GENERAL FRAMEWORK FOR SERVICE ENGINEERING ANALYSIS AND DESIGN

By
Purnomo Yustianto, (ST. MT.)

Submitted in fulfilment of the requirements for the degree of
Doctor of Philosophy

Deakin University
June, 2018

This thesis may be made available for consultation, loan and limited copying in accordance with the Copyright Act 1968.

I certify that I am the student named below and that the information provided in the form is correct.

Full Name: Purnomo Yustianto
Signed: [Signature Redacted by Library]
Date: 28 June 2018
I certify the following about the thesis entitled *General Framework for Service Engineering Analysis and Design* submitted for the degree of Doctor of Philosophy

a. I am the creator of all or part of the whole work(s) (including content and layout) and that where reference is made to the work of others, due acknowledgment is given.

b. The work(s) are not in any way a violation or infringement of any copyright, trademark, patent, or other rights whatsoever of any person.

c. That if the work(s) have been commissioned, sponsored or supported by any organization, I have fulfilled all of the obligations required by such contract or agreement.

d. That any material in the thesis which has been accepted for a degree or diploma by any university or institution is identified in the text.

e. All research integrity requirements have been complied with.

_I certify that I am the student named below and that the information provided in the form is correct._

Full Name: Purnomo Yustianto
Signed: Signature Redacted by Library

Date: 28 June 2018

Deakin University CRICOS Provider Code 00113B
Acknowledgement

On the surface, this thesis may appear as the reporting of an academic journey. But on a personal level, it is a map of an accomplished quest. As, any quest, the journey leads to several side-quests in collecting pertinent artefacts along the way. It has been a journey of enjoyment, fulfilling curiosities and ultimately leading to a self-discovery.

As a closure for this part of the journey, a humble gratitude is offered to a number of people taking part in the journey:

- Prof Robin Doss as the main supervisor of this research, with his patience in allowing a curiosity to turn into this research work.

- Dr Suhardi, as my local supervisor, with his guidance in initiating the research and facilitation in completing this work.

- All of Deakin University team: Prof Bas Baskaran, Prof John Yearwood, Prof Yong Xiang, Associate Professor Jo Coldwell-Neilson, Lina Darliana, Bec Anderson-Peters, Nghia Dang, and others, with their warm hospitality and support in making Deakin University my research almamater.

- My research colleagues, Novianto Budi K, Deni Prayitno, Haris Santika, Silvia, Agustinus Andriyanto, and all members of Service Computing Research Group of Institut Teknologi Bandung, which provide support, discussion and material for this research.
• The entire supporting group on Doctoral Endorsement Program of West Java Provincial Government: Dedi Mulyadi, Candrawulan, Nunik Mulyana, and the West Java Communication and Informatics Agency, which facilitate both my research and leave of absence while conducting the research.

• And ultimately to my beloved family: Mother, sisters, wife and kids, for their blessing for me to endure this strange quest, and most importantly, for their patience on the amount of my time given to it.

• All other parties whose name is left unmentioned here, which intentionally or not provided partial contributions to the overall completion of this work.

The academic contributions of this research are mainly focused, but hopefully a real dent is curved on the service science research landscape. And this dent is dedicated to fellow researchers.

Purnomo Yustianto
Bandung, Indonesia
28 June 2018
Publication List


5. Silvia, Suhardi, and Yustianto, P., 2016, October. *Business process improvement of district government innovation service: Case study Cimahi Tengah District of Cimahi*. In Information
Technology Systems and Innovation (ICITSI), 2016 International Conference on (pp. 1-6). IEEE.


based on SoaML (Submitted to Journal of Systems Science and Systems Engineering).

Abstract

The term ‘service engineering’ is generally understood as a process of devising a service system. But beyond that understanding, the term is still regarded as an ‘open’ research challenge due to unspecified details and conflicting perspectives. Within the multi-discipline nature of service science, two streams contribute to the ‘service engineering’ concept: the classic business-side and the emerging informatics side. This work consolidates these two overlapping perspectives.

This research produced a service engineering framework labelled as the General Service Engineering Framework (GSEF) which covers service aspects from the business side through to the informatics side, i.e. Business Capability, Business Model, Service Value, Interaction Model, Process Model and, Software-Service Model.

As a part of the framework, ontological basis of service engineering is defined. The ontology collects and specifies components pertinent within the context of service engineering, along with the relationship between the concepts. The software-service ontology was drawn from the informatics domain, while the generalized ontology of a service system was built from both a business management and the information system perspective.

The framework itself is presented in a structure of three component layers: Activity, Artefact, and Modelling. The activity characterizes the phases and steps of analysis and design process. The artefact defines the
product of each step while also implying the dependency and flow of the produced artefact. The modelling layer is an open container which proposes a format to present the artefact, based on a specific modelling language.

The proposed framework was empirically evaluated with a series of case studies and theoretically validated with an ontology evaluation. In its final version, the framework specifies four main activities with sixteen types of artefacts to produce within the context of service analysis and design.

As an additional contribution, the research also provides a structured landscape of a metamodel from both industrial and academic perspectives. The landscape is a product of exploring several potential metamodels for representing an artefact. The exploration enriches the suggested artefact format from the original eighteen formats to thirty alternatives of a metamodel.
# Table of Content

Acknowledgement ........................................................................................................ iv  
Publication List .............................................................................................................. vi  
Abstract ......................................................................................................................... ix  

## Chapter 1 Introduction .............................................................................................. 1  
1.1 Motivation .................................................................................................................. 1  
1.2 Objective and Scope .................................................................................................. 3  
1.3 Research Problem ...................................................................................................... 6  
1.4 Research Activities .................................................................................................. 7  
1.5 Contributions ............................................................................................................ 11  
1.6 Chapter Overview ..................................................................................................... 13  

## Chapter 2 Literature Review ...................................................................................... 16  
2.1 Defining a ‘Service’ .................................................................................................... 16  
2.2 Classic Service Engineering ..................................................................................... 19  
2.3 Service Oriented Architecture .................................................................................. 25  
   2.3.1 Service Oriented Methodology .......................................................................... 28  
   2.3.2 Modelling in Service Oriented Analysis ............................................................ 32  
2.4 Informatics Service Engineering ............................................................................. 34  
   2.4.1 Service Engineering and Software Engineering ............................................... 35  
   2.4.2 Service Engineering and SOA .......................................................................... 36  
2.5 Research Opportunities in Service Engineering ..................................................... 41  

## Chapter 3 Service Engineering Ontology .................................................................. 45  
3.1 Ontology Defined ....................................................................................................... 45  
3.2 Related Work ............................................................................................................ 48  
3.3 Software Service Ontology ...................................................................................... 52  
   3.3.1 Ontology of SoaML ........................................................................................... 53  
   3.3.2 ISO/IEC SOA Ontology .................................................................................... 60  
3.4 General Service Ontology ....................................................................................... 63  
   3.4.1 Ontology Source Material ................................................................................ 63
5.3.4 Interaction Modelling .................................................. 165

Chapter 6 Conclusion ............................................................ 172
6.1 Summary ......................................................................... 172
6.3 Limitations and Directions for Future Work ...................... 175

Bibliography ......................................................................... 177
List of Tables

Table 1.1: Dictionary Definition of ‘Framework’ ........................................... 3
Table 1.2: Selected Usage of ‘Framework’ Terminology ...................... 4
Table 2.1: An Incomplete List of Service Oriented Methodology .......... 29
Table 2.2: Stages of SOMA Methodology ................................................. 30
Table 2.3: Stages of MSOAM Methodology ............................................... 31
Table 2.4: Service Oriented Modelling Language ................................. 33
Table 2.5: Service Engineering Framework with SOA .................... 37
Table 3.1: SOA Characteristic Identifiers ................................................... 51
Table 3.2: SoaML Conceptual Components ............................................. 54
Table 3.3: ISO/IEC 18384:2016 SOA Conceptual Components ............. 61
Table 3.4: Identified Standard Documents for Concepts Definition ...... 64
Table 3.5: Entity-related Concept Definitions ............................................ 65
Table 3.6: Process-related Concept Definitions ......................................... 65
Table 3.7: Activity (sub-process)-related Concept Definitions ............ 66
Table 3.8: Task (atomic-activity)-related Concept Definitions .......... 66
Table 3.9: Collaboration-related Concept Definitions ......................... 66
Table 3.10: Choreography-related Concept Definitions .................... 66
Table 3.11: Interaction-point-related Concept Definitions ................. 67
Table 3.12: Interface-related Concept Definitions ............................... 67
Table 3.13: Operation-related Concept Definitions .............................. 67
Table 3.14: Operation-related Concept Definitions .............................. 67
Table 3.15: Service-related Concept Definitions ................................. 67
Table 3.16: Capability-related Concept Definitions ............................. 68
Table 3.17: Business Model-related Concept Definitions .................. 68
Table 3.18: Rule-related Concept Definitions ......................................... 68
Table 3.19: Contract-related Concept Definitions ............................... 68
Table 3.20: Value-related Concept Definitions ....................................... 68
Table 3.21: Role-related Concept Definitions ......................................... 68
Table 3.22: Hierarchy of 17 Merged Concept Definitions
Table 3.23: Typology of the Service Encounter with examples
Table 3.24: Concepts for Software Service Context
Table 4.1: Service Engineering Framework (prototype)
Table 4.2: Key Performance Indicator Analysis (Case Study 1)
Table 4.3: Service Engineering Modelling Aspects (from Case Study)
Table 4.4: Sample of Business Goals (Smart Campus)
Table 4.5: Sample of Service Goals (Smart Campus)
Table 4.6: Comparison of SoaML and UML for Service Modelling
Table 5.1: Metamodels Set (Coded and Sorted)
Table 6.1: Research Results
List of Figures

Figure 1.1: Design Science Research Cycles .................................................. 7
Figure 1.2: Research Flow ............................................................................. 9
Figure 1.3: Research Content Mapping .......................................................... 14
Figure 2.1: Service Input-Output Model ......................................................... 17
Figure 2.2: Areas of Service Engineering ....................................................... 21
Figure 2.3: Stages of Service Engineering ..................................................... 21
Figure 2.4: Synthesized NSD Framework ...................................................... 23
Figure 2.5: SSDP Frameworks ...................................................................... 25
Figure 2.6: Elements of Service Oriented Analysis and Design ................. 26
Figure 2.7: Structure of Service Oriented Methodology ............................... 28
Figure 2.8: Comparison of SOMA and MSOAM concepts coverage ........... 32
Figure 2.9: Scientific Publication of ‘Service’ Modelling [65] ................. 34
Figure 2.10: Business-IT Alignment Model Framework ......................... 38
Figure 2.11: Derivation of Business Service ................................................ 39
Figure 2.12: Derivation of Software Service ............................................... 40
Figure 2.13: Context Positioning of Research Scope ................................. 43
Figure 3.1: Model, Metamodel and Ontologies ............................................. 46
Figure 3.2: Research Flow and Content Map (Chapter 3) ......................... 47
Figure 3.3: Metamodel of Service System .................................................... 48
Figure 3.4: Metamodel of Service Concept .................................................. 49
Figure 3.5: Service Package of SOA Metamodel ......................................... 50
Figure 3.6: Succession of SOA Open Standard ........................................... 53
Figure 3.7: Interface-based Service version of SoaML Ontology ............. 56
Figure 3.8: Contract-based Service version of SoaML Ontology .............. 57
Figure 3.9: Service Interface-based Service version of SoaML Ontology ... 58
Figure 3.10: SoaML Ontological Structure ................................................ 60
Figure 3.11: ISO/IEC 18384:2016 SOA Ontological Structure ............... 62
Figure 3.12: General Service Engineering Ontology ................................. 69
Figure 3.13: Typology of Customer Contact [4] ............................................72
Figure 3.14: Consolidated Software Service Ontology .................................74
Figure 3.15: Model of a Simple Service System ............................................77
Figure 3.16: Model of a B2B service pattern................................................78
Figure 3.17: Model of a multi-provider service pattern.................................79
Figure 4.1: Research Flow and Content Map (Chapter 4).............................81
Figure 4.2: Integrated Methodology for Service-Oriented System ...............82
Figure 4.3: Service Engineering Framework (1st iteration) ..........................88
Figure 4.4: BMC context in Service Engineering .........................................90
Figure 4.5: Service Blueprinting Template..................................................91
Figure 4.6: BPMN of the ‘to-be’ business process (Case Study 1) ...............95
Figure 4.7: Integrated Service Architecture (Case Study 1) ..........................96
Figure 4.8: Adapted Service Engineering Framework (Case Study 1) .........96
Figure 4.9: BPMN of the ‘to-be’ Business Process (Case Study 2) .............98
Figure 4.10: Component Dependency Diagram (Case Study 2) .................98
Figure 4.11: Adapted Service Engineering Framework (Case Study 2) .......99
Figure 4.12: Component Business Model (Case Study 3) ...........................100
Figure 4.13: E-R Diagram (Case Study 3) ..................................................101
Figure 4.14: Responsibility Assignment Matrix (Case Study 3) ...............102
Figure 4.15: Software Service Architecture (Case Study 3) ......................103
Figure 4.16: Sequence Diagram in Case Study 3 .......................................104
Figure 4.17: Use Case Diagram in Case Study 3 .......................................104
Figure 4.18: Adapted Service Engineering Framework (Case Study 3) .......104
Figure 4.19: Adapted Service Engineering Framework (Case Studies) .......108
Figure 4.20: Service Engineering Framework (2nd iteration) .....................109
Figure 4.21: SoaML Top-Down Approaches Artefacts Flow ......................111
Figure 4.22: Regrouping SoaML Artefacts into Activities .......................112
Figure 4.23: Incorporating SoaML Artefacts into Framework .................113
Figure 4.24: Service Engineering Framework (3rd iteration) ....................113
Figure 4.25: Package Diagram of Project Domains (Smart Campus) .........114
Figure 4.26: Catalogue of Software-Services (Smart Campus) ..........115
Figure 4.27: SoaML Capability Diagram (Smart Campus) ..............117
Figure 4.28: SoaML Service Interface Diagram (Smart Campus) .......118
Figure 4.29: SoaML’s Participant Diagram (Smart Campus) ..........119
Figure 4.30: SoaML’s Service Architecture (Smart Campus) ..........119
Figure 5.1: Research Flow and Content Map (Chapter 5) ...............123
Figure 5.2: Analysis-Synthesis Bridge Model ................................125
Figure 5.3: Metamodel components ............................................128
Figure 5.4: Metamodel Relationship (arranged in similarity) ...........135
Figure 5.5: Metamodel Grouping ..................................................137
Figure 5.6: Service Engineering Framework Structure (3rd iteration) ..149
Figure 5.7: Artefact Ontological Position in (Activity 1.1) .......... 150
Figure 5.8: Artefact Ontological Position in (Activity 1.2) .......... 151
Figure 5.9: Artefact Ontological Position in (Activity 2.1) .......... 153
Figure 5.10: Artefact Ontological Position in (Activity 2.2) .......... 155
Figure 5.11: Sample of Business Motivation Model Artefact .......... 158
Figure 5.12: Sample of VDML Capability Library ...........................160
Figure 5.13: Sample VDML Capability Management ......................161
Figure 5.14: Sample of Service Dominant Business Model (SDBM) ....162
Figure 5.15: BM Cube components mapping with BMC .................163
Figure 5.16: Structure of Service BMC (S-BMC) .........................164
Figure 5.17: Sample of S-BMC Artefact with five party layers ..........164
Figure 5.18: Sample of Process Chain Network (PCN) Artefact ..........165
Figure 5.19: Network of Interactions in PCN .................................166
Figure 5.20: Diagrams in ServiceML .............................................167
Figure 5.21: Sample of Service Journey Modelling Language (SJML) ..168
Figure 5.22: SJML abstraction for multi-participant interaction ...........169
Figure 5.23: Metamodels Options for Service Engineering Framework 170
Figure 6.1: Map of Research Artefacts .........................................173
Chapter 1

Introduction

1.1 Motivation

The global business landscape trend in shifting its primary focus from products as an output to the service provision process is academically recognized as a shift from *Goods-dominant logic* (GDL) to *Services-dominant logic* (SDL). While the classic GDL only recognizes ‘goods’ as a basis of an exchange, SDL views ‘service’ as the fundamental basis of all economic exchange [1].

