

AN APPROACH FOR INCLUDING BUSINESS REQUIREMENTS
TO SOA DESIGN

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

MURAT OCAKTÜRK

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE
IN
COMPUTER ENGINEERING

FEBRUARY 2010

Approval of the thesis:

**AN APPROACH FOR INCLUDING BUSINESS REQUIREMENTS
TO SOA DESIGN**

submitted by **MURAT OCAKTÜRK** in partial fulfillment of the requirements for
the degree of **Master of Science in Computer Engineering Department, Middle
East Technical University** by,

Prof. Dr. Canan Özgen
Dean, Graduate School of **Natural and Applied Sciences**

Prof. Dr. Müslim Bozyiğit
Head of Department, **Computer Engineering**

Assoc. Prof. Dr. Ali Hikmet Doğru
Supervisor, **Computer Engineering Dept., METU**

Examining Committee Members:

Asst. Prof. Dr. Tolga Can
Computer Engineering Dept., METU

Assoc. Prof. Dr. Ali Hikmet Doğru
Computer Engineering Dept., METU

Dr. Bülent Mehmet Adak
Software Engineering Dept., Aselsan Inc.

Dr. Cevat Şener
Computer Engineering Dept., METU

Senior Lead Software Design Eng. Bülent Durak
Software Engineering Dept., Aselsan Inc.

Date:

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Last name : **Murat OCAKTÜRK**

Signature :

ABSTRACT

AN APPROACH FOR INCLUDING BUSINESS REQUIREMENTS TO SOA DESIGN

OCAKTÜRK, Murat

M.Sc., Department of Computer Engineering
Supervisor: Assoc. Prof. Dr. Ali Hikmet DOĞRU

February 2010, 62 pages

In this thesis, a service oriented decomposition approach: Use case Driven Service Oriented Architecture (UDSOA), is introduced to close the gap between business requirements and SOA (Service Oriented Architecture) design by including business use cases and system use cases into decomposition process. The approach is constructed upon Service Oriented Software Engineering (SOSE) modeling technique and aims to fill the deficits of it at the decomposition phase. Further, it aims to involve both business vision and Information Technologies concerns in the decomposition process. This approach starts with functional top-down decomposition of the domain. Then, business use cases are used for further decomposition because of their high-level view. This connects the business requirements and our SOA design. Also it raises the level of abstraction which allows us to focus on business services. Second step of the SOA approach uses system use cases to continue decomposition. System use cases help discovering technical web services and allocating them on the decomposition tree. Service oriented analysis also helps separating business and technical services in tightly coupled architecture conditions. Those two steps together bring quality in to both problem and solution domains.

Keywords: Service Oriented Architecture, Service Oriented System Design, Functional Decomposition, Process Models, Business Use Cases, System Use Cases, Service Oriented Software Engineering Modeling Language

ÖZ

İŞ GEREKSİNİMLERİNİ SERVİS YÖNELİMLİ MİMARİ TASARIMINDA KULLANMAK İÇİN BİR YAKLAŞIM

OCAKTÜRK, Murat

Yüksek Lisans, Bilgisayar Mühendisliği Bölümü

Tez Yöneticisi: Doç. Dr. Ali Hikmet DOĞRU

Şubat 2010, 62 sayfa

Bu tezde; iş kullanım durumlarını ve sistem kullanım durumlarını ayrıştırma sürecine katarak, iş gereksinimleri ile SOA (Servis Yönelimli Mimari) tasarımı arasındaki boşluğu kapatacak servis yönelimli ayrıştırma yaklaşımı olan UDSOA (Kullanım Durumu Güdümlü Servis Yönelimli Mimari) tanıtılmaktadır. Yaklaşım, SOSE (Servis Yönelimli Yazılım Mühendisliği) modelleme tekniği taban alınarak geliştirilmiştir ve onun ayrıştırma adımıdaki eksiklikleri gidermeyi amaçlamıştır. Ayrıca, iş geniş görüşü ile BT (Bilgi Teknolojisi) kaygılarını ayrıştırma işlemine katmayı amaçlamıştır. Yaklaşım, alanın süreçsel ayrıştırılması adımı ile başlar. Sonrasında, üst seviye görünümleri nedeniyle iş kullanım durumları kullanılarak ayrıştırma devam eder. Bu işlem, iş gereksinimlerimiz ile SOA tasarımımızı birbirine bağladığı gibi, soyutluk seviyesini arttırarak iş servislerine yoğunlaşmayı sağlar. SOA yaklaşımının ikinci adımı sistem kullanım durumları ile ayrıştırmaya devam eder. Sistem kullanım durumları, teknik servislerin keşfedilmesini ve ayrıştırma ağacına yerleştirilmesini sağlar. Son olarak, servis yönelimli analiz sıkı bağlı mimari durumlarında iş ve teknik servislerin birbirinden ayrılmasını sağlar. Bu iki adımın

birlikte yapılması hem problem hem de çözüm alanlarında doğru ayrıştırmayı getirecektir.

Anahtar Kelimeler: Servis Yönelimli Mimari, Servis Yönelimli Sistem Tasarımı, Süreçsel Ayrıştırma, Süreç Modelleri, İş Kullanım Durumları, Sistem Kullanım Durumları, Servis Yönelimli Yazılım Mühendisliği Modelleme Dili

ACKNOWLEDGMENTS

First of all, I would like to express my sincere thanks to my supervisor, Assoc. Prof. Dr. Ali Hikmet DOĞRU for his efforts and guidance throughout this thesis work.

I would like to thank my family for their support and patience.

I would also like to thank my friend Eren Koçak AKBIYIK, and all my friends not named for their support during my M.Sc.

TABLE OF CONTENTS

ABSTRACT	iv
ÖZ	vi
ACKNOWLEDGMENTS	viii
TABLE OF CONTENTS	ix
LIST OF TABLES	xi
LIST OF FIGURES	xii
LIST OF ABBREVIATIONS	xiv
CHAPTERS	
1. INTRODUCTION	1
1.1 Evolution of Decomposition in Software	2
1.2 SOSE Approach.....	3
1.3 Bridging Between Business and Technology	4
1.4 Organization of the Thesis.....	5
2. BACKGROUND	6
2.1 Service-Oriented Architecture.....	6
2.2 Web Services.....	7
2.2.1 Web Services Orchestration and Choreography.....	8
2.3 SOSE Approach.....	9
2.3.1 SOSEML Notation	9
2.3.2 Main Steps of the SOSE Approach	12
3. CONNECTING BUSINESS REQUIREMENTS AND SOA DESIGN	15
3.1 Philosophy of the Approach	15
3.2 Decomposition Steps of the Approach	16
3.2.1 Functional Decomposition with Business Use Cases	18
3.2.1.1 Introduction to Functional Decomposition	19
3.2.1.2 Defining Functional Areas.....	20
3.2.1.3 Business Use Cases Integration	20
3.2.2 Service Oriented Analysis.....	23

3.2.2.1	Identifying System Use Cases	24
3.2.2.2	Separating Technology from Business Services	25
3.3	Similar Studies.....	28
4.	ADAPTING SOSECASE TO UDSOA.....	29
4.1	Current Features of SOSECASE	29
4.2	Additions to SOSECASE	31
4.2.1	Use Cases and SOA Design Mapping Window	31
4.2.1.1	Tree View Notation	33
4.2.1.2	Menu Operations	34
5.	A CASE STUDY: MODELING A GIS APPLICATION.....	37
5.1	Business Scenario: GIS Application.....	37
5.2	Description of the System.....	40
5.3	Decomposition of the System.....	41
5.3.1	Decomposing in Business Scope.....	42
5.3.1.1	Defining Functional Areas.....	42
5.3.1.2	Defining Business Use Cases	43
5.3.2	Decomposition in System Scope	45
5.3.2.1	2D Analyses Subsystem	47
5.3.2.2	3D Analyses Subsystem	48
5.3.2.3	3D Terrain Layer Subsystem.....	48
5.3.2.4	Symbology Layer Subsystem	50
5.3.2.5	3D Layers Subsystem.....	52
5.3.2.6	Layer Operations Subsystem	53
5.3.2.7	Extent Operations Subsystem.....	54
5.4	Case Study Results	54
6.	CONCLUSION	56
6.1	Work Conducted.....	56
6.2	Comments.....	57
6.3	Future Work.....	58
	REFERENCES.....	60

LIST OF TABLES

TABLES

Table 1 – Graphical Modeling Elements in SOSEML.....	10
Table 2 – BPEL Symbols Used in SOSEML.....	11
Table 3 – Terms Used by GIS Application.....	38

LIST OF FIGURES

FIGURES

Figure 1 – Evaluation of Decomposition Approaches (Adapted from [13])	3
Figure 2 – Service Definition Hierarchy	4
Figure 3 – Web Services Orchestration and Choreography (Adapted from [21])	9
Figure 4 – General Structure of Decomposition Tree in SOSEML	13
Figure 5 – SOSE Approach Decomposition and Modeling Difference	14
Figure 6 – Steps of UDSOA Approach	17
Figure 7 – Comparison of SOSE and UDSOA	18
Figure 8 – Decomposed Domain into its Functional Areas in SOSEML Notation ...	20
Figure 9 – Relationship between business use cases and business processes	22
Figure 10 – Business Use Case in a Word Document	22
Figure 11 – Realization of Business Use Case and System Use Case	25
Figure 12 – Separating Business and Technology into Different Services	26
Figure 13 – General View of the SOSECASE Main Window	30
Figure 14 – Tools Menu Item to Reach New Window	32
Figure 15 – Use Case and Decomposition Tree Mapping Window	32
Figure 16 – Tree View Notation	33
Figure 17 – Menu Items of Main Menu	34
Figure 18 – Decomposition Tree View after Realization Operation	35
Figure 19 – Use Cases’ Realization States	36
Figure 20 – A Sample Snapshot of a GIS Application	41
Figure 21 – Domain Decomposition of a GIS Application Scenario	42
Figure 22 – Further Decomposed Domain with Business Use Cases	44
Figure 23 – Business Use Cases’ Realization in SOSECASE	45

Figure 24 – 2D Analyses Subsystem	47
Figure 25 – 3D Analyses Subsystem	48
Figure 26 – 3D Terrain Rendering before Analyzing	49
Figure 27 – 3D Terrain Rendering after Analyzing	50
Figure 28 – Symbology Layer Subsystem before Separation of Technology	50
Figure 29 – Symbology Layer Subsystem after Separation of Technology	51
Figure 30 – 3D Layers Subsystem	52
Figure 31 – Layer Operations Subsystem	53

LIST OF ABBREVIATIONS

API	Application Programming Interface
BPEL	Business Process Execution Language
BUC	Business Use Case
CORBA	Common Object Request Broker Architecture
COSEML	Component Oriented Software Engineering Modeling Language
COTS	Commercial off-the-shelf
CPU	Central Processing Unit
DBMS	Database Management System
DCOM	Distributed Component Object Model
DDS	Data Distribution System
DTED	Digital Terrain Elevation Data
GIS	Geographical Information Systems
HTTP	Hyper Text Transfer Protocol
IT	Information Technologies
MSMQ	Microsoft Message Queue
OO	Object Oriented
RCP	Rich Client Platform
RPC	Remote Procedure Call
SOA	Service Oriented Architecture
SOAP	Simple Object Access Protocol
SOSE	Service Oriented Software Engineering
SOSECASE	Service Oriented Software Engineering Tool
SOSEML	Service Oriented Software Engineering Modeling Language
SUC	System Use Case
SVG	Scalable Vector Graphics

UDDI	Universal Description, Discovery and Integration
UDSOA	Use case Driven Service Oriented Architecture
UML	Unified Modeling Language
W3C	World Wide Web Consortium
WCF	Windows Communication Foundation
WSDL	Web Service Description Language
XML	Extended Markup Language

CHAPTER 1

INTRODUCTION

As human beings, we are passionate about new ideas that promise to transform our lives and create new opportunities. We are also curious about rapidly replacing old technologies with new ones. This passion helps creating tomorrow's software by adding on top of the existing technology layers of yesterday and today.

Not long after the new millennium, displeasure over the maturities of interoperability, reusability, and other issues drove the software community to come up with the Service-Oriented Architecture (SOA) paradigm. Service-oriented architecture is a set of business-aligned services that are combined (composed and choreographed) to fulfill business goals [2]. Information Technology (IT) systems use these services to support rapidly changing business needs. The services are declared as a set of interfaces without any dependencies on their implementation mechanism or location. This architectural style allows better alignment of required business capabilities with IT functions. The SOA vision also addresses the challenges of tightly coupled software and advocates an architecture that relies on the loose coupling of assets [3]. Indeed, the list of advantages continues to grow.

After the debut of SOA paradigm, software community tried to identify new ideas that will make SOA more effective and raise the abstraction level again. A way to discover business services of complex systems was needed. So, they come up with decomposition approach, an old technique that has been used by software engineers since early 1960's. This approach prescribes decomposing complex systems into smaller, more manageable ones that are easier to control, and then treating the whole system as a composition of its parts. The same applies to complex software development initiatives [12].

1.1 Evolution of Decomposition in Software

The first software decomposition approach introduced was splitting mainframe applications into separate jobs, each implemented by a separate program. Later, as more understanding into the program internals was gained, each program itself was split into modules or subroutines, according to their function.

The object-oriented (OO) paradigm introduced by Simula and Smalltalk in the 1970s strengthened decomposition adoption by introducing data-centric objects, each of which implemented a model of a real thing. The idea was to represent in software the "things of the problem domain," such as customer, order, or trade [13]. However, the abstractions provided by objects were too fine-grained and had too many technical concepts that have no meaning at the business level.

In the continued search for a better design paradigm, a different approach to decomposition was introduced in the late 1990s: components. The idea was to fix the problems of OO by raising the level of abstraction, increasing granularity, and creating a tighter linkage with the business goals.

Introduction of component oriented software development allowed for the creation of flexible, better structured, and more manageable software applications. However, it did not solve the main enterprise IT problem: its application-centric nature. Both objects and components provide better design and development approaches for individual applications. SOA brings decomposition to a higher level, as shown in the Figure 1. Instead of trying to decompose individual applications, it decomposes the entire enterprise IT functionality.

