FAKULTÄT FÜR INFORMATIK

TECHNISCHE UNIVERSITÄT MÜNCHEN

Master's Thesis in Informatics

Designing a business platform using microservices.

Rajendra Kharbuja

FAKULTÄT FÜR INFORMATIK

TECHNISCHE UNIVERSITÄT MÜNCHEN

Master's Thesis in Informatics

Designing a business platform using microservices.

Entwerfen einer Business-Plattform mit microservices.

Author:	Rajendra Kharbuja
Supervisor:	Prof. Dr. Florian Matthes
Advisor:	Manoj Mahabaleshwar
Submission Date:	

I confirm that this master's thesis in informatics is my own work and I have documented all sources and material used.

Munich, Rajendra Kharbuja

Acknowledgments

Abstract

Contents

1 Service Candidate

1.1 Introduction

In the previous chapters, granularity and quality of a microservice was focussed. The qualitative as well as quantitative aspects of granularity was described. Additionally, various quality metrics were defined to measure different quality attributes of a microservce. Moreover, a set of principle guidelines and basic metrics to qualify the microservices were also listed.
In addition to that, it is also equally important to agree on the process to identify the service candidates. In this chapter and following chapters, the ways to recognize microservices will be discussed. There are three basic approaches to identify service candidates: Top-down, bottom-up and meet-in-the-middle.
The top-down approach defines the process on the basis of the business model. At first, various models are designed to capture the overall system and architecture. Using the overall design, services are identified upfront in the analysis phase.
The next approach is bottom-up, which base its process on the existing architecture and existing application. The existing application is studied to identify the cohesion and consistency of the features provided by various components of the system. This information is used to aggregate the features and identify services in order to overcome the problems in the existing system.
Finally, the meet-in-the-middle approach is a hybrid approach of both top-down and bottom-up approaches. In this approach, the complete analysis of system as a whole is not performed upfront as the case of top-down approach. Where as, a set of priority areas are identified and used to analyse their business model resulting in a set of services. The services thus achieved are further analysed critically using bottom-up approach to identify problems. The problems are handled in next iterations where the same process as before is followed. The approach is continued for other areas of the system in the order of their priority.[RS07][Ars04]

1.2 Related Work

There are various efforts made regarding the identification of service candidates. Among them, most follows top-down approach. Service Oriented Development Architecture (SODA) uses service driven modeling approach to design and implement services.[Nig05] Service Oriented Analysis and Design (SOAD) on the other hand uses existing modeling process and notation to desing services. [Ars05] Service Oriented Unified Process(SOUP) is highly influenced by Rational Unified Process and extreme programming techniques.[Mit05]
A bottom-up approach Feature Oriented Domain Analysis is also proposed to analysis the existing system features for identifying services.[Kan+90]
Among the few meet-in-the-middle approaches is Feature Analysis for Service-Oriented Reengineering which uses business modeling and FODA. [Che+05] Similarly, Service Oriented Modeling Architecture (SOMA) emphasise on using goal-oriented meet-in-the-middle approach.[Ars04]
Again, there are a majority of papers using use case modeling to identify services. The papers [Mil+07] [How+09] defines Service Responsibility and Interaction Design Method (SRI-DM) to utilize use case to identify the service with cohesive set of functionalities. Similarly, the paper [Far08] identifies various levels of services based on the abstraction level of the use cases.

2 Selection By Use Case

2.1 Use case

A use case is an efficient as well as a fancy way of capturing requirements. Another technique of eliciting requirement is by features specification. However, the feature specification technique limits itself to answer only "what the system is intended to do". On the other hand, use case goes further and specifies "what the system does for any specific kind of user". In this way, it gives a way to specify and most importantly validate the expectation and concerns of stakeholders at the very early phase of software development. Undoubtedly for the same reason, it is not just a tool for requirement specification but an important software engineering technique which guides the software engineering cycle. Using use cases, the software can be developed to focus on the concerns that are valueable to the stakeholders and test accordingly.[NJ04]
It can be valueable to see the ways it has been defined.

Definition 1: [Jac87]

"A use case is a sequence of actions performed by the system to yield an observable result of value to a particular user."

Definition 2: [RJB99]

"A use case is a description of a set of sequences of actions, including variants, that a system performs that yield an observable result of value to an actor."

