A Web Service Migration Framework

M. Mohanned Kazzaz
Department of Information Systems
Faculty of Information Technology
Brno University of Technology
Brno, Czech Republic
Email: ikazzaz@fit.vutbr.cz

Marek Rychlý
Department of Information Systems
Faculty of Information Technology
Brno University of Technology
Brno, Czech Republic
Email: rychly@fit.vutbr.cz

***Abstract*—Service-oriented software systems that operate in highly dynamic or mobile environments have to be able to adapt at runtime to changing environmental conditions. This includes not only adaptability of their individual services, but also ability of the whole systems as service compositions to preserve their integrity and to keep their best performance. This paper presents a concept of service migration in service-oriented architecture as an approach to enable the adaptation of service-oriented systems in changing environments. Moreover, a description of migratable services and service providers by means of migration conditions is proposed and used by service-oriented systems to keep functionality and quality of their services. Finally, the paper describes a prototype design of a framework for the service migration according to the migration conditions and to an user-defined migration decision strategy.**

***Keywords*—Web services; Quality of service; Adaptive systems.**

I. Introduction

In order to operate effectively in mobile environments, a software application has to be able to adapt at runtime to changing environmental conditions such as high volatility and fluctuation of available resources, variable quality of utilised services, unstable network topology, etc. In the case of a distributed component-based system, the changing environmental conditions means not only adaptability of the system's individual components, but also ability of the whole system to preserve its integrity and to keep the best performance [1].

Currently, information systems designed as the component-based systems often utilise service-oriented architecture (SOA) and Web service technology. The service orientation allows to decompose a complex software system into a collection of cooperating and autonomous components known as services. These services cooperate with each other to provide a particular functionality of the implemented system with defined quality.

This paper deals with service migration in SOA as an approach to enable the adaptation of component-based and service-oriented systems in highly dynamic and heterogeneous environments. While traditional models of SOA assume that services are provided permanently by service provides which are predefined at a system's deploy-time or found in service registries at its runtime [2], the service migration enables services to be transparently moved across various network nodes that act temporarily as service providers according to their availability and their resources. Through the service migration, the system is able to cope with the inherent environmental dynamics and to keep functionality and quality of its services (e.g., to employ temporarily available mobile devices as service providers, to react to possible failures of service providers with unreliable resources, etc.).

The rest of this paper is organised as follows. Section II describes a process of service migration including our logical model for migration decision process. The migration process is implemented in Section III as a framework for Web service migration. In Section IV, we review the main approaches that are relevant to our subject. Section V discusses advantages and disadvantages of the proposed framework, especially in comparison with other state-of-the-art approaches. Section VI outlines ongoing implementation of the framework and future research work. Finally, Section VII concludes the paper with summary of our work.

II. Process of Web Service Migration

The migration of a particular service should be considered if its provider is not able to guarantee the functionality or quality of the service and there is no alternative service or service composition that will match a semantic and qualitative description of the original service, i.e., that can provide the same functionality and required quality.

A. Migration Conditions and Migration Decision

We define two groups of *migration conditions*, i.e., the conditions which indicate the need of service migration. The first group contains predefined conditions concerning a provider's runtime state, e.g., network traffic, memory usage, CPU usage, and battery state. The second group consists of user-defined migration conditions, which can be used together with an user-defined migration decision strategy to fully customise a process of decision making for potential service migrations.

The *migration decision* whether start the migration of services provided by particular service providers and how to find target service providers (that will provide the services after the migration) is based on required quality of the services and on optimal and actual states of the service providers.

At the beginning, a migration controller is periodically gathering information about current state of each service provider (e.g., current battery state or resource utilisation) and about its migration conditions (e.g., minimal battery level or maximal resource utilisation to keep fully/optimally functional provider). According to this information and a particular migration decision strategy, the migration controller finds the most underperforming provider which needs migration of its services (this step will be referred later as "origin provider" decision).

 ISBN: 978-1-61208-280-6

After that, the controller checks every migratable service of the provider selected in the previous paragraph. The migratable services have to publish descriptions of required resources by means of the above defined migration conditions (e.g., the conditions can describe that a particular service can be provided by a provider with at least 180 kB of free RAM memory). According to the information provided by these services of the selected provider and according to a particular migration decision strategy, the controller finds the most appropriate service to be migrated (this step will be referred later as "migrated service" decision).

