UNIVERSITÀ DEGLI STUDI DI BERGAMO
DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE
E METODI MATEMATICI°

QUADERNI DEL DIPARTIMENTO

Department of Information Technology and Mathematical Methods

Working Paper

Series "*Information Technology*"

n. 3/IT – 2011

An Optimization Process for Adaptation Space Exploration of Service-oriented Applications: the Stock Trading System case study

by

R. Mirandola, P. Potena, P. Scandurra

° Viale Marconi. 5, I – 24044 Dalmine (BG), ITALY, Tel. +39-035-2052339; Fax. +39-035-562779

[§] L'accesso alle *Series* è approvato dal Comitato di Redazione. I *Working Papers* della Collana dei Quaderni del Dipartimento di Ingegneria dell'Informazione e Metodi Matematici costituiscono un servizio atto a fornire la tempestiva divulgazione dei risultati dell'attività di ricerca, siano essi in forma provvisoria o definitiva.

An Optimization Process for Adaptation Space Exploration of Service-oriented Applications: the Stock Trading System case study

Raffaela Mirandola
Politecnico di Milano
DEI, Milano, Italy
mirandola@elet.polimi.it

Pasqualina Potena
Univ. degli Studi di Bergamo
DIIMM, Dalmine (BG), Italy
pasqualina.potena@unibg.it

Patrizia Scandurra
Univ. degli Studi di Bergamo
DIIMM, Dalmine (BG), Italy
patrizia.scandurra@unibg.it

***Abstract*—We propose an automatic optimization process for *adaptation space exploration* of service-oriented applications based on trade-offs between functional and extra-functional requirements. The optimization method combines both *meta-heuristic search techniques* and *functional/extra-functional patterns* (i.e., architectural design patterns and tactics). Moreover, the proposed methodology relies also on the standard *Service-oriented Component Architecture*(SCA) for heterogeneous service assembly and related tools/running infrastructures in order to process architectural models (of the application to adapt) that are directly tight to the real assembled components implementations and their deployment.**

As a proof-of-concepts, this report provides an example of instantiation of the proposed process together with an experimentation on a stock trading application.

***Keywords*-Service-oriented applications; software adaptation and evolution; functional/extra-functional requirements; meta-heuristic search; architectural patterns and tactics**

I. Introduction

Service-oriented applications may require adaptation both at re-design time (evolution) and at run time (self-adaptation) to changing user needs, system intrusions or faults, changing operational environment, and resource variability. The adaptation decisions, required by the user or triggered whenever unsatisfactory behaviors and/or values are reported by monitoring modules, should involve the evaluation of new alternatives to the current design (e.g., by changing the selection of components, the configuration of components, the sizing, etc.) in order to achieve the right trade off among the functional requirements, software qualities (such as performance and reliability) and the adaptation cost itself. However, the generation and evaluation of design alternatives is often time-consuming, can be error-prone and lead to suboptimal designs, specially if carried out manually by system maintainers.

We propose an optimization process for *adaptation space exploration* of service-oriented applications based on trade-offs between functional and extra-functional requirements. The optimization method combines *meta-heuristic search techniques* [1] with *architectural design patterns* and *tactics* as concrete examples of functional/extra-functional adaptation patterns. Moreover, the proposed methodology relies also on the standard *Service-oriented Component Architecture* (SCA) [2] for heterogeneous service assembly and related tools/running infrastructures in order to process architectural models (of the application to adapt) that are directly tight to the real assembled components implementations and their deployment.

As a proof-of-concepts, this report illustrates the adaptation of a sample application from the Stock Trading System (STS) case study in [3] by exploiting the *Tabu search* metaheuristic technique [1] and some functional/extra-functional adaptation patterns. Specifically, starting with a given initial architectural model of the considered system we first describe the application of the metaheuristic search technique to systematically explore the design space spanned by different adaptation actions (adaptation options). We also confirm the importance of having such an automated support by reporting some experimental results that clearly show that values obtained by our approach exceed the one produced with other methods such as the *lexicographic method* [4] and the manual work of expert/non expert system maintainers. Then, we show the combined use of some architectural patterns and tactics to improve the system availability and performance thus generating new input architecture candidates to re-iterate the metaheuristic search again till stop criteria are satisfied.

A. Model parameters of the STS case study

Fig. 1 shows the initial architecture of the considered application of the STS in terms of an SCA assembly. Briefly, an STS user, through the `OrderWebComponent` interacting with the `OrderDeliveryComponent`, can check the current price of stocks, placing buy or sell orders and reviewing traded stock volume. Moreover, he/she can know stock quote information through the `StockQuoteComponent`. STS interacts also with the external Stock Exchange system, which we do not model.

This section reports the values of the parameters for the STS example. In order to derive these system parameters, we took inspired from the WEB-based data retrieval system adopted as case study in [5]. In such a system clients are equipped with the local processing capability and local database. A WEB connection is provided to execute data search and data updates in a distributed system, through the network. We have assumed, though not shown in Figure 1, that our STS application also instantiates an agent on the server side of the WEB-based data retrieval system.

The estimation of non-functional/functional parameters is a well-know problem. It has been discussed, for example in [6], where the state of existing approaches is identified. Possible solutions are proposed manly based on the usage of historical data (if available) or derived from similar situations.

As done in [5], the estimation of the parameters entering our system has been partly based on the monitoring of an existing data retrieval system at University of L'Aquila, partly extracted from software artifacts of the same system. However, incomplete documentation forced to adopt extrapolation techniques for providing certain values. For example, the number of invocations has been obtained by analyzing partial scenario descriptions and validating the analysis results with monitored average number of interactions.

Table I shows the initial values of COTS parameters that we have considered. Similarly, Table II shows the initial parameters that we have considered for in-house instances.

We assume that several instances of an existing component may be available as COTS, all *equivalent* from the functional viewpoint. Basically, the instances differ each other for costs and non-functional properties such as reliability, response time or security level.

We have associated the IDs to the components as follows: C_1 to *Order Web Component*, C_2 to *Stock Quote Component* and C_3 to *Order Delivery Component*.

The second column of Table I lists, for each component, the set of instance alternatives available. For each alternative: the adaptation cost c_{ij} (in KiloEuros, KE) is given in the third column, the average delivery time d_{ij} (in days) is given in the fourth column, the average number of invocations of the component in the system s_i is given in the fifth column, finally the probability of failure on demand μ_{ij} is given in the sixth column.

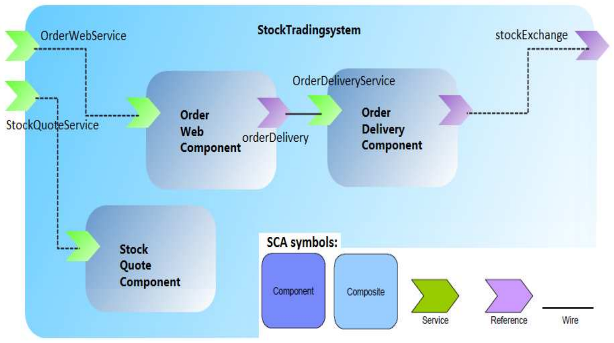

Figure 1. Stock Trading System

For each component in Table II: the estimated development time t_{i0} (in days) is given in the third column and the average time required to perform a test case τ_{i0} (in days) is given in the fourth column, the unitary development cost $\bar{c}_{i0}$ (in KE per day) is given in the fifth column, the average number of invocations s_i is given in the sixth column, and finally the component testability π_{i0} is given in the last column. The definition of testability π_i that we adopt is the one given in [7], that is the probability that a single execution of a software fails on a test case chosen from a certain input distribution. The input distribution represents the operational profile that we assume for the component, as obtained from the operational profile of the whole application [8].

In [5] suggestions for parameters estimation can be found.

Note that we have defined the parameters of components 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.

II. Metaheuristic search

Our approach takes advantage of the use of metaheuristic techniques. Their effectiveness and efficiency has been already demonstrated for supporting the service selection activity at run-time (e.g., see [9]). As remarked in [9], the global optimization, typically used by the approaches supporting such an activity driven by system quality, is definitely useful for small composition, but a significant performance penalty incurs for large-scale optimization problems, especially for runtime optimization.

Several metaheuristics [1] with different characteristics could be adopted depending on the problem: for example, considering the system reliability, a possible heuristic is to regard as increasing the whole reliability of the system when the reliability of the most used components increases. As remarked in [10], there exist design options for which we have no prior knowledge on how they affect the extra-functional property of a particular system. To this extent, undirected operations could be performed (e.g., random choices or exhaustive evaluation of all neighboring candidates).