2.2 Use case Refactoring

According to [Jac87] it is not always intuitive to modularize use cases directly. As per the definitions provided in 2.1, a use case consists of a number of ordered functions together to accomplish a certain goal. [Jac87] defines these cohesive functionalites distributed across usecases as clusters. And the process of identifying the clusters scattered around the use cases is the process of identifying services. [NJ04] and [Jac03] describes use case possibly consisting of various cross-cutting concerns. The papers use the term 'use case module' to define the cohesive set of tasks. The figure 2.1 demonstrate the cross cutting functionalities needed by a use case in order to accomplish its goal. For example, the use case 'does A' has to perform separate functionalities on different domain entities X and Y. Similarly, the use case 'does B' needs to perform distinct functions on entities X, Y and Z.

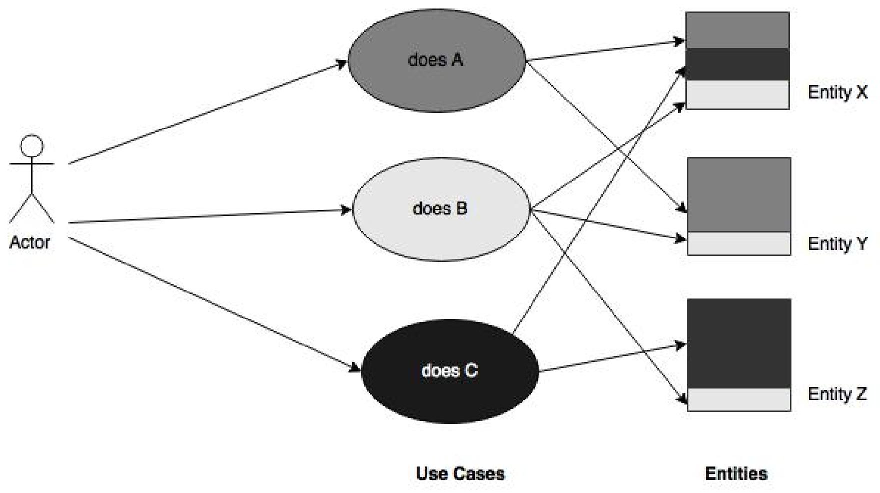

Figure 2.1: Use cases with cross-cutting concerns [NJ04]

This situation prompts for the analysis of the use cases for cross cutting tasks and refactoring them in order to map the cohesive functionalities and use cases. Refactoring helps to achieve the right level of abstraction and granularity by improving functionality cohesion as well as elimination of redundancy and finally promoting the reusablity. [DK07]

Furthermore, the refactoring is assisted by various relationships in use case model to represent dependencies between use cases, which are include, generalization and extend. [NJ04]

2.3 Process for Use Case Refactoring

In order to refactor the use cases and finally map use cases to the service, the papers [KY06], [YK06] and [DK07] provide a comprehensive method. It is a type of meet-in-the-middle approach where use case models are first created and then refactored to create new set of use cases to accomplish high cohesion in functionality and loose coupling. This will ultimately create use cases supporting modularity and autonomy. [Far08] There are three distinct steps for service identification using service refactoring.

1. **TaskTree Generation**
   In this step, the initial use case model created during domain analysis is used to create task trees for each distinct use case. The task trees provide sequence of individual tasks required to accomplish in order to achieve the goal of the use case.

2. **Use Case Refactoring**
   In this step, the task tree generated in the previous step is analysed. As already mentioned in the section **??**, that the initial use case consists of various cross cutting functionalities which runs through large number of business entities. In order to minimize that, refactoring is performed. The detail rules are provided in section 2.4.

3. **Service Identification**
   The use cases achieved after refactoring have correct level of abstractions and granularity representing cohesive business functionality. The unit use cases thus achieved appreciate reuse of common functionality. [DK07] Furthermore, the approach considers the concerns of stakeholders in terms of cohesive business functionalities to be represented by use cases.[Far08] Additionally, the process also clarifies the dependencies between various use cases. Thus, the final use cases obtained can be directly maped to individual services.

2.4 Rules for Use Case Refactoring

The context is first defined before distinct rules are presented.

Context 1

If U represents use case model of the application, t_i represents any task of U and T being the set of tasks of U. Then, $\forall t_i \in T$, t_i exists in the post-refactoring model U′. The refactoring of a use case model preserves the set of tasks.