SDL also identifies an abstract concept of ‘value’, co-created and exchanged between actors during a business transaction. The recognition of value co-creation delivers the concept of a ‘service ecosystem’, in which multiple actors or business entities are enabled to form a coordinated arrangement for performing certain services [2].

The dynamic of service ecosystems has stimulated a rapid pace of change in globalized business environments in which service innovation becomes a critical priority for business entities. Service providers strive to provide the best value for customers by optimizing their own service capability, while at the same time also assessing and forming an alliance with business partners in order to survive in a competitive service business landscape.
Information Technology (IT) has become an important component in reshaping services and customer experience in the service industry. It serves as an important component during customer contact and service delivery [3]. IT involvement in a service ranges from a simple technology-assisted contact to an automated self-service contact [4]. Within this range, the component can assist in a customer contact, as a user interface channel itself, or as an automated background process in delivering the service. In these forms, the component takes a central role for service innovations. New and improved business services deployed with the help of electronic services are often labelled as an e-service [5].

Considering the dynamic nature of the service ecosystem, the multitude of service offering concepts, and the pervasive role of IT, the process of designing a service is not always an easy task. To handle such an effort, a systematic and structured approach on the service design process will be helpful. The approach should be able to decompose the process into smaller manageable parts, while at the same time ensuring the coverage of multiple aspects of strategic business concepts through the technical implementation of the IT layer [6].

As the process of service analysis and design becomes a multidiscipline effort, the elaboration encompasses the domain of business strategy, business process, IT architecture and software engineering. Traditionally, each domain discipline had its own perspective on concept sets and structure. Competing perspectives and structures may also exist within a specific domain. A similar situation
persists in the topic of service engineering. This is the main challenge in defining a generalized service engineering framework [7][8].

So far, the conceptual foundation for ‘service engineering’ from industrial or academic works were produced with partial perspective. Two dominantly competing perspectives are from business-marketing perspective, and information technology perspective. A unifying perspective of ‘service engineering’ concept has not been produced. This is the research gap addressed by this research work by building a comprehensive and cohesive foundation for service engineering concept.

1.2 Objective and Scope

The objective of this research is to develop a generalized approach to the process of service analysis and design, with regard to the multidiscipline nature of the subject matter. To be specific, this generalized approach is labelled as the General Service Engineering Framework (GSEF).

Table 1.1: Dictionary Definition of ‘Framework’

<table>
<thead>
<tr>
<th>Meriam Webster Online Dictionary</th>
<th>Cambridge Online Dictionary</th>
</tr>
</thead>
<tbody>
<tr>
<td>• a basic conceptional structure (as of ideas)</td>
<td>• a supporting structure around which something can be built</td>
</tr>
<tr>
<td>• a skeletal, openwork, or structural frame</td>
<td>• a system of rules, ideas, or beliefs that is used to plan or decide something:</td>
</tr>
</tbody>
</table>

The term ‘framework’ is used in this research to moderate the expected result within the context of ‘Service Engineering’. The intended
usage of ‘framework’ stems from dictionary definitions (Table 1.1) which imply the conceptual nature of the work with an open-ended feature of the how-to aspect.

Various academic contributors have used the term ‘framework’ as an umbrella construct to define a way to produce something in various level of detail, commonly accompanied by a set of base understanding. Table 1.2 collects a sample of ‘framework’ terminology usage in academic articles. In general, a framework can be summarized as a set of usable components to pursue a certain purpose.

Table 1.2: Selected Usage of ‘Framework’ Terminology

<table>
<thead>
<tr>
<th>‘Framework’ Definition</th>
<th>Context</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tools for analysing and developing</td>
<td>Business modelling [9]</td>
</tr>
<tr>
<td>Grouped stages with process and artefacts flow</td>
<td>Software engineering [10]</td>
</tr>
<tr>
<td>Conceptual model for developing and describing</td>
<td>Web service modelling [11]</td>
</tr>
<tr>
<td>A set of linked processes as states of the development</td>
<td>Design process [12]</td>
</tr>
<tr>
<td>Processes guidance with phase definition, tools and theories</td>
<td>Service Engineering [13]</td>
</tr>
<tr>
<td>Process guidance with common understanding, methods and tools.</td>
<td>Requirement engineering [14]</td>
</tr>
<tr>
<td>Design and development tools equipped with building block and method</td>
<td>Enterprise architecture [15]</td>
</tr>
</tbody>
</table>

Framework departs from the ‘what’ aspect of conceptual understanding towards the ‘how’ aspect in achieving the purpose. A framework describes the stages of process flow but occasionally omit the detailed step normally prescribed in a method.
This research supports the definition of framework as an abstract form of a method [11][16]. A framework could also be seen as a basis for building a fully prescriptive method by offering a structured system to realize a targeted objective [17]. Compared to the more rigid and detailed prescription in a method, a framework allows flexibility in its implementation, with non-detailed prescription which accommodates variances in its adoption. This is an ideal feature for an evolving field of service science, with a diverging taxonomy of service classifications [18].

The targeted framework covers the basic requirement for practical usage in designing a service system, which includes:

- The ‘what’ aspect in the form of concepts definition, structure and relationship.

- The ‘how’ aspect in the form of stages definition, flow of artefacts and modelling tools.

To stay true as an abstracted form, the ‘how’ aspect of the framework is presented in a non-prescriptive fashion. While the framework omits the detail on the ‘how’ aspect, it should still be usable as a general guide by defining ‘what’ to produce in each step.

The framework covers both the business side and the technological side of a service system, but the elaboration of service system is focused onto the provision of a software service, as an implementation of a service-oriented software system. In focusing on the software service
analysis and design, the framework also omits the details commonly discussed within the context of software engineering.

1.3 Research Problem

As a multidiscipline concept, Service Engineering suffers from the lack of a cohesive perspective. Various concepts, terminologies and methods produced competitively within an academic domain, sometimes without regard to previously available produced context, while other academic domains may also produce intersecting concepts without a definitive attempt for a consolidation.

This research is an endeavour to consolidate the multiple perspectives of Service Engineering by pursuing formal answers to four research questions within the context of Service Engineering (SvE) analysis and design:

- RQ1: What concepts and component should be covered in SvE?
- RQ2: What are the activities to be undertaken during a SvE task?
- RQ3: What artefacts should be produced in SvE?
- RQ4: In what format should the artefact be presented?

The first research question covers the first requirement of a framework by defining the ontology for Service Engineering. The next two research questions cover the ‘how’ aspect of the Service Engineering by
defining the stages and produced artefacts. The last research question is an elaboration of the artefacts by suggesting an appropriate format or modelling language for each artefact to be produced.

1.4 Research Activities

With an objective of developing a General Service Engineering Framework (GSEF), the research activity is grounded on the *Design Science Research* (DSR) methodology [19]. DSR is characterized by a research motive of improving an environment by devising innovative artefacts [20], as illustrated in Figure 1.1.

Figure 1.1: Design Science Research Cycles [20]

In DSR frameworks, the environment provides the requirements for an emergence of a new artefact. In the context of this research, the requirement takes form as the need for a cohesive view on Service
Engineering framework. Therefore, the framework is the artefact to be built as the research output.

To propose the framework, the research draws from the existing knowledge base from several appropriate knowledge domains. The literature survey forms a foundation by identifying available theories, frameworks, methods, and models relevant to service engineering.

In DSR, the resulting artefact needs to fulfil the acceptance criteria of the environment to be applicable. In this context, the framework should be feasible and practical enough to design an intended service system. A series of case studies is performed to assess the utility and feasibility of the proposed framework in a practical environment. Adjustments and additional improvements are made along the conducted case study to improve the quality of the proposed framework.

A case study is defined as an empirical enquiry which examines a phenomena, not in an insular academic environment, but within its real-life context [21]. For the intended service engineering scope, the real-life context is the process of developing or improving a service system in an organization. The unit analysis is therefore the service engineering project with the involved stakeholders.

Two rounds of case studies are performed within this research. As an initial attempt, the first round of case studies is framed as both exploratory and confirmatory case study. As an exploratory study, three service engineering projects were undertaken based on the initial version
of the proposed framework to observe the sufficiency of the framework and the variance developed in the field.

The second round of case study was conducted as a confirmatory case study. A final case study is explored to assess the coverage of the proposed service engineering ontology. The case study also validates the refined framework produced from the first round of case studies.

In adherence to the DSR framework, the elaboration of the whole research activities can be grouped into four phases: (1) Synthesis phase,
(2) Case Study Verification phase, (3) Concept Refinement phase and, (4) Framework Enrichment phase. Figure 1.2 summarizes the flow of research activities.

During the synthesis stage, a literature survey is conducted to analyse the existing frameworks, to capture general requirement characteristics of a SvE and to identify components to be incorporated in the proposed GSEF. The phase will also identify pertinent artefacts and select modelling tools appropriate for each activity in the framework. During the synthesis phase, the first version of GSEF is defined to be tested on the case studies.

In the second phase, the case study verification uses the preliminary version of GSEF for three separate service engineering cases. Case studies were conducted within the scope of a municipal level government service. An evaluation is made for each case to assess the general feasibility of the framework, to capture the process variance and to identify previously uncovered important component of the framework. A consolidated analysis of the case studies results produces the second version of GSEF.

Towards the end of the second stage, the Concept Refinement phase is initiated in which another set of literature reviews is conducted. The goal of this stage is to define the ontological base of Service Engineering. Two set of ontologies were built: the software-service ontology and general service ontology. The two are correlated in which the software
service ontology is a specialization of general service ontology. To verify the resulting ontology, a fourth case study was undertaken.

During the final stage, the Framework Enrichment, an examination is conducted by mapping the framework to the ontology. The goal of this examination is to assess the sufficiency of the framework coverage in relation to the general ontology. The mapping also produced the stereotype of artefact modelling. Equipped with additional literature survey on existing metamodels, the final GSEF is produced: by proposing a set of available metamodel to present each artefact of the framework.

1.5 Contributions

The contribution of this research lies in its unique attempt to consolidate competing and overlapping perspectives of service engineering from both the business management domain and information system domain. While the framework addresses the concepts in an abstracted level, its presentation avoids formalized approach by emphasizing more on a pragmatic level and strives for simplification. This approach is chosen to improve the accessibility and practicality of the research result.

The main contribution of this research is the produced service engineering framework labelled as the General Service Engineering Framework (GSEF). The framework is presented in a structure of three layers component: Activity, Artefact, and Modelling. The activity characterizes the phases and steps of analysis and design process. The
artefact defines the output of each step while also implying the dependency and flow of the produced artefact. The modelling layer is an open container which suggests the available format to present the artefact, mostly based on specific modelling language. A combination of the layered components provides a general perspective in designing a service system.

As a part of the framework, an ontology set is also produced as a contribution. The ontology specifies and defines concepts pertinent within the context of service engineering, along with the defined relationship among the concepts. Departing from the Information System domain, the first produced ontology is a Software-Service Ontology. The second ontology is a generalization of the first ontology which defines generalized ontology of service system, drawn and consolidated from both the business management domain and the information system.

As an additional contribution, the research also provides the landscape of metamodel offering from both industrial and academic circle. The landscape is a product from exploring appropriate metamodel for representing an artefact. The metamodel landscape also offers a stereotype grouping of metamodels and illustrates the relationship between metamodel offerings.
1.6 Chapter Overview

This thesis is organized into six chapters. Figure 1.3 summarizes the partition of the thesis in relation with the flow of research activities. The first chapter, *Introduction*, provides the context of the research by elaborating the motivation, research objectives, research problems, research activities, and research contribution.

The second chapter, *Literature Review*, narrates the exploration into the available knowledge base relevant to service engineering in commencing the research. The chapter is divided into three parts: the classic conception of service engineering from the business-management domain, the information system domain perspective of service engineering, and the emerging trend of combining the two perspectives which create a consolidation gap to be fulfilled with this research.

The third chapter, *Service Engineering Ontology*, elaborates the basic foundation of the framework by describing a process of ontology building from a series of ontologies: from specific software-service ontology to general service system ontology. The chapter is divided into three parts: service-oriented architecture ontology, general service ontology, and software-service ontology. This chapter mainly addresses the first research question by defining and relating concepts covered in the framework.

The fourth chapter, *Service Engineering Framework*, describes the proposed framework and its refinement process. The chapter exposes the
preliminary version of the framework, describes the performed first round of case studies, and offers the second refined version of the framework. This chapter addresses the second and third research questions regarding the stages and artefacts of the framework.

Figure 1.3: Research Content Mapping
The fifth chapter, *Modelling in Service Engineering*, provides an exploration through the metamodel landscape. The chapter demonstrates the mapping of ontology and framework artefacts as a form of verification. Finally, the chapter proposes a general form of the framework by stereotyping the artefacts with a group of relevant metamodels. Consequently, this chapter focuses on the fourth research question, regarding artefact modelling.

The sixth chapter, *Conclusion*, acts as a thesis closure. Summary of results is provided in the context of research questions. A limitation and future work section is also provided to reflect and highlight the research limitation, followed by possible directions for future work in continuing the research result.

As a summary, this chapter has introduced the motivation of this research work, along with the selected approach taken in pursuing its goals. The next chapter is the first step taken in the approach by exploring existing literatures to examine state-of-the-art, to demonstrate a research gap and to identify pertinent concepts produced from previous works for proposing initial service engineering ontology and framework.
Chapter 2

Literature Review

2.1 Defining a ‘Service’

Given its nature as a multidiscipline endeavour, establishing a common paradigm is quite problematic in the service science field. Varying perspectives arise from different contributors with their own set paradigm influenced by a particular academic background. Therefore, one of the challenges in service science is to facilitate an interaction between various perspectives to define a shared and cohesive perspective [22].

