in the BPEL k.

The average response time Rt_{fk} of the *flow* node f can be obtained by means of a recursive procedure. Computation is carried out by visiting in postorder the process activity tree. Such an algorithm gives the largest response time among the component activities of the node f (i.e., the largest response time Rt_{ck} of the child nodes c of f). Therefore, the parameter Rt_{fk} can be expressed as follows:

$$Rt_{fk} = \max_{c\in d'(f)} Rt_{ck}$$

where $d'(f)$ is the set of child nodes of the node f after the adaptation phase. $d'(f)$ is composed by the existing child nodes of f (i.e., $d(f)$) and the additional child nodes of f (i.e., $\overline{d}(f)$), which the chosen plans suggest to introduce in the BPEL k.

The response time Rt_{ck} of the child node c can be expressed as follows:

$$Rt_{ck} = \begin{cases} \sum_{r=1}^{m}\left(\sum_{p=1}^{|AP_r|} y_{rp}\cdot\Delta C_{kp}(f,c)\cdot\Delta CRT_{kp}(f,c)\right) + & \\ +\left(\left(1-\sum_{r=1}^{m}(\sum_{p=1}^{|AP_r|} y_{rp}\cdot\Delta C_{kp}(f,c))\right)\cdot FRT_k(f,c)\right) & c\in d(f) \\ y_{rp}\cdot ACRT_{kp}(f,c) & c\in\overline{d}(f) \end{cases}$$

Rt_{ck} depends, in fact, on how the chosen adaptation plans suggest to modify the node c.

c is an exiting node: If a plan p suggests to modify c (i.e., if $\Delta C_{kp}(f,c)$ is equal to 1), then Rt_{ck} is equal to the response time $\Delta CRT_{kp}(f,c)$ suggested by the plan p, otherwise it is equal to the response time $FRT_k(f,c)$.

c is a new node added by the plan p: Rt_{ck} is equal to the response time $ACRT_{kp}(f,c)$ suggested by the plan p.

Case 2:Let f be a new node introduced by the plan p selected for the requirement r

Estimation of $\overline{inv}_{kf}$ and Rt_{fk}. The parameter $\overline{inv}_{kf}$ can be expressed as follows:

$$\overline{inv}_{kf} = y_{rp}\cdot nFINV_p(k,f)$$

$nFINV_p$ is a $K\times \max_k nF_{kp}$ matrix, where: (1) nF_{kp} is the number of additional "flow" nodes *direct descendant* of the root that the plan p suggests to add in the BPEL k; and, (2) an element $nFINV_p(k,f)$ represents the number of invocations of the additional node f.

The parameter Rt_{fk} can be estimated as follows:

$$Rt_{fk} = y_{rp}\cdot nFRT_p(k,f)$$

$nFRT_p$ is a $K\times \max_k nF_{kp}$ matrix, where an element $nFRT_p(k,f)$ represents the response time of the node f. In Appendix A.3.1 we show how to estimate it.

A.3.1. Parameters estimation of Cases 1 and 2

The value of response times $\Delta CRT_{kp}(f, c)$, $FRT_k(f, c)$, $ACRT_{kp}(f, c)$ and $nFRT_p(k, f)$, resulting after the application of the chosen adaptation plans, can be estimated by using the following general expressions, which we have derived by enhancing the *Theorem 1* of Cardellini et al. (2007) that predicts the response time of an activity l of a BPEL tree.

For an activity $l \in V_k$, and $k \in K$, the response time Rt_{lk} is:

$$Rt_{lk} = \begin{cases} \max_{l' \in \bar{d}(l)} Rt_{l'k} & l \in \bar{F}_k \\ \sum_{i=1, i \prec_{dd} l}^{n} \frac{inv''_{ki}}{\overline{inv}_{kl}} \sum_{j=1}^{|Inst_i|} x_{ij} rt_{ij} + & \\ + \sum_{h=1, h \prec_{dd} l}^{|NewS|} \frac{inv''_{kh}}{\overline{inv}_{kl}} z_h \overline{rt}_h + & \\ + \sum_{f \in \bar{F}_k, f \prec_{dd} l} \frac{\overline{inv}_{kf}}{\overline{inv}_{kl}} Rt_{fk} & l \notin \bar{F}_k \end{cases}$$

where:

- inv''_{ki} is the average number of invocations of the existing service i (a *direct descendant* of the node l). It can be easily estimated by parsing the paths, where the service is a *direct descendant*, and multiplying the p_a labels by the arcs along the paths.
- $\overline{inv}_{kl}$ is the average number of invocations of the node l, and it can be easily estimated by parsing the path from the root to l and multiplying the p_a labels by the arcs along the path.
- inv''_{kh} is the average number of invocations of the new service h (a *direct descendant* of the node l). It can be easily estimated by parsing the paths, where the service is a *direct descendant*, and multiplying the p_a labels by the arcs along the paths.
- $\overline{inv}_{kf}$ is the expected number of times that the node f is visited, and it can be easily estimated by parsing the path from the root to f and multiplying the p_a labels by the arcs along the path.

