# A stakeholder-driven Service Life Cycle Model for SOA

Qing Gu
Department of Computer Science
VU University Amsterdam, The Netherlands
qinggu@cs.vu.nl

Patricia Lago
Department of Computer Science
VU University Amsterdam, The Netherlands
patricia@cs.vu.nl

## ABSTRACT

Service-orientation is a relatively new paradigm aiming at developing software systems that are adaptive and dynamic. Service-oriented systems are developed by composing services that are shared across organizations. Because new roles and new development tasks are introduced in service-oriented development as opposed to traditional software engineering, a new approach to service life cycle management is required. In this paper, based on the observations of the state of the art in the field, we propose a stakeholder-driven service life cycle model for service oriented architecture (SOA). Horizontally, the model shows the activities that associated with the stakeholders in SOA. While vertically, the model shows the interactions and cooperation between the stakeholders. This model facilitates the researchers to gain further insight into service-oriented development and governance.

## 1. INTRODUCTION

Recently, the trend in software development has shifted from developing software systems to developing service-oriented systems. In service-oriented systems, software is built by composing software services. Service-oriented architecture (SOA [4]) is a software architecture style that enables the development of such systems. SOA and service-oriented computing attract tremendous attention from the industry and academia because they advocate bringing more flexible, agile and reusable applications.

In traditional software development, software project management typically faces a number of problems. Examples are software systems not satisfying user requirements, software systems developed over budget or over deadline, etc. Distributed software development faces additional problems, such as operational complexity, lack of communications between stakeholders; unclear development tasks; arguable responsibility distribution; etc.

Service-oriented systems are not only quite often jointly developed by a number of organizations like distributed systems, but they also require cooperation between the SOA stakeholders. SOA stakeholders cover architectural roles including service provider, service consumer and service broker. Furthermore, producing services shared across organizations requires good understanding of the business model and the relationship between the business partners. As a relatively new and immature development approach, service-oriented development confronts new challenges in addition to those in traditional and distributed software development. Examples of these new challenges include: how to deal with conflicting requirements, how to align the business requirements with the IT solutions, how to distribute services across organizational boundaries in a secure manner, etc.

Because the architectural roles of SOA and new development tasks are introduced in service-oriented development, the service life cycle model in traditional software engineering cannot be directly applied. In order to maximally achieve SOA's promises and reduce disruptions that are caused by conflicts to a minimum, a systematic approach to service-oriented development is required. A concrete service life cycle model is vital in this approach. It can help the researchers to build insight into the service-oriented development processes and it can be used as a preparatory step for SOA governance. SOA governance addresses guidance and control over SOA applications [15]. Because the cross-organizational nature of services, secure, reliable and high quality service oriented applications can be built efficiently and successfully only under the proper governance [9]. This governance has to be carried out through the entire service life cycle.

By studying the state of the art in the field, we notice that the models proposed by the industrial researchers are in general abstract. Although these models might be successful for their own purpose, for instance commercial, they are not suitable for academia research purposes. To the best of our knowledge, there is no commonly agreed service life cycle model in the literature. In particular, we observe that none of the proposed life cycle models has clear indication on their relationship to the SOA stakeholders and service life cycle stages (design time, runtime and change time). In Section 2, we give an overview on our observations of the current service life cycle models.

In Section 3, we propose a stakeholder-driven service life


*IW-SOSWE'07,* September 3, 2007, Dubrovnik, Croatia.

cycle model in SOA environment, which indicates for each stage the activities, their relationships and their stakeholders. By looking at the model horizontally, we can compare the roles of the stakeholders through their activities, and by looking at the model vertically, we are shown which activities belong to which life cycle stage and the interactions between the stakeholders.