A. Search Problem Formulation

Different adaptation actions are executed by modifying/-managing the SCA assembly (and related metadata) of the system. An SCA application can be adapted through the following actions: *a.* adding/removing a new component; *b.* changing a component implementation: the new component implementation maintains same component shape: same component type with the same services, same references, same properties; *c.* substitute a component: note that adding/removing services or references or properties to an existing component is to be intended as a component substituition; *d.* adding/removing reference-service wires and promotion wires (component interactions); *e.* changing

	Instance alternatives	Cost c_{ij}	Average delivery time d_{ij}	Average no. of invocations s_i	Prob. of fail. on demand μ_{ij}
C_1	C_{11}	1	4	180	0.0002
	C_{12}	2.5	4	180	0.0002
	C_{13}	2	4	180	0.0004
C_2	C_{21}	2	4	20	0.0002
	C_{22}	3	4	20	0.0002
	C_{23}	6	15	20	0.0004
C_3	C_{31}	10	4	60	0.0002
	C_{32}	14	10	60	0.0004
	C_{33}	10	10	60	0.0004

Table I
PARAMETERS OF INSTANCE AVAILABLE FOR EXISTING COMPONENTS

	Development Time t_{i0}	Testing Time τ_{i0}	Unitary development cost $\bar{c}_{i0}$	Average no. of invocations s_i	Testability π_{i0}
C_1	1	0.05	1	180	0.002
C_2	3	0.05	1	20	0.002
C_3	5	0.05	1	60	0.002

Table II
PARAMETERS FOR IN-HOUSE DEVELOPED INSTANCES

component'properties values; *f.* changing SCA domains (components re-deployment).

It is also possible changing the component interaction style in synchronous/asynchronous, stateful or not, unidirectional or bidirectional, and this fact is reflected in SCA by changing the shape, at interface level, of the components involved in the interaction and the wire type.

These changes are reflected at SCA level by changing the shape, at interface level, of the components involved in the interaction and the wire type (communication binding) used to interconnect the components. See [2] for more details. Moreover, with respect to changes in the system behavior (that is formally specified in terms of ASMs for functional analysis purposes), an adaptation action may imply also changes in the services interaction(s) and orchestration process. These are reflected also at ASM level, as refinement of the ASM transition rules specifying the components' services behavior and their orchestration.

For a specific system a range of options can be available for each adaptation action. For example, for the STS system, the SCA component C_1 can be replaced with one of its three instances (equivalent from the functional viewpoint) available on the market and the component C_2 may be removed by introducing into the system two new components.

The adaptation space AS, that is the search space of our optimization process (i.e., the set of all possible candidate solutions), is the Cartesian product of the option sets of all system-specific adaptation actions. Each candidate s can be expressed as a vector of chosen adaptations options, such as the vector $[C_{11}, C_{21}, C_{32}]$, where the element C_{ij} denotes either the COTS instance chosen for the i-th existing component of the STS system.

Obviously, constraints on the system-specific adaptation actions may be required (e.g., the STS component C_1 must be replaced by C_{11}). With these definitions, the problem is suited for metaeuristic techniques. In fact, this type of problem is NP-difficult because it can suffer of large elapsed time when the search space size increases (e.g., while adding new COTS instances).

B. Multi-objective Optimization and Pareto solutions.

Our optimization process implements the multi-objective optimization [11]. As claimed by the *weighted sum method*[1] the function objective $f(s)$ can be formulated as follows:

$$f(s) = \sum_{q=1}^{n} \alpha_q \cdot f_q(s)$$

where α_i's are real numbers[2] and $f_q(s)$ denotes the quality property of s for the quality criterion q to be minimized (e.g., adaptation cost or probability of failure).
Once fixed the values of the weights α_i, a metaheuristic method can be applied (see the second phase of the *Two-Phase Local Search* algorithm [12]) for finding the set of *Pareto solutions* (i.e., an approximation of the Optimal Pareto set).
In our context, we can state that a candidate SCA-ASM assembly is Pareto-optimal, if it is superior to any candidates evaluated so far in at least one quality criterion.

More formally: Let s be a candidate solution, let $C \subseteq AS$ be a set of candidate solutions evaluated so far, and let q be a quality criterion with a domain D_q, and an order $\leq_q$ on D_q

[1]It is the most common approach for multi-objective optimization.
[2]It holds: $\sum_{q=1}^{n} \alpha_q = 1$.

```
1.  $s \longleftarrow$ *GenerateInitialSolution()*
2.  *TabuList* $\longleftarrow \oslash$
    // s' memorizes the best solution of
    // the tabu search
3.  $s' \longleftarrow s$
4.  **while** termination conditions not met **do**
5.    *NeighboursOkSet* $\longleftarrow$ *ExploreNeighbourhood(s)*
6.    $s \longleftarrow$ *ChooseBestof(NeighboursOkSet)*
7.    *Update(TabuList)*
8.    **if** $(f(s') > f(s))$ **then**
9.      $s' \longleftarrow s$
      **end if**
    **end while**
```

Figure 2. Algorithm: Tabu Search.

so that $s_1 \leq_q s_2$ means that s_1 is better than or equal to s_2 with respect to quality criterion q. Then, candidate solution s is Pareto-optimal with respect to a set of evaluated candidate solutions C, iff

$$\forall s' \in C \exists q : f_q(s) \leq_q f_q(s')$$

If a candidate solution is not Pareto-optimal, then it is Pareto-dominated by at least one other candidate solution in C that is better or equal in all quality criteria. Analogously, a candidate is globally Pareto-optimal, if it is Pareto-optimal with respect to the set of all possible candidates AS.

C. Tabu Search

The Tabu Search (TS) is among the most cited and used metaheuristics for solving optimization models. It enhances the performance of a local search method by using memory structures describing the visited solutions. Once a solution is visited, it is marked as "taboo" so that the TS does not visit that possibility repeatedly. TS explicitly uses the history of the search, both to escape from local minima and to implement an explorative strategy.

The pseudo-code of the simple TS algorithm is shown in Figure 2. A description of its main steps follows.

Begin with a starting current solution The initial candidate s, representing a SCA-ASM assembly and fulfilling the existing/new functional and non-functional requirements is generated.[3].

Create new candidates The tabu search is based on a *short term* memory, which is implemented as a *tabu list*. This latter keeps track of the most recently visited solutions and forbids moves toward them.

At each iteration step, the neighborhood of the current solution[4] is restricted to the solutions that do not belong to the tabu list (i.e., definition of $NeighborsOkSet$ in Figure 2). Such a set of new candidates is obtained making changes to the current solution (these changes are also called *moves*) by applying user adaptation plans, service selection and service re-deployment.

Choose the best candidate The best candidate is then selected as the one minimizing the objective function (under possible constraints). This step is performed through the function $ChooseBestof(NeighboursOkSet)$ in Figure 2). The candidate becomes the basis for next candidates generation and the current best solution of all tabu search interactions. Additionally, such a solution is added to the tabu list and one of the solutions that were already in the tabu list is removed (usually in a FIFO order). The length of the tabu list is given as value of input to the tabu search[5].

Stopping criterion The process proceeds iteratively till stop criteria are satisfied by returning the best solution of all interactions. The algorithm can stop if the predefined number of iterations has elapsed in total. More sophisticated stop criteria could use convergence detection and stop when the global optimum is probably reached.

This simple TS could be specialized and enhanced depending on the problem, e.g., varying the tabu list length or leveraging on long-term memory (see [1] for details). Furthermore, heuristics operations could be used for improving its performance. For example, considering the system reliability, a possible heuristic is to regard as increasing the whole reliability of the system when the reliability of the most used components increases. As far as the performance domain knowledge, it could be exploit the fact that if the processing speed of a highly utilised resource increases, then the response time of a system will likely decrease (although there are exceptions) [6].

III. APPLICATION OF THE TABU SEARCH TO THE STS CASE STUDY

In this section we show an application of the tabu search (see Section II) to the STS case study.

This TS application is designed for replacing the STS components by buying or building components on the base of cost and non-functional factors (i.e., reliability and delivery time). The TS also provides the best amount of testing to be performed on each in-house developed component to fulfill the constraints while minimizing the adaptation costs. The TS solves the non-linear cost/quality optimization model [5] based on decision variables indicating the set

[3]Depending on several factors (e.g., search technique used) different strategies could be adopted [1], such as an algorithm combining heuristics, local search, user adaptation plans, service selection and re-deployment actions for finding a set of admissible (functional) solutions. In Section III we will provide an implementation of an algorithm for generating initial solutions.

[4]"A neighborhood structure is a function $N : S \rightarrow 2^S$ that assigns to every $s \in S$ a set of neighbors $N(s) \subseteq S$. $N(s)$ is called the neighborhood of s."[1]

[5]The tabu length can be varied during the search, leading to more robust algorithms[1].

[6]In [10] an example of application of such performance heuristics can be found.

of architectural components to buy and to build in order to minimize the software cost under the delivery time and reliability constraint (i.e., the system reliability and delivery time are required within a threshold R and T, respectively). Such a model belongs to the class of mixed integer nonlinear programming models can suffer of large elapsed time when its size increases (e.g., it grows exponentially in the number of components). We have implemented in *C* and optimized for fast execution the TS algorithm. The entire set of experiments, which we have performed, took practically no noticeable time (order of seconds) on standard computing equipment.

In this section we describe the main features and steps of the TS application. The TS is relies on: (i) the *Initial Solution Generation* algorithm for the generation of the starting solution of the first interaction of TS; and (ii) the *Testing Generation* algorithm to find the amount of testing for the in-house instances of a candidate s. Section III-A presents the *Initial Solution Generation* algorithm, whereas Section III-B details the *Testing Generation* algorithm.

