Design for Adaptation of Service-Based Applications: Main Issues and Requirements*

Antonio Bucchiarone[2], Cinzia Cappiello[1], Elisabetta Di Nitto[1], Raman Kazhamiakin[2], Valentina Mazza[1], and Marco Pistore[2]

[1] Politecnico di Milano
Piazza Leonardo Da Vinci 32 20133 Milano, Italy
[2] Fondazione Bruno Kessler
Via Santa Croce 77 38100 Trento, Italy
{bucchiarone,raman,pistore}@fbk.eu,
{cappiell,dinitto,vmazza}@elet.polimi.it

Abstract. Service-based applications are considered a promising technology since they are able to offer complex and flexible functionalities in widely distributed environments by composing different types of services. These applications have to be adaptable to unforeseen changes in the functionality offered by component services and to their unavailability or decreasing performances. Furthermore, when applications are made available to a high number of potential users, they should also be able to dynamically adapt to the current context of use as well as to specific requirements and needs of the specific users. In order to address these issues, mechanisms that enable adaptation should be introduced in the life-cycle of applications, both in the design and in the runtime phases. In this paper we propose an extension of a basic iterative service-based applications life-cycle with elements able to deal with the adaptation-specific needs. We focus, in particular, on the design phase and suggest a number of design principles and guidelines that are suitable to enable adaptation. We discuss about the effectiveness of the proposed methodology by means of real-world scenarios over various types of service-based applications.

1 Introduction

In the era of the Internet of Services, service-based applications (SBAs) are considered the most promising technology since they are able to offer complex and flexible functionalities in widely distributed environments by composing different types of services. Such services are often not under the control of systems developers, but they are simply exploited to obtain a specific functionality.

While this, on the one side, enables separation of concerns and highly simplifies the design effort of those in charge of building SBAs, on the other side, it introduces critical dependencies between SBA themselves and the services they are exploiting. These last ones, in fact, could change without notice or be unavailable for unprecised time intervals. Therefore, SBAs have to be able to *adapt* to these unforeseen changes. Adaptation

* The research leading to these results has received funding from the European Community's Seventh Framework Programme FP7/2007-2013 under grant agreement 215483 (S-Cube).

A. Dan, F. Gittler, and F. Toumani (Eds.): ICSOC/ServiceWave 2009, LNCS 6275, pp. 467–476, 2010.

can be accomplished through various strategies that we will discuss in this paper. While the literature presents a good number of approaches that deal with self-adaptation of SBAs, most of them address this issue by hard coding in the infrastructure supporting the execution of SBAs a limited number of adaptation strategies that are triggered only when some specific and known events happen. We argue that this approach does not necessarily cover all needs that may arise. In some cases these needs are unknown and cannot be foreseen once for all.

Current methodologies for service-oriented applications are based on the results carried out in the fields of classical software and system engineering. Moreover, while almost all of the proposed approaches for life-cycle, (noticeable are the proposals by SOUP (Service Oriented Unified Process) [9] or ASTRO [12] focusing on the possibility to monitor and intervene on SBAs in order to recovery from unwanted and unexpected behaviour), assume human interventions, Linner et al. [8] propose a life-cycle supporting self-adaptation of the service-based application even if they lack of a explicit guidelines for the design of adaptable service based applications. Various frameworks supporting adaptation have been defined in the literature, each of them addressing a specific issue. Some authors focus on triggering adaptation strategies as a consequence of a requirement violation [11], or for satisfying some application constraints [13]. Adaptation strategies could be specified by means of policies to manage the dynamism of the execution environment [4,3,2] or of the context of mobile service-based applications [10].

All aforementioned approaches show interesting features, but even those that enable the definition of various adaptation strategies lack a coherent design approach to support designers in this complex task. The methodology we propose in Section 3 can be considered as a first step in this direction. The approach that we advocate is based on the idea that adaptation strategies can be programmed at design/implementation time and be associated with triggering events whenever possible, either before the execution or during the execution itself. This approach is adopted in our earlier work [3] and in other works (e.g., [10]). However, even in these cases the emphasis is on the mechanisms offered to design strategies and to trigger them, more than on a holistic, coherent, and easy to apply *design for adaptation* approach that supports developers in the usage of the available mechanisms. The objective of this paper is to go in the direction of this design for adaptation approach. We define a life-cycle for SBAs where adaptation is a first class concern.