## 2. RELATED WORK

In order to achieve the promise of SOA and deliver high quality service-oriented software products, vendors devote lots of efforts on service life cycle management. We studied four life cycle models proposed by the leading industrial vendors in SOA development and one technique report. During our research, we also found out that the literature pays little attention to service life cycle models. We discuss two academic works that are relevant to service-oriented development at the end of the following list. Each of the models is summarized as following:

- In the service life cycle model proposed by Sun [17], the art of SOA development is nicely generalized by *conception, inception, elaboration, construction* and *transition*.
- The service life cycle model proposed by IBM [8] emphasizes on four iterative processes, namely *model, assemble, deploy* and *manage*.
- The service life cycle model proposed by webMethod [7] consists of *plan, design, manage, run, use* and *change*. Each of these processes are associated with a stakeholder. For instance, *plan* is aligned with an architect and *design* aligned with a developer.
- The service life cycle model proposed by Systinet [19] separates the life cycle of a service provider from the life cycle of a service consumer. Process *design, build, deploy* and *assure* are the concerns of the service - provider; while process *discover, bind, interact* and *monitor* are the concerns of the service consumer. In [19], these two life cycles are presented as parallel but distinct.
- In [25, 24] by Dev2Dev, a service life cycle model is presented with the separation of design time and runtime. The design time processes include *identify business process*, *service modeling* and *build and compose*. The runtime processes include *publish and provision*, *integrate and deploy*, *secure and manage* and *evaluate*.
- A web service life cycle hierarchy model and the service-oriented design and development methodology are proposed in [13]. The web service life cycle hierarchy separates the life cycle model into logical part and physical part. The logical part consists of the service domain, business processes and business services, while the physical part consists of infrastructure services, component-based service realizations and the operational systems. The service-oriented design and development methodology starts with a preparatory phase which is *planning* and eight main phases that concentrate on business process, namely: *analysis* and *design*, *construction* and *testing*, *provisioning*, *deployment*, *execution* and *monitoring*.
- A model-driven SOA development process is proposed in [21]. This process illustrates that the SOA development process consists of *requirement*, *modeling & specification*, *design*, *implementation*, *testing* and *operation and maintenance*.

These proposed models might prove to be useful in their own domain. For instance, the models proposed by the vendors might be more applicable for the marketing or commercial purposes than for the engineering purpose. In general, the works discussed above are not applicable for the researchers to gain better insight into service-oriented development.

Most of the models are made of processes that are at high level granularity. For instance, the model proposed by IBM includes only the most essential processes. Each of these processes can contain a number of sub-processes. However, these sub-processes are left implicit in the model.

Most of the models do not associate the processes with their SOA stakeholders, except for the one proposed by Systinet. However, it ignores that the nature of SOA requires tight cooperation between service consumers and service providers [20]. The interaction between these two stakeholders is missing in the life cycle model. Furthermore, when the SOA deployment becomes mature, more stakeholders in the life cycle have to be included. For instance, to increase the reusability of the services and to fully enable service discovery, a service broker is required. This model overlooks this important stakeholder.

Instead of linking the life cycle processes to SOA stakeholders, the model from WebMethods links each process with a particular development role. According to [26], where a detail analysis on which roles in the development team are responsible for which service-oriented development tasks is well explained, this model is not complete by only linking one process with one role. For instance, during the *plan* process, a project manager also plays an important role in addition to an architect; and not only a developer participates the *design* process but also an architect.

Service life cycle management is often divided into three stages, namely design time, runtime and change time [16, 6]. By considering the life cycle processes in three stages, a better understanding of service life cycle management can be established. Most of the models do not put the processes into service life cycle stages, except for the one proposed at Dev2Dev. However, it only covers the design time stage and the runtime stage. Due to the dynamic and adaptive nature of SOA, the stage change time is vital for successful service-oriented systems.

The web service life cycle hierarchy model and the model-driven SOA development process model in the two literatures present two different approaches. However, as a service life cycle model, the sequences of the processes in the life cycle have to be indicated in order to explain where the process should start and what can be followed. The lack of sequences between the processes does not help to build a good understanding on the service life cycle management.

Table 1: A comparison of the service life cycle models

| Model | the level of granularity | associate to the SOA stakeholders | indication on the SOA life cycle stages | sequences between processes |
|---|---|---|---|---|
| Sun | Very abstract | No | No | Yes |
| IBM | Very high | No | No | Yes |
| WebMethod | High | Link to development roles, but incomplete | No | Yes |
| Systinet | High | Yes, but missing service broker | No | Yes |
| Dev2Dev | High | No | Yes | Yes |
| Web service life cycle hierarchy | Low | No | No | No |
| Model-driven SOA development process | Low | No | No | No |
| Service-oriented design and development methodology | Good | No | No | Yes |

A summary of the comparison of the studied models is shown in Table 1.

