EVOLVABILITY IN SERVICE ORIENTED SYSTEMS

Anca Daniela Ionita
University "Politehnica" of Bucharest, Splaiul Independentei 313, 060042, Bucharest, Romania

Marin Litoiu
York University, 4700 Keele Street, Toronto, Canada

Keywords: Software Maintenance, Service Oriented Architecture, Cloud Computing, Performance Engineering.

Abstract: The paper investigates the evolution and maintenance of service oriented systems deployed in SOA and cloud infrastructures. It analyzes the challenges entailed by the frequent modifications of business environments, discussing their causes, grasping the evolution points in service architectures, studying classifications of human actors involved across the whole life cycle, as well as pointing out possible risks and difficulties encountered in the process of change. Based on the lessons learned in our study, four pillars for improving service evolvability are identified: orientation towards the users, increasing the level of abstraction, supporting automation and enabling adaptivity through feedback loops.

1 INTRODUCTION

The evolution of software conforming to a service model is triggered by frequent changes in the environment and the users' expectations. Shortening the maintenance life cycle and decreasing its costs are essential for improving the efficiency of service oriented systems. Maintenance efforts may generally be reduced by decomposing the system into small parts (Kafura, 1987). However, the subsequent composition of parts – in our case services – is also subject to change and it is difficult to evaluate its maintenance costs and to define metrics suitable for the high degree of distribution.

Service Oriented Architecture (SOA) allows a system to respond easily to new requirements and to assimilate new business services and new service providers, while the business is developing. Services may be created for processing data, streamlining and reusing functionality incorporated in legacy systems (Sommerville, 2006), integrating activities performed by multiple business partners (Ionita, 2008). The architectures support a wide distribution of the deployed software artefacts; agility and extensibility are increased with the use of services discovered at runtime, sometimes on the basis of semantic technologies (Stojanovic 2006).

SOA is meant to reduce the maintenance efforts and to prevent future problems (Lientz, 1980) through the decoupling implied by the composition of reusable and replaceable services; however, supplementary efforts are transferred to developing infrastructures capable to simplify the addition of new functionality, by publishing new services and orchestrating processes. SOA delegates a part of the concerns related to the necessity of change towards services supplied by external providers, for preserving the integrity of the architecture.

Similarly, cloud computing is emerging as a new computational model, where software is hosted, run and administered in large web data centers and provided as a service. Users of cloud services are exonerated from software licensing, installation, and maintenance, necessary if the programs are executed on their own computers. The long held dream of providing computing as a utility has been made easier by two emerging technologies: *virtualization* and *software as a service* (SaaS). Virtualization is a process that substitutes a physical resource by many logical (virtual) resources. SaaS is the delivery of software functionality seamlessly, over the Internet, instead of installing it on a local machine. Depending on the content of the service, a cloud can also offer Infrastructure as a Service (IaaS) - raw computing services such as CPU and storage - Platform as a Service (PaaS) - COTS, tools, middleware for developing / deploying applications.

However, similarly to SOA, when services are deployed in cloud infrastructures, maintenance efforts are reduced for the end users, but difficulties are augmented for technical actors. Research in cloud computing has recently ramped up, ranging from small scale to very large projects (CERAS, FP7 RESERVOIR). Despite of this, there is no clear vision of how different layers of the cloud, possibly in different administrative domains, can collaborate to satisfy stakeholders' goals.

This paper analyzes the challenges related to the evolution of service based systems, based on a study of SOA and cloud infrastructures, applications and services. First we present principles and models regarding the maintenance of service oriented systems (section 2). Then we identify four pillars for improving evolvability: orientation towards the users, increasing the level of abstraction, supporting automation and feedback control loops (Section 3).

2 MAINTENANCE FOR SERVICE ORIENTED SYSTEMS

2.1 Evolution Laws and Models

Service oriented systems enable an easier adaptation of the system to the "continuing change". The environment evolution may be supported through the evolution of external services, on condition that they respect the same interfaces and the same standard protocols for communication and information interchange, for not affecting the system architecture. In addition, "increasing complexity" is not applied to the system structure, but to the service registry, in the way new leaves appear on a tree. "Self regulation" is determined by the domain rules, imposed by service providers, infrastructure supporters and the end-user community. However, "organisational stability" should often be judged for virtual organizations, for which technical and managerial decisions that influence the system growth are highly distributed. End-users feel comfortable with the "conservation of familiarity", even if they appreciate to discover the new, desired functionalities, and they should not be aware of the complexity and the "continuing growth" of the system behind the curtain, nor should they be affected by "declining quality". Moreover, the evolution process for service infrastructures determine the creation of a "feedback system", influenced by the laws and policies of the application domain, by management and marketing decisions, by user requests and also by analyses of the actual run-time performance. Thus, new challenges and new techniques compensate each other and the software dynamics is still subject to Lehman laws (Lehman, 1997).