Table B.11
Parameters of the new available services.

Service ID	Cost $\bar{c}_h$	Reliability $\bar{r}_h$	Response time $\overline{rt}_h$
$news_1$	7	0.99999	1
$news_2$	6	0.9992	1
$news_3$	4	0.99994	1
$news_4$	7	0.91	1

Appendix B. Model parameters of the example in Section 3

Table B.10 shows the parameter values of the available instances for the existing services, likewise Table B.11 does for the new services that can be adopted into the system.

Table B.12
Existing paths parameters with respect to the adaptation plans in the first BPEL process.

Service ID	Service name	Adapt. Plan ID ap_{1p}	Ex. Path Par. (ΔINV_{1p}, Δ_{1p}, Δdd_{1p})	New Path Par. ($AINV_{1p}$, Add_{1p})	Adapt. Plan ID ap_{2p}	Ex. Path Par. (ΔINV_{1p} Δ_{1p}, Δdd_{1p})	New Path Par. ($AINV_{1p}$, Add_{1p})	Adapt. Plan ID ap_{3p}	Ex. Path Par. (ΔINV_{1p} Δ_{1p},	New Path Par. ($AINV_{1p}$ Add_{1p} Δdd_{1p})
s_1	*Client*	ap_{11}	(1,0,1) (1,0,1) (0.6,0,1) (0.4,0,1)		ap_{21}	(1,0,1) (1,0,1) (0.6,0,1) (0.4,0,1)		ap_{31}	(1,0,1) (1,0,1) (0.6,0,1) (0.4,0,1)	
		ap_{12}	(1,0,1) (1,0,1) (0.6,0,1) (0.4,0,1)	(1,0)	ap_{22}	(1,0,1) (1,0,1) (0.6,0,1) (0.4,0,1)		ap_{32}	(1,0,1) (1,0,1) (0.6,0,1) (0.4,0,1)	
					ap_{23}	(1,0,1) (1,0,1) (0.6,0,1) (0.4,0,1)		ap_{33}	(1,0,1) (1,0,1) (0.6,0,1) (0.4,0,1)	
s_2	*Multimedia*	ap_{11}	(1,0,1) (0.6,0,1)		ap_{21}	(1,0,1) (0.6,0,1)		ap_{31}	(1,0,1) (0.6,0,1)	
	Service		(0.6,0,1)(0.4,0,1) (0.4,0,1)			(0.6,0,1)(0.4,0,1) (0.4,0,1)			(0.6,0,1)(0.4,0,1) (0.4,0,1)	
		ap_{12}	(1,0,1) (0.6,0,1) (0.6,0,1)(0.4,0,1) (0.4,0,1)		ap_{22}	(1,0,1) (0.6,0,1) (0.6,0,1) (0.4,0,1) (0.4,0,1)		ap_{32}	(1,0,1) (0.6,0,1) (0.6,0,1)(0.4,0,1) (0.4,0,1)	
					ap_{23}	(1,0,1) (0.6,0,1) (0.6,0,1) (0.4,0,1) (0.4,0,1)		ap_{33}	(1,0,1) (0.6,0,1) (0.6,0,1)(0.4,0,1) (0.4,0,1)	
s_3	*Locations*	ap_{11}			ap_{21}			ap_{31}		
	Database	ap_{12}			ap_{22}			ap_{32}		
					ap_{23}			ap_{33}		
s_4	*Transcoding*	ap_{11}			ap_{21}			ap_{31}		
		ap_{12}			ap_{22}			ap_{32}		
					ap_{23}			ap_{33}		
s_5	*Translation*	ap_{11}	(1,0,1)		ap_{21}	(1,0,1)		ap_{31}	(1,0,1)	
		ap_{12}	(1,0,1)		ap_{22}	(1,0,1)		ap_{32}	(1,0,1)	
					ap_{23}	(1,0,1)		ap_{33}	(1,0,1)	
s_6	*Merging*	ap_{11}			ap_{21}			ap_{31}		
		ap_{12}			ap_{22}			ap_{32}		
					ap_{23}			ap_{33}		
s_7	*Compression*	ap_{11}	(0.6,0,1) (0.4,0,1)		ap_{21}	(0.6,0,1) (0.4,0,1)		ap_{31}	(0.6,0,1) (0.4,0,1)	
		ap_{12}	(0.6,0,1) (0.4,0,1)		ap_{22}	(0.6,0,1) (0.4,0,1)		ap_{32}	(0.6,0,1) (0.4,0,1)	
					ap_{23}	(0.6,0,1) (0.4,0,1)		ap_{33}	(0.6,0,1) (0.4,0,1)	

Table B.13
Existing paths parameters with respect to the adaptation plans in the second BPEL process.