## 3. THE PROPOSED LIFE CYCLE MODEL

Based on the observations of the current publications on service life cycle models, we intend to build a concrete service life cycle model. To avoid including too general service processes in the model, we take the granularity of the processes as the same level as the service-oriented design and development methodology that is proposed in [13]. In the rest of the paper, we name the modules in the model as *activities* to differentiate those from other existing models.

One of the design principles of SOA is to decouple the role of service provider and service consumer [1]. When changes occur at one role, the less dependencies between these two roles, the less influence on another role. For instance, suppose a service provider decides to change the operation system that a service currently runs on, service consumers should not be required to change correspondingly. Service provider, service consumer and service broker are three architectural roles (stakeholders) in SOA. In this model, we specifically present the service life cycle activities associated to these stakeholders.

Since services can not be directly used by the end users (except for composing other services), it is usually required to have an application to provide user interfaces. When an application provider integrates a service (services) into an application, it also acts as a service consumer. In this model, we make a distinction between service specific activities and application specific activities by associating the former with a service consumer and the latter with an application provider. The main motivation for this is again decoupling the role to avoid dependencies. Another advantage of separating the role of the application provider and service consumer is that when a service provider produces composed services either entirely or partially by existing services, its role switches between service provider and service consumer depending on whether it is performing service discovery or service publishing. Without the separation, this switch of roles can not be easily captured.

Although in some cases, the roles of these stakeholders can be taken by one or two departments or organizations, we regard these stakeholders as roles instead of physical development teams. For instance, an IT department of a company, which is responsible for building a service oriented application, can be both a service consumer and an application provider. However, we consider it as a service consumer when it invokes services and we consider it as an application provider when it integrates the discovered service into an application.

Service life cycle management is often divided into three stages, namely design time, runtime and change time [16, 6]. Design time refers to the life cycle of a service before it is available for use. During the runtime stage, services are put into production and the implementations start to work. The change time stage comes after runtime. It focuses on the life cycle of a service when adjustments have to be made when business requirements change. When services or service-oriented systems are put into operation in the real world, there is no guarantee that everything will work as what it is expected to be. Changes have to be made in order to meet the expectation of the user. Moreover, change is inevitable because one of the requests for enterprises to stay competitive is to be dynamic and adaptive [12]. Adapting to change is one of the promises of SOA. Therefore, change time has to draw extra attention.

Being aware of which activities are carried out at which stage is of great importance for the development team. Runtime activity is in general the most critical period because the software product is available for use and any problem occurred during this period is real-time problem. Change time activity is also very important because these activities ensure adaptations. Design time activities are relatively less impending yet crucial. Any problems occurred in this period usually allow more time to deal with comparing to runtime and change time.

