Selecting Optimal Maintenance Plans based on Cost/Reliability Tradeoffs for Software Subject to Structural and Behavioral Changes

Vittorio Cortellessa
Dipartimento di Informatica
Università dell'Aquila
Via Vetoio, 1, Coppito (AQ), 67010 Italy
vittorio.cortellessa@univaq.it

Raffaela Mirandola, Pasqualina Potena
Politecnico di Milano
Dipartimento di Elettronica e Informazione
Piazza Leonardo da Vinci, 32, Milano, 20133 Italy
{*mirandola,potena*}*@elet.polimi.it*

***Abstract*—Software maintenance is assuming ever more a crucial role in the lifecycle due to the high variability of software requirements and environment. New development paradigms are being defined to support the numerous decisions that have to be taken after the software deployment. On the basis of the increasing request of software quality, non-functional attributes should enter in the decisional process to avoid changes that compromise the software quality. In this paper we define an optimization model that drives the choice of a maintenance plan (i.e. a set of maintenance actions to be taken) in correspondence of a certain change scenario. A change scenario is a set of new requirements that induce changes in the structural and behavioral architecture of the software system. The solution of such model, as shown in this paper on a mobile application, provides the set of actions that minimize the maintenance cost while guaranteeing a certain level of software reliability. We also show how this instrument can be used to perform a sensitivity analysis of maintenance plans vs cost/reliability tradeoff.**

***Keywords*-software cost; software reliability; optimization model.**

I. Introduction

Software maintenance is assuming ever more a crucial role in the lifecycle due to the high variability of software requirements and environment. New development paradigms are being defined to support the numerous decisions that have to be taken after the software deployment. On the basis of the increasing request of software quality, non-functional attributes should enter in the decisional process to avoid changes that could compromise the software quality.

In this paper we define an optimization model able to find the set of maintenance actions needed to tackle a certain change scenario. A change scenario is a set of new requirements that induce changes in the structural and behavioral architecture of the software system. In particular, we consider as possible changes, the introduction of new functionalities and the modification of the dynamics of existing functionalities. The set of maintenance actions provided by the solution of the optimization model guarantees the required level of reliability for the whole system and minimizes the maintenance cost.

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 ([8], [9], [1]). 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 able to fulfill the new requirements with the minimum cost and guaranteeing a given level of reliability.

Specifically, in our model, for each new requirement in a change scenario, we consider the different set of maintenance actions (called *maintenance plans*) able to guarantee this new requirement and we obtain a set of decisions that lead to the definition of a new architecture for the system that guarantees the required reliability and at the same time minimize the maintenance cost. To keep as simple as possible the modelling aspects of our model, we assume that the set of maintenance actions able to deal with each requirements are independent from each other.

The new architecture can be obtained changing both the structure and the behavior of a system. Specifically, to modify the system structure the model solution suggests how to replace existing software units with different available instances and if the adoption of new software units is necessary.

With respect to changes in the system behavior the model solution suggests how to modify the system scenarios (expressed, for example, as UML Sequence Diagrams) by removing or introducing interaction(s) between existing units and between these latter and new units.

The optimization model verifies the impact of the changes on the system reliability by predicting the reliability of the system obtained after the application of a plan chosen for each change requirement. Furthermore, the model allows studying the cost to maintain the system while varying the

reliability bound, i.e. the system reliability can be also a requirement to be changed.

Besides, in order to open the solution space to additional possibilities, our model allows the combination of maintenance plans with additional unit replacement actions. In other words, in the optimization model we leave to the solver the possibility to choose additional replacement actions that have not been embedded in any selected plan. If the maintainer does not want to replace existing units, the model will not suggest such additional replacement actions.

The proposed optimization model could be modified to be used as well in another phase of the software life-cycle by specializing it to capture typical aspects of the different phase. For example, in the architectural design phase the model parameters have to be estimated in a different way because in such a phase the system implementation is not yet available. Therefore, for example, the choice to either keep or replace an existing unit does not have to be supported, whereas in the maintenance phase the model can support such a choice.

Besides, our approach is not tied to any particular application domain (e.g. component and service) and it could be enhanced for supporting the adaptation of elementary services at runtime.

The paper is organized as follows: in Section II we provide the formulation of the optimization model that represents the core of our approach; in Section III we apply our approach to an example; Section IV introduces related work and discusses the novelty of our contribution; and conclusions are presented in Section V.

II. Optimization Model Formulation

In this section we introduce the formulation of our model aimed at finding the optimal set of maintenance actions needed to tackle required changes to the software architecture. "Optimal" is here intended as the actions that incur in a minimum cost while guaranteeing a certain level of reliability for the whole system.

Let S be the software architecture of a deployed system composed by n elementary software units. Since our model may support different lifecycle phases, we adopt a general definition of software unit: it is a self-contained deployable software module containing data and operations, which provides/requires services to/from other elementary elements. A unit instance is a specific implementation of a unit [1].

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

Let u_i be the i-th unit $(1 \leq i \leq n)$.

