Autom Softw Eng (2008) 15: 313–341
DOI 10.1007/s10515-008-0032-x

A journey to highly dynamic, self-adaptive service-based applications

Elisabetta Di Nitto · Carlo Ghezzi · Andreas Metzger · Mike Papazoglou · Klaus Pohl

Received: 30 July 2008 / Accepted: 1 August 2008 / Published online: 5 September 2008

Abstract Future software systems will operate in a highly dynamic world. Systems will need to operate correctly despite of unespected changes in factors such as environmental conditions, user requirements, technology, legal regulations, and market opportunities. They will have to operate in a constantly evolving environment that includes people, content, electronic devices, and legacy systems. They will thus need the ability to continuously adapt themselves in an automated manner to react to those changes. To realize dynamic, self-adaptive systems, the service concept has emerged as a suitable abstraction mechanism. Together with the concept of the service-oriented architecture (SOA), this led to the development of technologies, standards, and methods to build service-based applications by flexibly aggregating individual services. This article discusses how those concepts came to be by taking two complementary viewpoints. On the one hand, it evaluates the progress in soft-

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

E. Di Nitto (✉) · C. Ghezzi
Politecnico di Milano, DEI, Piazza Leonardo da Vinci, 32, 20133 Milano, Italy
e-mail: dinitto@elet.polimi.it

C. Ghezzi
e-mail: ghezzi@elet.polimi.it

A. Metzger · K. Pohl
University of Duisburg-Essen, SSE, Schützenbahn 70, 45117 Essen, Germany

A. Metzger
e-mail: andreas.metzger@sse.uni-due.de

K. Pohl
e-mail: klaus.pohl@sse.uni-due.de

M. Papazoglou
Tilburg University, INFOLAB, PO Box 90153, 5000 LE Tilburg, The Netherlands
e-mail: mikep@uvt.nl

ware technologies and methodologies that led to the service concept and SOA. On the other hand, it discusses how the evolution of the requirements, and in particular business goals, influenced the progress towards highly dynamic self-adaptive systems. Finally, based on a discussion of the current state of the art, this article points out the possible future evolution of the field.

Keywords Service-oriented computing · Services · Adaptive systems · Self-adaptation

1 Introduction

Organizations have been evolving over the past decades from hierarchically structured, stable, monolithic, and centralized to distributed and dynamically federated entities. Such federations are more and more achieved by integrating and composing business services that are individually offered by each organization. The structure of such compositions is highly dynamic as it needs to optimize business performance and to adapt flexibly and rapidly to the evolution of business goals and requirements.

Consistently with the trend occurring at the business level, the underlying software systems that support the organizations' business processes have evolved from centralized and monolithic to highly distributed. Indeed, software systems are more and more required to be flexible, dynamic, and adaptive.

Such requirements are not only specific to software systems that support business processes, but they also become relevant in other domains such as pervasive or ubiquitous computing. In pervasive computing, the vision is that the physical environment is populated by electronic devices that interact with users. A pervasive computing application supports behaviors that are intrinsically context-dependent, where *context* captures the notion of information about the application physical environment that can trigger certain changes in the application. As an example, the physical position of a mobile user at a given time may trigger that the system switches from a certain communication platform (e.g., Wifi) to another (e.g., GSM). Also, the technologies used to detect the position of an entity—human or thing—may change from GPS, when the entity is outdoors, to Wifi-based or tag-based, when the entity is indoors. As in the case of federated business organizations, the capability of a software system to be distributed, dynamic, and adaptable to different situations is a crucial requirement of pervasive scenarios.

To address the above requirements, *software services* have emerged as a suitable abstraction mechanism. Software services are not just pieces of software; instead, they represent the functionality that the underlying pieces of software offer. Rather than building a software system from scratch, or developing it by selecting and gluing together off-the-shelf components, now designers can realize applications by composing services, possibly offered by third parties. This shift from adopting the piece of technology (the software) to using the functionality (the service) offers us a valuable tool to design those software systems that we call *service-based applications* at a higher level of abstraction, possibly building new value-added composed services.

In this context, the term *Service Oriented Architecture* (SOA) typically comprises a set of guiding principles for building service-based applications. These principles include loose coupling between services, coarse grained service interfaces, dynamic service discovery and binding, self-containment of services, service interoperability, and protocol independence (Erl 2004; Kaye 2003; Josuttis 2007). The software services used in a composition can be discovered and selected dynamically while the composition is running. This enables on the fly adaptation to changes. Thus, service-based applications, constitute a promising solution to realize dynamically adaptable distributed systems.

To take full advantage of dynamic adaptation, it should be possible to modify a service-based application in an automated manner. *Automation* is needed to support continuous change of functionality and/or quality of service. It takes the form of *self-adaptivity*, requiring that applications should automatically and autonomously adapt their behavior to respond to evolving requirements, changes in its context, as well as failures of component services. The application should be equipped with capabilities that support deciding on when and why it should perform an adaptive change. For instance, adaptation could be triggered when a new interesting service—more convenient than the one in use—is published, as this may indicate a change in the business environment. Indeed, adaptation would be necessary when the application is using services whose availability depends on the location of the final user and the user moves from one place to another.

The above vision of self-adaptation, is not yet fully supported by today's software technologies and methods. The purpose of this article is therefore to understand where we are, review the achievements that led to the current state of the art, and highlight the open issues and challenges that require further research, where we envision that significant contributions could come from the automated software engineering community.

The article is organized as follows: Sect. 2 focuses on the evolution of software technologies and methodologies that led to the current state of the art, which is then scrutinized in Sect. 3. Section 4 presents some intriguing application scenarios in the area of federated organizations and pervasive computing and shows how service-based applications could be used to address them. Focusing on self-adaptable systems, Sect. 5 discusses important open issues and research challenges to be addressed. Finally, Sect. 6 draws some general conclusions.

2 Towards highly dynamic, self-adaptive applications: a historical perspective

The evolution of software technology and methods can be seen as a progressive journey toward highly dynamic, self-adaptive applications. The following sub-sections provide a brief historic account of this evolution and its major mile-stones.

2.1 "Genesis": rigid processes and static, monolithic systems

The journey started between the late 1960's and early 1970's with the attempts to discipline the software process through the identification of well-defined stages and criteria to be met to progress from one stage of the process to the next. The goal was to avoid endless iterations of code-and-fix activities, improve predictability of the

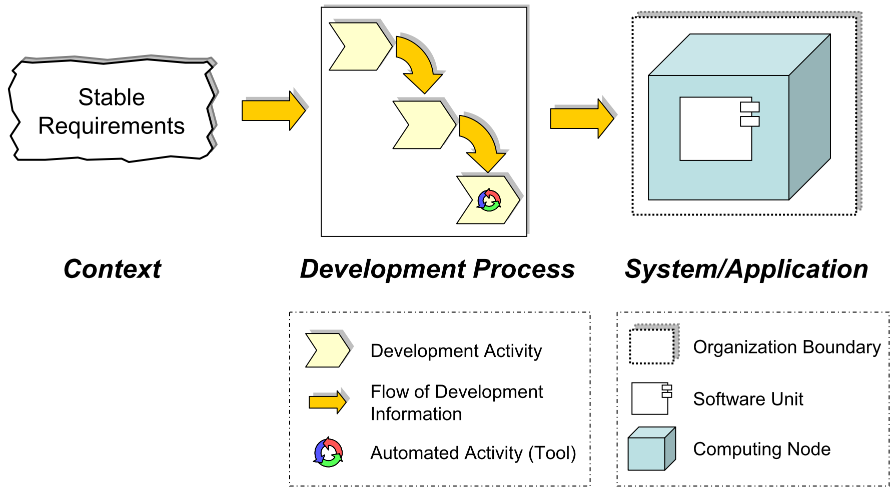

Fig. 1 Rigid processes and static, monolithic systems

development process, improve product quality, and reduce costs. Continuous changes to remove errors, to meet the customers' and users' expectations, and to improve the implementation were felt as the main culprits of poor software quality.

The waterfall development process (sketched in the middle of Fig. 1), which was proposed by Royce (1970)—after software engineering was "officially" born in 1968 (Naur and Randell 1968)—was an attempt to bring the attention to the needed discipline and predictability to software development. It clearly advocated the requirement for software developers to focus not only on coding, but also on higher-level activities (requirements analysis and specification, as well as software design) and on verification and validation.

It was also argued that careful requirements analysis could avoid later costly changes, and improve software quality. Software changes were considered harmful because they were viewed as the cause for disruption in the disciplined progress of the development process, and ultimately the cause for schedule slips and cost overruns.

These early approaches were based on a number of fundamental implicit assumptions about software and the context (the world) in which software was going to operate. At the time, the world was mostly fixed and static, meaning that it could be assumed that *stable requirements* existed, ready to be discovered and captured by carefully analyzing the system context and eliciting the stakeholders' needs. Requirements were supposed to be stable (see left hand side of Fig. 1), as were most of the organizations at the time. The recommended processes insisted on delaying design and implementation until an exhaustive elicitation and specification of the requirements had been performed.

Furthermore, organizations were mostly monolithic, and solutions to their needs were—accordingly—monolithic and centralized (see right hand side of Fig. 1). The main goal was to improve the efficiency of internal activities of organizations; interactions among organizations were minimal and were not perceived as critical.

Baresi et al. used the term *closed world assumption* to characterize the implicit hypotheses that underlie these initial approaches to software engineering (Baresi et al. 2006). The development process and the structure of software products were, static, and monolithic. In the case of complex systems decomposed into modules, the modules were supposed to be statically bound and the resulting "frozen" application was statically deployed on a physically centralized computing architecture.