Service ID	Service name	Adapt. Plan ID ap_{1p}	Ex. Path Par. (ΔINV_{2p} Δ_{2p}, Δdd_{2p})	New Path Par. ($AINV_{2p}$, Add_{2p})	Adapt. Plan ID ap_{2p}	Ex. Path Par. (ΔINV_{2p} Δ_{2p}, Δdd_{2p})	New Path Par. ($AINV_{2p}$, Add_{2p})	Adapt. Plan ID ap_{3p}	Ex. Path Par. (ΔINV_{2p} Δ_{2p}, Δdd_{2p})	New Path Par. ($AINV_{2p}$, Add_{2p})
s_1	*Client*	ap_{11}	(1,0,1) (0.2,0,1) (0.8,0,1)		ap_{21}	(1,0,1) (0.2,0,1) (0.8,0,1)		ap_{31}	(1,0,1) (0.2,0,1) (0.8,0,1)	
		ap_{12}	(1,0,1) (0.2,0,1) (0.8,0,1)	(1,0)	ap_{22}	(1,0,1) (0.2,0,1) (0.8,0,1)		ap_{32}	(1,0,1) (0.2,0,1) (0.8,0,1)	
					ap_{23}	(1,0,1) (0.2,0,1) (0.8,0,1)		ap_{33}	(1,0,1) (0.2,0,1) (0.8,0,1)	
s_2	*Multimedia*	ap_{11}	(1,0,1) (0.084,0,1)		ap_{21}	(1,0,1) (0.084,0,1)		ap_{31}	(1,0,1) (0.084,0,1)	
	Service		(0.2,0,1) (0.8,0,1)			(0.2,0,1) (0.8,0,1)			(0.2,0,1) (0.8,0,1)	
		ap_{12}	(1,0,1) (0.084,0,1) (0.2,0,1) (0.8,0,1)		ap_{22}	(1,0,1) (0.084,0,1) (0.2,0,1) (0.8,0,1)		ap_{32}	(1,0,1) (0.084,0,1) (0.2,0,1) (0.8,0,1)	
					ap_{23}	(1,0,1) (0.084,0,1) (0.2,0,1) (0.8,0,1)		ap_{33}	(1,0,1) (0.084,0,1) (0.2,0,1) (0.8,0,1)	
s_3	*Locations* *Database*	ap_{11} ap_{12}			ap_{21} ap_{22} ap_{23}			ap_{31} ap_{32} ap_{33}		
s_4	*Transcoding*	ap_{11}	(1,0,0)		ap_{21}	(1,0,0)		ap_{31}	(1,0,0)	
		ap_{12}	(1,0,0)		ap_{22}	(0,1,)	(1,0)	ap_{32}	(1,0,0)	
					ap_{23}	(0,1,)	(1,0)	ap_{33}	(1,0,0)	
s_5	*Translation*	ap_{11}	(0,1,)	(1,0)	ap_{21}	(1,0,0)		ap_{31}	(1,0,0)	
		ap_{12}	(0,1,)	(1,0)	ap_{22}	(1,0,0)		ap_{32}	(1,0,0)	
					ap_{23}	(1,0,0)		ap_{33}	(1,0,0)	
s_6	*Merging*	ap_{11}	(1,0,1) (0.084,0,1)		ap_{21}	(1,0,1) (0.084,0,1)		ap_{31}	(1,0,1) (0.084,0,1)	
		ap_{12}	(1,0,1) (0.084,0,1)		ap_{22}	(1,0,1) (0.084,0,1)		ap_{32}	(1,0,1) (0.084,0,1)	
					ap_{23}	(1,0,1) (0.084,0,1)		ap_{33}	(1,0,1) (0.084,0,1)	
s_7	*Compression*	ap_{11}	(1,0,1) (0.084,0,1)		ap_{21}	(1,0,1) (0.084,0,1)		ap_{31}	(1,0,1) (0.084,0,1)	
		ap_{12}	(1,0,1) (0.084,0,1)		ap_{22}	(1,0,1) (0.084,0,1)		ap_{32}	(1,0,1) (0.084,0,1)	
					ap_{23}	(1,0,1) (0.084,0,1)		ap_{33}	(1,0,1) (0.084,0,1)	