Multi-objective function The function objective $f(s)$ is the weighted sum of the adaptation cost and the system probability of failure. Note that, to sum such objectives we apply to the system probability of failure on demand the logarithm function. In fact, since the probability of failure on demand is a number that falls within the range of $[0,1]$ its logarithm is a negative number.

Begin with a starting current solution The initial candidate s, which fulfills the reliability and delivery time constraints is the vector $[C_{ij}]$, made of three elements, where an element C_{ij} denotes either a COTS instance or an in-house instance. The in-house instance of the component i is named C_{i0}. Besides, the name of the instance is paired with the number of test to perform on the instance. C_1 indicates the `OrderWebComponent`, C_2 the `StockQuoteComponent` and C_3 the `OrderDeliveryComponent`. The resulting system reliability, the cost and delivery time of the solution are predicted using the reliability, cost and delivery time model used in [5]. The candidate s is generated by using the *TS Initial Solution Generation* method described in Section III-A.

Create new candidates The set of new candidates is generated by replacing, one at a time, an existing component with either one available on the market or an in-house instance. The amount of testing of the in-house developed instances is found by using the *Testing Generation* algorithm described in Section III-B.

Choose the best candidate The best candidate is selected as the one minimizing the objective function under reliability and delivery time constraints. Additionally, the pair (i,j) is stored into the tabu list, where i is the index of component changed and j represents the new solution chosen for the component i, and the oldest components in the tabu list are removed as the list becomes full following a FIFO order.

Stopping criterion The TS stops if the predefined number of iterations has elapsed in total. At each iteration, upon examining the neighborhood, if no feasible solution is found, then the initial solution of the next interaction is generated using the *TS Initial Solution Generation* method.

To optimize the search we also exploit the heuristic to regard as increasing the whole reliability of the system when the reliability of the most used components increases. To this purpose, we order the components using the *Quick Sort* algorithm [13] with growing probability of failure on demand. Furthermore the components that do not satisfy the delivery time constraint are removed from the search space[7] are removed from the search space.

A. TS Initial Solution Generation algorithm

Depending on several factors (e.g., search technique used) different strategies could be adopted [1] for finding a set of feasible solutions, such as an algorithm combining heuristics, local search, user adaptation plans, service selection and re-deployment actions.

For this step, we draw inspiration from the solution construction phase of the Greedy Randomized Adaptive Search Procedure (GRASP) [1]. We generate a list of feasible solutions that fulfill the quality constraints (i.e., the system reliability and delivery time are assured within the required thresholds). A solution is the vector $[C_{ij}]$ where an element C_{ij} denotes either a COTS instance or an in-house instance. First the COTS component C_{1j} is chosen by picking it uniformly at random from the set of COTS instances available for C_1, then the set of solutions is generated by replacing, one at a time, another existing component with one available on the market instance.
Such a search is optimized by the reliability heuristic described above. If no solutions are returned, the process is repeated by choosing another component $C_{1\bar{j}}$ using the reliability heuristic, and a solution is randomly generated by considering also the in-house instances.

B. Testing Generation algorithm

The *Testing Generation* (TG) algorithm estimates the amount of testing of the n_{house} in-house developed instances of the candidate s. In the following we discuss the main steps and featured of TG, which implements another tabu search algorithm.

Begin with a starting current solution The initial candidate t, that fulfills the quality constraints is the vector $[t_h]$, made of n_{house} elements, where an element t_h denotes the maximum amount of unit test $maxt_h$ that could be performed on the in-house component h.

[7]Since we assume that that manpower is available to independently develop in-house component instances, the delivery time of each COTS (in-house) component have to be within the required threshold T (see [5]).

t_h is estimated as a function of the threshold T required for the system delivery time[8].

Create new candidates At each iteration step, the neighborhood of the current solution t is generated by varying, one at a time, the testing of an in-house instance h on the range $[0, maxt_h]$.

Choose the best candidate The best candidate is selected as the one minimizing the objective function $f(s)$ under reliability and delivery time constraints. The candidate becomes the basis for next candidates generation and eventually the current best solution of all TG interactions. Additionally, the pair (h, t_h) is added to the tabu list, where h is the index of in-house changed (with respect to the initial candidate t of the current TG interaction) and t_h represents the amount of unit test for the in-house h, and the oldest solution that were already in the tabu list are removed whether the list is full using a FIFO discipline. The length of the tabu list is given as value of input to the tabu search.

Stopping criterion The TG iteratively stops if the predefined number of iterations has elapsed in total by returning the best solution of all interactions. At each interaction, upon examining the neighborhood, if no feasible solution is found, then the initial solution for the next interaction is generated as follows.

A starting solution $t = [t_h]$ is generated with an algorithm similar to the *Initial Solution Generation* algorithm described above. First t_1 is chosen by picking it uniformly at random on the range $[0, maxt_1]$, then the other testing amounts are generated by varying the one of an in-house instance h ($h \neq 1$) on the range $[0, maxt_h]$. Such a search is optimized by using the reliability heuristic to regard as increasing the reliability of a component when its amount of testing increases. Note that such an algorithm could be also used for generating the initial candidate for the first interaction of TG.

IV. Other methods

For comparison purposes, in our experimentation we considered two other methods to generate alternative adaptation solutions: the *lexicographic method* [4], and the judgment of a group of (human) maintainers formed by expert/non expert with respect to the system and execution environment.

We have implemented the *lexicographic method* as follows. First we have solved the optmization model minimizing the adaptation cost under reliability and delivery time constraint (i.e., the model presented in [5]), then we have formulated the optimization model that minimizes the probability of failure under the cost constraint expressed as $f_1(x) \leq f_1(x*) + \epsilon$, where ϵ is a positive tolerance (real number). Finally, we have found the set of Pareto optimal solutions by varying ϵ (i.e., we have applied the ϵ-constraint approach [4]). We have proceeded the process till the stop criteria was satisfied[9]. For the experimentation we have used the LINGO tool [14], which is a non-linear model solver, to produce the results.

The group of maintainers was made of expert/non-expert people. The choices of non-expert ones were random. On the opposite, expert persons were guided by their knowledge of the system and execution environment. Therefore, they were driven by heuristics (e.g., the reliability of the most used components can more likely increase the system reliability). Similarly to the lexicographic method, while making their decisions they have collected the Pareto solutions till the stop criteria was satisfied.

V. Adaptation space exploration of the STS case study

Below, we apply to the STS case study the adaptation strategies adopted by our methodology. Specifically, starting from an initial system configuration, first we describe the application of the tabu search metaheuristic technique and of the other two methods described in the previous sections, and then we show the use of some tactics and an architectural design pattern as examples of extra-functional and functional, respectively, adaptation patterns.

We have applied the approaches on three different configurations of the STS system (characterized by the parameters discussed in Section I-A). In order to keep our model as simple as possible, in all configurations we assume that only one in-house instance for each component can be developed. The number of COTS instances does not change across configurations, but each configuration is based on a different set of component parameters. The configurations differ also for the values of reliability R and delivery time T bounds.

The configuration parameters have been set for showing the behavior of three approaches while increasing the search space complexity. The configurations differ for the probability to find a pareto solution: the first configuration, characterized by a lower threshold R, has an higher probability with respect to the other ones, characterized by a higher threshold R and a set of selected components more complex to be analyzed.

System configurations*:* The first configuration has the threshold on the delivery time and reliability to $T = 7$ and $R = 0.5$, respectively. In addition, the costs of C_{11} and C_{21} is increased to 5 units (i.e., C_{11} and C_{21} = 5). The reliability threshold (that may be unrealistic) has been set to show the behavior of three approaches in the case of a not complex search space.

The second configuration has the threshold on the delivery time and reliability to $T = 15$ and $R = 0.8$, respectively. In addition, the costs of C_{11}, C_{21} and C_{22} is increased to 5 units (i.e., C_{11} = C_{21} = C_{22} = 5), and the probability of

[8]A budget constraint could be also used.

[9]We have used the satisfaction of reliability constraint to determine the end of the search and a predefined number of interactions.

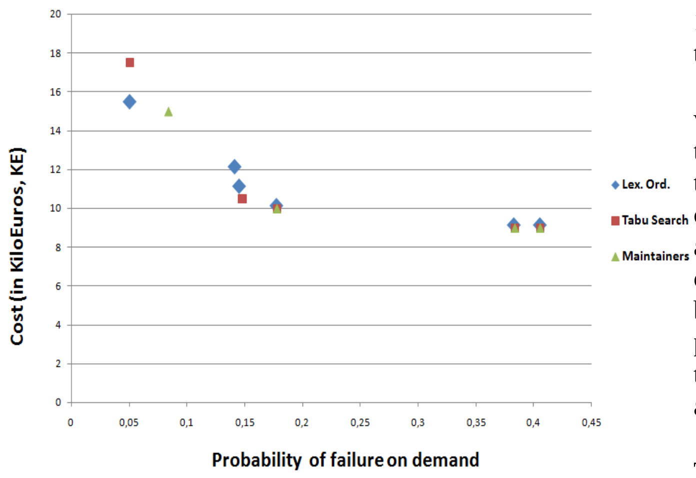

Figure 3. Comparison of the three approaches with respect to the first configuration

failure on demand of COTS instances available for C_1 is increased to 0.003 (i.e., $\forall j$ μ_{1j} = 0.003).