Finally, the controller starts looking for the best candidate provider which will serve as the migration destination (this step will be referred later as "destination provider" decision). This provider is selected according to a particular migration decision strategy, the state information previously provided by the providers (which describes their available resources), and the information provided by the migrated service (which describes the required resources).

B. Migration Decision Modelling

The migration decision can be described by Linear Time Logic (LTL) [3] as a sequence of states which are related to time. LTL formulae are combinations of terms using logical operators $\wedge$ and $\rightarrow$ and temporal operators $\Box$, $\diamond$, and $\circ$. Formulae $\Box p$ and $\diamond p$ means that p always or sometimes holds in the future, respectively, and $\circ p$ means that p is true in the next state.

Our work is based on [4] which provided a service migration logical framework based on LTL. We focus on the first migration phase and logically describe migration decisions by LTL meanwhile [4] dealt with the rest of the migration phases.

Let $P = \{P_1, P_2, \ldots, P_m\}$ is a set of existing providers and $S = \{S_1, S_2, \ldots, S_n\}$ is a set of the migratable services. Migration decision process D, which has been described informally in the previous section, can be defined as follows:

$$\begin{aligned} D \equiv\ & (D \uparrow \wedge OriginProvider \uparrow \wedge T_a) \\ & \wedge \circ (OriginProvider \downarrow \wedge MigratedService \uparrow \wedge T_b) \\ & \wedge \circ (MigratedService \downarrow \wedge DestinationProvider \uparrow \wedge T_c) \\ & \wedge \circ (DestinationProvider \downarrow \wedge T_d) \end{aligned} \quad (1)$$

In the formula above, $\uparrow$ and $\downarrow$ represent the start event and the end event of each process, respectively, and $T_a \leq T_b \leq T_c \leq T_d$ represent the corresponding events' times.

The migration decision process is described in accordance with the previous section as a composition of three sub-processes, namely: *OriginProvider* to find the most critical provider in the system that needs to migrate its services regarding its current state and its migration conditions; *MigratedService* to find the most appropriate migratable service to be migrated from the "OriginProvider"; and *DestinationProvider* to find the best destination provider for "MigratedService".

The low performance condition of provider P_i will be met sometime in the future when the migration decision process will start. Then, the current provider's performance $ProviderLevel$ will not satisfy preferred performance $DefaultLevel$ of the provider according to a migration decision strategy:

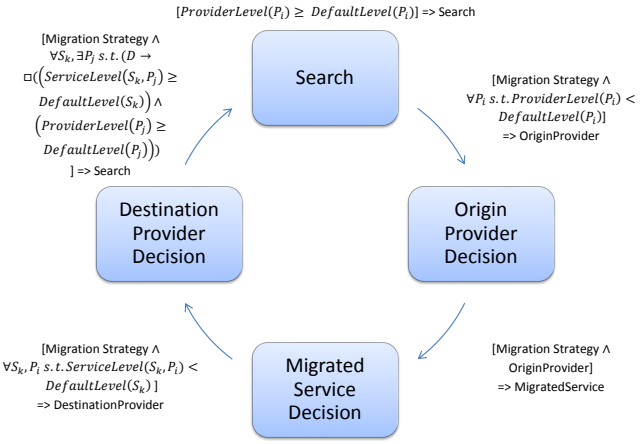

Figure 1. The migration-decision as a finite state automata.

$$\forall P_i \ \ s.t. \ \ (ProviderLevel(P_i) < DefaultLevel(P_i)) \rightarrow \diamond D \quad (2)$$

Sub-process *MigratedService* will be started sometime in the future to determine the most appropriate service to be migrated after a decision about "OriginProvider" has been made:

$$(D \wedge OriginProvider) \rightarrow \diamond MigratedService \quad (3)$$

The necessity condition, the pre-condition of *DestinationProvider* process, becomes true when $ServiceLevel$ describing the current quality of service S_k for $k \in \{1, \ldots n\}$ running on provider P_i for $i \in \{1, \ldots m\}$ is no longer in its preferred level which is denoted as $DefaultLevel$:

$$\begin{aligned} \forall S_k, P_i \ s.t. \ & ((ServiceLevel(S_k, P_i) < DefaultLevel(S_k)) \\ & \wedge OriginProvider) \rightarrow \diamond DestinationProvider \end{aligned} \quad (4)$$