The term ‘service’ carries a broad connotation according to each domain and context. Therefore, putting down a definitive definition of the term ‘service’ is not an easy task. In the most generic level, service can be broadly defined as ‘an act of beneficial activity’ [23]. From this general definition, a service can already be characterized as having at least four components: Service provider, service consumer, act of service, and benefit produced. The third component, the actual act of service, characterizes the whole process.

In Unified Service Theory, a service is defined as a production process where the customer possess a role in the process, by providing some of the input for the particular unit of production [22], as visualized in Figure 2.1. In a non-service-based process, the design idea of the
product may have been contributed by a group of customers, but there shall be no further involvement from customers in the production process. After the process, the customer role is limited only in selecting, purchasing, and consuming the product. This is the key difference between a service and a non-service process.

Figure 2.1: Service Input-Output Model [22]

All production processes typically require a resource. A transformative task is performed by the provider consuming the resources. In a service, the resource can take a non-physical form, e.g. knowledge, skill, or information. Ultimately, the process produces a (potential) value to be transferred for the benefit of consumer.

A wider definition of ‘service’ definition is offered in six contexts[16]:

1. *Service as an Interaction*: an interaction process or event between provider and client that creates and captures value.

2. *Service as a Capability*: an ability of an entity to produce an intangible benefit to its environment.

3. *Service as an Operation*: a part of an object’s behaviour defined in a component which can be invoked by a client via an interface.
4. **Service as an Application**: a piece of software accessible over the web supporting machine-to-machine interaction over a network.

5. **Service as a Feature**: an additional value offering provided on top of a basic item, such as in call-forwarding in a telephony system.

6. **Service as an Observable Behaviour**: a set of provider interaction behaviour externally visible by clients, over an interface.

Four analytical characteristics of service are derived from these six contexts: (1) involves interaction, (2) provides some value, (3) denotes a composite or decomposed unit of analysis, and (4) can be abstracted through a successive design process [16].

In a more pragmatic context of the Service Engineering (SvE) perspective, the term ‘service’ is used in three different contexts: Business service, electronic service, and software service [5].

- **Business Service**. In this context the term ‘service’ refers to a generic definition of the service as a business activity provided by a service provider to a service consumer to create value for the consumer.

- **Electronic Service**, or e-service in short, is a term used to denote any service that can be conducted via computer networks, i.e. the Internet. E-Service can be viewed as a subset of a business service, for a business service that employs computer networks in the encounter.
• **Software Service** is a type of e-service that can be accessed via web-based protocols or web-based programs from the consumer side. Web service, or a collection of it, can serve as a component of an electronic service.

The three service contexts form a service abstraction hierarchy in SvE, from business service as the highest abstraction to its technical form, software service. A higher abstraction can be partly composed from the lower abstraction. Specific in the IT technical context, i.e. the Service Oriented Architecture (SOA) perspective, the term ‘service’ usually refer to the third definition, software service, as a set of implemented logical process consumable by other software component [24].

The particular scope of this research encompasses the three definitions, specifically in determining a business service scope and then decomposing it into several relevant software service components.

### 2.2 Classic Service Engineering

The growing importance of the service industry motivates many originally non-service companies, e.g. manufacturing enterprise, to transform their business model into a *Product Service System* (PSS). In this endeavour the product offering is enriched with additional services through *servitization*. This trend initiated and stimulates studies in the service concept [25].
The early approaches in developing a service were originally published under the label of New Service Development (NSD), which was introduced in the business marketing domain during the eighties [26][27]. NSD is the ‘service’ version of the more established innovation process of New Product Development (NPD) in traditional goods-based industry [28].

Departing from service quality research with a focus on customer satisfaction, NSD emphasizes more on end-user (consumer) services rather than on the business-to-business (B2B) aspect [6]. NSD studies typically address particular service development issues, such as service pre-requisites, service blueprinting, and service development enablers. Several NSD approaches were proposed as a framework or (partial) processes without detailed methods and tools for the actual service development [27].

‘Service Engineering’ (SvE) terminology was coined in the mid-1990s as a competing term for describing the activity of service development. SvE is defined as a technical discipline to develop and design service systematically using appropriate models, methods, and tools [26]. SvE therefore enlarges the coverage of NSD by accommodating engineering and software development perspectives in service development [27]. SvE is also differed from NSD with the inclusion of business-to-business (B2B) services in its coverage [6].

In a broader context, the term service engineering carry three connotations: (1) as a systematic design and development task, (2) as an
organizational function, and (3) in the context of human resource management [6]. The first context is often supported with components of information technology. Therefore, a full scope of SvE concerns with the design and the development of a service system, composed of: (1) organizational design aspect, (2) human resources aspect and (3) information technology aspect [26]. The scope of this research will be limited to the context of design and development with emphasize on the information technology aspect. Figure 2.2 summarizes this perspective.

![Figure 2.2: Areas of Service Engineering [26].](image)

![Figure 2.3: Stages of Service Engineering [26].](image)

Figure 2.3 shows SvE in its original form. The activities are grouped into five sequential stages in a waterfall model: (1) Idea generation, (2)
requirement analysis, (3) concept development, (4) implementation, and (5) market launch. Alternatively, the stages can also be designed to follow an iterative prototype pattern with several cycles of Design–Analysis–Synthesis activities, before settling into a final master design [26].

Studies and research around service later coalesced into a new discipline, Service Science, which was later reintroduced as Service Science, Management and Engineering (SSME) [29][13]. This new discipline emphasizes on an interdisciplinary approach, combining management and engineering theories, including concepts from the IT domain. The underlying motive of this discipline is to advocate a continuous improvement of the organisation’s competitiveness by assessing the growing services needs of a business environment [30].

Under this multidisciplinary approach, several service design frameworks have been proposed. As a multidisciplinary approach, the framework commonly employs various qualitative analysis tools such as: Feasibility Study, Socio Techno-Economical Analysis, Environmental Scanning, Trend Analysis, Boston Consulting Group (BCG) matrix, and Quality Function Deployment (QFD). Two of such frameworks are presented here as representative of classic SvE perspective.

A synthesis of service engineering frameworks under NSD label is provided in [31]. It is structured as a 3-tuple SAT format: *Stage*, *Activity* and *Technique*. *Stage* is the highest conceptual level of the methods. A stage is a container of *activity set*, while the *technique* operationalizes the activity with several instruments or tools for performing the activity.
As illustrated in Figure 2.4, the frameworks consisted of 18 activities which grouped into five stages: (1) service identification, (2) service value net formation, (3) service modelling, (4) service implementation, and (5) service commercialization stages. Each activity employs management research techniques, such as survey or Strengths, Weaknesses, Opportunities and Threats (SWOT) analysis.

Figure 2.4: Synthesized NSD Framework [31]

This framework performs ‘Service Design’ process in the service modelling stage. The service modelling is composed of four components: product model, process model, resource model, and marketing concept as in [26]. The UML notation is employed as a tool for the service product model, while the Service blueprinting technique is suggested for process
model. The QFD matrix is proposed as the main analysis tool to relate customer needs with proposed service characteristics and features.

The NSD framework introduces the *Value Net Formation* stage as a special stage for business side analysis of the proposed service by way of Delphi, SWOT analysis and expert interview. Business model of the service is developed based on the needs of the consumers and the profits expectation of the firm. A service value net defines the roles of service participants with interdependent value proposition relationship.

Another SvE framework is proposed as the Service System Development Process (SSDP) [32] based on the stages of ITIL (Information Technology Infrastructure Library) [33], an industrial best practice of IT Service Management. Figure 2.5 illustrates the framework with 18 activities grouped into four iterative stages: (1) Service Need/Strategy/Concept, (2) Service Design & Development, (3) Service Transition/Deployment, and (4) Service Operation.

The framework divides the *service design and development* stage into (1) service requirement analysis, (2) entities and linkage definition, and (3) service modelling. The service modelling defines (1) functions (2), interfaces, (3) interoperability, and (4) agreements between participating entities.
In its core form, the process of service development within classic service engineering can be categorized into three major stages: service planning, service conception, and service implementation [6]. The first stage, service planning, involves idea generation, formation, and evaluation. In the second stage, service conception, the ideas are to be more precisely analysed and detailed, to be implemented in the last stage.

**2.3 Service Oriented Architecture**

In a related parallel during the emergence of service science and SSME, a new architectural style of software system gained popularity; the *Service Oriented Architecture* (SOA). SOA is formally defined as an approach to
manage the distributed capability of different domain ownerships[34]. In the technical context, the capability is encapsulated into a *software service* provided by software components.

In the SOA perspective, services are the building blocks of an enterprise. An enterprise is defined by a service pool, and its interaction pattern. SOA platform provides a unified mechanism for service offering, discovery, and interaction. Figure 2.6 visualizes the implementation of business process as a set of software service components. With this, SOA promises a better alignment between business process requirements and IT development processes.

![Diagram](image-url)

*Figure 2.6: Elements of Service Oriented Analysis and Design [35]*

Conceptually, SOA can be seen as an alignment between of two classes of enterprise artefacts: Business-oriented building blocks (BoB), and Technology-Oriented building blocks (ToB) [36]. BoB artefacts consist of Business Process, Business Event, Business Object, and Business Functions. A ‘*business-service*’ is an encapsulated provision of Business
Function data transaction into a Business Object, available to be invoked for Business Process execution. ToBs are implemented in the IT layer to reflect BoB’s behaviour, where some of the ‘business services’ can be automated as ‘software services’.

In *Software Engineering* (SwE) perspective, SOA is seen as a continuation of component engineering approach [37][38], in which instead of directly focusing on software components as an end product for a business, the process is mediated with the conception of business services and software services.

Service based software is built upon the idea of service reusability and composition. Several services can be combined to create new service. The orchestration configuration may be implemented (1) in the user interface, e.g. client web application, or (2) as a new composite service component. The services involved in the composition can be: (1) specially developed for the application, (2) developed within a company for another project, or (3) from an external provider [37].

The software service interaction is therefore no longer limited internally within an enterprise. The business landscape can be comprised of interacting software services among various organizations. These features are well-suited for the vision of a SSME’s business environment, the modern approach of SvE, and the demand for flexibility and agility in service operations.
2.3.1 Service Oriented Methodology

Since its inception around 2005, SOA concept experienced rapid growth from numerous contributors. But up to the point where its popularity subsides in and around 2010, a cohesive body of knowledge was not formed. SOA consistently suffers from the lack of a common ontology and structural definition [7][38]. During this period, numerous methodologies were offered by contributors to guide the implementation of a service-oriented system, as Service Oriented Methodology (SOM) [39].

The general objective of SOM is to translate enterprise business processes to a set of services by providing guidelines, standardized activities, and techniques. Figure 2.7 illustrates the components of SOM. It typically consists of two elements: development process, and modelling language [39]. The development process covers the elements of guidelines, techniques, activities, roles and responsibilities, verification-validation mechanisms, quality assurance, metrics, coding standards and tools. The modelling language element is used to represent produced artefacts in the development process phases.

![Figure 2.7: Structure of Service Oriented Methodology [39]](image)
Table 2.1: An Incomplete List of Service Oriented Methodology

<table>
<thead>
<tr>
<th>Acronym</th>
<th>SOM Name</th>
<th>Initiator</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>SOAD</td>
<td>Service-Oriented Analysis and Design</td>
</tr>
<tr>
<td>2</td>
<td>SOSE</td>
<td>Service Oriented Software Engineering</td>
</tr>
<tr>
<td>3</td>
<td>MSOAM</td>
<td>Mainstream SOA Methodology</td>
</tr>
<tr>
<td>4</td>
<td>SDLC</td>
<td>Web Services Development Life-Cycle</td>
</tr>
<tr>
<td>5</td>
<td>SEPA</td>
<td>SOA Evolutionary Programming Approach</td>
</tr>
<tr>
<td>6</td>
<td>SOUP</td>
<td>Service-Oriented Unified Process</td>
</tr>
<tr>
<td>7</td>
<td>MSA</td>
<td>Methodology for Service Architecture</td>
</tr>
<tr>
<td>8</td>
<td>SOAF</td>
<td>Service-Oriented Architecture Framework</td>
</tr>
<tr>
<td>10</td>
<td>SOADAS</td>
<td>SOAD for Adaptable Service</td>
</tr>
<tr>
<td>11</td>
<td>SOMA</td>
<td>Service Oriented Modeling and Architecture</td>
</tr>
<tr>
<td>12</td>
<td>METS</td>
<td>Method for Engineering True SOA</td>
</tr>
</tbody>
</table>

Table 2.1 lists 12 different SOMs collected from academic survey papers [39][49][50][51]. Some of the obscure SOMs source material may no longer be available, but two consistently popular SOMs are IBM's SOMA [47] and Thomas Erl's Mainstream SOA methodology (MSOAM) [24][50]. For illustrative purpose, these two methodologies are described in the following narratives.

IBM’s SOMA is a recommended SOA method due to its comprehensiveness and its vast industrial adoption [51][49]. It covers a complete cycle of ‘service engineering’, from the business side to the technical implementation. As can be observed in table 2.2, the method
A variation of SOMA proposes the use of a Component Business Model (CBM), as a capability analytical tool in the early stage of the methodology [35]. CBM is a matrix of the enterprise’s business competencies with levels of accountability, i.e. directing, controlling and executing [52]. The resulting elements of the matrix are defined as business components. Each component has the definition of: (1) purpose, (2) activities, (3) resources, (4) governance, and (5) offered services.

Mostly referred as SOA study material, Thomas Erl’s MSOAM is the definitive source for understanding SOA concepts. As summarized in table 2.3, the methodology consists of seven stages of activities: (1)
Ontology Definition, (2) Business Model Alignment, (3) Service Oriented Analysis, (4) Service Oriented Design, (5) Service Development, (6) Service Testing, and (7) Service Deployment. Despite the seven-stage process, the reference only explores and elaborates the analysis and design stages.

