An Optimization Process for Adaptation Space Exploration of Service-oriented Applications

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—This paper proposes 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 the use of functional/extra-functional patterns (i.e., architectural design patterns and tactics). Moreover, the proposed approach relies on the standard Service-oriented Component Architecture (SCA) for heterogeneous service assembly and its runtime platforms.

Keywords—Service-oriented applications; software adaptation and evolution; meta-heuristic search techniques

I. INTRODUCTION

Service-oriented applications are playing an important role in several application domains (e.g., health care, defense and aerospace) since they offer complex and flexible functionalities in widely distributed environments by composing, possibly on demand, different types of services. These applications may require dynamic adaptation to changing user needs, system intrusions or faults, changing operational environment, resource variability, etc. Extra-functional properties of services are often specified as quality of service (QoS) constraints and their values are dynamic [1]. For example, the system response time depends on environmental factors among which input data, server load, and network latency.

In order to achieve the right trade off among the functional requirements, software qualities (such as performance and reliability) and the adaptation cost itself, the adaptation decisions 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.). A decision, for example, taken for modifying the dynamic of a service may be good for the satisfaction of the system reliability, but at the same time it may require a high adaptation cost for adapting the interfaces of services [2]. A major challenge is then finding the best balance between different, possibly conflicting, quality requirements that a system has to meet and cost constraints (e.g., maximize performance and availability, while minimizing cost). For these multi-attribute problems, there is usually no single global solution, and the generation and evaluation of design alternatives can be error-prone and lead to suboptimal design decisions, especially if carried out manually by system maintainers. Besides, this evaluation can suffer of large elapsed time when the search space size increases. In such cases, the complete enumeration of possible alternatives results inefficient and metaheuristic techniques has been proposed as a viable solution to deal with this problem (e.g., see [1]). Other promising approaches are the application of architecture patterns and tactics [4], [5]. Architecture patterns are chosen in response to early design decisions, and provide the major structures in which multiple design decisions are realized. When design decisions concern quality attributes they are often called tactics. So, a tactic is a design decision whose goal is the improvement of one specific design concern of a quality attribute.

This paper focuses on these recent results and proposes 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 the application of both meta-heuristic search techniques and architectural design patterns and tactics. Moreover, the proposed methodology relies on the OSOA standard Service-oriented Component Architecture (SCA) [6] for heterogeneous service assembly and related runtime platforms to process architectural models (of the application to adapt). A formal and executable service-oriented component model, called SCA-ASM [7], is also adopted for the behavioral specification and functional analysis of service-oriented components by exploiting the Abstract State Machine (ASM) [8] formal method and the associated analysis toolset ASMETA [9].

This paper focuses on the design of the reasoning process and of the involved optimization algorithms/techniques. As example of adaptation framework in which this reasoner module could be used, see [3].

The remainder of this paper is organized as follows: Section II briefly surveys related works, while Section III discusses some background information on the SCA-ASM language. The proposed process for adaptation space exploration and some evaluation results are presented in Section IV. Section V concludes the paper.
II. RELATED APPROACHES

Here we shortly review the main methods for adaptation space exploration with the goal of improving system quality. 

Rule-based Approaches, for example, include the work of Xu et al. [12] presenting a semi-automated approach to find configuration and design improvement on the model level. Parsons et al. [13] present a framework for detecting performance anti-patterns in Java EE architectures. The method requires an implementation of a component-based system, which can be monitored for performance properties. Cortellessa et al. [14] propose an approach for automated feedback generation for software performance analysis, which aims at systematically evaluating performance prediction results using step-wise refinement and the detection of performance problem patterns. All rule-based approaches share a common limitation. The model can only be changed as defined by the improvement rules. However, especially performance is a complex and cross-cutting quality criterion. Thus, optimal solutions could lie on search paths not accessible by rules. 