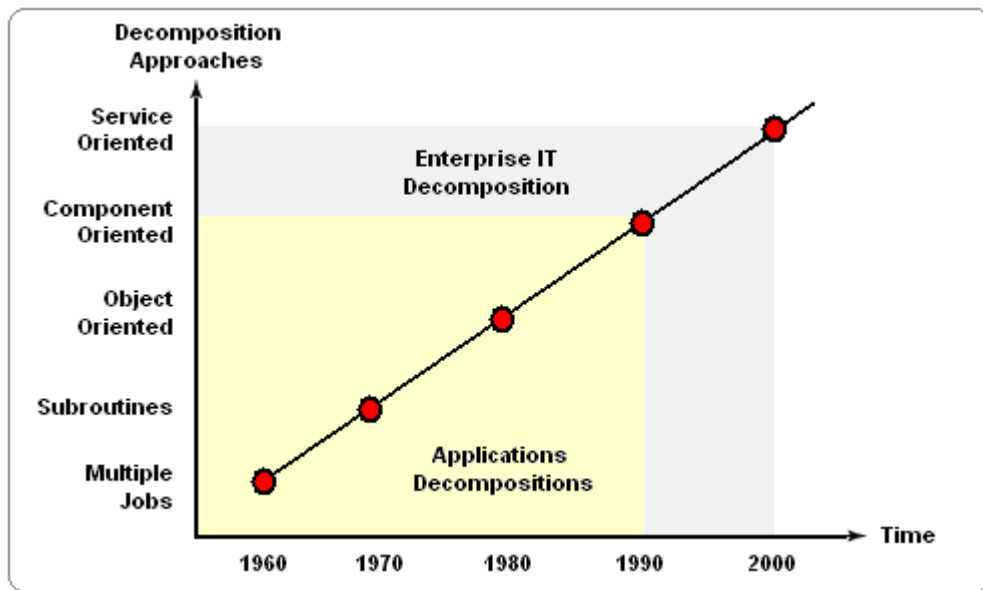


Figure 1 – Evaluation of Decomposition Approaches (Adapted from [13])

1.2 SOSE Approach

The works of A.H. Dogru, M.M. Tanik in 2003 and A. Manzer, A.H. Dogru in 2007 triggered new ideas of adapting those works to SOA. Service Oriented Software Engineering (SOSE) approach that introduced a SOA alignment and hierarchical decomposition was introduced by Eren Kocak Akbiyik in 2008. This approach enhanced the ideas of constructing decomposition tree based on SOA elements and tried to present a modeling approach for SOA based software systems.

This methodology basically offers a procedural (functional) top-down decomposition of a given software system allowing several abstraction levels. At the higher levels of the decomposition, the system is divided into abstract nodes that correspond to process models in the decomposition tree. Any node is a process and keeps the sequence and the state information for the possible sub-processes in this decomposition tree. Nodes which are defined as process models may include some sub-nodes to present details for the intermediate levels of the model. Eventually at

the leaf level, process models are decomposed into existing web services as the atomic units of system execution. All processes constructing the system decomposition tree are modeled with Business Process Execution Language (BPEL) to expose the flow-level details of the design. This modeling technique is also supported with a graphical modeling language referred to as Service Oriented Software Engineering Modeling Language (SOSEML) [1].

1.3 Bridging Between Business and Technology

The power of SOA is in its ability in business process integration and reuse. SOA achieves this in two ways: By encapsulating functional capabilities separated from their implementations and by providing facilities for managing coupling between functional capabilities [8]. Modeling can be used to bridge the gap between business requirements and a deployed service-based solution. SOA model should raise the level of abstraction so that, it should allow to focus on business services. SOSE approach provides abstraction by hierarchical decomposition as the Figure 2 depicts.

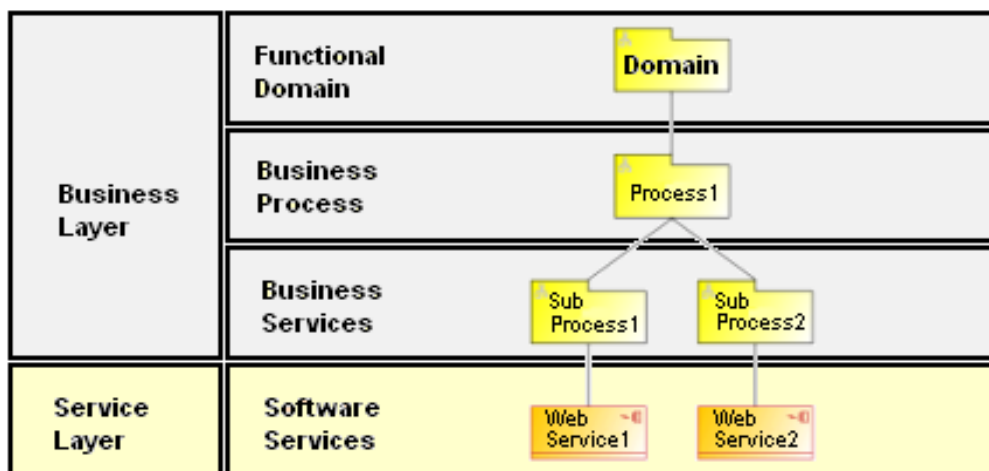


Figure 2 – Service Definition Hierarchy

SOSE approach aimed to decompose domain just because it is complex at the top. So SOSE applied a functional top-down decomposition to make the domain more manageable. However, the decomposition of domain is not that much easy and cannot be done by just one group of people. Business analysts must be involved in decomposition process, because they know business organizational and operational requirements necessary to meet business goals and objectives that achieve some business vision. On the other hand, technical or software team must be involved in decomposition process too. Because they are skilled to deal with IT concerns, such as reuse, coupling, distribution, security, persistence, data integrity, concurrency, failure recovery, and so forth.

Connecting business requirements and SOA solution is hard to do if business requirements are given as a simple list and SOA solution describes a collection of web services. The research in this thesis tries to put business use cases and system use cases into the process of decomposition, thus business requirements will be included to SOA solution. Business use cases are used to state WHAT system does, whereas system use cases are used to state HOW system does. The approach in this thesis is called as UDSOA (Use Case Driven Service Oriented Architecture), since it uses use cases as the key decomposition element.

1.4 Organization of the Thesis

This thesis work includes 6 chapters. In Chapter 2, necessary background on service oriented architecture, web services and the basics of the SOSE philosophy are included. Chapter 3 defines the proposed approach from domain decomposition to separating web services on the leaf level. In Chapter 4, the additions to SOSECASE modeling tool are introduced. Chapter 5 includes an extensive case study to represent all steps of the approach with graphical presentations. Finally Chapter 6 concludes the thesis work and states future work.

CHAPTER 2

BACKGROUND

2.1 Service-Oriented Architecture

There are several different definitions of Service Oriented Architecture (SOA). The World Wide Web Consortium (W3C) for example refers to SOA as “A set of components which can be invoked, and whose interface descriptions can be published and discovered”. However, a wider definition for SOA is required. CDBI recommends SOA is more usefully defined as:

“The policies, practices, frameworks that enable application functionality to be provided and consumed as sets of services published at a granularity relevant to the service consumer. Services can be invoked, published and discovered, and are abstracted away from the implementation using a single, standards-based form of interface [18].”

SOA aims to allow users to string together fairly large chunks of functionality to form ad hoc applications that are built almost entirely from existing software services. The larger the chunks, the fewer the interface points required implementing any given set of functionality; however, very large chunks of functionality may not prove sufficiently granular for easy reuse. The great promise of SOA suggests that the cost of creating the n-th application is low, as all of the software required already exists to satisfy the requirements of other applications. Ideally, one requires only orchestration to produce a new application.

Architectures can operate independently of specific technologies. Designers can implement SOA using a wide range of technologies, including SOAP, RPC, DCOM,

CORBA, Web Services, and WCF. SOA enables the development of applications that are built by combining loosely coupled and interoperable services. These services interoperate based on a formal definition (e.g. WSDL) that is independent of the underlying platform and programming language. The interface definition hides the implementation of the language-specific service. SOA-based systems can therefore function independently of development technologies and platforms.

A service in SOA is an application function packaged as a reusable component for use in a business process. It either provides information or facilitates a change to business data from one valid and consistent state to another.

2.2 Web Services

A Web service is a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL). Other systems interact with the Web service in a manner described by its description using SOAP messages, typically by using HTTP with an XML serialization in conjunction with other Web-related standards [19].

SOA presents the big picture of what you can do with web services. Web services specifications define the details needed to implement services and interact with them [14]. However, SOA is an approach to build distributed systems that deliver application functionality as services to end-user applications or to build other services.

Although web services are similar to some of their predecessors, this technology differs from them in several aspects. First of all, web services are supported by all major software vendors. Therefore they are the first technology that fulfills the promise of universal interoperability between applications running on different platforms.

Web services are based on invocation using SOAP messages which are described using WSDL over a standard protocol such as HTTP. Use of web services is a best practice when communicating with external business partners. SOAP, WSDL, and UDDI are XML based, making web services protocol messages and descriptions not just machine-processable, but also human readable.

2.2.1 Web Services Orchestration and Choreography

IT organizations need the agility to adapt to customer requirements and changing market conditions. Web services orchestration and choreography standards are efforts that can be long-term solutions for those problems. The terms orchestration and choreography describe two aspects of emerging standards for creating business processes from multiple Web services. However, the two terms overlap somewhat.

Orchestration refers to an executable process that may interact with both internal and external web services. Orchestration describes how web services interact at the message level, including the business logic and execution order of the interactions. Choreography is more collaborative, where each party involved in the process describes the part they play in the interaction. Orchestration differs from choreography that it describes a process flow between services, controlled by a single party; choreography tracks the sequence of messages involving multiple parties, where no party owns the conversation [21].

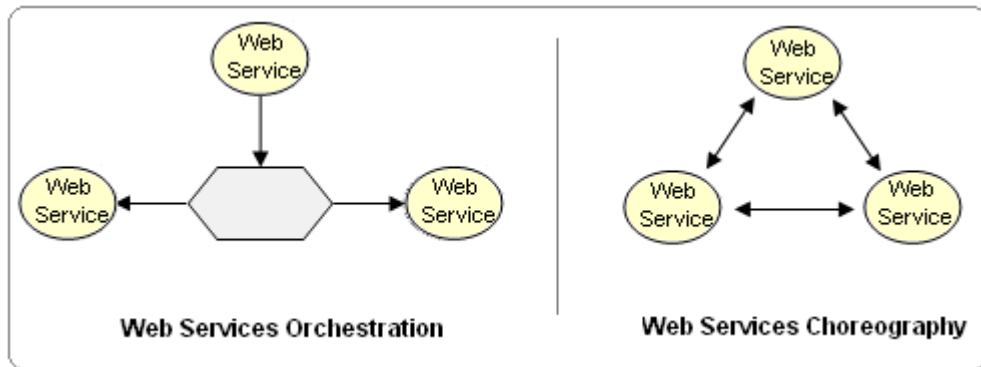


Figure 3 – Web Services Orchestration and Choreography (Adapted from [21])

2.3 SOSE Approach



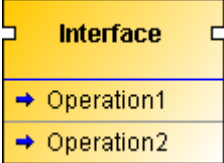
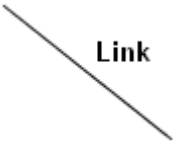
The approach introduced in this thesis is constructed upon SOSE modeling technique, which is introduced in a thesis study by Eren Kocak Akbiyik. Technique is supported with a modeling language referred to as SOSEML (Service Oriented Software Engineering Modeling Language).

SOSEML is a graphical modeling language which supports a top down approach in the engineering of complex business processes. It purposes to provide a common and acceptable system design approach for SOA based software development [1].

2.3.1 SOSEML Notation















SOSEML is a completely graphical modeling language that is used to construct hierarchical decomposition tree for the modeling activity of SOSE methodology. There are four basic graphical modeling elements in SOSEML.

Table 1 – Graphical Modeling Elements in SOSEML

Graphical Element	Description
 <p>Process</p>	<p>A “process” is represented by a yellow package symbol with a small process icon on the left upper corner. Processes are the main building blocks in a decomposition tree. The whole system and all sub processes are shown by process symbols in the model.</p>
 <p>Web Service Interface</p>	<p>“Web services” are represented by orange boxes. The small icon on the right corner of the box implies that a web service is a remote component. A web service can publish various methods or operations in its interface where some of those methods can be used in different processes. Therefore, in SOSEML, a web service can have more than one web service interface. Names of all interfaces belong to the web service are also shown in the box symbol.</p>
 <p>Interface → Operation1 → Operation2</p>	<p>“Web service interfaces” are also shown in SOSE models and represented by orange boxes similar to web service symbols. An interface symbol contains the names of the operations which can be called by the requesters through this interface.</p>
 <p>Link</p>	<p>“Links” are represented by standard black lines in the model and are used to connect other graphical elements.</p>

Those graphical elements above are used in building decomposition tree. In the modeling phase, BPEL (Business Process Execution Language) is used as a process modeling language. BPEL is an XML based language with operational semantics; its control constructs are to those found in programming languages such as Java, C++, or C# [15]. In SOSEML, all process models are created and designed graphically using the exact BPEL syntax. Table 2 shows BPEL symbols that are used in SOSEML.

Table 2 – BPEL Symbols Used in SOSEML

Name	Symbol	Description
Invoke		The <invoke> activity is used to invoke the web service operations provided by partners.
Receive		A <receive> activity is used to receive requests in a BPEL business process to provide services to its partners.
Reply		A <reply> activity is used to send a response to a request previously accepted through a <receive> activity.
Assign		The <assign> activity is used to copy data from one variable to another and construct and insert new data using expressions and literal values.
Empty		An activity that does nothing is defined by the <empty> tag.
If		The <if> activity expresses a conditional behavior.
Pick		The <pick> activity is used to wait for the occurrence of one of a set of events and then perform an activity associated with the event.
While		A <while> activity is used to define an iterative activity. The iterative activity is performed until the specified Boolean condition no longer holds true.
For Each		A <For Each> activity is also used to define iterative activities over a counter value for a group of items.
Repeat Until		A <Repeat Until> activity is also used to define an iterative activity similar to <while> activity.
Wait		A <wait> activity is used to specify a delay for a certain period of time or until a certain deadline is reached.
Sequence		A <sequence> activity is used to define activities that need to be performed in a sequential order.
Scope		A <scope> defines behavior contexts for activities. They provide fault handlers, event handlers, compensation handlers, data variables, and correlation sets for activities.
Flow		The <flow> activity provides concurrent execution of enclosed activities and their synchronization.

2.3.2 Main Steps of the SOSE Approach

SOSE consist basically of two main steps:

- Constructing the hierarchical decomposition tree,
- Creating process models for each process in decomposition tree.

