SOA PRACTICES AND PATTERNS APPLIED IN GLOBAL SOFTWARE DEVELOPMENT

Marcelo Zilio Pereira, Jorge Luis Nicolas Audy, Rafael Prikladnicki
Faculdade de Informática, Pontifícia Universidade Católica do Rio Grande do Sul, Av. Ipiranga, Porto Alegre, Brazil

Mayara Figueiredo
Instituto de Ciências Exatas e Naturais, Universidade Federal do Pará, Belém, Brazil

Cleidson de Souza
IBM Research – Brazil, São Paulo, SP, Brazil

Keywords: Service oriented architecture, Design patterns, Practices, Framework, Distributed software development, Global software development.

Abstract: Prior research has established a relationship between coordination of software development activities and software architecture both in collocated and distributed projects. Despite the recognized importance of the software architecture in the coordination of development activities, it is still unclear how software architects design the architecture of software systems in distributed projects. To better understand this scenario, this paper reports from a qualitative empirical study where we interviewed software architects to collect information about the software architecture of distributed projects. Information collected has exposed the wide adoption of Service Oriented Architectures (SOA), indicating a trend towards the usage of this low coupling architectural style by companies developing projects with distributed teams. More detailed data collected by follow-up interviews suggested a set of best practices for designing SOA architectures to facilitate the work of the project members.

1 INTRODUCTION

Software Architecture (SA) is an important area within Software Engineering as a transition between the functional specification and coding. It represents a mapping from the abstract concepts defined in the specification to concrete concepts enabling the software coding (Pressman, 200) (Sommerville, 2006).

Motivated by the growth of software development companies with subsidiaries and offices worldwide, researchers in the areas of Global (or distributed) Software Development (GSD), and Computer Supported Cooperative Work (CSCW) have revisited the work of (Conway, 1968) that suggests that software architecture is an important aspect used in the coordination of collocated and distributed software development activities. Examples of research along these lines include (Cataldo, 2009), (De Souza, 2004) and (Garlan, 2000).

(Herbsleb, 2007) argues that the need to manage a variety of dependencies between distributed sites is the essential problem of GSD. Additionally, to achieve substantial progress in GSD is necessary to deep the current understanding about the types of coordination required in distributed projects and the coordination principles adopted, aiming to reduce the amount of communication. In other words, a process should take distribution into account, or even eliminate the incompatibilities in the process through well-defined software architecture.

To design a SA, architects consider existing architectural styles and their adherence to a particular problem. These architectural styles are

Zilio Pereira M., Luis Nicolas Audy J., Prikladnicki R., Figueiredo M. and de Souza C..
SOA PRACTICES AND PATTERNS APPLIED IN GLOBAL SOFTWARE DEVELOPMENT.
DOI: 10.5220/0003507502860292
In *Proceedings of the 13th International Conference on Enterprise Information Systems* (ICEIS-2011), pages 286-292
ISBN: 978-989-8425-55-3

abstractions of how software should be structured to solve a certain problem.

Interviews conducted with expert software architects in software development companies involved in distributed projects indicated that most of these companies are designing their projects based on Service Oriented Architectures (SOA) (Papazoglou, 2007). This led us to conduct additional interviews to identify best practices and patterns adopted in that SOA projects in order to propose a framework of practices in SOA focused on development phase of DSD projects.

This paper presents information gathered from those interviews and proposes the development of SOA practices and patterns aiming to facilitate the coordination of activities and the reduction of communication gaps in DSD projects. The next Section presents related work, while Section 3 presents the methodology used to collect data. Information about the research settings is presented in Section 4. Finally, Section 5 presents our results, and section 6 concludes the paper.

2 RELATED WORK

Empirical studies have focused on identifying how the software architecture and coordination of tasks related each other in a DSD environment. (Grinter, 1999) studied the system architects work and how they have coordinated design across boundaries, tools and processes used to support the work.

(Herbsleb and Grinter, 1999) have proposed to observe problems of coordination in geographically distributed projects with the aim of identifying the types of unforeseen events that may cause coordination problems.

(Ovaska, 2004) found examples of coordination problems in software projects and tried to identify categories of cases that explain the coordination problems encountered. After that, they used these categories to compare centralized and decentralized development to finally have a list of requirements for a development methodology that uses the architecture to support coordination.