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

Let $pexec_k$ be the probability that the k-th system functionality will be invoked. It must hold: $pexec_k \geq 0$ for all $k = 1 \dots K$ and $\sum_{k=1}^{K} pexec_k = 1$. This information can be synthesized from the operational profile [25] ([2]). It is obvious that for a non-active functionality k we get $pexec_k = 0$.

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

In order to introduce a new functionality one of the SDs that are not active must be activated, namely it enters the set of SDs that contribute to the costs, reliability, etc. of the whole software architecture. In order to modify the dynamics of an existing functionality, certain interactions of the SD related to the functionality have to be added/removed.

In order to satisfy such new requirements some maintenance actions have to be performed. A *maintenance plan* is a set of actions that address a certain requirement. Obviously, for each requirement several maintenance plans may be available. Since they suggest different maintenance actions, they may differ for maintenance cost and/or for the system reliability achieved after the application of their actions. Let MP_r be the set of maintenance plans available for the r-th requirement in cs ([3]).

In order to keep as simple as possible the modelling aspects of our work, we have taken into account only software maintenance actions. The model can be enhanced, for example, by suggesting 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. In this paper we consider the following maintenance activities:

1) *Introducing new software units*: A maintenance action may suggest to embed into the system one or more

[1]Our framework can work for any semantics given to software units under the condition that the parameters are associated to the correct units. The only difference, of course, is in the techniques needed to estimate the model parameters, but this is out of the scope of this paper.

[2]Note that such assumption might be not realistic in all cases the operational profile may be not (fully) available. However, in such cases the domain knowledge and the information provided by the software architecture could be used for estimating it, as suggested for example in [27].

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

new software units ([4]). We call $NewS$ the set of new available units that can provide different functionalities, whereas $newun_h$ represents the h-th unit of $NewS$.

2) *Replacing existing unit instances with functionally equivalent ones*: A maintenance action may suggest to replace a software unit with one of additional instances available for it (e.g. a COTS component available in the market). We assume that the additional instances available for the unit u_i are functionally compliant with it, i.e. each instance provides at least all services provided by u_i and requires at most all services required by u_i (in Section II-A we discuss how to relax such an assumption by introducing integration/adaptation costs). Besides, the instances may differ from u_i for cost and reliability. We call $Avail_i$ the set of instances available for the u_i, while u_{ij} is the j-th instance of $Avail_i$.
3) *Modifying the interactions between software units in a certain functionality*: A maintenance action may suggest to modify the system dynamics by introducing/removing interactions between software units within a certain functionality.

It is obvious that any combination of such maintenance actions may have a considerable impact on the cost and reliability of the software architecture. Therefore, our optimization model aims at quantifying such impact to suggest the best maintenance plan that still minimizes the costs while satisfying the reliability constraints.

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

A. Model Parameters

- Parameters of (Existing and New) Software Units -

The parameters that we define for an existing software unit u_i $(1 \leq i \leq n)$ are:

- the number bp_{ki} of busy periods that it shows in the Sequence Diagram k;
- the cost c_{ij} of its j-th instance available $u_{ij} \in Avail_i$;
- the probability of failure on demand θ_{ij} of its j-th instance available $u_{ij} \in Avail_i$.

bp_{ki} is the number of invocations of u_i within a certain SD, and it can be easily estimated by parsing the diagram and counting the number of activations along its lifeline.

Although the estimate of c_{ij} is outside the scope of our work, we remark that it could be estimated as a function of the cost for acquiring a new version for the unit i (e.g. the purchase cost of a COTS component or the cost for upgrading a web service) and the one for integrating (adapting) the instance j into the system. About the latter, Yakimovich et al. have suggested guidelines for estimating the cost to adapt the architectural style of a COTS component to the one of a software architecture [32] [5]

In case of an in-house developed component, it can be estimated as a function of the per-day cost of a software developer for developing it. In this case it may also depend on the development process adopted (see, for example, [20], where a list of software cost estimation approaches is provided).

θ_{ij} represents the probability for u_{ij} to fail in one invocation [30]. As remarked in [11], a rough upper bound $1/N_{nf}$ of the probability θ_{ij} of a component can be obtained when observed that u_{ij} has been invoked for a N_{nf} number of times with no failures. However, several empirical methods to estimate failure rates of components can be found in [22].

If u_{ij} is an existing unit, then its parameters have to be estimated by considering that it is already into the system. Its purchase cost will be equal to 0, whereas its probability of failure on demand can be collected by monitoring the unit.

Finally, the parameters that we define for a new software unit $newun_h \in NewS$ are:

- the cost $\bar{c}_h$;
- the probability of failure on demand $\bar{\theta_h}$.

They can be estimated using the same techniques described for existing units.

- Parameters of Maintenance Plans -

The parameters that we define for a maintenance plan $p \in MP_r$ are:

- the new unit parameters (NS_p, BP_p);
- the variations in the busy periods of existing units VBP_p;
- the instances used for replacing existing units $VERS_p$;

NS_p is the (possibly empty) set of new units to be introduced due to some maintenance action ($NS_p \subseteq NewS$). BP_p is a $|NS_p| \times K$ matrix, where an element $BP_p(h,k)$ represents the number of busy periods of the h-th new unit in the k-th functionality (as represented in the related SD).