The tools employed by the software engineers at the time were mostly assemblers, compilers and debuggers, which were used during the implementation phase (see the central part of Fig. 1). Although the foundations for automated software verification had been established, the computational power and efficiency of the algorithms was not yet capable of addressing problems of practical relevant size. These assumptions were valid at the time, but soon became outdated.

2.2 "Enlightenment": agile processes and dynamic system structures

It was soon realized that the closed world assumption was not realistic. In most practical cases, requirements cannot be fully gathered upfront and cannot be "frozen". It was realized that—due to the many stakeholders involved—requirements are conflicting and intrinsically decentralized. It became clear that often stakeholders do not know beforehand exactly what they expect from a system. Thus, an exhaustive gathering of requirements and a complete pre-plan of the process to avoid future changes are illusory: changing requirements (see left hand side of Fig. 2) are not a nuisance to avoid, but an intrinsic factor that must be dealt with.

The history of software engineering shows a progressive departure from the strict boundaries of the closed world assumption toward more flexibility to support continuous evolution. This becomes clear already in the early 1970's when Parnas introduces the idea of design for change (Parnas 1972). From there on, systems progressively evolved from fixed, static, and centralized to adaptable, dynamic, and distributed systems. Methods, techniques, and tools were developed to support the need for change without compromising software quality and cost.

The evolution has been at the level of both the development process and the product. Evolutionary process models—such as incremental and prototyping-based

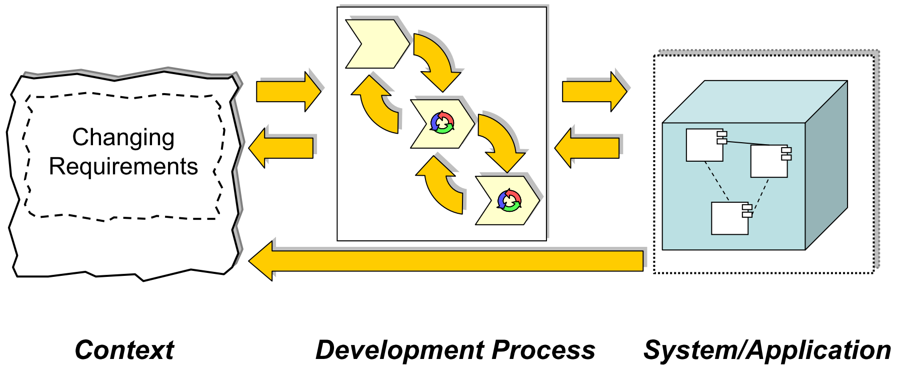

Fig. 2 Agile processes and dynamic system structures

models—were introduced to achieve better tailoring of solutions to customer and user needs, and to reduce risks. More recently, these evolved into agile methods, like extreme programming. Tracing all the steps of this evolution would go beyond the scope of this article, but we want to highlight two important trends in many process models proposed so far: (1) the idea of continuously iterating through the various phases of the development process (as illustrated by the "feedback" arrows in the middle of Fig. 2); (2) the establishment of increasingly connections with end users and customers.

Static, centralized, and monolithic systems implied (partial) recompilation and re-deployment of the software in order to address changes. Modular and distributed architectures (see right hand side of Fig. 2) led to design methods that better support the change of the software architecture. The principles of information hiding, encapsulation, and separation of a module interface from its implementation (Parnas 1972) were eventually embedded in new programming languages to enforce good design practices. In particular, it is interesting to notice how object-oriented programming languages, like C++ and Java, can safely accommodate certain anticipated changes in the implementation. If changes to an existing module (a class) can be cast into the subclass mechanism, changes can be made even while the system is running. Thanks to polymorphism and dynamic binding (Meyer 1997), a client invocation of a certain operation (method) may be bound dynamically to an operation redefined in a subclass without affecting the implementation of the client.

To support prototyping-based development models, tools that were able to generate executable code from models or higher-level languages were introduced (illustrated in the middle of Fig. 2). An example is the STATEMATE tool suite conceived in the 1980's (Harel et al. 1990).

2.3 "Industrialization": automation of software development and distributed systems

The evolution of software technology allowed bindings among modules not only to become dynamic, but also to extend across network boundaries and thereby enable the distributed execution of the software (see right hand side of Fig. 3). In Java, for example, it is possible to invoke a method on a remote server object, running on a different computing node in the network.

Middleware had a crucial role in extending communication between objects outside the machine on which they are running (Emmerich et al. 2008). The concept of a Remote Procedure Call (RPC) was the first simple way to call a remote procedure in a similar fashion as if it was a local procedure. It has been followed, among others, by CORBA that introduced the concept of remote objects. Other interesting middleware concepts that have enabled the possibility of communicating with some remote modules without being bound to them are message queues. These completely decouple the sender of a request, which simply posts it in a queue, from the receivers, that observe the queue and, according to some scheduling algorithm, select requests from the queue. While they are still introducing a centralization point (the queue) into the architecture, distributed publish/subscribe middleware (e.g., see Cugola et al. 2001) has overcome this issue as well, thus enabling the possibility of developing highly dynamic and distributed systems.

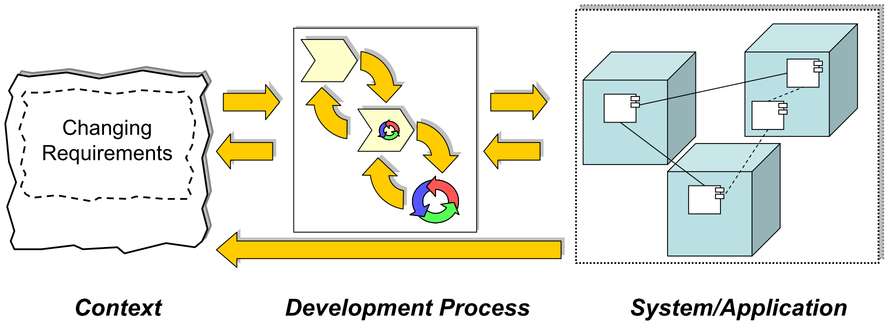

Fig. 3 Automation of software development and distributed systems

The extent of the automation of software development activities (illustrated in the central part of Fig. 3) significantly increased in that period, supported by the availability of a relevant amount of processing power. There was a strong interest in the area of CASE (Computer Aided Software Engineering) tools and CASE tool integration (Fuggetta 1993). A more recent trend toward automation concerns Model-Driven Architecture (MDA) (Miller and Mukerji 2003), which aims at generating code from high-level architectural models. Further, automatic verification techniques and tools have reached a level of maturity that allows them to be applied to real-life problems. Examples are model checkers, like SPIN or Alloy, or Boolean satisfiability checkers (SAT solvers), like SAT4J or zChaff.

2.4 "Globalization": shared ownership and adaptive systems

Another major evolution path was in terms of the ownership of an application. In the beginning, system development was under the control of a single organization, which ultimately owned it completely. Next, component-based software development became dominant. Off-the-shelf components started to be developed and provided by third parties, who are also responsible for their quality and their evolution. Application development thus became (partly) decentralized. At an extreme, application development boiled down to gluing components together.

The demand for software to live in an open world and to evolve continuously as the world evolves, however, is now reaching unprecedented levels of dynamism. In particular, over the past years a further major step in the evolution was the birth of the *service* concept and the *service-oriented architecture* (SOA) paradigm (Erl 2004; Kaye 2003; Josuttis 2007). This led to the development of technologies and standards to build *service-based applications* by flexibly aggregating individual services into service compositions.

Services have taken the concept of ownership to the extreme: not only, as off-the-self components, their development, quality assurance, and maintenance are under the control of third parties, but they can even be executed and managed by third parties (see right hand side of Fig. 4). Similar to what has been enabled by globalization in the physical world, software services enable organizations to flexibly outsource

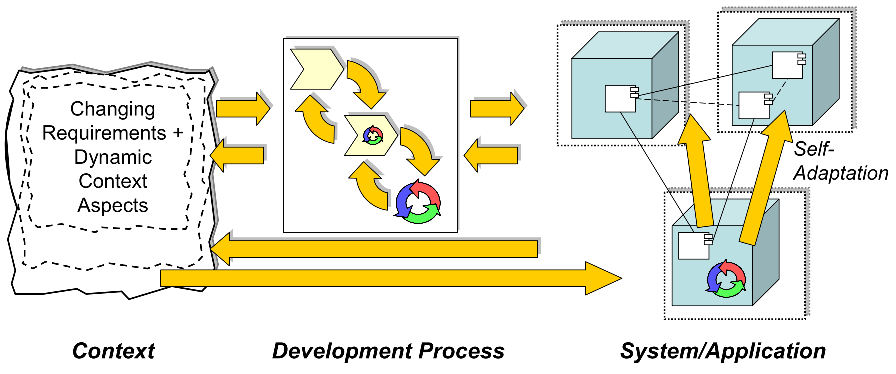

Fig. 4 Shared ownership and adaptive systems

business functions (typically commodity functions) and to focus on the innovative functions, which differentiates one organization from another.

Finally, the success in automating software engineering activities like code generation or automatic verification, started to allow applications to automatically monitor their own behavior and context and to trigger self-adaptation to compensate for deviations from their expected behavior (see right hand side of Fig. 4).

In conclusion, the evolution of software development methods and tools made it possible to support seamless evolution of the product structure to incorporate certain anticipated changes, such as additional and/or redefined module functionalities, and to support decentralization and distribution. Software development processes evolved from sequential, centralized, and monolithic to iterative, decentralized, and distributed processes. Applications can now be designed in such a way that changes can be incorporated without violating the overall correctness, by carefully decoupling modules via well-defined interfaces. Thus, it can be argued that software engineers were quite successful in moving from a strict closed world situation toward more flexibility.

