Frontiers of Information Technology & Electronic Engineering
www.zju.edu.cn/jzus; engineering.cae.cn; www.springerlink.com
ISSN 2095-9184 (print); ISSN 2095-9230 (online)
E-mail: jzus@zju.edu.cn

Towards a self-adaptive service-oriented methodology based on extended SOMA

Alireza PARVIZI-MOSAED‡, Shahrouz MOAVEN, Jafar HABIBI,
Ghazaleh BEIGI, Mahdieh NASER-SHARIAT

(*Computer Engineering Department, Sharif University of Technology, Tehran 11155-11365, Iran*)

E-mail: aparvizi@ce.sharif.edu; moaven@ce.sharif.edu; jhabibi@sharif.edu; ghbeigi@ce.sharif.edu; nasershariat@ce.sharif.edu

Received Feb. 7, 2014; Revision accepted Aug. 15, 2014; Crosschecked Dec. 8, 2014

Abstract: We propose a self-adaptive process (SAP) that maintains the software architecture quality using the MAPE-K standard model. The proposed process can be plugged into various software development processes and service-oriented methodologies due to its explicitly defined inputs and outputs. To this aim, the proposed SAP is integrated with the service-oriented modeling and application (SOMA) methodology in a two-layered structure to create a novel methodology, named self-adaptive service-oriented architecture methodology (SASOAM), which provides a semi-automatic self-aware method by the composition of architectural tactics. Moreover, the maintenance activity of SOMA is improved using architectural and adaptive patterns, which results in controlling the software architecture quality. The improvement in the maintainability of SOMA is demonstrated by an analytic hierarchy process (AHP) based evaluation method. Furthermore, the proposed method is applied to a case study to represent the feasibility and practicality of SASOAM.

Key words: Service-oriented architecture, Self-adaptive process, Architectural pattern, Quality attribute, Adaptation pattern, Architectural tactic

doi:10.1631/FITEE.1400040 **Document code:** A **CLC number:** TP311

1 Introduction

In the software development life cycle (SDLC), maintenance is a costly and time-consuming step, in which both time and cost are growing rapidly due to the flexibility and dynamicity features of service-oriented systems (SOSs) (Rosen *et al.*, 2008). Quality assurance is one of the most complex activities of this step due to its dependency on the SDLC assets; e.g., the quality of design models and source codes affects system quality. Besides, stakeholders do not have a common point of view about the quality attributes due to the ambiguity of these concepts. To this aim, some quality models such as Bohem (Rawashdeh and Matalkah, 2006), McCall, ISO/IEC-9126 (Fitzpatrick, 1996), and Dromey (Dromey, 1995) have been defined as templates, which decompose quality attributes into several measurable sub-quality attributes in order to integrate different viewpoints of stakeholders. Due to the side effect of different quality attributes on each other, quality models can be customized by feature models to consider the trade-off among them.

As the manipulation of software architecture significantly impacts software quality, the quality of SOS can be controlled by architectural models. Software architectures can be defined with several points of view to reduce software complexity by handling different aspects of software quality (Bass *et al.*, 1998); e.g., the 4+1 View Model organizes software architectures in logical, process, development, physical, and validation views (Kruchten, 1995) to measure software quality from the

‡ Corresponding author
ORCID: Alireza PARVIZI-MOSAED, http://orcid.org/0000-0002-1957-2960

aforementioned views. Moreover, model driven architectures (MDAs) can be applied to software architectures to automatically generate source codes from architectural models (di Nitto *et al.*, 2008).

Software architects usually encounter a major problem when they demand to measure system quality. Ignoring the implementation of components at the architectural level makes it difficult to quantify the system quality based on software architecture models. As architectural patterns provide practical solutions for distinct problems in certain contexts, pattern composition is an appropriate solution to resolve the aforementioned challenge (Coad, 1992). Moreover, architectural and design patterns represent system quality according to some criteria such as cohesion, coupling, and shared caching of interactions between components (Fielding, 2000). Therefore, in our previous work (Moaven *et al.*, 2008a; 2008b), we have proposed a few fuzzy-based and analytic hierarchy process (AHP) based methods to select the best composition of architectural patterns. In addition, architectural patterns provide specific structures for software architectures to simplify their reconfigurations.

Quality assurance is a complex and repetitive activity, which analyzes and maintains the quality of systems. Automating this activity reduces quality detection errors besides the improvement in the precision of software architecture analysis and reconfiguration. The self-adaptive service-oriented architecture is an automated approach which adapts service-oriented architecture (SOA) to the manipulation of its context. Self-adaptation, sometimes named the self-property, is a general concept which describes the capability of systems to adapt their properties to environmental changes; e.g., self-configuring and self-managing are two automatic approaches for configuring and managing software architectures, respectively (Romay *et al.*, 2011).

The self-adaptive SOA is a novel concept in the software architecture domain for which some approaches and frameworks have been defined. The Self-Architecting Software SYstems (SASSY) framework introduced by Malek *et al.* (2009) and Menasce *et al.* (2011) offers a self-architecting framework based on the MAPE-K model, in which a pattern selection method redesigns SOA at runtime to satisfy software quality. MAPE-K is an automation standard model with five components: monitoring, analyzing, planning, executing, and knowledge management (Kephart and Chess, 2003). In other words, pattern composition and pattern selection are useful methods for self-adaptive systems at the architectural level due to the certain structures of patterns. As mentioned in our previous work (Moaven *et al.*, 2009), some straightforward pattern composition methods have been proposed recently.

Besides, software methodology is a framework that takes advantage of software engineering practice to produce software systems (Ramsin and Paige, 2008); e.g., service-oriented modeling and application (SOMA) is a practical methodology with some extension points that allows engineers to append more steps, methods, and tasks to it (Arsanjani *et al.*, 2008). Self-adaptive service-oriented methodologies frequently monitor their contexts (di Nitto *et al.*, 2008) to become aware of their environments (Gu *et al.*, 2005; Papazoglou and Heuvel, 2007). Hence, researchers have been motivated to concentrate on self-aware methods. Yang *et al.* (2008) identified and extracted the SOA context with an ontology-based model, and used a Java expert system to select the appropriate context of SOA. A context-aware architecture was proposed by Gu *et al.* (2005) to produce suitable prototypes of context-aware services. Strang and Linnhoff-Popien (2004) presented an overview of some context-aware methods and a classification of their modeling approaches.

The problem domain is a prerequisite for context recognition as the context is identified, analyzed, and modeled in it. Domain analysis is an activity in which domain information is identified, collected, and presented by studying systems and their histories and acquiring tacit or explicit knowledge of expert users (Kang *et al.*, 1990). Moreover, the knowledge management component, which makes a model of problem domain besides its analyzing and storing, is a major component of automatic systems (Moaven *et al.*, 2008a) such as self-adaptive systems. The significance of this component is motivating researchers and companies to produce some useful knowledge modeling tools, languages, and technologies; e.g., ontology is a practical technology which formalizes the problem domain by OWL language (McGuinness and van Harmelen, 2004) and stores concepts in the repository (Gu *et al.*, 2005; Malek *et al.*, 2009; Menasce *et al.*, 2011). In addition, different tools

have been developed for modeling, analyzing, designing, and representing knowledge; the most important tools may encompass Protege (Noy *et al.*, 2003), NeOn (Haase *et al.*, 2008), and Hozo (Kozaki *et al.*, 2002).

As mentioned earlier, methodologies are among the principal elements of self-adaptive frameworks that determine the relationship between other elements of frameworks. Recently, a few self-adaptive methodologies (SAMs) have been defined independently or based on available self-adaptive frameworks. In spite of their benefits, many of them have ignored the relationship between the software development process (SDP) and the self-adaptive process (SAP), whereas SDP defines the domain of SAP (di Nitto *et al.*, 2008), or they have not been evaluated.

In this paper, we assume that the software architecture is mapped to a composition of architectural patterns; then, a novel pattern-oriented SAP has been proposed based on the MAPE-K model to monitor and analyze SOA quality, and redesign the software architecture whenever its quality is far from the expectations of stakeholders. As the architectural tactic provides a bridge between software architecture and quality attributes (Bachmann *et al.*, 2003), tactic composition is a suitable method to evaluate the quality of software architecture. Hence, the proposed process has customized the architectural patterns according to the architectural tactics to evaluate SOA quality by a proper composition of tactics. In fact, the customized patterns streamline the deployment of architectural tactics on the software architecture. Moreover, a utility function has been defined to measure the quality of software architecture from the quality of its subsystems.

Architectural and adaptive patterns have been applied to the software architecture in order to redesign and deploy the updated software architecture, respectively. Pattern composition has been applied to the planning step of the MAPE-K model to redesign the software architecture. To reconfigure the architecture, a rule engine service has been proposed to convert one pattern to another by predefined rules. Moreover, adaptive patterns execute the aforementioned rules according to the state of patterns mapped to the software architecture.

The inputs and outputs of SAP have been identified explicitly to compensate for the absence of SAP in SDLC, to decrease the aforementioned defects, and to apply the adaptation process to service-oriented SDLCs. Therefore, non- or semi-automatic tasks, such as tactic selection, are committed to SDLC to create an automated SAP.

Furthermore, to provide a self-adaptive SOA methodology (SASOAM), clarify the proposed SAP, and compare SASOAM with SOMA (Arsanjani *et al.*, 2008) based on some evaluation criteria, this paper has combined SAP and SOMA in a two-layered model consisting of the domain and context layers; e.g., although SAP does not suggest any specific approach to produce an appropriate tactic composition, SASOAM describes a practical method that decomposes the general scenarios into concrete ones. Due to the use of general scenarios for evaluating the quality attributes, this method helps architects distinguish more accurate architectural tactics from concrete scenarios for quality attributes.

In this methodology, SAP automatically manages system quality to improve the maintenance activity of SOMA. In addition, to indicate the practicability of the proposed SAP, a practical case study has been implemented with the IBM Rational Software Architecture tool, Feature Modeling Plugin (FMP) (http://gp.uwaterloo.ca/fmp), RBML-PI add-on component (Kim and Whittle, 2005), and some simulation codes.

2 Related work

Siljee *et al.* (2005) divided SAP into the following five activities: (1) monitoring the context and collecting its data, (2) measuring quality attributes from acquired data, (3) evaluating quality of SOA against its desired quality attributes, (4) choosing another software architecture configuration, and (5) deploying the updated architecture. Nevertheless, this process is not defined based on any particular framework or methodology; it is defined in a top-level structure without expressing details of its steps. Furthermore, it is applied to the streaming video service case study to evaluate the applicability of the proposed method, whereas a stronger evaluation is required to prove the adaptability of SAPs.

André *et al.* (2010) introduced an automatic mechanism to adapt the quality of services to their

environmental changes by using the MAPE model (MAPE is similar to the MAPE-K model, but it does not contain the knowledge service). While it can manage adaptation at several levels (e.g., service, application, SOA, or infrastructure), it refuses to consider how to plan an adaptation policy in order to modify current services to achieve the interested services. Bhakti *et al.* (2010) proposed a self-organizing approach which uses the MAPE-K standard model; although it supplies the case base reasoning method to extract the optimum architectural patterns from the knowledge repository, it just focuses on the service detection activity rather than quality assurance.

Cardellini *et al.* (2009; 2012) provided two self-adaptive SOA methods that discover the optimal composition of services with MAPE and MAPE-K models to satisfy the expected quality attributes. These methodologies organize workflows of services in a weighted tree, and introduce a few grammatical patterns to detect the proper composition of services that satisfies the quality attributes of SOA. While they have focused on the service composition activity to satisfy the expected quality of the system, they have avoided specifically explaining the steps of their methodologies.

Malek *et al.* (2009) and Menasce *et al.* (2011) introduced an almost complete self-architecting framework that changes the software architecture by use of pattern selection methods. However, a clear boundary between SAP and SDLC is not determined in this framework, and it contains no specific method to measure quality metrics.

In addition, Andersson *et al.* (2013) divided self-adaptive systems into two parts, on-line and off-line, which correspond to the context and domain of applications, respectively. Although they have defined the point-to-point relationships between in-line and off-line adaptation parts, the definition was not based on the software architecture and they did not focus on quality attributes. Gacek *et al.* (2008) proposed two repetitive processes for self-adaptation systems. The inner process is an adaptation process which adapts systems to their contexts automatically using adaptation rules, and the outer process is an evolution process that provides and updates requirements of the adaptation process. While these processes are defined for a self-managing framework to cover the domain and context of systems, Gacek *et al.* (2008)'s work has the same weaknesses as that of Andersson *et al.* (2013).

Lane *et al.* (2012) proposed an S-cube life cycle of the adaptable service-based applications (SBAs). This model consists of the evolution and adaptation steps. In the evolution step, stakeholders answer some questions to detect the following requirements: (1) What aspects of SBAs are demanded to adapt them to the environment changes? (2) What features of the SBA environment are monitored to evaluate the selected adaptation aspects? (3) How to design, implement, and deploy the adaptation mechanism and monitor services? When SBA is deployed, the adaptation step monitors its context with a monitor service. If adaptation aspects are not satisfied, the adaptation service executes a proper mechanism to improve the application based on the aforementioned aspects. This method provides a high level process to design an adaptable SBA for each adaptation aspect. Therefore, it can support our suggested SAP when quality satisfaction is the goal of adaptation. In fact, our SAP is a special case of this method with more technical details.

de Lemos *et al.* (2013) classified the major challenges of self-adaptive systems in the following categories: design space for adaptation solutions (i.e., deciding how to express the system environment, selecting the adaptation mechanism, and proving the elected mechanisms), processes, change centralized to decentralized control (i.e., using MAPE distribution patterns (Weyns *et al.*, 2013) to distribute adaptation control mechanism among different components), and practical runtime verification and validation. SASOAM addresses each of these challenges by providing pattern- and tactic-based adaptation mechanisms, expressing the steps of the SAP process, taking advantage of the MAPE-K adaptation loop, and representing a quality analysis approach, respectively.