<table>
<thead>
<tr>
<th>Table 2.3: Stages of MSOAM Methodology [24]</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Define Relevant Ontology</td>
</tr>
<tr>
<td>2. Align Relevant Business Models</td>
</tr>
<tr>
<td>3. Perform Service Oriented Analysis</td>
</tr>
<tr>
<td>3.1. Define Business Requirement</td>
</tr>
<tr>
<td>3.2. Identify Automation System</td>
</tr>
<tr>
<td>3.3. Model Candidate Services</td>
</tr>
<tr>
<td>3.3.1. Decompose Business Process</td>
</tr>
<tr>
<td>3.3.2. Identify Operation Candidate</td>
</tr>
<tr>
<td>3.3.3. Abstract Orchestration Logic</td>
</tr>
<tr>
<td>3.3.4. Create Service Candidates</td>
</tr>
<tr>
<td>3.3.5. Refine &amp; Apply Service Orientation</td>
</tr>
<tr>
<td>3.3.6. Identify Service Compositions</td>
</tr>
<tr>
<td>3.3.7. Revise Operation Grouping</td>
</tr>
<tr>
<td>3.3.8. Analyze Processing Requirements</td>
</tr>
<tr>
<td>3.3.9. Identify Application Service Operations</td>
</tr>
<tr>
<td>3.3.10. Create Appl. Service Candidates</td>
</tr>
<tr>
<td>3.3.11. Revise Service Compositions</td>
</tr>
<tr>
<td>3.3.12. Revise Operation Grouping</td>
</tr>
<tr>
<td>4. Perform Service Oriented Design</td>
</tr>
<tr>
<td>4.1. Compose SOA</td>
</tr>
<tr>
<td>4.1.1. Choose Service Layer</td>
</tr>
<tr>
<td>4.1.2. Position Core Standards</td>
</tr>
<tr>
<td>4.1.3. Choose SOA Extensions</td>
</tr>
<tr>
<td>4.2. Design Entity-Centric Business Serv.</td>
</tr>
<tr>
<td>4.3 Design Application Services</td>
</tr>
<tr>
<td>4.3.1. Review Existing Services.</td>
</tr>
<tr>
<td>4.3.2. Confirm Context</td>
</tr>
<tr>
<td>4.3.3. Derive Initial Interface</td>
</tr>
<tr>
<td>4.3.4. Apply Service-Orientiation.</td>
</tr>
<tr>
<td>4.3.5. Standardize Service Interface</td>
</tr>
<tr>
<td>4.3.6. Add Speculative Features</td>
</tr>
<tr>
<td>4.3.7. Design Serv. Oriented Business Process</td>
</tr>
<tr>
<td>4.3.8. Derive Initial Interface</td>
</tr>
<tr>
<td>4.3.9. Apply Service-Orientiation.</td>
</tr>
<tr>
<td>4.3.10. Design Task-Centric Business Serv.</td>
</tr>
<tr>
<td>4.3.11. Map Out Interaction Scenarios</td>
</tr>
<tr>
<td>4.3.12. Design Process Service Interface</td>
</tr>
<tr>
<td>4.3.13. Formalize Partner Serv. Conversation</td>
</tr>
<tr>
<td>4.3.14. Define Process Logic</td>
</tr>
<tr>
<td>4.3.15. Align Interaction Scenarios &amp; Refine</td>
</tr>
<tr>
<td>5. Develop Services</td>
</tr>
<tr>
<td>6. Develop Test Service Operations</td>
</tr>
<tr>
<td>7. Deploy Services</td>
</tr>
</tbody>
</table>

Theoretically, SOA approach can be divided into two strategies: Top down, and Bottom up [24]. In a top down strategy, the completion of a full inventory analysis must be achieved prior to the design, development, and delivery of services. In a bottom-up strategy, a focus is set for a fulfilment of specific business priority requirements and existing applications can be used as the starting point for an analysis.
By comparing SOMA and MSOAM concept coverage in figure 2.8, it can be observed that SOM mostly covers the technical part of the analysis, in detailing the concepts emerged from business process aspect. In SOMA case, a SOM can also be considered as a SvE framework or a partial SvE framework integrated with business side analysis [53]. In this sense, a SOM can also be viewed as a SvE framework with emphasize on IT perspective.

![Figure 2.8: Comparison of SOMA and MSOAM concepts coverage](image)

### 2.3.2 Modelling in Service Oriented Analysis

The heart of a SOM is the analysis and design activity which employs modelling notation. MSOAM framework traditionally uses a simplified UML class-diagram, in which a circle symbolizes a service, an arrow for message between services, and tree-like hierarchy for service
composition. Other UML notations are also commonly used, e.g. class
diagram, and interaction diagram.

As summarized in table 2.4, six modelling approaches for SOA were
identified: (1) UML, (2) SOMF, (3) PIM4SOA, (4) SoaML, (5) SCA, (6)
SRML [54][55]. Some of these modellings are related: SoaML is an
extension of UML, SRML is inspired from SCA and some of PIM4SOA
concepts were brought forward to SoaML.

Table 2.4: Service Oriented Modelling Language

<table>
<thead>
<tr>
<th>Acronym</th>
<th>SOM Name</th>
<th>Initiator</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 UML</td>
<td>Unified Modelling Language</td>
<td>OMG, 1997 [56]</td>
</tr>
<tr>
<td>2 SCA</td>
<td>Service Component Architecture</td>
<td>OASIS, 2005 [34]</td>
</tr>
<tr>
<td>3 PIM4SOA</td>
<td>Platform Independent Modelling for SOA</td>
<td>Benguria G,2007 [57]</td>
</tr>
<tr>
<td>4 SRML</td>
<td>Sensoria Reference Modelling Language</td>
<td>Sensoria, 2007 [58]</td>
</tr>
<tr>
<td>5 SOMF</td>
<td>Service Oriented Modelling Framework</td>
<td>Bell M, 2008 [59]</td>
</tr>
<tr>
<td>6 SoaML</td>
<td>Mainstream SOA Methodology</td>
<td>OMG, 2009 [60]</td>
</tr>
</tbody>
</table>

Only two of them are still popular as an operational SOA modelling
notation: UML and SoaML. UML notation is often used for modelling a
simple SOA design [61][62]. For a relatively more complex SOA system,
SoaML is suggested [63][64].
2.4 Informatics Service Engineering

Besides contributing with the concept of IT Service Management, i.e. ITIL, in the service sector, IT also provides the concept of Service Orientation in computing and enterprise architecture. With the advent of SOA, enterprises were drawn to adapt their enterprise applications into a service-oriented system, to increase the business agility.

![Figure 2.9: Scientific Publication of ‘Service’ Modelling][65]

With the growing pervasiveness of IT in service industry, especially in providing e-Services, the study on SvE gained momentum in the IT field. This trend can be observed from the number of SOA related ‘service’ publications as visualized in graph of figure 2.9. The trend grew exponentially around 2003 before reaching its peak in 2009 and experienced downward trend afterward [65]. More recent academic studies on SvE tend to be produced by IT contributors with a tight conceptual relation with IT components.
In the context of e-service, the SvE is defined as an approach of using models and techniques in guiding the sequential process of e-service conception, analysis, design, implementation, deployment, operation, maintenance or modification [5]. Within this scope, the objective of the SvE framework is to provide services in the IT layer, specifically in the form of web services which will allow a degree of automated interaction among software component.

2.4.1 Service Engineering and Software Engineering

Differences between SvE and SwE were explored long before SOA concepts gained traction in [66]. Two identified differentiators are in the (1) life-cycle, and (2) stakeholder. Service development and deployment is commonly performed in an environment with continuous system maintenance, change and enhancement process. Software product in the other hand normally has a life-cycle in more traditional sense.

The second difference is that unlike SwE, which normally has only one responsible organization for system deployment, SvE often involves more than one organisation in a service deployment [66] in forming a multi-provider service system from a ‘service ecosystem’ [2].

SvE introduces new roles traditionally not discussed in traditional SwE, i.e. service provider, service requester and intermediaries [8]. Therefore, software services are not always designed and built for one
particular business unit but can also take into account the exposure potential for others in a totally new business context.

Additionally, software service development introduces a new concept of ‘dynamic service binding’ to support runtime collaborative processes. But to enable this, a mechanism of automated service discovery, service selection and service invocation is required in the service system [8].

While the development of software services can be seen as a special case of SwE, SvE normally covers a wider context of analysis in identifying services by taking consideration of an enterprise-scale business process and application architecture [8]. In terms of SOA adoption, SvE is seen as a higher abstraction of a software engineering process. If the specific objective of software engineering is a (software) system, the objective of SvE is more of an enterprise-wide business service system.

2.4.2 Service Engineering and SOA

Incorporating SOA into classic SvE has been naturally proposed. An early proposal for including SOA in SvE is found in [13], as detailed in table 2.5. The proposed SvE framework consists of three stages: Requirement Identification, Service Design, and Service Delivery. SOA approach is integrated into the second phase, Service Design, as service technology that provides principles and governing concepts for systems development and integration. SOA is employed for its ability to combine several
functionalities to form ad-hoc applications that can be built almost entirely from existing software services.

Table 2.5: Service Engineering Framework with SOA [13]

<table>
<thead>
<tr>
<th>Stage</th>
<th>Name</th>
<th>Brief Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Requirement investigation</td>
<td>Quantitative</td>
<td>Questionnaire Close-formed market research</td>
</tr>
<tr>
<td></td>
<td>Qualitative</td>
<td>Kano Classify customer preferences to five categories</td>
</tr>
<tr>
<td></td>
<td>Interpretive Method</td>
<td>Customer club, personal interviews</td>
</tr>
<tr>
<td></td>
<td>Behaviour Observation</td>
<td>Direct observation of human behaviour for design requirement</td>
</tr>
<tr>
<td></td>
<td>Critical Theory</td>
<td>Customer complaints, feedback of lost customers, call centre</td>
</tr>
<tr>
<td></td>
<td>Complaint Analysis</td>
<td>Investigation of complaints</td>
</tr>
<tr>
<td></td>
<td>Focus group, Delphi, other</td>
<td>A group of people are asked interactively about their attitudes towards a product</td>
</tr>
<tr>
<td></td>
<td>Meeting forms etc</td>
<td>Service QFDidentifies critical significance activities</td>
</tr>
<tr>
<td></td>
<td>Critical Incident Technique</td>
<td>Identifies potential failure modes, determine their effect, and identify mitigative action</td>
</tr>
<tr>
<td>Service Designing</td>
<td>System</td>
<td>SOA Architecture IT architectural approach integrating business as linked,</td>
</tr>
<tr>
<td></td>
<td>Service Blueprinting</td>
<td>Describes detailed service implementation</td>
</tr>
<tr>
<td></td>
<td>House of Quality</td>
<td>Graphic tool to define the relationship between customers’ desires and the firm’s</td>
</tr>
<tr>
<td></td>
<td>Service QFD</td>
<td>Quality approach in designing based on customers’ need and values</td>
</tr>
<tr>
<td></td>
<td>Service FMEA</td>
<td>Identify potential failure modes, determine their effect, and identify mitigative</td>
</tr>
<tr>
<td>Service Delivery &amp;</td>
<td>Test</td>
<td>Service Prototyping Prototyping for user testing</td>
</tr>
<tr>
<td>Improvement</td>
<td>User Satisfaction</td>
<td>Satisfaction Questionnaires Ex-post analysis of customer satisfaction</td>
</tr>
</tbody>
</table>

From an IT perspective, SvE is often analogous with SOA adoption, in which (software) services are developed for reuse in service oriented applications [37]. Figure 2.10 illustrates SvE as a service-orientated re-engeneering of an enterprise-wide system in [67]. The process is performed by aligning the layers of (1) Business Model, (2) Business Architecture, and (3) IT Architecture. SOA resides in the IT architecture layers as a resource providing infrastructure services. In turn, the
technological changes brought by SOA allow for new business strategies and adjustment, which eventually can also stimulate service innovation.

A more detailed description of the role of SOA in SvE is provided by [53]. A SvE framework is proposed under the name of a ‘consolidated approach for business and software services identification and analysis. The proposed framework divides the approach into two parts: (1) derivation of a business service and (2) derivation of software services. Both parts are composed of three stages: (1) preparation stage, (2) identification stage, and (3) detailing stage. A fourth stage, prioritization stage, is added in the end of the first part, connecting the two parts.
Figure 2.11: Derivation of Business Service [53]

Figure 2.11 visualizes the derivation of business services, as a top down analysis to produce a hierarchy of identifiable services. At the end of the detailing stage, a model of interacting business services is formalized as a ‘non-technical SOA design’. The business services are
later prioritized to determine services to be enabled with software services.

Figure 2.12: Derivation of Software Service [53]

The derivation of the software services part in figure 2.12 uses a *meet-in-the-middle* approach by taking into consideration the legacy system to be juxtaposed with required (atomic) software services for a
‘technical SOA design’. During the identification stage, the potential for composition and consolidation of services is also analysed.

In the detailing phase, a decision must be made if the existing software service should be extended or adapted. Services are specified to avoid overlapping and to make it more reusable and autonomous. The resulting services are then published in a service inventory.

This consolidated approach embodies the purpose of SvE in identifying services that may be provided on the business and technical levels. The approach covers an end-to-end process of mapping business services to software services. The work clearly suggests that SOA can be an integral part of SvE in the case of software-service provision.

From these examples of IT perspective frameworks in [67][53], SvE can be viewed as a broad process of applying service orientation analysis to an organization, or implementing the SOA approach in the enterprise. This view is also supported in the software engineering domain [37].

As an approach of adopting SOA, the framework addresses more on the issue of business-IT alignment, rather than creating a new (business) service [68]. Service innovation is an additional result brought by the increased agility.

2.5 Research Opportunities in Service Engineering

Despite the research contribution over two decades, Service engineering (SvE) is generally still regarded as an unfinished research challenge in
the literature due to mostly partial and fragmented approach taken [7][8]. A gap still exists in consolidating the perspectives related to the multi-discipline nature of service science discipline, especially to answer the dynamic future of service ecosystem [27].

To date, various SvE frameworks have been proposed. In its original conception, SvE is defined as a process of identifying and developing newly conceived business services [31][26][32]. Most of the frameworks reside in an abstract level by focusing on value propositions instead of conceptualizing the targeted service systems.

The emergence of smart device and contextual data suggests that the future of service innovations lies in the recombination of service offerings, where the innovation is not solely derived from an internal top-down process, but also by considering resources and solutions supplied by different service participant in forming a collaborative network and virtual organization [27][69]. Current SvE frameworks mostly still neglect the prospect of a multi-provider service system [27].

The dominant use of service blueprinting in the classic SvE stream might be insufficient to portray the formation of a multi-provider service, the detailed back-end process, and the involvement of IT components. The incomplete model may compromise the quality of the service development process.

In the technological perspective, influenced by service-orientation, SvE is positioned as a top-down analysis process of an existing enterprise business service structure to be translated as software services in an IT
infrastructure [67][53] as in the SOA approach. The goal is to improve the business-IT alignment and the agility to introduce new service innovations.

In this technical perspective, SvE often takes form as Service Oriented Methodologies (SOM). Unlike the case in Software Engineering, where certain methodologies become popular and dominant, e.g. RUP or Agile, SvE experiences a continuous proliferation of SOM, while none of them can be considered comprehensive and integrated enough to cover both business and software services [53].

A comprehensive service engineering methodology is therefore expected to provide an integrated approach to cover both the business side and the software side of services, to ensure business-technical alignment, and to improve process agility.

![Figure 2.13: Context Positioning of Research Scope](image)

The positioning of this research is visualized in figure 2.13. The research attempts to consolidate the two perspectives of SvE for specific business services innovations, as originally defined in classic SvE. The
The proposed framework focuses on specific business services analysis and design rather than assessing the whole of service provider’s capabilities.