2.5 Observations

Several paths have been concurrently followed along the journey to highly dynamic, self-adaptive service-based applications. Those include:

- understanding that the world (context) is not static but constantly changing and that closed world assumptions are thus not realistic;
- moving away from monolithic and centralized systems to distributed systems with dynamic structures;
- changing from single ownership of systems to shared ownership of the software units;
- raising the levels of automation and tool support from assemblers and compilers to code generation and automatic verification;
- moving from rigid development processes to "agile" ones.

Next generation service-based applications will adapt themselves in an automated manner to unexpected changes of, for example, environmental conditions, technology, regulations, and market opportunities. They will function within a mixed environment of people, content and systems. Service-based applications will thus possess the ability to continuously adapt and re-configure themselves in reaction to unexpected context changes and changing stakeholder requirements.

In addition, the next generation of service-based applications will have to be able to predict problems, such as potential degradation scenarios, future erroneous behavior, and deviations from expected behavior, and move toward resolving them not just automatically, but even *proactively*, i.e., before they occur. Service-based applications are thus driven by requirements that bring the concepts of decentralization, dynamism, adaptation, and automation to an extreme.

Starting from what exists today, the challenges involved in building, evolving, and maintaining such highly dynamic, self-adaptive applications and the potential directions for new research—including automated software engineering—are explored in the remainder of this article.

3 Current technology and methods for service-based applications

Service-based applications encompass three layers of abstraction as shown in Fig. 5.

The *service infrastructure* layer supports describing, publishing, discovering, and binding services. It also provides the run-time environment for the execution of service-based applications as well as primitives for service communication, mechanisms for connecting heterogeneous services, facilities for service description, and capabilities that support the discovery of services.

The *service composition and coordination* layer supports the (hierarchical) aggregation of multiple (individual) services into service compositions. Service compositions can—in turn—be offered to service clients and be composed into service-based applications, also called *service compositions*. By relying on the service infrastructure layer, the service composition and coordination layer is responsible for control-

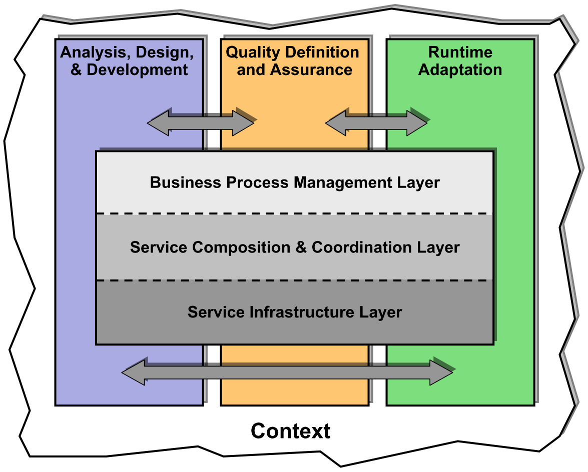
Analysis, Design, & Development
Quality Definition and Assurance
Runtime Adaptation
Business Process Management Layer
Service Composition & Coordination Layer
Service Infrastructure Layer
Context

Fig. 5 Layers and aspects relevant for service-based applications

ling and coordinating the execution of the aggregated services of a composition and for managing the data flow as well as the control flow between those services.

The *business process management* layer provides end-to-end visibility and control over all parts of a long-lived, multi-step business process that spans multiple organizations and can involve human actors. The business process management layer provides mechanisms for expressing, understanding, representing and managing an organization in terms of a collection of business processes which are realized in a service-oriented fashion.

During the development and the operation of a service-based application three main activities are performed, which are concerned with all the above layers of service-based applications (see Fig. 5). These activities are (1) *analysis, design, and development*; (2) *quality definition, negotiation, and assurance*; and (3) *run-time adaptation*. Quality definition and negotiation require techniques and methods that are significantly different from the ones needed for quality assurance. Thus, those two issues will be treated separately in the following.

Figure 5 also highlights the concept of *context*, which surrounds the service-based application, as an essential aspect that needs to be considered. Context is, in fact, the main driver of dynamic adaptation strategies, as we discussed in Sects. 1 and 2.

In the following sub-sections, we will briefly discuss the current state of the art in each of the layers and activities relevant for a service-based application.[1] This will help us to understand the gaps in existing technologies and methods that need to be addressed to enable highly dynamic, self-adaptive applications (see Sect. 5).

3.1 Business process management

Business Process Management (BPM) (Burlton 2001) is an IT-enabled management discipline that treats business processes as assets to be valued, designed, and enhanced. BPM technologies support both human-centric processes (like claim processing or customer servicing) and system-intensive processes (like trade settlement), as well as a mixture of both (like loan granting).

In the past few years, BPM evolved from its early roots in workflow management to a more comprehensive area that addresses graphical process design, process execution, as well as process monitoring and reporting capabilities for human-centric processes. In that respect BPM comprises more than traditional workflow management as it adds conceptual innovations and technology from Enterprise Application Integration and Business-to-Business integration and re-implements it on an e-Business infrastructure based on Web and XML standards.

The objective of BPM is to manage the life cycle of a process starting from business goals over process definition, through deployment, execution, measurement, analysis, change, and redeployment. The management of business processes includes process analysis, process definition and redefinition, resource allocation, scheduling, measurement of process quality and efficiency, and process optimization. Process

[1]This overview of the state of the art builds on comprehensive state of the art surveys compiled within the *S-Cube* Network of Excellence. These surveys are available as project deliverables from http://www.s-cube-network.eu/.

optimization includes collecting and rendering real-time measures/metrics (monitoring), interpreting past measures/metrics (process analysis and data mining) and strategic measures/metrics (performance management), and correlating them as the basis for process improvement and innovation. Further key aspects that are considered in the BPM literature are business process discovery (van der Aalst 2005) and business transactions (Papazoglou and Kratz 2007).

One important issue in BPM is auditing and adapting business processes such as to ensure that they are compliant to policies, business rules, and regulations, like Basel II or Sarbanes-Oxley (SOX). In a broader perspective, compliance can pertain to any explicitly stated rule or regulation that prescribes any aspect of an internal or cross-organizational business process, including for example public policies, customer preferences, partner agreements and jurisdictional provisions. Currently, compliance to such rules and regulations is typically achieved on a per-case basis. Although such ad-hoc solutions achieve their objective, from a management perspective they have several undesirable characteristics as they are hard to maintain, reuse, evolve, and formally verify. An important characteristic of service-based applications is that they are impacted heavily by industry and sectorial regulations. Without explicit business process definitions, flexible rules frameworks, and audit trails that provide for non-repudiation, organizations thus could face litigation risks.

3.2 Service composition and coordination

Service composition and coordination issues have been addressed to some extent by both industry and academia.

For what concerns languages to specify service composition and coordination, competing—mostly industry-driven—initiatives exist. The terms *orchestration* and *choreography* have been widely used in this context to describe interaction protocols involving collaborating services.

Orchestration describes how services can interact with each other at the message level, including the business logic and execution order of the interactions from the perspective and under control of a single endpoint. With orchestration, the business process interactions are always specified from the local (private) perspective of one of the business parties involved in the process. Orchestration is targeted by a family of XML-based quasi-standard languages, the most representative of which is the Business Process Execution Language (BPEL) for Web Services (Andrews et al. 2003).

Choreography is typically associated with the global (public) message exchanges, rules of interaction and agreements that occur between parties. Choreography is more collaborative in nature than orchestration. It is described from the perspectives of all parties (common view) and specifies the complementary observable behavior between participants in a business process collaboration. Choreography offers a means by which the rules of participation for collaboration can be clearly defined and jointly agreed upon. Service choreography is targeted, for instance, by the Web Services Choreography Description Language (WS-CDL) (Kavantzas 2004).

This sharp distinction between orchestration and choreography is rather artificial and the consensus is that they should both coalesce into a single language and environment.

On the research front, further activities mainly concentrate on dynamic compositions (Yang and Papazoglou 2004), on modularizing compositions (Charfi and Mezini 2004; Yang and Papazoglou 2004), as well as on enhancing service descriptions (with, for instance, compositional assertions or semantic descriptions) such that compositions can be assessed and formally verified (Solanki et al. 2004; Baresi et al. 2007). In addition, there has been some work in the area of Automated Planning, which includes automating the retrieval and composition of Web services (e.g., Traverso and Pistore 2004; Pistore et al. 2005; Lazovik et al. 2006), as well as the verification and monitoring of service-based applications (e.g., Kazhamiakin and Pistore 2006; Baresi and Guinea 2005; Bianculli et al. 2007). However, these efforts are still either at the conceptual or at a preliminary stage of development.

Many of the existing approaches toward service composition largely neglect the context in which composition takes place. It is only recently that research approaches have focused on developing context-aware approaches that take into account the business and social context of service compositions as the basis for process specification and verification (e.g., see Colombo et al. 2005).

3.3 Service infrastructure

Research on service infrastructures is performed by the service-oriented computing (SOC) community and the grid computing community.

Research in the SOC community has been coalescing into the concept of the Enterprise Service Bus (ESB) (Papazoglou and Georgakopoulos 2003); ESBs provide the distributed processing, standards-based integration, and enterprise-class backbone required by federated enterprises. They offer support to the life cycle of services, their deployment, and their composition. ESBs usually support the coexistence of both synchronous and asynchronous communication mechanisms. Asynchronous mechanisms, in particular, are thought as the way to support loosely coupled integration in a business-to-business context that crosses the boundaries of a single organization. Examples of existing ESBs are the open source Mule[2] and the IBM WebSphere Message Broker (IBM-MB 2008).