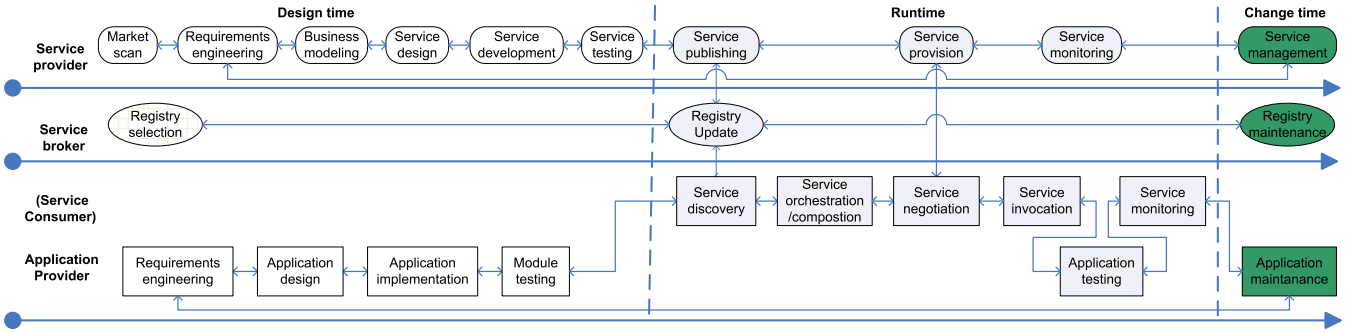


**Figure 1: Life cycle activities associated with services in SOA environment**

As depicted in Figure 1, the proposed model consists of three rows and three columns. The rows are partitioned by the stakeholders: *Service provider, Service broker* and *Application provider* (*Service consumer*). Each row comprises service life cycle activities associated with each stakeholder. The columns are segmented by *Design time, Runtime* and *Change time*. Each column comprises service life cycle activities across the stakeholders. Each of these activities is linked if the output of an activity is the input of another activity. This interaction indicates an iterative and incremental process. This means two linked activities often executed iteratively until the result is satisfactory. The horizontal links imply the interactions within a single stakeholder; the vertical links imply the interactions across the stakeholders; while the diagonal link implies the interactions between *Service consumer and* Application provider. When an application provider integrates services into an application, these diagonal links can be treated as horizontal links.

This proposed life cycle model is designed to be used in a mature SOA development environment. Currently, lots of SOA applications are developed simply by wrapping existing components by service interfaces [14]. Although they are also advocated as SOA applications, we consider it as fake SOA applications since they do not bring main SOA characteristics, such as dynamic and adaptive, into play. The proposed model can be well adapted when service oriented systems are developed across organizational borders; when applications are outsourced to other IT companies; when service providers intend to deliver composed services, etc. Depending on the role of the stakeholder that a develop participant plays, this model can be applied accordingly.

In the following sections, the life cycle activities are explained in terms of design time, run time and change time for the selected stakeholders: service provider, application provider (service consumer) and service broker.

## 3.1 Service provider

A service provider refers to the role of a development party that produces and publishes services which are ready to be executed. Service providers are the owners of services. They are responsible for implementing services as well as maintaining services.

### *3.1.1 Design time*

***Market scan.*** In general, a service provider starts to produce services either under the request of a client or initiated by ideas extracted from experiences or market requests. In case of the former scenario, the service life cycle for the service provider starts with requirements engineering; while in case of the latter scenario, the service life cycle should start with a market scan. The objective of this activity is to investigate the market demands and orient the service production. A good market scan prevents producing services that are either having no beneficial usage or redundant. Inspirations from the market scan help the service provider to decide which kind of service it should produce.

***Requirements engineering.*** After the initial ideas of the services are collected or requested, requirements have to be analyzed specifically. Service providers concern how to achieve the reusability of the services so that they can gain investment returns as soon as possible, how to maintain services and compete with the other service providers, and so on. Unlike the traditional software producers, service providers produce services and these services remain with service providers instead of delivering to software clients. Therefore, during the requirements engineering activity, service providers not only need to analyze the objective, functionality, interface and quality of services, but also need to consider the issues relating to by which means their clients are able to access these services, for instance, by making service level agreements, defining policies, etc.

***Business modeling.*** After the requirements engineering is ready, service providers have a clear goal about what services they are going to produce and what are their target markets. The next step is to model the business process. During this phase, service providers have to capture all the requirements gathered in the requirements engineering activity and model business processes into low level processes which can be defined without going into the technical details. Usually business analysts participate only in this activity. Since services are actually units of business functionalities, a good business modeling is therefore crucial to guarantee services can be well implemented by IT developers without deep knowledge on business processes.

***Service design.*** As soon as the business models are developed, designers or architects start to design services. During this activity, they search the service registry to identify

reusable service, they determine whether an existing service is applicable, they decide whether to implement a new service or modify an existing service to meet the requirements. The objective of this activity is to make a specific design of the services which complies with all the functional and non-functional requirements that are concluded from previous activities. The outputs of this activity include refined service interface, interaction style between services and their clients such as asynchronous or synchronous, invocation method such as RPC or document based, etc.

***Service development.*** When the service design is ready, the service development team starts to implement new services or to modify the existing services identified during the service design activity. The development team not only has to perform the actual coding tasks, but also has to perform service testing against all the requirements defined in previous activities. A number of collaborate services might be needed in order to accomplish the test. In this case, simulation of the services might be used to perform the test. After a number of iteration of coding, testing, integration, simulation, mapping requirements, services are ready to be handed over to a testing team.