Context 2

A refactoring rule R is defined as a 3-tuple (Parameters, Preconditions, Postconditions). Parameters are the entities involved in the refactoring, precondition defines the condition which must be satisfied by the usecase u in order for R to be applied in u. Postcondition defines the state of U after R is applied.
The various refactoring rules are as listed below.

2.4.1 Decomposition Refactoring

When the usecase is complex and composed of various functionally independent tasks, the tasks can be ejected out of the task tree of the usecase and represented as a new use case. The table 2.1 shows the decomposition rule in detail.

Parameters	u: a use case to be decomposed t: represents task tree of u t′: a subtask tree of t
Preconditions	t′ is functionally independent of u
Postconditions	1. new use case u′ containing task tree t′ is generated 2. a dependency is created between u and u′

Table 2.1: Decomposition Rule

2.4.2 Equivalence Refactoring

If the two use cases share their tasks in the task tree, we can conclude that they are equivalent and redundant in the use case model. The table 2.2 shows the rule for the refactoring.

Parameters	u_1 , u_2: two distinct use cases
Preconditions	the task trees of u_1 and u_2 have same behavior
Postconditions	1. u_2 is replaced by u_1 2. all the relationship of u_2 are fulfilled by u_1 3. u_2 has no relationship with any other use cases

Table 2.2: Equivalence Rule

2.4.3 Composition Refactoring

When there are two or more small-grained use cases such that they have related tasks, the use cases can be represented by a composite unit use case.

Parameters	u_1, u_2: the fine grained use cases t_1, t_2: the task trees of u_1 and u_2 respectively
Precondition	t_1 and t_2 are functionally related.
Postconditions	1. u_1 and u_2 are merged to a new unit use case u 2. u has new task tree given by $t_1 \cup t_2$ 3. the dependencies of u_1 and u_2 are handled by u 4. u_1 and u_2 are deleted along with their task trees t_1 and t_2

Table 2.3: Composition Rule

2.4.4 Generalization Refactoring

When multiple use cases share some volume of dependent set of tasks in their task trees, it can be implied that the common tasks set can be represented by a new use case. The table 2.4 provides the specific of the rule.

Parameters	u_1 , u_2: two distinct use cases t_1 , t_2: task trees of u_1 and u_2 respectively
Precondition	t_1 and t_2 share a common set of task $t = \{x_1, x_2 ... x_n\}$
Postconditions	1. a new use case u is created using task tree $t = \{x_1, x_2 ... x_n\}$ 2. relationship between u with u_1 and between u with u_2 is created 3. the task tree $t = \{x_1, x_2 ... x_n\}$ is removed from task tree of both u_1 and u_2 4. the common relationship of both u_1 and u_2 are handled by u and removed from them

Table 2.4: Generalization Rule

2.4.5 Merge Refactoring

When a use case is just specific for another use case and the use case is only the consumer for it, the two use cases can be merged into one.

Parameters	u, u′: the use cases r defines the dependency of u with u′ t, t′: the task trees of u and u′ respectively
Precondition	there is no dependency of other usecases with u′ except u
Postconditions	1. u′ is merged to u 2. u has new task tree given by $t \cup t'$ 3. r is removed

Table 2.5: Merge Rule

2.4.6 Deletion Refactoring

When a use case is defined but no relationship can be agreed with other use cases or actors then it can be referred that the use case represent redundant set of tasks which has already been defined by other use cases.

Parameters	u: a distinct use case
Precondition	the use case u has no relationship with any other usecases and actors
Postconditions	the use case u is deleted

Table 2.6: Deletion Rule

2.5 Example Scenario

In this section, a case study will be taken as an example. The case study will be modeled using use case. Finally, the use case will be refactored using the rules given by **??**.

Case Study

The case study is about an international hotel named ’XYZ’ which has branches in various locations. In each location, it offers rooms with varying facilities as well as prices. In order to make it easy for customers to find room irrespective of the location of customer, it is planning to offer an online room booking application. Using this application, any registered customer can search for a room according to his/her requirement in any location and when he/she is satisfied, can be able to book the room online as well. At present, the payment will be accepted in person, only after the customer gets into the respective branch. Finally, if the customer wants to cancel the room, he/she can do online anytime. It is an initial case study so only priority cases and conditions are considered here.