Grid computing is a way of organizing large numbers of networked computers belonging to diverse organizations so that they can be flexibly and dynamically allocated, accessed, and used as an ensemble to solve massive computational problems that require many collaborating resources (Foster and Kesselman 2004). Grid computing involves sharing heterogeneous resources (based on different platforms, hardware/software architectures, and programming languages) over a network using open standards. Those resources can be located in different places and possibly belong to different administrative domains. The vision of grid computing is that distributed resources can be managed as a single virtual system, irrespective of whether services are confined to a single enterprise or whether they extend to encompass external service providers. The management of those resources means that these can be discovered, accessed, allocated, monitored, accounted for, billed for, and so on.

[2] http://mule.mulesource.org/display/MULE/Home.

The grid community has developed the Open Grid Services Architecture (OGSA) in a service-oriented grid computing environment (Foster et al. 2002). The OGSA is intended for business and scientific purposes and addresses the need for standardization by defining a set of core capabilities and behaviors that represent key concerns in Grid systems. Grid services resemble Web services especially from a programmatic point of view. A grid service is (in practice) a Web service that conforms to a particular set of coding practices, namely, a set of XML coding conventions in the form of standards. For example, grid services can be defined in terms of the standard Web Service Description Language (WSDL), with perhaps some minor language extensions. These grid services can now take advantage of Web service technologies and standards for messaging, reliable communicating, providing standardized access and management of resources, addressing, notification, transactions, and security (these standards include SOAP, WS-ReliableMessaging, WS-Resource Framework, WS-Addressing, WS-Notification, WS-Transaction, and WS-Security).

3.4 Analysis, design, and development

The aim of research on the analysis, design, and development of service-based applications is to offer suitable principles, life cycle models, methods, and tools. A number of key principles for service-based applications have emerged and are reflected in the *Service-Oriented Architecture (SOA)* paradigm (e.g., see Erl 2004; Kaye 2003; Josuttis 2007). These principles are loose coupling between services, coarse grained service interfaces, dynamic service discovery and binding, self containment of services, service interoperability, and protocol independence.

Different life cycle models are being suggested both by industry and academia (e.g., see Arsanjani 2004; Papazoglou 2007). Important aspects of these life cycle models are the attention to the business process perspective of services and the translation of such a business perspective into specific software services. Most importantly, due to the dynamic nature of services and service-based applications, such life cycle models exercise special focus on the provisioning and operation phase. There is however one aspect that does not seem to be completely captured yet and indeed has an impact on the life cycle, namely, the fact that the services in a service composition can be owned by different stakeholders. This implies that service-based applications have to be built in a way that makes them resilient to changes and failures of their component services (e.g., when such services become unavailable). Thus, aspects concerned with on-the-fly discovery of new services, with the definition and refinement of Service Level Agreements (SLAs), and with the need of explicitly considering adaptation strategies as part of the life cycle are to be deeply analyzed. As an example, Colombo et al. (2006) show an example of an approach that partially addresses these issues.

Several methods and techniques addressing specific phases of the life cycle have been developed. An example for a general purpose approach that has been adopted from the software engineering discipline is the goal modeling approach i* (Antón 1996), which can be used mainly during the analysis phase to identify the objectives and the roles within a service-based application and the dependences among roles.

A service-specific analysis approach has been proposed by Zachos et al. (2007). The authors illustrate a novel technique based on the idea that the availability of services within a well stuffed registry can help in the requirements analysis phase, since it allows new requirements to emerge simply because the analyst finds these services and realizes that they could be beneficial to extending or improving the functionality of the application under development. This means that in a SOA context it is quite natural to build service compositions in a "bottom-up" way from the available services.

In the current state of practice, the control and data flow between the aggregated services in a service composition is designed and coded in some executable language (typically BPEL). Yet, coding is an arduous procedure that requires experience, is error-prone, and needs to be repeated each time that a slightly different application needs to be constructed. More expressive and higher-level notations should be investigated with the goal of supporting end-user driven service compositions.

The deployment and operation phases have to be considered under a perspective different than the one adopted in traditional software engineering. Deployment is to a certain extent simpler since it does not have to account for the installation of each single service, which is usually already running and operated by its provider. However, it has to ensure that the system satisfies all constraints defined by the aggregated services and that the continuous monitoring mechanisms needed to control and adapt the execution of the system are put in place. In the operation phase, self-adaptation and run-time discovery and binding of services should be enabled and supported thus requiring new and smart execution infrastructures. This aspect will be described in more detail in Sect. 3.7

3.5 Service quality definition and negotiation

Contracts between parties enable the creation of precise links between service consumers and service providers. In addition to functional aspects, contracts should deal with quality aspects of the service provision (Curbera 2007). While the issue of how to achieve such contracts through manual and automatic negotiation has been deeply studied within the domain of e-commerce, the problem is largely unexplored in the case of software services. Here, the objective of the negotiation is the definition of Service Level Agreements (SLAs), which are more or less formal contracts that discipline the way services are provided to consumers and, in turn, state the obligations to be fulfilled by the consumers to obtain the services.

Negotiation of SLAs can either be performed directly by the involved parties or it can be automated. In the latter case, human beings are replaced by automated negotiators, which try to achieve the objective that has been assigned to them. Automated negotiation is particularly important when the consumer of a service is a software system that has to negotiate—on the fly—the SLA with the service.

Research on automated negotiation has mainly produced architectural solutions for negotiation (see for instance Chhetri et al. 2006; Rolli et al. 2004), as well as algorithms and models for protocols and negotiation decision strategies (see for instance Comuzzi and Pernici 2005).

Among the others, Di Nitto et al. propose a flexible infrastructure supporting various protocols (e.g., peer-to-peer and auction-based negotiation) and decision strategies for negotiation (Di Nitto et al. 2007).

In all the aforementioned approaches, the focus lies on negotiating QoS properties like performance or cost (Jaeger et al. 2004). However, the impact of service compositions on the negotiation of SLAs with each single component service requires further investigation. Indeed, it is still not clear how SLAs are maintained and created, nor how the relevant quality factors to be agreed are identified and elicited, and how they can be quantified in a meaningful way to make them measurable and monitorable across all layers in Fig. 5. Finally, the role of social and cultural issues as well as strategic partnerships within SLA negotiation deserve further attention.

3.6 Service quality assurance

To assure that a service-based application meets its expected quality, two complementary strategies can be followed:

- *Constructive quality assurance*: The goal of constructive quality assurance is to prevent the introduction of defects while artifacts are being created. Examples of such techniques include code-generation (model-driven development), development guidelines, and templates.
- *Analytical quality assurance*: The goal of analytical quality assurance is to uncover defects in an already existing artifact. Three major classes of analytical quality assurance techniques for service-based applications can be identified: testing, monitoring, and static analysis (see Baresi and Di Nitto 2007).

Most *testing* approaches for service-based applications addressed the problem of generating and automatically executing test cases (e.g., see Dai et al. 2007; Bai et al. 2007b; de Almeida and Vergilio 2006; Heckel and Mariani 2005). This includes deriving test cases from service descriptions (black box tests) and from service compositions (white box tests). Several other approaches aim at providing solutions to support regression testing of service-based applications (e.g., see Di Penta et al. 2007; Ruth and Tu 2007). Regression testing aims at checking whether changes of (parts of) a system negatively affect the existing functionality of that system. Regression testing is essential due to the fact that service compositions can change in response to the evolution of the aggregated services or to other changes in the context of the service-based application. Yet, although dynamicity has been acknowledged as an important characteristic of service-based applications and many of the testing techniques support the automatic generation and execution of test cases, there are only few initial contributions that propose to perform online tests, i.e., to execute tests during the operation phase of the service-based application (Deussen et al. 2003; Bai et al. 2007a). An interesting approach in this area is proposed by Bertolino et al. They extend a registry in a way that it can behave as an accredited testing body by performing an “audition testing” on new services (Bertolino and Polini 2005).

Monitoring observes running services or service-based applications in order to, for example, detect deviations from the expected behavior, support the optimization of a service-based application during run-time, or enable its context-driven adaptation. Existing monitoring approaches mainly focus on events (information sources) that are internal to the service-based application (e.g., see Ghezzi and Guinea 2007; Pistore and Traverso 2007; Benbernou et al. 2007; Spanoudakis et al. 2007; Rozinat and van der Aalst 2008). There are currently a few approaches that deal with a key aspect needed to enable self-adaptive service-based applications; namely, observing the context of the application. Also, most approaches focus on specific layers of a service-based application in isolation (see Fig. 5) but they lack the ability to cross-cut all the layers of a service-based application.

During *static analysis*, artifacts (e.g., a service specification) are systematically *examined*—without execution—in order to determine certain properties or to ascertain that some predefined properties are met. Most static analysis approaches for service-based applications build on analysis techniques and methods from software engineering. Many of the state of the art approaches thus include classic verification technology, such as model checking, applied to the service oriented world (e.g., see Bultan et al. 2007; Fu et al. 2004; Bianculli et al. 2007; Foster et al. 2006). Other approaches address the problem of analyzing non-functional quality attributes (e.g., see Marzolla and Mirandola 2007). None of these approaches, however, provides means for analyzing the adaptation behavior of a service-based application, which—of course—could be faulty.