The post-condition of decision process D guarantees acceptable performace of destination provider P_j, where $j \in \{1, \ldots, m\} \setminus \{i\}$, and the necessity condition of its service S_k:

$$\begin{aligned} & \forall S_k, \exists P_j \ \ s.t. \ \ (D \rightarrow \Box(\\ & (ProviderLevel(P_j) \geq DefaultLevel(P_j)) \\ & \wedge (ServiceLevel(S_k, P_j) \geq DefaultLevel(S_k))) \end{aligned} \quad (5)$$

When the migration controller service starts the migration decision process, the output of each sub-process is determined by evaluating provider's and service's conditions. Figure 1 shows the controller's four states ("search", "origin provider decision", "migrated service decision", and "destination provider decision") as a finite state automata with corresponding pre- and post-conditions. The transitions are labelled by $[Conditions] \Rightarrow Process$ where $Conditions$ are the migration conditions of providers and services and $Process$ is a particular sub-process of the migration decision process.

C. Migration of a Serivce

The service migration itself can start when a particular service is selected to be migrated to a particular destination service provider, which is described in the previous section. Then, the migration controller starts migrating the service by

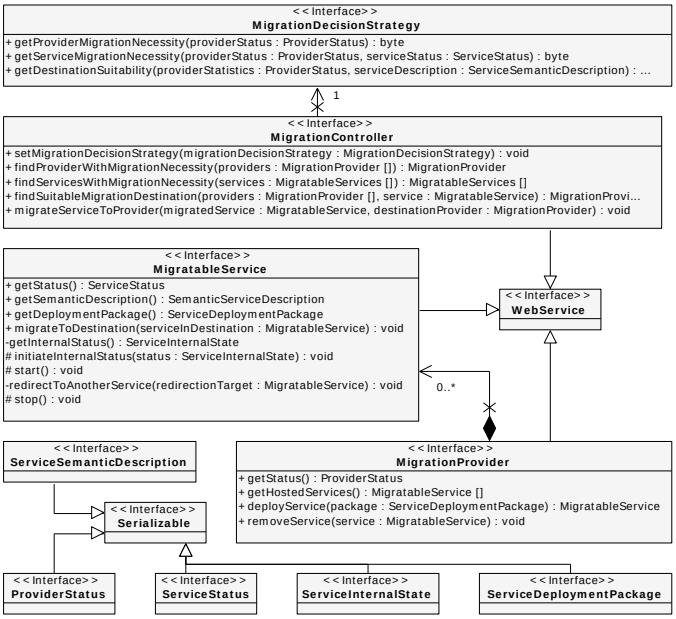

Figure 2. The interface of a control service provided by the framework and the interfaces implemented by participating services and service providers to enable the service migration.

getting a deployment package of the service and deploying it to the destination provider. During this process, the migrating service is stopped and its internal state is stored and sent to the destination provider. All further incomming calls of the service are postponed until the migration is completed, i.e., until the migrated service is initiated in the new location, its internal state is restored, and until the service is able to handle incomming messages.

III. A Framework for Web Service Migration

To support the proposed process of Web service migration, we designed a generic framework, which is introduced in this section. The framework describes an overall service-oriented architecture supporting the service migration and defines interfaces which can be implemented to adapt the framework to a particular Web service implementation technology. It also provides extension points for user-defined migration decision strategies, i.e., the strategies deciding when the migration of a particular service is needed and how it will be performed. The framework's architecture is described in Figure 2 by classes representing specific services with defined interfaces.

To utilise the framework, an implementation of interface *MigrationDecisionStrategy* and auxiliary classes with interfaces *ProviderStatus*, *ServiceStatus*, and *ServiceSemanticDescription*, representing state and semantic information, have to be provided. Migration decisions are based on state information extracted from service providers (e.g., available resources, system workload, battery state, etc.) and their services (e.g., utilised resources, number of requests per an unit of time, etc.) and on the services' semantic descriptions (e.g., provided functionality, inputs and outputs, required runtime conditions, etc.). The migration decision strategy has to be able to acquire instances of the mentioned classes (i.e., the objects representing the state and semantic information) from providers supporting service migration and migratable services.