First step accepts the whole system as a large and complex business process which realizes the target business goal. Then it decomposes the whole system into high level sub-processes. These high level sub-processes are also decomposed into different sub-processes too. By this way, a decomposition tree is constructed. Processes are decomposed iteratively until atomic processes are reached. An atomic process is a process that does not include any sub-processes. In SOSE, these are accepted as existing web services residing at the leaf level of the decomposition tree [1]. The book named “Executing SOA” accepts this idea by that line:

“In a decomposed process model, each lower-level (typically the leaf-level, that is, the lowest level of decomposition) sub-process is a strong candidate to be exposed as a service [10].”

In SOSE, highest level processes are the most abstract part of the decomposition. Intermediate processes are part of functional decomposition and used by parent processes. The leaf level processes are web services. Finally, these web services are connected to the leaf level process to indicate the “uses” relationship. Furthermore, different service interfaces can also be added at the bottom of the decomposition tree. General structure of a decomposition tree in SOSEML notation is depicted in the Figure 4.

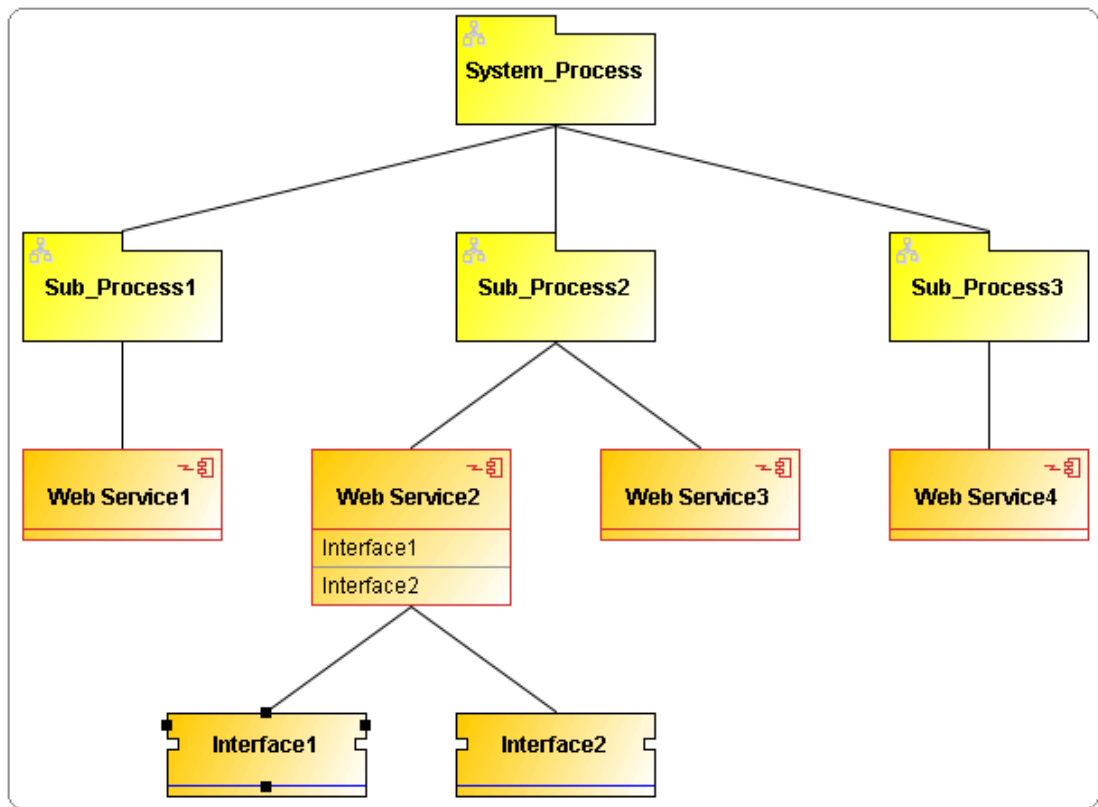


Figure 4 – General Structure of Decomposition Tree in SOSEML

The second step of SOSE modeling activity is designing process models for each process in decomposition tree. Business process models are used to define the initial details of the business flow in an algorithmic way. Each process includes flow, resource allocation, variables, and synchronization data with their children processes.

SOSE methodology uses XML based standard Business Process Execution Language (BPEL) to model business processes. Simply, BPEL process receives a request (XML based message) to start operating, then invokes the children web services, and finally responds to the caller. BPEL invocations can be either synchronously or asynchronously.

As mentioned in previous sections, SOSE methodology starts with decomposition of software system in a top-down manner. After constructing decomposition tree, SOSE methodology offers modeling composition process starting from the leaf level processes and continuing towards the high level processes and finally modeling the root process. This approach is depicted in the Figure 5.

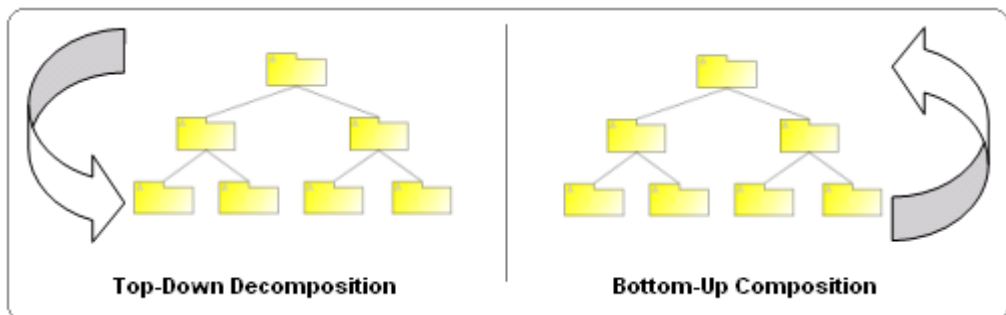


Figure 5 – SOSE Approach Decomposition and Modeling Difference

CHAPTER 3

CONNECTING BUSINESS REQUIREMENTS AND SOA DESIGN

3.1 Philosophy of the Approach

Since the introduction of Service-Oriented Architecture (SOA), software community has been trying to make it more effective. Those efforts bring SOA a particular maturity level especially in the technological dimension. Then, the work is shifted to SOA design approaches. Service Oriented Software Engineering (SOSE) is an example for such kind of effort.

SOSE provides an acceptable system design technique for SOA based software development. It supports top-down decomposition and then bottom-up composition approach in the engineering of complex systems. However, SOSE methodology didn't give the exact details of how we can do top-down decomposition for a complex problem.

The enterprise is still seeking a way of designing service oriented architecture solutions that are connected to the business requirements that they fulfill. The approach in this thesis is referred to as Use case Driven Service Oriented Architecture (UDSOA) that aims to provide a way to construct a better decomposition tree by connecting our business requirements and our SOSE based SOA design. UDSOA tries to fill the communication gaps between the problem and solution domains. If it succeeds, the SOA design will be more reusable and will be ready for future software updates.

SOA design needs to more closely resemble to business services and give answer to how those services might meet business goals and objectives [8]. So, UDSOA tries to formalize business requirements and raise the level of abstraction.

3.2 Decomposition Steps of the Approach

The UDSOA approach basically has two steps in constructing a decomposition tree:

- Decomposition of the domain with business use cases,
- Service oriented analysis with system use cases

Figure 6 depicts the steps of UDSOA approach in detail.

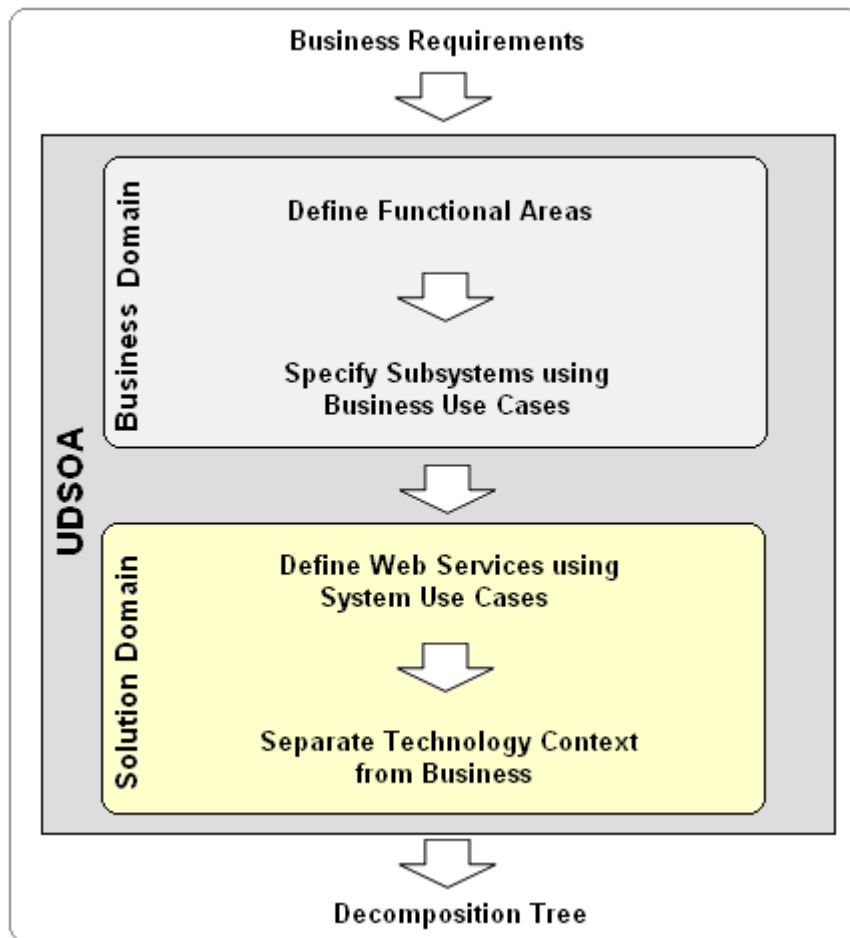


Figure 6 – Steps of UDSOA Approach

In the first step, the domain will be decomposed into its business architecture consisting of business processes and sub-processes. To add business requirements to the process, business use cases will be used to further decompose the domain and to define subsystems. Subsystems represent the business operational requirements necessary to achieve some business goal or objective or functional and non-functional requirements specified in a business use case. The reason for using business use cases is that they provide higher-level view of the business requirements. At the end of the first step, we will have subsystems that realize our business use cases.

In the second step, we move more into design-driven and architecture-driven decisions. Instead of business use cases, we will use system use cases to decompose further, the domain. System use cases are produced by hardware or software development team for their requirements. At the end, the services that have business and technical functionality will be broken up into smaller and distinct parts to decrease integration challenges and ease future software updates. After the last step, the decomposition tree will be ready to be composed into an orchestration as SOSE methodology suggests. Figure 7 shows the situation of UDSOA approach within the efforts related to SOSE methodology.

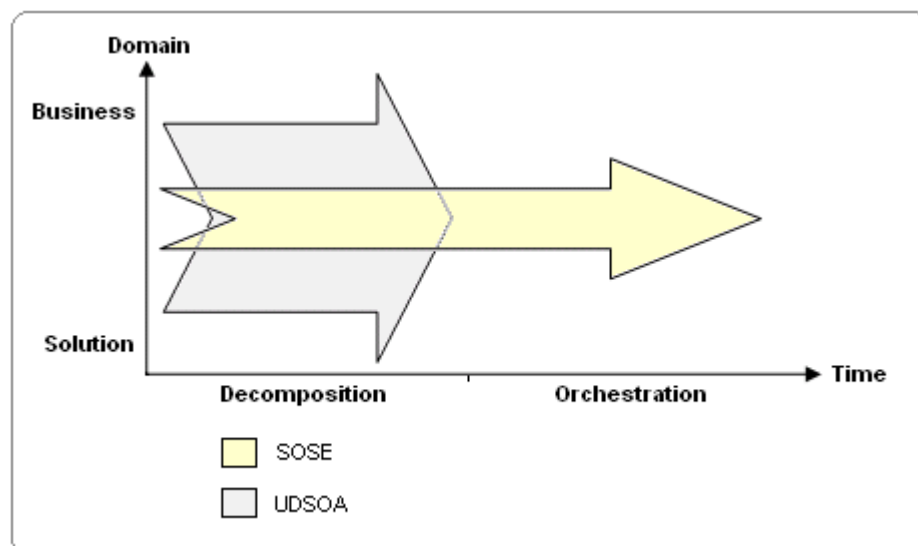


Figure 7 – Comparison of SOSE and UDSOA

3.2.1 *Functional Decomposition with Business Use Cases*

Decomposition is partitioning of large, coarse-grained, and unwieldy services into smaller, fine-grained services. The logical breakdown is typically applied to services that offer a wide range of business and technology functionality. Additionally, it

encourages the separation of software assets into private entities to increase their reusability [3].

3.2.1.1 Introduction to Functional Decomposition

“Divide and conquer” paradigm is not a new idea to solve complex systems and it is used in both COSE and SOSE [6]. However, dividing, or decomposition in software jargon, can be done in different dimensions of software systems. For all methodologies, the offered models represent three dimensions of software systems which are data, function and structure [7]. Old methodologies suggested software systems which have set of functions. So, a “functional” decomposition of the system was essential. Object oriented methodologies breaks a large system down into smaller classes or objects (data) that are responsible for some part of the problem domain. Component based methodologies; on the other hand, decompose systems with respect to their “structure”. The UDSOA approach uses functional (procedural) decomposition like SOSE.

From of the business perspective, the domain consists of a set of functional areas, corresponding to the top layers. Functional areas are used to categorize and group the services that will be identified. Specifying those functional areas is called “functional decomposition”. The grouping of services derived from the functional areas results in service partitions that are represented in separate packages in the service design model [10].

When defining the functional areas, we must define the scope of the function. Here, scope means being within the enterprise or a business line. A function can be within the enterprise or it can be across one or more business lines. The more a function is within the enterprise, the higher it will be at the decomposition tree. However, the services that implement those functions will be coarse-grained and tend to have a rather low reuse potential, which means that reuse potential and business value correlate negatively [5].

3.2.1.2 Defining Functional Areas

UDSOA approach begins with constructing a hierarchical decomposition tree by functional decomposition. Figure 8 shows a domain which is decomposed into its functional areas in SOSEML notation.