3.7 Run-time adaptation

Run-time adaptation is performed to achieve different goals, including (1) overcoming the mismatches between aggregated services, (2) repairing faults, or (3) reconfiguring the applications in order to better meet their requirements. *Self-adapting* service-based applications automatically, autonomously, and dynamically adapt to changes in their execution environment and/or in their requirements. Such changes could include the deployment of new instances of a particular kind of service, the removal of existing ones, and even more global changes in the required application. Self-adapting service-based applications should be able to function despite changes in the behavior of the services they use, are they controlled internally or offered by external service providers. Moreover, they should reduce as much as possible the need of human intervention for adapting services to subsequent evolutions. Self-adaptation requires that service-based applications are capable of automatically discovering new services, automatically choosing among available service providers, selecting from different options available for contracts (see Sect. 3.5), and automatically aggregating services into service compositions.

In order to realize self-adaptive behavior, control loops are established that collect details from the application and its context and act accordingly (Kephart and Chess 2003; Kreger et al. 2005; Leymann 2006). In the case of self-adaptive applications, control loops can address different goals, like *self-healing*, *self-optimizing*, and *self-protecting* actions.

Self-healing compositions can discover, diagnose, and react to disruptions in a service-based application due to failures of its individual services. They can detect system malfunctions and initiate corrective actions.

Self-protecting compositions can anticipate, detect, identify, and protect against threats. Self-protecting service-based applications can detect hostile behaviors as they occur (e.g., unauthorized access and use, virus infection and proliferation, and denial-of-service attacks) and take corrective actions to make themselves less vulnerable. Self-protecting capabilities allow business processes to consistently enforce security and privacy policies.

Self-optimizing services can monitor and tune individual services in the composition automatically. Self-optimizing service-based applications can thus tune themselves to better meet end-user or business needs. The tuning actions may imply, for instance, choosing new service providers to improve overall utilization.

One key shortcoming of today's approaches is that they only perform simple automated tasks. Furthermore, they do not comprehensively deal with the issue of adaptation across all the functional layers of a service-based application (see Fig. 5). Yet, taking such a cross-layer view on adaptation is essential, as otherwise conflicts between the local adaptations in the individual functional layers can occur. As an example, conflicting adaptations could be concurrently triggered by deviations in the BPM as well as in the infrastructure layer leading to unpredictable results. Another major shortcoming of current implementations of service-based applications is that the adaptation of an application occurs in a re-active fashion, i.e., the adaptation is triggered once a deviation from the application's requirements or a change in its context has been observed. This has significant drawbacks: first, it takes time to remedy the situation, which can lead to user dissatisfaction when having to wait for the application to become available again; second, the deviation can lead to undesired consequences, like loss of money.

3.8 Context

There is still no common agreement in the literature about the definition of *context*. We follow a generic definition proposed by Dey: "Context is any information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application" (Dey 2001).

The awareness of the context has an important role since the early phases of system development. In the area of requirement engineering, the Problem Frames approach proposed by Jackson (2000) is worth of being noted. It allows for a systematic analysis of (1) the desired properties of the context (world) that are to be achieved by introducing the system ("machine"), (2) the assumed properties of the context the system can rely on, and (3) the behavior of the software system (the "machine") to realize the required change in the context. Another approach to capture the context is exploiting so called context facets (e.g., see Mylopoulos et al. 1990; Jarke and Pohl 1993): The subject (or object) facet describes the domain in which the system is embedded, e.g., the organizational environment. The system facet describes—at different levels of abstraction—what the system does. These levels may include functional and quality (non-functional) requirements as well as design and implementation. The usage facet describes how the system is used and by whom. The development facet comprises information about the software development process and project management.

One key capability of pervasive and adaptive computing systems is the ability to collect context information in order to react automatically in ways that depend on the context situation in which they are currently operating (Abowd et al. 2002; Dey 2001). Such systems mostly use sensors to collect information about their *physical environment* in order to, for instance, identify the current location or determine the current activities of its users. However, the physical environment reflects only parts of a system's context.

Interestingly, currently, only a few service-specific approaches explicitly address context issues. Examples are first approaches for context-aware adaptation (e.g., see Batini et al. 2007; Pernici 2006) or techniques for context-aware discovery of services (e.g., see Spanoudakis et al. 2007). Yet, all these approaches either assume stable context boundaries or require humans to prescribe the potential context situations. This presents severe limitations for truly self-adaptive service-based applications that can autonomously respond to unforeseen situations.

4 Today's technology and methods vs. application scenarios

In this section we present two application scenarios, illustrate how service-based applications could address them, and identify some open issues. The two scenarios are derived from the domain of business-to-business interaction and federation of enterprises, as well as from the pervasive computing domain.

4.1 Federated organizations

Enterprises and other organizations can federate for various reasons, either because they want to cooperate on the development of some novel product or service, or because they belong to the same value chain and want to strengthen their interaction channel. The scenario that follows belongs to this second category.

A wine producer selling its products on different markets federates with shipping organizations, retailers, and even specific grocery stores and/or restaurants. This federation offers to the various actors several advantages:

- The wine producer can monitor the wine conditions even after the wine has been shipped. In particular, the producer can check whether the wine temperature has always remained within a certain range.
- Grocery stores and restaurants can offer customers the possibility of tracing the history of the product.
- The wine producer can receive feedback from the final consumers, as well as general statistics about sales from stores and restaurants.

Given the need for interaction across the boundaries of single organizations, a service-based approach offers a good solution to this problem. All functional layers that have been presented in Sect. 3 as well as the corresponding technologies can play a relevant role here. For instance, monitoring the wine conditions can be defined in terms of a business process that crosses the wine producer domain requiring an interaction with the other actors and domains. Such a business process can be implemented as the composition of the services for wine inspection offered at the sites of the various actors. In turn, the composition has to rely on a service infrastructure that has to incorporate not only the information systems of the federated enterprises, but also small devices such as temperature sensors and RFID (Radio Frequency Identification) tags that can temporarily store part of the data acquired by sensors.

Of course, all the vertical cross-cutting activities introduced in Sect. 3 are also important in this case. Among others, self-adaptation is needed to allow the system to reconfigure on the basis of the movements of the bottles of wine and of the specific retailers and other actors that are currently in contact with the wine producer. Such adaptation not only concerns the infrastructural layer, but it has also an impact at the business layer where the entrance of a new actor (e.g., a new retailer offering higher quality of service) into the market could change business relationships and rules. Also, the context in which the wine is located at a certain time plays an important role. As an example, it is likely that, while the wine is being transported, the details about its status are not accessible dynamically, but that these data (including their history) can only be communicated as soon as the wine enters a logistic center.

As we will further discuss in Sect. 5, this scenario still raises issues that are not solved by the current approaches. For instance, as mentioned above, the need for adaptation could arise both at the level of the business process and at the level of the infrastructure. When both adaptation requests occur in the same time interval, some coordination among adaptation strategies at different layers should take place. For instance, it might happen that the rules for registering the arrival of wine in a logistic center change. Let us assume that this change happens just before the sensors that are connected to an incoming set of bottles try to bind to the logistic center information system to upload the data collected during the trip. In this case, the change of rules at the business process management layer may have an impact on the way the small devices connect to the other parts of the system at the service infrastructure layer. So far, such mutual dependences have not been explored and deserve a deeper analysis.

4.2 A pervasive computing scenario

Hereafter we focus on a pervasive computing scenario that deals with energy consumption, a crucial issue of our modern society.

Specifically, we analyze the case of energy sharing between groups of residential homes. In a residential home, energy consumption is not constant over time, while the threshold limiting its availability is fixed and defined by the way the electricity distribution infrastructure is built.

Each residential home is usually instrumented with a meter, and has a certain threshold assigned that limits the energy that can be consumed at any given point in time. Most of the time the consumption is far lower than the threshold; however, from time to time, a peak in the consumption can occur (e.g., the oven, the washing machine, and the air conditioner are turned on at the same time) causing the meter to stop the delivery of electricity.

To solve the problem in a sustainable way, each individual in a future world could let others absorb more energy when he/she is not using all the allocated energy. For instance, assuming that the threshold is 3 kW, if a certain residential home at a given point in time consumes 2 kW, other residential homes could use part of the remaining 1 kW assigned to it, thus increasing their allowance.

Even in this case we could imagine a service-based solution in which each residential home offers a `lend threshold` service and an orchestrator is in charge of invoking this service on various residential home sites to collect the needed quantity of energy. In this case self-adaptation resides:

- in the ability of each residential home software system to enable and disable the `lend threshold` service based on the current energy consumption status at the corresponding site, and to modify the value of the threshold at the meter as needed;
- in the possibility for the orchestrator to select the `lend threshold` services that appear more suitable at a given time, e.g., the ones offered by the residential homes with the lowest energy consumption at the time.

In both cases, the knowledge about the context plays an important role. In the last example the selection of the `lend threshold` services depends on the context in which the service itself is executed.

Even in this case, however, there are aspects that still need to be properly addressed. One of these concern the possibility of building the service-based application in a peer-to-peer way, avoiding the presence of the orchestrator that constitutes a single point of failure and can have negative effects on the scalability of the whole system. Research in autonomic computing tends to go in this direction (e.g., see Babaoglu et al. 2006; Devescovi et al. 2007). However, the research results are still at a theoretical level and are yet to be framed into some concrete service-based technologies and solutions.

5 Open issues and challenges

Although many approaches introduced in Sect. 3 provide an initial understanding of how to design and implement highly-dynamic, self-adaptive service-based applications, there are still many issues to be resolved by the research community. In the following we discuss the main challenges for research, focusing on the layers and activities introduced in Sect. 3. Several of those challenges have been identified within the European Network of Excellence *S-Cube* and during two workshops organized by the European Commission on "Longer term research challenges in Software and Services" (Papazoglou and Pohl 2008).