Recently, (Cataldo, Nambiar and Herbsleb, 2009) presented the initial results of a qualitative study about the decisions that architects had to take on the design on DSD projects. This study revealed some design patterns that were used to solve the problem both from a technical viewpoint as well as an organizational viewpoint, suggesting there is a relationship between the organizational structure and SA designed.

3 METHODOLOGY

While the design of software architectures is a practical problem to every company doing distributed software development, we took the opportunity to plan this work following a rigorous research process The research question that has guided this work is the following: “Which practices in SOA could facilitate development in DSD projects?” This research is an empirical study, conducted through qualitative data collection and analysis to identify the strategies adopted by software architects to design software architectures. These architects are from software development companies, an aspect considered of relative importance in the field of DSD (Herbsleb, 2001).

This empirical study is intended to enable, through a qualitative data analysis based on Grounded Theory methods, a deep understanding of good practices in SA and how these practices could facilitate software development activities in distributed projects. Grounded Theory approach allows the process of data collection, data analysis and collection of new data analyzed to continue indefinitely, causing researchers to discover a way forward, not knowing where to go or come to an end when, based on samples to test and refine new theories, ideas or categories as those that are emerging from data collected and analyzed. This process ends only when you can no longer get through the data collected new categories or theories (Oates, 2006). Yet, the current stage of the research allowed us to make use of part of Grounded Theory (there were no refinements of the theories that have emerged or exhaustion of possibilities for research). Because this was a qualitative study should be clear the limitations of this type of research, mainly in relation to organizational environments studied, limiting the generalization of the results.

4 RESEARCH SETTING

The review of researches like (Cataldo, Nambiar and Herbsleb, 2009), (Grinter, 1999), (Herbsleb and Grinter, 1999) and (Ovaska, 2004), that focus on the relationship between SA and DSD, the need for further research in this field was raised. Those

works tried to identify: software architect roles, SA designs to improve the coordination of activities, coordination problems in DSD environments and events that may cause these problems, and even the relationship between SA and organizational structures. Aiming to better understand the relationship between DSD and SA and to identify how software architects are working on designing the systems to a distributed environment, in practice, we held a research divided in three phases of interviews with specialists (Software Architects), with expertise in DSD projects.

Data was collected using semi-structured interviews conducted in five companies involved with distributed software development projects, located in Porto Alegre, RS, Brazil. Last round of interviews, an in-depth interview, revealed solutions adopted to solve architectural problems in SOA and in business rules. Below are presented the problems that were identified and the solutions adopted to solve or minimize them. These solutions stand as a set of practices in SOA in DSD projects.

4.1 Practices Adopted by Companies

Situation 1: Service Composition

- Problem: A particular requirement allows a user to have one or more types of Internet product accounts, and for each account type there is a service (program) responsible for its creation, however there is one type of account that must always be created.
- Solution: To avoid inconsistencies in the creation of user accounts, they have created a service that comprises the sum of transactions of two or more services. With this kind of implementation two or more services can become in one without the need for changes in the implementations of those services.

Situation 2: Service Security Facade

- Problem: The Company has a great concern for the safety of their operations. Therefore, it is necessary that all services implements a minimum set of security requirements.
- Solution: To avoid failure in the development of services security layer, they've designed a bus that is responsible for receiving all requests, interpret the entries and validate the safety rules and then invoke the target service, which performs only the business rule and also does not need to translate communication protocols.

Situation 3: Sandbox

- Problem: The large number of service requests and data traffic would cause an overload on the company's ESB.
- Solution: To prevent service disruption could hardly implement in the ESB, service requests are passed to an intermediate service, isolating the bus from external problems, preserving the continuity of other services.

Situation 4: Contract Versioning

- Problem: During the development of a service, changes can happen in the format of the output responses of this service or on how to interpret the incoming messages.
- Solution: To prevent other development teams, who need to access this particular service are constantly affected by these changes, has been implemented what is known as versions of the contract, where a service can serve multiple clients simultaneously with different versions.

Situation 5: Multi-system Transaction

- Problem: A particular service starts a transaction that triggers actions in several other subsystems and modules that make up the architecture of a system of sales.
- Solution: It was developed a mechanism for multi-system rollback and commit to meet the business needs.

Situation 6: Messages Marshaling / Un-marshaling

- Problem: The use of parsers to read and write an XML object can become very expensive depending on the amount of information.
- Solution: It was used SOAP message components that translate into specific objects of the system, eliminating the need to use an XML parser manually.