Finally, the third configuration has the threshold on the delivery time and reliability to $T = 15$ and $R = 0.8$, respectively. In addition, the costs of C_{11}, C_{21} and C_{22} is increased to 5 units (i.e., C_{11} = C_{21} = C_{22} = 5), and the probability of failure on demand of COTS instances available for all components is increased to 0.003 (i.e., $\forall i, j$ μ_{ij} = 0.003).

The experiments were run on a Windows workstation equipped with a Intel Centrino Processor 1.3 GHz CPU and a 512 MB RAM. The tabu search algorithm was compiled using lcc-win32 3.3. We imposed a number of interactions of 50, and the tabu list length limit of 45 to each experiment. Finally, the optimization model for the lexicographic method was solved using LINGO 11.0.

A. A comparison among the tabu search, the lexicographic method, and the work of system maintainers

In Figure 3 we report the approximate Pareto curves obtained from solving the optimization problem of the first system configuration using the three approaches. In this configuration a maximum threshold $T = 7$ has given on the delivery time of the whole system, and a minimum threshold $R = 0.5$ is given on the reliability of the whole system. The reliability threshold (that may be unrealistic) has been set to show the behavior of three approaches in the case of a not complex search space.

Each Pareto solution represents a configuration of components that minimizes both the system adaptation cost and its probability of failure on demand. For example, the tabu search results claim that if the probability of failure is equal to 0.405479 (represented on the x-axis), then the minimum cost to adapt the system is 9 KE. The tabu search results also show that the optimal solution cost increases (up to 17.5 KE) while raising the system probability of failure (up to 0.050671).

By looking at the details of the solution, for example, we observe that for the tabu search point (0.405479, 9 KE) the solution point is: [(C_{10},0), (C_{20},0), (C_{30},0)]. This means that, in order to achieve the optimal cost of adaptation all components have to be in-house built without suggesting an amount of testing. As done in [5], we express the possibility of reducing the probability that an in-house component fails by means of a certain amount of test cases. We define its probability of failure on demand under the assumption that the on-field users' operational profile is the same as the one adopted for testing [15].

In Table III we report the detailed results of Figure 3. The table is organized as follows: the first, second and third columns represent the tabu search, the lexicographic method and the maintainers judgment, respectively. In each entry (row, column) we represent the choice of components (i.e., a Pareto solution). The choice is represented as a vector, where each element can be either a COTS instance or an in-house instance. In the latter case, the name of the instance is paired with the number of successful (i.e., failure-free) test to perform on the instance[10]. The in-house instance of component i is named C_{i0}. Beside the vector of instance components, the resulting system probability of failure and the cost of the solution are reported in each entry. Furthermore, specific parameters (i.e., α_i, ϵ, and the tabu search execution time $Time$) of the approaches are also reported. The working time of the maintainers can not be quantified with a number because they have taken their decisions during different meetings. However, they used a short computation time (order of minutes) by making their decisions randomly or analyzing simple search space (e.g., the one of the first configuration). On the other hand, by leveraging on their personal expertise and experience they have sometimes found good solutions, but they have spent time for discussions (order of hour, e.g., one hour and half for reaching good solutions in the third system configuration).

The results highlight, in general, that the solutions of the three approaches do not show discrepancies: the probability of failure and the cost of their Pareto solutions are slightly different.

On the other hand, increasing the reliability threshold R, such as in second and third system configuration the discrepancies become more evident.

In Figure 4 we report the approximate Pareto curves obtained from solving the optimization problem the three approaches with respect to the second system configuration, where it is required $T = 15$ and $R = 0.8$. Similarly to Figure 3 and 4, in Figure 5 we report the approximate Pareto curves

[10]The total number of tests performed on the instance can be obtained as a function of its testability (see [5] for details).

Lexicographic	Tabu Search	Maintainers
$[(C_{10}, 1), (C_{20},1), (C_{30},1)]$ $FRsys = 0.404862$, $Cost = 9.150301$ KE	$[(C_{10},0), (C_{20},0), (C_{30},0)]$ $FRsys = 0.405479$, Cost=9 KE, Time=13.06 sec $\alpha = 1$	$[(C_{10},0), C_{22}, (C_{30},0)]$ $FRsys = 0.383686$, $Cost = 9$ KE
$[(C_{10},3), C_{22}, (C_{30},0)]$ $FRsys = 0.382359$, $Cost = 9.150301$ KE	$[(C_{10},0), C_{22}, (C_{30},0)]$ $FRsys = 0.383687$, Cost=9 KE Time=14.482 sec $\alpha = 0.6$	$[(C_{10},0), (C_{20},0), (C_{30},0)]$ $FRsys = 0.405479$, Cost=9 KE
$[C_{13}, C_{22}, (C_{30},3)]$ $FRsys = 0.177398$, $Cost = 10.1503$ KE, $\epsilon = 0.999$	$[C_{13}, C_{22}, (C_{30},0)]$ $FRsys = 0.177988$, Cost=10 KE, Time=11.938 sec $\alpha = 0.4$	$[C_{13}, C_{22}, (C_{30},0)]$ $FRsys = 0.177987$, Cost=10KE
$[C_{12}, C_{22}, (C_{30},13)]$ $FRsys = 0.145230$, Cost=11.1513 KE, $\epsilon = 2$	$[C_{12}, C_{22}, (C_{30},0)]$ $FRsys = 0.147856$, Cost=10.5 KE Time=11.998 sec $\alpha = 0.2$	$[C_{13}, C_{22}, C_{31}]$ $FRsys = 0.842391E - 01$, Cost=15 KE
$[C_{12}, C_{22}, (C_{30},33)]$ FRsys=0.141306, Cost=12.15331 KE, $\epsilon = 3$	$[C_{12}, C_{21}, C_{31}]$ $FRsys = 0.050671$, Cost=17.5KE Time=9.855 sec $\alpha = 0$	
$[C_{12}, C_{22}, C_{31}]$ $FRsys = 0.506711E - 01$, Cost=15.5KE, $\epsilon = 8$		

Table III
RESULTS FROM LEXICOGRAPHIC, TABU SEARCH AND MAINTAINERS FOR THE FIRST CONFIGURATION

Lexicographic	Tabu Search	Maintainers
$[(C_{10},1), (C_{20},1), (C_{30},1)]$ $FRsys = 0.404862$, $Cost = 9.150301$ KE	$[(C_{10}, 277), C_{21}, C_{31}]$ $FRsys = 0.199833$ $Cost = 29.87776$ Time= 11sec $\alpha = 1, 0.2, 0.01, 0$	$[(C_{10},0), (C_{20},0), (C_{30}, 0)]$ $FRsys = 0.405479$ $Cost = 9$ KE
$[(C_{10}, 3), (C_{20},0), (C_{30},0)]$ $FRsys = 0.404199$, $Cost = 9.150301$ KE		$[(C_{10},27), C_{22}, C_{31}]$ $FRsys = 0.300292$ $Cost = 17.35271$ KE
$[(C_{10}, 23), (C_{20},0), (C_{30},0)]$ $FRsys = 0.395788$, $Cost = 10.1523$ KE $\epsilon = 0.999$		$[(C_{10},276), C_{22}, C_{31}]$ $FRsys = 0.200163$ $Cost = 29.82766$ KE
$[(C_{10},3), C_{22}, (C_{30},0)]$ $FRsys = 0.382359$, $Cost = 11.1503$ KE $\epsilon = 2$		
$[(C_{10},23), (C_{20},0), C_{31}]$ $FRsys = 0.326879$, $Cost = 15.1523$ KE $\epsilon = 6$		
$[(C_{10}, 63), C_{21}, C_{31}]$ $FRsys = 0.283528$, $Cost = 19.15631$ KE $\epsilon = 10$		
$[(C_{10}, 280), C_{21}, C_{31}]$ $FRsys = 0.198841$, $Cost = 30.02806$ KE $\epsilon = 23$		

Table IV
RESULTS FROM LEXICOGRAPHIC, TABU SEARCH AND MAINTAINERS FOR THE SECOND CONFIGURATION

obtained from solving the optimization problem the three approaches with respect to the third system configuration, where it is required $T = 24$ and $R = 0.8$. In the figures we have circumscribed the feasible solutions. Similarly to Table III, in Table IV and V we report the detailed results of the experimentation for the second and third system configuration, respectively.

The tabu search, even when the search space became more complex, has returned feasible solutions in a short time: its execution time increased from few seconds (about eleven seconds) to few minutes (about one minute). On the opposite, the lexicographic method has taken more time while increasing the search space. Finally, the maintainers have also used a short computation time (usually not finding feasible solutions) by making their decisions randomly. On the other hand, by leveraging on their personal expertise and experience they have sometimes found good solutions, but they have spent time for discussions.