***Service testing.*** After services are developed, they have to be tested before they are published for commercial usage. The purposes of testing include not only fault detection but also quality control. Reducing the errors to a minimum is of course the essential requirement for testing, while testing the services against the quality metrics defined in the service design is more significant for a service provider to gain competitive advantage against other providers which deliver similar services.

#### 3.1.2 Run time

***Service publishing.*** Once services are developed and tested, they are ready to be published to a service registry. The objective of this activity is to make produced services available so that the service consumer can find and use them when needed.

***Service provision.*** Although services are published, they are not yet ready to be executed. In order to guarantee that service providers are able to charge their clients for what they offer or to ensure security, usually services are only accessible by authorized users. A service level agreement (SLA), an agreement between a service provider and a service consumer regarding the promise or assurance of a service, is often used as a contract. SLAs define how well services are delivered in terms of cost, availability, performance, etc, by the service provider. The service consumer consequently needs to agree on the expectations of services as defined in SLAs and its obligations on using the services. The objective of service provision is to define rules for service consumers to invoke the published services, for instance, to specify different SLAs for different clients (if necessary). Only when SLAs are signed, or equivalent access control is performed, a service consumer is authorized to use the services deployed by the service provider.

***Service monitoring.*** Once services are published for execution as commercial products, monitoring services to ensure SLAs or policies are met is crucial for service providers. Failing to meet those rules could cause severe financial consequences. Without proper service monitoring, service providers are not able to keep track of the behavior of the published services. By monitoring response time, quite often service providers are able to conclude the availability, reliability and performance of these services.

#### 3.1.3 Change time

***Service management.*** In the business world, the reality shows that there is no way to avoid change. Especially when e-commerce is introduced as a new approach to perform business transactions, changes occur quickly than ever. One of the benefits of SOA is to have the capability to improve business agility through a service oriented IT solution. Services have to be changed in order to fulfill the ever changing business requirements. Quite often, at this point, the service life cycle has to go back to the beginning of the cycle, which is requirements engineering.

The main role of service management is to manage the services so that when changes occur, the impact of change on the clients can be reduced to a minimum. In a SOA system, not only services can be changed but also the associated policies or rules can be changed. For instance, a policy of the conditions under which the consumers are entitled to access the services is changed, a number of services that apply this policy might meet severe problems. Service management has to ensure that any change occurs in SOA environment is handled so that the consumers of the services can continue to use the services.

### 3.2 Service broker

A service broker acts as an intermediary role between a service provider and a service consumer. The main role of a service broker is to provide service location information which is contained in a service registry. A service registry acts as a directory for published services, such as the yellow pages for phone numbers. Service providers use the registry to publish their services and service consumers use it to look up services. Even though so far brokers are associated with registries, they might play a more advanced role, e.g. providing themselves brokering services.

#### 3.2.1 Design time

***Registry selection.*** Registries become more and more important when the number of services increases. It provides a central location for tracking and managing services. Without a service registry, the reusability of services has to be limited because services are hard to be shared across organizations. Moreover, a service registry can also help service providers avoid wasting time to develop services that already exist. Currently, there are two standard registry solutions, one is web service registry [10], which is focused on service description registrations; another is ebXML Registry [11], which not only supports service description but also other SOA metadata. Other extended registry based on these two standards include Sun's Service Registry [18], BEA AquaLogic Service Registry [2], IBM WebSphere Service Registry and Repository [5], etc. As a service broker, it is important to select a proper service registry tool to manage the service registry.

#### 3.2.2 Run time

***Registry update.*** Since a service registry contains the service descriptions and all the location information of published services, it has to be updated when there is new service to be published. If we consider a service registry as a database, services can be discovered only if an entry of the new published service is created and updated with correct information about the service. When a service consumer discovers a service, the service broker returns the address of the service. In this way, the service consumer can use this information to invoke the service later on.

#### 3.2.3 Change time

***Registry maintenance.*** When more and more services contained in the service registry expose to the public, the maintenance of a service registry is of great importance. A good registry maintenance ensures that an update of an already in use service does not break the invocation from existing service consumers, the services that are never been discovered or hardly been invoked should be considered to be removed from the registry, etc. Ideally, the service registry can also maintain a rating function to the services so that it enhances the efficiency of service discovery. For instance, by collecting the feedbacks from the service consumers, the registry is able to give scores to the services that provide similar functionalities. These scores can facilitate the service consumers to decide on the best service to choose and also ensure a healthy market competition.