Ardagna and Pernici (2007) suggested a service composition modeling approach in which concrete services are composed automatically based on the abstract business process and quality concerns at runtime. In fact, users design the business processes of applications and declare distinct thresholds for quality attributes; then, the proposed approach monitors the environment of applications to check whether their qualities are satisfied or not. This approach

plans a new composition of services whenever the quality attributes are not satisfied according to the thresholds. Although this approach appropriately supports both the analyzing and planning steps of the MAPE-K model, it neglects the monitoring and executing steps. In fact, it is supposed that qualities of services are measured with another component. Moreover, it assumes that the deployment of new architecture is managed with another component while the details are neglected. Another challenge is utility functions of quality attributes. This paper has provided some utility functions for a few quality attributes. Although these utility functions are true, there is no guarantee that experts can define utility functions for other quality attributes due to their complexities.

Wang *et al.* (2010) provided a quality-based service composition approach in which a fuzzy synthetic evaluation method was applied to the context of the application to evaluate the quality of services. Then they analyzed the importance of service quality based on service providers, system histories, and the context of customers. Finally, a mathematical formula was supplied to measure the quality of the system. Although their approach supports the monitoring, analyzing, and planning steps of the MAPE-K model, it does not describe how to manage the inconsistencies of software architecture deployment at runtime.

3 Self-adaptive service-oriented architecture methodology (SASOAM)

Several self-adaptive frameworks, approaches, and methods have been proposed recently to satisfy the quality of the software architecture. However, they often come across a few defects such as lack of attention to the following options: SAPs and their communications with SDPs, a proper evaluation of self-adaptive methodologies, and an applicable method for measuring the quality of SOA. Moreover, these frameworks usually do not provide a comprehensive self-adaptive methodology for the aforementioned problems.

In this paper, the SAP which is responsible for maintaining the quality of systems is a repetitive and automatic process. To this aim, it provides feedbacks for different phases of SDP according to the communication of context and domain; e.g., changes to the system do not always occur in the scope of context and must be analyzed in the scope of the domain, and then results are transmitted to the context. Therefore, although SAP controls the quality of software architecture from the system context, some functions such as deciding on the type of non-functional requirements, creating business processes, making suitable composition of architectural tactics, changing utility functions, and defining pattern conversion rules and the threshold function are assigned to SDP.

Fig. 1 presents the high-level structure of SASOAM which is divided into two major layers: software development process and self-adaptive process. The communication between these two layers is defined by using architectural and adaptation patterns, architectural tactics, SOA models, quality attributes, the threshold function, pattern conversion rules, and utility functions. Due to the non- or semi-structural properties of these assets, the knowledge management service (KMS) is used as a mediator between these layers to handle the aforementioned problem. As will be described later, KMS maintains and updates context information while providing useful solutions for decision making in the system domain by exploiting previous information from the knowledge repository.

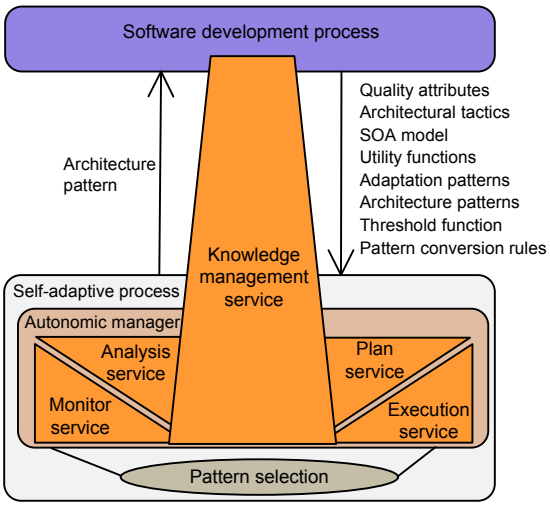

Fig. 1 High-level structure of the self-adaptive service-oriented architecture methodology (SASOAM)

In this paper, we assume that the initial software architecture is mapped to a composition of architectural patterns. This means that subsystems are

covered by a composition of architectural patterns which expedites evaluation and quality assurance of the software architecture, its reconfiguration and transformation of architectural changes to design models and source codes (di Nitto *et al.*, 2008). Architectural patterns are therefore composited and used through SAP when they are extracted from the knowledge repository.

As presented in Fig. 1, SAP uses the MAPE-K standard model in which the monitor service measures the essential criteria for evaluating software architecture quality with respect to architectural patterns, the analysis service analyzes metrics to evaluate the quality of software architecture, and the plan service defines a fresh configuration for the software architecture to improve or preserve system quality. The plan service selects some appropriate architectural patterns and makes an up-to-date pattern composition according to the desired quality attributes of stakeholders in order to transfer changes to the more concrete models such as design and code with the help of the execution service. In this structure, KMS stores, updates, and manages semi- or non-structured assets, such as architectural patterns, to isolate their complexity using encapsulation.

Fig. 2 depicts more details of SAP. As mentioned earlier, we have supposed that the software architecture is made up of a composition of architectural patterns such that each subsystem is applied to a distinct architectural pattern. In the following subsections, the five services of SAP and basic manipulations of SOMA will be briefly discussed.

3.1 Monitor service

Combining architectural tactics and patterns is a useful method to simplify quality measurement. According to the Unified Modeling Language (UML) model in Fig. 3, which illustrates the relationship between inner components of the monitor service and also their relationships with outer components, stakeholders define several public scenarios that cover some quality attributes, in order to measure software architecture quality. Quality attributes are decomposed into accurate and comprehensible

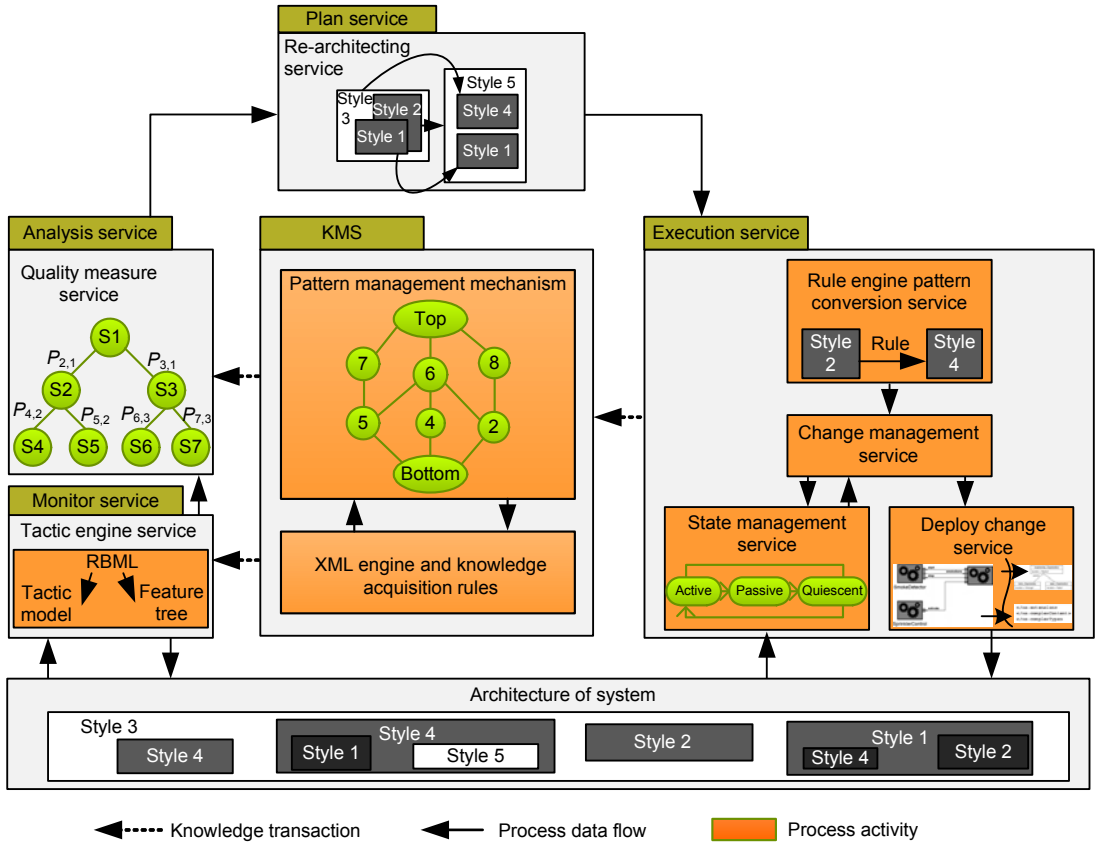

Fig. 2 High-level structure of a self-adaptive process (SAP)

sub-quality attributes which are measured by particular metrics, to facilitate the quantification of quality attributes. On the other hand, architectural tactics are applied to software architectures to measure their quality by computing specific metrics. Scenarios can be measured by the composition of several tactics which are modeled by the Role-Based Modeling Language (RBML). Moreover, architectural tactics have positive or negative influences on each other due to their dependency on quality attributes, which can be determined by feature models separately defined for each quality attribute.

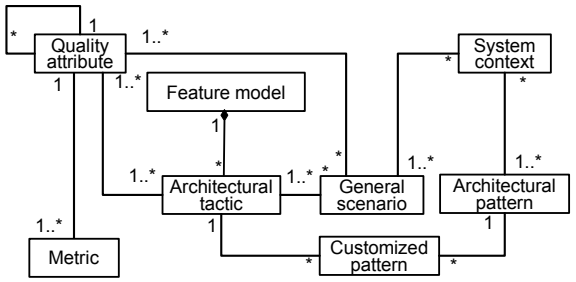

Fig. 3 The relationship among architectural tactics, general scenarios, architectural patterns, and quality attributes

Composing architectural tactics and applying them to software architectures are both time-consuming activities. Due to this problem, architectural tactics are combined with architectural patterns to consider customized patterns. Therefore, architectural tactics are composed according to the patterns deployed in the system to improve the quality evaluation activity.

3.2 Analysis service

In the analysis service, the measured quality of each pattern is placed in a weighted tree structure which quantifies a specific quality attribute in its root. As mentioned before, it is assumed that each subsystem is covered by either an architectural pattern or a composition of architectural patterns such that nested and aligned relations of subsystems can be represented in a quality measure tree similar to Fig. 4. While we have supposed that architectural styles and patterns refer to the same concept, this tree exhibits the relationships between architectural styles that have been introduced in our previous work (Moaven *et al.*, 2009). However, in the aforementioned work the following four types of relationship between architectural styles have been defined: parallel, sequential, embedded, and hybrid. This paper has supposed that patterns specify how the components communicate with each other. Concrete subsystems are placed in leaves of the tree, where the higher level nodes compose them by architectural styles to make a higher level subsystem. Moreover, the corresponding weights of each edge, which represent the correlation coefficient between architectural styles and their components, are determined by votes of stakeholders.

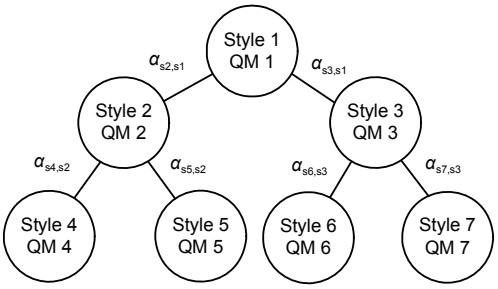

Fig. 4 Quality measurement tree

Due to the aforementioned issues, the measured quality is placed in its corresponding node to compute the quality of each subsystem using the following equation:

$$\mathrm{QM}_{\rho}[\theta]=\begin{cases}\sum\limits_{\forall c\in\{\text{children of }\rho\}}\mathrm{QM}_{c}[\theta]\cdot\alpha_{c,\rho}[\theta], \\ \qquad \rho\notin\{\text{leafs}\},\theta\in\{\text{quality attributes}\}, \\ T(\rho)\cdot W_{\rho}[\theta], \\ \qquad \rho\in\{\text{leafs}\},\theta\in\{\text{quality attributes}\},\end{cases} \tag{1}$$

where

$$\begin{cases}\sum\limits_{\forall c\in\{\text{concrete components}\}}W_{c}[\theta]=1,\ \theta\in\{\text{quality attributes}\}, \\ \sum\limits_{\forall c\in\{\text{components of }\rho\}}\alpha_{c,\rho}[\theta]=1, \\ \qquad \theta\in\{\text{quality attributes}\},\rho\in\{\text{patterns}\}, \\ T(\rho)\le\gamma,\quad \rho\in\{\text{patterns}\}.\end{cases} \tag{2}$$

Herein quality measure (QM) variables, which are measured according to the quality of subsystems and the correlation coefficient weights, indicate the

quality of each subsystem. $\alpha_{c,\rho}[\theta]$ represents the influence of the component c of architectural pattern ρ upon the quality attribute θ of this pattern, and $T(\rho)$ is the integrated value of metrics for a specific quality attribute which is evaluated from the architectural tactic for pattern ρ. γ is a constant value which restricts the quantification of quality attributes to a distinct range to normalize $T(\rho)$ and QM[θ] values between 0 and the upper bound. $W_\rho[\theta]$ indicates the significance of ρ concrete subsystems on the θ quality attribute of the system.

Now there are multiple quality attributes which measure the value of system quality by using a utility function. This function catches the quality attribute values of the system in the root of the quality measurement tree as input parameters to compute system quality. To evaluate software architecture quality, the root value should be compared to the threshold function which shows the least stakeholder satisfaction with system quality. If this quality is less than the aforementioned threshold, the software architecture should be modified again.

3.3 Plan service

During the design phase, different architectural patterns are chosen and combined by means of optimal algorithms such as evolutionary algorithms with regard to the desired quality attributes. This phase is decomposed into two parts, selecting patterns from the pattern storage and combining them together. Pattern combination is a difficult task because of the relationship between quality attributes and the way they are combined. Due to this complexity, in our previous work (Moaven *et al.*, 2008b), a decision support system was proposed to select architectural patterns by taking advantage of fuzzy logic. Because of the high complexity of this phase, it is usually implemented using a framework such as SASSY that takes advantage of knowledge repositories and search algorithms (Menasce *et al.*, 2011).