As we think that adaptation works properly only in the case the application is designed to be adaptable, we focus, in particular, on the identification of a number of design principles and guidelines that are suitable to enable adaptation. The effectiveness of such principles and guidelines is analyzed with reference to some real-world scenarios. The rest of the paper is structured as follows: Section 2 discusses about the various facets of adaptation and evolution we deal with. Section 2.1 presents our life-cycle and Section 3 defines the design for adaptation strategies, principles, and guidelines. Finally, Section 4 assesses them with respect to the case studies and Section 5 concludes the paper.

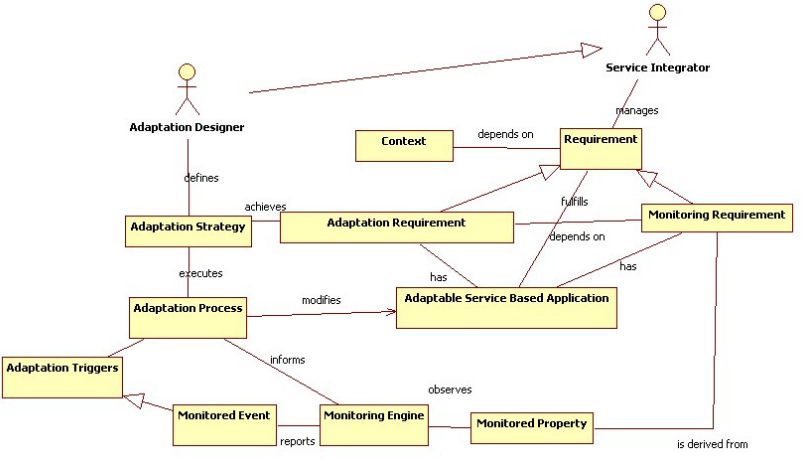

Fig. 1. Main concepts related to the definition and operation of Adaptable SBAs

2 Adaptation and Evolution in Service-Based Applications

Figure 1 shows the main ingredients that are needed for building and operating *Adaptable Service-Based Applications*. An adaptable SBA not only is usually able to satisfy some *Requirements*, but it also poses new requirements in terms of monitoring and adaptation aspects. *Monitoring Requirements* concern the need for detecting (part of) those situations that may trigger the need for adapting an SBA. From these requirements, designers should derive the properties to be monitored. These are then observed at runtime by a *Monitoring Engine* that, based on their values, is able to emit some *Monitored Events*. *Adaptation Requirements* are fulfilled by *Adaptation Strategies* that can be executed during the *Adaptation Process* that is triggered by *Monitored Events* or by any other external stimulus that can be acquired by the system and that leads to the modification of the Adaptable SBA. Note that analogously to the classification traditionally used to characterize software maintanance [1] we can identify similar types of adaptation: Perfective Adaptation, Corrective Adaptation, Adaptive Adaptation, Preventive Adaptation, and Extending Adaptation. An important role in our view is played by the *Context*. It includes users and execution properties. Users' characteristics and preferences can be obtained explicitly, for instance, by filling a user profile, or derived implicitly by profiling users at run-time. Other information such us the users' geographical position, the temporal details, and the actions that characterize the interaction of the users with the surrounding space can be obtained through monitoring. Execution properties are those that concern the conditions under which the SBA and its component services execute. Generally adaptation requires some temporary modification permitting to respond to changes in the requirements and/or in the application context or to faulty situations. An example of adaptation for a service composition could be the re-execution of a unavailable service or a substitution of a unsuitable service. Other situations could require the re-design and/or the re-engineering of the application modifying it permanently, in such case adaptation is called *evolution*. Moreover evolution could

be needed if a faulty situation requiring adaptation happens very often: in such case, a modification of the application logic would be preferred to the frequent enactment of the needed adaptation strategies.