Metaheuristic-based Approaches: Evolutionary approaches are applied in [15] for the improvement of availability and costs, in [16] for the study of the trade-off between performance and cost. Evolutionary algorithms together with architectural patterns are used in the SASSY framework for generating service-oriented architectures based on quality requirements [17]. In these approaches the derived optimization process is time-consuming. 

Software architecture analysis approaches such as SAAM and ATAM [18] analyze the software architecture with respect to multiple quality attributes exploring also tradeoffs concerning software qualities in the design. The outputs of such analysis include potential risks of the architecture and the verification result of the satisfaction of quality requirements. Methods using architectural patterns and tactics for quality attributes are presented in [4], [19], while [20], [5] describe how software adaptation patterns can be used to help with the adaptation of service-oriented software systems after original deployment. These methods provide qualitative results and are mainly based on the experience and the skill of designers and on the collaboration with different stakeholders.

With respect to the state-of-art, our approach is the first optimization process (to the best of our knowledge) that supports the adaptation of service-oriented systems including both static and dynamic aspects at runtime and at redesign time. It uses a mixed approach of metaheuristic search technique and of (repository-based) design solutions. Moreover, this methodology relies on a formal and executable service-oriented component model, i.e. SCA-ASM, both for the functional and non-functional analysis driving the adaptation activity. Finally, our optimization process dynamically generates adaptation strategies facilitating the work of a maintainer.

III. BACKGROUND ON THE SCA-ASM LANGUAGE

The SCA-ASM modeling language [7] adopts a suitable subset of the OSOA standard Service Component Architecture (SCA) [6] for heterogenous service assembly, and complements it with the Abstract State Machine (ASM) [8] model of computation\(^1\) to provide an ASM-based SCA component implementation type for modeling notions of service behavior, interactions, orchestration, and compensation in an abstract, formal, and executable way.

According to the SCA-ASM implementation type and exploiting the notion of distributed multi-agent ASMs, a service-oriented component is an ASM endowed with (at least) one agent (a business partner or role instance) able to be engaged in conversational interactions with other agents by providing and requiring services to/from other service-oriented components’ agents. The service behaviors encapsulated in an SCA-ASM component are captured by ASM transition rules and state invariants, which are the basis for the behavioral specification and functional analysis of service-oriented components. An open framework, the ASMETA tool set [9] (based on the Eclipse/EMF platform) is also available for editing, exchanging, simulating, testing, and model checking ASM models.

Mechanisms for enacting and managing self-adaptation of SCA applications are being developed for SCA run-time platforms (see, e.g., [21]).

IV. ADAPTATION SPACE EXPLORATION

Basically, an SCA assembly (i.e., the architecture of the considered application) can be adapted through the following atomic adaptation actions: adding/removing components, component services, references, properties, reference-service wires and promotion wires (component interactions); changing a component implementation (but keeping its shape); changing component properties values; changing SCA domains (components re-deployment). It is also possible to change the component interaction style in synchronous/asynchronous, stateful or not, unidirectional or bidirectional. 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 [6] 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.

Adaptations actions can be combined into an adaptation plan, which is a set of actions modifying the static and

\(^1\)ASMs are an extension of FSMs: states are arbitrary complex data (multi-sorted first-order structures) and the transition relation is specified by rules describing how functions change from one state to the next.
dynamic parts of a system architecture to address a certain requirement. Adaptation plans may differ for adaptation cost and/or for the system quality achieved after their application.

The proposed adaptation space exploration process is shown in Fig. 1. It starts from a set of initial SCA-ASM assemblies (initial candidates or population) fulfilling the existing/new functional requirements. It proceeds iteratively till stop criteria\(^2\) [22] are satisfied. At each iteration step, new candidates are generated from the initial population (whose size depends on the specific search technique) by two subprocesses executed in parallel: (i) metaheuristic search by applying user adaptation plans, service selection and service re-deployment; (ii) functional/extra-functional patterns application by exploiting architectural design patterns and tactics. Then the functional and quality analysis of the resulting candidates are performed together with an assessment of the adaptation costs. In case of self-adaptation, resulting SCA-ASM assemblies are automatically selected as solutions according to predefined selection criteria (e.g., cost minimization). In case of evolution, the solution can be more accurately selected also considering a possible feedback from the user [10].