3.4 Execution service

The goal of the execution service is propagation of the modifications from the software architecture to the source code by the following services.

3.4.1 Role engine pattern conversion service

During the plan phase, the combination of architectural patterns is converted into a new combination in which some architectural patterns are turned into other ones. The rule engine pattern conversion service uses the rules defined by domain experts and stored in the knowledge repository to convert architectural patterns. Due to the definition of software architecture (Bass *et al.*, 1998), which is a set of components, their relationships, and the external properties that each component shares, these rules consider the elimination, addition, and manipulation of components and connectors of architectural patterns. Based on these rules, adding a certain component does not imply the addition of functional requirements to the system; instead, it includes appending common components of architectural patterns such as adding the same proxy component to the broker and client-dispatcher-server patterns.

3.4.2 Change management service

Modifying the configuration of the running system without considering its status may result in poor consequences; e.g., if parts of the running system are covered with a client-server architectural pattern, and the client is waiting to receive a response from the server, removing the server not only reduces the availability of the system, but also leads to the client request being rejected for consistency. To prevent such problems, the configuration should be altered using the change management service (CMS) according to the status of subsystems mapped to the patterns and also modifications that have been defined by the rule engine pattern conversion service. CMS tries to modify the system configuration independently using specific rules to manage them more easily (Kramer and Magee, 1990).

3.4.3 State management service

As mentioned before, SAP requires more attention be paid to the system configuration status in order to reconfigure the software architecture. As subsystems are covered with architectural patterns, the specification of the pattern status involves enough information to recognize the system status. To issue commands about the system status or configuration modification based on management rules, adaptation patterns declare the status of subsystems to CMS as passive, active, or quiescent as depicted in Fig. 5 (Gomaa *et al.*, 2010; Gomaa, 2011). These rules include commands which either change system status or prepare subsystems for reconfiguration.

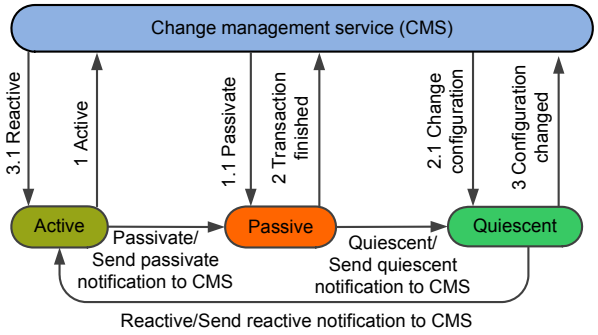

Fig. 5 Structure of the state management service

3.4.4 Deploy change service

To apply architecture-level changes to the source code, CMS exports appropriate instructions of minor configuration modification to the deploy change service. Changes may be either so simple that altering the operation of the service controller components would be enough, or so difficult that service interface modification would be required. Finally, a new system will be produced in which modifications are applied to or deployed on the running system. To this aim, Matinlassi (2005) suggested a model transformation method to convert architectural patterns to other ones. They have proposed two types of automatic transformation method, style-based and rule-based. The rule-based method uses transformation rules to convert the architectural pattern *A* (source model) to pattern *B* (target model). This method improves CMS to describe modifications of the software architecture with predefined transformation rules when the plan service determines source and target models. Major transformation rules include reusing the source model elements in the target model and creating new components or connectors in the target model. When CMS modifies the architectural model by means of transformation rules, MDA approaches broadcast architectural changes to the source codes.

Fig. 6 shows the process of SAP in detail, clarifying the relationship between SAP and SDP. As depicted, SAP is an infinite process in which quality attributes, composition of tactics and the SOA model are retrieved from the knowledge repository in the beginning, based on a request in the maintenance activity of SDP.

Because an architectural tactic cannot often measure the quality of general scenarios alone, a combination of architectural tactics is commonly used. Since available resources of architectural tactics are mainly textual documents, automatic detection of their compositions is a big challenge. Therefore, domain experts mine general scenarios and identify the most appropriate composition of architectural tactics in SDP; then, the relationship between quality attributes is assessed by a combination of architectural tactics simultaneously.

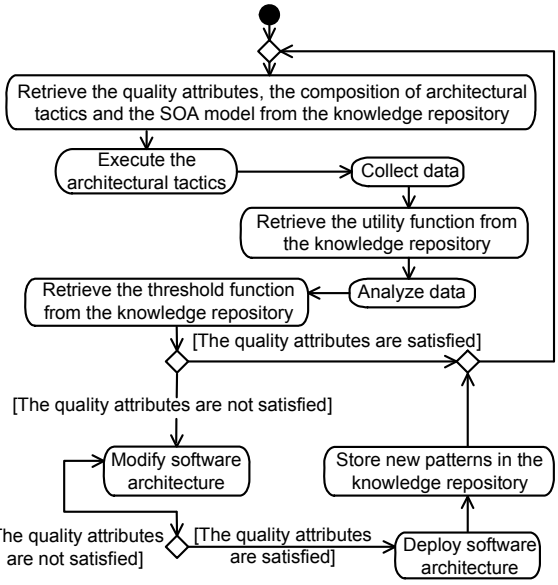

Fig. 6 Diagram of the self-adaptive process (SAP) activity

As presented in Fig. 2, adaptation is working on SOA, which is created by SDP and stored in the knowledge repository. Although various models of SOA have been defined, it is supposed that SOA is a top-down three-layered model, which at least consists of the business process, services, their relations, and foundation layers. In this regard, changes are made at the middle layer and transferred to the lowest layer.

Fig. 6 depicts architectural tactics that are identified by the 'execute architectural tactics' activity. Thus, a hybrid model of tactics is created and then mounted on the target subsystem to measure system quality. Since architectural tactics are modeled with the use of component and sequence diagrams, they can be merged together (Kim *et al.*, 2009). According to Fig. 4, in the 'collect data' activity, the quality measurement tree is created each time the qualities of subsystems are evaluated.

The 'analyze data' activity calculates the root of this tree by a utility function. The root is a vector consisting of values of quality attributes, which should be converted to a single value that represents the total quality of the system using the utility function defined by an expert user. This function is usually a linear function of quantitative attributes conformed to the following equation in which coefficients β_i are used to weight quality attributes x_i:

$$u(x_1, x_2, ..., x_n) = \left\{ \sum_{i=1}^{n} \beta_i x_i \mid 0 \le \beta_i \le 1,\ 0 \le x_i \le \gamma \right\}. \quad (3)$$

As mentioned in Eqs. (1) and (2), x_i is lower than the upper bound which is a constant value, whereas β_i is normalized in [0, 1]. The utility function is stored in the knowledge repository in the SDP layer and retrieved from the knowledge repository before the beginning of the 'analyze data' activity. The result of Eq. (3) is compared with a threshold function stored in the knowledge repository. This numerical function is usually defined by expert users or architects to represent the minimum stakeholder expectation from system quality. The architecture is redesigned until the result of the utility function satisfies the requirements of stakeholders. Since architectural patterns are composed to optimize the quality attributes of the software architecture, pattern composition is defined as a redesigning method to merge architecture modification with evaluation activities.

Architectural changes are broadcast automatically to lower level models such as design and even source code because the composed pattern represents a top-level abstraction of the system. This pattern is a valuable asset of SAP, which is used not only in the next iteration of SAP but also in SDP as mentioned in Section 3.6. Therefore, it is imported to the knowledge repository to update the collection of patterns.

3.5 Knowledge management service

KMS consists of repositories that store patterns, utility functions, and features of interested quality attributes. Therefore, KMS requires both data storing and data restoring mechanisms which support the aforementioned assets. Due to the runtime characteristics of self-adaptive systems and the variation of quality attributes, these mechanisms must provide the following capabilities:

Extendibility: As mentioned above, the architectural patterns are experimental solutions for distinct problems. In fact, the new architectural patterns are developed when they are frequently detected in the software architecture of an application. Moreover, various quality attributes are measured with different architectural tactics. Therefore, the storage mechanism must be sufficiently extendable to support new compositions of architectural tactics and patterns.

Ability to quickly store and acquire data: Knowledge acquisition, which aims to explore the required information from the data properties, is a significant activity of SAP. Due to the complexity of data relations, knowledge acquisition may take a long time to fetch proper information from the knowledge repository, whereas SAP is a real-time process. Hence, KMS must provide a real-time knowledge acquisition method.

Formal description: Formal languages provide a standard description of applications that improve the development of KMS activities. In other words, the formal description of knowledge management mechanisms is a standard document to implement KMS independent of the development platform. It also streamlines the validation and verification of KMS mechanisms.

As Fig. 2 depicts, KMS consists of an eXtensible Markup Language (XML) engine and pattern management mechanism. XML is a formal language to describe the properties of software assets and their relationships. XML is constructed by three major parts, including elements (i.e., logical components of the system), their properties and values. Moreover, an XML element can be defined as child, parent, or sibling of other elements (Castro, 2000). Therefore, properties and nested definition of elements provide suitable approaches for classification of XML elements. Although XML produces a textual document, a logical tree model is suggested to manipulate XML elements and query the XML documents due to the nested structure of XML elements; e.g., QDomDocument (http://qt-project.org/doc/qt-4.8/qdomdocument.html#details), which stores the XML documents in a logical tree model, is a library of the Qt project. In addition, it has developed some functions to add, remove, or update the XML content.

The XML engine organizes information of the aforementioned assets in an XML document. In fact, patterns, utility functions, and quality attributes constitute elements of the XML document while their characteristics form properties of elements; e.g., stability is an instance of quality attributes represented by an XML element. Based on the ISO/IEC-9126 quality model (Fitzpatrick, 1996), stability is a subquality of the maintainability quality attribute. To represent this dependency, the type of quality attributes is added to the properties of quality elements. Thus, maintainability will be the value of this property of the stability quality attribute. Moreover, elements inherit external properties from their nested elements; e.g., architectural tactics are represented by an XML element in which some properties such as evaluation metrics are defined. To compose the architectural tactic and pattern, the tactic element is added to the children list of the pattern element. As a result, tactic elements append external properties to the pattern elements. They determine how to evaluate architectural patterns, what metrics are required to measure the quality of patterns, and how to assign pattern components to the architectural components in order to evaluate the quality of the system.

The XML engine provides a well-defined structure to organize SAP information. Moreover, it supplies suitable mechanisms to manipulate the XML document at runtime. When the XML document is generated by the XML engine, some XML-based rules are supplied to this document to export proper information. Even though many XML rule generator mechanisms exist, a regular expression is a suitable method to query XML databases for useful information due to the grammatical structure of XML documents.

As mentioned, quality management of the composition of architectural patterns is a significant activity of SAP when the software architecture is produced by a pattern-oriented approach. Although the plan service makes an appropriate composition of the architectural pattern, KMS is a critical service to inform the plan service about the relationship between patterns and quality attributes. The following options are the other minimum services that KMS must provide:

1. Specify the impact of quality of architectural patterns on each other.

2. Improve the plan service on detecting the proper composition of patterns.

3. Detect various suitable pattern compositions besides the best one.

In this paper, the concept analysis method proposed by Arevalo *et al.* (2004) has been customized to enhance KMS. This method supplies a formal concept analysis (FCA) approach to classify architectural patterns based on quality attributes. FCA takes advantage of lattice theory to categorize elements according to their common properties. Context and concept are two fundamental issues of FCA (Ganter and Wille, 1999). As Table 1 shows, a set of elements, their properties, and their binary relationships define the FCA context. Besides, FCA concepts are common properties among context elements.

The proposed method tracks the following steps to make the FCA model: (1) generate the FCA context, (2) make FCA concepts from the FCA context, and (3) produce the FCA lattice model from FCA concepts. The first step demands to know the primary quality attributes of the FCA context and the order of context. The order of context specifies the length of context elements; e.g., the length of context elements in Table 1 is three, and thus its order is three.

Table 1 shows the order 3 context of architectural patterns which are used in a hypothetical architecture. Although this table exhibits the order 3 context, designers can plan other orders of the context according to the patterns used in the software architecture. It is supposed that availability and performance are critical quality attributes of this architecture. The FCA context must export the three-member composition of patterns, which is detected in the software architecture. Moreover, the binary relationships between availability and performance of each pair of three-member elements must be recognized from the software architecture; e.g., the following relationships are defined for this example:

1. $(\alpha, \beta)_{GA}$: the αth element has greater availability than the βth element.

2. $(\alpha, \beta)_{EA}$: the availability of the αth element is equal to that of the βth element.

3. $(\alpha, \beta)_{GP}$: the αth element has greater performance than the βth element.

4. $(\alpha, \beta)_{EP}$: the performance of the αth element is equal to that of the βth element.

Table 1 Order 3 context for the composition of architectural patterns

	$(1, 2)_{GA}$	$(2, 3)_{GA}$	$(1, 3)_{EA}$	$(2, 3)_{EA}$	$(3, 2)_{GP}$	$(2, 1)_{GP}$	$(2, 3)_{EP}$	$(3, 1)_{EP}$	$(2, 3)_{GP}$
$\{P_1, P_2, P_3\}$	*		*				*	*	
$\{P_1, P_4, P_5\}$	*	*							
$\{P_5, P_1, P_3\}$	*		*			*		*	*
$\{P_3, P_7, P_6\}$		*	*				*	*	
$\{P_3, P_2, P_4\}$	*		*					*	*
$\{P_1, P_2, P_7\}$		*			*			*	

Step 2 classifies the elements of the FCA context to several groups according to their common properties. As mentioned, these groups make the FCA concepts. Table 2 presents the FCA concepts of the architecture, which are generated from the FCA context.