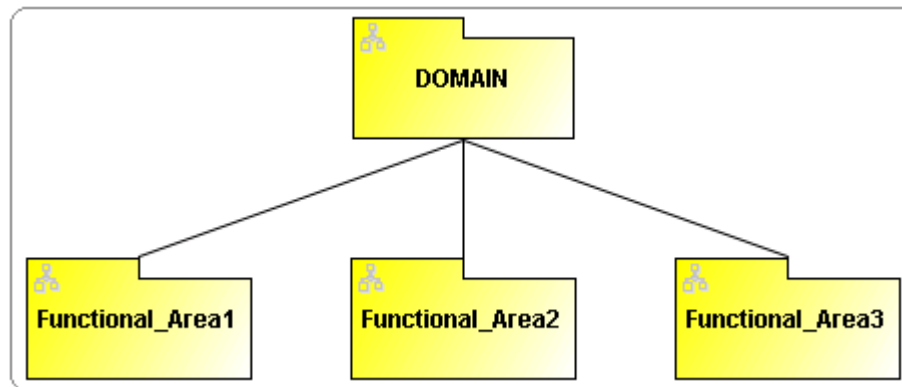


Figure 8 – Decomposed Domain into its Functional Areas in SOSEML Notation

Figure 8 shows a domain that has three main groups of operations. Those functional areas categorize and group the processes and sub-processes under them. The lines between the domain and functional areas are called “business lines”. All processes and services are going to be under one of those business lines as we go on decomposing. In the following steps, we decompose each functional area into processes and sub-processes.

3.2.1.3 Business Use Cases Integration

A use case is a technique used in software engineering to capture the functional requirements of a system. It describes the interaction between a primary actor, the initiator of the interaction, and the related function, represented as a sequence of

simple steps. Each use case is a complete series of events, from the point of view of the actor.

To further decompose the functional areas described above, “business use cases” will be used. Business use cases describe an interaction between external participants and the business functional requirements that produce value. In other words, business use case is a “sequence of actions that shows how the business responds to a business event, to yield a business benefit” [10]. Business use cases are effective when capturing these high-level functional requirements because of their simplicity, informal nature, simple and well known tool interfaces (often Word documents) [17]. Furthermore, business use cases permits a suitable end-to-end description of problems, and they have potential to remain stable while technologies come and go [16].

Business use case descriptions state “what” and not “how”. They determine who the source of the information is and who the target is. They do not imply a technical implementation of the flow. They do not explicitly define whether information is to be pushed or pulled, whether the information exchange is batch or online, or whether caching should be used. Such technical decisions relate mainly to non-functional requirements [11].

Hereafter, we will capture business use cases that specify requirements necessary to meet our goals and objectives. Business use cases will form a top-level view of the business as perceived by the outside world [10]. We must link every use case with a process that realizes it. "Realization" is a formal term in use case modeling [8].

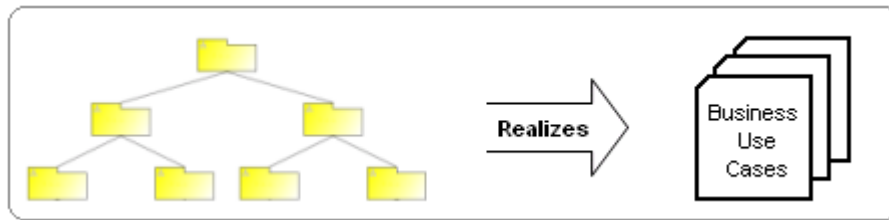


Figure 9 – Relationship between business use cases and business processes

For simplicity, the names of the use cases will be used throughout the thesis. However, use cases are generally saved in a document with all its other properties. Figure 10 shows a business use case in a word document.

Use Case 1: Get Paid for Car Accident

Primary Actor: Claimant
Scope: Insurance Company
Stakeholders and Interests:
 Claimant – to get paid the most possible
 Insurance company – to pay the smallest appropriate amount.
 Department of Insurance – to see that all guidelines are followed.
Preconditions: None.
Success Guarantee: Claimant and Insurance company agree on amount to be paid; claimant gets paid that.
Trigger: Claimant submits a claim.
Main Success Scenario:

1. Claimant submits claim with substantiating data.
2. Insurance company verifies claimant owns a valid policy.
3. Insurance company assigns agent to examine case.
4. Insurance company verifies all details are within policy guidelines.

Figure 10 – Business Use Case in a Word Document

We can now use the business use cases (high-level, coarse-grained use cases) to further decompose the domain. The business use cases identified are good candidates for business services (or functions) exposed on an enterprise component. The business use case definitions offer common and reusable business functionality.

With the business use cases, it is important to define the data to be input and output from each process at least at a high level. These definitions are then refined during component and service specification.

As we move into design, each functional area is mapped to one or more subsystems in the architecture. “Functional areas” is a business notion whereas the “subsystems” is a technology notion. There is a straightforward mapping between them, so we often find that at least one subsystem will then be identified and elaborated for each functional area. Similarly, business level use cases can be mapped to system level use cases in this stage.

In this way, each functional area or business process can be thought of as an IT subsystem that creates a business-driven boundary for the large-grained enterprise components that provide services.

In contrast, object-oriented analysis and design tend to produce object diagrams of tightly coupled finer-grained objects, which make component reuse very difficult [2]. With the SOA approach this is avoided by identifying the larger virtual structures (functional areas) first. Remaining elements are then refined through a more top-down approach (However, OO still is a valuable approach for design of the underlying class and component structure within a defined service).

3.2.2 Service Oriented Analysis

After completing domain decomposition, we have an idea of the functional areas in the business domain and how they interact. We have a view of the business domain: business processes and use cases within the business lines.

Now we move more into design-driven and architecture-driven decisions. Service oriented analysis, also accounts for subsystem analysis, refines the business use cases into system use cases that support a given business process. Subsystems are composed of business services and technical services.

During subsystem analysis, business and technical services are identified as follows:

- Analyze the process flow within the subsystem (often a sequence of use cases) to discover candidate business services.
- Use non-functional requirements to find technical services.
- Identify the required functionality (system level use cases) for each business service.

3.2.2.1 Identifying System Use Cases

Business process people write business use cases to describe the operations of their business. On the other hand, hardware or software development team writes system use cases for their requirements. The design team may write other system use cases to guide their design or to break down the requirements for small subsystems [4].

A business use case's design scope is business operations. It is about an actor outside the organization achieving a goal with respect to the organization. The business use case often contains nothing about technology, since it is concerned with how the business operates.

A system use case is one in which the design scope is the computer system to be designed. It is about an actor achieving a goal with the computer system; it is about technology.

The high-level business use cases discovered during domain decomposition are good candidates to be represented on the interfaces of these subsystems. These business use cases often collaborate to support a business process.

Each business use case relies on a set of system use cases encapsulated in the subsystem. The subsystem uses the business and technical services to realize the use case and support the exposed business service.

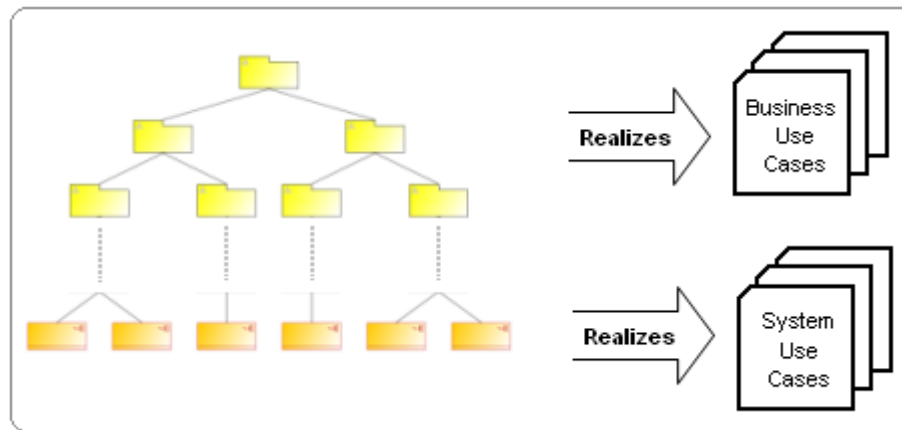


Figure 11 – Realization of Business Use Case and System Use Case

Hereafter, we will define business and technical services that realizes our system use cases. The decomposition tree will be grown with the new business and technical services and we will reach to the leaves of the decomposition tree. As SOSE methodology states; leaves of the decomposition tree are good candidates to be exposed as a web service.

3.2.2.2 Separating Technology from Business Services

The decomposition operation can facilitate the separation of technology from business context in tightly coupled architecture conditions. Thus, the services that

have business and technical functionality should be broken up into smaller and distinct parts to decrease integration challenges and ease future software updates. Decomposition operations should also be applied to separate different lines of business or partition different business functionalities [3].

Service ecosystem promotes an open environment in which web services are universally available across organizations using various technologies. Because systems are constantly creating and removing services, not only are the requirements constantly changing but also the system. This results in an even larger separation of business needs from system capabilities [9].

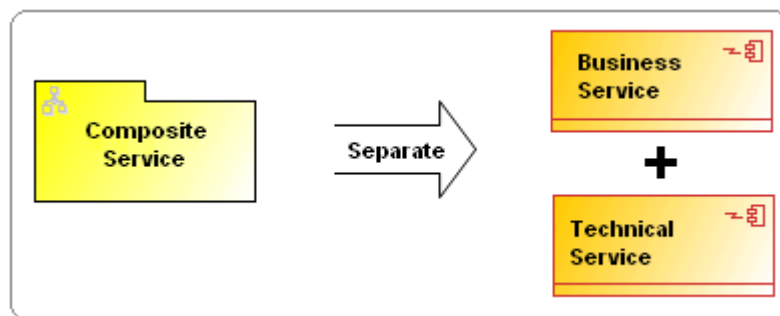


Figure 12 – Separating Business and Technology into Different Services

The need to separate technology from business especially comes from following reasons:

- Possible future software updates,
- The need to adapt to a new technology,
- Decreasing integration challenges.

One of the goals of the architecture style of SOA is to provide unprecedented flexibility in responsiveness to new business threats and opportunities [20]. System designers must be ready to use different technology even when using the same business service. Let's take a look at this technical problem:

“Internet banking users unable to reach their accounts or they are complainant of slow system when there is a huge increase of users every Monday morning.”

The reason for slow response time is the increase of transactions and locks that are conducted by DBMS. Using a better DBMS can handle the problem. If organization has separated DBMS related technical services from business services, then changing such technology can be done simply.

Another reason for separating technology is adapting to a new technology quickly. Every system needs refactoring after a maturity level. Especially these times that technology is progressing so fast. Below scenario is about adaptation to a new technology.

“A distributed system uses Microsoft Message Queue (MSMQ) for a data centric publish-subscribe model. However, technical team decides to use Data Distribution Service (DDS) technology instead of MSMQ.”

To achieve the adaptation to DDS, data distribution service must be changed. If data distribution service is separated before, switching MSMQ dependant service with a DDS dependant service would be enough to adapt to the new technology, and business service don't have to be changed.

When all these steps applied successfully, a decomposition tree that realizes all business needs would be constructed. Also, it would be ready for future updates and

to integrate to new systems. The next process is to compose the decomposition tree into an orchestration.

3.3 Similar Studies

SOSE is the ancestor of this thesis study. Also other works in SOA world affected this study. Admittedly, IBM is the flagship that supports SOA solutions for business. Thomas Behrens from IBM started studies about capturing business requirements using use cases. Those works triggered new ideas like using use case models as input to service identification. IBM's Rational RequisitePro is the tool that uses these studies. RequisitePro is used to describe the business goals and objectives and the business processes implemented to meet those objectives, and then to explain how to use the processes to identify relevant services necessary to fulfill the requirements that they represent.

RequisitePro's decomposition process is completely different from the UDSOA approach. UDSOA approach aims to construct a hierarchical decomposition tree with the exposed web services first. Then exposed business processes are modeled using BPEL. Instead, RequisitePro doesn't use hierarchical decomposition tree. It allows you to create business processes. For each business process; user can assign business use cases that are realized, business goals that are met and key performance indicators. Also, created business processes can be modeled directly even before all web services are identified. RequisitePro uses UML for all type of diagrams.

When we compare IBM's approach and UDSOA; it is observed that, UDSOA is simpler to understand and also to create a decomposition tree. Using business use cases and system use cases creates traceability between problem and solution domains when making decomposition more effective. RequisitePro uses key performance indicators instead of system use cases to identify non-functional requirements.

CHAPTER 4

ADAPTING SOSECASE TO UDSOA

4.1 Current Features of SOSECASE

Service Oriented Software Engineering Tool (SOSECASE) is a graphical modeling tool for SOA based system design and modeling and it supports the Service Oriented Software Engineering Modeling Language (SOSEML) notation. The tool provides easy-to-use and completely graphical modeling interfaces to the users for constructing system decomposition trees and creating exact Business Process Execution Language (BPEL) process models.

New SOSE models can be created, edited and saved using SOSECASE. Most of the graphical modeling concepts offered by different commercial tools such as Unified Modeling Language (UML) editors are included in the tool. Basic graphical modeling activities such as dragging and dropping graphical elements, editing features such as cut, copy, paste, delete and find operations are supported by SOSECASE.

SOSECASE uses the Eclipse's BPEL Designer plug-in for modeling BPEL processes graphically. This plug-in is completely an open source product and externally integrated to the main tool by using Eclipse's RCP (Rich Client Platform) architecture. The BPEL editor used in SOSECASE produces pure BPEL 2.0 codes (files with extensions .bpel and .wsdl which include the whole process model description). BPEL process models designed in SOSECASE can be used in anywhere else and by any other editor that supports BPEL 2.0 specifications.

For each SOSE model, a system decomposition tree can be constructed and each process in the tree can be modeled with BPEL by using SOSECASE graphical modeling tool. Figure 13 depicts the general view of the main window of the tool.