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

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

[5]Note that the introduction of integration/adaptation costs, although increasing the model complexity, would relax our assumption that the instances available for a software are functionally equivalent to each other. Besides, it could be exploited for supporting corrective maintenance activities (e.g. bugfixing or exception handling).

$VERS_p$ is a $n \times max_i|Avail_i|$ matrix, where an element $VERS_p(i,j)$ (also called $[v_{ij}]_p$ here below for sake of readability) is equal to 1 if the maintenance plan p suggests to update the existing unit i with its j-th available instance, and it is equal to 0 otherwise [6].

B. Model Variables and Elementary Constraints

We introduce the following variables to select a maintenance plan for each requirement of a change scenario.

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

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

$$\sum_{p \in MP_r} y_{rp} = 1, \qquad \forall r = 1 \dots m \tag{1}$$

We introduce the following variables to select an instance available for the i-th existing software unit.

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

For each existing unit i, exactly one instance must be selected. The following equation represents this constraint:

$$\sum_{j=1}^{|Avail_i|} x_{ij} = 1, \quad \forall i = 1 \dots n \tag{2}$$

If there are no available instances, we assume that $Avail_i$ includes the element i itself. [7]

If a plan is part of the model solution then all instances that it suggests for the existing units have to belong to the solution. The following equation represents this constraint:

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

We introduce the following variables to select the new software units to be introduced.

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

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

$$y_{rp} \leq z_h, \qquad \forall r = 1 \dots m, \forall p = 1 \dots |MP_r| \quad \forall h \in NS_p \tag{4}$$

C. Cost Objective Function (COF)

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

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

D. Reliability Constraint (REL)

We consider systems that may incur only in crash failures, that are failures that (immediately and irreversibly) compromise the behavior of the whole system. In order to relax such assumption a closed formulation the error propagation property [3] should be formulated as a closed equation.

A minimum threshold R is given on the reliability on demand [30] of the whole system.

Since we assume that for each functionality provided by the system we dispose of a Sequence Diagram, the reliability of the k-th functionality Rel_k can be expressed as follows:

$$Rel_k = \prod_{i=1}^{n} \Big(\sum_{j=1}^{|Avail_i|} x_{ij}(1-\theta_{ij})^{newbpex_{ki}} \Big) \cdot \cdot \prod_{h=1}^{|NewS|} (1-\bar{\theta_h} z_h)^{newbpn_{kh}} \tag{6}$$

where $newbpex_{ki}$ is the number of busy periods that the existing unit i shows in the Sequence Diagram k after the maintenance phase (i.e. after the application of the set of maintenance plans), and $newbpn_{kh}$ is the number of busy periods that the new unit h shows in the Sequence Diagram k after the maintenance phase.

The parameter $newbpex_{ki}$ can be expressed as follows:

$$newbpex_{ki} = bp_{ki} + \sum_{r=1}^{m} \sum_{p \in MP_r} VBP_p(i,k) \cdot y_{rp} \tag{7}$$

$newbpex_{ki}$ depends, in fact, on the added/canceled busy periods $VBP_p(i,k)$ that the chosen maintenance plans for the change requirements suggest for the unit i within the k-th functionality.

The parameter $newbpn_{kh}$ can be expressed as follows:

$$newbpn_{kh} = \sum_{r=1}^{m} \sum_{p \in MP_r} BP_p(h,k) \cdot y_{rp} \tag{8}$$

$newbpn_{kh}$ depends, in fact, on the number of busy periods $BP_p(h,k)$ that the chosen maintenance plans for the change

[6] It obviously holds: $\forall j > |Avail_i|\ [v_{ij}]_p = 0$

[7] Note that this equation holds if all existing units are still used after the maintenance phase, whereas equation (2) should be appropriately modified if this assumption is relaxed.

requirements suggest for the h-th new unit within the k-th functionality.

In a worst case setting all functionalities provided by the system should satisfy the reliability constraint, therefore the latter can be formulated as follows:

$$\min_{k=1...|K|}(Rel_k) \geq R \tag{9}$$

As opposite, in an average case setting the reliability constraint can be formulated as follows:

$$\sum_{k \in K} pexec_k \cdot Rel_k \geq R \tag{10}$$

In [12] we have summarized the model formulation and the main assumptions underlying the model.

III. A RUNNING EXAMPLE: SMARTPHONE MOBILE APPLICATION

In this section we describe an example that we use for showing the practical usage of our optimization model.

In the service-oriented domain, we have been inspired by the example used in [5]. Shortly, a smartphone user can require the latest news from a service provider (here called *Multimedia Service*). The news includes 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.

Figure 1 illustrates the software architecture of the system. It is a client/server system, where the *Client* is connected via a wireless network to the *Multimedia Service*. The latter uses other services for sending news to the client. In particular, the service *Trancoding* that adapts the video content for the smartphone format, the service *Compression* that adapts the news content to the wireless link, the service *Translation* that adapts the text for 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 Sequence Diagrams are represented in Figures 2, 3 and 4 ([8]). Figure 2 describes the system behavior while the *Multimedia Service* provides news in textual format, Figure 3 describes the system behavior while the *Multimedia Service* provides news with both textual and video content, whereas Figure 4 describes the system behavior while the *Multimedia Service* generates a geographical map with the user location.