Table 2 Concepts of the composition of architectural patterns

Top	(All elements G, ϕ)
8	$(\{\{P_3, P_2, P_4\}, \{P_1, P_4, P_5\}, \{P_5, P_1, P_3\}, \{P_1, P_2, P_3\}\}, \{(1, 2)_{GA}\})$
7	$(\{\{P_1, P_2, P_7\}, \{P_3, P_7, P_6\}, \{P_1, P_4, P_5\}\}, \{(2, 3)_{GA}\})$
6	$(\{\{P_1, P_2, P_7\}, \{P_3, P_2, P_4\}, \{P_1, P_2, P_3\}, \{P_5, P_1, P_3\}, \{P_3, P_7, P_6\}\}, \{(3, 1)_{EP}\})$
5	$(\{\{P_3, P_7, P_6\}, \{P_1, P_2, P_7\}\}, \{(2, 3)_{GA}, (3, 1)_{EP}\})$
4	$(\{\{P_3, P_7, P_6\}, \{P_5, P_1, P_3\}, \{P_1, P_2, P_3\}, \{P_3, P_2, P_4\}\}, \{(1, 3)_{EA}, (3, 1)_{EP}\})$
3	$(\{\{P_3, P_7, P_6\}, \{P_1, P_2, P_3\}\}, \{(1, 3)_{EA}, (2, 3)_{EP}, (3, 1)_{EP}\})$
2	$(\{\{P_5, P_1, P_3\}, \{P_1, P_2, P_3\}, \{P_3, P_2, P_4\}\}, \{(1, 3)_{EA}, (1, 2)_{GA}, (3, 1)_{EP}\})$
1	$(\{\{P_3, P_2, P_4\}, \{P_5, P_1, P_3\}\}, \{(3, 1)_{EP}, (2, 3)_{GP}, (1, 2)_{GA}, (1, 3)_{EA}\})$
Bottom	(ϕ, all properties M)

The FCA lattice is constructed from the FCA concepts, where its nodes correspond to FCA concepts and its edges signify that one FCA concept covers all elements of the other concept. In other words, lattice edges connect two concepts together. These downward edges show that the bottom concepts are subsets of top concepts. As Fig. 7 illustrates, the FCA lattice includes the aforementioned minimum services for KMS.

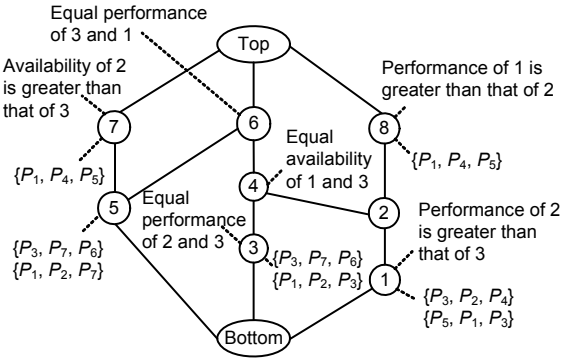

Fig. 7 Lattice of FCA concepts

3.6 Customization of SOMA methodology

In this paper, SAP is integrated with SOMA and then this extension is evaluated by some service-oriented methodology criteria to show that SAP improves SOMA. This section reviews SOMA and after that some of its activities are customized to handle SAP. As Fig. 8 depicts, SOMA is a repetitive process with six phases and 21 activities. In summary, the business modeling and transformation phase decides on policies, strategies, and starting of the project; identification, specification, and realization phases explore, create, and refine the required services respectively; the implementation and deployment, monitoring, and management phases implement, test, deploy, and maintain services, respectively (Arsanjani *et al.*, 2008).

As initialization, the essential patterns (including architectural and adaptation patterns) are imported into the knowledge repository. Thus, activity 2 is customized to assign a pattern storing task to activity 4 by selecting a suitable solution template because activity 2 defines variation points of the SOA solution and method content (Arsanjani *et al.*, 2008). Also, the selected solution template addresses the next changes to SOMA, which will be described in this section. Then activity 4 models and stores utility functions, composition of tactics, and the threshold function in the knowledge repository because they are useful assets in SAP for monitoring, analyzing, and evaluating the quality of SOA.

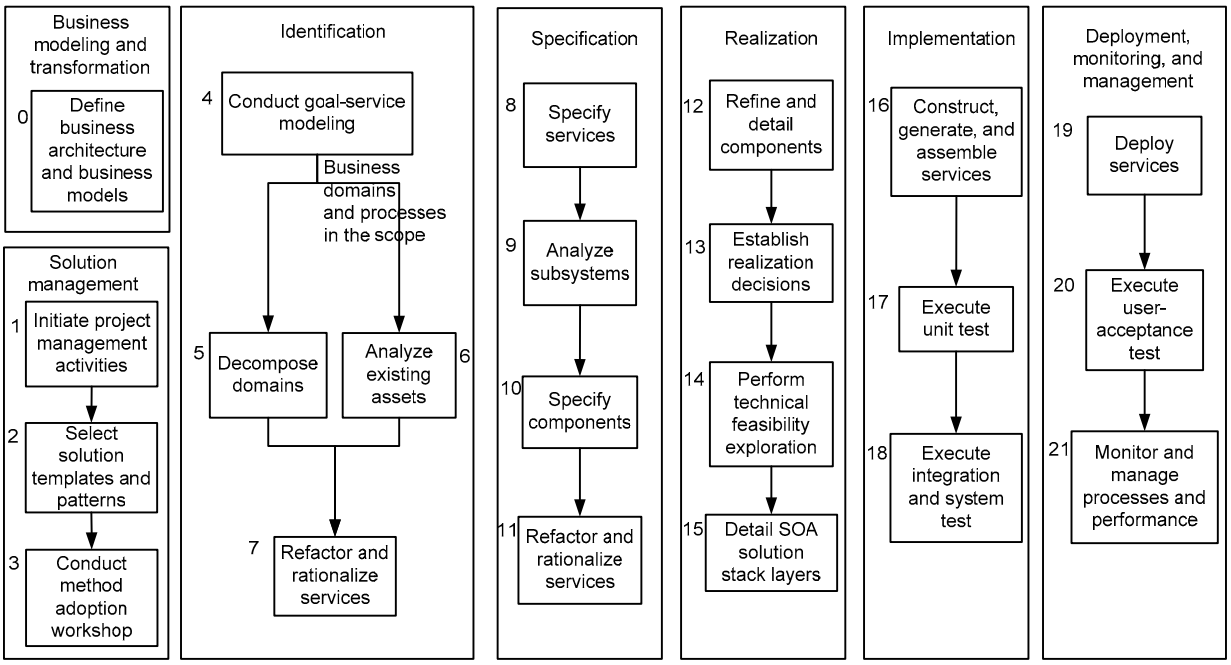

Fig. 8 Steps of the service-oriented modeling and application (SOMA) (Arsanjani *et al.*, 2008)

We believe that the goal-service model (GSM) is a suitable model to integrate all of them and define their relationships with quality attributes. As Table 3 shows, the GSM template has four parameters:

1. Goals and subgoals: Introduce required quality attributes and provide some general scenarios based on system context to quantify the corresponding quality attributes (Kazman *et al.*, 2000; Bass and John, 2003; Babar, 2004).

2. Key performance indicators (KPIs): Represent the relationship between the utility function and the quality attribute threshold. The utility function is an objective function, which can be either a simple mathematical formula or a complex model. Note that any quality attribute has an analytical model extracted from scenarios (Kazman *et al.*, 2000).

3. Metrics: Compose some architectural tactics to cover a specific KPI. Due to the side effects of tactics, Kim *et al.* (2009) arranged some tactics in a feature model and defined the relationship between tactics. Bachmann *et al.* (2002; 2007) and Scott and Kazman (2009) introduced major tactics for some quality attributes.

4. Services: Customize architectural patterns based on tactics introduced in the third column by adding some operations. To this aim, in Harrison and Avgeriou (2010a; 2010b), the relationship between architectural patterns and tactics was defined and components and connectors were modified according to the manipulations of patterns. The activity diagram of the 'conduct goal-service modeling' phase is presented in Fig. 9.

Table 3 Template of GSM for quality attributes

Goal and subgoal	KPI	Metric	Services
Quality attribute (goal) General scenario (subgoal)	Relationship between the utility function and the threshold function	Architectural tactics composition	1. Composing architectural tactics based on architectural patterns 2. Modifying architectural patterns based on the selected architectural tactics

KPI: key performance indicator

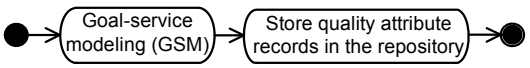

Fig. 9 Diagram of the 'conduct GSM' activity

The communicational messages between services are suitable assets that enhance the 'collect data' and 'deploy architecture' activities to both

execute scenarios on and apply architectural patterns to concrete services. Some modeling methods and standards exist to store these assets; e.g., the Web Service Definition Language (WSDL) is a formal language that models external properties of services, the XML Schema Definition (XSD) is an XML format to model message types, and the service model represents the relationship between services. These assets are produced by activity 11 of Fig. 8, whereas the realization phase appends additional information such as component details to services; thus, the specification phase activity diagram is as shown in Fig. 10.

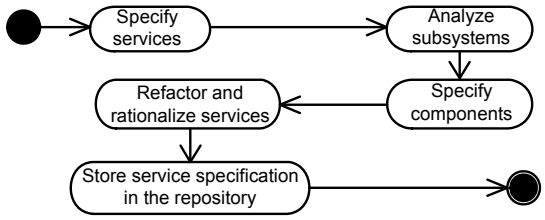

Fig. 10 Diagram of the 'specification phase' activity

The 'specify services' activity demands the predefined and runtime-explored architectural patterns, which are stored in the knowledge repository, to compose services based on the quality attributes. Therefore, patterns are retrieved from the knowledge repository to define a service composition roadmap. Fig. 11 depicts an extension of the 'specify services' activity diagram introduced by Fuhr *et al.* (2013).

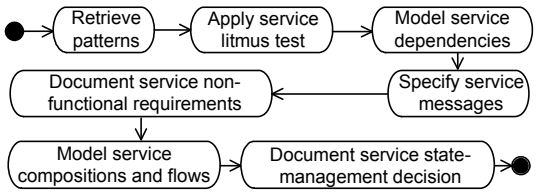

Fig. 11 Diagram of the 'specify service' activity

4 SASOAM evaluation

We have claimed that the maintenance activity of SDLC is improved by SAP, where SAP controls the quality of SOA at runtime by modification of architectural models. Therefore, the maintainability is a major criterion for evaluation of the proposed self-adaptive methodology. As mentioned above, the evaluation of maintainability is a difficult activity due to its vague concept. To this aim, quality models decompose the maintainability quality attribute into several sub-quality attributes; e.g., McCall and ISO/IEC-9126 models decompose maintainability into four and five sub-quality attributes in a split-level respectively, whereas the Boehm model decomposes it into nine attributes in two-level divisions (Dromey, 1995; Fitzpatrick, 1996; Rawashdeh and Matalkah, 2006).

We have developed the Travel Planning e-Service (TPS) project with SOMA and SASOAM methodologies. To recognize the impact of these methodologies on the maintainability of TPS, an analytic hierarchy process (AHP) based method is applied to the Boehm quality model. AHP is a general problem-solving method that is useful for making complex decisions (e.g., multi-criterion decisions) based on variables that suffer from inaccurate numerical results (Saaty, 1980). Hence, this method can select more maintainable methodologies based on the sub-quality attributes of maintainability.

4.1 Travel Planning e-Service

TPS is a travel arrangement SOS, which includes some independent services. End-users can log in this system to order their requests. When TPS receives a request, it either implements an atomic service, or combines the available atomic services to respond to the end-user request based on his/her role; e.g., when tourists and researchers order the same request, TPS composes different services to respond to the tourist request rather than the researcher request (Hull *et al*., 2005). The purpose of TPS is to generate a suitable composition of atomic services to satisfy the functional and non-functional requirements of the end-user. Even though the generated service realizes a generic requirement of an end-user, providing a reusable service that addresses the requirements of multiple end-users is the primary goal of TPS.

We have developed TPS with SOMA and SASOAM to evaluate the maintainability of these solutions. Although a functional requirement can be released by different structures of a service composition, their configurations affect the quality of SOS. In this regard, we expect that SAP would enhance the maintainability of SOMA methodology.

4.2 Evaluation process

A criteria-based evaluation method has been applied to the TPS project to compare SASOAM with SOMA. It gathers a collection of maintenance criteria from the Boehm quality model and primary scenarios of the TPS project. The maintainability quality attribute is decomposed into testability, understandability, and modifiability attributes. Boehm *et al.* (1976) believed that these quality attributes should be divided into sub-quality attributes due to their high-level definitions. In fact, their definitions do not provide deterministic concepts for stakeholders. The aforementioned attributes are decomposed into the following sub-quality attributes (Fig. 12):

1. Accountability: Usage of assets can be measured (e.g., determine what specific portion of components and source codes are exercised).

2. Communicativeness: The software has been enhanced by some concepts, symbols, or structures, which facilitate testers to understand its inputs and outputs (e.g., inputs and outputs of components and functions have well-defined structures).

3. Self-descriptiveness: Provide sufficient information of software assets such as software components, their inputs and outputs, constraints, and the objective.

4. Structuredness: Interdependent parts of the application are organized by deterministic patterns. In other words, the application evolves by structural models and in systematic manners.

5. Consistency: Provide uniform notations, symbols, protocols, and terminologies in SDLC (e.g., similarity between function headers and their definitions). Moreover, the development process improves the traceability of content to requirements.

6. Conciseness: Neglect excessive information (e.g., refuse to develop redundant modules).

7. Readability: Modules, components, and functions, which improve the readability of functional requirements, are well defined.

8. Augmentability: Facilitate the expansion of functional and data storage requirements (Boehm *et al.*, 1976).

Before applying the AHP method to the TPS project, the importance of one quality attribute must be determined against the other one which is placed in the same level of the hierarchical model. In addition, the impact of quality attributes on their parents is a primary requirement of the AHP method. Due to the impossibility of quantification of some quality attributes, the experience of expert users is a helpful source to measure the aforementioned weights.