The third column in Table B.10 lists the set of alternatives for each existing service. For each alternative: the cost c_{ij} (kE) is given in the fourth column; the reliability r_{ij} is given in the fifth column; the response time rt_{ij} (s) is given in the sixth column; the number of times $INV_{1i}(t)$ that the service i is invoked in the path t (in the first BPEL process) is given in the seventh column; the vector dd_{1i} that indicates if the service i is a *direct descendant* (with respect to the paths of the first BPEL) is given in the eighth column; the number of times $INV_{2i}(t)$ is given in the ninth column; the vector dd_{2i} is given in the tenth column; the number of times $INV_{3i}(t)$ is given in the eleventh column; the vector dd_{3i} is given in the twelfth column.

The first column in Table B.11 lists the new services available. For each new service: the cost $\overline{c}_h$ (kE) is given in the second column, the reliability $\overline{r}_h$ is given in the third column, and the response time $\overline{rt}_h$ (s) is given in the fourth column.

Tables B.12–B.17 summarize how the adaptation plans available for each change requirement suggest to change the system behavior (i.e., to remove/introduce interaction(s) between existing services and/or new services).

The second column in Table B.12 lists the name of the existing services. For each service i: the parameters (ΔINV_{1p}, Δ_{1p}, Δdd_{1p}) of its existing paths in the first BPEL process, for adaptation plans available for the first change requirement (i.e., ap_{11} and ap_{12}), are given in the fourth column; the parameters ($AINV_{1p}$, Add_{1p}) of its new paths (in the first BPEL process), for adaptation plans available for the first change requirement, are given in the fifth column; the parameters of its existing paths (in the first BPEL process), for adaptation plans available for the second change requirement (i.e., ap_{21}, ap_{22} and ap_{23}), are given in the seventh column; the parameters of its new paths (in the first BPEL process), for adaptation plans available for the second change requirement, are given in the eighth column; the parameters of its existing paths (in the first BPEL process), for adaptation plans available for the third change requirement (i.e., ap_{31}, ap_{32} and ap_{33}), are given in the tenth column; the parameters of its new paths (in the first BPEL process), for adaptation plans available for the third change requirement, are given in the eleventh column.

The second column in Table B.13 lists the name of the existing services. For each service i: the parameters (ΔINV_{2p}, Δ_{2p}, Δdd_{2p}) of its existing paths in the second BPEL process, for adaptation plans available for the first change requirement, are given in the fourth column; the parameters ($AINV_{2p}$, Add_{2p}) of its new paths in the second BPEL process, for adaptation plans available for the first change requirement, are given in the fifth column; the parameters of its existing paths in the second BPEL process, for adaptation plans available for the second change requirement, are given in the seventh column; the parameters of its new paths in the second BPEL process, for adaptation plans available for the second change requirement, are given in the eighth column; the parameters of its existing paths in the second BPEL process, for adaptation plans available for the third change requirement, are given in the tenth column; the parameters of its new paths in the second BPEL process, for adaptation plans available for the third change requirement, are given in the eleventh column.

Table B.14
Existing paths parameters with respect to the adaptation plans in the third BPEL process.