In order to support such proposed optimization process, we have also been working on developing a suitable software-based “reasoner module” that can be easily embedded in any general-purpose adaptation framework. This software module (see Fig. 2) may be activated after receiving (as input) either adaptation requests from the user or alerts from monitors (or probes). By executing the process described above, this reasoner produces as output Pareto-efficient alternatives. Each point of a Pareto curve is a chain of software adaptation actions that, if executed, lead to the new system architecture, i.e., the adapted SCA-ASM assembly. These changes applied at SCA model level must be related then (through the use of effectors) to the underlying mechanisms and runtime infrastructure. In the case of SCA, mechanisms like introspection and reconfiguration, for managing and enacting self-adaptation of SCA assemblies [21] are applied.

A more detailed description of the main phases follows.

**New candidates generation:** As first generation method, we rely on the use of metaheuristic search techniques. Their effectiveness and efficiency has been already demonstrated for supporting the service selection activity at run-time (e.g., see [1]). As remarked in [1], 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 metaheuristic techniques [23] with different characteristics could be adopted depending on the nature of the problem: for example, considering the system reliability, a possible heuristic is to regard as the reliability of the whole system increases when the reliability of the most used components increases. As remarked in [24], 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). In our optimization approach, we adopt the Tabu Search heuristic technique described below, in the Subsection IV-A. For comparison purposes only, we also consider other two methods that are described below in Subsection IV-B.

The second method generates new candidates by applying recurring software design solutions, such as architectural design patterns and tactics. Architectural design patterns are templates for concrete software architectures. They are adopted to embody functional requirements and, in particular, to enable self-adaptability by introducing sensors/effectors components (e.g., Microkernel pattern, Reflection pattern, Interception pattern) [11]. To build new design solutions embodying extra-functional requirements, we adopt architectural tactics [25], which are reusable architectural

---

\(^2\)We currently use a predefined number of iterations to determine the end of the search. More sophisticated stop criteria could use convergence detection and stop when the global optimum is probably reached.
building blocks that provide a generic solution to address issues pertaining to quality attributes.

**Functional and Quality Analysis:** This phase aims at guaranteeing the functional correctness of the resulting assembly and that changes claimed by the adaptation actions do not compromise the satisfaction of existing non-functional requirements. To this purpose, a set of external analysis tools can be invoked for the functional and non-functional analysis, respectively. The functional analysis is assumed (but not limited to) that is executed with the help of the ASM analysis toolset ASMETA [9]. Preliminary analysis of the functional requirements satisfiability of the SCA-ASM assembly would be performed by easier techniques as simulation or scenario-based validation. Later, heavier formal verification techniques (as model checking) can be exploited when more complex functional properties [26] must be proved to guarantee behavioral system correctness.

For the non-functional analysis, external tools for performance and reliability analysis like qnetworks [27] and LQNsolver [28] are exploited. Examples of adaptation costs prediction techniques can be found in [2]. In general, different approaches/strategies can be used depending on several factors due mainly to the use of our process for evolution or self-adaptation. If permanent changes, for example, are requested or a safe-critical service has to be adapted, precise (often expensive) analysis must be performed (e.g., see [29] for performance analysis). As opposite, if runtime changes are claimed and these require, for example, only the adaptation of parameters without using more sophisticated analysis, faster approaches must be adopted allowing a prompt run-time adaptation (see, e.g., techniques for estimation of quality at runtime, such as [30]).

Similar considerations also apply for functional analysis.