5.1 Business process management

One of the most serious open research problem in BPM is how to audit and adapt business processes such as to ensure that they are compliant to policies, business rules and regulations (see Sect. 3.1). Novel service technologies should play a crucial role in allowing various types of users (including management) to ascertain that internal control measures, which govern their key business processes, can be checked, tested, and potentially certified. In the example of Sect. 4.1 this would mean for the wine producer to be able to guarantee that it is always possible to gather information about wine bottles at the lowest level of granularity as possible.

All of this requires continuously adjusting and aligning services within end-to-end business processes that span organizations to cater for regulatory needs. Such changes should not be disruptive by requiring radical modifications in the very fabric of services or the way that business is conducted. This poses enormous methodological and technological challenges as the complexity and scale of service-based applications will expand by orders of magnitude due to the increasing need for flexibility and dynamicity posed by distributed service policies and regulatory compliance.

The challenges of compliant business processes has so far received only limited attention and has never been addressed to its entirety. For a holistic approach to compliant business process management (one that covers the entire compliance life-cycle from design time checking to run-time monitoring and adaptation of services), many research themes have to be addressed (also see Papazoglou 2008b). Those themes include: (1) advanced mechanisms for continuously auditing service-based applications, (2) understanding the need for changes of business processes due to compliance issues and their potential deep effects (see Papazoglou 2008a and Sect. 5.2), (3) validating compliance at run-time (e.g., by means of monitoring), and providing adaptation mechanisms as a remedy for compliance violations.

5.2 Service composition and coordination

The main challenge for service composition and coordination is to understand the kinds of changes that are required during run-time adaptation and to provide proper patterns to realize them. Services can evolve typically by accommodating a multitude of changes along the following functional trajectories (Papazoglou 2008a):

- *Structural changes*: these focus on changes that occur on the service types, messages, and operation (method) interfaces.
- *Business protocol changes*: these concern changes in the conversations that occur between services and are due to changes in policies, regulations, and in the operational behavior of services.
- *Policy induced changes*: these describe changes in policy assertions and constraints. For instance, existing processes could need to be redesigned or improved to conform with new corporate strategies and goals.
- *Operational behavior changes:* these concern changes in the behavior of the operations without this having an impact on their interfaces.

Not all changes have the same level of pervasiveness. We can thus distinguish between two main kinds of service changes (Papazoglou 2008a):

- *Shallow service changes*: where the change effects are localized to a service or are restricted to the clients of that service.
- *Deep service changes*: where change effects are extending beyond the clients of a service, possibly affecting the entire value-chain, i.e., clients of these service clients such as outsourcers or suppliers.

Typical shallow changes are structural and business protocol changes. Typical deep changes include operational behavior and policy induced changes.

While shallow changes need an appropriate versioning strategy, deep changes are quite intricate and require the assistance of a *change-oriented service life cycle* where the objective is to allow services to predict and respond appropriately to changes as they occur (Papazoglou 2008a). A change-oriented service life cycle provides a foundation for business process changes in an orderly fashion and allows end-to-end services to avoid the pitfalls of deploying a maze of business processes that are not appropriately (re)-configured, aligned and controlled as changes occur. In addition to functional changes, a change-oriented service life cycle must deal with non-functional changes which are mainly concerned with end-to-end QoS (Quality of Service) issues, and subsequent SLA guarantees for end-to-end service networks. This aspect is further discussed in Sects. 5.4 and 5.6.

Another aspect specifically highlighted in Sect. 4.2 is the need for building peer-to-peer service-based applications, where the composition logic is distributed through the various peers thus enabling flexibility and scalability of the entire system. Such need is typical of pervasive systems where various independent devices tend to co-operate in a very distributed way. While there are some initiatives in this direction, such as, the so called choreography languages that offer mechanisms to describe the interaction among all peers, this issue has to be further analyzed.

5.3 Service infrastructure

Various new business and computing models are emerging, for instance, utility computing, on-demand computing, cloud computing, and e-science. Distributed, heterogeneous, and multi-vendor service infrastructures will be a key enabler for the usage of service-based applications in these domains. An essential feature of the service infrastructures will be virtualisation capabilities to reduce the complexity of managing heterogeneous systems and to handle diverse resources in a unified way.

As suggested by the scenario in Sect. 4.1, the ability to handle a completely heterogeneous environment in which devices of different capabilities could be incorporated in a seamless way is a very important characteristic.

To navigate the vast space of services which will be offered across the future Internet, advanced and novel service discovery facilities must become an integral feature of future service platforms (see Sect. 5.7). Such facilities will not be only based on proactive search by the users but also on the ability of the system itself to recognize the needs of the users and to offer possible solutions in terms of services.

Finally, as we will further discuss in Sect. 5.7, an important characteristic of service-based applications will be their ability to select and capture the relevant contextual data from the huge amount of data that could be in principle logged (users' information, composition execution logs, service execution logs, etc.).

5.4 Analysis, design, and development

To support the analysis, design, and development activities, mechanisms for handling complexity and for partially automating the development process are needed. One possible way to address these aspects would be the identification of proper design abstractions which are closer to the problem space. Research is thus required in high-level declarative language concepts for the specification of services and service-based applications that allow lay and experienced users and other stakeholders to express their views and requests in terms of "what" rather than "how". One direction which could be followed is expressing the requests of the stakeholders at the intentional level, i.e., as high-level business requirements or goals. Research results on the transformation of applications expressed in terms of the high-level language to service-based realizations and their run-time support will further increase the extent of automation.

Already during the design of the service-based application its adaptation behavior must be considered (“design for adaptation”). Ideally, one would prescribe the complete behavior during design. However, due to the open world assumption (see Sect. 2), service-based applications cannot be specified completely in advance due to the incomplete knowledge about the interacting parties as well as the application’s context. As we have discussed in Sect. 3.4, there are some approaches focusing on design for adaptation. However, a coherent approach that integrates the initiatives in the various areas (requirements, design, testing, deployment, and operation) to create a proper life cycle is still to come.

What we have learnt so far is that due to the fact that many development decisions can only be taken during run-time (e.g., deciding on which of the services to actually bind to the application), the phase of engineering and design shrinks compared to the operation phase of the service-based application. The results achieved in automating many tasks (like code-generation or automatic verification) will further contribute to shrinking the engineering and design phase in comparison to the operation phase. As a consequence, new life cycle models and methodologies will have to support the agility, flexibility, and dynamism required to leverage the potential of service-based applications to dynamically adapt to changing social and economic relationships between communities and business partners.

5.5 Service quality definition, negotiation, and assurance

As businesses in the future will depend on globally distributed service-based applications, the service platform will need to support the specification, dynamic agreement, and monitoring of service levels. This requires that not only QoS aspects are negotiated and agreed, but also that they are checked during the operation in order to determine whether the service provider meets the desired QoS or whether there is a need for re-negotiating the SLAs. As of today, there are only a few techniques and methods that address the QoS characteristics in a comprehensive and cross-cutting fashion across all layers of a service-based application and integrating all phases of SLA conception, negotiation, monitoring, and refinement. In addition, only few approaches address observing the context of a service-based application and its impact on QoS.

Another issue that requires a deep analysis is the dichotomy between the growing need for automating the negotiation and checking of SLAs, on the one hand, and the fact that negotiation remains—by definition—a human-specific activity, on the other hand. Thus, solutions for automatic negotiation will have to consider the social and cultural issues that may impact on the negotiation itself. This aspect certainly requires a multidisciplinary effort in which technology researchers will have to interact with researchers in the social context.

Focusing on the quality assurance aspect, techniques are needed to aggregate individual QoS levels of the services involved in a service composition in order to determine and thus ultimately check end-to-end QoS. This aggregation will typically span different layers of a service-based application and thus a common understanding of what the different QoS characteristics mean within these layers is needed.

Given the need for adapting service-based applications at run-time, quality assurance techniques that can be applied at run-time are essential. Therefore, standard and consolidated "off-line" software quality assurance techniques (like testing and analysis) need to be extended to be applicable while the application operates ("online techniques"). One important requirement to those online techniques is that their overhead and costs should respect the resource constraints of the environment under which they are executed. Those overheads and costs typically include time, computational resources (e.g., processing power and memory) as well as communication resources (e.g., network bandwidth).

In addition to merely extending the quality assurance techniques to the operation phase, synergies between the different classes of analytical quality assurance techniques need to be exploited. As an example, where testing can be used as a preventive mechanism for uncovering faults before the application is deployed, monitoring can be used during the operation of the service-based application in order to "catch" residual faults (Canfora and Di Penta 2006a, 2006b). Another important issue is to understand how to debug and test services without generating side-effects on the current execution (actual operation) of service-based applications. This also means that current service interfaces must be enhanced for testability, diagnosis and debuggability. Yet, these characteristics obviously must be balanced with other important characteristics of services, such as isolation (keeping failures local), information hiding and loose coupling.

Overall, as self-adaptation of a service-based applications will be a key capability of future service-based applications, it becomes important to assure that the adaptation itself behaves as expected. This requires specific quality assurance techniques to verify the adaptation behavior.

5.6 Run-time adaptation

The coordination of adaptation across all functional layers of a service-based application is an issue. In Sect. 4.1 we have highlighted its importance in the wine logistics example, but its importance obviously transcends this single application scenario. Currently, no solutions to this problem exists, leading to situations where the "local" adaptation in the individual functional layers can conflict with each other. Research is thus needed to understand the dependencies of the adaptation in the individual functional layers (which involves different research communities like Grid or BPM) and to devise technologies and methods to enable cross-cutting adaptations.

