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

Acknowledgments **iii**

Abstract **iv**

1 Domain Driven Design **1**
1.1 Introduction 1
1.2 Ubiquitous Language 1
1.3 Strategical Design 3

Acronyms **5**

List of Figures **7**

List of Tables **8**

Bibliography **9**

1 Domain Driven Design

1.1 Introduction

Understanding the problem space can be a very useful way to design software. The common way to understand the problem and communicate them is by using various design models. The design models are the abstraction as well as representation of the problem space. However, when the abstractions are created, there are chances that important concepts or data are ignored or misread. This will highly impact the quality of software produced. The software thus developed may not reflect the real world situation or problem entirely.
Domain Driven Design provides a way to represent the real world problem space so that all the important concepts and data from real world remains intact in the model. The domain model thus captured respects the differences as well as the agreement in the concepts across various parts of the problem space. Moreover, it also provides a way to divide the problem space into manageable independent partitions and makes it easy for developers as well as stakeholders to focus on the area of concern as well as be more agile. Finally, the domain models act as the understandable and common view of the business for both domain experts as well as the developers. This will make sure that the software developed using domain driven design will comply with business need.[**Evans:2003aa**][**Vernon:2013aa**]
There are three basic parts to implement domain driven design:

1. Ubiquitous Language
2. Strategical Design
3. Tactical Design

However, in order to accomplish the true value of domain driven design, the tactical design is not as compulsory as the first two parts.

1.2 Ubiquitous Language

Ubiquitous Language is a common language agreed among domain experts and developers in a team. It is important to have a common understanding about the

concepts of a business which is being developed and ubiquitous language is the way to assure that. Domain Experts understand the domain in terms of their own jargon and concept. It is difficult for a developer to understand them. Usually, developers translates those jargons into the terms they understand easily during desing and implementation. However along the way of translation, major domain concepts can get lost and the immediate value of the resulting solution might decrease tremendously. In order to prevent creation of such low valued solution, there should be a meet-in-the-middle approach to define common vocabularies and concepts understood by all domain experts as well as the developers. These common vocabularies and concepts make the core of the ubiquitous language.[**Evans:2003aa**][**Vernon:2013aa**]
Along with the common vocabulary and concepts, domain models provide backbone to create ubiquitous language. The models represents not only artifacts but also functionalities, rules and strategies. It is a way to express the common understanding in a visual form providing an easy tool to comprehend.[**Evans:2003aa**][**Fowler:2006aa**]
According to [**Fowler:2003aa**], domain model should not be confused with data model which represents the business in datacentric view but rather each domain object should contain data as well as logic closely related to the data contained. Domain models are conceptual models rather than software artifacts but can be effectively visualized using UML.[**Scott:2001aa**]