The research covers the aspects from service business needs to derivation of software services. The goal is not to devise the service-oriented software system, but to identify applicable software services accommodating transaction interactions.

In elaborating the result of a literature survey, this chapter has explicitly demonstrated an academic necessity and a research gap to be fulfilled by this research work. In the same time, the chapter also collates relevant previously proposed concepts to be collected in building the research target, i.e. the ontology and the framework. This collation of relevant concepts is to be elaborated in the next chapter by identifying components related to the ‘service engineering’ concept to define an ontological basis for service engineering framework.
Chapter 3
Service Engineering Ontology

3.1 Ontology Defined

‘Ontology’ departed as a branch of philosophical study in the 17\textsuperscript{th} century which explores the concept of ‘existence’ [70]. As a study, it concerns with the conceptualized abstraction form of nature’s entities and their relationship. From this philosophical root, the term ‘ontology’ is defined as a set of structured (abstract) concepts within a defined domain. The structure may consist of categorization, attributes and relation between concepts. It serves a basis for a language, to be used as a communication tool to share an understanding regarding a specific domain.

Due to its capability in abstracting a real-world system, the term ‘ontology’ was later adopted by computer and information science in the 20\textsuperscript{th} century [70], first in data modelling and later as ‘domain ontology’ in artificial intelligence domain. In this pragmatic level, ontology is constrained as the definitions and inference rules for a set of words. The definition represents the concepts, while the inference rules define the relations between concepts.

Ontology is tightly related to modelling activity. A ‘model’ is defined as a representation of a reality within a definite purpose. To facilitate a common understanding between multiple parties, a model is usually built
based on a specific modelling language, i.e. metamodel. The metamodel specifies a palette of concepts and constraint rules for a valid model for a specific modelling language[71].

Figure 3.1 describes the relation between an ontology and a model. Ontology is an explicit and formal specification of a shared conceptualization, in both model and metamodel level. Two types of ontology are therefore involved: (1) ontology of meta models, and (2) ontology of problem domain, or a ‘domain ontology’ [72]. A model is an instantiation of a ‘meta model’ and similarly, a ‘domain ontology’ is an instantiation of a ‘meta model ontology’. Both model and metamodel are semantically interpreted by their respective ontology.

Figure 3.1: Model, Metamodel and Ontologies [72]

A shared common understanding is essential within a collaborative effort. During the construction of this research objective, i.e. “Service Engineering Framework”, a series of ontologies is developed. As the framework is a container for development activity, the ontology built is in
the ‘metamodel ontology’ level. The idea is to juxtapose the ontology with metamodel usage during each analysis and design activity.

Two ontologies are presented in this chapter: (1) Software Service Ontology and (2) General Service Ontology. The first ontology is built from the relatively established SOA stream, which is later adapted and generalized to cover non-technical perspectives originated from classic service engineering. Figure 3.2 provides a map of this chapter in the context of the overall research flow presented in figure 1.3.
3.2 Related Work

As mentioned earlier, service science still experiences a lack of standard ontology for ‘service’, and ‘service system’ concepts. Recent service engineering studies mostly emerges from IS/IT contributors with a useful perspective on ontology. While the volume is lower than the SOA methodology offerings, several ontology propositions do exist. Several of such works is summarized here.

Figure 3.3 is one of early attempts in defining a service ontology, as a service system metamodel, within the context of a SOA methodology [73]. The simple abstraction only covers nine components. While the focus is on software-service, it already relates to one non-IT concept: business process.

Figure 3.3: Metamodel of Service System [73]
In this ontology, service is differentiated based on participant type:

1. For service consumers, a service is a unit of expected functionality with a service level agreement as ‘Target Service’. (2) For service providers, a service is a unit of deployed functionality as ‘Publishable Service’. ‘Service Interface’ serves as a front-end for ‘Service Component’, which can be in an atomic or a composite form.

Another ontology proposition from software service-orientation perspective is found in [16]. It defines three successive abstractions of a service: (1) single interaction, (2) multiple interactions (choreography), and (3) multi-provider (orchestration). Five overlapping aspects of a service model were also defined: (1) structure, (2) behaviour, (3) information, (4) goal and (5) quality. The structural aspect of a service is conceptualized as a metamodel covering 12 concepts, entirely from a SOA perspective (figure 3.4.).
Figure 3.5 presents SOA metamodel proposed by CBDI [74], a consultant firm known as an active participant in Object Management Group’s (OMG) SoaML initiative. The metamodel already covers the concept of ‘capability’.

![Service Package of SOA Metamodel](image)

Figure 3.5: Service Package of SOA Metamodel [74]

An enrichment of CBDI’s SOA metamodel is proposed in [75]. Within a service engineering process, a business modelling activity is followed by a service modelling. The service is specified based on five components: (1) inner structure, (2) interface, (3) operation sequence, (4) information type, and (5) contract.

This notion of service components is also reflected in the business level. A business service can be seen as an aggregation of overlapping components such as: (1) underlying process, (2) contract, (3) owner, i.e. providing participant, (4) exposed operation, i.e. interface, (5) operation model, i.e. operation sequence, an external interaction model, i.e. interface and contract [53].
As an attempt to consolidate the non-orthogonality of competing SOA concepts, a literature survey on SOA concepts is performed in [38] and presented in table 3.1. Nine core identifiers which characterize a service-orientation are extracted: (1) architecture, (2) binding, (3) capability, (4) composition, (5) contract, (5) delivery, (6) distributed sources, (7) identity, and (8) interoperability. Further attempts to operationalize these characteristic into service modelling did not lead to a ontological view of SOA or service [62].

Table 3.1: SOA Characteristic Identifiers [38]

<table>
<thead>
<tr>
<th>No</th>
<th>Identifier</th>
<th>Description</th>
<th>Related Terms</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Architecture</td>
<td>Overall system organization built from services elements, interacting through a mechanism</td>
<td>Architectural paradigm, architectural style, software architecture</td>
</tr>
<tr>
<td>2</td>
<td>Binding</td>
<td>The time at which a particular service (and provider) is chosen</td>
<td>Agility, dynamic binding, flexibility, loose coupling, on demand</td>
</tr>
<tr>
<td>3</td>
<td>Capability</td>
<td>The purpose of an SOA as viewed from an end-user perspective</td>
<td>Business functions, resource management</td>
</tr>
<tr>
<td>4</td>
<td>Composition</td>
<td>The assembly process of services set to provide a single service that meets a need.</td>
<td>Choreography, integration, orchestration, service composition</td>
</tr>
<tr>
<td>5</td>
<td>Contract</td>
<td>The agreement mechanisms for the terms and conditions for delivered service</td>
<td>Service contracts, service negotiation</td>
</tr>
<tr>
<td>6</td>
<td>Delivery</td>
<td>The process where service functionality is supplied by the service providers the needs.</td>
<td>service interaction, service invocation, service provider, service consumer</td>
</tr>
<tr>
<td>7</td>
<td>Distributed Sources</td>
<td>Delivery across network, owned and controlled by different parties</td>
<td>different ownership, distributed system architecture, network environment,</td>
</tr>
<tr>
<td>8</td>
<td>Identity</td>
<td>Description characteristic of a service and the means of access</td>
<td>service description, service discovery, service publication, service registry,</td>
</tr>
<tr>
<td>9</td>
<td>Interoperability</td>
<td>Service deployment masks location and technical details</td>
<td>interfaces, platform independence, standards, messaging protocols</td>
</tr>
<tr>
<td>10</td>
<td>Packaging</td>
<td>Unique and distinct identity characteristic of service implementation</td>
<td>component, granularity, reusability, self-containment, web services</td>
</tr>
</tbody>
</table>

From these previous works, selected concepts that can be considered as potential components for targeted ontology are: (1) architecture, mostly for software level abstraction, (2) binding, related to ‘role’ concept, (3)
capability, (4) composition, related to atomicity or composite nature of service, and (5) contract. Additionally, structural modelling for a service is defined to be consisted of: (1) service operation, (2) service component, and (3) service interface [62].

### 3.3 Software Service Ontology

As demonstrated in [62], integrating SOA concepts from individual contributors is not always a feasible and usable approach. A more practical approach is hence taken to use collaborative SOA standards as the source for ontology building. Over the years, several standard groups have produced SOA open standards: OASIS, The Open Group, OMG and ISO/IEC.

The standards published is not always compatible to each other, and actually competing in its overlapping terminology [76]. As illustrated in figure 3.6, two products can be considered as the state-of-the-art: (1) OMG's SoaML, as the definitive SOA metamodel originated from OASIS stream, and (2) ISO/IEC's SOA Reference Architecture, as the latest SOA ontology definition originated from The Open Group stream.

Ideally the two should be related semantically, in which the (SoaML) metamodel uses the concept defined in the (SOA-RA) ontology [72]. In reality, that is not the situation due to the difference in the originating sources. Therefore, these two products are treated separately in this
research as sources for the Software Service Ontology. Comparison is performed between the two as a research triangulation.

![Figure 3.6: Succession of SOA Open Standard [76]](image)

### 3.3.1 Ontology of SoaML

SoaML was first formalized in 2009, with minor updates later in 2012 [60]. SoaML is conceived based on the limitations of UML in representing SOA concepts [63]. While its popularity in the industry is very limited, SoaML is consistently referenced in academic publication as the definitive metamodel for SOA [64][77]. Table 3.2 lists SOA concepts covered by SoaML.

SoaML can be seen as an ‘agnostic’ metamodel for SOA. The specification offers options to be decided by the modeller to cater a broad range of SOA implementation types, from simple SOA, e.g. flat REST-
based web-services, to complex hierarchical SOA, e.g. SOAP/WSDL-based with a centralized service bus.

Table 3.2: SoaML Conceptual Components [60]

<table>
<thead>
<tr>
<th>No</th>
<th>Concept</th>
<th>Description</th>
<th>SoaML Representation</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Participant</td>
<td>Entities (physical or software) that provide or use services</td>
<td>Participant (ports and interfaces container), part of service architecture</td>
</tr>
<tr>
<td>2</td>
<td>Port</td>
<td>Participant’s service interaction points in providing or consuming services</td>
<td>(Small square) Part of participant, might contain (interface) spoke or socket</td>
</tr>
<tr>
<td>3</td>
<td>(UML) Interface</td>
<td>A type of service interaction description for synchronous unidirectional interaction</td>
<td>(UML) Interface, might be part of Service Interface diagram, spoke or socket in participant diagram</td>
</tr>
<tr>
<td>4</td>
<td>Service Interface</td>
<td>A type of service interaction description for (multiple) asynchronous interaction</td>
<td>Service interface diagram, might contain several (UML) Interface</td>
</tr>
<tr>
<td>5</td>
<td>Service Contract</td>
<td>A type of service description based on roles and rules as an agreement for multi-party interaction</td>
<td>Service contract, might be part of Service Architecture, might contain several (UML) Interface</td>
</tr>
<tr>
<td>6</td>
<td>Capability</td>
<td>Ability owned, or required, by participant to affect some changes.</td>
<td>Capability diagram</td>
</tr>
<tr>
<td>7</td>
<td>Role</td>
<td>A specific functionality assumed by participant in an instance of service interaction</td>
<td>A label in Service Architecture, Service Contract, and Interaction Protocol</td>
</tr>
<tr>
<td>8</td>
<td>(Role) Binding</td>
<td>A pairing instance of a participant with a role within a specific service interaction context</td>
<td>(Instance of service contract, contact of participant's spoke and hub)</td>
</tr>
<tr>
<td>9</td>
<td>Interaction protocol</td>
<td>(Sequential arrangement of operation invocation between role/interface)</td>
<td>UML behaviour diagram as part of service interface, or service contract</td>
</tr>
<tr>
<td>9</td>
<td>Operation</td>
<td>(An atomic invokable software behaviour with input-output message passing feature)</td>
<td>Labelled component in Interface, or labelled invocation in interaction protocol</td>
</tr>
<tr>
<td>10</td>
<td>Message type</td>
<td>Data values that can be sent between participants</td>
<td>Message type, might be related to (UML) data type and entity attribute</td>
</tr>
<tr>
<td>11</td>
<td>Service Architecture</td>
<td>High level description of connection between participants through service contracts within a specific service community</td>
<td>Service architecture: contains participants, service contracts, and role-labelled connectors</td>
</tr>
<tr>
<td>12</td>
<td>Method</td>
<td>Owned behaviour of a participant</td>
<td>UML behaviour of participant describing its inner behaviour</td>
</tr>
</tbody>
</table>

While providing a formal specification of its stereotyping extension from the original UML specification, SoaML document specification is surprisingly lacks an ontological definition. SoaML specification actually offers two types of service modelling approaches: (1) Interface-based and,
(2) Contract-based [60][78], and therefore multiple forms of service abstraction is permissible [77], [79].

Apart from the OMG’s specification document, the availability of SoaML reference material is quite limited. The combined factor of the complexity, the multi-interpretative characteristic, and the limitation of source forms a barrier for entry in SoaML exploration. With the declining trend of SOA popularity as a terminology, this may contribute to its lack of adoption in industrial cases.

A peculiar feature of SoaML is the absence of specific abstraction for ‘service’. Three abstractions are offered superimposed to service description components [60] as:

1. Interface, accommodating atomic services containing only self-contained operations

2. Service Contract, accommodating atomic services and composite services by combining interfaces into a service contract.

3. Service Interface, accommodating atomic services and for composite services by combining interfaces a service interface.

Consequently, there are three versions of an ontological structure that can be inferred from the specification: (1) Interface-based, (2) Service Contract-based, and (3) Service Interfaced-based. These versions are elaborated in the following paragraph. To simplify the visualization of the differences, a special technique is employed where: (1) Compositional
relationship is visualized in the form of a Venn diagram, whereas components are placed inside a container representing the compositional parent, (2) the service abstraction is visualized with a thicker border in the superimposed service description component.

Figure 3.7 visualizes the first version in which ‘service’ abstraction is superimposed to the (UML) ‘interface’ concept. The approach is used for simple SOA where the whole architecture is composed of flat atomic services without the possibility of a service composition. Each interaction is synchronous involving exactly two participants with an interface embedded with a specific role; either as a service requester in a consumer role or as a service responder in a provider role.

![Figure 3.7: Interface-based Service version of SoaML Ontology](image)
Figure 3.8 visualizes the second ontological version, in which a ‘service’ is abstracted with the ‘service contract’ concept. Service contract is equipped with an interaction protocol as an internal logic that governs invocation sequences, both synchronous and asynchronous, between participants. The approach is able to accommodate a service interaction with more than two participants. Service composition is possible in the form of a compound service contract [60][78].
Figure 3.9 illustrated the third version of the ontology, where ‘service’ is abstracted as both ‘interface’ (for atomic service) or ‘service interface’ (for composite service). Service interface has an interaction protocol as an internal logic to govern invocation sequencing in both synchronous and asynchronous invocation between participants. Service interface can also accommodate an interaction service with more than two participants. Service contract can be omitted if there is no need for service invocation rules.