Interface *MigrationDecisionStrategy* defines methods *getProviderMigrationNecessity*, *getServiceMigrationNecessity*, and *getDestinationSuitability*. The first two methods decide whether some services of a particular service provider or a particular service of this provider, respectively, need to be migrated for some reasons. The third method decides whether a particular provider can be a migration destination for a particular service (i.e., whether a given service can be provided by a given provider after the migration). Returning values of the methods are directly proportional to the necessity of migration of the services or the suitability of the migration destinations.

To be able to migrate, services need to implement interface *MigratableService* with the following public methods. Method *getStatus* returns state information that is used in a migration decision strategy to decide whether a particular service needs to be migrated. Method *getSemanticDescription* provides a semantic description of a migrated service which is used in a migration decision strategy to select a appropriate destination service provider. Method *getDeploymentPackage* returns a service deployment package which is used to deploy a new instance of a migrated service at a destination service provider. Finally, *migrateToDestination* transfers a service's internal state from the service's old instance to its previously deployed new instance and finalises the migration.

Service providers with migratable services need to implement interface *MigrationProvider* with the following public methods. Method *getStatus* returns state information that is used in a migration decision strategy to decide whether services hosted by a particular service provider need to be migrated. Method *getHostedServices* returns all migratable services of a service provider. Method *deployServiceFromPackage* should be able to deploy a service package to create a new instance of the deployed service on a destination service provider. Finally, method *removeService* removes a migrated service from its origin provider.

A. Migration Controller Service

The migration controller is a service provided by the framework which orchestrates services and service providers participating in the service migration. The controller is provided as a "black-box", i.e., it can not be modified and prospective utilisation of the framework can be done solely by implementations of particular migration strategies, migratable services, and their providers. In this way, the controller is also technology independent. It does not need to interpret state information regarding migrated services or their providers and to know how the information has been acquired, it does not need to know a particular technology for service deployment, etc.

The controller implements interface *MigrationController* and orchestrates participating services and service providers as it is described in Figure 3. It periodically checks available service providers for their state information and uses a migration decision strategy to decide whether the information indicate demands for service migration (step (2) in Figure 3). In the case of a service provider which needs service migration, the controller obtains state information from the provider's services (3) and uses the migration decision strategy to decide which

 ISBN: 978-1-61208-280-6

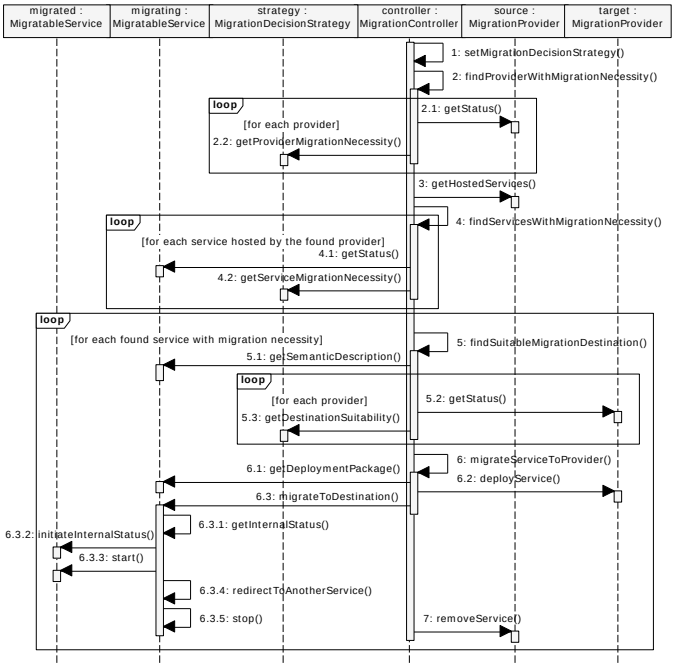

Figure 3. The controller orchestrating service migration.

services need to be migrated and in which order (4). For each such service, the controller obtains its semantic description and tries to find, by means of the migration decision strategy, a service provider suitable as the service's migration destination (5). After that, the controller obtains a deployment package of the migrating service from its origin provider (6.1), deploys a new instance of the migrated service to the destination service provider (6.2), and initiates state transfer from the service's old to its new instance (6.3). Finally, the controller removes the old (inactive) instance from the origin service provider (7).

IV. Related Work

Several works directly address or touch on the migration of components in component-based systems or services in SOA.

Lange et al. [5] described an implementation of mobile agents in Java by the Aglets framework. The framework allows reusing system components, i.e., aglets, in different contexts, however, without any utilisation in making migration decisions.