Further, in current implementations of service-based applications, the adaptation of an application occurs reactively, i.e., the application is re-configured in reaction to changes in the context or to deviations in the application once these have happened. This has severe drawbacks (see Sect. 3.7). There is thus a strong need to not only adapt reactively but to adapt the application before such changes or deviations take place and thus have an undesired effect. In the energy sharing example of Sect. 4.2 this happens when each residential home software system disables the `lend threshold` service when the energy consumption is approaching the threshold value.

Proactive adaptation requires novel techniques and methods for diagnosing the application and for observing its context in order to anticipate possible deviations or changes that require an adaptation. Also, the potential consequences of those deviations and changes need to be estimated in order to determine whether it is worth performing the proactive adaptation (e.g., if the consequences are only minor it might be worthwhile not to trigger an adaptation).

Currently, monitoring techniques are employed for assuring the quality of an application during its operation. The problem with monitoring is that it only checks the current (actual) execution. It does not allow to proactively identify faults which are introduced, for example, due to a change in the application, if they are not leading to a failure in the current execution. Other techniques, such as testing or static analysis, examine sets of classes of executions and thus would allow to proactively identify such faults before they lead to an actual failure during the operation of the system. As an example—to address proactive adaptation—the testing activities could be extended from the design phase to the operation of the service-based application ("online testing"), and the test results could be used to proactively trigger an adaptation.

Finally, the self-adaptation policies identified so far for services are in many cases limited to a few possibilities (mostly, binding services and (re-)negotiation of SLAs). To achieve a fully dynamic self-adaption, service-based applications will need to discover and compose services automatically. This vision of self-adaptation is limited by current technology, because human service consumers still need to provide specifications of these applications (see Sect. 3.4) as well as their context (see Sect. 3.8). The automated adaptation of service-based applications requires the automated manipulation of knowledge about the execution context as well as about the problem domain that is addressed by the service-based application.

5.7 Context

Determining relevant aspects of the context in which the service-based application is executed becomes a run-time issue. As an example, the types of users which interact with the service-based application, or the actual situation or location in which the application is operated is only determined at run-time. Existing requirements engineering and design methods (see Sects. 3.4 and 3.8) assume stable boundaries between the systems and their context. Yet, the run-time invocation of new kinds of services with previously unforeseen features and capabilities (e.g., triggered by the need for adaptation) can lead to dynamic changes of those boundaries. This in turn might require additional knowledge about the problem domain in order to configure, orchestrate, and monitor the services of the application. As a consequence, new descriptive and predictive context models are needed to devise requirements engineering and design methods that are applicable to highly-dynamic self-adaptive service-based applications.

In addition, this requires that future service discovery mechanisms—which go beyond current service registries (see Sect. 5.3)—must be able to take the knowledge about the problem domain into account when retrieving software services. In the future, the automated discovery of software services and problem domain knowledge have to be tightly interleaved. Discovering, reasoning with, and applying problem domain knowledge in order to enable service discovery, composition, and monitoring imposes new research challenges.

6 Conclusions

Future service-based applications will operate in a highly-dynamic world. Technology, regulations, market opportunities and a mixed environment of people, content, and systems will continuously change and evolve. Service-based applications will thus have to continuously adapt themselves to react to changes in their context and to address changing user requirements. Adaptation must be achieved in an automatic fashion: service-based applications should exhibit self-healing, self-optimizing, and self-protecting capabilities. In addition, they should be able to predict problems, such as potential degradation scenarios, future faulty behavior, and deviations from expected behavior, and move towards resolving those issues before they occur. This means that future service-based applications will need to become truly proactive.

The challenges involved in building, evolving and maintaining such highly-dynamic, proactive, self-managing applications cannot be addressed by any research group or research community in isolation. They require the synergy and integration of a variety of research communities including Grid Computing, Service Oriented Computing, Software Engineering, Automated Planning, Information Retrieval, Semantics, and Human Computer Interaction.

On a European level, first systematic efforts are being made to bring those research communities together in order to work on the common goal of enabling future service-based applications. *S-Cube*, the European Network of Excellence on Software Services and Systems,[3] is one such effort, in which 14 European research institutes and universities are jointly working together on the scientific foundations for future service technologies and methods.

Acknowledgements We cordially like to thank all S-Cube members who have contributed to the initial state of the art surveys in this project. Those surveys have provided an excellent basis for the discussions in Sect. 3. In addition, we like to extend our gratitude to Andreas Gehlert and Raffaela Mirandola for comments and pointers on the issue of context. Finally, we like to thank Bashar Nuseibeh—the editor in chief—for giving us the opportunity to contribute to this special issue.

References

Abowd, G.D., Ebling, M., Gellersen, H.W., Hunt, G., Lei, H.: Guest editors' introduction: Context-aware computing. IEEE Pervasive Comput. **01**(3), 22–23 (2002)

Andrews, T. et al.: Business process execution language for Web services. Tech. rep., IBM (2003) http://www.ibm.com/developerworks/library/ws-bpel

Antón, A.: Goal-based requirements analysis. In: Proceedings of the 2nd International Conference on Requirements Engineering (ICRE '96), April 15–18, 1996, Colorado Springs, Colorado, USA, pp. 136–144. IEEE Computer Society

[3]http://www.s-cube-network.eu.

Arsanjani, A.: Service-oriented modeling and architecture. IBM developerWorks article (November 2004) http://www.ibm.com/developerworks/library/ws-soa-design1

Babaoglu, O., Canright, G., Deutsch, A., Di Caro, G., Ducatelle, F., Gambardella, L., Ganguly, N., Jelasity, M., Montemanni, R., Montresor, A., Urnes, T.: Design patterns from biology for distributed computing. ACM Trans. Auton. Adapt. Syst. **1**(1), 26–66 (2006). http://doi.acm.org/10.1145/1152934.1152937

Bai, X., Chen, Y., Shao, Z.: Adaptive web services testing. In: 31st Annual International Computer Software and Applications Conference (COMPSAC 2007), pp. 233–236 (2007a)

Bai, X., Xu, D., Dai, G., Tsai, W., Chen, Y.: Dynamic reconfigurable testing of service-oriented architecture. In: 31st Annual International Computer Software and Applications Conference (COMPSAC 2007), vol. 1, pp. 368–375 (2007b)

Baresi, L., Guinea, S.: Towards dynamic monitoring of ws-bpel processes. In: 3rd International Conference on Service-Oriented Computing (ICSOC 2005), pp. 269–282 (2005)

Baresi, L., Di Nitto, E.: Test and Analysis of Web Services. Springer, Berlin (2007)

Baresi, L., Di Nitto, E., Ghezzi, C.: Toward open-world software: Issue and challenges. Comput. **39**(10), 36–43 (2006)

Baresi, L., Bianculli, D., Ghezzi, C., Guinea, S., Spoletini, P.: Validation of web service compositions. IET Softw. **1**(6), 219–232 (2007)

Batini, C., Bolchini, D., Ceri, S., Matera, M., Maurino, A., Paolini, P.: The UM-MAIS methodology for multi-channel adaptive Web information systems. World Wide Web **10**(4), 349–385 (2007)

Benbernou, S., Meziane, H., Hacid, M.S.: Run-time monitoring for privacy-agreement compliance. In: 5th International Conference on Service-Oriented Computing (ICSOC 2007), pp. 353–364 (2007)

Bertolino, A., Polini, A.: The audition framework for testing web services interoperability. In: EUROMICRO-SEAA, pp. 134–142. IEEE Computer Society, New York (2005)

Bianculli, D., Ghezzi, C., Spoletini, P.: A model checking approach to verify BPEL4WS workflows. In: IEEE International Conference on Service-Oriented Computing and Applications (SOCA 2007), pp. 13–20 (2007)

Bultan, T., Fu, X., Su, J.: Run-time monitoring in service-oriented architectures. In: Baresi, L., Di Nitto, E. (eds.) Test and Analysis of Web Services, pp. 57–85. Springer, Berlin (2007)

Burlton, R.: Business Process Management: Profiting from Process. SAMS publishing, Indianapolis (2001)

Canfora, G., Di Penta, M.: SOA: Testing and self-checking. In: International Workshop on Web Services—Modeling and Testing (WS-MaTE), pp. 3–12 (2006a)

Canfora, G., Di Penta, M.: Testing services and service-centric systems: Challenges and opportunities. IT Prof. **8**(2), 10–17 (2006b)

Charfi, A., Mezini, M.: Hybrid web service composition: Business processes meet business rules. In: 4th International Conference on Service Oriented Computing (ICSOC 2004)

Chhetri, M., Lin, J., Goh, S., Zhang, J., Kowalczyk, R., Yan, J.: A coordinated architecture for the agent-based service level agreement negotiation of Web service composition. In: Australian Software Engineering Conference (ASWEC'06), pp. 90–99 (2006)

Colombo, E., Mylopoulos, J., Spoletini, P.: Modeling and analyzing context-aware composition of services. In: 3rd International Conference on Service Oriented Computing (ICSOC 2005)

Colombo, M., Di Nitto, E., Mauri, M.: Scene: A service composition execution environment supporting dynamic changes disciplined through rules. In: ICSOC, pp. 191–202 (2006)

Comuzzi, M., Pernici, B.: An architecture for flexible Web service QoS negotiation. In: 9th IEEE International Enterprise Distributed Object Computing Conference (EDOC'05), pp. 70–82 (2005)

Cugola, G., Di Nitto, E., Fuggetta, A.: The JEDI event-based infrastructure and its application to the development of the OPSS WFMS. IEEE Trans. Softw. Eng. **27**(9), 827–850 (2001)

Curbera, F.: Components contracts in service-oriented architectures. IEEE Comput. **11**, 74–80 (2007)