There is no evidence to support the ability of a service interface to refer to another service interface. Therefore, only one level of service composition is assumed possible. SoaML is very unrestricted in specifying service composition techniques. Apart from using the service contract or service interface as an abstraction of a composite service, a composition can also be modelled at the participant level (i.e. component) by defining sub-participants interaction inside a super-participant.
Theoretically, there is a fourth version in which the three options are combined, where a service can take the form of an interface, service interface or service contract within a single design context. But this may lead to an inconsistency and unnecessary complexity. In this attempt to be implementation agnostic, the unrestricted specification makes SoaML a complex and a rather confusing modelling tool.

To simplify the SOA ontology building, a specific SoaML perspective is prescribed. By considering to the availability and popularity of a practical SoaML guidance material in [77], which adheres to the service interface-based approach, the third version of the ontological structure is selected. The SOA ontology is therefore presented in the figure 3.10 as a reformulation from similar relationships in figure 3.9. Concept definitions are consistent with the previously listed table 3.2.

The produced ontology covers most of the components mentioned in the section 3.2. The components are mostly abstract design concepts, except for three components: (1) participant, to become some form of software component, (2) simple interface and (3) service interface to be implemented as an atomic or a composite software service, e.g., web service and WSDL interface.
Considering the fact that the paired component (e.g. Participant vs. Software Component) is actually the same object in different stages of the engineering process (i.e. design vs. implementation), the implementation instance of ‘component’ is not included in the ontology.

### 3.3.2 ISO/IEC SOA Ontology

Apart from the SoaML minor revision in 2012, ISO/IEC 18384:2016 (SOA Reference Architecture) [80] is a rare example of SOA standard published after SOA popularity decline in the 2010s. In actuality, ISO/IEC18384:2016 is mostly based on two previous products of The

Table 3.3: ISO/IEC 18384:2016 SOA Conceptual Components [80]

<table>
<thead>
<tr>
<th>Concept</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Element</td>
<td>Unit at a given level of abstraction with a clearly defined boundary.</td>
</tr>
<tr>
<td>2 System</td>
<td>An organized collection of Element instances</td>
</tr>
<tr>
<td>3 Entity</td>
<td>Individual Element of system which can act as a provider or consumer</td>
</tr>
<tr>
<td>4 Actor</td>
<td>Person or system component that interacts with the system.</td>
</tr>
<tr>
<td>5 Service</td>
<td>Logical representation of activities set with specified outcomes</td>
</tr>
<tr>
<td>6 Atomic Service</td>
<td>A self-contained service</td>
</tr>
<tr>
<td>7 Composite Service</td>
<td>An assembling of other services to provide a higher-level service.</td>
</tr>
<tr>
<td>8 Architecture</td>
<td>Fundamental concepts of a system (elements, relationships, design principles)</td>
</tr>
<tr>
<td>9 Endpoint</td>
<td>Location at which information is received to invoke and configure interaction</td>
</tr>
<tr>
<td>9 Capability</td>
<td>An ability possessed by an Element as a basis for service offering</td>
</tr>
<tr>
<td>10 Interface</td>
<td>Shared boundary between two units (functions, interconnections and exchanges)</td>
</tr>
<tr>
<td>11 Service interface</td>
<td>Interface by which other elements can interact and exchange information</td>
</tr>
<tr>
<td>12 Service Contract</td>
<td>Terms, conditions, and rules that interacting participants binded to agree on.</td>
</tr>
<tr>
<td>13 Role</td>
<td>An activity set assignable to an entity for a specific function in an interaction.</td>
</tr>
<tr>
<td>14 Information Type</td>
<td>Information provided to or receive from upon service usage</td>
</tr>
<tr>
<td>15 Task</td>
<td>Atomic action which accomplishes a defined result</td>
</tr>
</tbody>
</table>

The standard formally defined 61 concepts. A selected subset of the components is listed in table 3.3. An inconsistency is found in the standard. The definition part is not fully correlated with the provided ontology structure. The defined ontology only covers 14 concepts (figure 3.11) without mentioned detail on the other concepts.

The ontology is quite simple, and its coverage is superficial if compared to SoaML ontology. The coverage is lacking in a detailed level of (software) service abstraction, i.e. port, operation, role, and interaction protocol. The ontology is also missing some of the key concepts defined in the terminology part, i.e. Capability, Service Architecture, and Role.
Despite these differences, correlation with SoaML ontology occurs in: (1) Service Interface, (2) Service Contract, (3) Information Type, and (4) Element (Participant in SoaML). The relationships are also consistent:

- **Element** perform **Service** (via Port and Role in SoaML)
- **Service** has **Service Contract** and **Service Interface** as service description
- **Service Interface** has Information Type as attribute (via Operation and Parameter in SoaML)

It can be observed that the previously produced SoaML ontology (figure 3.10) is relatively consistent with the other software service ontologies. It is also a superior ontology compared to the ISO's ontology in coverage and level of detail. The produced SoaML ontology is therefore established as an intermediary for a true Software Service Ontology.
3.4 General Service Ontology

Similar to the fact that SoaML and UML are only used within the context of designing a software system, SOA is rarely discussed beyond the context of software architecture. Even though the adoption of service-oriented concepts is not apparent beyond the technological sphere, its conception always strives to provide a generalized abstraction covering both IT-based and non-IT-based systems.

While in reality, the distinction for IT and non-IT context of a service system needs to be made, the generality feature of SOA conceptions is useful to build the General Service Ontology. The objective is therefore to generalize and enlarge the coverage of the Software Service Ontology. The generalization is achieved by applying the concept from a software-service context to the general context of a service system, which includes the physical and the manual system. The practical target is to cover concepts included in the classic service engineering context but missing in informatics service engineering, such as (1) ‘value’, (2) ‘business process’, (3) ‘business model’ and (4) ‘capability’. These four concepts are the target components to be integrated with concepts already covered within the Software Service Ontology.

3.4.1 Ontology Source Material

Inspired by the existence of ISO/IEC 18384:2016, the general ontology is built based on the available standard documents published by the
International Organization for Standardization (ISO) and Object Management Group (OMG). Sixteen standard documents were identified to cover the definition of targeted concepts. The identified documents originated from both business and technical domain, including IT domain. The ISO/IEC 18384:2016 (SOA Reference Architecture) is re-used here for its general feature. Table 3.4 lists the source document for building the consolidated ontology.

Table 3.4: Identified Standard Documents for Concepts Definition

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 ISO 2382</td>
<td>Information Technology (IT) Vocabulary</td>
<td>ITV</td>
</tr>
<tr>
<td>2 ISO 9000</td>
<td>Quality Management Systems</td>
<td>QMS</td>
</tr>
<tr>
<td>3 ISO 14662</td>
<td>Open Electronic Data Interchange (EDI)</td>
<td>EDI</td>
</tr>
<tr>
<td>4 ISO 15288</td>
<td>System Life Cycle</td>
<td>SYLC</td>
</tr>
<tr>
<td>5 ISO 15944</td>
<td>Business Operational View</td>
<td>BOV</td>
</tr>
<tr>
<td>6 ISO 16500</td>
<td>Digital Audio-Visual (AVI) System</td>
<td>DAVI</td>
</tr>
<tr>
<td>7 ISO 12207</td>
<td>Software Life Cycle</td>
<td>SWLC</td>
</tr>
<tr>
<td>8 ISO 18384</td>
<td>Service Oriented Architecture (SOA) Ref. Architecture</td>
<td>SOA</td>
</tr>
<tr>
<td>9 ISO 19505</td>
<td>Unified Modeling Language (OMG’s UML)</td>
<td>UML</td>
</tr>
<tr>
<td>10 ISO 19510</td>
<td>Business Process Model &amp; Notation (OMG’S BPMN)</td>
<td>BPMN</td>
</tr>
<tr>
<td>11 ISO 30102</td>
<td>Distributed Application Platforms and Services</td>
<td>DAPS</td>
</tr>
<tr>
<td>12 ISO 90003</td>
<td>Software Engineering</td>
<td>SWE</td>
</tr>
<tr>
<td>13 ISO 14813</td>
<td>Intelligent Transport System</td>
<td>ITS</td>
</tr>
<tr>
<td>14 OMG - VDML</td>
<td>Value Definition Modeling Language</td>
<td>VDML</td>
</tr>
<tr>
<td>15 OMG - BMM</td>
<td>Business Motivation Model</td>
<td>BMM</td>
</tr>
<tr>
<td>16 OMG - SoaML</td>
<td>Service Oriented Architecture Modeling Language</td>
<td>SoaML</td>
</tr>
</tbody>
</table>

Some of the source documents contain a partial ontological view of concepts covered in the document, i.e. ISO 18384 (SOA-RA), ISO 19505 (OMG-UML), ISO 19510 (OMG-BPMN) and OMG-BMM. In these cases, the targeted concept definition is extracted, along with the available defined relation between them. For other documents, only the concept definitions were extracted. If available, the definitions were captured.
from the formal terminology definition section. In other cases, implied definition is extracted from the descriptive narration.

A total of 74 concept definitions are identified. These concepts are arranged and grouped based on similarity (table 3.5 to table 3.21). Concepts observed to be covering similar idea are later merged into a representative label.

### Table 3.5: Entity-related Concept Definitions

<table>
<thead>
<tr>
<th>id</th>
<th>Concept</th>
<th>Description</th>
<th>Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Element</td>
<td>Unit at a given level of abstraction and with a clearly defined boundary</td>
<td>[SOA] [DAPS]</td>
</tr>
<tr>
<td>2</td>
<td>Participant</td>
<td>An entity or a role that controls or is responsible for a business process.</td>
<td>[BPMN]</td>
</tr>
<tr>
<td>3</td>
<td>Participant</td>
<td>Anything that fill a role in collaboration, e.g. actors, collaborations or roles</td>
<td>[VDML]</td>
</tr>
<tr>
<td>4</td>
<td>Participant</td>
<td>A party or component that provides and/or consumes services</td>
<td>[SOAML]</td>
</tr>
<tr>
<td>5</td>
<td>Actor</td>
<td>An indivisible participant</td>
<td>[VDML]</td>
</tr>
<tr>
<td>6</td>
<td>Actor</td>
<td>Entity that fulfils a role</td>
<td>[ITV] [ITS]</td>
</tr>
<tr>
<td>7</td>
<td>Actor</td>
<td>Specification of an interactive role played by a user or other system</td>
<td>[UML]</td>
</tr>
<tr>
<td>8</td>
<td>Actor</td>
<td>A person or system component who interacts with the system as a whole and who provides stimulus which invoke action</td>
<td>[DAVI] [DAPS]</td>
</tr>
<tr>
<td>9</td>
<td>Actor</td>
<td>Person or system component that interacts with the system as a whole and provides stimulus which invokes actions</td>
<td>[SOA]</td>
</tr>
<tr>
<td>10</td>
<td>Entity</td>
<td>Anything that exists, did exist, or might exist, including associations of things</td>
<td>[BOV] [ITV]</td>
</tr>
<tr>
<td>11</td>
<td>Entity</td>
<td>System’s element in which can act as a service provider or consumer.</td>
<td>[SOA]</td>
</tr>
<tr>
<td>12</td>
<td>Party</td>
<td>Organization entering into an agreement</td>
<td>[SYLC]</td>
</tr>
</tbody>
</table>

### Table 3.6: Process-related Concept Definitions

<table>
<thead>
<tr>
<th>id</th>
<th>Concept</th>
<th>Description</th>
<th>Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>13</td>
<td>Business</td>
<td>A series of information exchange processes, involving more than one person, directed towards mutually agreed goal, over a period of time</td>
<td>[EDI], [BOV]</td>
</tr>
<tr>
<td>14</td>
<td>Business process</td>
<td>A container of processing steps, sequences, structure, interactions, and connection to events that trigger the processes.</td>
<td>[BMM]</td>
</tr>
<tr>
<td>15</td>
<td>Business process</td>
<td>A defined set of activities as steps to achieve an objective, includes the flow and use of information and resources.</td>
<td>[BPMN]</td>
</tr>
<tr>
<td>16</td>
<td>Business process</td>
<td>A process that is designed to deliver outputs that satisfy business objectives.</td>
<td>[QMS]</td>
</tr>
<tr>
<td>17</td>
<td>Process</td>
<td>A set of interrelated or interacting activities that use resources to produce a result by transforming inputs into outputs.</td>
<td>[QMS] [SYLC]</td>
</tr>
<tr>
<td>18</td>
<td>Process</td>
<td>Type of composition whose elements are composed into a sequence or flow of activities and interactions with the objective of carrying out certain work</td>
<td>[SOA] [DAPS]</td>
</tr>
<tr>
<td>19</td>
<td>Process</td>
<td>A sequence or flow of activities in an organization with the objective of carrying out work</td>
<td>[BPMN] [VDML]</td>
</tr>
<tr>
<td>20</td>
<td>Process</td>
<td>Series of actions or events taking place in a defined manner leading to the accomplishment of an expected result</td>
<td>[BOV]</td>
</tr>
</tbody>
</table>
### Table 3.7: Activity (sub-process)-related Concept Definitions

<table>
<thead>
<tr>
<th>id</th>
<th>Concept</th>
<th>Description</th>
<th>Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>21</td>
<td>Sub-process</td>
<td>A compound activity that is included within a process or choreography can be broken down into a finer level of detail through a set of sub-activities.</td>
<td>BPMN</td>
</tr>
<tr>
<td>22</td>
<td>Activity</td>
<td>A work performed in a process, atomic (sub-process) or non-atomic (task).</td>
<td>BPMN</td>
</tr>
<tr>
<td>23</td>
<td>Activity</td>
<td>A collection of related (cohesive) tasks of a process</td>
<td>[WARC] [SWM]</td>
</tr>
<tr>
<td>24</td>
<td>Activity</td>
<td>Output producing work element required by a process, as tasks or operations.</td>
<td>OMS</td>
</tr>
<tr>
<td>25</td>
<td>Activity</td>
<td>Sequence and conditions for coordinating lower level behaviours.</td>
<td>UML</td>
</tr>
</tbody>
</table>

### Table 3.8: Task (atomic-activity)-related Concept Definitions

<table>
<thead>
<tr>
<th>id</th>
<th>Concept</th>
<th>Description</th>
<th>Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>26</td>
<td>Task</td>
<td>An atomic activity that is included within a process</td>
<td>BPMN</td>
</tr>
<tr>
<td>27</td>
<td>Task</td>
<td>Atomic action which accomplishes a defined result.</td>
<td>SOA</td>
</tr>
<tr>
<td>28</td>
<td>Behaviour</td>
<td>A respond specification of a classifier instance toward a request.</td>
<td>UML</td>
</tr>
<tr>
<td>29</td>
<td>Action</td>
<td>A fundamental unit of behaviour specification.</td>
<td>UML</td>
</tr>
<tr>
<td>30</td>
<td>Action</td>
<td>Activity to achieve something</td>
<td>QMS</td>
</tr>
</tbody>
</table>