Service ID	Service name	Adapt. Plan ID ap_{1p}	Ex. Path Par. (ΔINV_{3p} Δ_{3p}, Δdd_{3p})	New Path Par. ($AINV_{3p}$, Add_{3p})	Adapt. Plan ID ap_{2p}	Ex. Path Par. (ΔINV_{3p} Δ_{3p}, Δdd_{3p})	New Path Par. ($AINV_{3p}$, Add_{3p})	Adapt. Plan ID ap_{3p}	Ex. Path Par. (ΔINV_{3p} Δ_{3p}, Δdd_{3p})	New Path Par. ($AINV_{3p}$, Add_{3p})
s_1	*Client*	ap_{11}	(1,0,1) (0.2,0,1) (0.8,0,1)		ap_{21}	(1,0,1) (0.2,0,1) (0.8,0,1)		ap_{31}	(1,0,1) (0,1,) (0,1,)	(1,1) (1,1)
		ap_{12}	(1,0,1) (0.2,0,1) (0.8,0,1)	(1,0)	ap_{22}	(1,0,1) (0.2,0,1) (0.8,0,1)		ap_{32}	(1,0,1) (0.2,0,1) (0.8,0,1)	
					ap_{23}	(1,0,1) (0.2,0,1) (0.8,0,1)		ap_{33}	(1,0,1) (0.2,0,1) (0.8,0,1)	
s_2	*Multimedia*	ap_{11}	(1,0,1) (0.2,0,1)		ap_{21}	(1,0,1) (0.2,0,1)		ap_{31}	(1,0,1) (0,1,) (0,1,)	(1,1) (1,1)
	Service		(0.2,0,1) (0.8,0,1) (0.8,0,1)			(0.2,0,1) (0.8,0,1) (0.8,0,1)			(0,1,) (0,1,)	
		ap_{12}	(1,0,1) (0.2,0,1) (0.2,0,1) (0.8,0,1) (0.8,0,1)		ap_{22}	(1,0,1) (0.2,0,1) (0.2,0,1) (0.8,0,1) (0.8,0,1)		ap_{32}	(1,0,1) (0.2,0,1) (0.2,0,1) (0.8,0,1) (0.8,0,1)	
					ap_{23}	(1,0,1) (0.2,0,1) (0.2,0,1) (0.8,0,1) (0.8,0,1)		ap_{33}	(1,0,1) (0.2,0,1) (0.2,0,1) (0.8,0,1) (0.8,0,1)	
s_3	*Locations*	ap_{11}	(1,0,1) (0.2,0,1)		ap_{21}	(1,0,1) (0.2,0,1)		ap_{31}	(1,0,1) (0,1,)	(0.2,1)
	Database	ap_{12}	(1,0,1) (0.2,0,1)		ap_{22}	(1,0,1) (0.2,0,1)		ap_{32}	(1,0,1) (0.2,0,1)	
					ap_{23}	(1,0,1) (0.2,0,1)		ap_{33}	(1,0,1) (0.2,0,1)	
s_4	*Transcoding*	ap_{11}			ap_{21}			ap_{31}		
		ap_{12}			ap_{22}			ap_{32}		
					ap_{23}			ap_{33}		
s_5	*Translation*	ap_{11}	(1,0,1)		ap_{21}	(1,0,1)		ap_{31}	(1,0,1)	
		ap_{12}	(1,0,1)		ap_{22}	(1,0,1)		ap_{32}	(1,0,1)	
					ap_{23}	(1,0,1)		ap_{33}	(1,0,1)	
s_6	*Merging*	ap_{11}			ap_{21}			ap_{31}		
		ap_{12}			ap_{22}			ap_{32}		
					ap_{23}			ap_{33}		
s_7	*Compression*	ap_{11}	(0.2,0,1) (0.8,0,1)		ap_{21}	(0.2,0,1) (0.8,0,1)		ap_{31}	(0,1,) (0,1,)	(1,1) (1,1)
		ap_{12}	(0.2,0,1) (0.8,0,1)		ap_{22}	(0.2,0,1) (0.8,0,1)		ap_{32}	(0.2,0,1) (0.8,0,1)	
					ap_{23}	(0.2,0,1) (0.8,0,1)		ap_{33}	(0.2,0,1) (0.8,0,1)	

The second column in Table B.14 lists the name of the existing services. For each service i: the parameters (ΔINV_{3p}, Δ_{3p}, Δdd_{3p}) of its existing paths in the third BPEL process, for adaptation plans available for the first change requirement, are given in the fourth column; the parameters of its new paths in the third BPEL process, for adaptation plans available for the first change requirement, are given in the fifth column; the parameters of its existing paths in the third BPEL process, for adaptation plans available for the second change requirement, are given in the seventh column; the parameters of its new paths in the third BPEL process, for adaptation plans available for the second change requirement, are given in the eighth column; the parameters of its existing paths in the third BPEL process, for adaptation plans available for the third change requirement, are given in the tenth column; the parameters of its new paths in the third BPEL process, for adaptation plans available for the third change requirement, are given in the eleventh column.

Table B.15
New services paths parameters with respect to the adaptation plans.

Service ID	Adapt. Plan ID ap_{1p}	Path Par. ($NINV_{1p}$ Ndd_{1p})	Path Par. ($NINV_{2p}$ Ndd_{2p})	Path Par. ($NINV_{3p}$ Ndd_{3p})	Adapt. Plan ID ap_{2p}	Path Par. ($NINV_{1p}$ Ndd_{1p})	Path Par. ($NINV_{2p}$ Ndd_{2p})	Path Par. ($NINV_{3p}$ Ndd_{3p})	Adapt. Plan ID ap_{3p}	Path Par. ($NINV_{1p}$ Ndd_{1p})	Path Par. ($NINV_{2p}$ Ndd_{2p})	Path Par. ($NINV_{3p}$ Ndd_{3p})
$news_1$	ap_{11}	(1,1)	(1,0)	(1,1)	ap_{21}				ap_{31}			
	ap_{12}				ap_{22}				ap_{32}			(1,1)
					ap_{23}				ap_{33}			
$news_2$	ap_{11}				ap_{21}				ap_{31}			(1,1)
	ap_{12}				ap_{22}				ap_{32}			
					ap_{23}				ap_{33}			
$news_3$	ap_{11}				ap_{21}				ap_{31}			
	ap_{12}				ap_{22}		(1,0)		ap_{32}			
					ap_{23}				ap_{33}			
$news_4$	ap_{11}				ap_{21}				ap_{31}			
	ap_{12}	(1,0)	(1,0)	(1,0)	ap_{22}				ap_{32}			
					ap_{23}		(1,0)		ap_{33}			