### 3.3 Application provider (Service consumer)

An application provider integrates services into an application which eventually fulfill the requirements of the end user. An application provider is also called a service consumer when it tries to find services in the service registry and execute the services. Although these two roles are often tightly coupled, we purposely make a distinction between the service consumer and the application provider in the proposed model. As shown in Figure 1, a service consumer focuses on the service level development such as service discovery, service negotiation, etc; while a service provider concentrates on the application level development including integrating services to the application.

#### 3.3.1 Design time

***Requirements engineering.*** Because service consumers and service providers have distinction concerns in their own business domains, the requirements are obviously in different directions. For instance, service consumers target at accomplish a particular task through the usage of a service. Concerns of service consumers include whether the functionalities matches the requirements, whether the performance of the service meets the demands, whether the usage of the service can improve the business efficiency, and so on.

***Application design, implementation and module testing.*** Application design, implementation and module testing are not too much different as the approach in traditional software engineering. Although the services that a system is going to invoke are still unknown, interface descriptions can be used for future integration. This approach is quite similar with component based development.

#### 3.3.2 Run time

***Service discovery.*** After the requirements are defined, service consumers can start to look up published service candidates which comply with the requirements in service registries. Service discovery is a key activity in the whole SOA environment due to the capability that a service can be discovered as one of the characteristics of services. Since services are self-described, they can be discovered either manually or dynamically by means of matching the requirements with service interfaces description or service contracts [23].

***Service orchestration/composition.*** After the application is implemented, the next step is to integrate services to complete the functionality of the application. Service orchestration happens when services are discovered or composed and can be successfully invoked. In this activity, the application provider maps each discovered services or composed services to business activities. All these services should be able to cooperate each other and they have to be integrated into an application. In other words, service orchestration guarantees that services in an application can be well assembled to work together to fulfill business requirements.

***Service negotiation.*** As soon as an application has decided to consume which services that it discovered, it has to invoke the services to execute them. Services are not always free to be invoked. Typically, the non-commercial services which are not required to guarantee the QoS or internal services which do not across organization boundary do not really require the service negotiation activity. In the business world, economically supported services can be a success only if the service providers can benefit from what they offer. Therefore, access control policies are usually made by service providers during the service provision activity. During the service negotiation activity, a service consumer exchanges a number of contract messages with a service provider in order to reach an agreement [3]. The result of the negotiation might be a SLA.

***Service invocation.*** As soon as a service, which can fulfill the user requirements, is discovered, the next step is to invoke the service for execution. Services can be invoked either statically or dynamically [22]. Each of the services that are discovered and composed in the earlier activities has to be invoked for execution.

***Application testing.*** Application testing faces new challenges opposed to traditional software engineering. Integration and non-functional requirements such as performance, compatibility, security, etc, are the main focuses in this activity. It has to ensure that the application not only fulfills all the requirements defined in previous activities, but also the integrated services perform as expected.

***Service monitoring.*** Service monitoring is not only essential for service providers, but also for service consumers. By monitoring services that are paid for to use, service consumer are able to be aware of how reliable, available and how well the services are performed as what they are contracted according to the SLAs or policies. In case of under performance, penalty policies can be applied to guarantee a fair business.

#### 3.3.3 Change time

***Application maintenance.*** In addition to the maintenance tasks defined in traditional software engineering, this activity focuses especially on the changes that might occur. If the changes come from the end user requirements, an iterative procedure might be required to start with the requirements engineering activity. If the changes come from the integrated services, for instance, a better service is available to replace an existing one, or an integrated service is updated, the application maintenance activity has to ensure that those changes can be updated in the application.

## 4. CONCLUSIONS

A good service life cycle model can not only facilitate the life cycle management of service-oriented systems but also can enhance their governance. Good management and governance ensure high quality SOA software products.

In this paper, based on the observations of the related work, we proposed a stakeholder-driven service life cycle model for SOA. This model refines the service life cycle activities that are commonly discussed in the literature. It extends the existing models by associating the activities with their stakeholders and service life cycle stages in an explicit way. It also clearly indicates the interactions between and across the stakeholders.