Discussion on the compared approaches: The comparison of the results has revealed that the reasoned choices of expert maintainers are convincingly better than the random ones of non expert persons, whereas the lexicographic

Lexicographic	Tabu Search	Maintainers
[(C_{10},1), (C_{20},1), (C_{30},1)] $FRsys = 0.404862$, $Cost = 9.150301$ KE	[(C_{10},460), (C_{20},267), (C_{30},380)] $FRsys = 0.199966$, $Cost = 64.46092$ KE Time= 65 sec $\alpha = 1, 0.2$	[(C_{10},0), (C_{20},0), (C_{30},0)] $FRsys = 0.405479$, $Cost = 9$ KE
[(C_{10},3), (C_{20},0), (C_{30},0)] $FRsys = 0.404199$, $Cost = 9.150301$ KE	[(C_{10},460), (C_{20},420), (C_{30},380)] $FRsys = 0.195004$, $Cost = 72.12625$ KE Time=65.725 sec $\alpha = 0$	[(C_{10},302), (C_{20},0), (C_{30},0)] $FRsys = 0.300117$, $Cost = 24.13026$ KE
[(C_{10},23), (C_{20},0), (C_{30},0)] $FRsys = 0.395788$, $Cost = 10.1523$ KE $\epsilon = 0.999$		[(C_{10},460), (C_{20},266), (C_{30},380)] $FRsys = 0.200004$, $Cost = 64.41082$ KE
[(C_{10},123), (C_{20},0), (C_{30},0)] $FRsys = 0.356958$, $Cost = 15.16232$ KE $\epsilon = 6$		
[(C_{10},183), C_{20}, C_{30}] $FRsys = 0.336165$, $Cost = 18.16834$ KE $\epsilon = 9$		
[(C_{10},303), (C_{20},0), (C_{30},0)] $FRsys = 0.299842$, $Cost = 24.18036$ KE $\epsilon = 15$		
[(C_{10},460), (C_{20},0), (C_{30},143)] $FRsys = 0.239423$, $Cost = 39.21042$ KE $\epsilon = 30$		
[(C_{10},460), (C_{20},163), (C_{30},380)] $FRsys = 0.204292$, $Cost = 59.2505$ KE $\epsilon = 50$		
[(C_{10},460), (C_{20},420), (C_{30},380)] $FRsys = 0.195004$, $Cost = 72.12625$ KE $\epsilon = 100$		

Table V
RESULTS FROM LEXICOGRAPHIC, TABU SEARCH AND MAINTAINERS FOR THE THIRD CONFIGURATION

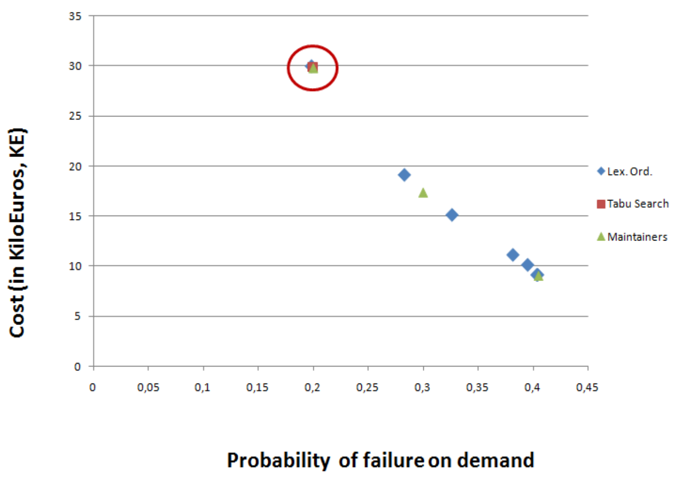

Figure 4. Comparison of the three approaches with respect to the second configuration

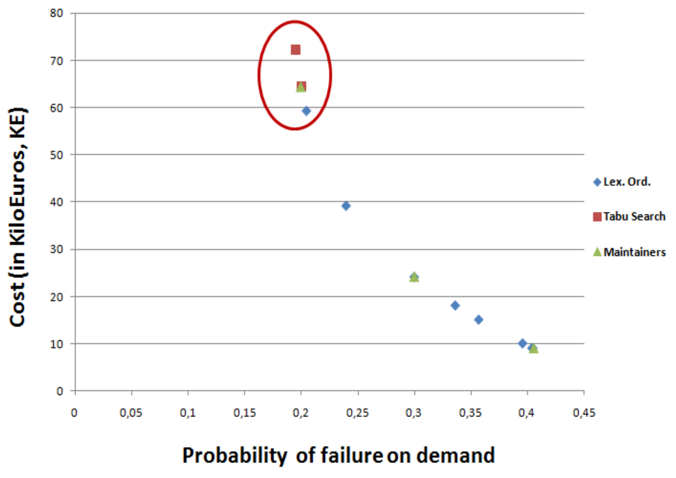

Figure 5. Comparison of the three approaches with respect to the third configuration

method convincingly outperforms the expert judgment approach. In fact, the lexicographic finds optimal solution with a short time while increasing the search space complexity with respect to the maintainers[11]. On the opposite, the tabu search convincingly outperforms the expert/non expert judgment approaches and the lexicographic method. For example, the tabu algorithm allows tackling well-known drawback such as the specification of preferences to arrange the objective functions in order of importance. In fact, it may be difficult to specify preferences with no/limited knowledge

[11]Note that in the figures of results we report Pareto solutions. We have discarded Pareto-dominated solutions.

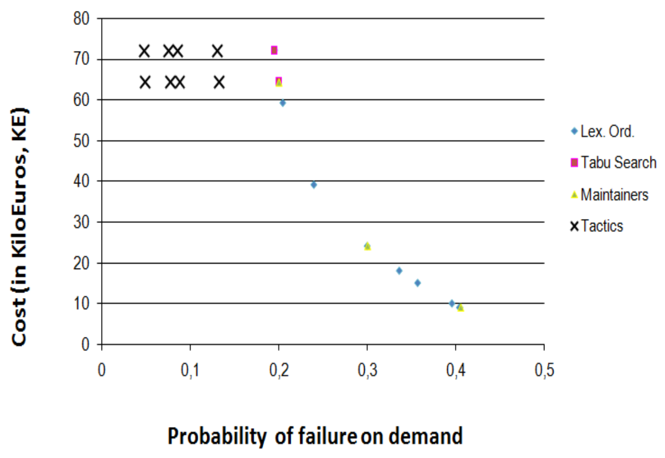

Figure 6. Comparison of the three approaches and the tactics application with respect to the third configuration

on the optimal objective values and, as a consequence, the Pareto-optimal solutions consistent with the given preferences could present the effects of wrong choices.

B. Application of reliability tactics to a tabu search solution

In order to improve the reliability quality attribute of the current solution – indeed the one obtained by the tabu search (as first process iteration) from the third system configuration – some tactics can be taken in consideration and properly composed. Specifically, the *Fault Detection Tactic* can be composed to the *Recovery Reintroduction Tactic* or *Recovery-Preparation and Repair Tactic* (see [3], for details). The *Fault Detection Tactic* is for the detection and notification of a fault to a monitoring component or to the system administrator. The *Recovery Reintroduction Tactic* is for the restoring of the the state of a failed component, whereas the *Recovery Preparation and Repair Tactic* is for the recovering and repairing of a component from a failure. Each kind of these tactics can be refined into other ones (e.g., the *Fault Detection Tactic* in *Ping/Echo*, *Heartbeat* and *Exception* tactics).

The (implementation of one) combination of such reliability tactics can be specified by using basic parameters of the adaptation process, e.g., the reliability of a component depends on the redundant components used by the tactics for it (see the *Voting* tactic in appendix), and be a measure of developer skills (e.g., two applications of the same tactics may impact differently on the in-house instances parameters).

As shown in Figure 6 for the third system configuration, the application of reliability tactics to the solutions provided by the tabu search may increase the system reliability. As done in [16], we have formulated, the probability of failure on demand of an in-house developed instance as a function of the probability that the instance is faulty, the testability and the number of successful test cases performed. The probability that an instance is faulty is an intrinsic property of the instance that depends on its internal complexity. The more complex the internal dynamics of the component instance is, the higher is the probability that a bug has been introduced during its development. The testability expresses the conditional probability that a single execution of a software fails on a test case following a certain input distribution. In [16] procedures to estimate such parameters are suggested. Obviously, different (implementation of) combination reliability tactics may impact differently on the such in-house instance parameters.

Experimental results: In Tables VI and VII we report the detailed results of the reliability tactics application to the solutions provided by the tabu search (with respect to the third system configuration). The tables are organized as follows: each column represents a value of the probability that the in-house instances are faulty p_{i0}, which we have estimated after tactic application[12], each entry (row, column) represents the system reliability resulting after the tactics application.

By analyzing the results we can observe that while varying the tactics application the solution with cost 72.12625 becomes Pareto-dominated by the solution with cost 64.46092 (i.e., the reliability of the solution with cost 64.46092 becomes slightly different to the one of solution with cost 72.12625). Therefore, the application of design solutions may decrease the cost to adapt the system. This highlights the novelty and capabilities of our approach. In fact, this difference would have not been perceived by using approaches that do not predict the quality attributes (and the adaptation cost) of the system resulting after the design solutions application application (like, for example, the works in [17], [18] and [19]).

On the other hand, the application of more sophisticated adaptation actions (e.g., the ones of tactics) may require an higher adaptation cost. This cost could be required, for example, to introduce new components required by the tactics (e.g., the *Recovery* tactics application may involves concepts of clients, a primary component, backup components and a state resynchronization manager). These new components can likely increase the average time required to perform a test case on an in-house instance. For example, if the average time required to perform a test case of all in-house components increases from 0.05 to 0.1, the adaptation cost would increase to 135.2525 KE and 119.9218 KE, respectively, from 72.12625 KE and 64.46092 KE.