The importance of maintenance can be seen in various models of software architecture. The SEI model, called "views and beyond" (Clements, 2003), takes into account the maintainer as a stakeholder that needs detailed architectural documentation. The model introduces three viewtypes, for which one can apply various styles and create specific views, documented with more or less details, in respect with stakeholder necessity. The maintainer needs details for all the views related to Module viewtype (regarding decomposition, generalization, uses and layers) and almost all from Component-and-connector viewtype (shared data, communicating-processes, peer-to-peer and client-server). Another model, dedicated to service oriented systems and called Service Views (Ibrahim, 2006), considers the maintenance staff as part of the category "production support". It defines a special group of quality attributes dedicated to maintenance, which are important for the service providers and crosscut eight specific views. Some maintenance attributes are: (i) tools and procedure manageability; (ii) architecture extensibility related to services and their contracts; (iii) scalability at load increases by adding new hardware or tuning existent infrastructure; (iv) audit logging at service request and execution.

The question is if the existent models of maintenance and evolution grasp the particular nature of systems based on services and include enough details for a precise evaluation. This is difficult with the big diversity of architectures, whose maintainability is hard to predict.

2.2 Aspects of Service Evolution

In this paper we analyze the "evolution" term as defined in the stage model (Bennet, 2000) – the stage when changes do not damage the architecture integrity. In order to determine the key issues for improving evolvability of systems based on services, we analyzed five aspects (see **Figure 1**): (1) the causes that determine a quick evolution; (2) the architectural elements introduced to support variabilities (we call them evolution points) (3) the existent risks; (4) the large number of the actors engaged and (5) the process of change.

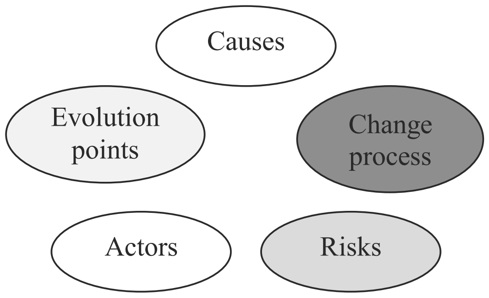

Figure 1: Aspects of service evolution.

2.2.1 Causes

Software services enable and support business operations in an economic environment characterized by globalization and increased competition. On one side, this imposes a quick response to opportunities inside the enterprise, for adapting to the rapid changes of the business environment. For attracting new customers and supporting a rapid growth, including merges and acquisitions, it is necessary to:

- adjust the operational behaviour of services;
- adapt to new rules, regulations and policies;
- adapt to changes in operating conditions;
- transform manual into automated services;
- redesign business processes;
- improve the quality of service.

On the other side, there are more business services involving cross enterprise cooperation, determined by outsourcing, or by the creation of larger consortiums. Service based systems have to face the scaling challenges they have opened themselves. Large infrastructures are being developed, like in FP7 SOA4All project, and cloud infrastructures are also challenged to support dynamic scalability on demand.

2.2.2 Evolution Points

Identifying architectural elements that are supposed to evolve is very important for facilitating maintenance. One makes the difference between “shallow changes” (Papazoglou, 2008), localized in one service and only affecting its clients, and “deep changes”, influencing the entire value chain of a business process. The evolution of services may concern their structure and behaviour; furthermore, these changes may modify specifications, values of various QoS (Quality of Service) metrics and content of SLAs (Service Level Agreements).

To exemplify some possibilities to introduce variations in SOA, let us consider LD-CAST - a prototype for supporting cross-border business cooperation using services provided by European Chambers of Commerce (Ionita, 2008); some of its evolution points allow one to introduce:

- new business processes for orchestration;
- new Web services;
- process and Web service annotation with concepts of a business ontology;
- ontology changes reflecting domain evolution;
- new service providers and local agencies;
- new multilingual content of the portal.