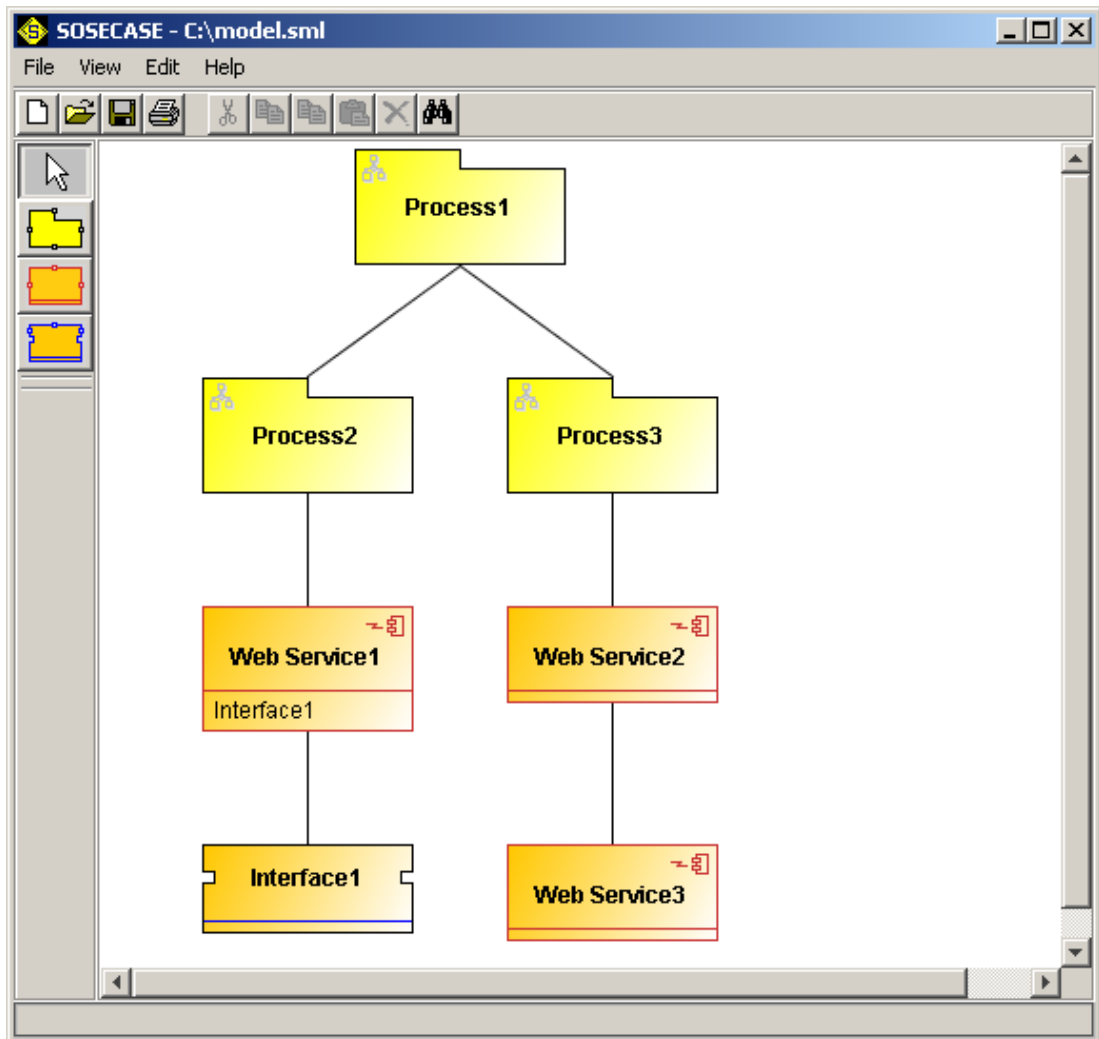


Figure 13 – General View of the SOSECASE Main Window

In the SOSECASE main window, model panel is the main region where the decomposition tree is drawn with SOSEML notation. SOSEML tool bar includes the

graphical modeling elements that can be used in the model; processes, web services and web service interfaces. A SOSE tree can have only one root node and this node must be always a process which represents the whole system. All graphical elements in SOSEML tool bar can be added to the model by dragging over the model panel. Properties of a model element can be displayed by double clicking on the graphical symbol of the node on model panel.

4.2 Additions to SOSECASE

4.2.1 Use Cases and SOA Design Mapping Window

Use Case Oriented Service Oriented Architecture (UDSOA) tries to connect the business requirements and SOA design. It uses business use cases and system use cases instead of business requirements because of their high level view. For adapting the approach to SOSECASE, business use cases and system use cases must be integrated to SOSECASE. However, SOSECASE shouldn't be complicated by the additions and the main window should keep the simple view.

The mapping window aims to construct a mapping between use cases and SOA design by providing user interface that shows every use case and their realization states. Furthermore, it shows business processes with the business use cases that they realize, and web services with the system use cases that they realize. Both business and technology related people can check that all use cases are realized by at least one business process or web service by using that window. It uses tree views to keep the relation with decomposition tree of SOSECASE. The notation that is used in tree views is developed especially to show that realization is completed or not.

To keep the simplicity of the SOSECASE, additions can be reached via Tools menu item. Main window of SOSECASE has no differences with the older version. Figure 14 shows how to reach to new window.



Figure 14 – Tools Menu Item to Reach New Window

Use case and SOA design mapping is done in one window. This window has two tree views. Tree view in the left side lists the business use cases and system use cases whereas tree view in the right side shows the decomposition tree with the realized use cases. Figure 15 shows the window before realization starts.

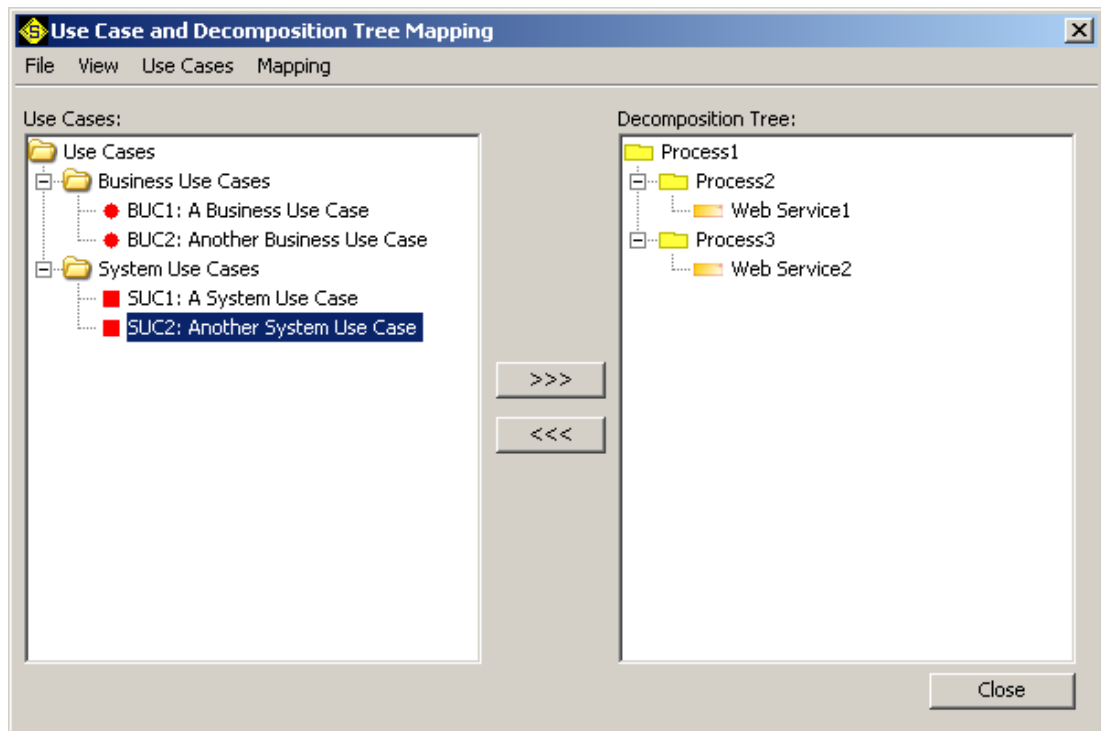


Figure 15 – Use Case and Decomposition Tree Mapping Window

4.2.1.1 Tree View Notation

There are two tree views in mapping window. The nodes in the tree views are supported with icons to simplify usage and show the state of the use case. Use cases are grouped by their use case type respectively; business use case group and system use case group. The use cases that are not realized are shown in red colored icon whereas realized use cases are shown in green colored icon. Business use cases are circle shaped and system use cases are in square shaped.

Decomposition tree view uses same notation with SOSEML. However, tree view's icons are smaller and inside of the box is not gradient colored. Web service interfaces are not included in the realization process. So there exists no notation for web service interfaces. Figure 16 shows the notation used in tree views.

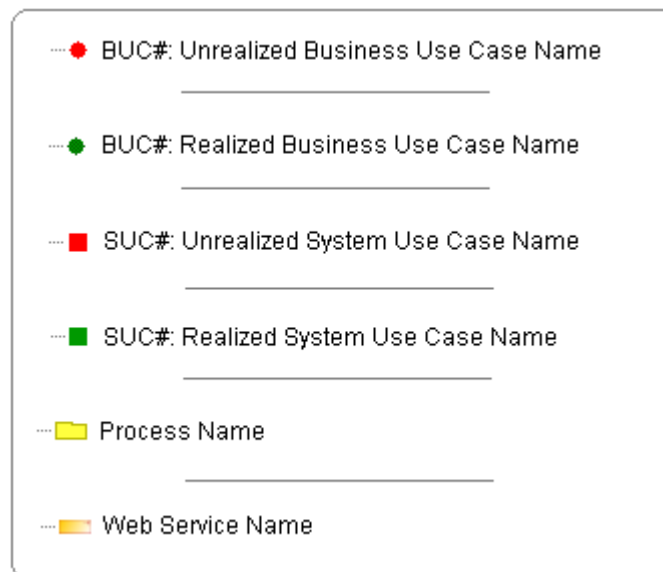


Figure 16 – Tree View Notation

4.2.1.2 Menu Operations

Figure 17 depicts the menu items of the mapping window.

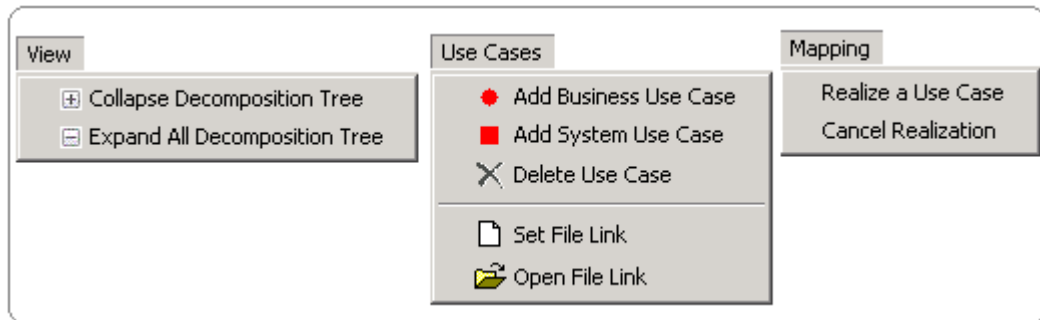


Figure 17 – Menu Items of Main Menu

When “mapping window” is opened, tree view in the left side will be filled with use cases and tree view in the right side will be filled with the decomposition tree that is constructed using the SOSECASE main window. Decomposition tree view can be collapsed or expanded using menu items in the “View” menu.

Business or system use case addition can be done using “Add Business Use Case” and “Add System Use Case” menu items. When adding a new use case, use case name will be asked. The identifier number of the use case is given by the system automatically. Identifier numbers are unique in business use cases and system use cases respectively. Selected use cases can be deleted using “Delete Use Case” menu item.

Use cases are saved generally in word documents. The linkage between the use cases in tree view and documents can be done using the menu items. “Set File Link” menu item gives possibility to choose a file and associate it with the selected use in the tree

view. Then, associated file can be opened with “Open File Link” menu item, after selecting a use case.

“Mapping” menu item includes realization operations. As described above in this thesis; business use cases are realized by business processes, whereas system use cases are realized by business and technical services. The notation doesn’t support the distinction between the business and technical services. So system use cases are realized by web services in this notation.

Mapping can be done by selecting a business use case and a business process, or selecting a system use case and a web service at the same. Realized use case will be added to decomposition tree, below the realizing process or service. Figure 18 shows the decomposition tree view after realization.

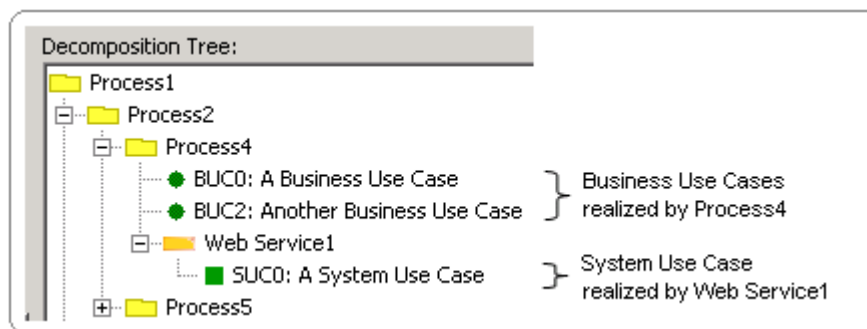


Figure 18 – Decomposition Tree View after Realization Operation

Business or system use cases can be realized by more than one process or web service. When a use case is realized for the first time, the color of the icon of it turns to green. The color of the icon doesn’t change after the second realization. Whenever all realizations are canceled, use case’s icon’s color will be red again. Use cases tree view as shown in Figure 19, can be used to check if all use cases are realized or not.

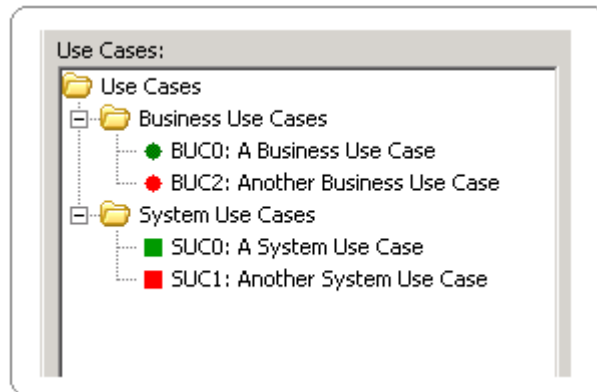


Figure 19 – Use Cases’ Realization States

For a successful decomposition, every use case that system must handle should be realized by at least one process or service. This window for SOSECASE can be used to capture unrealized use cases, and make the decomposition better as described above in this thesis.

CHAPTER 5

A CASE STUDY: MODELING A GIS APPLICATION

In this chapter, the steps of the UDSOA (Use Case Driven Service Oriented Architecture) approach that is explained in this thesis will be demonstrated with an example business scenario. The aim of this case study is not to develop a completed and running software. It is intended to show how a business scenario is decomposed from top to bottom with the integration of business and system use cases. Web services used in the study are not existing services.