### Table 3.9: Collaboration-related Concept Definitions

<table>
<thead>
<tr>
<th>id</th>
<th>Concept</th>
<th>Description</th>
<th>Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>Business Network</td>
<td>A collaboration between independent entities in an economic exchange.</td>
<td>VDML</td>
</tr>
<tr>
<td>32</td>
<td>Business transaction</td>
<td>Predefined set of activities initiated to accomplish a shared goal and terminated upon agreed conclusions by all involved</td>
<td>BOV [EDI]</td>
</tr>
<tr>
<td>33</td>
<td>Transaction</td>
<td>A sub-process that represents a set of coordinated activities of independent systems in accordance with a contractual relationship.</td>
<td>BPMN</td>
</tr>
<tr>
<td>34</td>
<td>Collaboration</td>
<td>The act of sending messages between two participants in a process.</td>
<td>BPMN</td>
</tr>
<tr>
<td>35</td>
<td>Collaboration</td>
<td>Collection of participants joined together for a shared purpose or interest</td>
<td>VDML</td>
</tr>
<tr>
<td>36</td>
<td>Collaboration</td>
<td>A composition whose elements interact in a non-directed fashion, according to their own purposes without a predefined pattern of behaviour</td>
<td>SOA [DAPS]</td>
</tr>
</tbody>
</table>

### Table 3.10: Choreography-related Concept Definitions

<table>
<thead>
<tr>
<th>id</th>
<th>Concept</th>
<th>Description</th>
<th>Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>37</td>
<td>Choreography</td>
<td>A composition with a non-directed interaction following a predefined pattern of behaviour for the entire composition</td>
<td>SOA [DAPS]</td>
</tr>
<tr>
<td>38</td>
<td>Choreography</td>
<td>Interaction scenario with roles rules and information bundles of that scenario, in a predefined pattern and non-directed fashion for the entire instantiation</td>
<td>BOV [DAPS]</td>
</tr>
<tr>
<td>39</td>
<td>Choreography</td>
<td>An ordered sequence of business-to-business (B2B) message exchanges between participants.</td>
<td>BPMN</td>
</tr>
<tr>
<td>40</td>
<td>Protocol</td>
<td>Set of message formats (semantic, syntactic, and symbolic rules) and the rules for message exchange between peer layer entities</td>
<td>DAVI</td>
</tr>
<tr>
<td>41</td>
<td>Protocol</td>
<td>Set of rules that determines the behaviour of objects in achieving communication and exchanging messages</td>
<td>ITV</td>
</tr>
</tbody>
</table>
### Table 3.11: Interaction-point-related Concept Definitions

<table>
<thead>
<tr>
<th>id</th>
<th>Concept</th>
<th>Description</th>
<th>Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>42</td>
<td>End point</td>
<td>Extension point of an element as a service address</td>
<td>BPMN</td>
</tr>
<tr>
<td>43</td>
<td>Endpoint</td>
<td>Location at which information is received (or sent) to invoke an interaction</td>
<td>SOA</td>
</tr>
<tr>
<td>44</td>
<td>End node</td>
<td>Endpoint node at the end of only one branch</td>
<td>ITV</td>
</tr>
<tr>
<td>45</td>
<td>Port</td>
<td>Participant's interaction point for a service provision or consummation.</td>
<td>SOAAML</td>
</tr>
<tr>
<td>46</td>
<td>Port</td>
<td>Unit through which data can enter or leave a network</td>
<td>ITV</td>
</tr>
<tr>
<td>47</td>
<td>Port</td>
<td>An abstraction of destinations identity associated with particular applications</td>
<td>DAVI</td>
</tr>
<tr>
<td>48</td>
<td>Reference point</td>
<td>A set of interfaces between any two blocks through which information flows</td>
<td>DAVI</td>
</tr>
<tr>
<td>49</td>
<td>Channel</td>
<td>A connection conveying information between blocks or with the environment.</td>
<td>DAVI</td>
</tr>
<tr>
<td>50</td>
<td>Channel</td>
<td>Mechanism to execute a deliverable flow</td>
<td>VDML</td>
</tr>
<tr>
<td>51</td>
<td>Channel</td>
<td>A component of a communication system that connects the source with target</td>
<td>ITV</td>
</tr>
</tbody>
</table>

### Table 3.12: Interface-related Concept Definitions

<table>
<thead>
<tr>
<th>id</th>
<th>Concept</th>
<th>Description</th>
<th>Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>52</td>
<td>Interface</td>
<td>A set of operations that are implemented by services.</td>
<td>BPMN</td>
</tr>
<tr>
<td>53</td>
<td>Interface</td>
<td>Shared boundary between two functional units, defined by various characteristics pertaining to the functions, physical interconnections, signal exchanges</td>
<td>SOA, ITV</td>
</tr>
<tr>
<td>54</td>
<td>Interface</td>
<td>A point of demarcation between two blocks through which information flows</td>
<td>DAVI</td>
</tr>
<tr>
<td>55</td>
<td>Interface</td>
<td>A declaration of public features and obligations set to be fulfilled by another classifier.</td>
<td>UML</td>
</tr>
</tbody>
</table>

### Table 3.13: Operation-related Concept Definitions

<table>
<thead>
<tr>
<th>id</th>
<th>Concept</th>
<th>Description</th>
<th>Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>56</td>
<td>Operation</td>
<td>A definition of messages that are consumed and, optionally, produced when the operation is called.</td>
<td>BPMN</td>
</tr>
<tr>
<td>57</td>
<td>Operation</td>
<td>A behavioural feature of a classifier that specifies the name, type, parameters, and constraints for invoking an associated behaviour.</td>
<td>UML</td>
</tr>
</tbody>
</table>

### Table 3.14: Operation-related Concept Definitions

<table>
<thead>
<tr>
<th>id</th>
<th>Concept</th>
<th>Description</th>
<th>Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>58</td>
<td>Information bundle</td>
<td>Formal description of the semantics of the recorded information to be exchanged by parties playing roles in a scenario.</td>
<td>BOV, EDI</td>
</tr>
<tr>
<td>59</td>
<td>Message</td>
<td>The contents of a communication between two participants (as defined by a business role or a business entity)</td>
<td>BPMN</td>
</tr>
</tbody>
</table>

### Table 3.15: Service-related Concept Definitions

<table>
<thead>
<tr>
<th>id</th>
<th>Concept</th>
<th>Description</th>
<th>Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>60</td>
<td>Service</td>
<td>Logical representation of a set of 'activities' that has specified outcomes, self-contained, and is a &quot;black box&quot; to consumers of the service</td>
<td>SOA</td>
</tr>
<tr>
<td>61</td>
<td>Service</td>
<td>Output of an organization with at least one activity necessarily performed between the organization and the customer.</td>
<td>QMS</td>
</tr>
<tr>
<td>62</td>
<td>Service</td>
<td>A mechanism to enable access to one or more capabilities, accessible through an interface with constraints and policies as specified by the service description</td>
<td>VDML</td>
</tr>
<tr>
<td>63</td>
<td>Service</td>
<td>Capability of a layer provided to the entities of the next higher layer</td>
<td>ITV</td>
</tr>
<tr>
<td>64</td>
<td>Service</td>
<td>A set of elementary streams offered to the user as a program.</td>
<td>DAVI</td>
</tr>
</tbody>
</table>
Table 3.16: *Capability*-related Concept Definitions

<table>
<thead>
<tr>
<th>id</th>
<th>Concept</th>
<th>Description</th>
<th>Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>65</td>
<td>Capability</td>
<td>Ability of an entity to realize something that will fulfil the requirements</td>
<td>[QMS]</td>
</tr>
<tr>
<td>66</td>
<td>Capability</td>
<td>Ability to perform a particular kind of work and deliver desired value</td>
<td>[VDML]</td>
</tr>
<tr>
<td>67</td>
<td>Capability</td>
<td>Ability to act and produce an outcome that achieves a result.</td>
<td>[SOAML]</td>
</tr>
</tbody>
</table>

Table 3.17: *Business Model*-related Concept Definitions

<table>
<thead>
<tr>
<th>id</th>
<th>Concept</th>
<th>Description</th>
<th>Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>68</td>
<td>Business plan.</td>
<td>Provisions made to fulfil mission, and vision and applying values in terms of the strategy, objectives, measures, targets and enabling processes.</td>
<td>[QMS]</td>
</tr>
<tr>
<td>69</td>
<td>Business model</td>
<td>A description of value creation, delivery, and capturing to fulfil the motivations</td>
<td>[VDML]</td>
</tr>
</tbody>
</table>

Table 3.18: *Rule*-related Concept Definitions

<table>
<thead>
<tr>
<th>id</th>
<th>Concept</th>
<th>Description</th>
<th>Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>70</td>
<td>Business rules</td>
<td>A structured, discrete, specific, and practicable guidance for business process in the form of obligation or necessity.</td>
<td>[BMM]</td>
</tr>
</tbody>
</table>

Table 3.19: *Contract*-related Concept Definitions

<table>
<thead>
<tr>
<th>id</th>
<th>Concept</th>
<th>Description</th>
<th>Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>71</td>
<td>Constraint</td>
<td>Rule, explicitly stated, that prescribes, limits, governs or specifies a transaction</td>
<td>[BOV]</td>
</tr>
<tr>
<td>72</td>
<td>Contract</td>
<td>A binding agreement</td>
<td>[QMS]</td>
</tr>
</tbody>
</table>

Table 3.20: *Value*-related Concept Definitions

<table>
<thead>
<tr>
<th>id</th>
<th>Concept</th>
<th>Description</th>
<th>Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>73</td>
<td>Value</td>
<td>A measurable factor of benefit to a recipient, in association with a business item</td>
<td>[VDML]</td>
</tr>
<tr>
<td>74</td>
<td>Value stream</td>
<td>Flow of material and information service from customer to suppliers.</td>
<td>[QMS]</td>
</tr>
</tbody>
</table>

Table 3.21: *Role*-related Concept Definitions

<table>
<thead>
<tr>
<th>id</th>
<th>Concept</th>
<th>Description</th>
<th>Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>75</td>
<td>Role</td>
<td>An expected behaviour pattern associated with participation in a collaboration</td>
<td>[VDML]</td>
</tr>
<tr>
<td>76</td>
<td>Role</td>
<td>A defined set of activities assigned to an entity to performs a specific function</td>
<td>[SOA]</td>
</tr>
<tr>
<td>77</td>
<td>Role</td>
<td>External intended behaviour specification within an interaction scenario</td>
<td>[BOV][EDI]</td>
</tr>
<tr>
<td>78</td>
<td>Thematic role</td>
<td>A function of that an entity may perform during the execution of a script.</td>
<td>[ITV]</td>
</tr>
</tbody>
</table>

Some of the captured definitions may seem a little specific or out-of-context, but they provide peripheral definition which is necessary within the goal of generalization.
3.4.2 Consolidated Ontology

Table 3.22 lists the merged concepts into a hierarchy of 17 concepts as the components of the ontology.

### Table 3.22: Hierarchy of 17 Merged Concept Definitions

<table>
<thead>
<tr>
<th>Concept</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Entity</td>
<td>Individual element in a system which can act as a service provider or consumer.</td>
</tr>
<tr>
<td>1a. Interaction point</td>
<td>Location at which information is received (or sent) to invoke an interaction</td>
</tr>
<tr>
<td>2. Business model</td>
<td>A description of value creation, delivery, and capturing to fulfill the motivations</td>
</tr>
<tr>
<td>3. Process</td>
<td>A set of business activities as steps to achieve a business objective</td>
</tr>
<tr>
<td>3a. Activity</td>
<td>A collection of related (cohesive) tasks of a process</td>
</tr>
<tr>
<td>3b. Task</td>
<td>An atomic activity that is included within a process, accomplishes a defined result.</td>
</tr>
<tr>
<td>3c. Rule</td>
<td>A structured, discrete, specific, and practicable guidance for business process</td>
</tr>
<tr>
<td>4. Service</td>
<td>A set of a capability’s activities accessible through an interface with constraints.</td>
</tr>
<tr>
<td>4a. Interface</td>
<td>Shared boundary between two units, characterized by operations</td>
</tr>
<tr>
<td>4b. Operation</td>
<td>A definition of messages consumed and, optionally, produced when called.</td>
</tr>
<tr>
<td>5. Capability</td>
<td>Participant ability to act and produce an outcome</td>
</tr>
<tr>
<td>6. Value</td>
<td>A measurable factor of benefit to a recipient, in association with a business item</td>
</tr>
<tr>
<td>7. Collaboration</td>
<td>Predefined set of activities and/or processes initiated to accomplish a shared goal</td>
</tr>
<tr>
<td>7a. Role</td>
<td>A defined set of activities assigned to an entity to perform a specific function</td>
</tr>
<tr>
<td>7b. Choreography</td>
<td>An ordered sequence of message exchanges between two or more entities</td>
</tr>
<tr>
<td>7c. Contract</td>
<td>Explicitly stated rule, that prescribes, limits, governs or specifies transaction</td>
</tr>
<tr>
<td>7d. Message</td>
<td>The contents of a communication between two participants</td>
</tr>
</tbody>
</table>

Figure 3.12: General Service Engineering Ontology

69
The ontological relations between each merged concept are visualized in the figure 3.12. For ease of reference, the numbering in the figure is correlated with the number in table 3.22.

As defined in the table, *Service* is a container for *Interface* and its *Operation*. In the visualization, these service components are also aggregated with the underlying process component, i.e. *Activity* and *Task*, to form a larger abstraction of *Service*, between the front-end interface and the back-end supporting activities. This also reflects a SoaML perspective regarding the relation between ‘process’ and ‘service’ as different views of a similar object. ‘Process’ view focuses on the how and why of the interaction, while ‘service’ focuses on participant activities in provision and consumption of services [60].

A shaded container is introduced in the visualization to define the ‘Entity’ ownership boundary. Three components protrude beyond the boundary: (1) *Collaboration*, as an abstraction of atomic or composite interaction, (2) *Choreography*, where the arrangement of interaction sequence is defined as a ‘contract’ agreement with an outside entity, and (3) *Message*, which is exchanged with parties beyond the boundary.

A pair of concepts is merged in the ontology visual: *Choreography* (7b) with *Contract* (7c). This merger is not only implemented for visual simplification, but also to show the strong intersection between the two as an interaction arrangement. To be precise, *choreography* refers to the sequencing aspect, while *contract* refers to the rule and constraint.
3.4.3 Differentiating Software from General Context

As can be observed in the previous section, the produced general ontology (figure 3.12) captures concepts from a software-service context (section 3.3) with concepts from classic service engineering, i.e. (1) ‘value’, (2) ‘business process’, (3) ‘business model’ and (4) ‘capability’. In practical engineering level, different treatments are performed for each IT-context and non-IT context. This section explores the abstraction separation between a software-service and a non-software service.