The remaining part of this section presents the steps to indentify the services using use case refactoring using the process defined in section 2.3 and rules presented in the section 2.4

Step 1:

The initial analysis of the case study will produce use case model as given by figure 2.2.

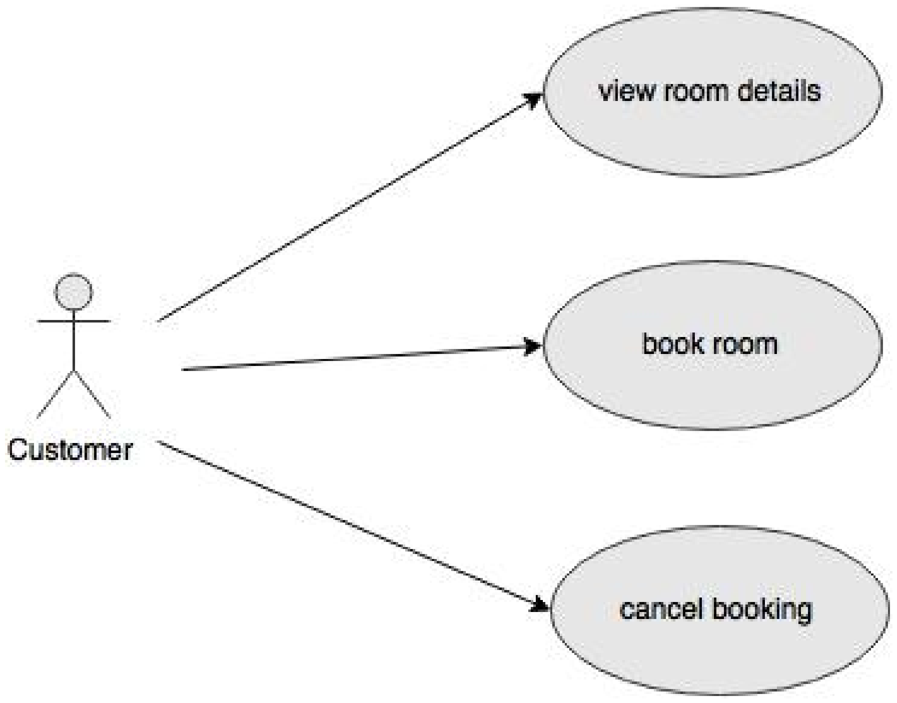

Figure 2.2: Initial Use Case Model for Online Room Booking Application

Step 2:

For each initial use case, task trees are generated which are needed to accomplish the desired functionality of respective use cases.

Use Case	Task Tree
book room	customer enters access credentials system validates the credentials customer enters location and time details for booking system fetch the details of empty rooms according to the data provided system displays the details in muliple pages customer choose the room and submits for booking system generates a booking number System updates the room system sends notification to the customer regarding booking
cancel booking	customer enters access credentials system validates the credentials customer enters the booking number system validates the booking number customer cancels the booking system updates the room system sends notification to the customer regarding cancelation
view room details	customer enters access credentials system validate the credentials customer enter location and time system fetch the details of empty rooms according to the data provided system display the details in multiple pages

Table 2.7: Task Trees for Initial Use Cases

Step 3:
The initial task trees created for each use case at Step 2 is analysed. There are quite a few set of tasks which are functionally independent of the use case goal and also few tasks which are common in more use use cases. The following table 2.8 lists those tasks from the tasks trees 2.7 which are either functionally independent from their corresponding use cases or common in more use cases.

Tasks Type	**Tasks**
Independent Tasks	system validates the credentials system fetch the details of empty rooms according to the data provided system generates a booking number system validates the booking number system updates the room system sends notification to the customer
Common Tasks	system validates the credentials system fetch the details of empty rooms according to the data provided system updates the room system sends notification to the customer

Table 2.8: Common and Independent Tasks

Now, for independent tasks the docomposition rule given by 2.1 can be applied and for common tasks, the generalization rule given by 2.4 can be applied. The use cases obtained after applying these rules is shown by the figure 2.3