2.2.3 Risks

One evolution point may attract other necessary changes. If one does not cover the entire chain, it may become inconsistent and may induce spurious results, or even discontinuities. There are risks of not respecting compatibility, compliance or conformity of services (Papazoglou, 2008) or of deploying business processes not appropriately configured.

Coming back to the LD-CAST example, for preserving the system consistency when adding a new process, it is necessary to perform the following steps: model the process and transform it into an executable workflow; annotate the process activities according to the business ontology; register and annotate new services; publish the new process (Ionita, 2009). The evolution is supported by various tools and actors of the system, so the risks for producing incoherent processes are quite large.

It is important to predict propagation of changes in the architecture, and to define dependencies between the supported evolution points. In order to reduce risks, one has to define policy and regulation elements at a distinct level, to separate declarative from control issues, and to reduce coupling by creating independently evolving subsystems. Taking into account service versioning and minimizing the propagation of changes may also reduce the risks of inconsistencies.

2.2.4 Involved Actors

One of the difficulties related to SOA is that evolution points are maintained by multiple actors, because they require different competencies. There should be specialists in business process modelling, in ontology maintenance, or service design. There should be administrators for portals, actors for collecting and evaluating change requests, service provider clerks for execution monitoring.

Lin et al. identify two types of actor-roles in a framework for semantic annotation of business processes (Lin, 2009): social actors and technical actors. A comprehensive description of existent roles is given in (Kajko-Mattsson, 2007) – 6 that are generally valid, and another 18 that are specific to SOA, dedicated to: front-end and back-end support, management, design, and quality assurance. With Software as a Service running in cloud infrastructures, new actors and responsibilities are required. The responsibilities of three actors (cloud administrator, application administrator and web service developer) were also described for deploying and managing the web services deployed in a cloud, to ensure the quality of services (Li, 2009).

With all the involved actors and their tools, architecture should be carefully designed and modification rights should be carefully granted for avoiding inconsistent states of the deployed system.

2.2.5 The Process of Change

The classical cycle of change (Yau, 1978) contains five phases: request, planning, implementation, verification & validation, and documentation. Besides, service based systems also have to cope with keeping portals and data bases behind services up-to-date and consistent.

The implementation phase may be unburdened by designing an extensible architecture, to allow adding new services and business processes. One may need to introduce new services by migrating software legacy to SOA, but this has non-technical implications also and it is often sustained by a big effort for changing organization culture and for adopting strategic approaches for human resource management (O'Brian, 2008). One may need to introduce new business processes by making the transition between "as-is" and "to-be" service models, which can be helped by a gap analysis, measuring the impact of changes and realizing the strategy of implementation (Papazoglou, 2008).

The verification and validation phase also faces specific difficulties (Sommerville, 2006). The code of services delivered by external providers is not available; there are no standards for service versioning; payment models may increase costs for this phase; late bounding involves that real life execution uses other services than the system tests.

One searches solutions for eliminating the ad hoc character of change management, avoiding the increase of complexity and assuring a consistent propagation of changes. SAKE project proposes a change ontology for e-Government (Stojanovic, 2006), which enables agile response to unpredictable, frequent changes in the environment. Their change management system aims to harmonize the requests of change and their resolution in a systematic way, and to improve the decision-making process with a unified propagation to the collaborative and knowledge space.

3 INCREASING EVOLVABILITY

3.1 Orientation Towards the Users

In the classical process of change, end-users are only involved in the first and last phases, those of request and validation; in between, the activities are handled in the back-end and involve a great effort for highly specialized actors.

Efforts are currently made to involve end-users furthermore and let them act on the system, instead of simply reporting their change requests. The trend is to increase end-user involvement and to get the evolution points closer to the causes of change, making them visible not only to the clients in the technical sense, but also to the users, who should make choices and impose the desired configurations.

By implementing the third generation of services (Fitzgerald, 2006), which are context-determined, consumer-driven and dynamically composed, the system can run according to the user preferences, like cost, delivery time or trusted service providers.

Empowering end-users with new service front-ends (Lizcano, 2009)) can also speed up some maintenance cycles dedicated to customizing or composing new services, or even to creating new applications. Moreover, ontologies may self evolve by the integration of folksonomies defined by the user community. Thus, some evolution points related to business processes, domain ontology and even multilingual content may be managed directly.

The user-centric approach may also be expanded to an approach leaning on increasing the power of domain experts for performing the system evolution and reducing the necessity of specialised technical actors. Interdisciplinary studies, concerned with the end-user business domain, are recognized to be essential for this (Bennett, 2000) and can lead to an increase of the abstraction level, as argued below.