Therefore, some questions were delivered to five expert users with the architect, designer, developer, analyzer, and end-user roles. We assigned weights of 4, 3, 2, 5, and 1 to the aforementioned roles respectively based on their experience. As Table 4 shows, a few questions have been planned by expert users according to the requirements of the TPS project and the aforementioned sub-quality attributes. Although three questions are selected for sub-quality attributes, the structuredness attribute is evaluated with five questions due to its usage for the testability, understandability, and modifiability attributes. As each question corresponds to one scenario, it is recognized by a scenario number. When the questionnaire forms of Table 4 were filled by the

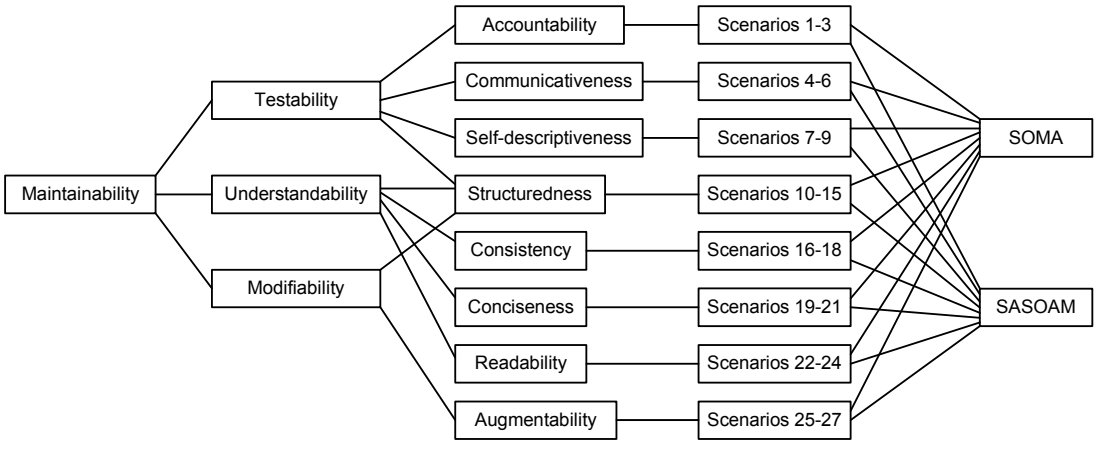

Fig. 12 Maintainability of the Boehm quality model

aforementioned users, their votes were integrated in Table 5 by the arithmetic mean formula.

As Table 5 shows, stakeholders assign a number between −4 and +4 for each pair of scenarios to represent the importance of the source scenario (S-scenario) against the destination scenario (D-scenario). In other words, this score exhibits the significance of each scenario in Table 4, named the S-scenario, against the other scenario in this table, named the D-scenario; then the higher number indicates that the S-scenario is more important than the D-scenario.

In addition, stakeholders voted for the importance of each pair of sub-quality attribute and quality attribute. As shown in Tables 6 and 7, the votes of stakeholders were integrated with the arithmetic mean formula to compare the importance of source attributes (S-attributes) against the destination attributes (D-attributes).

Table 4 Concrete scenarios of maintainability quality attributes

Attribute	Scenario
Accountability	S1: How many times are the atomic services used according to the requirements of stakeholders?
	S2: Can it predict the fault-tolerance, availability, or performance of the service composition?
	S3: How many tools are used to facilitate service composition?
Communicativeness	S4: Do interfaces of services provide a well-defined, similar, and structural definition of functions and variables?
	S5: Are there proper description of inputs and outputs of legacy services, where their structures, symbols, or definitions are different?
	S6: Is there a standard or pattern for specifying inputs and outputs of a composition of services?
Self-descriptiveness	S7: Are functions of services defined so proper that service testing would be streamlined?
	S8: Does it provide a specific structure to inform the impact of a service composition on the software quality?
	S9: Does it specify the dependency of services by means of commentary, documents, or program structures?
Structuredness	S10: Are service compositions transformed to others at runtime by specific models?
	S11: Does it use standard structures to organize SOA?
	S12: Is there a specific model to compose services based on the quality attributes?
	S13: Does it compose services with well-defined structures?
	S14: Are documents of services and their communications created during the development process?
	S15: Is there a suitable structure to add new services to SOA at runtime?
Consistency	S16: Does it store details of non-functional requirements in the repository with specific structures (e.g., metrics, evaluation method, and threshold)?
	S17: Are services described by a standard formal language?
	S18: Does it measure the impact of a service composition on the quality of others when they share some services?
Conciseness	S19: Does it compose services to satisfy some non-functional requirements of stakeholders simultaneously?
	S20: Does it provide the reusable composition of services to reply to the requests of users with different roles?
	S21: Does it reduce some services from a service composition while it still satisfies the quality attributes?
Readability	S22: Does it describe SOA, fundamental features of services, and other information that improve the readability of the software architecture with a formal language?
	S23: Does it provide an approach to describe functional and non-functional purposes of service compositions (e.g., using architectural pattern or categorizing services)?
	S24: Is there an approach to document the comparison of service compositions based on the quality attributes?
Augmentability	S25: Does it describe the SOA manipulation with a formal language?
	S26: Does it suggest an approach to insert new atomic services to the SOA at runtime?
	S27: Can it define intermediate functions to control quality attributes when SOA is changed at runtime?

Table 5 Maintainability questionnaire form

	Conciseness			Consistency			Self-descriptiveness			Communicativeness			Accountability		
S-scenario	S20	S19	S19	S17	S16	S16	S8	S7	S7	S5	S4	S4	S2	S1	S1
D-scenario	S21	S21	S20	S18	S18	S17	S9	S9	S8	S6	S6	S5	S3	S3	S2
Importance	−3	−3	0	+1	+3	+2	−1	+3	−2	−3	+3	+2	−2	+2	+3
Structuredness-testability															
S-scenario	S14	S13	S13	S12	S12	S12	S11	S11	S11	S11	S10	S10	S10	S10	S10
D-scenario	S15	S15	S14	S15	S14	S13	S15	S14	S13	S12	S15	S14	S13	S12	S11
Importance	+1	+3	+2	+2	+1	−1	+4	+4	+3	+4	+4	+3	+1	+2	−2
Structuredness-understandability															
S-scenario	S14	S13	S13	S12	S12	S12	S11	S11	S11	S11	S10	S10	S10	S10	S10
D-scenario	S15	S15	S14	S15	S14	S13	S15	S14	S13	S12	S15	S14	S13	S12	S11
Importance	+3	+4	+1	+1	−3	−4	+3	−1	−2	+2	+2	−2	−3	+1	−2
Structuredness-modifiability															
S-scenario	S14	S13	S13	S12	S12	S12	S11	S11	S11	S11	S10	S10	S10	S10	S10
D-scenario	S15	S15	S14	S15	S14	S13	S15	S14	S13	S12	S15	S14	S13	S12	S11
Importance	−4	−2	+2	−1	+2	+1	−4	+1	−2	−3	−3	+2	−1	+2	+1

	Augmentability			Readability		
S-scenario	S26	S25	S25	S23	S22	S22
D-scenario	S27	S27	S26	S24	S24	S23
Importance	+1	−2	−3	−1	+2	+3

Table 6 Importance of different pairs of sub-quality attributes (S-attribute and D-attribute)

D-attribute	Importance					
	Accountability	Communicativeness	Self-descriptiveness	Structuredness	Consistency	Conciseness
Communicativeness	+4					
Self-descriptiveness	+3	−1				
Structuredness	+1	−3	−2			
Augmentability				+2		
Consistency				+2		
Conciseness				+3	+1	
Readability				+1	−1	−2

As Fig. 12 depicts, the AHP method decides on the importance of SASOAM against SOMA. To this aim, AHP demands to know how concretely the scenarios are satisfied by the aforementioned methodologies. When stakeholders answered this question, their votes were merged similar to the previous questionnaires (Table 8).

This data set was imported to the expert choice application to select a more maintainable methodology. As Fig. 13 shows, modifiability is the most effective attribute on the maintainability of methodologies, whereas understandability is the least effective. Moreover, this figure depicts that SASOAM is more maintainable than SOMA.

Table 7 Importance of different pairs of quality attributes (S-attribute and D-attribute)

D-attribute	Importance	
	Testability	Understandability
Understandability	+1	
Modifiability	−1	−2

Fig. 14 shows that the understandability, modifiability, and testability of SASOAM are more than their counterparts of SOMA. Among these criteria SASOAM satisfies understandability best, whereas this criterion is the least significant in SOMA. Thus, the growth of the understandability weight makes SASOAM more maintainable than SOMA.

Table 8 Satisfaction degree of scenarios in SOMA against SASOAM

Scenario	Importance	Scenario	Importance	Scenario	Importance
S1	0	S10	−3	S19	−1
S2	−2	S11	0	S20	−1
S3	−1	S12	−2	S21	−3
S4	0	S13	−2	S22	−1
S5	−1	S14	0	S23	−2
S6	−1	S15	0	S24	−3
S7	0	S16	−1	S25	−2
S8	−2	S17	0	S26	0
S9	−1	S18	−3	S27	−3

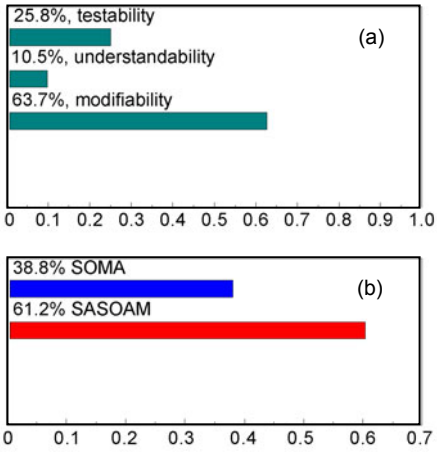

Fig. 13 Dynamic analysis of SOMA and SASOAM
(a) Sensitivity of maintainability to testability, understandability, and modifiability attributes; (b) Maintainability of SOMA and SASOAM

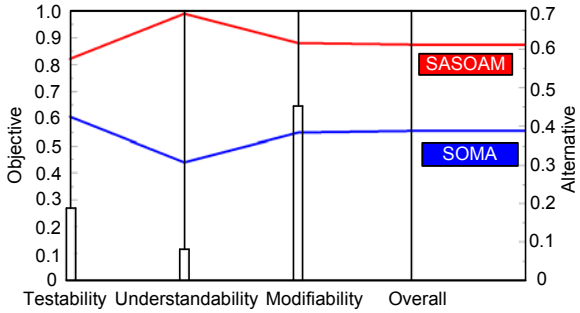

Fig. 14 Performance analysis of SOMA and SASOAM

4.3 Analysis of evaluation results

The results demonstrate that SASOAM is more maintainable than SOMA. Since the Boehm model is a standard quality model, these results are dependent on the selected scenarios. As Table 4 shows, these scenarios focus on the use of the following options in SOMA and SASOAM: formal description, model-based methods, change management at runtime, service composition, and quality management. SOMA supports these options by the WSDL formal language, business process model, rule-based methods, architectural patterns, and experience of expert users. However, SAP enhances SASOAM by adding the following options:

1. Using an XML-based formal language to store primary information in the repository.

2. Using a pattern-based approach to analyze, manipulate, and transform service compositions.

3. Providing a rule-based pattern transformation method and adaptive patterns.

4. Planning suitable service compositions based on the quality attributes by style selection methods.

5. Providing a tactic-based method to analyze both quality attributes and their impacts on software quality.

These options illustrate that the superiority of SASOAM over SOMA is acceptable. Although the result of the AHP method depends on the selected scenarios and votes of stakeholders, this is a rational result for two reasons: (1) SAP controls the quality of applications at runtime, whereas it is independent of SOMA. Therefore, the overload of SAP on SOMA is negligible. (2) SAP provides standard models, data structures, and formal description documents, which facilitate the verification and validation of applications. Besides, it helps humans understand legacy applications and applications developed from scratch.

5 Case study

In this study, SASOAM has been applied to a plug&play weapon system (PPWS) to make a self-adaptive system. Aircraft weapons are controlled by the following major services:

Comms if service (CIS): It provides some supervisory operations to manage both services and weapons.

Mission service (MS): It supplies essential operations such as destination identification for military missions.

Store mgmt service (SMS): It provides major operations to manage various weapon stores based

on the destination, type, and other significant features of military missions.

Carriage store service (CSS): It wraps weapon stores to manage their connections with other services (Windisch and Schlatt, 2005).

We have developed basic operations of PPWS in Visual Studio by using the WCF service application template. PPWS is not a legacy system, where a composition of the client-server, microkernel, and layers architectural patterns produces its initial architecture. Moreover, a socket programming approach reconfigures the architecture of PPWS in which sockets determine the collaboration between services. As Fig. 15 depicts, the client-server pattern has been instantiated to specify the collaboration between CIS, MS, and SMS services: CIS defines the mission of aircraft for MS and receives the status of SMS to fire toward the destination, MS obliges SMS to execute missions, and SMS manages weapon stores by use of the CSS service.

In addition, a two-layered architectural pattern has been applied to CSS and store services to manage weapons. In the end, CSS and store services are placed in top and down layers of the two-layered pattern, respectively. The microkernel pattern has linked the aforementioned patterns together, by including CIS, MS, and SMS in the microkernel component as the internal server component consists of CSS and store services.