Situation 7: Publish / Subscribe Service

- Problem: There are main services that centralize information on the system. Those services have a great demand, causing possible overload on it.
- Solution: It was implemented a mechanism that includes the use of queues for asynchronous processing of information. Information is published in a queue where a component responsible for the consumption of items from the queue, do a broadcast to client services.

Situation 8: Centralized Error Log Service

- Problem: Error logs must be stored in a central repository in order to be verified by a specific team.
- Solution: Every service deployed solution must write the error log, invoking a specific service so that another team, responsible for maintenance, could have access to this information.

As we could get from data collected from these interviews, there are practices that had adopted solutions in the development phase of projects and

could be related to some known design patterns. An example is the situation (4) adopted by the company B to solve problems caused by changes in the specification of a service that fits the SOA patterns classification proposed by (Erl, 2009) as a Service Contract Design Pattern known as Concurrent Contracts, which let us to direct this research towards the development of a conceptual framework of SOA practices.

Therefore, it is believed that the specification of a framework is the most suitable at the moment, as a better scientific contribution, and may serve as a reference for the initial development of Service Oriented Architectures in DSD environments, enabling that in the future, further research could develop the studies of modeling practices in SOA. Thus, next section presents the set of practices in SOA and the proposed framework.

5 SET OF PRACTICES IN SOA

The set of practices in SOA that is proposed in this work is based on concepts extracted from a literature review in Software Engineering and on practices adopted by software architects working for the companies involved in distributed software development projects.

There were problems and solutions on SOA programming, and some of these solutions are reported in the literature related to SOA Design Patterns, as defined by (Erl, 2009). According to (Erl, 2008), the focus of a design pattern is to offer a solution to a common problem, but that does not mean that this will be the best solution for all situations. Table 1 below connects the situations described by architects in section 4.1 with SOA design patterns, defined by (Erl, 2009), giving rise to a set of practices proposed in this research.

Table 1: Mapping from situations to practices to SOA Design Patterns.

Situations	Practices	SOA Design Patters
Situation 1	Practice 1	Capability Composition
Situation 2	Practice 2	Service Perimeter Guard
		Protocol Bridging
Situation 3	Practice 3	Asynchronous Queuing
Situation 4	Practice 4	Concurrent Contracts
Situation 5	Practice 5	
Situation 6	Practice 6	
Situation 7	Practice 7	Event Driven Messaging
Situation 8	Practice 8	

The table above shows the mapping from the situations described by the software architects to some SOA design patterns found in the literature. Also, based on table 1 (section 4.1), we can see that practices extracted from those situations can be associated with none, one or more design patterns. To highlight those practices, we describe below some definitions of each, according to (Erl, 2009):

- Practice 1 (Capability Composition Design Pattern): Defined as a service composition, where a functionality encapsulated by a service includes logic able to access functionality from other services. Thus, a service is able to create a composition between the features of one or more services. This practice is related to situation 1, described by company A.
- Practice 2 (Service Protocol Bridging and Service Perimeter Guard Design Patterns): This practice covers two types of patterns and is related to the situation #2, described by company A. Perimeter Guard Service, which provides an intermediate service at the perimeter of a private network, which plays the role of a safe contact point with external services that need to interact with internal private network services, and the Protocol Bridging, which allows two services to connect to a broker instead of connecting directly. In other words, a service responsible for translating the communication protocols between the service provider and consumer.
- Practice 3 (Situation 3 Queuing Asynchronous design pattern): As the situation described by the architect of Company A, this practice implements a mechanism for asynchronous requests and responses to ensure that service consumers could not inhibit performance and compromise system reliability.
- Practice 4 (Concurrent Contracts Design Patterns): This includes the establishment of multiple service contracts for a single service, and each such contract is directed toward a specific type of consumption, thereby facilitating multi-consumer coupling requirements and abstraction concerns at the same time.
- Practice 5: This practice is related to the situation 5 described by the architect from company B, which we couldn't relate it to a similar position available in the literature so far.

It is believed that this may be linked to the fact that the situation mentioned is attached to a very specific business rule.