3.2 Increasing the Level of Abstraction

Maintenance costs more than development and inside it program comprehension consumes more than 50% of the resources (Bennett, 2000).

Therefore, it is essential to perform the changes as much as possible without programming, whose costs are twofold: code understanding plus developing new code – as taken into account by estimation models like COCOMO 2 (Sommerville, 2006).

A solution can be to perform more activities at a higher level of abstraction. New service compositions can be done by using a business modeller based on a general process metamodel (Estublier, 2005) or by defining process models using concepts dedicated to the specific domain, for example public administration (Peristeras, 2006).

The level of abstraction can be increased for many evolution points of a service-based system. One possibility is to define portal configuration tools specific for the application domain, allowing their administrators to manage new service providers, as well as new identity providers. A friendlier environment should also support domain knowledge updates, necessary for semantic characterization of process activities and Web services, for automatic search and discovery of concrete services matching user-specified criteria, and for rule-based reasoning that improves the system adaptation. Process modeling, including new designs and re-engineering, has to be performed by domain experts and to be seamlessly transformed into executable artifacts.

The complexity of service-based architectures requires a large variety of actors for the system maintenance and evolution; the definition of more specific languages and tools for the social actors can diminish the contributions of technical actors, which are more expensive and less agile.

At a large scale, an increase of the level of abstraction may also be observed in the current efforts for creating reference architectures. One of the proposals is a holistic approach, where a SOA based system is assimilated to an ecosystem (Jansen, 2009) with multiple interactions between parts. This is also the philosophy behind the OASIS Reference Architecture for SOA – a viewpoint model with 3 views corresponding to multiple stakeholders: Business via Services, Realizing SOA and Owning SOA, each of them containing several models defined with UML diagrams. Another effort for creating a reference architecture is currently undertaken by the project NEXOF-RA, dedicated to a generic open platform facilitating the collaboration between service providers and third parties.

3.3 Supporting Automation

When the system evolution is defined at a high level of abstraction, there is a need to transform the specified changes into an executable form. Transformation criteria are given in the standard specification of BPMN (Business Process Modeling Notation), and an analysis of the context for generating workflows from process models can be found in (Roser, 2007).

Moreover, the architecture should create support for gradually replacing manual activities with automatic ones, in order to adapt to the rhythm enterprises need to migrate to SOA. This also requires a framework for self-validation and self-testing of newly registered services.

3.4 Enabling Adaptivity through Feedback Loops

The techniques of service late binding and the new semantic technologies allow a run-time adaptation that diminishes the needs for system change. On a very fine granular time scale, Web Services need to often evolve in order to satisfy SLAs. Degradation in performance due to changes in workload or in the underlying hosting infrastructure has to be compensated by changes in individual Web services or in their orchestration, so that the SLAs are met. In a classical approach, those corrective changes have to be done by a human actor. *Adaptive* services and applications are able to sense the change in operating conditions and to readjust automatically.

For services deployed in a cloud, to sense the changes in operating conditions, the web services at SaaS layers might require access to performance and availability counters at PaaS layer. As an example, Model Identification Adaptive Control (MIAC) (Brun, 2009) can be used for adapting the web services in cloud computing. The utilization of a server is not directly accessible to the web service unless it is exposed by PaaS. At the same time, PaaS might need to expose control levers so the adaptation algorithms can influence the behavior of the quality of services of the controlled entities. For example, if the response time of a web service is above that specified by the SLA, the adaptation algorithm should ask for more CPU from PaaS layer.

4 CONCLUSIONS

The purpose of this paper was to identify the key issues for improving evolvability in service oriented systems, including cloud infrastructures and applications. We started with the analysis of the way maintenance laws and models apply to service oriented systems, then we identified five main aspects that characterize service evolution: causes, evolution points, involved actors, possible risks and process of change.

The analysis led to the definition of some key issues for obtaining an easier evolution of services and of systems based on them: a user-centric design, a process of change with more activities performed at a high level of abstraction, supported by increased automation of services and of the processes that orchestrate them, as well as enabling a continuous adaptation of the system to satisfy service level agreements. We consider these issues as the pillars for improving evolvability in service oriented systems and for finding solutions to the challenges raised by business environment dynamics

ACKNOWLEDGEMENTS

This work was supported by CNCSIS - UEFISCSU, Romania, project number PNII – IDEI 1238/2008, by OCE, the Ontario Centers of Excellence, and by the IBM Toronto Centre for Advanced Studies, as part of the program of the Centre for Research in Adaptive Systems (CERAS).