In Table I we illustrate the change scenario that we have considered for this example. It is composed by three new requirements to be satisfied. In Table II, for each

[8]An extensive experimentation on such example can be found in [12].

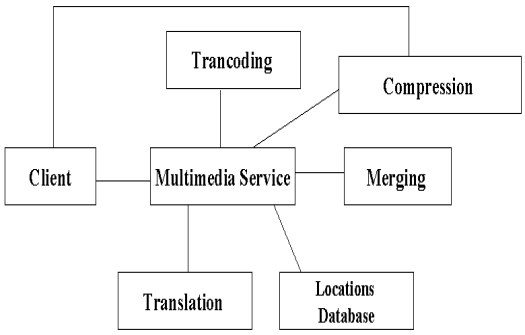

Figure 1. Software Architecture of the Smartphone application example.

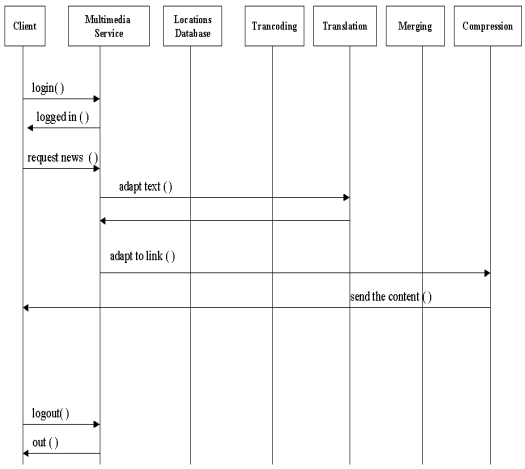

Figure 2. Requiring news in textual format.

requirement, we provide a set of maintenance plans that can be adopted to satisfy the requirement.

We have associated the IDs to the services as follows: u_1 to *Client*, u_2 to *Multimedia Service*, u_3 to *Locations Database*, u_4 to *Trancoding*, u_5 to *Translation*, u_6 to *Merging* and u_7 to *Compression*.

We have parameterized the optimization model with the values reported in [12].

In Figure 5 we report the results obtained from solving the optimization model for multiple values of the probability of failure on demand of new units $newun_1$ and $newun_4$ and of the reliability bound R. We have varied the reliability bound R from 0.92 to 0.99 in four steps. All bars in Figure 5 having the same color refer to the same value of R. We have varied the probability of failure on demand of $newun_1$ from 0.0003 to 0.005 and the one of $newun_4$ from 0.00002 to 0.005. Each group of four bars refers to the same pair of values assigned to these probabilities of failure.

The length of each bar, along the y-axis, represents the optimal value of the objective function of our model, that is the minimum cost of maintenance for the considered change scenario. For instance, if we consider the pair [$\bar{\theta_1}$ = 0.0003, $\bar{\theta_4}$ = 0.005] on the x-axis, then the minimum cost to maintain

Requirement ID	Requirement Specification
req_1	*The compression should be performed also in a different way*
req_2	*The video should be provided also in AVI format*
req_3	*The map should provide additional information such as a list of the closer hotels or restaurants*

Table I
CHANGE SCENARIO.

Requirement ID	Maintenance Plan ID	Maintenance Plan Description
req_1	mp_{11}	*Adding a new service $newun_1$ that interacts with the service Compression (i.e. u_7)*
	mp_{12}	*Adding a new service $newun_4$ that interacts both with Compression and Client AND Replacing Client (i.e. u_1) with its second instance available*
req_2	mp_{21}	*Replacing the service Trancoding (i.e. u_4) with its second instance available*
	mp_{22}	*Adding a new service $newun_3$*
	mp_{23}	*Adding a new service $newun_4$*
req_3	mp_{31}	*Adding a new service $newun_2$*
	mp_{32}	*Adding a new service $newun_1$ AND Replacing Locations Database (i.e. u_3) with its first instance available*
	mp_{33}	*Replacing Multimedia Service (i.e. u_2) with its second instance available*

Table II
MAINTENANCE PLANS AVAILABLE FOR SATISFYING THE REQUIREMENTS OF THE CHANGE SCENARIO.

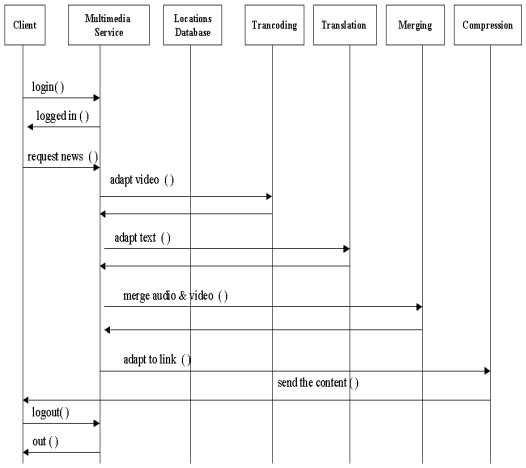

Figure 3. Requiring news with text and video.