- Practice 6: This practice is related to the situation 6 described by company B, which we couldn't relate it to a similar position available in the literature so far. It is believed that this may be linked to the fact that the situation referred to is on the use of proprietary software components.
- Practice 7 (Event Driven Messaging Design Pattern): Implements a pattern where a service consumer requests to service provider to be automatically notified about important events. Thus, whenever the service provider receives information updates it will notify all services that had subscribed.
- Practice 8: This practice is related to the situation 8 described by Company C, which we couldn't relate it to a similar position available in the literature so far. It is believed that this may be tied to a specific organizational structure.

From this initial set of practices, we propose a framework that will implement these practices to be applied in designing DSD-based on SOA. In addition, the purpose of this framework is not to be the solution for SOA implementations in DSD projects, but to serve as an initial reference to support development task on scenarios of that kind.

Moreover, not all practices listed in Table 1 are part of the framework, since some of them were specific business solutions, or even, lack data that characterize as a differential for DSD. To illustrate, Figure 1 below, presents an overview of the conceptual framework, divided in types of services that in turn are related to groups of Design Patterns, established by (Erl, 2009).

The layers of the framework are based on groups of services patterns defined by (Erl, 2009), which means it is possible to associate almost each type of service to a practice. The following describes each of those layers and practices that make them up.

- **Service Security & Transformation Layer:** This part of the framework contains services that reach vertically all other services in the architecture and aims to implement a security layer to services and further treatment and conversion of messages. Implementation of this layer was based on the second practice, which falls in the Transformation and Service Security Patterns groups, suggested by (Erl, 2009). Moreover, it is believed that the implementation of this layer would be beneficial to distributed teams of developers, since the implementation of a comprehensive security mechanism to the project may relieve developers, letting them focus on the implementations of the functional requirements of the system.

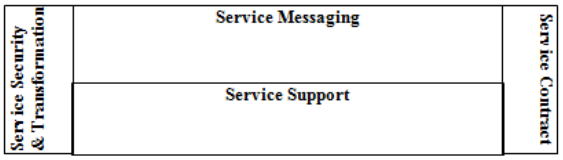

Figure 1: Framework conceptual view.

- **Service Contract Layer:** Again, according to (Erl, 2009), although there are efforts in the phases of analysis and modeling services, those services will still be subject to new situations and new requirements that could force a change to its original design. Therefore, patterns have emerged to help upgrade a service without compromising their original responsibilities. One of these patterns is described by practice #4 and cataloged as Concurrent Contracts. This kind of pattern can prevent distributed teams of being constantly affected by changes in services that are under development and holds essential information requested by other services.
- **Messaging Service Layer:** According to (Erl, 2009), several factors may be involved in service design to estimate the possible scenarios that may occur at run time. Messaging Service Layer, represented by the group called Messaging Service Patterns, provides processing and coordination techniques to the exchange of data between services, as observed in the case mentioned by the third practice, where it is necessary to maintain isolation between service bus and target services. This situation is covered by the Asynchronous Queuing Pattern, which includes the exchange of messages between services via intermediate buffer, preventing overloading of service provider and stuck service consumer, which may cause performance problems. Practice #7 fits this group of patterns, in which a service has access control to basic information and

needs to broadcast changes on that information. This pattern was defined as event-driven messaging, characterized as services that automatically send notifications to consumers about some change in the information it holds.

- **Service Support Layer:** This framework' layer aims to group the services exemplified by practice #8, where all sorts of service logging is centralized in a dedicated logging service, enabling specific QoS teams to take actions in time. Despite not having been found in the literature so far a pattern to this type of service, we chose to include it within the framework by understanding that distributed teams of developers who work in the maintenance of the software can be positively affected by this approach.

As highlighted earlier, not all the practices described in table 1 were part of the framework. In this situation are: practice #1, practice #5 and #6. These practices are not part of the framework because we understood that they represent specific solutions and may not have a relationship with DSD. In this sense another possible design for the proposed framework is depicted in Figure 2.

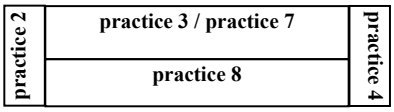

Figure 2: Practices over framework's layers.

In the above figure, Security Service & Transformation layer is represented by practice #2, Messaging Service layer is represented by practices #3 and #7, Service Contract layer is represented by practice #4 and the Service Support layer is represented by practice #8.

6 FUTURE WORK

The contribution of this paper is a set of practices in SOA for distributed software development projects, as well as an initial version of a framework that will implement these practices. The following items are aspects that we are evaluating as next steps in this research.