Another goal of the case study is to detect the achievements or deficits of the approach. The requirements of the application are not provided since business and system use cases are used by the approach. The requirements of the application can be more involved, but only a simplified version of it is used to define business and system use cases in order to focus on the UDSOA approach instead of complicated application details.

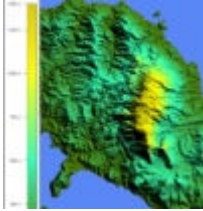


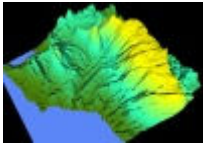
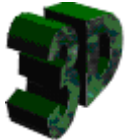
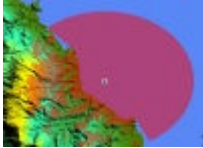

5.1 Business Scenario: GIS Application

In the following sub-sections, UDSOA approach is described through a Geographical Information System (GIS) application scenario. The GIS application opens different types of maps and shows them on the screen. It considers every map as layers and allows all layers to be rendered in a row. Layers can also be filtered. GIS application can do 2D and 3D analyses on maps as well. Applications need 3D rendering COTS API for displaying 3D terrain layers. GIS applications will have the same graphical user interface as commercial products such as Google Earth and NASA World Wind.

In Table 3, the terms that GIS application uses are listed and described.

Table 3 – Terms Used by GIS Application

	GIS Term Description
	<p>Zoom: Zoom is to show greater or lower detail for a portion of the map. User can zoom-in or zoom-out the map screen using either mouse or keyboard.</p>
	<p>Pan: Pan is to slide the map. User can pan map screen in four directions. Panning is done using either mouse or keyboard.</p>
	<p>Rotate map: User can rotate map screen so that, direction of the top of the map screen can be changed.</p>
	<p>Change projection: World's shape is not plane, so GIS application has to represent (project) the surface of the sphere to other shape on plane. Projection is done using a projection method like Mercator (Google Earth uses) or Geographic. User can change projection to change displaying of maps.</p>
	<p>Raster map: Raster (bitmap) maps are saved in a data structure representing rectangular grid of pixels. Raster maps displaying quality is lower as we zoom in to map. Satellite photos are in raster format.</p>
	<p>Vector map: Instead of saving maps in bitmap format, they can be saved in vector based collection of data (polygon, line, ellipse...). Vector maps' displaying quality is clear in every scale. The road maps or country borderlines of Google Earth are in vector format.</p>

GIS Term Description	
	Relief map: Relief maps represent 3D surface in detail by producing shadowing or other methods.
	Symbols: Symbols are used in maps to symbolize an object. Different symbols can be used for different types of objects. Symbols can be saved in different formats like bitmap or Scalable Vector Graphics (SVG).
	Scalable Vector Graphics (SVG): SVG is XML based file format for describing two-dimensional vector graphics. The SVG specification is an open standard.
	3D Terrain: Terrain is the third or vertical dimension of the land surface. Terrains can be rendered by using a 3D rendering API like DirectX or OpenGL.
2D	2D analyses: 2D analyses are the calculations that are done irrespective of vertical dimension. Calculating air distance is one of 2D analysis.
	3D analyses: Calculations or analyses that respective to vertical dimension are called 3D analyses. Querying elevation of a location is one of 3D analysis.
	View shed analysis: It is one of the 3D analyses that show the land which is visible to the human eye from a fixed point.
	Line of sight analysis: It is one of the 3D analyses that show visible points to the human eye along a line.

5.2 Description of the System

GIS applications are being used in many different areas today. They use plenty of technological components and services to handle the requirements about disk access, database access, caching and so on. A GIS application does not only have technological requirements but it also has complex algorithmic details. The need such as for running fast makes GIS application a complex system to implement, and for this thesis, a good case study for decomposing.

2D analyses on the surface of world are being done for many years. The algorithms used in this scope have been matured enough and finding a COTS product is not difficult. On the other hand, 3D analyses and 3D terrain operations have been maturing since the last decade. We owe this progress to the fast computer systems of the last decade. Because, 3D operations need a strong CPU and also strong graphics cards.

GIS application uses layering for displaying the maps. Layering separates the maps to layers, which brings a faster rendering when there is an update in a layer. A layer can be added to map or deleted from the map; also it can be filtered and updated. Every item on a map must be on a layer.

Symbology has become a strong factor when choosing a GIS system to buy today. That is because, every business that uses GIS have data to show on a map, and data can be shown in a standard symbology. The GIS application in this thesis uses the APP-6A symbology, which is the NATO standard for military map marking symbols. An example view of the map is depicted in Figure 20.

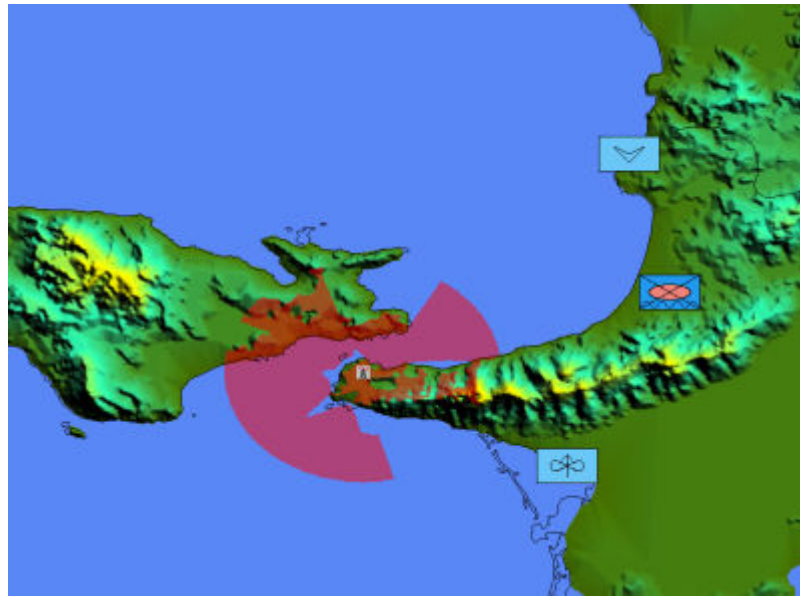


Figure 20 – A Sample Snapshot of a GIS Application

5.3 Decomposition of the System

As described in the previous chapters, we will try to decompose the system into processes, sub-processes and web services. Decomposition will take the business as input and try to output business and technical services descriptions ready to use. The output of UDSOA approach should be orchestrated at the end as SOSE methodology suggests.

Decomposition of a system involves two steps. First, we will try to define the business use cases, and try to decompose the system so that it realizes all business use cases. After that, service oriented analysis will create a complete decomposition tree with web services at the leaves.

5.3.1 Decomposing in Business Scope

5.3.1.1 Defining Functional Areas

As stated earlier, the decomposition starts with the decomposition of the domain. We will try to define the functional areas, which are the biggest groupings in decomposition tree. Defining functional areas is called functional decomposition. Figure 21 shows the domain decomposition of a GIS application scenario after the decomposition of the domain into its functional areas.

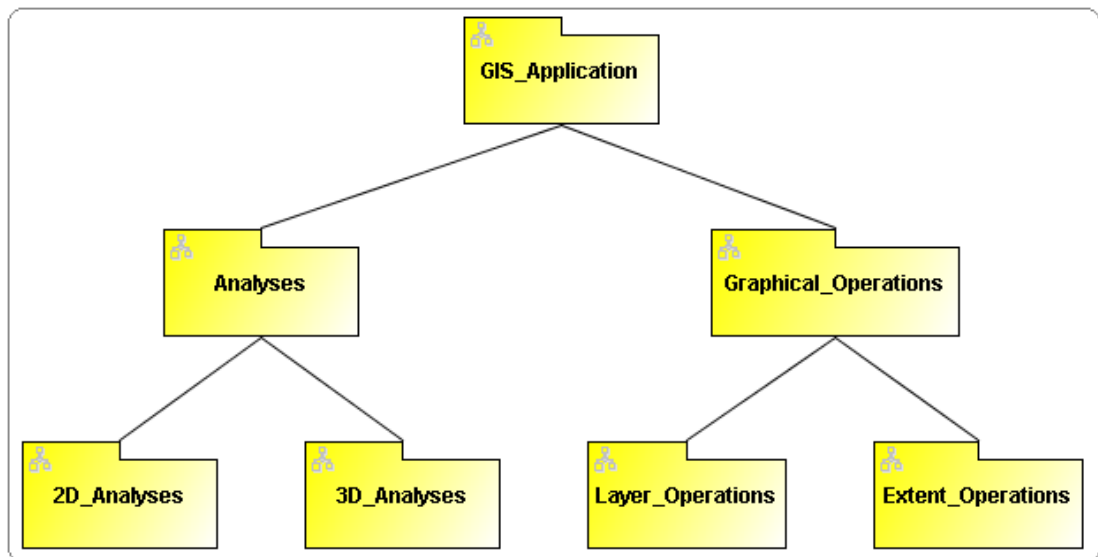


Figure 21 – Domain Decomposition of a GIS Application Scenario

The GIS application offers services in two main groups; “Analyses” and “Graphical operations”. So, those will be our biggest functional areas and they will be allocated after “GIS Application”. “Analyses” functional area is also decomposed into two other functional areas; “2D Analyses” and “3D Analyses”.

As the GIS domain is analyzed, it is obvious that “Graphical Operations” functional area can be decomposed into two business lines; “Extent Operations” and “Layer Operations”. “Extent Operations” covers all services that change the extent (scope) of the map screen. “Layer Operations”, on the other hand, involves all services that are in layer business line, like adding layer, deleting layer and filtering layer.

After decomposing GIS application domain into functional areas, we get four business lines. All processes and services are going to be under one of those business lines as we go on decomposing. In the following steps, we decompose each functional area into subsystems.

5.3.1.2 Defining Business Use Cases

To decompose the functional areas above more, “business use cases” will be used. Hereafter, we will capture business use cases that specify requirements necessary to meet our goals and objectives. Business use cases will form a top-level view of the business. Then, we will try to realize all business use cases. GIS application scenario has the following business use cases:

- BUC1: Zoom,
- BUC2: Pan,
- BUC3: Rotate map,
- BUC4: Change projection,
- BUC5: Show raster layer,
- BUC6: Show vector layer,
- BUC7: Show relief layer,
- BUC8: Show 3D terrain layer,
- BUC9: Show APP-6A symbols layer,
- BUC10: Calculate azimuth,
- BUC11: Calculate distance,
- BUC12: Convert azimuth,

- BUC13: Query elevation,
- BUC14: Line of sight analyze,
- BUC15: View shed analyze

If the use cases are analyzed, it can be noticed that they have no technological meaning as a business use case is supposed to have. They tell us WHAT the system should do. They have nothing about HOW the system should do that.

We can now use the business use cases (high-level, coarse-grained use cases like Query Elevation) to further decompose the domain. Figure 22 shows the further decomposed domain with business use cases.

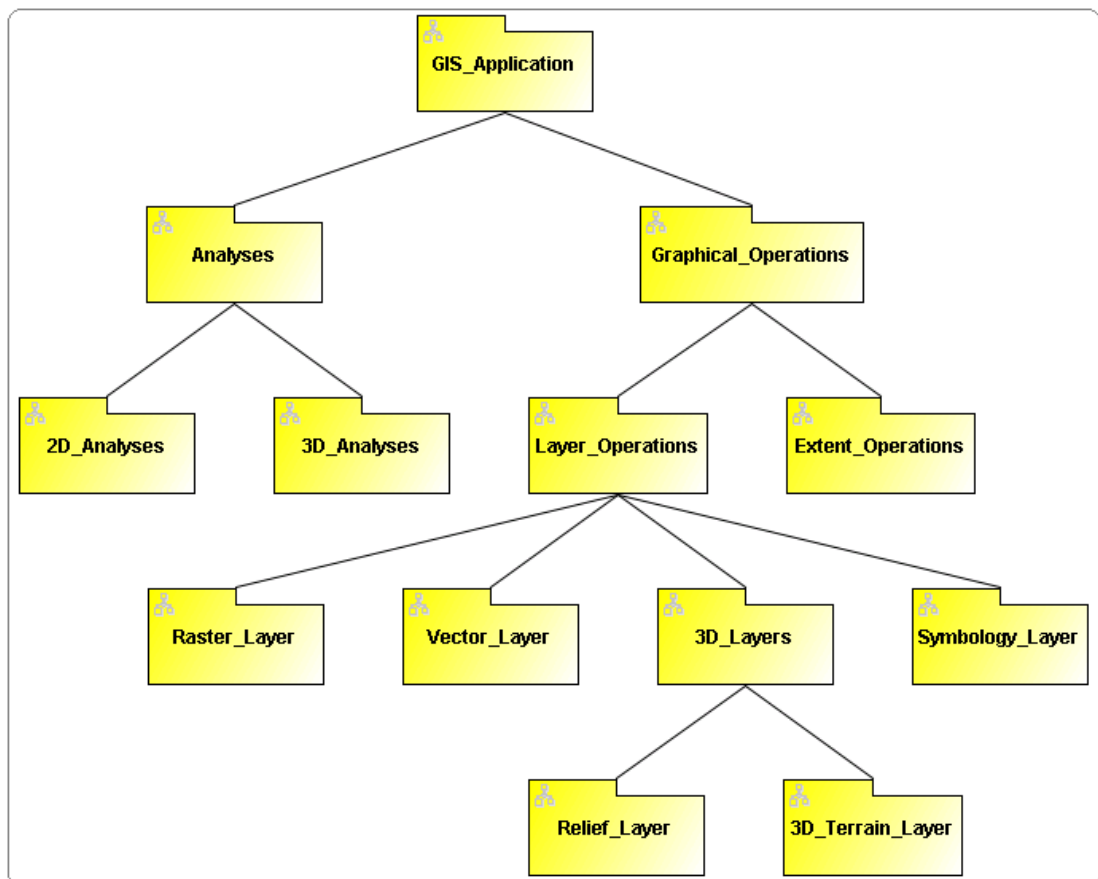


Figure 22 – Further Decomposed Domain with Business Use Cases

Furthermore, we can use SOSECASE to map business use cases and subsystems. Figure 23 shows that every business use case is realized by the subsystems defined.