Table B.16
Model parameters for existing nodes of type *flow*.

Node ID	Numb. of invoc. $FINV_2$	Resp. time FRT_2	Adapt. Plan ID ap_{1p}	Numb. of invoc. $\Delta FINV_p$	Resp. time (ΔCRT_{2p}), $(ACRT_{2p})$	Adapt. Plan ID ap_{2p}	Numb. of invoc. $\Delta FINV_p$	Resp. time (ΔCRT_{2p}), $(ACRT_{2p})$	Adapt. Plan ID ap_{3p}	Numb. of invoc. $\Delta FINV_p$	Resp. time (ΔCRT_{2p}), $(ACRT_{2p})$
1	1	1 1	ap_{11}	1	(1 0), (2)	ap_{21}			ap_{31}		
			ap_{12}	1	(1 0), (2)	ap_{22}	1	(0 1) (2)	ap_{32}		
						ap_{23}	1	(0 1) (2)	ap_{33}		

B.17
Model parameters for new nodes of type *flow*.

Node ID	Numb. of invoc. $nFINV_p(1,f)$	Resp. time $nFRT_p(1,f)$	Numb. of invoc. $nFINV_p(3,f)$	Resp. time $nFRT_p(3,f)$
1	1	1	1	1

The first column in Table B.15 lists the name of the new available services. For each service h: the parameters ($NINV_{kp}$, Ndd_{kp}) for the adaptation plans available for the first change requirement (i.e., ap_{11} and ap_{12}) in the first, second and third BPEL process, are given respectively in the third, fourth and fifth column; the parameters ($NINV_{kp}$, Ndd_{kp}) for the adaptation plans available for the second change requirement (i.e., ap_{21}, ap_{22} and ap_{23}) in the first, second and third BPEL process, are given respectively in the seventh, eighth and ninth column; the parameters ($NINV_{kp}$, Ndd_{kp}) for the adaptation plans available for the third change requirement (i.e., ap_{31}, ap_{32} and ap_{33}) in the first, second and third BPEL process, are given respectively in the eleventh, twelfth and thirteenth column.

The first column in Table B.16 lists the existing node of type *flow* of the second BPEL process. For each node f: the number of time the node f is invoked $FINV_2(f)$ is given in the second column; the response time FRT_2 of the child nodes of f is given in the third column; the number of invocations $\Delta FINV_p(1,f)$ of the node f after the application of the plan p (available for the first change requirement) is given in the fifth column; the response time (ΔCRT_{1p}, $ACRT_{1p}$) of the child nodes of f after the application of the plans for the first change requirement is given in the sixth column; the number of invocations $\Delta FINV_p(2,f)$ of the node f after the application of the plan p is given in the eighth column; the response time (ΔCRT_{2p}, $ACRT_{2p}$) of the child nodes of f after the application of the adaptation plans available for the second change requirement is given in the ninth column; the number of invocations $\Delta FINV_p(3,f)$ of the node f after the application of the plan p is given in the eleventh column; the response time (ΔCRT_{3p}, $ACRT_{3p}$) of the child nodes of f is given in the twelfth column.[12]

The first column in Table B.17 lists the new nodes of type *flow direct descendant* of the root of the BPEL process k, which the adaptation plan ap_{12}, suggest to introduce.[13] For each node f: the number of invocations $nFINV_p(1,f)$ in the first BPEL process is given in the second column; the response time $nFRT_p(1,f)$ in the first BPEL process is given in the third column; the number of invocations $nFINV_p(3,f)$ in the third BPEL process is given in the third column; the response time $nFRT_p(3,f)$ in the third BPEL process is given in the fourth column.

References

Alrifai, M., Risse, T., 2009. Combining global optimization with local selection for efficient QoS-aware service composition. In: WWW, pp. 881–890.

André, F., Daubert, E., Gauvrit, G., 2010. Towards a generic context-aware framework for self-adaptation of service-oriented architectures. In: Proceedings of the Fifth International Conference on Internet and Web Applications and Services.