Hao et al. [6] developed a Web service migration environment and used a genetic algorithm to find the optimal or near-optimal migration decisions. The algorithm calculates the cost of a total round trip including dependency calls for each service request and it is used to decide migration according to this cost. However, the authors did not take into account user-defined conditions affecting the migration decision, e.g., specific requirements on a migrated service or a destination provider.

Zheng and Wu [7] presented an infrastructure for runtime migration of services in a cloud which consists of five components with different roles and of specific criteria to control the migration decision. One of the components collects CPU load data from all known hosts. Then, when the CPU load of a particular host reaches a predefined threshold, a flag is set to indicate that service migration is needed on this host. The approach does not check compatibility of services and providers during migration and does not address possibility of running several services on a single provider at the same time.

Schmidt et al. [8] implemented a prototype of an adaptive Web service migration with two types of migration possibilities, namely context-based migration and functionality-based migration. In the context-based migration, services are migrated to the providers which meet the services' requirements, while in the functionality-based migration, the services are migrated according to their roles in a workflow (i.e., to optimise their communication in the workflow). Both of these migration possibilities can be implemented also in our approach by an appropriate migration decision strategy.

Messig et al. [9] proposed to provide service migration facility in Service Oriented Grid environment which enables taking migration decision based on matching providers' and services' needs and requirements. In this approach, services are hosted by service providers including the resources needed for execution of the services' operations. The authors made several experiments of service migration between two geographically sparse grids where the first grid had high-performance devices and faster network than the second one. While these experiments demonstrated the process of service migration, they are not suitable for the demonstration of migration decisions (e.g., selection of a migration destination) which should be discussed in more detail.

V. Discussion

Contrary to the previously mentioned approaches, the approach proposed in this paper provides a general framework without preference of a particular algorithm or technology for the service migration. The migration decision algorithm and the service migration technology are abstracted by the proposed interfaces (see Section III) and can be implemented and fully customised during utilisation of the framework. The abstraction makes our approach and the corresponding framework more flexible and applicable to different use cases (e.g., in the case of Web services acting as mobile agents), however, it is limited by the proposed architecture and interfaces between the abstracted components and the rest of the framework (e.g., the agent-based Web services may require local migration decisions based on their individual beliefs, desires and intentions, not the centralised approach represented by a single migration decision strategy). Despite the mentioned limitation, our approach is convenient in the cases where the service migration is utilised to keep functionality and quality of services of a service-oriented system and to cope with its inherent environmental dynamics.

Service migration can keep survivability of a system in case of its defect or an attack which damage the system's components or their resources. In this case, endangered services previously hosted by the damaged components can be migrated and started in new and safe locations. Our framework supports this scenario by the migration conditions and the migration decision concepts which allow to detect the damaged components as origins of the migration and the safe locations as possible migration destinations. Focusing on the migration process itself and ignoring the preceding migration decision phase, which is quite common for the approaches mentioned in the previous

 ISBN: 978-1-61208-280-6

section, is not sufficient for the utilisation of service migration in achieving critical system survivability.

Finally, we should discuss a cost of service migration which has not been mentioned before, however, it is important factor of migration decisions as well as of implementations of the migration process. The "migration cost" can be defined as a quantitative metric of difficulty of the service migration. Services in SOA should be designed with respect to SOA principles, and therefore, they should be reusable and stateless [10] which make the service migration easier. Unfortunately, in practice, it is often necessary to break some of these principles, e.g., to use stateful services. In the case of breaking of the SOA principles, migration of the resulting services is more difficult, e.g., it may necessary to migrate resources or state information along with a migrated stateful service. To reflect this issue, the migration conditions and the migration decision proposed in our approach can consider also the migration cost and, for example, migration of stateless services can be preferred to migration of stateful services.

VI. Future Work

The framework proposed in this paper is still a work in progress and needs further elaboration. The future work will focus mainly on implementation of a fully functional prototype of the framework and on its evaluation with particular Web service technologies and migration strategies, covering the scenarios and open issues described in the previous section. Specifically, the future implementation can be divided into two stages following the migration-decision and migration processes.

The migration-decision process which precedes the migration will utilise ontologies and ontology reasoning for description of classes, properties, and relationships, of services, services providers, and devices potentially acting as the service providers. The ontology reasoning will be utilised to evaluate migration conditions, i.e., to nominate services and service providers demanding the service migration, and to select the best destination for migrated services as it is described in Section II-A. We already finished the first version of the ontology and we are currently working on automation of the migration-decision process by ontology reasoning tools.