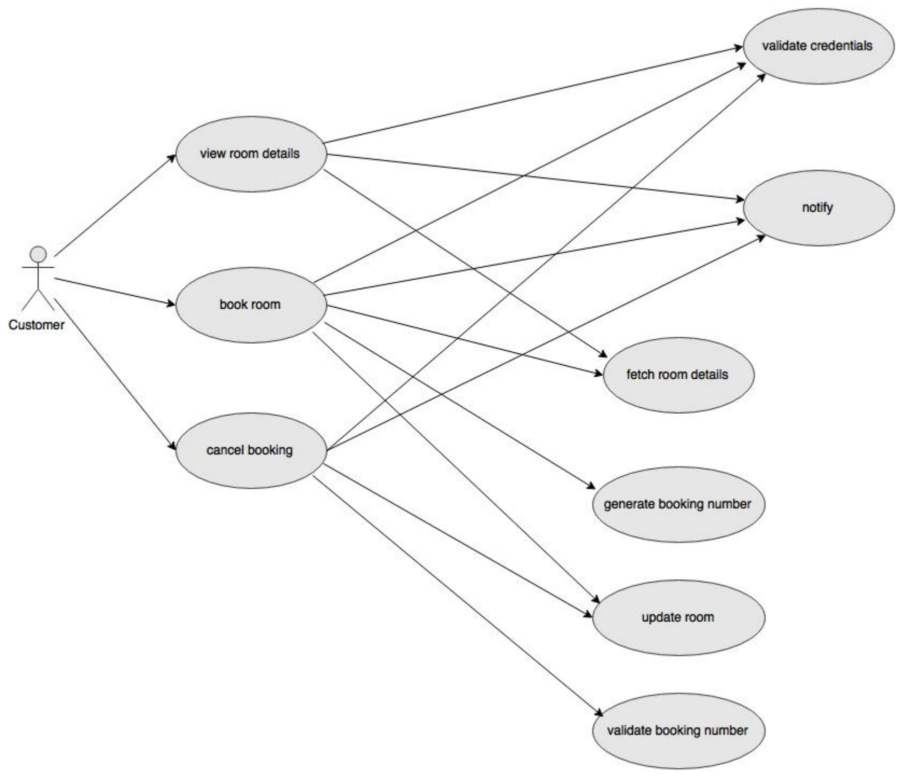

Figure 2.3: Use Case Model after applying Decomposition and Generalization rules

Step 4:
The use case model 2.3 obtained in Step 3 is analysed again for further refactoring. It can seen that the use cases 'fetch room details' and 'update room' are fine grained and related to same business model 'Room'. Similarly, the use cases 'generate booking number' and 'validate booking number' are related to business entity 'Booking Number'. The composition rule given by 2.3 can be applied to these use cases, which will then result in use case model given by figure 2.4