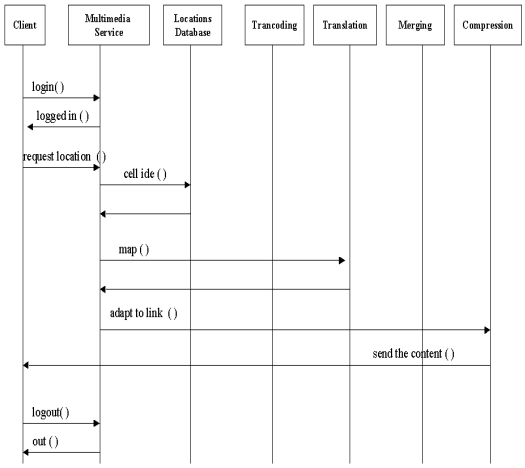

Figure 4. Requiring a map of location.

the system is 46 if we intend to guarantee a minimum system reliability equal to $R = 0.92$. In the same group of bars, the optimal solution cost obviously increases (up to 59) while raising the reliability threshold (up to $R = 0.99$).

By looking at the detail of the solution, for example we observe that for the entry ($[\bar{\theta_1} = 0.0003, \bar{\theta_4} = 0.005], R = 0.92$) the solution tuple is: [u_{11}, u_{21}, u_{31}, u_{42}, u_{51}, u_{61}, u_{71}] [$newun_1$] [$mp_{11}, mp_{21}, mp_{32}$]. This means that, in order to achieve the optimal cost of maintenance the following plans have to be adopted: the maintenance plan mp_{11} for the first requirement, the maintenance plan mp_{21} for the second requirement, and the maintenance plan mp_{32} for the third requirement. In addition, all the replacements of existing units (i.e. u_{ij}) are specified, as well as the adoption of the new unit $newun_1$ is claimed.

As expected, for a given value of the probability of failure on demand of the new units $newun_1$ and $newun_4$ the total cost of maintenance always does not increase while raising the reliability threshold R.

On the other hand, for a given value of the probability of failure on demand of $newun_1$ (i.e. 0.0003 or 0.005), the cost does not increase while decreasing the probability of failure of $newun_4$. In particular such cost sensibly decreases for the case $R = 0.99$.

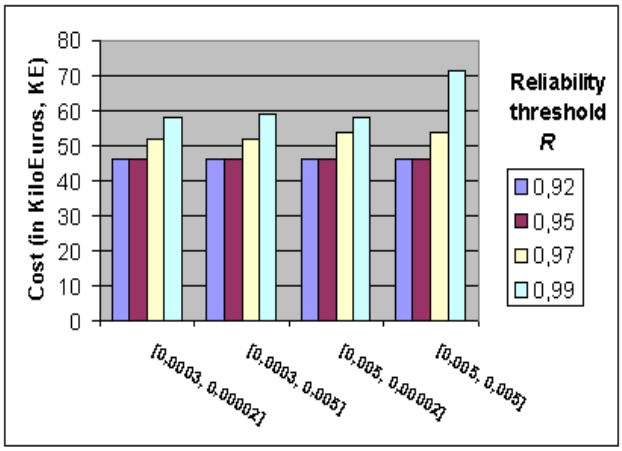

Figure 5. Objective function values for different cases.

In [12] we report the detailed results of all experiments that we have conducted. They highlight, in general, that the maintenance cost increases when higher reliability thresholds have to be guaranteed. For example, once fixed the probability of failure of $newun_1$ and $newun_4$ to 0.0003 and 0.005, respectively, the model provides two different solutions for $R = 0.95$ and $R = 0.97$. Such solutions differ for the instance selected for u_7. In fact, in order to satisfy the reliability constraint for $R = 0.97$ a more reliable version, which is also more expensive, is suggested for it.

Besides, once fixed a reliability threshold, the maintenance actions could be not different while varying the probability of failure of a new unit. For example, once fixed the probability of failure of $newun_4$ to 0.00002 and $R = 0.99$, the model suggests exactly the same solution when the probability of failure of $newun_1$ is equal to 0.0003 and when it is equal to 0.005. In both cases it suggests to add to the system the new unit $newun_4$.

On the other hand, while varying the probability of failure of a new unit, once fixed the reliability threshold, different actions could be suggested. For example, once fixed the probability of failure of $newun_1$ to 0.0003 and $R = 0.99$, if $newun_4$ is equal to 0.005 the model suggests to include $newun_1$, whereas if the probability of failure of $newun_4$ is equal to 0.00002 the model suggests to include $newun_4$. The introduction of either $newun_1$ or $newun_4$, 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 units. The solution given for the probability of failure of $newun_4$ equal to 0.00002, which is also less expensive than the other one, is not a good solution if $newun_4$ is equal to 0.005 because the reliability constraint would be not satisfied. In fact, the reliability of the system would be equal to 0.9804258. In [12] we show how the static and dynamic structure of the system changes after the application of the maintenance actions suggested by the two solutions.