The implementation of the migration process which performs the actual Web service migration will consist of several steps as it is described in Section III-A. At first, a deployment package and an internal state representation of a migrated service in its original location are obtained. Then, the service is deployed from the package to its new location and the resulting new instance of the service is set to the previously obtained internal state. At this stage, all state-modification operations processed by the service in original location are mirrored to the service in the new location, which keeps internal states of both services synchronous. Finally, the service instance in the new location is activated and the service instance in the original location is deactivated and removed. After that, all incoming messages will be processed by the service in new location, which can be ensured by updating of service registries (i.e., removing the old and adding the new location of the service into a registry) and by forwarding messages going to the original location to the service in the new location (e.g., by a forwarding proxy or by HTTP response code 301 "Moved Permanently" for HTTP-based Web services).

For the service deployment package, native deployment package formats and SOA server management interfaces will be used, e.g., Web Application Resource (WAR) files in the case of Web services implemented in Java EE. The service internal state information will be represented in XML by XML serialisation, e.g., by means of Java API for XML Binding (JAXB) in the case of Web services implemented in Java.

VII. Conclusion

In this paper, we introduced the approach to migration of Web services. We described the process of the service migration including migration decision making and transfer of migrated services to their destination service providers.

Contrary to the previously existing approaches, the approach proposed in this paper provides a general framework without preference of a particular algorithm or technology for the service migration. By introducing fully customisable migration strategies, which evaluate migration conditions and take migration decisions at runtime, our approach can be used to utilise the service migration in different scenarios, e.g., to achieve critical system survivability by the service migration or to assess a cost of the eventual service migration.

Acknowledgment

This work was supported by the research programme MSM 0021630528 "Security-Oriented Research in Information Technology" and by the BUT FIT grant FIT-S-11-2.

References

[1] F. Kon, F. Costa, G. Blair, and R. H. Campbell, "The case for reflective middleware," Commun. ACM, vol. 45, no. 6, Jun. 2002.

[2] A. Michlmayr, F. Rosenberg, C. Platzer, M. Treiber, and S. Dustdar, "Towards recovering the broken SOA triangle: a software engineering perspective," in Proceedings of the 2nd International Workshop on Service Oriented Software Engineering. New York, NY, USA: ACM, 2007, pp. 22–28.

[3] Z. Manna and A. Pnueli, The temporal logic of reactive and concurrent systems. New York, NY, USA: Springer-Verlag New York, Inc., 1992.

[4] Y. Zuo, "Towards a logical framework for migration-based survivability," in Proceedings of the 7th Annual Symposium on Information Assurance / Secure Knowledge Management, Jun. 2012, pp. 29–33.

[5] D. B. Lange and M. Oshima, Programming and deploying Java mobile agents with Aglets. Addison-Wesley, Aug. 1998.

[6] W. Hao, T. Gao, I.-L. Yen, Y. Chen, and R. Paul, "An infrastructure for web services migration for real-time applications," in Second IEEE International Symposium on Service-Oriented System Engineering. Los Alamitos, CA, USA: IEEE Computer Society, Oct. 2006, pp. 41–48.

[7] L. Zheng and S. Wu, "An infrastructure for web services migration in clouds," in International Conference on Computer Application and System Modeling. Los Alamitos, CA, USA: IEEE Computer Society, Oct. 2010, pp. 554–556.

[8] H. Schmidt, R. Kapitza, F. J. Hauck, and H. P. Reiser, "Adaptive web service migration," in Proceedings of the 8th IFIP WG 6.1 International Conference on Distributed Applications and Interoperable Systems. Berlin, Heidelberg: Springer-Verlag, 2008, pp. 182–195.

[9] M. Messig and A. Goscinski, "Service migration in autonomic service oriented grids," in Proceedings of the sixth Australasian workshop on Grid computing and e-research, vol. 82. Darlinghurst, Australia: Australian Computer Society, 2008, pp. 45–54.

[10] T. Erl, Service-Oriented Architecture: Concepts, Technology, and Design. Upper Saddle River, NJ, USA: Prentice Hall PTR, Aug. 2005.

 ISBN: 978-1-61208-280-6