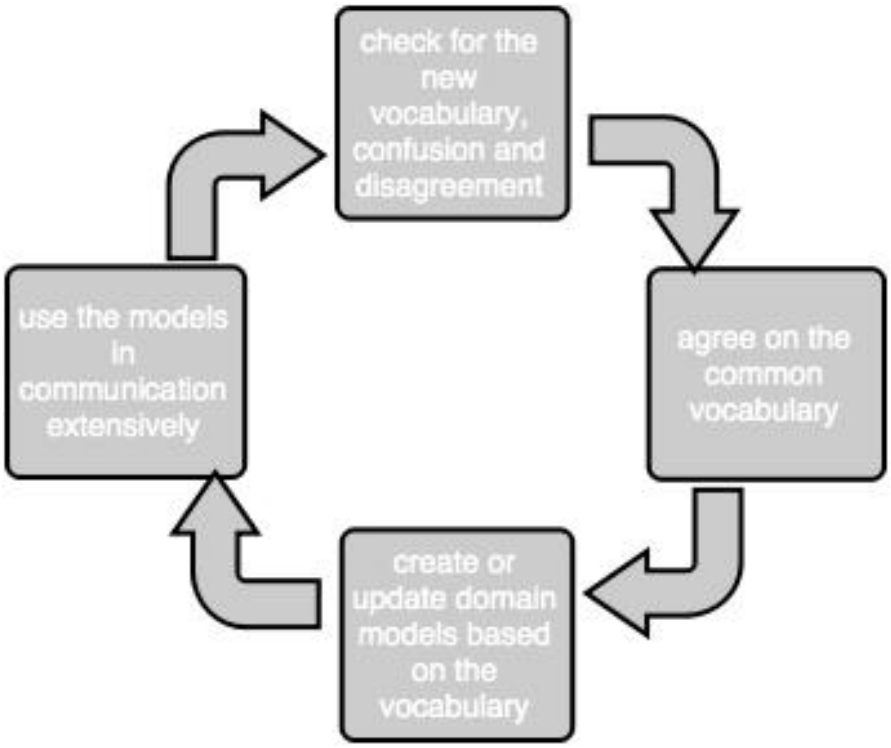

Figure 1.1: Process to define Ubiquitous Language [**Evans:2003aa**]

The figure 1.1 visualizes the process of discovering the ubiquitous language in any domain. It is not a discretely timed phenomenon but a continous procedure with various phases occuring in a cycle throughout the development cycle.
On arrival of any new term, concept or any confusion, the contextual meaning of those terms are made clear before adding to the domain vocabulary. Next, the vocabularies and concepts are used to create various domain models following UML. The domain models and various vocabularies for the common laguage among domain experts and developers within the team. It is very important to use only domain vocabularies for communication and understanding which will not only help to reflect the hidden domain concepts in the implementation but also create opportunities to create new vocabularies or refine existing ones in the event of confusion and disagreement.[**Evans:2003aa**] Thus, the common vocabulary and domain models form the core of the ubiquitous language and the only way of coming up with the better one is by applying it in communication extensively. Ubiquitous language is not just a collection or documentation of terms but is the approach of communication within a domain.

1.3 Strategical Design

When applying model-driven approach to an entire enterprise, the domain models get too large and complicated. It becomes difficult to analyse and understand all at once. Furthermore, it gets worse as the system gets bigger. The strategical design provides a way to divide the entire domain models into small,manageable and interoperable parts which can work together with low dependency in order to reflect the functionalities of the entire domain. The goal is to divide the system into modular parts which can be easily integrated. Additionally, the all-cohesive unified domain models of the entire enterprise cannot reflect differences in contextual vocabularies and concepts.[**Fowler:2014ab**][**Evans:2003aa**][**Vernon:2013aa**] The strategical design specifies three major steps.

Step 1: Divide problem domain into subdomains

The domain represents the problem being solved by the software. The domain can be divided into various sub-domains based on the organizational structure of the enterprise, each sub-domain responsible for certain area of the problem. The [**Engels:2015aa**] provides a comprehensive set of steps to identify sub-domains.

1. **Identify core business functionalities and map them into domain:**
   A core business functionality represents the direct business capability which holds high importance and should be provided by the enterprise in order for it to succeed. For example, for any general e-commerce enterprise, the core focus will on the high amount orders being received from the customers and maintain the inventory to fulfil the orders. So, for those kind of e-commerce enterprises, 'Order Management' and 'Inventory Management' will be the some of the core domains.

2. **Identify generic and supporting subdomains:**
   The business capabilities which are viable for the success of business but not represent the specialization of the enterprise falls into supporting subdomains. The supporting subdomain does not require the enterprise to excel in these areas. Additionally, if the functionalities are not specific to the business but in a way support the business functionalities, then these are covered by generic subdomains. For example: for an e-commerce enterprise, 'Payment Management' can be a

supporting subdomain whereas 'Reporting' and 'Authentication' can be generic subdomains.

3. **Divide existing subdomains having multiple independent strategies:**
   The subdomains found from earlier steps are analyzed to check if there are mutually independent straties to handle the same functionality. The subdomain can then be further divided into multiple subdomains along the dimension of strategies. For example: the 'Payment Managment' subdomain discovered in earlier step can have different way of handling online payment depending upon the provider such as bank or paypal etc. In that case 'Payment Management' can be further divided into 'Bank Payment Management' and 'Paypal Payment Management'.

Step 2: Indentify bounded contexts

Step 3: Discover relationship among bounded contexts using Context Map

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.

UML Unified Modeling Language.

List of Figures

1.1 Process to define Ubiquitous Language [**Evans:2003aa**] 2

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.