### A. Multi-objective Optimization and Pareto solutions.

As already stated, we deal with multi-attribute problems and the proposed optimization process exploits the multi-objective optimization [31], where the objectives represent different quality attributes. The aim of these techniques is to devise a set of solutions, called Pareto optimal solutions or Pareto front, each of which assures a trade-off between the conflicting qualities. In other words, while moving from one Pareto solution to another, there is a certain amount of sacrifice in one objective(s) to achieve a certain amount of gain in the other(s).

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 $D_q$ 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 $s$

1. $s \leftarrow \text{GenerateInitialSolution}()$
2. $\text{TabuList} \leftarrow \emptyset$
3. \hspace{1cm} // $s'$ memorizes the best solution of
4. \hspace{1cm} // the tabu search
5. \hspace{1cm} $s' \leftarrow s$
6. \hspace{1cm} while termination conditions not met do
7. \hspace{1cm} \hspace{1cm} $\text{NeighboursOkSet} \leftarrow \text{ExploreNeighbourhood}(s)$
8. \hspace{1cm} \hspace{1cm} $s \leftarrow \text{ChooseBestOf}(\text{NeighboursOkSet})$
9. \hspace{1cm} \hspace{1cm} $\text{UpdateTabuList}$
10. \hspace{1cm} \hspace{1cm} if ($f(s') > f(s)$) then
11. \hspace{1cm} \hspace{1cm} \hspace{1cm} $s' \leftarrow s$
12. \hspace{1cm} \hspace{1cm} end if
13. \hspace{1cm} end while

Figure 3. Algorithm: Tabu Search.

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 $AS$.

In the remainder of this section we review the main methods that we use to find these Pareto optimal solutions.

1) **Tabu Search** The Tabu Search (TS) is among the most cited and used heuristics 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 3. A description of its main steps follows.

**Begin with a starting current solution** The initial candidate $s$ (an SCA-ASM assembly) fulfilling the existing/new functional and non-functional requirements is generated.

**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\(^3\) is restricted to the solutions that do not belong to the tabu list (i.e., definition of NeighboursOkSet in Figure 3). 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 3. The candidate becomes the basis for next candidates

\(^3\)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 $s$ [25]
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. **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 [23] for details).

### B. An experimental comparison with other methods

We exemplified and experimented the proposed adaptation process through a sample application from the Stock Trading System (STS) in [25]. Details on the experimental data set and results of this case study can be found in [35]. For comparison purposes, in our experimentation we considered two other methods to generate alternative adaptation solutions: the lexicographic method [32], and the judgment of a group of (human) maintainers formed by expert/non expert with respect to the system and execution environment.

We implemented the *lexicographic method* as follows. First we solved the optimization model minimizing the adaptation cost under reliability and delivery time constraint (i.e., the model presented in [33]), then we 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 \( \epsilon \) is a positive tolerance (real number). Finally, we found the set of Pareto optimal solutions by varying \( \epsilon \) (i.e., we have applied the \( \epsilon \)-constraint approach [32]). We proceeded the process till the stop criteria was satisfied. For the experimentation we used the LINGO tool [34], 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.

We applied the approaches on three different configurations of the STS system (see [35] for details). In order to keep our model as simple as possible, in all configurations we assumed 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. As conclusive example, Fig. 4 reports the approximate Pareto curves obtained from solving the optimization problem using the three approaches with respect to the third configuration presenting the most complex search space with 4. In the figure, the feasible solutions are circumscribed.

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.

### V. Conclusions and future work

This paper presented an automatic optimization process for adaptation space exploration of service-oriented applications. We intend to enhance our methodology towards several directions. In particular, we intend to support the right trade-off between the adaptation overhead (due, e.g., to the frequent execution of the reasoning algorithms) and the accrued benefits of changing the system. Moreover, by observing the work of the maintainers and experimenting the optimization process personally, we aim at providing means to allow the automatic composition of design patterns and tactics and their application on SCA-ASM system models.

### References