- **Confirmation**: allow to confirm with interviewed architects whether this set practices could improve developers task in a distributed environment;
- **Experimentation**: allow to perform experiments to evaluate the benefits, which in practice may result from the use of the framework implementing SOA practices in DSD projects. It is intended in this experiment to compare different parameters when an SOA project in DSD is developed using the framework and when it is developed as adhoc;
- **Framework consolidation**: After experiments, refinement and evaluation of results is expected to propose a final version of the SOA practices framework to DSD projects.

With this paper we expect to bring important benefits to both the theory and practice of software development:

i) Design software architectures based on SOA to a better management of development of distributed software projects;

ii) Adopt design practices used in the DSD projects to facilitate the coordination of software development activities; and

ACKNOWLEDGEMENTS

The first three authors were supported by the PDTI program, financed by Dell Computers of Brazil Ltd. (Law 8.248/91). The third author was also supported by CNPq (483125/2010-5, and 560037/2010-4). The fourth and fifth authors were supported by the Brazilian Government under grant CNPq 473220/2008-3 and by the Fundação de Amparo à Pesquisa do Estado do Pará through "Edital Universal N° 003/2008". The fifth author conducted this work while at Federal University of Pará.

REFERENCES

Cataldo, M. and Nambiar, S. 2009. On the relationship between process maturity and geographic distribution: an empirical analysis of their impact on software quality. In Proceedings of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering (Amsterdam, The Netherlands, August 24 - 28, 2009). ACM, New York, NY, 101-110

Cataldo, M., Nambiar, S., and Herbsleb, J. D. 2009 Socio-Technical Design Patterns: A Closer Look at the relationship between Product and Organizational Structures. In 2nd International Workshop on Socio-Technical Congruence (Vancouver, Canada, May 16-24, 2009). ICSE Companion '09. IEEE Computer Society, Washington, DC, 476-477.

Conway, M. E. (1968). How Do Committees invent? Datamation, 14(4), 28-31.

Erl, Thomas. Introducing SOA Design Patterns. SOA World Magazine. 8,6 (June 2008). 2-7.

Erl, T. 2009 SOA Design Patterns. 1st. Prentice Hall PTR.

Garlan, D. 2000. Software architecture: a roadmap. In Proceedings of the Conference on the Future of Software Engineering (Limerick, Ireland, June 04 - 11, 2000). ICSE '00. ACM, New York, NY, 91-101.

Grinter, R. E. 1999. Systems architecture: product designing and social engineering. SIGSOFT Softw. Eng. Notes 24, 2 (Mar. 1999), 11-18.

Herbsleb, J. D. and Grinter, R. E. 1999. Architectures, Coordination, and Distance: Conway's Law and Beyond. IEEE Softw. 16, 5 (Sep. 1999), 63-70.

Herbsleb, J. D. and Moitra, D. 2001. Global software development. IEEE Softw. 16,5 (Mar/Apr 2001), 16-20.

Herbsleb, J. D. 2007. Global Software Engineering: The Future of Socio-technical Coordination. In 2007 Future of Software Engineering (May 23 - 25, 2007). International Conference on Software Engineering. IEEE Computer Society, Washington, DC, 188-198.

Oates, B. J. 2006 Researching Information Systems and Computing. Sage Publications Ltd.

Ovaska, P., Rossi, M., Marttin, P. (2004). Architecture as a Coordination Tool in Multi-site Software Development. Software Processes: Improvement and Practices. 8,4 (Set 2004). 233-247.

Papazoglou, M. P. and Heuvel, W. 2007. Service oriented architectures: approaches, technologies and research issues. The VLDB Journal 16, 3 (Jul.2007), 389-415.

Pressman, R. S. 2000 Software Engineering: a Practitioner's Approach. 5th. McGraw-Hill Higher Education.

Sommerville, I. 2006 Software Engineering: (Update) (8th Edition) (International Computer Science). Addison-Wesley Longman Publishing Co., Inc.

De Souza, C. R., Redmiles, et al. 2004. Sometimes you need to see through walls: a field study of application programming interfaces. In Proceedings of the 2004 ACM Conference on Computer Supported Cooperative Work (Chicago, Illinois, USA, November 06 - 10, 2004). ACM, New York, NY, 63-71.