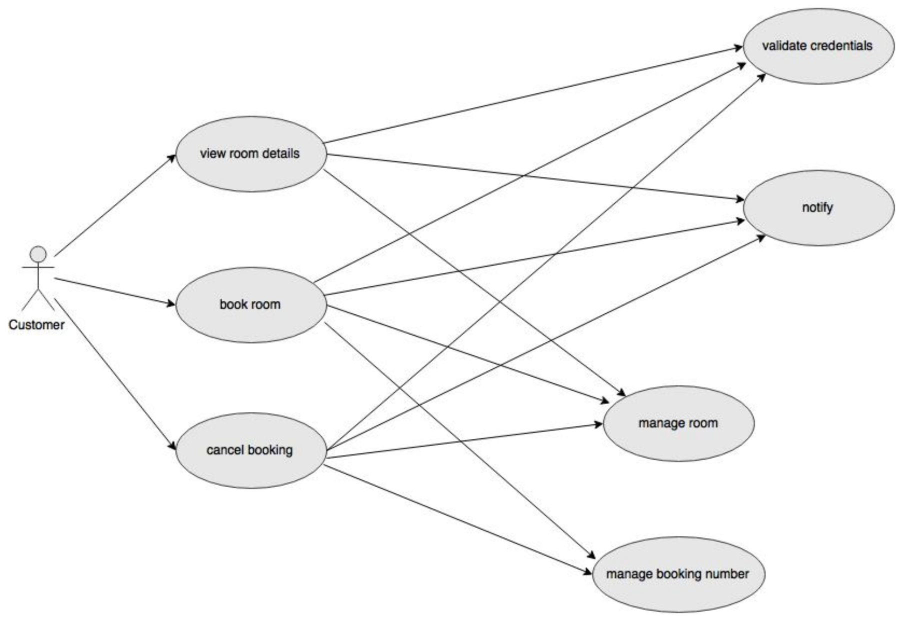

Figure 2.4: Use Case Model after applying Composition rule

Step 5:
Finally, the use case obtained in step 4 is used to identify the service candidates. The final use cases obtained in Step 4 have appropriate level of granularity and cohesive functionalities to be identified as individual services. Most importantly, the refactoring has now separated the cross cutting concerns in terms of various reusable fine grained use cases as shown by the figure. If we compare the the final use case model 2.5 with respect to figure 2.1, then it can be implied that the functionalities operating on business entities as well as the business logic serving the candidate concerns are separated and represented by individual use cases. So, the each use case can be mapped to individual services. The services are as listed in the bottom of the figure 2.5.

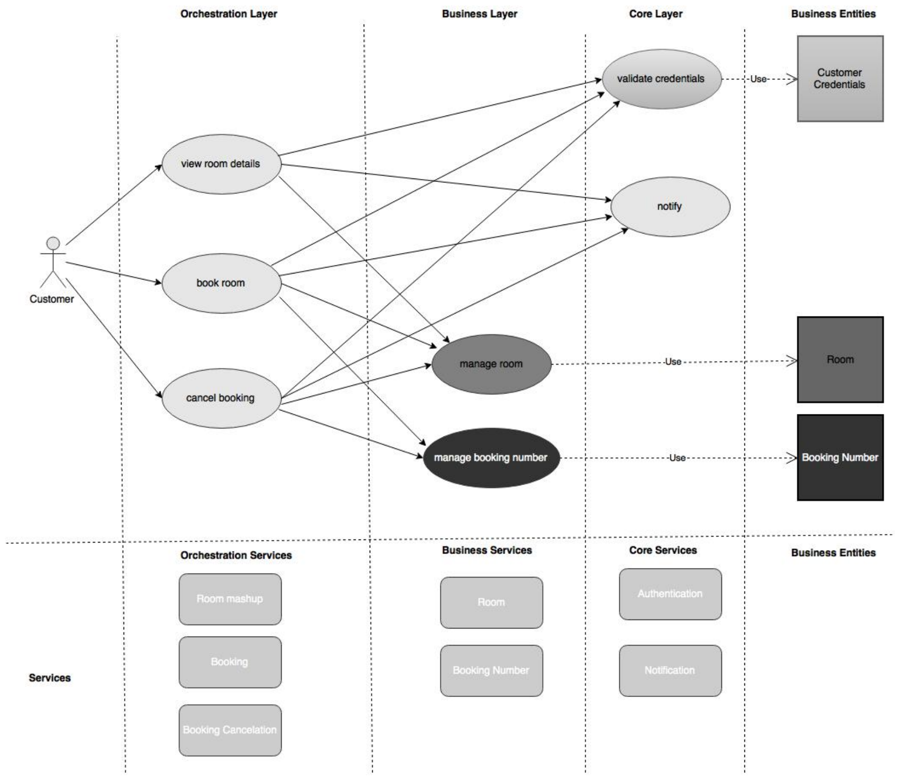

Figure 2.5: Use Case Model for identification of service candidates

Service Layers

The services or use cases obtained from the refactoring creates various levels of abstractions. The levels of abstractions represents different level of functionality. The services at the bottom layer are core services which serve many higher level services. These services are not dependent on any other services. ′Notification′ and ′Authentication′ are core services obtained from refactoring. The layer above the core layer is business layer. The business service can either have entity services, which control some related business entities or can have task services which provides some specific business logic to higher level services. ′Room′ and ′Booking Number′ are the entity services obtained from refactoring. Finally, the top layer is mashup layer which contains high level business logic and collaborates with business services as well as core services. ′Room mashup′ , ′Booking′ and ′Booking Cancelation′ are orchestration layer services obtained. The individual services as well as their respective layers are given in figure 2.5. [Far08][Emi+np][Zim+05]

Acronyms

CRUD	create, read, update, delete.
DEP	Dependency.
IDE	Integrated Development Environment.
IFBS	International Financial and Brokerage Services.
ISCI	Inter Service Coupling Index.
ODC	Operation Data Granularity.
ODG	Operation Data Granularity.
OFG	Operation Functionality Granularity.
RCS	Relative Coupling of Services.
RIS	Relative Importance of Services.
SCG	Service Capability Granularity.
SDG	Service Data Granularity.
SDLC	Software Development Life Cycle.
SFCI	Service Functional Cohesion Index.
SIDC	Service Interface Data Cohesion.
SIUC	Service Interface Usage Cohesion.
SIUC	Service Sequential Usage Cohesion.
SLC	Self Containment.
SMCI	Service Message Coupling Index.