2.1 Capturing Adaptation and Evolution Aspects in a Life-Cycle

As discussed in the previous sections, there is a need for introducing a life-cycle for SBAs that takes adaptation into explicit account. The life-cycle shown in Figure 2 highlights not only the typical design-time iteration cycle, but it also introduces a new iteration cycle at runtime that is undertaken in all the cases in which the adaptation needs are addressed on-the-fly. The two cycles coexist and support each other during the lifetime of the application. In particular the design time activities allow for *evolution* of the application, that is, for the introduction of permanent and, usually, important changes, while the runtime activities allow for temporary *adaptation* of the application to the specific circumstances that are occurring at a certain time. Figure 2 also shows the various adaptation- and monitoring-specific actions (boxes) carried out throughout the life-cycle of the SBA, the main design artifacts that are exploited to perform adaptation (hexagons), and the phases where they are used (dotted lines). At the *requirements engineering and design* phase the adaptation and monitoring requirements are used to perform the design for adaptation and monitoring.

During SBA *construction*, together with the construction of the SBA, the corresponding monitors and the adaptation mechanisms are being realized. The *deployment* phase also involves the activities related to adaptation and monitoring: deployment of the adaptation and monitoring mechanisms and deployment time adaptation actions (e.g., binding). During the *operation and management* phase, the run-time monitoring is executed, using some designed properties, and help the SBA to detect relevant context and system changes. After this phase the left-side of the life-cycle is executed. Here, we can proceed in two different directions: executing evolution or adaptation of the SBA. In the first case we re-start the right-side of the cycle with the requirements engineering and design phase while in the second case we proceed identifying adaptation needs that can be triggered from monitored events, adaptation requirements or context conditions.

For each adaptation need it is possible to define a set of *suitable strategies*. Each adaptation strategy can be characterized by its complexity and its functional and non functional properties. The identification of the most suitable strategy is supported by a *reasoner* that also bases its decisions on multiple criteria extracted from the current situation and from the knowledge obtained from previous adaptations and executions. Details on these issues are discussed in Section 3. After this selection, the *enactment of the adaptation strategy* is performed.

3 Design for Adaptation: Main Ingredients

As discussed in the previous sections, in order to offer efficient and reliable applications, it is necessary to guarantee that the service components are always aligned with the changing world around them. At design time possible alternatives to support service adaptation should be identified. For the same SBA, several adaptation strategies can

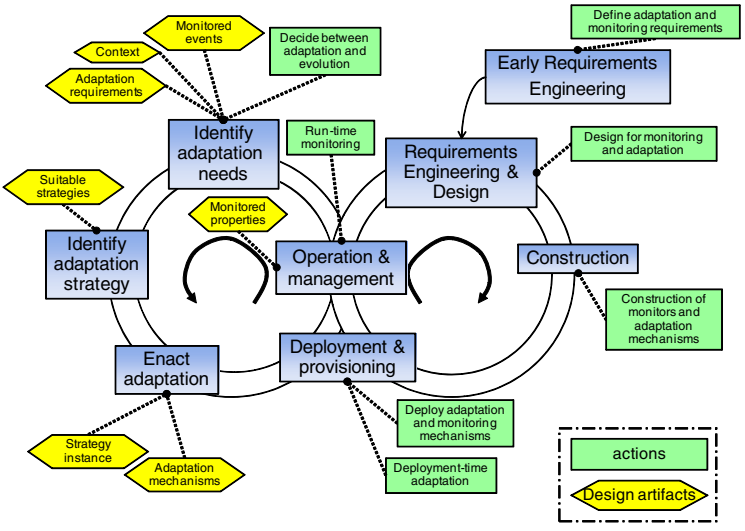

Fig. 2. The Life-Cycle of Adaptable SBAs

be adopted and the selection of the most suitable one to activate can be a complex issue multiple criteria have to be considered. In the following, guidelines to support this selection are provided.