The proposed model brings three advantages for the management of service-oriented development. Firstly, the separation of the activities associated with the stakeholders makes the duties for each stakeholder clear and explicit. The management team can focus better if they analyze activities per stakeholder. Secondly, the indication of activities belonging to the three life cycle stages helps in clarifying priorities to the actions required. Thirdly, the relationships between the activities provide a clear overview on the order in which these activities should be carried out.

As future work, we intend to analyze industrial case studies to validate the proposed model. By comparing the model to industry practice during their development of service-oriented systems, we can further adjust the model based on the lessons learned from the case studies.

## 5. REFERENCES


[1] D. J. Artus. Soa realization: Service design principles. *developerWorks*, 2006.

[2] BEA. Aqualogic service registry. www.bea.com, 2006.

[3] A. Elfatatry and P. Layzell. Negotiating in service-oriented environments. *Commun. ACM*, 47(8 (August 2004)):103 – 108, 2004.

[4] T. Erl. *Service-Oriented Architecture: Concepts, Technology, and Design.* Prentice Hall, 2005.

[5] IBM. Websphere service registry and repository v6.0.2, 2007.

[6] Infravio. Soa adoption and governance, 2006.

[7] M. Matsumura. The definitive guide to soa governance and lifecycle management, 2007.

[8] G. McBride. The role of soa quality management in soa service lifecycle management. *developerWorks*, 2007.

[9] T. Mitra. A case for soa governance. www-128.ibm.com/developerworks/webservices, 2005.

[10] OASIS. Universal description, discovery and integration v3.0.2 (uddi). cgi.omg.org/docs/formal/00-06-27.pdf, 19 July 2002.

[11] OASIS. Oasis ebxml registry. www.oasis-open.org, 2005.

[12] B. Orriens and J. Yang. Modeling and managing service oriented business collaboration. In *Proceedings of the 2005 International Workshop on Middleware for Web Services (EDOC-MWS05)*, Enschede, The Netherlands, 2005.

[13] M. P. Papazoglou and W.-J. v. d. Heuvel. Service-oriented design and development methodology. *Int. J. Web Engineering and Technology (IJWET)*, 2(4):412–442, 2006.

[14] M. P. Papazoglou, P. Traverso, S. Dustdar, and F. Leymann. Service-oriented computing research roadmap, 2006.

[15] E. Pulier and H. Taylor. *Understanding Enterprise SOA*. Manning, 2006.

[16] G. So. The technologies behind soa governance. soa.sys-con.com/read/314081_2.htm, 2006.

[17] Sun. Sun soa enterpriseexcel, 2006.

[18] Sun. Sun's service registry. www.sun.com/products/soa/registry#registry, 2006.

[19] Systinet. Soa governance: Balancing flexibility & control within an soa. Mercury White Paper, 2006.

[20] W. T. Tsai. Service-oriented system engineering: A new paradigm. In *Service-Oriented System Engineering, 2005. SOSE 2005. IEEE International Workshop*, pages 3– 6, Beijing, China, 2005.

[21] W.-T. Tsai, X. Wei, R. Paul, J.-Y. Chung, Q. Huang, and Y. Chen. Service-oriented system engineering (sose) and its applications to embedded system development. *Service Oriented Computing and Applications*, pages 3–17, 2007.

[22] S. Vinoski. Invocation styles. *IEEE Internet Computing*, 7(4):83–85, 2003.

[23] T. Vitvar, M. Zaremba, and M. Moran. Dynamic service discovery through meta-interactions with service providers. In *Proceedings of the 4th European Semantic Web Conference (ESWC 2007)*. Springer, 2007.

[24] Q. Wall. Understanding the service lifecycle within a soa: Design time. Dev2Dev at dev2dev.bea.com/pub/a/2006/08/, 2006.

[25] Q. Wall. Understanding the service lifecycle within a soa: Run time. Dev2Dev at dev2dev.bea.com/pub/a/2006/11/, 2006.

[26] O. Zimmermann and F. Mueller. Web services project roles. www-128.ibm.com/developerworks/webservices 2004.