Changing the maintenance plans for a requirement

The model may obviously suggest different actions while changing the maintenance plans for a change requirement. For example, once fixed the probabilities of failure to 0.0003 and 0.005, respectively, of $newun_1$ and $newun_4$, if $R = 0.97$ and we assume that the first maintenance plan available for the first change requirement (i.e. mp_{11}) is *Replacing the service Compression with its second instance available* instead of *Adding a new service* $newun_1$ *that interacts with the service Compression*, we get the following solution: $[u_{11}, u_{21}, u_{31}, u_{42}, u_{51}, u_{61}, u_{72}]$ $[newun_2]$, $[mp_{11}, mp_{21}, mp_{31}]$. The cost is equal to 51, while the system reliability is equal to 0.9762613. Figures 6 and 7, respectively, show how the static and dynamic structure of the system change after the application of the maintenance actions suggested by the solution ([9]). In the figures we have marked as bold the modifications brought after the application of the plans and we have put a cross on the interactions (i.e. messages in the Sequence Diagram) that are removed.

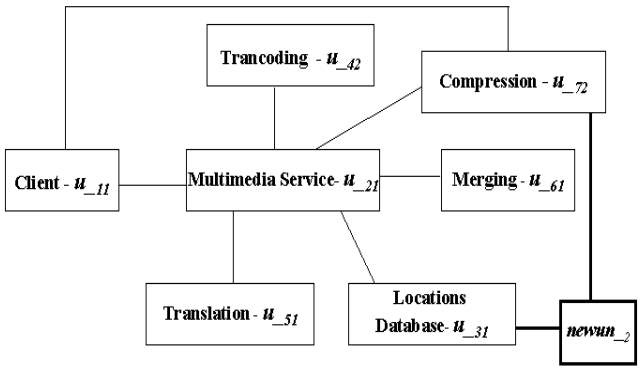

Figure 6. Resulting static structure of the system after the application of mp_{11}, mp_{21} and mp_{31}.

If we raise R to 0.99 the model suggests the solution: $[u_{13}, u_{22}, u_{31}, u_{42}, u_{53}, u_{61}, u_{72}]$ $[mp_{11}, mp_{21}, mp_{33}]$. The cost increases from 51 to 59, but the system reliability achieves the value of $R = 0.9966600$. In this case the model does not suggest to add new units, but instead to maintain the system by only replacing the initial units, as opposite to the previous solution.

This highlights that sometimes, to maintain a system, it is convenient to embed new units rather than replacing the existing units with available instances.

Removing a requirement from a change scenario

Starting from the initial values of the model parameters described in [12], let us assume that the change scenario in Table I does not include the first requirement req_1.

[9]Note that Figure 7 shows only the third Sequence Diagram because the other functionalities do not change.

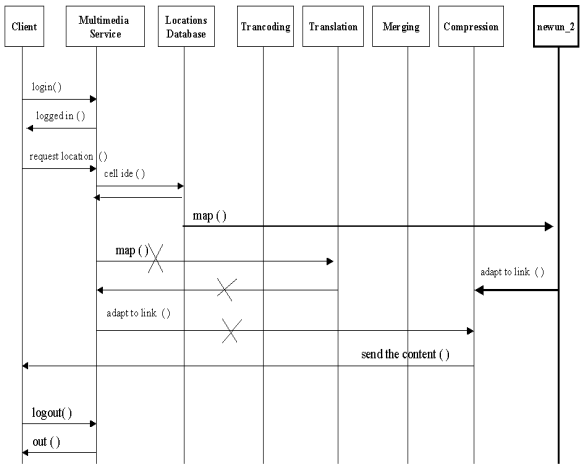

Figure 7. Resulting Sequence Diagram of Figure 4 after applying mp_{11}, mp_{21} and mp_{31}.

In this case, for $R = 0.99$ we get the following solution: [u_{11}, u_{22}, u_{31}, u_{42}, u_{51}, u_{61}, u_{71}] [mp_{21}, mp_{33}]. The cost is equal to 46, while the system reliability is equal to 0.9910393.

Moving from such a solution, if we apply the first plan mp_{11} for the first requirement then the system reliability decreases from 0.9910393 to 0.9877697, whereas the cost increases from 46 to 53. If, instead, we apply the second plan mp_{12} for the first requirement then the system reliability decreases from 0.9910393 to 0.9779168, while the cost increases from 46 to 55.

This highlights that it may be necessary to not fully maintain the system (i.e. to fully implement a change scenario) to keep the cost under a certain threshold.

In [12] we report a more detailed description of this sensitivity analysis, and we show other aspects that our model is able to capture and quantify. In particular, we show how it can support the choice to either keeping or replacing an unit and how it helps to combine (and, in general, to reason about) the decisions to be taken for each change requirement. Besides, we highlight how the model helps to analyze the maintenance cost and the system reliability while changing the interactions between units.

IV. Related work

As already outlined the topic of trade-off analysis between quality attributes and cost plays a key role during the whole software development. For example, several approaches have been introduced to explicitly analyze the impact of architectural decisions on system quality. For example, in [16] a method based on the definition of a multi-objective optimization model is proposed to allow the best architectural decisions maximizing the satisfaction of the quality attributes of the system under some constraints, such as related to budget limitations. Different kind of methods mainly based on the use of qualitative approaches have been proposed (see for example, [7], [21]), among which the well-known Architecture Tradeoff Analysis Method (ATAM) [21].