5.1 Specifying PPWS non-functional requirements

Reliability, as the primary quality attribute of the PPWS, depends on the fault-tolerance, recoverability, and maturity sub-quality attributes (Harrison and Avgeriou, 2010b). As Table 9 shows, two general scenarios have been proposed by stakeholders to measure the reliability of PPWS due to its dependency on the fault-tolerance and availability architectural tactics. For availability, the firing request must be propagated from CIS to MS, SMS, CSS, and store services in less than 3 ms to attack and destroy the destination. The exception, ping/echo, and heartbeat tactics have been applied to the PPWS architecture to assess its availability, where the exception tactic detects software faults in the internal server component. Furthermore, the composition of the ping/echo and heartbeat tactics has periodically measured the

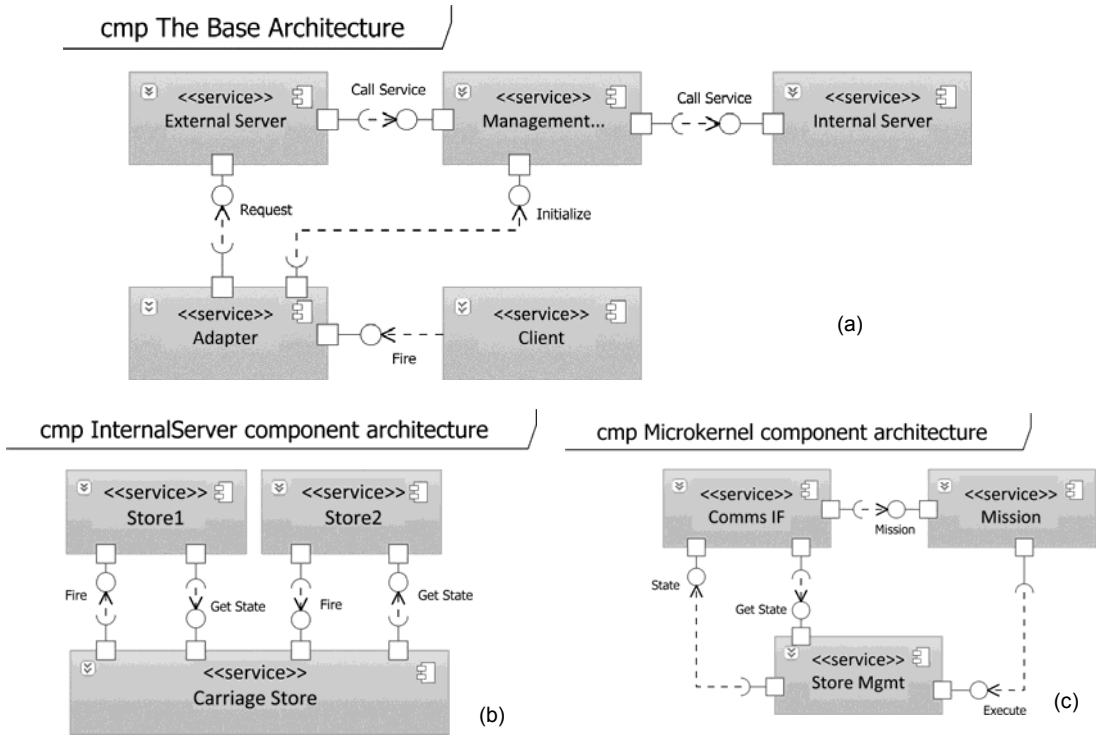

Fig. 15 Structural view of the plug&play weapon system: (a) base architecture; (b) internal server component architecture; (c) microkernel component architecture

Table 9 GSM template for reliability of PPWS

Goal and subgoal	KPI	Metric	Service
Reliability: Availability: The maximum firing delay is 3 ms Fault-tolerance: Prevent firing when a fault is detected in the weapon manager devices and services	U<53	1. Composition of the ping/echo and heartbeat tactics for the microkernel pattern and component 2. Exception tactic for the internal server component	1. Detect: client-server patterns between CIS, MS, and SMS; a layers pattern between CSS and store services; a microkernel pattern between services of PPWS 2. Modify the client-server and microkernel patterns based on the ping/echo and heartbeat tactics 3. Modify the layers pattern based on the exception tactic 4. Compose the exception, heartbeat, and microkernel tactics based on the client-server, layers, and microkernel patterns

number of either missed or delayed responses to check availability of both the microkernel pattern and the microkernel component.

The fault exception metric detects fault events in the store services to stop firing, where it is defined as the number of fault events occurring in the source code of PPWS stores. Reliability scenarios of PPWS have been developed as follows:

1. Detect the composition of the client-server pattern in CIS, MS, and SMS, the layers pattern in CSS and store services, and the microkernel pattern in PPWS services, prior to the execution of tactics. They have been distinguished from the architecture of PPWS in the specification step of SOMA.

2. Extend operations of the aforementioned patterns to customize them based on the exception, heartbeat, and microkernel tactics.

3. Compose the aforementioned tactics based on the client-server, layers, and microkernel patterns to assess reliability metrics of PPWS.

5.2 PPWS quality measurement

As Fig. 16 depicts, the client-server and microkernel patterns have been combined with the ping/echo and heartbeat tactics, whereas the layers pattern has been composed with the exception tactic based on architectural tactics modeled by Kim *et al.* (2009). RBML-PI has instantiated tactics where they have been modeled by the RBML language (Kim and Whittle, 2005). As Figs. 16a and 16b show, messages are propagated to the internal server in the microkernel pattern based on the ping/echo and heartbeat tactics; in contrast, Figs. 16c and 16d depict the corresponding phenomenon in the microkernel pattern.

Moreover, as Fig. 17 shows, the customized exception tactic has been applied to a two-layered pattern to model the internal server component. As Fig. 18 depicts, the RBML-PI add-on component has been added to the IBM Rational Software Architect to create the composition of aforementioned architectural patterns. Furthermore, Feature Modeling Plug-in (FMP) has been imported into the Rational Software Architect (RSA) tool to configure the feature model of quality attributes.

The reliability feature model has been designed based on the ISO/IEC-9126 quality model (Fitzpatrick, 1996). As Fig. 19 illustrates, the maturity, fault-tolerance, recoverability, and availability attributes influence the reliability quality attribute while both fault-tolerance and availability are grouped as mandatory features, whereas both recoverability and maturity are classified as optional features. Moreover, fault-tolerance is the prerequisite for availability assessment, while both maturity and recoverability are appropriate quality attributes to improve the accuracy of availability measurement (Kim *et al.*, 2009).

As Fig. 20 shows, the reliability feature model has been modeled by FMP in the RSA tool. The inclusive-or relationship between recoverability and maturity features has been determined by the rule <0-2> in the feature comparison tab. Furthermore, the relationship between the availability and fault-tolerance attributes has been defined by the following rule in the constraints tab:

(//availability)→(//fault-tolerance).

5.3 Quality measurement tree construction

The aforementioned RBML models are exported into C++ codes to measure the availability metric. As Fig. 3 depicts, tactics take advantage of metrics to assess quality attributes; e.g., both the ping-echo and heartbeat tactics count the number of expired packets, whereas the exception tactic recognizes how many

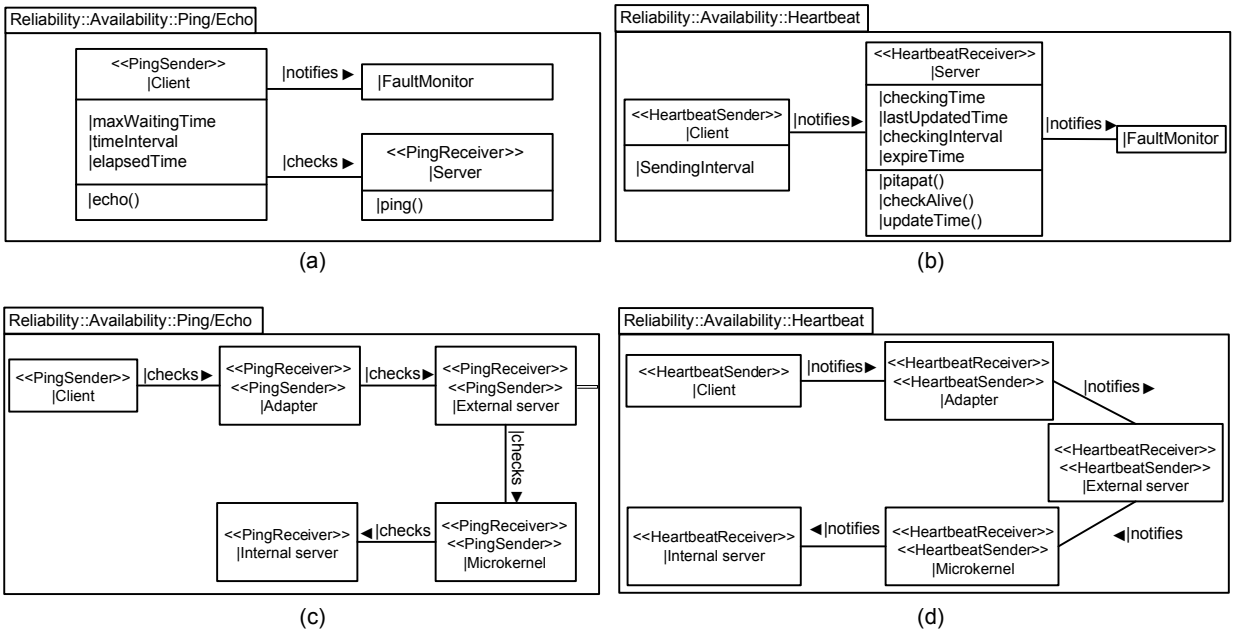

Fig. 16 Merging architectural tactics and patterns

(a) Merging client-server pattern with ping/echo tactic; (b) Merging client-server pattern with heartbeat tactic; (c) Merging microkernel pattern with ping/echo tactic; (d) Merging microkernel pattern with heartbeat tactic

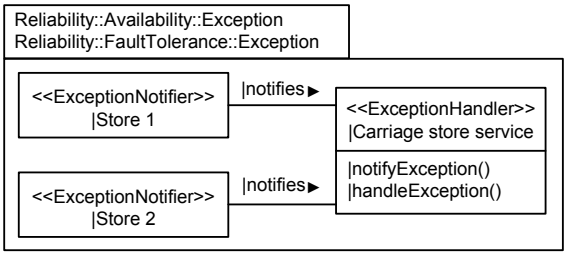

Fig. 17 Merging layers pattern and exception tactic

fault events occur in the PPWS source code. The instantiated tactics provide abstraction models for the base architecture of PPWS, which develops concrete operations. In other words, a two-layered structure, the same as the abstract factory pattern, has been applied to both tactic compositions and the base architecture to estimate quality attributes. Numerous scenarios have been applied to the tactic composition to estimate the probability of both received packets and fault numbers, where their results have negligible fluctuation.

The expert users begin an architecture tradeoff analysis method (ATAM) evaluation meeting (Bass *et al.*, 1998) to distinguish vector $\boldsymbol{W}$ for PPWS and vectors $\boldsymbol{\alpha}$'s for the client-server, two-layered, and microkernel patterns. As mentioned in Section 3.2, $\boldsymbol{W}$ specifies the impact of sub-systems on the reliability of PPWS while $\boldsymbol{\alpha}$'s determine the influence of components on the quality of their pattern. The results are summarized in Table 10 for the availability and fault-tolerance sub-quality attributes.

The probability values of both received packets and fault numbers (Table 11) have been applied to Eq. (1) to make a quality measurement tree based on the coefficient vectors. As shown in Fig. 21, the availability and fault-tolerance values have been represented by couples (QM[availability], QM[fault-tolerance]) in nodes, while couples (α_i, α_j) in edges show the influence of availability and fault-tolerance of subsystems on their parents, respectively.

The total quality of PPWS has been calculated from the following utility function:

$$U(\mathrm{QM}_{\mathrm{PPWS}}[\text{availability}], \mathrm{QM}_{\mathrm{PPWS}}[\text{fault-tolerance}]) = 5\times\mathrm{QM}_{\mathrm{PPWS}}[\text{availability}]+3\times\mathrm{QM}_{\mathrm{PPWS}}[\text{fault-tolerance}] = 5377,$$

where its coefficients are defined by stakeholders.

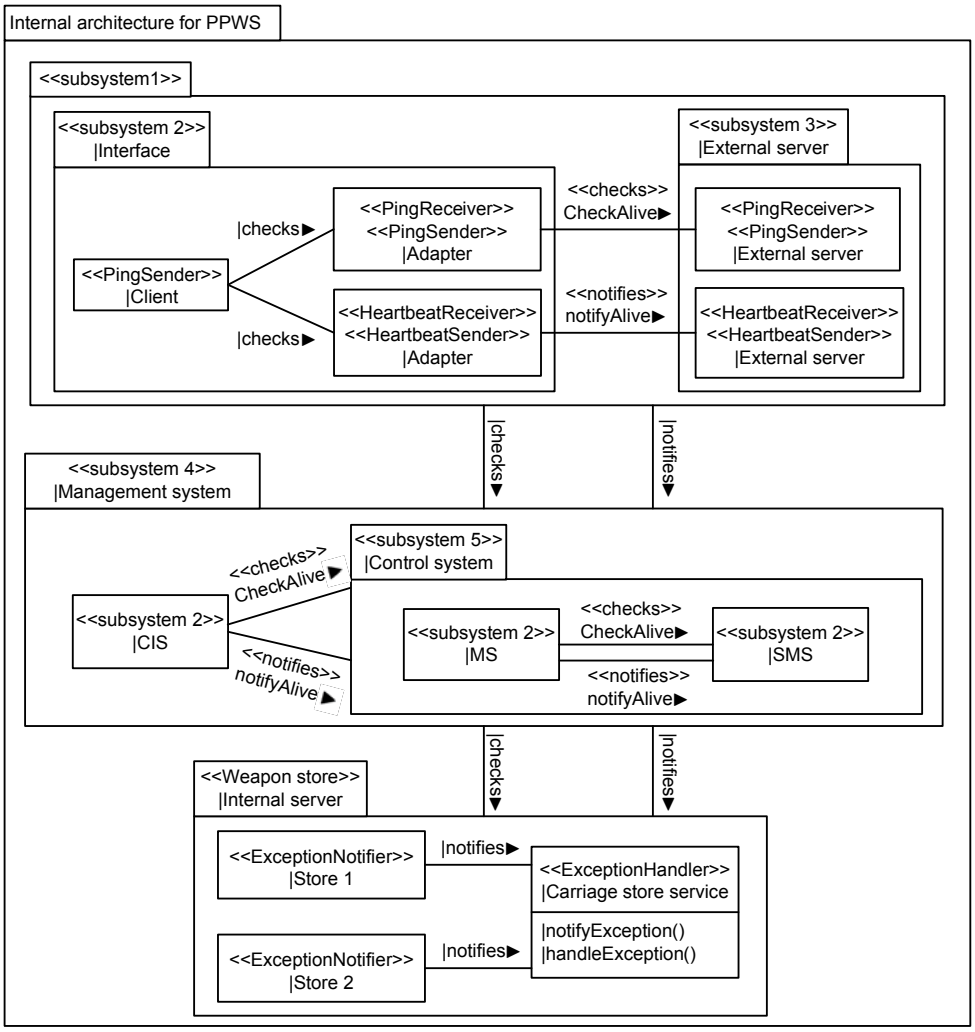

Fig. 18 The composition of the client-server, layers, and microkernel architectural patterns