Andrews, T., Curbera, F., Dholakia, H., Goland, Y., Klein, J., Leymann, F., Liu, K., Roller, D., Smith, D., Thatte, S., Trickovic, I., Weerawarana, S., 2003. Business Process Execution Language for Web Services, Version 1.1, May.

Babar, M., Zhu, L., Jeffery, D., 2004. A framework for classifying and comparing software architecture evaluation methods. In: Proceedings of Australian Software Engineering Conference, pp. 309–319.

Balsamo, S., Di Marco, A., Inverardi, P., Simeoni, M., 2004. Model-based performance prediction in software development: a survey. IEEE Transactions on Software Engineering 30 (5), 295–310.

Bang, J., Popescu, D., Edwards, G., Medvidovic, N., Kulkarni, N., Rama, G., Padmanabhuni, S., 2010. CoDesign: a highly extensible collaborative software modeling framework. In: ICSE '10: Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering, pp. 243–246.

Bass, L., Klein, M., Bachmann, F., 2001. Quality attribute design primitives and the attribute driven design method. In: PFE, pp. 169–186.

Becker, S., Grunske, L., Mirandola, R., Overhage, S., 2004. Performance prediction of component-based systems—a survey from an engineering perspective. In: Architecting Systems with Trustworthy Components, pp. 169–192.

Bhattacharya, S., Perry, D., 2007. Predicting emergent properties of component based systems. In: ICCBSS '07: Proceedings of the Sixth International IEEE Conference on Commercial-off-the-Shelf (COTS)-Based Software Systems, pp. 41–50.

Boehm, B.W., 1981. Software Engineering Economics, 1st ed. Prentice Hall PTR, Upper Saddle River, NJ, USA.

Bucchiarone, A., Cappiello, C., Di Nitto, E., Kazhamiakin, R., Mazza, V., Pistore, M., 2010. Design for adaptation of service-based applications: main issues and requirements. In: ICSOC/ServiceWave 2009 Workshops, LNCS, pp. 467–476.

Calinescu, R., Kwiatkowska, M., 2009. Using quantitative analysis to implement autonomic IT systems. In: ICSE, pp. 100–110.

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

Cardellini, V., Casalicchio, E., Grassi, V., Presti, F.L., 2007. Flow-based service selection for web service composition supporting multiple QoS classes. In: ICWS, pp. 743–750.

Cardoso, J., Sheth, A., Miller, J., Arnold, J., Kochut, K., 2004. Modeling quality of service for workflows and web service processes. Journal of Web Semantics 1 (3), 281–308.

Chiu, D., Deshpande, S., Agrawal, G., Li, R., 2009. A dynamic approach toward QoS-Aware service workflow composition. In: ICWS, pp. 655–662.

Cortellessa, V., Potena, P., 2007. Path-based error propagation analysis in composition of software services. In: Software Composition, pp. 97–112.

Cortellessa, V., Marinelli, F., Potena, P., 2006. Automated selection of software components based on cost/reliability tradeoff. In: EWSA, pp. 66–81.

Cortellessa, V., Mirandola, R., Potena, P., 2010. Selecting optimal maintenance plans based on cost/reliability tradeoffs for software subject to structural and behavioral changes. In: Proceedings of the 14th European Conference on Software Maintenance and Reengineering (CSMR), IEEE, pp. 21–30.

Davidsson, P., Johansson, S., Svahnberg, M., 2005. Characterization and evaluation of multi-agent system architectural styles. In: SELMAS, pp. 179–188.

Di Nitto, E., Ghezzi, C., Metzger, A., Papazoglou, M., Pohl, K., 2008. A journey to highly dynamic, self-adaptive service-based applications. Automated Software Engineering 15 (3–4), 313–341.

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

Garcia, J., Popescu, D., Edwards, G., Medvidovic, N., 2009. Toward a Catalogue of Architectural Bad Smells. In: QoSA, pp. 146–162.

[12] If a plan does not change the node, the entries of its parameter values are empty.
[13] Only the plan ap_{12} suggests to introduce *flow* nodes.