Adaptation Strategies: While a SBA is executing, different changes might occur in the environment and cause inefficiencies. In order to avoid the application performance degradation, it is necessary to identify the most suitable adaptation strategy that is able to maintain aligned the application behaviour with the context and system requirements. Among the adaptation strategies, it is possible to distinguish domain-independent or domain-dependent strategies. The former are applicable in almost every application context while the adoption of the latter is limited to specific execution environments. Table 1 defines the most common domain-independent adaptation strategies.

Identification of Adaptation Triggers: The adaptation in SBA may be motivated by variety of factors, or *triggers*. Such triggers may concern the *component services* or the

Table 1. Description of the most common domain-dependent adaptation strategies

Adaptation Strategy	Description
Service substitution	Reconfiguration of the SBA with a dynamic substitution of the a service with another one
Re-execution	The possibility of going back in the process to a point defined as safe for redoing the same set of tasks or for performing an alternative path
(Re-)negotiation	Simple termination of the service used on the requester side and re-negotiation of the SLA properties to complex management on reconfiguration activities on the provider side
(Re-)composition	Reorganization and rearrangement of the control flow that links the different service components in the business application
Compensation	Definition of ad-hoc activities that can undo the effects of a process that fails to complete
Trigger evolution	Insertion of workflow exception able to activate the application evolution
Log/update adaptation information	Storage of all the information about the adaptation activities for different goals (e.g., service reputation, QoS analysis, outcome of adaptation)
Fail	The system reacts to the changes by storing the system status and causing the failure of the service and re-executing it

context of SBAs. As for the former we can identify changes in the *service functionality*: variation of the service interface (e.g., signatures, data types, semantics), variation of service interaction protocol (e.g., ordering of messages), and failures; and changes in the *service quality*: service availability, degrade of QoS parameters, violation of SLA, decrease of service reputation (e.g., black lists), etc. As for the contextual triggers, one can distinguish: changes in the *business context*, such as changes in agile service networks, new business regulations and policies; changes in the *computational context*, such as different devices, protocols, networks; and changes in the *user context*, such as different user groups and profiles, social environment or physical settings (e.g., location/time), different user activities. Some of these aspects may be interleaved. For example, if a user moves to a new location (i.e., change in the user context), new set of services may be available (i.e., change in the business context) with different bandwidth (i.e., change in the computational context). As represented in Table 2 each trigger can be associated with a set of adaptation strategies that are suitable to re-align the application within the system and/or context requirements. In order to select the adaptation strategy to apply, it is necessary to consider that adaptation triggers may be associated with other requirements that are important for designing and performing adaptation, in particular: the *Scope* of the change, i.e., whether the change affects only a single running instance of the SBA or influences the whole model and, the *Impact* of the change, i.e., the possibility of the application to accomplish its current task. Depending on these parameters different strategies may apply. For example, when the scope of the change concerns the whole application model "trigger evolution" strategy applies. As for the impact, such strategies as "re-execution" or "substitution" may apply when the SBA state did not change and the task still can be accomplished. On the other hand, "compensation", "fail", or "trigger evolution" apply when there is no way to complete the current task.

3.1 Design Guidelines

In order to design adaptable SBAs, it is necessary to come up with the principles and guidelines for:

- *Modeling adaptation triggers*, i.e., both the situation when the adaptation is needed (monitored property) and the specific adaptation need.
- *Realizing adaptation strategies*. This includes modeling strategies, their properties, and their aggregation, and relating them to the underlying mechanisms and run-time infrastructure.

Table 2. Relationships between Adaptation Triggers and Adaptation Strategies

Adaptation Trigger	Adaptation Strategy
Changes in the service functionality	Service Substitution, Re-execution, Re-negotiation, Re-composition, Compensation, Fail
Changes in the service quality	Service Substitution, Re-Negotiation
Changes in the business context	Service Substitution, Re-Negotiation, Re-composition, Trigger Evolution, Log/update relevant adaptation information
Changes in the computational context	Service Substitution, Re-negotiation, Re-composition, Trigger Evolution, Log/update relevant adaptation information
Changes in the user context	Service Substitution, Re-negotiation, Re-composition, Trigger Evolution, Log/update relevant adaptation information

– *Associating adaptation strategies to triggers*. We have already demonstrated how the scope and impact of change influence this relation. Other factors may include autonomy (i.e., if the adaptation should be done without human involvement) or performance (e.g., how fast an adaptation strategy is).