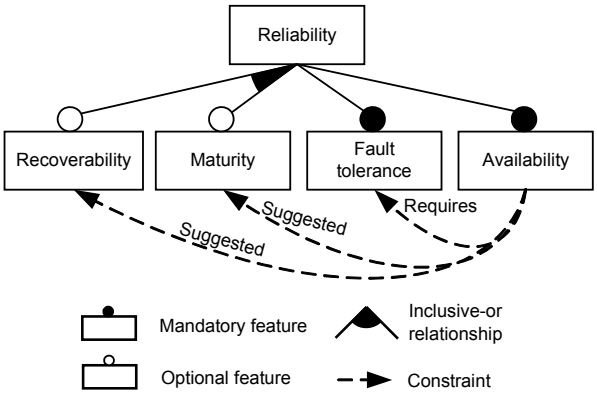

Fig. 19 Reliability feature model (Kim *et al.*, 2009)

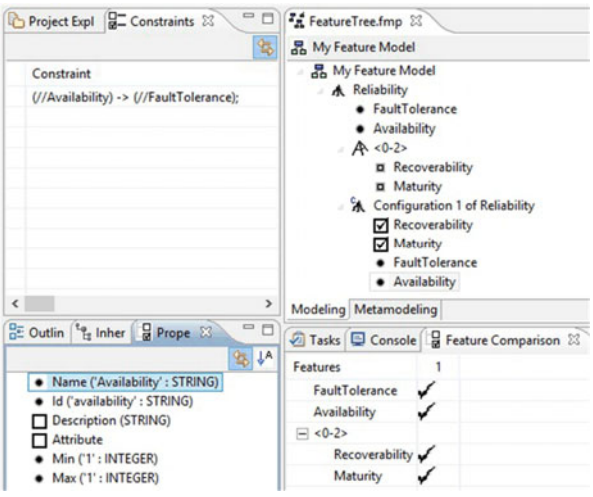

Fig. 20 FMP reliability feature model

Table 10 Coefficient vectors for PPWS components

Quality attribute	Coefficient[#]					
	CIS	MS	SMS	CSS	Store 1	Store 2
Availability	0.05	0.1	0.15	0.3	0.2	0.2
Fault-tolerance	0.07	0.08	0.15	0.2	0.25	0.25

Quality attribute	Coefficient[*]			
	Adapter	External server	Micro-kernel	Internal server
Availability	0.02	0.03	0.5	0.45
Fault-tolerance	0.03	0.07	0.5	0.4

Quality attribute	Coefficient[**]		Coefficient[***]	
	Client	Server	Layer 1	Layer 2
Availability	0.2	0.8	0.6	0.4
Fault-tolerance	0.1	0.9	0.7	0.3

[#] $\boldsymbol{W}$ for PPWS. $\boldsymbol{\alpha}$ for the [*]microkernel, [**]client-server, or [***]two-layered pattern

Table 11 $T(\rho)$ of PPWS

Quality attribute	CIS	SMS	MS	SS	Store 1	Store 2
Availability	0.8	0.3	0.2	0.1	0.9	0.9
Fault-tolerance	0.3	0.4	0.5	0.7	0.4	0.4

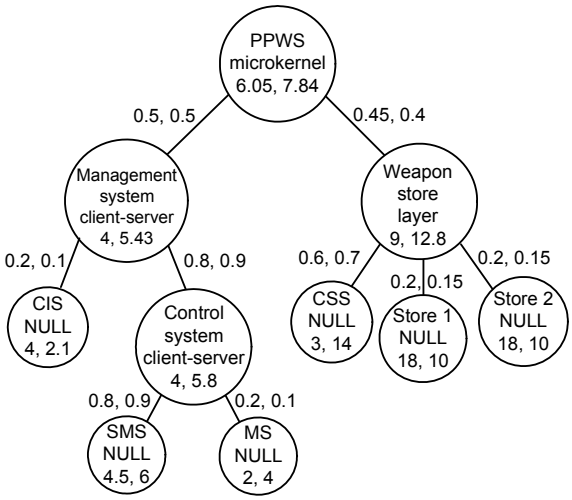

Fig. 21 The PPWS quality measurement tree

Since the value of the utility function is greater than the quality threshold, the architecture must be redesigned. To convert the aforementioned three architectural patterns into a useful composition of patterns, the plan service makes all possible quality measure trees with the layers, microkernel, and client-server patterns to explore one appropriate composition pattern which satisfies the quality attributes. This activity has recognized that the microkernel pattern must be changed to a layers pattern to satisfy the reliability threshold.

5.4 Deploying up-to-date PPWS architecture

Some distinct rules have been defined to convert patterns together; e.g., the following rules convert the microkernel pattern into a three-layered pattern:

$$\text{Client, adapter, external server} \xrightarrow{\text{import to}} \text{Layer 3},$$
$$\text{Microkernel} \xrightarrow{\text{imports to}} \text{Layer 2},$$
$$\text{Internal server} \xrightarrow{\text{imports to}} \text{Layer 1}.$$

In these rules, the client, adapter, and external server components move to the highest layer, the microkernel component moves to the middle layer, and the internal server moves to the lowest layer of the three-layered pattern. As the following XML template illustrates, configuration rules have been implemented by meaningful XML tags in which the passivate, activate, and reactive tags prepare PPWS subsystems for the deploy service to reconfigure its architecture by the unlink, link, remove, or create rules. CMS has developed an XML parser to interpret configuration files and order suitable commands to either the deploy service or the state management service. In the following, management rules have been explored to convert the microkernel pattern to a three-layered pattern based on the Kramer management rules (Kramer and Magee, 1990):

```
<passivate>
   <component name="connector">adapter connector,
   microkernel connector, external connector</component>
</passivate>
<unlink>
   <source name="microkernel" type="bi direct">CIS, SMS,
   MS</source>
   <destination name="connector">microkernel connector
   </destination>
   <source name="internal server" type="bi direct">CSS
   </source>
   <destination name="connector">microkernel connector
   </destination>
</unlink>
<create type="connector">layer 1, layer 2</create>
<remove type="connector">microkernel connector
</remove>
```

```
<link>
   <source name="microkernel" type="bi direct">CIS, SMS,
   MS</source>
   <destination name="connector">layer 2</destination>
   <source name="internal server" type="bi direct">CSS
   </source>
   <destination name="connector">layer 1</destination>
   <source name="connector" type="bi direct">layer 1
   </source>
   <destination name="Microkernel">CIS, SMS, MS
   </destination>
   <source name="connector" type="bi direct">layer 2
   </source>
   <destination name="component">adapter</destination>
</link>
<activate>
   <component name="connector">layer 1, layer 2
   </component>
</activate>
<reactivate>
   <component name="connector">adapter connector,
   external connector</component>
</reactivate>
```

The appropriate adaptation pattern has been developed for microkernel controllers based on the Gomma adaptation pattern (Gomaa and Hussein, 2004). As Fig. 22 depicts, the adapter component distributes service requests between the microkernel and external server components. The connector buffer requests when destination servers are busy. Moreover, the microkernel component diverts requests to the internal server when it provides no operations for responding to the requests. The status of the microkernel pattern has been managed by three connectors to isolate concerns of components from dynamic adaptation; i.e., the connectors develop adaptation state machines to change the status of the pattern according to either CMS commands or transactions between components. Although this study has implemented external and adapter connectors similar to the coordinator connector in Gomaa *et al.* (2010), as shown in Fig. 23, the microkernel connector has provided a different state machine. CMS sends an active command to start the microkernel connector. When the connector is waiting for the service request, a passivate command transfers the connector to the quiescent state. The E1 and E2 variables will be enabled when the microkernel and internal server are busy. When the connector is active, the passivate command changes its state to passive in order to prepare the connector for reconfiguration. The passivate state prevents the microkernel connector from sending requests toward the internal server and microkernel. The connector is transferred to the quiescent state when the internal server and microkernel disable the E1 and E2 variables by releasing their requests. Furthermore, CMS is notified of status changes in the connector.

Connectors have been programmed to support the link, unlink, active, and reactive commands. The link command establishes a connection between the source and destination components, whereas the unlink command disconnects two components. Moreover, the deploy service has instantiated connectors to support the create command, whereas the remove command deletes a connector instance permanently.

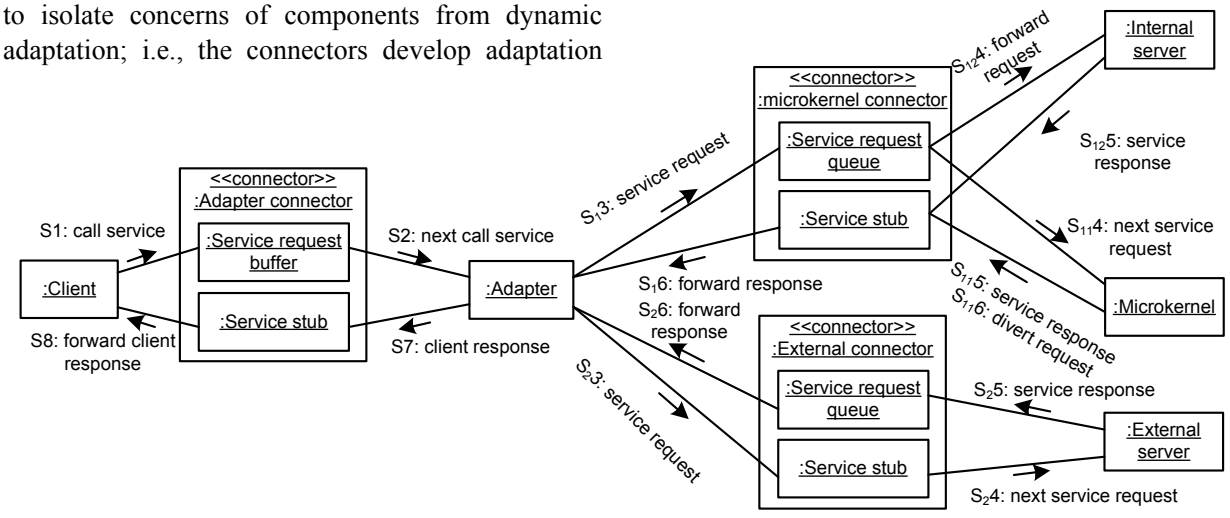

Fig. 22 Microkernel communication diagram

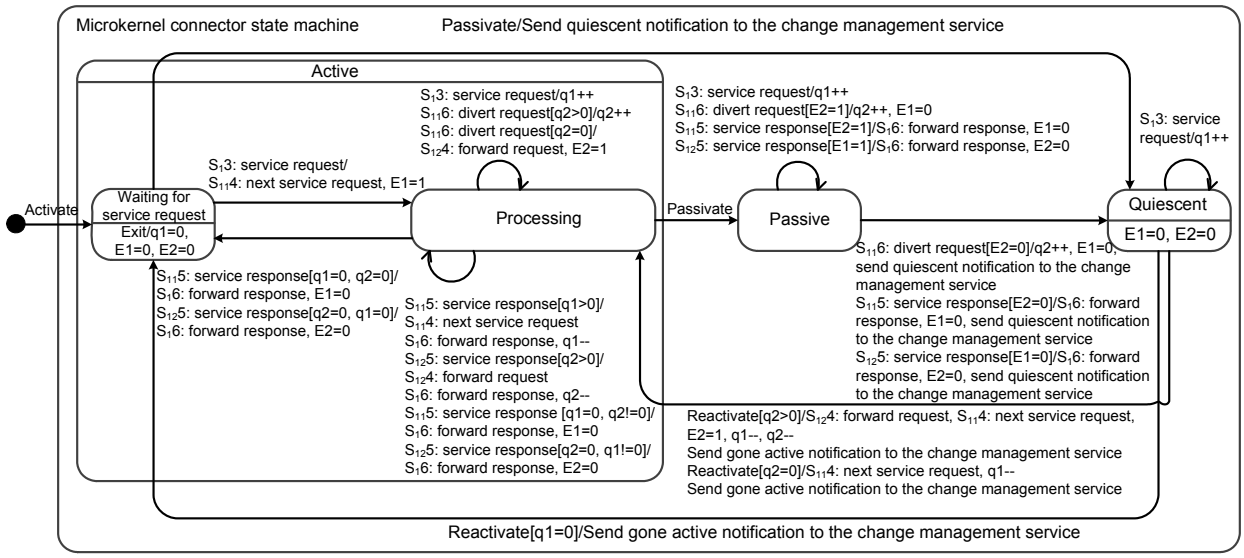

Fig. 23 Microkernel connector state machine

6 Conclusions and future work

The self-adaptive process (SAP) is an automatic approach for assuring software quality by a repetitive process. The abstraction, impracticality, and independency of SAP from the software development process are major challenges in the self-adaptive architecture domain. This paper has presented a novel SAP based on the available self-adaptive frameworks, approaches, technologies, and tools. The MAPE-K automation model is used to adapt SOA in case of deviance in its quality. Architectural patterns have been applied to software architectures in order to manage software quality by predefined architectural tactics, management rules, and adaptation patterns. Moreover, the proposed process has been appended to SOMA in a two-layered structure to generate a novel methodology, named SASOAM, which automates the quality assurance activity of SOMA. When an AHP-based method was applied to SOMA and SASOAM, the results showed that the proposed process enhances the maintainability of SOMA. In addition, SASOAM steps were applied to the plug&play weapon case study to show the practicality of this methodology.