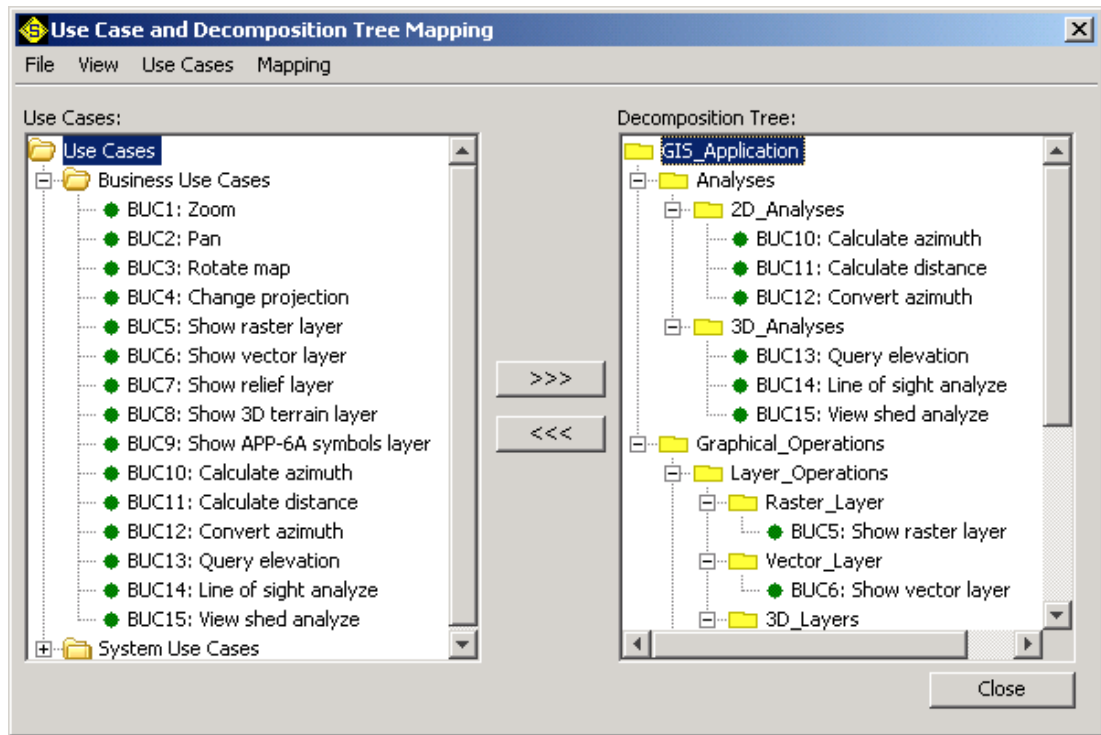


Figure 23 – Business Use Cases’ Realization in SOSECASE

5.3.2 Decomposition in System Scope

After completing domain decomposition, we have an idea of the functional areas in the business domain and how they interact. We have a view of the business domain: business processes and use cases within the business lines. Hereafter, we will define subsystems which are composed of business services (such as 2D analysis service, Extent operations service) and technical services.

The system use cases identified for GIS application scenario is as follows:

- SUC1: Zoom,
- SUC2: Pan,
- SUC3: Rotate map,
- SUC4: Change projection,
- SUC5: Show raster layer,
- SUC6: Show vector layer,
- SUC7: Show relief layer,
- SUC8: Show 3D terrain layer,
- SUC9: Show APP-6A symbols layer,
- SUC10: Calculate azimuth,
- SUC11: Calculate distance,
- SUC12: Convert azimuth,
- SUC13: Query elevation,
- SUC14: Line of sight analyze,
- SUC15: View shed analyze,
- *SUC16: Read DTED formatted elevation data files,*
- *SUC17: Use “Hill Shaded” relief’s,*
- *SUC18: Use double buffer rendering,*
- *SUC19: Use symbols in SVG file format.*

When we analyze system use cases identified for this scenario, it is observed that every business use case is being used in system use cases as well. Four new use cases (written in italic format) are added to the system use cases list which defines HOW the system should behave. SUC16 defines HOW the system should get elevation data. SUC17 defines HOW the system should draw relief layers. SUC18 defines HOW the system should render layers. SUC19 defines HOW the system should save symbols.

Those system uses cases will be used to define business and technical services at the leaf level of the decomposition tree. We have eight main subsystems identified; “2D analyses”, “3D analyses”, “extent operations”, “raster layer”, “vector layer”, “relief

layer”, “3D terrain layer” and “symbology layer”. Those subsystems are explained below.

5.3.2.1 2D Analyses Subsystem

“2D Analyses” subsystem is responsible for the 2D analyses and coordinate transformation that GIS application uses. Figure 24 shows subsystem with the two services it uses.

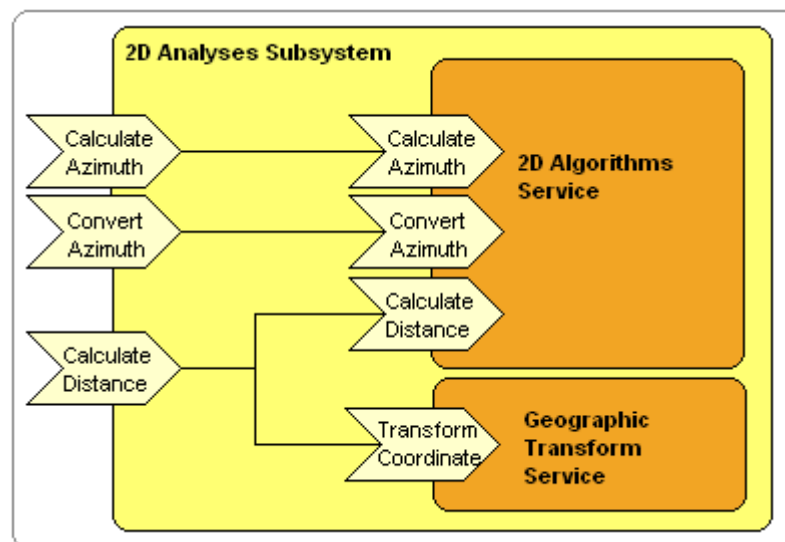


Figure 24 – 2D Analyses Subsystem

2D algorithms are business information, therefore “2D Algorithms“ service is a business service. Subsystem realizes “BUC10: Calculate Azimuth”, “BUC11: Calculate Distance” and “BUC12: Convert Azimuth” business use cases. The geographic transformation needed for calculating distance is provided by the “Geographic Transform” service.

5.3.2.2 3D Analyses Subsystem

“3D Analyses” subsystem is responsible for the 3D analyses that are done using elevation data. Figure 25 depicts the subsystem with the two services it uses.

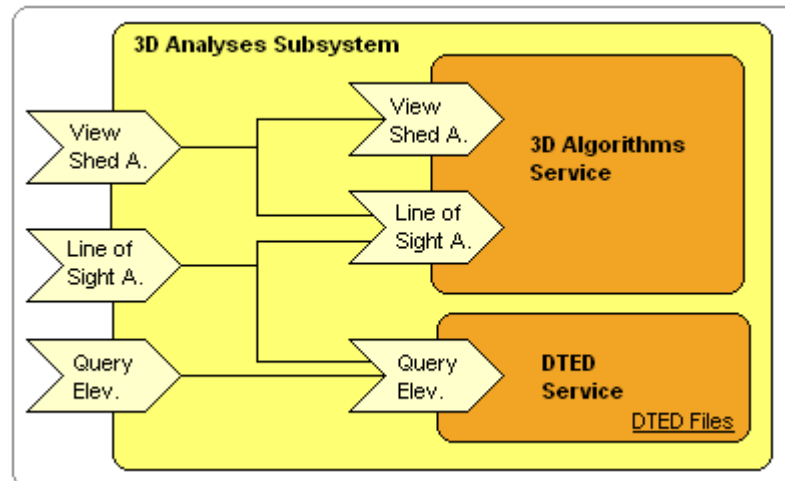


Figure 25 – 3D Analyses Subsystem

“3D Analyses” subsystem exposes three business use cases; “BUC13: query elevation”, “BUC14: line of sight analysis” and “BUC15: view shed analysis”. For algorithmic methods, “3D Algorithms Service” is added under the subsystem. To get the elevation data that is being used in algorithms, “DTED Service” is added. System use case 16 (SUC16) is realized by the “DTED Service”. “3D Algorithms Service” saves business information, so it is a business service, whereas the “DTED Service” is a technical service.

5.3.2.3 3D Terrain Layer Subsystem

One of the reasons for choosing GIS application scenario for the case study of this thesis is they particularly make use of technology too much; furthermore they have to

be changed to adapt new technologies in future. The clearest example to separation of technology services form business services in GIS application is in “3D Terrain” layer. Figure 26 shows 3D terrain service before analyzing.

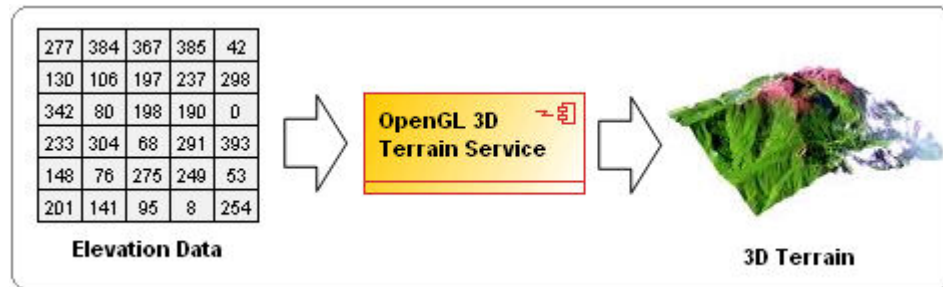


Figure 26 – 3D Terrain Rendering before Analyzing

As the Figure 26 depicts, “3D Terrain Service” takes elevation data in the matrix as input. Then this service creates a data structure which saves vectors for each cell. Assume that the GIS application uses OpenGL for 3D Graphics API. The filled data structure is then sent to OpenGL. OpenGL processes the data and renders a 3D view of the given terrain.

Even though everything seems fine, the service created above has both domain specific code and OpenGL specific code. In case of a change in 3D Graphics API (OpenGL), part of the service had to be coded again, which would force all the service to be tested again. Figure 27 shows the solution to this problem.

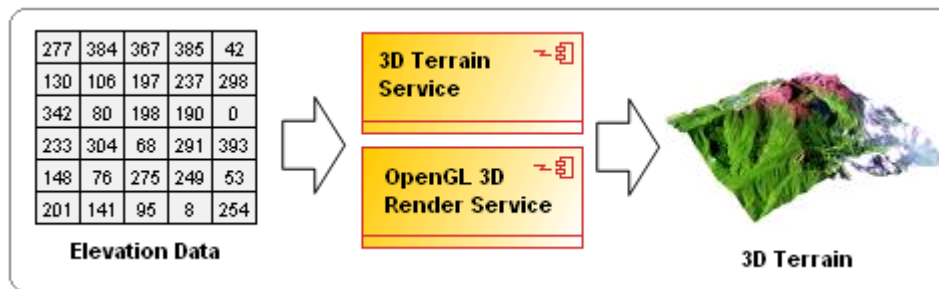


Figure 27 – 3D Terrain Rendering after Analyzing

In the solution above, we separate the technology specific part of the service (the OpenGL specific part). “3D Terrain” service just takes elevation data as input and outputs data structure that saves vectors. Then “OpenGL 3D Render” service takes the output of “3D Terrain” service and sends the data to OpenGL in appropriate format. “OpenGL 3D Render” service would probably be a small service (wrapper service). However, using a new 3D Graphics API in the future, let’s say DirectX, won’t bother us. All we need to have would be a wrapper service to DirectX.

5.3.2.4 Symbology Layer Subsystem

After decomposing system more with business use cases, we identified “symbology Layer” subsystem. Figure 28 depicts the “symbology layer” subsystem after decomposition.

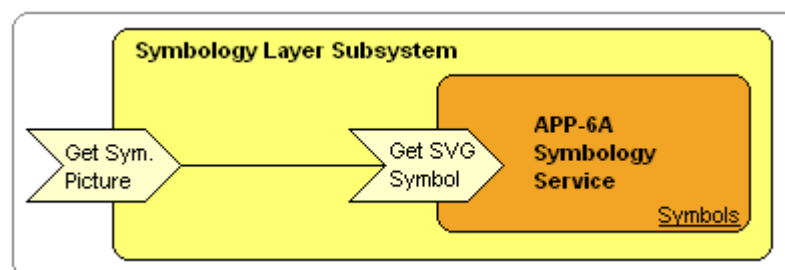


Figure 28 – Symbology Layer Subsystem before Separation of Technology

“Symbology layer” subsystem draws APP-6A symbols to appropriate locations on a picture. “APP-6A Symbology” service gives symbol drawing information in Scalable Vector Graphics (SVG) format. Subsystem renders (draws) this symbol by processing SVG format.

If we take a look at the subsystem again, we see that “APP-6A Symbology” service is a business service. Because it saves business information about APP-6A Symbology, which is a commonly used symbology in NATO applications. On the other hand, “Symbology layer” subsystem has the ability of rendering SVG files, which is a technology capability. We should separate technology part. Figure 29 shows “Symbology Layer” subsystem again, but this time it has two services.

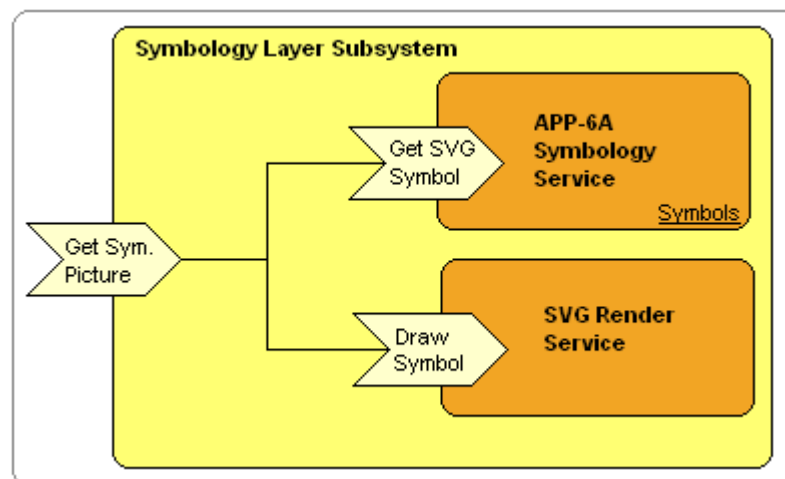


Figure 29 – Symbology Layer Subsystem after Separation of Technology

“SVG Render” service is added to subsystem. This service has no information about the business; instead it has SVG rendering capability which can be used in any other projects later. So we can say that “SVG Render” service is a technology service. If

this technology service is found inefficient in future, it can be replaced with another, which is the aim of separating.