One of the key aspects cross-cutting to these design tasks is the dynamicity of the environment with respect to the adaptation problem. This refers to the *diversity* of specific adaptation needs and of factors the adaptation strategies depend on. According to this distinction, the following design approaches may be defined:

– *Built-in adaptation*. If possible adaptation needs and possible adaptation configurations are fixed and known a priori, it is possible to completely specify them at design time. The specification may be performed by extending the standard SBA notations (e.g., BPEL) with the adaptation-specific tools [5] using ECA-like (event-condition-action) rules [3], or aspect-oriented approaches [7]. Typical strategies suitable for such adaptations are: service substitution (by using a predefined list of alternative service), re-execution, compensation, re-composition (by using predefined variants), fail.
– *Abstraction-based adaptation*. When the adaptation needs are fixed, but the possible configurations in which adaptation is triggered, are not known a priori, the concrete adaptation actions cannot be completely defined at design time. In such a case, a typical pattern is to define an abstract model of an SBA and a generic adaptation strategy, which are then made concrete at deployment/run-time. For example, it is possible to use the abstract composition model in which concrete services are discovered and bound at run-time based on the context [14]. Otherwise, it is also possible to define at design time only the final goal or utility function and then it is achieved or optimized by dynamic service re-composition at run-time on the basis of based on the specific environment and available services [15]. Strategies that may be used for such adaptation are service concretization, service substitution (by dynamic discovery), re-composition (based on predefined goal/utility function), re-negotiation.
– *Dynamic adaptation*. It is possible that adaptation needs that may occur at run-time are not known or cannot be enumerated at design time. In such a case, it is necessary to provide specific mechanisms that select and instantiate adaptation strategies depending on a specific trigger and situation. The scenarios in which such adaptation is needed may include modifications or corrections of business process instances via ad-hoc actions and changes performed by business analyst, changes in the user activities that entail modification of current composition and creation of new ones. At run-time, these mechanisms are exploited to (i) identify one or more suitable adaptation strategies depending on a concrete situation, (ii) define concrete actions and parameters of those strategies, and (iii) execute them using the appropriate mechanisms. This type of adaptation may be built on top of the others to realize specific adaptation needs; the focus, however, is on the mechanisms for extracting specific adaptation strategies and actions at run-time. Accordingly, different strategies may apply here: re-composition, service substitution, and compensation, re-execution, evolution, fail. The realization mechanisms, however, are

different; they may require active user involvement (e.g., for making decisions, for performing ad-hoc changes, etc.).

4 Discussion

In this section we illustrate how the design for adaptation activities may be performed in different scenarios that target different domains and focus on different adaptation aspects. In particular, given the specific characteristics of the scenario, we show the factors triggering adaptation, the types of adaptation realization suitable for the scenarios, and the appropriate adaptation strategies. Table 3 summarizes all these aspects with reference to the considered case studies.

Automotive scenario. Let us consider complex supply-chain business processes in the automobile production domain. The activities of the processes include ordering and importing automobile body parts from suppliers, manufacturing activities, customization of the specific products according to the needs of the customers, etc. The processes are usually long-running and involve a wide range of enterprise services provided by organizations such as various suppliers, logistics providers, warehouses, and regional representatives. All these participate in an Agile Service Network (ASN) where they rely one on each other services in a dynamic way. The critical changes that require adaptation in this scenario range from instance-specific problems (e.g., failures and SLA violations, specific customers) to the changes that affect the whole SBA (e.g., changes in business context). In the former case, it is possible to apply built-in adaptation and define the reactions at design-time by completely describing the corresponding strategy (compensation activities, process variants for different customers) or its parameters (for SLA re-negotiation, for service substitution). In the latter case, the specific adaptation strategy is chosen at run-time as the effect of changes on the system is not known. In the business settings, such a choice can hardly be automated; the business requirements and decisions require human involvement. In particular, business analysts make decisions on triggering evolution and/or on how the running process instances should be changed (i.e., ad-hoc process modifications).