Grunske, L., 2006. Identifying "good" architectural design alternatives with multi-objective optimization strategies. In: ICSE, pp. 849–852.
Guo, H., Huai, J., Li, H., Deng, T., Li, Y., Du, Z., 2007. ANGEL: optimal configuration for high available service composition. In: IEEE International Conference on Web Services, pp. 280–287.
Harney, J., Doshi, P., 2006. Adaptive web processes using value of changed information. In: International Conference on Service-Oriented Computing (ICSOC), pp. 179–190.
He, Q., Yan, J., Jin, H., Yang, Y., 2008. Adaptation of web service composition based on workflow patterns. In: ICSOC.
Ibrahim, N., Mouël, F.L., 2012. A Survey on Service Composition Middleware in Pervasive Environments. CoRR abs/0909.2183.
Ibrahim, N., Mouël, F.L., Frénot, S., 2009. MySIM: a spontaneous service integration middleware for pervasive environments. In: ICPS '09: Proceedings of the 2009 International Conference on Pervasive Services, pp. 1–10.
Immonen, A., Niemelä, E., 2008. Survey of reliability and availability prediction methods from the viewpoint of software architecture. Software and System Modeling 7 (1), 49–65.
Jung, H.-W., Choi, B., 1999. Optimization models for quality and cost of modular software systems. European Journal of Operational Research 112 (3), 613–619.
Kazman, R., Klein, M., Barbacci, M., Longstaff, T., Lipson, H., Carrière, S., 1998. The architecture tradeoff analysis method. In: ICECCS, pp. 68–78.
Kim, S., Kim, D., Lu, L., Park, S., 2009. Quality-driven architecture development using architectural tactics. Journal of Systems and Software 82 (8), 1211–1231.
Martens, A., Koziolek, H., Becker, S., Reussner, R., 2010. Automatically improve software models for performance reliability and cost using genetic algorithms. In: Proceedings of WOSP/SIPEW.
Mirandola, R., Potena, P., 2010. Self-adaptation of service based systems based on cost/quality attributes tradeoffs. In: Proceedings of the 12th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNACS 2010)—Workshop on Software Services: Frameworks and Platforms, pp. 493–501.
Mohamed, A., Ruhe, G., Eberlein, A., 2008. Sensitivity analysis in the process of COTS mismatch-handling. Requirements Engineering 13 (2), 147–165.
Musa, J., 1993. Operational profiles in software-reliability engineering software. IEEE 10 (2), 14–32.
Oh, M., Baik, J., Kang, S., Choi, H., 2008. An efficient approach for QoS-aware service selection based on a tree-based algorithm. In: ACIS-ICIS, pp. 605–610.
[Online]. Available: www.lindo.com
Rao, J., Su, X., 2004. A survey of automated web service composition methods. In: SWSWPC, pp. 43–54.
Rosenberg, F., Celikovic, P., Michlmayr, A., Leitner, P., Dustdar, S., 2009. An end-to-end approach for QoS-Aware Service composition. In: EDOC, pp. 151–160.
Rosenberg, F., Müller, M., Leitner, P., Michlmayr, A., Bouguettaya, A., Dustdar, S., 2010. Metaheuristic optimization of large-scale QoS-aware service compositions. In: Proceedings of the 2010 IEEE International Conference on Services Computing, pp. 97–104.
Roshandel, R., Medvidovic, N., Golubchik, L., 2007. A Bayesian model for predicting reliability of software systems at the architectural level. In: QoSA, pp. 108–126.
Tang, S., Peng, X., Lau, Y., Zhao, W., Jiang, Z., 2008. An adaptive software architecture model based on component-mismatches detection and elimination. In: Proceedings of the 32nd Annual IEEE International Computer Software and Applications Conference (COMPSAC '08), pp. 369–372.
Trivedi, K., 2001. Probability and Statistics with Reliability, Queueing, and Computer Science Applications, 2nd ed. Wiley-Interscience.
Van Der Aalst, W.M.P., Ter Hofstede, A.H.M., Kiepuszewski, B., Barros, A.P., 2003. Workflow patterns. Distributed and Parallel Databases 14 (1), 5–51.
Wattanapongskorna, N., Coit, D., 2007. Fault-tolerant embedded system design and optimization considering reliability estimation uncertainty. Reliability Engineering and System Safety 92, 395–407.
WOSP. Proceedings of the International Workshop on Software and Performance (1998–2007).
Yang, J., Huang, G., Zhu, W., Cui, X., Mei, H., 2009. Quality attribute tradeoff through adaptive architectures at runtime. Journal of Systems and Software 82 (2), 319–332.
Zeng, L., Benatallah, B., Ngu, A., Dumas, M., Kalagnanam, J., Chang, H., 2004. QoS-Aware Middleware for Web Services Composition. IEEE Transactions on Software Engineering 30, 311–327.

Pasqualina Potena received the degree in Computer Science from the University of L'Aquila (Italy) and the Ph.D. degree in Science from the University "G. D 'Annunzio" Chieti e Pescara (Italy). She is currently research fellow at the University of Bergamo. Her research interests include: non functional-aspects, architecture-based solutions for self-adapting/evolving software systems, optimization models.