In future work, we will simulate the customized patterns with queuing theory to estimate the correlation coefficients between the quality of patterns and the quality of their components. Moreover, we want to customize our previous style composition genetic algorithm to support the proposed SAP.

References

Andersson, J., Baresi, L., Bencomo, N., *et al.*, 2013. Software engineering processes for self-adaptive systems. Software Engineering for Self-Adaptive Systems II, p.51-75. [doi:10.1007/978-3-642-35813-5_3]

André, F., Daubert, E., Gauvrit, G., 2010. Towards a generic context-aware framework for self-adaptation of service-oriented architectures. 5th Int. Conf. on Internet and Web Applications and Services, p.309-314. [doi:10.1109/ICIW.2010.52]

Ardagna, D., Pernici, B., 2007. Adaptive service composition in flexible processes. *IEEE Trans. Softw. Eng.*, **33**(6): 369-384. [doi:10.1109/TSE.2007.1011]

Arevalo, G., Buchli, F., Nierstrasz, O., 2004. Detecting implicit collaboration patterns. 11th Working Conf. on Reverse Engineering, p.122-131. [doi:10.1109/WCRE.2004.18]

Arsanjani, A., Ghosh, S., Allam, A., *et al.*, 2008. SOMA: a method for developing service-oriented solutions. *IBM Syst. J.*, **47**(3):377-396. [doi:10.1147/sj.473.0377]

Babar, M.A., 2004. Scenarios, quality attributes, and patterns: capturing and using their synergistic relationships for product line architectures. 11th Asia-Pacific Software Engineering Conf., p.574-578. [10.1109/APSEC.2004.91]

Bachmann, F., Bass, L., Klein, M., 2002. Illuminating the Fundamental Contributors to Software Architecture Quality. Technical Report No. CNU/SEI-2002-TR-025, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA.

Bachmann, F., Bass, L., Kelin, M., 2003. Deriving Architectural Tactics: a Step Toward Methodical Architectural Design. Technical Report No. CMU/SEI-2003-TR-004, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA.

Bachmann, F., Bass, L., Nord, R., 2007. Modifiability Tactics. Technical Report No. CMU/SEI-2007-TR-002, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA.

Bass, L., John, B.E., 2003. Linking usability to software architecture patterns through general scenarios. *J. Syst. Softw.*, **66**(3):187-197. [doi:10.1016/S0164-1212(02)00076-6]

Bass, L., Clements, P., Kazman, R., 1998. Software Architecture in Practice. Addison-Wesley, Boston, USA, p.1-307.

Bhakti, M.A.C., Abdullah, A.B., Jung, L.T., 2010. Autonomic, self-organizing service-oriented architecture in service ecosystem. 4th IEEE Int. Conf. on Digital Ecosystems and Technologies, p.153-158. [doi:10.1109/DEST.2010.5610655]

Boehm, B.W., Brown, J.R., Lipow, M., 1976. Quantitative evaluation of software quality. 2nd Int. Conf. on Software Engineering, p.592-605.

Cardellini, V., Casalicchio, E., Grassi, V., *et al.*, 2009. QoS-driven runtime adaptation of service oriented architectures. Proc. 7th Joint Meeting of the European Software Engineering Conf. and the ACM SIGSOFT Symp. on the Foundations of Software Engineering, p.131-140. [doi:10.1145/1595696.1595718]

Cardellini, V., Casalicchio, E., Grassi, V., *et al.*, 2012. MOSES: a framework for QoS driven runtime adaptation of service-oriented systems. *IEEE Trans. Softw. Eng.*, **38**(5):1138-1159. [doi:10.1109/TSE.2011.68]

Castro, E., 2000. XML for the World Wide Web: Visual QuickStart Guide. Peachpit Press, USA, p.21-54.

Coad, P., 1992. Object-oriented patterns. *Commun. ACM*, **35**(9):152-159. [doi:10.1145/130994.131006]

de Lemos, R., Giese, H., Müller, H.A., *et al.*, 2013. Software engineering for self-adaptive systems: a second research roadmap. Software Engineering for Self-Adaptive Systems II, p.1-32. [doi:10.1007/978-3-642-35813-5_1]

di Nitto, E., Ghezzi, C., Metzger, A., *et al.*, 2008. A journey to highly dynamic, self-adaptive service-based applications. *Autom. Softw. Eng.*, **15**(3-4):313-341. [doi:10.1007/s10515-008-0032-x]

Dromey, R.G., 1995. A model for software product quality. *IEEE Trans. Softw. Eng.*, **21**(2):146-162. [doi:10.1109/32.345830]

Fielding, R.T., 2000. Architecture Styles and the Design of Network-Based Software Architectures. PhD Thesis, University of California, Irvine, USA.

Fitzpatrick, R., 1996. Software Quality: Definitions and Strategic Issues. Technical Report No. 1, Staffordshire University, Staffordshire, UK.

Fuhr, A., Horn, T., Riediger, V., *et al.*, 2013. Model-driven software migration into service-oriented architectures. *Comput. Sci.-Res. Devel.*, **28**(1):65-84. [doi:10.1007/s00450-011-0183-z]

Gacek, C., Giese, H., Hadar, E., 2008. Friends or foes?—a conceptual analysis of self-adaptation and IT change management. Proc. Int. Workshop on Software Engineering for Adaptive and Self-Managing Systems, p.121-128. [doi:10.1145/1370018.1370040]

Ganter, B., Wille, R., 1999. Formal Concept Analysis: Mathematical Foundations. Springer, Heidelberg. [doi:10.1007/978-3-642-59830-2]

Gomaa, H., 2011. Pattern-based software design and adaptation. 3rd Int. Conf. on Pervasive Patterns and Applications, p.90-95.

Gomaa, H., Hussein, M., 2004. Software reconfiguration patterns for dynamic evolution of software architectures. Proc. 4th Working IEEE/IFIP Conf. on Software Architecture, p.79-88. [doi:10.1109/WICSA.2004.1310692]

Gomaa, H., Hashimoto, K., Kim, M., *et al.*, 2010. Software adaptation patterns for service-oriented architectures. Proc. ACM Symp. on Applied Computing, p.462-469. [doi:10.1145/1774088.1774185]

Gu, T., Pung, H.K., Zhang, D.Q., 2005. A service-oriented middleware for building context-aware services. *J. Network Comput. Appl.*, **28**(1):1-18. [doi:10.1016/j.jnca.2004.06.002]

Haase, P., Lewen, H., Studer, R., *et al.*, 2008. The NeOn ontology engineering toolkit. WWW.

Harrison, N.B., Avgeriou, P., 2010a. How do architecture patterns and tactics interact a model and annotation. *J. Syst. Softw.*, **83**(10):1735-1758. [doi:10.1016/j.jss.2010.04.067]

Harrison, N.B., Avgeriou, P., 2010b. Implementing reliability: the interaction of requirements, tactics and architecture patterns. Architecting Dependable Systems VII, **6420**: 97-122. [doi:10.1007/978-3-642-17245-8_5]

Hull, R., Hill, M., Berardi, D., 2005. Semantic Web Services Usage Scenario: e-Service Composition in a Behavior Based Framework. Semantic Web Services Initiative Language.

Kang, K.C., Cohen, S.G., Hess, J.A., *et al.*, 1990. Feature-Oriented Domain Analysis (FODA) Feasibility Study. Technical Report No. CMU/SEI-90-TR-21, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA.

Kazman, R., Carrière, S.J., Woods, S.G., 2000. Toward a discipline of scenario-based architectural engineering. *Ann. Softw. Eng.*, **9**(1-2):5-33. [doi:10.1023/A:1018964405965]

Kephart, J.O., Chess, D.M., 2003. The vision of autonomic computing. *Computer*, **36**(1):41-50. [doi:10.1109/MC.2003.1160055]

Kim, D.K., Whittle, J., 2005. Generating UML models from domain patterns. 3rd ACIS Int. Conf. on Software Engineering Research, Management and Applications, p.166-173. [doi:10.1109/SERA.2005.44]

Kim, S., Kim, D., Lu, L., *et al.*, 2009. Quality-driven architecture development using architectural tactics. *J. Syst. Softw.*, **82**(8):1211-1231. [doi:10.1016/j.jss.2009.03.102]

Kozaki, K., Kitamura, Y., Ikeda, M., *et al.*, 2002. Hozo: an environment for building/using ontologies based on a fundamental consideration of “role” and “relationship”.

LNCS, **2473**:213-218. [doi:10.1007/3-540-45810-7_21]

Kramer, J., Magee, J., 1990. The evolving philosophers problem: dynamic change management. *IEEE Trans. Softw. Eng.*, **16**(11):1293-1306. [doi:10.1109/32.60317]

Kruchten, P.B., 1995. The 4+1 view model of architecture. *IEEE Softw.*, **12**(6):42-50. [doi:10.1109/52.469759]

Lane, S., Bucchiarone, A., Richardson, I., 2012. SOAdapt: a process reference model for developing adaptable service-based applications. *Inform. Softw. Technol.*, **54**(3):299-316. [doi:10.1016/j.infsof.2011.10.003]

Malek, S., Esfahani, N., Menasce, D.A., *et al.*, 2009. Self-Architecting Software SYstems (SASSY) from QoS-annotated activity models. ICSE Workshop on Principles of Engineering Service Oriented Systems, p.62-69. [doi:10.1109/PESOS.2009.5068821]

Matinlassi, M., 2005. Quality-driven software architecture model transformation. Proc. 5th Working IEEE/IFIP Conf. on Software Architecture, p.199-200. [doi:10.1109/WICSA.2005.56]

McGuinness, D.L., van Harmelen, F., 2004. OWL Web Ontology Language Overview. W3C Recommendation.

Menasce, D., Gomaa, H., Malek, S., *et al.*, 2011. SASSY: a framework for self-architecting service-oriented systems. *IEEE Softw.*, **28**(6):78-85. [doi:10.1109/MS.2011.22]

Moaven, S., Ahmadi, H., Habibi, J., *et al.*, 2008a. A decision support system for software architecture-style selection. 6th Int. Conf. on Software Engineering Research, Management and Applications, p.213-220. [doi:10.1109/SERA.2008.26]

Moaven, S., Habibi, J., Ahmadi, H., *et al.*, 2008b. A fuzzy model for solving architecture styles selection multi-criteria problem. 2nd UKSIM European Symp. on Computer Modeling and Simulation, p.388-393. [doi:10.1109/EMS.2008.45]

Moaven, S., Kamandi, A., Habibi, J., *et al.*, 2009. Toward a framework for evaluating heterogeneous architecture styles. 1st Asian Conf. on Intelligent Information and Database Systems, p.155-160. [doi:10.1109/ACIIDS.2009.68]

Noy, N.F., Crubézy, M., Fergerson, R.W., *et al.*, 2003. Protege-2000: an open-source ontology-development and knowledge-acquisition environment. Annual Symp. of the American Medical Informatics Association, p.953.

Papazoglou, M.P., Heuvel, W.V.D., 2007. Service oriented architectures: approaches, technologies and research issues. *VLDB J.*, **16**(3):389-415. [doi:10.1007/s00778-007-0044-3]

Ramsin, R., Paige, R.F., 2008. Process-centered review of object oriented software development methodologies. *ACM Comput. Surv.*, **40**(1):3.1-3.89. [doi:10.1145/1322432.1322435]

Rawashdeh, A., Matalkah, B., 2006. A new software quality model for evaluating COTS components. *J. Comput. Sci.*, **2**(4):373-381. [doi:10.3844/jcssp.2006.373.381]

Romay, M.D.P., Fernández-Sanz, L., Rodríguez, D., 2011. A systematic review of self-adaptation in service-oriented architectures. 6th Int. Conf. on Software Engineering Advances, p.331-337.

Rosen, M., Lublinsky, B., Smith, K.T., *et al.*, 2008. Applied SOA: Service-Oriented Architecture and Design Strategies. Wiley Publishing, Indianapolis, p.260-350.

Saaty, T.L., 1980. The Analytic Hierarchy Process. McGraw Hill, New York.

Scott, J., Kazman, R., 2009. Realizing and Refining Architectural Tactics: Availability. Technical Report No. CMU/SEI-2009-TR-006, Software Engineering Institute.

Siljee, J., Bosloper, I., Nijhuis, J., *et al.*, 2005. DySOA: making service systems self-adaptive. *LNCS*, **3826**:255-268. [doi:10.1007/11596141_20]

Strang, T., Linnhoff-Popien, C., 2004. A context modeling survey. 1st Int. Workshop on Advanced Context Modeling, Reasoning and Management.

Wang, S., Sun, Q., Yang, F., 2010. Towards web service selection based on QoS estimation. *Int. J. Web Grid Serv.*, **6**(4):424-443. [doi:10.1504/IJWGS.2010.036406]

Weyns, D., Schmerl, B., Grassi, V., *et al.*, 2013. On patterns for decentralized control in self-adaptive systems. Software Engineering for Self-Adaptive Systems II, p.76-107. [doi:10.1007/978-3-642-35813-5_4]

Windisch, A., Schlatt, H., 2005. AADL-modelling of plug & play weapon system architecture. Systems Engineering AADL Workshop, p.121-132.

Yang, S.J.H., Zhang, J., Chen, I.Y.L., 2008. A JESS-enabled context elicitation system for providing context-aware Web services. *Expert Syst. Appl.*, **34**(4):2254-2266. [doi:10.1016/j.eswa.2007.03.008]