Wine production scenario. In the wine production application domain, the activities of vineyard cultivation handling, the control of grapes maturation, their harvesting and

Table 3. Adaptation characteristics of the scenarios in Section 4

Case Study	Properties	Adaptation Trigger	Design Approach	Adaptation Strategy
Automotive	Stable context and potential partners, long-running SBAs, diversity of adaptation needs, decisions require human involvement	functional changes, failures, SLA violations, changes in business context	Dynamic adaptation (human-driven); built-in adaptation (for compensation or process customization)	Service substitution (selecting from ASN partners); SLA re-negotiation; re-composition by ad-hoc changes of process control/data; re-composition by selecting predefined process variants; compensation; trigger evolution
Wine	Fully dynamic and unreliable services, fully autonomous SBA	degrade of service (sensor) QoS	Abstraction-based adaptation	Re-composition of services (to optimize resource utility function), domain-specific actions (e.g., data transfer frequency changes)
Mobile user	Strong dependency from context and goals of users	context changes, changes of user activities	Abstraction-based (for context changes), dynamic	service substitution (by dynamic discovery); re-composition.

fermentation rely on extensive use of a SBA realized on top of a Wireless Sensor and Actuator Network (WSAN). In this context sensors and actuators are seen as service providers able, respectively, to report information regarding the state of the vineyard and to execute some specific actions. These devices are not fully reliable. They may crash, run out of battery, or provide incorrect information. This may happen due to changes in physical context (e.g., humidity) or to the activation of new measurement activities (e.g., depending on the season). In this scenario the dynamically changing state of the WSAN network requires continuous monitoring and optimization of the resource usage. For this purpose, the adaptation should (i) re-arrange the sensor network in order to minimize the sensor energy consumption, and (ii) optimize the modes, in which the sensors operate, e.g., by optimizing the data transfer frequency. While the latter solution requires domain-specific realization mechanisms, the former may be achieved by dynamic re-composition of services to minimize of the utility function corresponding to the energy consumption (see, e.g., [15]).

Mobile user application scenario. An increasing number of modern applications aims to give end users access to various services through mobile devices. Such services include route planning, transport ticket booking, services for accessing social networks, and a wide range of information services mashed up by those applications. In this scenario the SBA should adapt (i) to the changes in its context (e.g., changing location and time, different user settings), and (ii) to the changes in the user activities and plans. The former may be very dynamic: different services may apply for different locations or user settings. Abstraction-based adaptation is indeed required in this case: the abstract activities (e.g., buy a ticket for local transportation) are defined at design-time and made concrete at run-time using service concretization techniques (e.g., buy a ticket using online service of public transport company). To deal with the changes in user activities and plans, it is necessary to understand the impact of those changes on the current processes and state of the SBA (i.e., perform dynamic adaptation). Depending on the outcome, different adaptations may apply (e.g., compensate or re-compose some tasks, fail). Differently from automotive scenario where the business analysts are high-level domain experts, these decisions can not be delegated to the mobile user, as they may have no expertise on the low-level technical details of the SBA. Therefore, it is necessary to design such decision mechanisms that at run-time may reason on the specific situation in order to reveal an appropriate strategy and its parameters (e.g., to decide whether re-composition may be done, to derive concrete composition goal and the corresponding composition, etc.). In [6], in particular, such decision mechanism relies on the analysis of personal information of the user (e.g., context, agenda, tickets and reservations, etc.).