Applying again the metaeuristic search: After the tactics application, if the metaeuristic search is applied (as second iteration process) again then a better candidate solution could be found. For example, if after the tactics application, that involves $p_{10} = 0.2$, $p_{20} = 0.2$ and

[12]Larger or more complex implementations (e.g., with a higher number of redundant components) likely improve the probability that an instance is faulty.

$p_{10} = 0.5$ $p_{20} = 0.4$ $p_{30} = 0.4$	$p_{10} = 0.3$ $p_{20} = 0.4$ $p_{30} = 0.4$	$p_{10} = 0.3$ $p_{20} = 0.3$ $p_{30} = 0.3$	$p_{10} = 0.2$ $p_{20} = 0.2$ $p_{30} = 0.2$
$Rsys = 0.867397$	$Rsys = 0.911908$	$Rsys = 0.922630$	$Rsys = 0.950988$

Table VI
TACTICS APPLICATION RESULTS FOR THE SOLUTION OF COST 64.46092

$p_{10} = 0.5$ $p_{20} = 0.4$ $p_{30} = 0.4$	$p_{10} = 0.3$ $p_{20} = 0.4$ $p_{30} = 0.4$	$p_{10} = 0.3$ $p_{20} = 0.3$ $p_{30} = 0.3$	$p_{10} = 0.2$ $p_{20} = 0.2$ $p_{30} = 0.2$
$Rsys = 0.869397$	$Rsys = 0.914010$	$Rsys = 0.924281$	$Rsys = 0.952146$

Table VII
TACTICS APPLICATION RESULTS FOR THE SOLUTION OF COST 72.12625

$p_{30} = 0.2$, we use the tabu search the following candidates are returned.

First candidate: [(C_{10},90), (C_{20},67), (C_{30},76)] The system reliability is equal to 0.913413 and the cost is equal to 20.67335 KE. Note that such a solution involves a lower adaptation cost with respect to the one of solutions obtained only with the tactics application (see Table VI and VII).

Second candidate: [(C_{10},470), C_{23}, (C_{30},500)] The system reliability is equal to 0.902940 and the cost is equal to 60.59719 KE. Note that in this case the tabu search, other than changing the number of test, selects the COTS instance C_{23} for the second component.

C. Application of reliability and performance tactics

The choice of design solutions (e.g., tactics) for a quality attribute is often dictated by the trade-off with other quality attributes. We here show how to compose reliability and performance tactics to embody extra-functional requirements of the STS example into its architecture. Let us assume the following non-functional requirements:
- NFR1.*The STS reliability should be greater than 0.85.*
- NFR2.*The SES sends the trading information of about 600 items every second on average to the STS. Updating such a high volume of information imposes an intensive load on the STS's database, which may cause slow performance. In order to minimize the impairment of performance, updates should be the least possible*[13].

In order to satisfy such new requirements different tactics can be applied[14]. Since they suggest different adaptations actions, they may differ for adaptation cost and/or for the system quality achieved after the application of their actions. Our optimization process allows to combine automatically the tactics by predicting the resulting system quality.

As we have remarked in Section V-B, to address NFR1 the *Fault Detection Tactic* can be composed to the *Recovery Reintroduction Tactic* or *Recovery-Preparation and Repair Tactic*.

As in [3], NFR2 can be addressed by using the *Maintain Multiple Copies tactic* (one of the *Resource Management* tactics). Such a requirement "is concerned with the performance of the STS database which may be decreased by the intensive updates from the SES. The requirement states that the STS receives about 600 items per second. In general, when updates are received at such a high frequency, some items (e.g., stocks having less trades) may not have changes in every update. Taking into account this, many systems use caches to filter out the actual items that need to be updated by comparing the received update with the previous update. To support such selective updates, the *Maintain Multiple Copies* tactic can be used, introducing a cache, a cache client and a cache manager. Using the tactic, we have the cache client receive the update from SES, instead of the database. The cache client then requests the cache manager to update the received update. The cache manager looks up the previous update in the cache and compare it with the one that is received and identify the items that have changes. Only those items that have changes are updated in the database, which reduces the update load on the database"[3].

The resulting SCA assembly obtained by applying all the tactics mentioned above is shown in Fig. 7. The assembly contains a new composite component `MonitoringComponent` for the fault detection tactics *Ping/Echo* and *Heartbeat*. The `OrderDeliveryComponent` is refined into a composite for adding this monitoring functionality. Similarly, the `StockQuoteComponent` is refined into a composite (see Fig. 8)[15] to support selective updates through the *Maintain Multiple Copies* tactic. It contains three components namely a cache `StockQuoteChaceComponent`, a cache client `StockQuoteReceiveComponent`, and a cache manager `StockCacheMgrComponent`. The cache client receives the updates from the external Stock Exchange system, and then it requests the cache manager to update the received update. The cache manager looks

[13]Such a requirement corresponds to the requirement NFR3 of the case study in [3].

[14]The formalization of tactics (for different attributes or concerning a certain quality) composition is outside the scope of this paper. However, to this extent, the binding roles and the composition roles defined in [3] could be exploited.

[15]For the sake of space, we do not report all SCA diagrams.

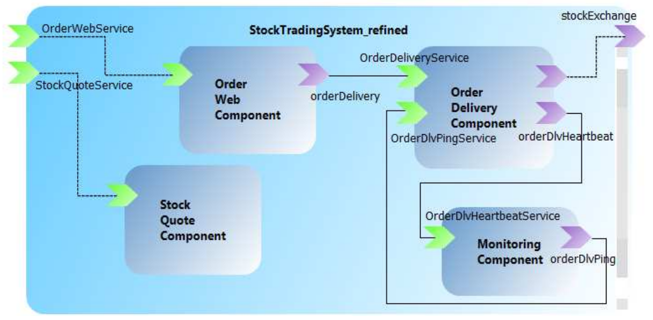

Figure 7. Adapting the STS by applying tactics for NFR1 and NFR2

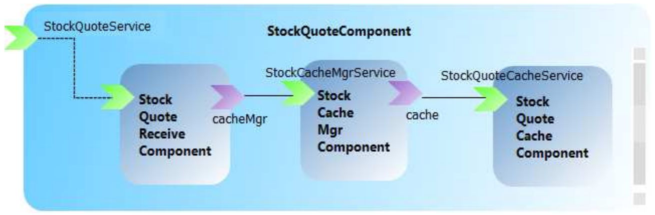

Figure 8. The StockQuote composite for NFR1 and NFR2 tactics

up the previous update in the cache and compare it with the one that is received and identify the items that have changes. Only those items that have changes are updated in the database, thus reducing the database update load [3]. Of course, this adaptation implies a change of the components shape (i.e., in the required/provided interfaces) and also of their behavior.

***Experimental results**:* Starting from the candidate [(C_{10},460), (C_{20},420), (C_{30},380)] (with a cost equal to 72.12625 KE and system reliability equal to 0.804996) returned by the tabu search for the third system configuration, let us apply the *Maintain Multiple Copies (Cache)* tactic for addressing NFR2. Such a tactic is concerned with adaptation actions on the *Stock Quote Component* (i.e., C_2).

As we remarked in Section V-B, the (implementation of one) combination of tactics can be specified by using basic parameters of the adaptation process and different (implementation of) combination tactics may impact differently on the in-house instances parameters.

If to implement the tactic (i.e., a cache, a cache client and a cache manager) the unitary development cost of the in-house instance C_{20} increases from 3 to 5, and its testability increases from 0.002 to 0.006, the reliability of the solution (predicted using the reliability model used in [16]) increases from 0.804996 to 0.811168, and its cost increases from 72.12625 KE to 74.21093 KE. This is due to the fact that, once fixed the number of test cases successfully performed, the probability of failure on demand of a component decreases while increasing its testability.

Since, as remarked in [3], the important issue (i.e., on maintaining the consistency of the copies and keep them synchronized) of the *Maintain Multiple Copies* tactic can be addressed by using the *State Resynchronization* tactic (one of the *Recovery Reintroduction* tactics), let us apply such a tactic (combined with the *Fault Detection Tactic*) to address NFR1 (i.e., satisfy also the reliability requirement). Let us consider an application of such performance and reliability tactics, that increases the average time required to perform a test case for C_{10}, C_{20} and C_{30} from 0.05 to 0.1, from 0.05 to 0.2 and from 0.05 to 0.1, respectively. In the following we report examples of candidates generated for different values of the unitary development cost of the in-house components.

- *First Candidate*: If the unitary development cost of C_{10}, C_{20} and C_{30}, increases from 1 to 2, from 5 to 6 and from 5 to 6, respectively, as well as the probability that the instance is faulty results equal to 0.4, 0.4 and 0.3, then the system reliability increases to 0.903389 and the cost increases to 182.6754 KE.
- *Second Candidate*: If the unitary development cost of C_{10}, C_{20} and C_{30}, increases from 1 to 3, from 5 to 7 and from 5 to 9, respectively, as well as the probability that the instance is faulty results equal to 0.3, 0.2 and 0.2, then the reliability increases to 0.934843 and the cost increases to 187.6754 KE.
- *Third Candidate*: If the unitary development cost of C_{10}, C_{20}, C_{30} increases from 1 to 2, from 5 to 6 and do not change, respectively, as well as, the probability that the instance is faulty results equal to 0.4, 0.4 and 0.5, then the system reliability increases to 0.887101 and the cost increases to 181.6754 KE.