Another area quite close to our research concerns the software project planning where project managers construct their plans in a way that would be able to deal also with uncertainty [4], [6], [18], [17], [28], [33]. For example, in [6] the authors investigate the use of search-based software engineering techniques to solve the problem of resource allocations for different project planning solutions. Another interesting study can be found in [17], where search-based techniques, and specifically a multi-objective genetic algorithm, have been introduced to devise project planning solutions balancing two key objectives such as robustness and early completion time.

Hereafter, we focus on approaches specifically designed to support the maintenance of a system. To this end different maintenance actions are considered, such as the replacement of existing services or components through a selection process based on simple optimization models [2], [11], [13] or on multi-objective optimization models returning a Pareto solution maximizing a set of objectives (e.g. functionality,usability) under some constraints [10], [19], [26], [31].

Different kind of maintenance actions based on reconfigurations such as replacing, adding and/or deleting components in an architecture have been discussed in [14], [15] where methods verifying the soundness of a reconfiguration with respect to the system architecture are proposed.

Other challenges related to the maintenance phase are represented by the construction of the set of architectural alternatives for each change requirement (initial solution based on the use of meta-heuristic techniques can be found in [23]), and the detection and resolution of mismatches between requirements and existing elementary services/components ([24]) or between different services/components [29].

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

- This is the first paper (to the best of our knowledge) supporting the transformation of the system architecture (including both static and dynamic models) using an optimization model that suggests how to implement some of the required changes (i.e. introduction of new functional requirements or changing of already implemented system functionalities) while minimizing the cost and assuring a certain system reliability.
- The proposed approach is general and does not rely on specific architectural style, development process or application domain (e.g. component and service). It could be specialized, with respect to the application domain chosen. For example, it could be enhanced to support the adaptation of elementary services at runtime.
- In this paper we have instantiated the optimization

model for the maintenance phase, but it could be adopted in another phase of the software life-cycle. This adoption may require to specialize the model in order to capture typical aspects of the new phase. For example, in the architectural design phase the structure of model would not have to be changed, but it would be sufficient to estimate the model parameters in a different way.

- The proposed optimization model is independent from the methodology adopted to represent the scenarios and from the strategy used to generate the maintenance plans for each change requirement, we only need the number of busy periods of an elementary element.
- The defined model can facilitate the work of a software engineer (i.e. maintainer). In fact, in our model it is not necessary to insert as input value architectures satisfying all changes required, but possible architectures for each change requirement. When the computation time becomes too big, e.g. while increasing the number of maintenance plans, the adoption of metaheuristic techniques possibly combined with the introductions of dependencies between maintenance plans of different change requirements could allow to reduce the research space.

V. Conclusions

In this paper we have defined an optimization model that supports the choice of the best set of maintenance actions that address certain additional system requirements. The solution of such model, as shown in this paper on a smartphone mobile application, provides the set of actions that minimize the maintenance cost while guaranteeing a certain level of software reliability. The usage of this instrument to perform a sensitivity analysis of maintenance plans vs cost/reliability tradeoff is also illustrated through the example.

We intend to apply our approach on realistic examples in order to validate it and to study its scalability. To this end we intend to investigate the use of meta-heuristic techniques (e.g. the tabu-search algorithm) to improve the overall model complexity and scalability.

Other interesting research directions we intend to investigate concern the introduction/evaluation of risk factors associated to change requirements, the evaluation of dependencies among different requirements, the extension of our optimization model by introducing other quality constraints (e.g. security or performance constraints) using multi-optimization models, and its application to support the adaptation of software units at runtime.

Finally, we are designing a tool that automatically generates the optimization model starting from system models (such as UML diagrams) annotated with the appropriate parameters.

Acknowledgment

This research has been partially funded by the European Commission, Programme IDEAS-ERC, Project 227077-SMScom, the project Q-ImPrESS (215013) funded under the European Unions Seventh Framework Programme (FP7) and the project PACO (Performability-Aware Computing: Logics, Models, and Languages) funded by MIUR.

References

[1] Wosp : Proceedings of the international workshop on software and performance, 1998-2007.

[2] M. Abdallah, R. Guenther, and E. Armin. Cots Selection: Past, Present, and Future. In *ECBS '07: Proceedings of the 14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems*, pages 103–114, 2007.

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

[4] E. Alba and J. F. Chicano. Software project management with GAs. *Inf. Sci.*, 177(11):2380–2401, 2007.

[5] M. Alrifai and T. Risse. Combining global optimization with local selection for efficient QoS-aware service composition. In *WWW*, pages 881–890, 2009.

[6] G. Antoniol, M. D. Penta, and M. Harman. Search-Based Techniques Applied to Optimization of Project Planning for a Massive Maintenance Project. In *ICSM*, pages 240–249. IEEE Computer Society, 2005.

[7] M. Babar, L. Zhu, and D. Jeffery. A Framework for Classifying and Comparing Software Architecture Evaluation Methods. In *Proc. of Australian Software Engineering Conference*, pages 309–319, 2004.

[8] S. Balsamo, A. D. Marco, P. Inverardi, and M. Simeoni. Model-Based Performance Prediction in Software Development: A Survey. *IEEE Trans. Software Eng.*, 30(5):295–310, 2004.