SOAF Service Oriented Architecture Framework.

SOCI Service Operational Coupling Index.

SOG Service Operations Granularity.

SRI Service Reuse Index.

SWIFT Society for Worldwide Interbank Financial Telecommunication.

List of Figures

List of Tables

Bibliography

[Ars04] A. Arsanjani. *Service-oriented modeling and architecture How to identify, specify, and realize services for your SOA*. Tech. rep. IBM Software Group, 2004.

[Ars05] A. Arsanjani. *How to identify, specify, and realize services for your SOA*. Tech. rep. IBM, 2005.

[Che+05] F. Chen, S. Li, H. Yang, C.-H. Wang, and W. C.-C. Chu. *Feature Analysis for Service-Oriented Reengineering*. Tech. rep. De Montfort University, National Chiao Tung University, and TungHai University, 2005.

[DK07] K.-G. Doh and Y. Kim. *The Service Modeling Process Based on Use Case Refactoring*. Tech. rep. Hanyang University, 2007.

[Emi+np] C. Emig, K. Langer, K. Krutz, S. Link, C. Momm, and S. Abeck. *The SOA's Layers*. Tech. rep. Universität Karlsruhe, np.

[Far08] N. Fareghzadeh. *Service Identification Approach to SOA Development*. Tech. rep. World Academy of Science, Engineering and Technology, 2008.

[How+09] Y. Howard, N. Abbas, D. E. Millard, H. C. Davis, L. Gilbert, G. B. Wills, and R. J. Walters. *Pragmatic web service design:An agile approachwith the service responsibility and interaction designmethod*. Tech. rep. University of Southampton, 2009.

[Jac03] I. Jacobson. *Use Cases and Aspects - Working Seamlessly Together*. Tech. rep. Rational Software Corporation, 2003.

[Jac87] I. Jacobson. *Object Oriented Development in an Industrial Environment*. Tech. rep. Royal Institute of Technology, 1987.

[Kan+90] K. C. Kang, S. G. Cohen, J. A. Hess, W. E. Novak, and A. S. Peterson. *Feature Oriented Domain Analysis (FODA)*. Tech. rep. Carnegie-Mellon University, 1990.

[KY06] Y. Kim and H. Yun. *An Approach to Modeling Service-Oriented Development Process*. Tech. rep. Sookmyung Women's University, 2006.

[Mil+07] D. E. Millard, H. C. Davis, Y. Howard, L. Gilbert, R. J. Walters, N. Abbas, and G. B. Wills. *The Service Responsibility and Interaction Design Method: Using an Agile approach for Web Service Design*. Tech. rep. University of Southampton, 2007.

[Mit05] K. Mittal. *Service Oriented Unified Process(SOUP)*. Tech. rep. IBM, 2005.

[Nig05] S. Nigam. *Service Oriented Development of Applications(SODA) in Sybase Workspace*. Tech. rep. Sybase Inc, 2005.

[NJ04] P.-W. Ng and I. Jacobson. *Aspect-Oriented Software Development with Use Cases*. Addison-Wesley Professional, 2004.

[RJB99] J. Rumbaugh, I. Jacobson, and G. Booch. *The Unified Modeling Language Reference Manual*. Addison-Wesley Professional, 1999.

[RS07] P. Reldin and P. Sundling. *Explaining SOA Service Granularity– How IT-strategy shapes services*. Tech. rep. Linköping University, 2007.

[YK06] H. Yun and Y. Kim. *Service Modeling in Service-Oriented Engineering*. Tech. rep. Sookmyung Women's University, 2006.

[Zim+05] O. Zimmermann, V. Doubrovski, J. Grundler, and K. Hogg. *Service-Oriented Architecture and Business Process Choreography in an Order Management Scenario: Rationale, Concepts, Lessons Learned*. Tech. rep. IBM Software Group and IBM Global Services, 2005.