REFERENCES

Bennett, K. H., Rajlich, V. T, 2000. Software Maintenance and Evolution: a Roadmap. In *The Future of Software Engineering*, Finkelstein A., ed. ACM Press.

Brun, Y. et al., 2009. Engineering Self-Adaptive System through Feedback Loops. In *Software Engineering for Self-Adaptive Systems,* Cheng B. et al. ed. Springer Verlag.

Chinneck Li, J., Woodside, J., Litoiu, M., Iszlai, G., 2009. Performance Model Driven QoS Guarantees and Optimization in Clouds. *ACM/IEEE ICSE Workshop on Cloud Computing*, Vancouver, 2009, pp. 15-22.

Clements, P. et al., 2003. *Documenting Software Architectures: Views and Beyond.* Addison-Wesley.

Estublier, J. and Sanlaville, S., 2005. Extensible Process Support Environments for Web Services Orchestration, *Int. Journal of Web Services Practices*, 1(1-2), pp. 30-39.

Ibrahim, D., Misic, V. B., 2006. Service Views: a Coherent View Model of the SOA in the Enterprise, *IEEE /SCC'06*, Chicago, USA 2006, pp. 230-237.

Ionita, A. D., Catapano, A., Giuroiu, S. and Florea, M., 2008. Service oriented system for business cooperation, *ICSE / SDSOA*, Leipzig 2008, pp. 13-18.

Ionita, A. D., Florea, M., Jelea, L., 2009. 4+1 Views for a Business Cooperation Framework Based on SOA, *IAENG Int. Journal of Computer Science*, 36(4).

Kafura, D., Reddy, G. R., 1987. The use of software complexity metrics in software maintenance, *IEEE Transactions on Software Engineering*, SE-13(3), pp. 335-43.

Kajko-Mattsson, M., Lewis, G. A., and Smith, D. B. 2007. A Framework for Roles for Development, Evolution and Maintenance of SOA-Based Systems. In *SD-SOA'07,* Minneapolis, May 20 - 26, 2007.

Fitzgerald, B., Olsson, C. M. ed. 2006. *The Software and Services Challenge, Contribution to the preparation of the Technology Pillar on "Software, Grids, Security and Dependability" FP7*, Ver 1.1.

Jansen, S., Finkelstein, A., Brinkkemper, S., 2009. A sense of community: A research agenda for software ecosystems. *ICSE Companion 2009*, pp. 187-190.

Lehman, M. M. 1997. Laws of Software Evolution Revisited, *EWSPT96*, Oct. 1996, *LNCS* 1149, pp. 108-124.

Lientz, B. P., Swanson, E. B., 1980. *Software Maintenance Management*, Addison-Wesley.

Lin, Y., Krogstie, J., 2009. Quality Evaluation of a Business Process Semantic Annotations Approach, *IBIS*, 3 (1), pp. 9-29.

Lizcano, D., Soriano, J., Reyes, M., and Hierro, J. J. 2009. A user-centric approach for developing and deploying service front-ends in the future internet of services. *Int. J. Web Grid Serv.* 5(2), pp. 155-191.

O'Brian, J., Marakas, G., 2008. *Management Information Systems*, 8th ed. McGraw-Hill.

Papazoglou, M. P. 2008. The Challenges of Service Evolution. In *CAISE'08* Z. Bellahsène and M. Léonard eds. *LNCS*, 5074, pp. 1-15.

Peristeras, V. and Tarabanis, K. 2006. Reengineering the public administration modus operandi through the use of reference domain models and Semantic Web Service technologies, *AAAI /SWEG*, California, USA.

Roser, St., Lautenbacher, F. and Bauer, B. 2007. Generation of workflow code from DSMs, *OOPSLA Workshop on Domain-Specific Modeling*, Montréal, Canada 2007.

Sommerville, J., 2006. *Software Engineering*, 8th ed. Addison-Wesley.

Stojanovic, N., Mentzas, G., Apostolou, D., 2006. Semantic – enabled Agile Knowledge-based e-government, in *AAAI /SWEG*, California, USA 2006.

Yau, S. S., Collofello, J. S., MacGregor, T., 1978. Ripple effect analysis of software maintenance, *IEEE/Compsac*, Computer Society Press, pp. 60 – 65.