[9] S. Becker, L. Grunske, R. Mirandola, and S. Overhage. Performance Prediction of Component-Based Systems - A Survey from an Engineering Perspective. In *Architecting Systems with Trustworthy Components*, pages 169–192, 2004.

[10] E. Bondarev, M. Chaudron, and P. de With. A Process for Resolving Performance Trade-Offs in Component-Based Architectures. In *CBSE*, pages 254–269, 2006.

[11] V. Cortellessa, F. Marinelli, and P. Potena. Automated Selection of Software Components Based on Cost/Reliability Tradeoff. In *EWSA*, pages 66–81, 2006.

[12] V. Cortellessa, R. Mirandola, and P. Potena. Selecting Optimal Maintenance Plans based on Cost/Reliability Tradeoffs for Software Subject to Structural and Behavioral Changes: Model Summary and Sensitivity Analysis. Technical report, Dip. Informatica, Università de L'Aquila, [Online]. Available: http://www.di.univaq.it/cortelle/docs/maintenance-techrep.pdf.

[13] V. Cortellessa and P. Potena. How Can Optimization Models Support the Maintenance of Component-Based Software? *Search Based Software Engineering, International Symposium on*, pages 97–100, 2009.

[14] P. David, M. Léger, H. Grall, T. Ledoux, and T. Coupaye. A Multi-stage Approach for Reliable Dynamic Reconfigurations of Component-Based Systems, booktitle = DAIS, year = 2008, pages = 106-111.

[15] V. Grassi, R. Mirandola, and A. Sabetta. A model-driven approach to performability analysis of dynamically reconfigurable component-based systems. In *WOSP*, pages 103–114, 2007.

[16] L. Grunske. Identifying "good" architectural design alternatives with multi-objective optimization strategies. In *ICSE*, pages 849–852, 2006.

[17] S. Gueorguiev, M. Harman, and G. Antoniol. Software project planning for robustness and completion time in the presence of uncertainty using multi objective search based software engineering. In *GECCO*, pages 1673–1680. ACM, 2009.

[18] M. Harman. The Current State and Future of Search Based Software Engineering. In *FOSE*, pages 342–357, 2007.

[19] M. Harman and L. Tratt. Pareto optimal search based refactoring at the design level. In *Proc. of the 9th annual conference on Genetic and evolutionary computation (GECCO 2007)*, 2007.

[20] M. Jorgensen and M. Shepperd. A Systematic Review of Software Development Cost Estimation Studies. *IEEE Trans. Softw. Eng.*, 33(1):33–53, 2007.

[21] R. Kazman, M. Klein, M. Barbacci, T. Longstaff, H. Lipson, and S. Carrière. The Architecture Tradeoff Analysis Method. In *ICECCS*, pages 68–78, 1998.

[22] J. Li, F. O. Bjoernson, R. Conradi, and V. Kampenes. An Empirical Study of Variations in COTS-based Software Development Processes in Norwegian IT Industry. *Software Metrics, IEEE International Symposium on*, pages 72–83, 2004.

[23] A. Martens and H. Koziolek. Performance-oriented Design Space Exploration. In *Proceedings of the 13th Int. Workshop on Component Oriented Programming (WCOP08)*, 2008.

[24] A. Mohamed, G. Ruhe, and A. Eberlein. Sensitivity analysis in the process of COTS mismatch-handling. *Requir. Eng.*, 13(2):147–165, 2008.

[25] J. Musa. Operational profiles in software-reliability engineering. *Software, IEEE*, 10(2):14–32, Mar 1993.

[26] T. Neubauer and C. Stummer. Interactive Decision Support for Multiobjective COTS Selection. pages 283b–283b, Jan. 2007.

[27] R. Roshandel, N. Medvidovic, and L. Golubchik. A Bayesian Model for Predicting Reliability of Software Systems at the Architectural Level. In *QoSA*, pages 108–126, 2007.

[28] M. O. Saliu and G. Ruhe. Bi-objective release planning for evolving software systems. In *ESEC/SIGSOFT FSE*, pages 105–114. ACM, 2007.

[29] S. Tang, X. Peng, Y. Lau, W. Zhao, and Z. Jiang. An Adaptive Software Architecture Model Based on Component-Mismatches Detection and Elimination. In *COMPSAC '08: Proceedings of the 2008 32nd Annual IEEE International Computer Software and Applications Conference*, pages 369–372, 2008.

[30] K. Trivedi. *Probability and Statistics with Reliability, Queueing, and Computer Science Applications, 2nd Edition*. Wiley-Interscience, 2001.

[31] A. Vescan. Pareto Dominance-Based Approach for the Component Selection Problem. *Computer Modeling and Simulation, UKSIM European Symposium on*, pages 58–63, 2008.

[32] D. Yakimovich, J. Bieman, and V. Basili. Software architecture classification for estimating the cost of COTS integration. In *ICSE '99: Proceedings of the 21st international conference on Software engineering*, pages 296–302, 1999.

[33] Y. Zhang, M. Harman, and S. A. Mansouri. The multi-objective next release problem. In *GECCO*, pages 1129–1137. ACM, 2007.