Dai, G., Bai, X., Wang, Y., Dai, F.: Contract-based testing for Web services. In: 31st Annual International Computer Software and Applications Conference (COMPSAC 2007), vol. 1, pp. 517–524 (2007)

de Almeida, L.F.J., Vergilio, S.R.: Exploring perturbation based testing for web services. In: IEEE International Conference on Web Services (ICWS 2006), pp. 717–726 (2006)

Deussen, P., Din, G., Schieferdecker, I.: A TTCN-3 based online test and validation platform for Internet services. In: 6th International Symposium on Autonomous Decentralized Systems (ISADS), pp. 177–184 (2003)

Devescovi, D., Di Nitto, E., Dubois, D., Mirandola, R.: Self-organization algorithms for autonomic systems in the selflet approach. In: 1st International Conference on Autonomic Computing and Communication Systems (Autonomics '07), pp. 1–10 (2007)

Dey, A.: Understanding and using context. Pers. Ubiquitous Comput. **5**(1), 4–7 (2001)

Di Nitto, E., Di Penta, M., Gambi, A., Ripa, G., Villani, M.: Negotiation of service level agreements: An architecture and a search-based approach. In: 5th International Conference on Service-Oriented Computing (ICSOC 2007), pp. 295–306 (2007)

Di Penta, M., Bruno, M., Esposito, G., Mazza, V., Canfora, G.: Web services regression testing. In: Baresi, L., Di Nitto, E. (eds.) Test and Analysis of Web Services, pp. 205–234. Springer, New York (2007)

Emmerich, W., Sventek, J., Aoyama, M.: The impact of research on the development of middleware technology. ACM Trans. Softw. Eng. Methodol. **17**(4) (2008)

Erl, T.: Service-Oriented Architecture. Prentice-Hall, Englewood Cliffs (2004)

Foster, I., Kesselman, C.: The Grid: Blueprint for a New Computing Infrastructure. Morgan Kaufmann, Los Altos (2004)

Foster, I., Kesselman, C., Nick, J., Tuecke, S.: The physiology of the grid: An open grid services architecture for distributed systems integration (2002). citeseer.ist.psu.edu/foster02physiology.html

Foster, H., Uchitel, S., Magee, J., Kramer, J.: LTSA-WS: A tool for model-based verification of web service compositions and choreography. In: 28th International Conference on Software Engineering (ICSE '06), pp. 771–774 (2006)

Fu, X., Bultan, T., Su, J.: Analysis of interacting BPEL Web services. In: 13th International Conference on World Wide Web (WWW 2004), pp. 621–630 (2004)

Fuggetta, A.: A classification of case technology. Computer **26**(12), 25–38 (1993). http://dx.doi.org/10.1109/2.247645

Ghezzi, C., Guinea, S.: Run-time monitoring in service-oriented architectures. In: Baresi, L., Di Nitto, E. (eds.) Test and Analysis of Web Services, pp. 237–264. Springer, New York (2007)

Harel, D., Lachover, H., Naamad, A., Pnueli, A., Politi, M., Sherman, R., Shtull-Trauring, A., Trakhtenbrot, M.: STATEMATE: A working environment for the development of complex reactive systems. IEEE Trans. Softw. Eng. **16**(4), 403–414 (1990)

Heckel, R., Mariani, L.: Automatic conformance testing of web services. In: Fundamental Approaches to Software Engineering (FASE 05). LNCS, pp. 34–48 (2005)

IBM-MB, IBM websphere message broker (2008). http://www-306.ibm.com/software/integration/wbimessagebroker/

Jackson, M.: Problem Frames: Analyzing and Structuring Software Development Problems. Addison-Wesley Longman, Reading (2000)

Jaeger, M., Rojec-Goldmann, G., Muhl, G.: QoS aggregation for web service composition using workflow patterns. In: 8th IEEE International Enterprise Distributed Object Computing Conference (EDOC '04), pp. 149–159 (2004). http://dx.doi.org/10.1109/EDOC.2004.19

Jarke, M., Pohl, K.: Establishing visions in context: Toward a model of requirements processes. In: 14th International Conference on Information Systems (ICIS'93), pp. 23–34 (1993)

Josuttis, N.: SOA in Practice: The Art of Distributed System Design. O'Reilly Media, Sebastopol (2007)

Kavantzas, N.: Web services choreography description language 1.0. Tech. rep., W3C (2004)

Kaye, D.: Loosely Coupled: The Missing Pieces of Web Services. RDS Press (2003)

Kazhamiakin, R., Pistore, M.: A parametric communication model for the verification of BPEL4WS compositions. In: 15th International Conference on World Wide Web (WWW 2006)

Kephart, J., Chess, D.: The vision of autonomic computing. IEEE Comput. **36**(1), 41–50 (2003)

Kreger, H. et al.: Management using web services: A proposed architecture and roadmap. Tech. rep., IBM, HP and Computer Associates (2005). http://www-128.ibm.com/developerworks/library/specification/ws-mroadmap

Lazovik, A., Aiello, M., Papazoglou, M.: Planning and monitoring the execution of web service requests. Int. J. Digit. Libr. **6**(3), 235–246 (2006)

Leymann, F.: Choreography for the grid: Towards fitting BPEL to the resource framework. Concurr. Comput. Pract. Exp. **18**(10), 1201–1217 (2006)

Marzolla, M., Mirandola, R.: Performance prediction of Web service workflows. In: Third International Conference on Quality of Software Architectures (QoSA 2007). LNCS, vol. 4880, p. 127. Springer, New York (2007)

Meyer, B.: Object-Oriented Software Construction, 2nd edn. Prentice-Hall, Englewood Cliffs (1997)

Miller, J., Mukerji, J.: Mda guide version 1.0.1. OMG report (2003)

Mylopoulos, J., Borgida, A., Jarke, M., Koubarakis, M.: Telos: Representing knowledge about information systems. ACM Trans. Inf. Syst. **8**(4), 325–362 (1990)

Naur, P., Randell, B. (eds.): Software engineering: Report of a Conference Sponsored by the NATO Science Committee, Scientific Affairs Division, NATO (1968)

Papazoglou, M.: Web Services: Principles and Technology. Addison-Wesley, Reading (2007)

Papazoglou, M.: The challenges of service evolution. In: Advanced Information Systems Engineering Conference (CAISE 2008). Springer, New York (2008a)

Papazoglou, M.: Compliance requirements for business-process driven SOAs. In: IFIP World Computer Congress Conference (WCC 2008). Springer, New York (2008b)

Papazoglou, M., Georgakopoulos, D.: Special issue: Service-oriented computing—Introduction. Commun. ACM **46**(10), 24–28 (2003)

Papazoglou, M., Kratz, B.: Web services technology in support of business transactions. Service Oriented Comput. Appl. **1**(1), 51–63 (2007)

Papazoglou, M., Pohl, K.: Report on longer term research challenges in software and services. Results from two workshops held at the European Commission premises at 8th of November 2007 and 28th and 29th of January 2008, European Commission, http://www.cordis.lu, with contributions from Boniface, M., Ceri, S., Hermenegildo, M., Inverardi, P., Leymann, F., Maiden, N., Metzger, A., Priol, T. (2008)

Parnas, D.: On the criteria to be used in decomposing systems into modules. Commun. ACM **15**(12), 1053–1058 (1972)

Pernici, B. (ed.): Mobile Information Systems: Infrastructure and Design for Adaptivity and Flexibility. Springer, New York (2006)

Pistore, M., Traverso, P.: Assumption-based composition and monitoring of web services. In: Baresi, L., Di Nitto, E. (eds.) Test and Analysis of Web Services, pp. 307–335. Springer, New York (2007)

Pistore, M., Traverso, P., Bertoli, P., Marconi, A.: Automated synthesis of executable Web service compositions from BPEL4WS processes. In: Special Track, 14th International Conference on World Wide Web (WWW 2005)

Rolli, D., Luckner, S., Momm, C., Weinhardt, C.: A framework for composing electronic marketplaces—From market structure to service implementation. In: 3rd Workshop on e-Business (WeB 2004), Washington, DC, USA (2004)

Royce, W.: Managing the development of large software systems. In: IEEE WESCON, San Francisco, CA, USA, pp. 1–9 (1970)

Rozinat, A., van der Aalst, W.M.P.: Conformance checking of processes based on monitoring real behavior. Inf. Syst. **33**(1), 64–95 (2008)

Ruth, M.E., Tu, S.: Towards automating regression test selection for web services. In: 16th International Conference on World Wide Web (WWW 2007), pp. 1265–1266 (2007)

Solanki, M., Cau, A., Zedan, H.: Augmenting semantic Web service descriptions with compositional specification. In: 13th International Conference on World Wide Web (WWW 2004)

Spanoudakis, G., Mahbub, K., Zisman, A.: A platform for context aware runtime web service discovery. In: 2007 IEEE International Conference on Web Services (ICWS 2007), pp. 233–240 (2007)

Traverso, P., Pistore, M.: Automatic composition of semantic Web services into executable processes. In: 3rd International Semantic Web Conference (ISWC 2004)

van der Aalst, W.M.P.: Business alignment: Using process mining as a tool for delta analysis and conformance testing. Requir. Eng. **10**(3), 198–211 (2005)

Yang, J., Papazoglou, M.: Service components for managing the life-cycle of service compositions. Inf. Syst. **28**(1) (2004)

Zachos, K., Maiden, N., Zhu, X., Jones, S.: Discovering web services to specify more complete system requirements. In: 19th International Conference on Advanced Information Systems Engineering (CAiSE 2007), pp. 142–157 (2007)