5 Conclusions

This paper proposes a design method for SBAs that targets the adaptation requirements of those applications and aims at overcoming the fragmentation in current approaches for SBA adaptation. The approach is based on a novel life-cycle that considers adaptation as a first class concern and that covers the different facets of adaptation, both during the design phase and at run-time. Admittedly, this paper is just a first step towards our ultimate goal of defining a holistic design method for adaptable SBAs. Still,

the effectiveness of such principles and guidelines is witnessed by their capability to capture the key aspects of adaptation in the different, heterogeneous real-world scenarios considered in this paper. Our future roadmap includes a refinement of guidelines and principles presented in this paper, their formalization into patterns, and the definition of more precise criteria to decide on the patterns that are most appropriate for a given adaptation need. We also intend to work on the development of mechanisms and tools supporting the methodology, building on top of the actions and artifacts identified in Figure 2. Finally, we intend to work on a stronger empirical evaluation of the proposed methodology, by applying it to the real-world scenarios we already exploited in this paper.

References

1. International Standard - ISO/IEC 14764 IEEE Std 14764-2006. pp. 1–46 (2006)
2. Baresi, L., Guinea, S., Pasquale, L.: Self-healing BPEL processes with Dynamo and the JBoss rule engine. In: ESSPE 2007, pp. 11–20. ACM, New York (2007)
3. Colombo, M., Nitto, E.D., Mauri, M.: Scene: A service composition execution environment supporting dynamic changes disciplined through rules. In: Dan, A., Lamersdorf, W. (eds.) ICSOC 2006. LNCS, vol. 4294, pp. 191–202. Springer, Heidelberg (2006)
4. Erradi, A., Maheshwari, P., Tosic, V.: Policy-driven middleware for self-adaptation of web services compositions. In: van Steen, M., Henning, M. (eds.) Middleware 2006. LNCS, vol. 4290, pp. 62–80. Springer, Heidelberg (2006)
5. Karastoyanova, D., Houspanossian, A., Cilia, M., Leymann, F., Buchmann, A.P.: Extending BPEL for Run Time Adaptability. In: EDOC, pp. 15–26 (2005)
6. Kazhamiakin, R., Bertoli, P., Paolucci, M., Pistore, M., Wagner, M.: Having Services "YourWay!": Towards User-Centric Composition of Mobile Services. In: FIS (2008)
7. Kongdenfha, W., Saint-Paul, R., Benatallah, B., Casati, F.: An Aspect-Oriented Framework for Service Adaptation. In: Dan, A., Lamersdorf, W. (eds.) ICSOC 2006. LNCS, vol. 4294, pp. 15–26. Springer, Heidelberg (2006)
8. Linner, D., Pfeffer, H., Radusch, I., Steglich, S.: Biology as Inspiration Towards a Novel Service Life-Cycle. In: Xiao, B., Yang, L.T., Ma, J., Muller-Schloer, C., Hua, Y. (eds.) ATC 2007. LNCS, vol. 4610, pp. 94–102. Springer, Heidelberg (2007)
9. Mittal, K.: Service oriented unified process, `http://www.kunalmittal.com/html/soup.html`
10. Rukzio, E., Siorpaes, S., Falke, O., Hussmann, H.: Policy based adaptive services for mobile commerce (2005)
11. Spanoudakis, G., Zisman, A., Kozlenkov, A.: A Service Discovery Framework for Service Centric Systems. In: IEEE International Conference on Services Computing, vol. 1, pp. 251–259 (2005)
12. Trainotti, M., Pistore, M., Calabrese, G., Zacco, G., Lucchese, G., Barbon, F., Bertoli, P., Traverso, P.: ASTRO: Supporting Composition and Execution of Web Services. In: Benatallah, B., Casati, F., Traverso, P. (eds.) ICSOC 2005. LNCS, vol. 3826, pp. 495–501. Springer, Heidelberg (2005)
13. Verma, K., Gomadam, K., Sheth, A.P., Miller, J.A., Wu, Z.: The METEOR-S Approach for Configuring and Executing Dynamic Web Processes. Technical report, University of Georgia, Athens (June 2005)
14. Verma, K., Gomadam, K., Sheth, A.P., Miller, J.A., Wu, Z.: The METEOR-S Approach for Configuring and Executing Dynamic Web Processes. In: Technical report (2005)
15. Zeng, L., Benatallah, B., Dumas, M., Kalagnanam, J., Sheng, Q.Z.: Quality driven web services composition. In: WWW, pp. 411–421. ACM, New York (2003)