***Applying again the metaeuristic search**:* As we have remarked in V-B, if we would apply again the metaeuristic technique, then better solutions might be found. For example, if we would use the tabu search by starting from the *third candidate*, then we will obtain the following candidate: [(C_{10},800), C_{23}, (C_{30},700)]. The reliability is equal to 0.881299 and the cost decreases from 181.6754 KE to 163.3006 KE. Such solutions differ for the number of test for C_{10} and C_{30}, and the component selected for C_2.

D. Combining reliability, performance and security tactics

Let us assume that, other than requiring NFR1 and NFR2, a user of the STS system also requires NFR3 (i.e., "*Only authenticated users can access the system. The user credentials must not be seen to unauthorized personnel while transmitting the information to the system*[16](see [3])).

To address NFR3, as suggested in [3], the *ID/Password* and *Maintain Data Confidentiality* tactics could be adopted.

Such security tactics may impact, for example, on the system response time (e.g., the introduction of a components for user login may increase the user response time). In fact, the response time and the adaptation cost of *Order Web Component* will likely increase.

Starting from the *Third Candidate* (see previous section) let us apply such security tactics. If to implement such

[16]Such a requirement corresponds to the requirement NFR4 of the case study in [3].

tactics the average response time of *Order Web Component* increases (i.e., "the number of high-level instructions of the component / the number of instructions per second that the host running the component can execute" increases from 0.03 to 0.1), then average system response time (predicted using [20]) increases from 11.2 sec to 23.8 sec.

E. Combining tactics and architectural patterns

We here show how our optimization process allows to combine different design solutions, such as tactics and architectural design patterns.

Let us assume that, other than NFR3, the following requirement is claimed for the STS:
- F1. *The STS should convert the stock price in the user preferred currency*.

To address *F1* we adopt the *Pipe and Filter* architectural pattern. This pattern provides a structure for systems that process a stream of data. Each processing step is encapsulated in a filter component. Data is passed through pipes between adjacent filters [21].

A filter is added to `OrderDeliveryComponent` for converting the currency into the required one, while the users check the current price of stocks, place buy or sell orders and review traded stock volume. Similarly, a filter is added to `StockQuoteComponent` for converting the currency into the required one when the users want to know stock quote information.

As we have remarked in Section V-D, NFR3 can be addressed by using the *ID/Password* and the *Maintain Data Confidentiality* security tactics (belong to the *Resisting Attacks* tactic).

Note that in this case the tactics do not impact on the architectural pattern: they are applied on different components of the system.

On the other hand, as we show below, tactics may impact on architectural patterns (see, for example, [17], [18] where a qualitative analysis of the interaction between reliability tactics and architectural pattern is provided).

Let us assume that for the STS system it is required, other than F1, the requirement NFR1* (i.e., *The STS reliability should be greater than 0.9.*).

To address F1 and NFR1* the *Pipe and Filter* architectural pattern and the *Fault Detection Tactic* can be composed, for example, to the *Voting* (or *Active Redundancy*) tactic. In [17] the combined use of the *Pipe and Filter* and the *Voting* (or *Active Redundancy*) tactic is classified as "Good Fit" (i.e., the structure of the pattern is highly compatible with the structural needs of the tactic). In [17] can be also found details, which we do not report for the sake of space, on the implementation of these tactics and architectural pattern.

Starting from the candidate [(C_{10},460), (C_{20},420), (C_{30},380)] (with a cost equal to 72.12625 KE and system reliability equal to 0.804996) returned by the tabu search for the third system configuration (see Section II), let us apply the *Pipe and Filter* and the reliability tactics.

Let us consider a their implementation, where the average time required to perform a test case on C_{10}, C_{20} and C_{30}, increases from 0.05 to 0.1, from 0.05 to 0.2 and from 0.05 to 0.1, respectively, as well as the unitary development time of the instances increases from 1 to 3, from 3 to 6 and does not change. Besides, the probability that C_{10}, C_{20} and C_{30} are faulty results equal to 0.3, 0.2 and 0.2, respectively. NFR1* is not satisfied with respect to this tactics and design pattern application. In fact, the system reliability is equal to 0.863466, while the adaptation cost is equal to 182.6754 KE.

Applying again the metaeuristic search: If we would apply again a tabu search algorithm[17] after the tactics and design pattern application, for example, the following candidate is returned: [(C_{13}), (C_{20},420), (C_{30},380)] with reliability equals to 0.954962. Note that NFR1* is satisfied. Besides, the cost decreases from 182.6754 KE to 139.5832 KE. The tabu search returns such a solution by applying a user adaptation plan[18] suggesting to replace C_1 with a new component. Its probability of failure on demand is equal to 0.00001, while its cost is equal to 6 KE.

This highlights the novelty and capabilities of our approach. In fact, this difference would have not been perceived by using approaches that do not combine the metaeuristic techniques and the design solutions, and do not predict the quality attributes of the system resulting after the adaptation actions application (like, for example, the works in [17], [18] and [19]).

APPENDIX

In this appendix we describe the tactics used in the STS case study. In [3] and [23] more details can be found. The formalization of tactics composition is outside the scope of this paper. However, to this extent, the binding roles and the composition roles defined in [3] could be exploited.

A. Reliability Tactics

As remarked in [23], it does not exist a universally accepted terminology for the various tactics of fault tolerance.

As shown in Figure 9, the several tactics for reliability can be categorized into *Fault Detection*, *Fault Recovery Preparation and Repair*, and *Recovery Reintroduction of a*

[17]Such algorithm enhances the tabu search, which we have used in Section II, by generating new candidates also using user adaptation plans.

[18]This could require the direct interaction between our optimization process and the maintainers. Obviously, sometimes such direct interaction could be required (e.g., as remarked in [22], the application of an anti-pattern solution may require manual interaction because, for example, a performance model of the system usually does not contain enough information to decide whether a cache is applicable). However, such a kind of manual interaction may be required only for system evolution (at re-design time) where speedy answers are not essential, whereas it is not required for self-adaptation (at run time).

Failed Component tactics[19]. In figure we have circumscribed the tactics, which we have used.

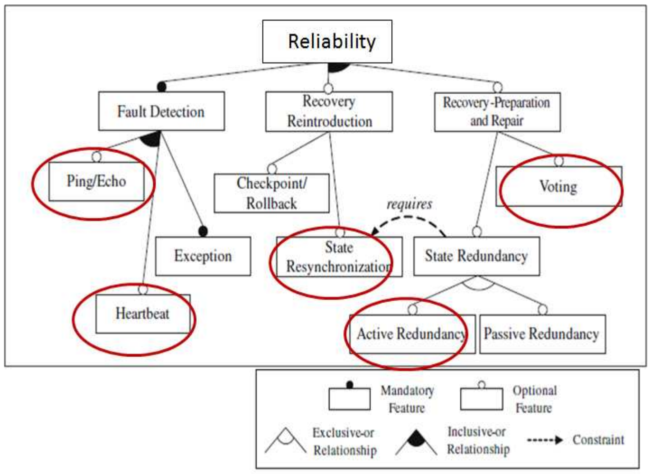

Figure 9. Reliability architectural tactic feature model.

The *Fault Detection Tactic* is for the detection and notification of a fault to a monitoring component or to the system administrator. The *Recovery Reintroduction of a Failed Component* is for the restoring of the the state of a failed component, whereas the *Fault Recovery Preparation and Repair* is for the recovering and repairing of a component from a failure. Each kind of these tactics can be refined into other ones.

Fault Detection Tactic

In the STS case study, we have refined the *Fault Detection Tactic* in *Ping/Echo*, *Heartbeat* tactics. The following description of such tactics are taken from [23]. In [3] diagrams better illustrating their specification can be also found.

- *Ping/Echo*: A monitoring component issues a ping message to one or more components under scrutiny, and expects to receive an echo message back within a predetermined time. If a component does not respond within the time limit, the monitoring component considers that component to be in failure mode, and takes corrective actions. Implementation requires that a monitoring process be created or used, and that all components being watched must be modified to handle the echo messages.

- *Heartbeat*: A component emits a heartbeat message at regular intervals and a monitoring component listens for it. If no heartbeat message is received within a predetermined time, the originating component is assumed to have failed, and corrective actions are taken. This tactic requires a monitoring component, and all components must be modified to send heartbeats at the proper intervals.

Recovery Reintroduction of a Failed Component

[19]The figure has been inspired by the figure in [3] for the availability tactic. The figure could be refined/exploited by considering other reliability tactics.

In Section V-C we have combined the *Fault Detection Tactic* and the *State Resynchronization* tactic (one of the *Recovery Reintroduction* tactics).

Such a tactic restores the state of a component through resynchronization with the state of a backup component. It involves concepts of a state resynchronization manager, source components and backup components. In [3] more details and diagrams illustrating the tactic can be found.