5.3.2.5 3D Layers Subsystem

The layer operations that uses 3D data (elevation data) are grouped in “3D Layers” subsystem. Figure 30 depicts subsystem with the services in it.

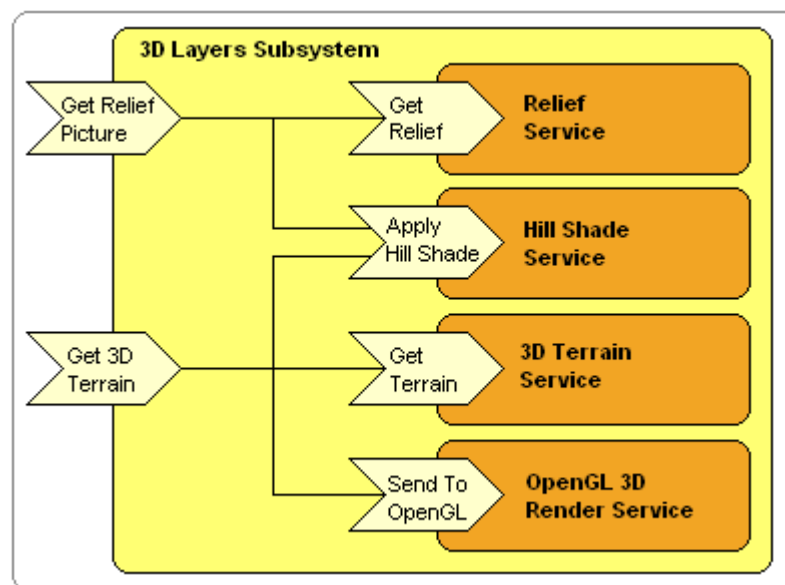


Figure 30 – 3D Layers Subsystem

“Relief Service” forms a relief picture for the given extent. However, this picture is not meaningful for human eye. So, hill shade algorithm is applied to that picture by “Hill Shade” service. “Hill Shade” service realizes “SUC2: Use Hill Shaded relief’s” system use case. So, it is a technical service. On the other hand, “3D Terrain” service creates data structure that saves vectors for elevation data as expressed before in this chapter. Then a hill shade cover is applied on it by “Hill Shade” service. At last, this data structure is sent to the “OpenGL 3D” service to send data to 3D graphics API.

5.3.2.6 Layer Operations Subsystem

“Layer Operations” subsystem is one level higher in the tree than subsystems described before. This situation usually defined as being coarser-grained. The subsystems or services in the lower levels are called as fine-grained which means that subsystems in this layer have more specific services. “Layer Operations” subsystem uses subsystems under it as if they are services. This is a rule in SOSE methodology. Figure 31 shows the “Layer Operations” subsystem.

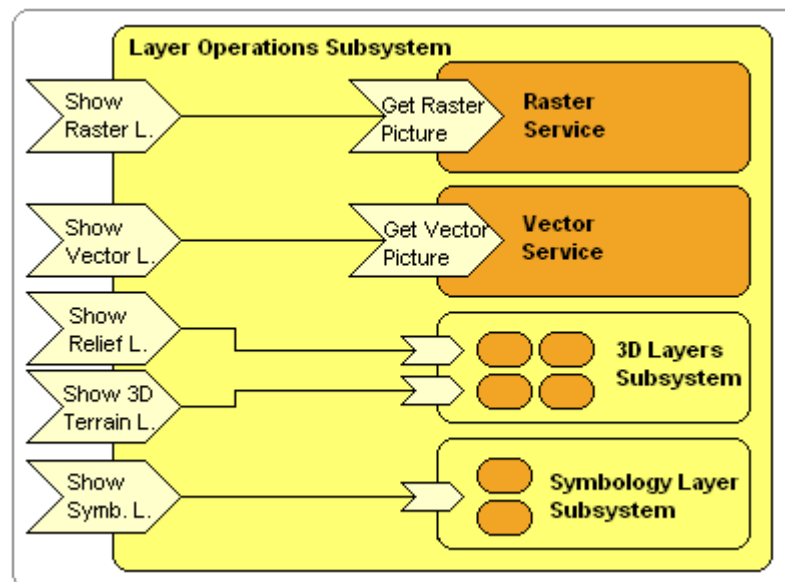


Figure 31 – Layer Operations Subsystem

The subsystem realizes “BUC5: Show Raster Layer”, “BUC6: Show Vector Layer”, “BUC7: Show Relief Layer”, “BUC8: Show 3D Terrain Layer” and “BUC9: Show APP-6A symbols layer” business use cases. For realizing, it uses two services and two subsystems. For the “Layer Operations” subsystem, there is no difference between the services it uses and the subsystems it uses.

Raster service reads TIFF files and merges them to get the whole picture. So that, it can give raster picture for a specified extent when its service is called. Vector service reads and caches vector data from file. When an extent is specified, it filters vector data that is out of extent and renders others on the layer picture. “3D Layers” and “Symbology Layer” subsystems also fill the picture with the information they have due to the given extent.

5.3.2.7 Extent Operations Subsystem

“Extent Operations” subsystem is responsible for preparing last picture that is shown to user. It saves the extent that is shown on screen. When user changes the extent by zooming, panning or rotating, it calculates the new extent and prepares new picture that is going to be shown on screen. “Extent Operations” subsystem uses double buffer rendering when rendering different layers which provides a faster rendering. The subsystem realizes “BUC1: Zoom”, “BUC2: Pan”, “BUC3: Rotate Map” and “BUC4: Change Projection” business use cases.

5.4 Case Study Results

At the end of the case study, it is observed that choosing GIS application as case study was right, since its technology dimension provided better examination of the approach.

After specifying functional areas, business use cases of the GIS application are identified. From this step on, decomposition steps progressed by use cases. Using use cases at the functional decomposition brought reusable business services for business ecosystems. The experiences of the author in the GIS domain show that, the exposed business processes can be used by other business partners in the business too.

System use cases were similar to business use cases in the case study. But in fact, they would be different if they were written in the long form (in word documents, they are written in long form). Normally, business use cases are written in business

language, whereas system use cases are written in a software language. However, only names of the use cases are written in this thesis. So they might seem similar even they are not.

Another observation is that, using system use cases at the decomposition in the system domain made the decomposition simple. The exposed web services are identified for each system use cases. Separating technology dimension from business web services need is observed during the case study and added as a SOA pattern to be applied at the end of the UDSOA approach.

A missing part that is observed during case study is that; decomposition should be continued to components level. However, SOSE methodology suggests decomposition until web services level. Identified components during decomposition can be reused even in our application's web services.

CHAPTER 6

CONCLUSION

6.1 Work Conducted

In this thesis study, a service oriented decomposition approach, Use Case Driven Service Oriented Architecture (UDSOA) is introduced for SOA based software development. The main purpose of the approach is to include business requirements into SOA analysis phase, so that resulting exposed services will have more value on service value-net.

The idea of the UDSOA approach is based on the Service Oriented Software Engineering (SOSE) modeling technique. SOSE brought too much to SOA based software development with functional decomposition of whole system and bottom-up orchestration with BPEL. The only purpose of the decomposition in the SOSE methodology was to scope the project down to a manageable and realistic set. However, the message in this thesis is, using the power of both business analysts and system analysts which will provide a better decomposition, resulting in the key services for the business.

The technique for using business requirements in decomposition is basically including business use cases and system use cases into decomposition process. Business use cases provide focusing on business at the domain decomposition phase. Business use cases are also used by system analysts and software architects to understand the way a software system fits into the organization and helps them to form system use cases. System use cases will bring the most suitable web services for the organization.

Including business analysts into the decomposition phase will bring value to the service ecosystem. Before SOA, applications were assumed to be built for one enterprise or one line of business. After SOA, applications tend to be used in a supply chain and are exposed to business partners who might compose, combine, and encapsulate them into new applications, which is called service ecosystem.

UDSOA is also supported with the additions to SOSECASE tool. SOSECASE tool become more powerful with the use cases and business processes mapping operation. The adaptation of the SOSECASE tool is waiting to be used in the validation of the approach.

To observe the possible benefits or difficulties of the approach in practice, a well defined real GIS application is decomposed into its web services in this thesis. The GIS application does not only have a business domain but also has technical dimensions which provide a better examination of the approach and brings more realistic results.

6.2 Comments

Various design paradigms are considered and different approaches are evaluated. Especially, SOSE methodology is evaluated carefully before this thesis study. It is observed that, SOSE methodology falls short when it comes to effectively decomposing. Taking both business and system domains into account makes decomposition more effective as this approach proposes. Finally, it is claimed that the approach in this thesis is theoretically acceptable and usable by both business analysts and system analysts according to the throughputs.

During the case study, it is observed that using business use cases at the decomposition phase brings more reusable business processes to the business ecosystem. Especially, the additions of “separation of business and technical services” to reusability are observed in the case study. Subsystem analyses make

resulting web services much more reusable and ready for future updates, which is the aim of all enterprises.

There exist similar approaches that use business goals or business use cases when decomposing complex systems. But UDSOA also uses system use cases to decompose in solution domain. This provides a better traceability from problem domain to solution domain, since business use cases rely on a set of business use cases.

Author's experiences in his organization show that, decomposition process always seems to be the responsibility of software engineers which caused failures most of the time. UDSOA suggests adding system engineers to the process, which can provide converging domain decompositions that were not easily finalized before.

It is unrealistic for an approach to claim that it is completely successful. The success of an approach comes when it brings another approach, just like SOSE is the motivation of this approach. All evaluations and case study done in the scope of this thesis study point out that this approach can take the benefits of adding business requirements to SOA design.

More experiments and tests on this approach are needed. The benefits of this approach will come in a long run. Because future brings refactoring, changing and adapting needs. Success of the SOA comes in future, as the business needs change.

6.3 Future Work

As future work, another key element of the SOA, *components* realizing services, can be included in the decomposition process. Three key elements of SOA are Services, Flows and Components. Addition of components to decomposition process would make decomposition tree deeper and make approach closer to realization of services.

Another future work is adding SOSEML the distinction of business web services and technical web services. This change in SOSEML will force user to separate technology from business context in web services.

REFERENCES

- [1] Eren Kocak Akbiyik, "Service Oriented System Design through Process Decomposition", August 2008.
- [2] Mark Endrei, Jenny Ang, Ali Arsanjani, Sook Chua, Philippe Comte, Pal Krogdahl, Min Luo, Tony Newling, "Patterns: Service-Oriented Architecture and Web Services", IBM e-book, pp. 79-106, 1st Ed., April 2004.
- [3] Michael Bell, "Service Oriented Modeling; Service analysis, Design, and architecture", John Wiley & Sons Ltd, pp. 140-143, 2008.
- [4] Alistair Cockburn, "Writing Effective Use Cases", Addison-Wesley Longman Publishing Co., 3rd printing, pp. 154-156, February 2001.
- [5] Olaf Zimmermann, Niklas Schlimm, Gunter Waller, Marc Pestel, "Analysis and Design Techniques for Service-Oriented Development and Integration", INFORMATIK 2005 – Informatik Live, September 2005.
- [6] Eren Kocak Akbiyik, Selma Suloglu, Cengiz Togay, Ali Hikmet Dogru, "Service Oriented System Design Through Process Decomposition", Integrated Design and Process Technology 2008, Taiwan, June 2008.
- [7] Ali Hikmet Dogru, Murat M. Tanik, "A Process Model for Component-Oriented Software Engineering", IEEE Software, IEEE Computer Society, Vol. 20, No. 2, pp. 34-41, April 2003.
- [8] Jim Amsden, "Modeling SOA: Part 1. Service identification", http://www.ibm.com/developerworks/rational/library/07/1002_amsden/, last accessed 22/12/2009.
- [9] M. Brian Blake, "Decomposing Composition: Service-Oriented Software Engineers," IEEE Software, Vol. 24, No. 6, pp. 68-77, Nov./Dec. 2007.

[10] Norbert Bieberstein, Robert G. Laird, Keith Jones, Tilak Mitra, “Executing SOA: A Practical Guide for the Service-Oriented Architect”, IBM Press, 1st Ed., pp. 111-112, May 2008.

[11] Thomas Behrens, “Capturing business requirements using use cases”, <http://www.ibm.com/developerworks/rational/library/dec04/behrens/>, last accessed 02/01/2010.

[12] Boris Lublinsky, “Defining SOA as an architectural style”, <http://www.ibm.com/developerworks/architecture/library/ar-soastyle/>, last accessed 22/12/2009.

[13] Olaf Zimmermann, Pal Krogdahl, Clive Gee, “Elements of Service-Oriented Analysis and Design”, <http://www.ibm.com/developerworks/webservices/library/ws-soad1/>, last accessed 26/12/2009.

[14] Mark Colan, “Service-Oriented Architecture expands the vision of Web services, Part 1”, “<http://www.ibm.com/developerworks/library/ws-soaintro.html>”, last accessed 15/01/2010.

[15] Olaf Zimmermann, Vadim Doubrovski, Jonas Grundler, Kerard Hogg, “Service-Oriented Architecture and Business Process Choreography in an Order Management Scenerio: Rationale, Concepts, Lessons Learned”, OOPSLA’05, Oct. 2005.

[16] Ian F. Alexander, Neil Maiden, “Scenarios, Stories, Use Cases through the Systems Development Life-Cycle”, John Wiley & Sons Ltd, pp. 471-472, 2004.

[17] Jim Amsden, “Business Services Modeling” http://www.ibm.com/developerworks/rational/library/05/1227_amsden/, last accessed 22/12/2009.

[18] David Sprott, Lawrence Wilkes, “Understanding Service-Oriented Architecture”, “<http://msdn.microsoft.com/en-us/library/aa480021.aspx>”, last accessed 15/01/2010.

[19] David Booth, Hugo Haas, Francis McCabe, Eric Newcomer, Michael Champion, Chris Ferris, David Orchard, "Web Services Architecture", "<http://www.w3.org/TR/ws-arch/>", last accessed 15/01/2010.

[20] Ali Arsanjani, "Service-oriented modeling and architecture", <http://www.ibm.com/developerworks/library/ws-soa-design1/>, last accessed 22/12/2009.

[21] Chris Peltz, "Web Services Orchestration and Choreography," *Computer*, vol. 36, no. 10, pp. 46-52, Oct. 2003.