Fault Recovery Preparation and Repair

In Section V-E we have combined the *Fault Detection Tactic* and the *Voting* (or *Active Redundancy*) tactic. The following description of such tactics are taken from [23].

- *Voting*: Processes running on redundant processors each take equivalent input and compute a single out value that is sent to a voter. The voter component decides which of the results is correct using an algorithm such as majority rules. The strongest approach is to implement each voting component independently; otherwise you can only detect hardware faults, and not algorithm faults. (If the voting components are running the same software, this tactic becomes very similar to Active Redundancy; see below.) To implement voting, create a voter component, and either replicate or write a new voting component.

- *Active Redundancy*: redundant components receive events in parallel, thus they are always in the same state. If one component fails, the other can immediately take over. This tactic that the processing component(s) to be replicated. It usually requires a central arbitrating component, although it is possible to make the redundant components perform the arbitrating without a central component.

B. Performance Tactics

As shown in Figure 10 (taken from [3]), the several tactics for performance can be categorized in *Resource Arbitration* and the *Resource Management* tactics. The first is used for improving performance by scheduling requests for expensive resources (e.g., processors, networks), whereas the latter improves performance by managing resources affecting response time (see [3] for details). In figure we have circumscribed the tactics, which we have used.

In Section V-C we have used the *Maintain Multiple Copies tactic* (one of the *Resource Management* tactics). This tactic allows to manage resources by keeping replicas of resources on separate repositories, so that contention for resources can be reduced. “The tactic involves concepts of clients, a cache, a cache manager and a data repository. The cache maintains copies of data that are frequently requested for faster access. When a data request is received, the cache manager first searches the cache. If the data is not found, then the cache manager looks up the repository and makes copies of the data into the cache.”[3]

C. Security Tactics

As shown in Figure 11 (taken from [3]), the several tactics for security can be categorized in *Resisting Attacks* and the

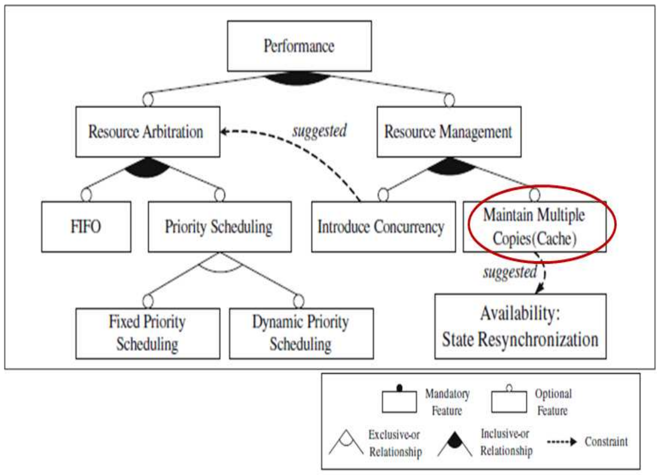
Performance
Resource Arbitration
suggested
Resource Management
FIFO
Priority Scheduling
Introduce Concurrency
Maintain Multiple Copies(Cache)
suggested
Fixed Priority Scheduling
Dynamic Priority Scheduling
Availability: State Resynchronization
Mandatory Feature
Optional Feature
Exclusive-or Relationship
Inclusive-or Relationship
Constraint

Figure 10. Performance architectural tactic feature model.

Recovering from Attacks tactics (see [3] for details). In figure we have circumscribed the tactics, which we have used in Section V-D and V-E.

We have applied the *ID/Password* and the *Maintain Data Confidentiality* tactics. Such tactics are used for protecting the system from attacks. The *ID/Password* tactic checks authentication of the user using the users credentials (i.e., user IDs, passwords), whereas the *Maintain Data Confidentiality* allow to protect data from unauthorized modifications using encryption and decryption.

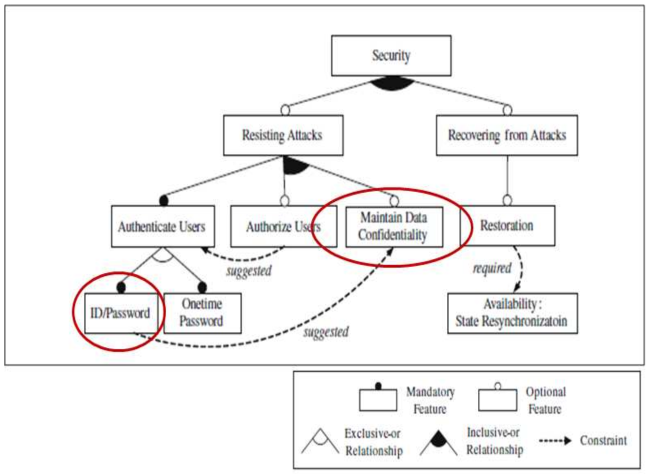

Figure 11. Security architectural tactic feature model.

References

[1] C. Blum and A. Roli, "Metaheuristics in combinatorial optimization: Overview and conceptual comparison," *ACM Comput. Surv.*, vol. 35, no. 3, pp. 268–308, 2003.

[2] "OSOA. Service Component Architecture (SCA) `www.osoa.org`."

[3] S. Kim, D. Kim, L. Lu, and S. Park, "Quality-driven architecture development using architectural tactics," *Journal of Systems and Software*, no. 8, pp. 1211–1231, 2009.

[4] R. Marler and J. Arora, "Survey of multi-objective optimization methods for engineering," *Structural and Multidisciplinary Optimization*, vol. 26, pp. 369–395, 2004.

[5] V. Cortellessa, F. Marinelli, and P. Potena, "An optimization framework for "build-or-buy" decisions in software architecture," *Computers & OR*, vol. 35, no. 10, pp. 3090–3106, 2008.

[6] S. Gokhale, "Architecture-based software reliability analysis: Overview and limitations," *IEEE Trans. Dependable Sec. Comput.*, vol. 4, no. 1, pp. 32–40, 2007.

[7] J. M. Voas and K. W. Miller, "Software testability: The new verification," *IEEE Software*, vol. 12, pp. 17–28, 1995.

[8] D. Hamlet, D. Mason, and D. Woit, "Theory of software reliability based on components," in *Proceedings of the 23rd International Conference on Software Engineering*, ser. ICSE '01, 2001, pp. 361–370.

[9] F. Rosenberg, M. Müller, P. Leitner, A. Michlmayr, A. Bouguettaya, and S. Dustdar, "Metaheuristic optimization of large-scale qos-aware service compositions," in *Proceedings of the 2010 IEEE International Conference on Services Computing*, 2010, pp. 97–104.

[10] A. Martens and H. Koziolek, "Automatic, model-based software performance improvement for component-based software designs," *Electronic Notes in Theoretical Computer Science*, vol. 253, no. 1, pp. 77 – 93, 2009, proceedings of the Sixth International Workshop on Formal Engineering approches to Software Components and Architectures (FESCA 2009).

[11] Y. Censor, "Pareto optimality in multiobjective problems," *Applied Mathematics & Optimization*, vol. 4, pp. 41–59, 1977.

[12] L. Paquete and T. Sttzle, "A study of stochastic local search algorithms for the biobjective qap with correlated flow matrices," *European Journal of Operational Research*, vol. 169, no. 3, pp. 943 – 959, 2006.

[13] R. Sedgewick, "Implementing quicksort programs," *Commun. ACM*, vol. 21, pp. 847–857, 1978.

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

[15] A. Bertolino and L. Strigini, "On the use of testability measures for dependability assessment," *IEEE Trans. Softw. Eng.*, vol. 22, pp. 97–108, 1996.

[16] V. Cortellessa, F. Marinelli, and P. Potena, "Automated selection of software components based on cost/reliability tradeoff," in *Proc. of 3rd European Workshop on Software Architecture (EWSA 2006)*, ser. Lecture Notes in Computer Science, vol. 4344, 2006, pp. 66–81.

[17] "How do architecture patterns and tactics interact? a model and annotation," *Journal of Systems and Software*, vol. 83, no. 10, pp. 1735 – 1758, 2010.

[18] N. Harrison and P. Avgeriou, “Implementing reliability: The interaction of requirements, tactics and architecture patterns,” in *Architecting Dependable Systems VII*, ser. Lecture Notes in Computer Science, 2010, vol. 6420, pp. 97–122.

[19] K. Vallidevi and B. Chitra, “Effective self adaptation by integrating adaptive framework with architectural patterns,” in *Proc. of the 1st Amrita ACM-W Celebration on Women in Computing in India*, ser. A2CWiC ’10. ACM, 2010.

[20] B. Boone, T. Verdickt, B. Dhoedt, and F. D. Turck, “Design time deployment optimization for component based systems,” in *Proc. of IASTED 2007*, 2007, pp. 242–248.

[21] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, and M. Stal, *Pattern-Oriented Software Architecture, Volume 1: A System of Patterns*. Wiley, 1996.

[22] A. Martens and H. Koziolek, “Performance-oriented Design Space Exploration,” in *Proc. of WCOP’08*, 2008.

[23] N. B. Harrison and P. Avgeriou, “Incorporating fault tolerance tactics in software architecture patterns,” in *Proceedings of the 2008 RISE/EFTS Joint International Workshop on Software Engineering for Resilient Systems*, ser. SERENE ’08, 2008, pp. 9–18.