The type of ‘service encounter’ is the differentiating factor between software and non-software service. Two types of service encounters are defined in [23]: (1) Physical, and (2) Virtual. If an interaction of an encounter is mediated by technical devices, the encounter is categorized as a virtual encounter. In this typology, technology may facilitate the encounter in various forms, e.g. from e-mail to website. The software-services context resides in a specific situation, in which a software component offers service consumables by other software components (example 5 and 7 in table 3.23).

<table>
<thead>
<tr>
<th>Type of Encounter</th>
<th>Direct (end-customer facing)</th>
<th>Indirect (external interface usage)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Physical</td>
<td>(1) Service Counter (Consumer Visit) (2) Provider visitation</td>
<td>(3) B2B service flow through shared physical collaboration space</td>
</tr>
</tbody>
</table>
In a more detailed fashion, figure 3.13 illustrates five archetypes of customer contact in relation to technology is defined [3][4]:

1. **Technology-free**, a face-to-face service encounter without direct technology involvement in service provision

2. **Technology-assisted**, face-to-face encounter is facilitated by technology without consumer involvement in technology access.

3. **Technology-facilitated**, a face-to-face encounter where both provider and consumer access the same technology.

4. **Technology-mediated**, provider and consumer are not physically connected and technology is employed to enable communication.

5. **Technology-generated**, provider is represented by a technological component to form a degree of a self-service encounter.
While software-services may exist to facilitate the last four archetypes (2 to 5), the software-service context is mostly apparent in the last two (4 and 5), technology mediated, and technology generated, where no physical encounter is established between the provider and consumer. In these two types of encounter, the interaction is performed over a network, e.g. Internet, facilitated by software components, e.g. websites, and software-service.

### 3.4.4 Consolidated Software Ontology

As intended, the produced General Service Engineering Ontology (figure 3.12) covers the additionally targeted business concept. To refocus on the software context, the ontology is trimmed to only include software related concepts. By excluding the three business analysis level components, i.e. (1) values, (2) capability, and (3) business model, the 17 components in the general context are reduced to 14 components for software-service context, as listed in table 3.24.

To achieve a uniformity and as a form of triangulation, SoaML ontology is also juxtaposed (in table 3.24) and adapted with the structure of the consolidated software-service ontology into the resulting ontology as presented in figure 3.14.
Table 3.24: Concepts for Software Service Context

<table>
<thead>
<tr>
<th>General Context (table 3.22)</th>
<th>Software-service context</th>
<th>SoaML label (table 3.2)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Entity</td>
<td>1. Software component</td>
<td>1. Participant</td>
</tr>
<tr>
<td>1a. Interaction point</td>
<td>1a. Port</td>
<td>2. Port</td>
</tr>
<tr>
<td>3. Process</td>
<td>3. Software service*</td>
<td>-</td>
</tr>
<tr>
<td>3b. Task</td>
<td>3b. Atomic service</td>
<td>4. (UML) Interface</td>
</tr>
<tr>
<td>3c. Rule</td>
<td>3c. Service contract**</td>
<td>5. Service Contract†</td>
</tr>
<tr>
<td>4. Service</td>
<td>4. Software service*</td>
<td>-</td>
</tr>
<tr>
<td>4a. Interface</td>
<td>4a. Service interface</td>
<td>3. (UML) Interface</td>
</tr>
<tr>
<td>7a. Role</td>
<td>7a. Component role</td>
<td>7. Role</td>
</tr>
<tr>
<td>7c. Contract</td>
<td>7c. Service contract**</td>
<td>5. Service Contract†</td>
</tr>
<tr>
<td>7d. Message</td>
<td>7d. Message</td>
<td>10. Message type</td>
</tr>
</tbody>
</table>

Merged concepts are marked with pairs of * † ‡ symbols

![Consolidated Software Service Ontology](image_url)

Figure 3.14: Consolidated Software Service Ontology

In this software service context, the term ‘service’ represents an abstraction of externally accessible software components. It therefore
covers both the underlying software component behaviour (3a and 3b) originated from the ‘process’ context, and the related published description (4a and 4b). Similarly, the term ‘service contract’ merges the two aspects of: internal process rule (3c), with the externally shared and agreed rule (7c).

These characteristics are derived from SoaML’s feature in superimposing a service component with its description. This merging is also coherent with SoaML perspective that treats ‘process’ and ‘services’ as different views of the same object [60].

Two additional SoaML components are not represented in this context: (1) Capability, and (2) Service Architecture. The two is decidedly to reside in the business analysis level of service engineering.

Components decoupling and its binding mechanism is an important principal in SOA conception [24]. It is also the underlying motive in introducing SoaML over UML limitation [63]. As an exercise, these concepts are narrated in the context of the produced software service ontology diagram in figure 3.14.

A service decoupling is implemented as a separation between a published service description (component 4a and 4b) and its underlying supporting behaviours (component 3a and 3b). Service behaviours (component 3) are actually a part of a specific software component (component 1).

Binding, or more precisely role-binding, is an execution time instance when a software component (component 1) assumes a role
(component 7a) within a context of specific software interaction (component 7), using the service interface (component 4a) as the guidance in invoking its internal behaviours (component 3a and 3b), via its defined port (component 1a) as the location address, for message (component 7d) passing operations (component 4b).

The ontology visualization structure is not only describing a service providing software component. In a case where a component requires services from other component within its own composite behaviour (component 3a), it follows the previously described role binding mechanism. The difference is that the software component (component 1) assumes ‘consumer’ role (component 7a) and adheres to a collaboration mechanism (component 7), which is implemented by service (component 3a and 3b) from the providing software component.

These conceptual exercises for decoupling, binding and service consumption demonstrate the capability of the ontology in covering the basic SOA concepts for the context of software service engineering.

3.5 Patterns of Service System

As an attempt to assess the feasibility of the ontology, various patterns of a service system are applied to the ontology. The patterns of service system are implied and analogous with the three abstraction level of service modelling: (1) single interaction, (2) choreography, and (3) orchestration [16]. These exercises can be seen as a proto-
operationalization of the metamodel ontology toward domain ontology (figure 3.1)

Figure 3.15 visualizes the first pattern for a simple interaction between a service provider entity and a service user, e.g. individual end-user consumer. Here, the whole ontology set is positioned as the service-providing entity. To illustrate the first pattern of interaction, the ontology is paired with a simple consumer outside the entity boundary.

Among others, the resulting models covers the concepts of: (1) capability offered by the service provider, (2) value offered and requested by the consumer, (3) value brought by the consumer (e.g. in the form of monetary asset), (4) choreographed activity between the two entities, and value exchanged during the transaction.

In the second pattern, choreography level of abstraction, a service is modelled as a complex process with multiple interactions between two
entities. A business-to-business (B2B) arrangement between a company and its supplier is an example of this pattern. Figure 3.16 illustrates this pattern by pairing two ontology sets as two interacting entities.

![Diagram of B2B service pattern](Image)

Figure 3.16: Model of a B2B service pattern

The resulting model visualizes pairs of external behaviour requested and offered by each participant in the pattern. This pattern specifies and analyses interoperability between two service participants. The model structure also introduces the concept of ‘collaboration space’ in which the interactions take place. It may reside (i.e. owned) within one of the participant boundary, or in independent third-party location. In the software-service context, the ‘collaboration space’ relates to the operator and controller of software-service repository, i.e. service registry and service publication.
In the third pattern, orchestration abstraction, an offered service is modelled as a composition of other services. Figure 3.17 illustrates this pattern by combining the first and second pattern approaches with the introduction of both a simple customer, and a partnering service co-provider.

This pattern is related with indirect type of service encounter in the typology of service encounter [23], where an external party is involved in the service process, as co-provider or intermediary, and may make a direct contact to the service-consumer. In a more complex pattern, multiple co-providers may forms service architecture over a set of services. Consequently, this model can be used to analyse and specify possible implementation of the offered service.
The model also raises the issue of ‘collaboration spaces’. It relates to the existence of a service coordinator, with the central role of interacting and orchestrating other providers. While the arrangement can be made to be in equal term (distributed and federated), each particular of collaboration tends to require a dominant participant role as the main operator.

Other types of service system patterns and combinations may exist, related to elaboration of provider role and components of collaboration space, but the three illustrated patterns adequately demonstrate the feasibility and flexibility of the produced ontology in covering various types of service system.

In summary, this chapter has demonstrated the process of foundational ontology building in the context of ‘service engineering’. The produced ontologies are an integral part of the preliminary framework to be discussed in the next chapter, in defining the context, scope, and terminologies definition within the framework. As a form of verification in the fifth chapter, a concept mapping is performed between the ontologies, from this chapter, and the framework of the next chapter.
Chapter 4  
Service Engineering Framework

4.1 Introduction

This chapter addresses the second and third research question, i.e. the ‘activity’ and ‘artefact’ of a service engineering, in the form of service engineering. Several iterations of the framework were produced throughout this research scope. Figure 4.1 describes the content of this chapter in terms of the flow of the research.

Figure 4.1: Research Flow and Content Map (Chapter 4)
An engineering process can be viewed as a transformative activity bridging an analysis perspective into an architectural perspective, in which functional and behavioural views are adapted into structural views of functional components. In the case of software engineering, the initial definition UML’s use cases are to be implemented as (software) objects with certain hierarchy and dependencies [81].

A similar principal is also applicable in the service engineering context, where business directives and process flow definitions are to be implemented as an architecture of service components. In this regard, service engineering tends to have a greater scope than a regular software engineering, where the business side is not always a given parameter but rather a modifiable context to be re-assessed and re-designed during the process [66].

This perspective is shared in service engineering literature [10][35][31][32][13]. For example, in a bigger context of a service-oriented system, service engineering is associated with process engineering, market engineering and ontology engineering as illustrated in figure

Figure 4.2: Integrated Methodology for Service-Oriented System[82]
Consequently, business and process analysis are often performed within service engineering activities.

The service engineering framework targeted in this research is emphasized in the IT context, therefore limiting its concern in business side analysis and manual services. An abstraction for business and process analysis is provided but their elaboration is considered to be out of scope for this research context.

An important abstraction representing business side analysis and design in service engineering is the concept of the ‘business model’, as an intermediary layer between the ‘strategic layer’ and ‘process layer’[83]. The development of a new service, i.e. service engineering, is essentially a design process of anew or improved business model [84]. In fact, a business model can be considered as the key artefact to produce during business side analysis, as an input for the subsequence service design stage[27][75].

The business side analysis is commonly initiated with activities relating to business idea generation and assessment[27], in parallel with business Strengths, Weaknesses, Opportunities, and Threats (SWOT) analysis[7]. The selected business concept is then elaborated into a business case description which covers the reasoning from market analysis, customer requirement analysis, and financial reflection, which are integrated into a business model.

As a dominant format in business modelling, Business Model Canvas (BMC) [85] is an ubiquitous format for representing a business
case. BMC covers all core aspects of a business model within a simple visualization. Therefore, the format facilitates a holistic view and allows an easy comparison between different business models [84].

The actual service design stage should follow the business-side analysis and design activity [11][75]. In this stage, the service concept is elaborated by detailing the defined business model. Aspects covered in this stage are: service description, service processes, service resources (employees, facilities), customer benefit (tangible and intangible), and customer engagement from a marketing aspect [27].

Two fundamental aspects of service design activity are (1) the process model and (2) customer engagement[86][87]. The process model identifies and describes all the activities required to fulfil a service, while the customer engagement concept elaborates the manner of interaction between the service consumer and the providing participants.

These two aspects are actually overlapping in describing a service process. The difference is that the second aspect, customer engagement, is a higher abstraction of the process with the focus only on the detail of the interaction, while the process model elaborates all the activity involved in a service.

The classic service engineering literatures tends to focus only on the second aspect, while the informatics service engineering emphasizes on the first aspect. A specific focus on the series of interactions is beneficial for a detailed analysis from a consumer perspective, e.g. interaction mode
and facilitation, which is often neglected in the overall process model description.

In the context of a service process, the Service Blueprinting [88] format is often suggested in classic service engineering literature [89][6][13]. On the other hand, the artefacts for process model commonly use the dominant BPMN metamodel format [90][91][40][92]. By combining both the interaction and process aspects, a comprehensive analysis can be expected which equally covers all perspective of an involved participant.

Following the service design stage, the subsequent stage further elaborates the service design for the use in the implementation stage. In case of identified potential use of software interactivities, the design activity is shifted into the IT perspective in designing software-services. The process is a special case of software-engineering with service-orientation analysis as prescribed in various SOA methodologies (SOM). During this stage the design artefacts adopt the format of software-engineering modelling, e.g., UML or SoaML.

To summarize, the analysis and design activities in service engineering can be divided into two stages: (1) Pre-Service Design, and (2) Service Design. The pre-service design covers business side analysis with an end result of a business model. The service design stage produces the description of service model in two aspects interaction model and process model. For the special case of software interactivity, the service design
stage is continued to the software-service design sub-stage, which is performed in a manner of software engineering analysis and design.

4.2 Preliminary Framework

Two research streams are identified in the context of Service Engineering. The first stream takes the more classical perspectives of service by emphasizing on ‘business-service’ context [26][31][32]. The second stream enlarges the context, but puts more emphasize on the ‘software-service’ context [13][67][53]. The second stream pursues an enterprise-wide alignment of business-services with the architecture of a ‘software-service’, often by adopting various Service Oriented Architecture Methodologies (SOM), such as MSOAM [24] or SOMA [47].

Combining these streams, a SvE framework prototype is proposed. The framework is composed based on the SAT (Stage-Activity-Technique) framework structure [31]. ‘Stage’ is a high-level concept in partitioning the process in steps. A stage is a container for one or more ‘activities’, and ‘technique’ is a tangible instrumental aid for performing an activity, e.g. Focus group, Delphi, SWOT analysis [13].

In generalizing the structure, and to reduce the prescriptive detail of the framework, the ‘Technique’ component is replaced with an ‘Artefact’ component. Artefact is defined as an output for an activity and may serve as an input for the subsequent activities. Certain techniques may be involved in producing the artefact, but this detailed ‘how’ aspect is
omitted to retain the abstract level of the targeted framework and to allow development flexibility.

Instead of adopting the ‘technique’ component, specific format is specified for each type of artefact as suggested by literature, i.e. the Business Model Canvas (BMC) [85], Service Blueprint (SBP) [88], and Business Process Model and Notation (BPMN) [93]. For software modelling, UML is proposed as the most common notation for software analysis and design. These formats imply that the technique of each activity should use and produce the formatted artefact during the process of analysis and design. This approach ensures the open-ended nature of the proposed framework. Table 4.1 summarizes these components into a prototype of the service engineering framework.

Table 4.1: Service Engineering Framework (prototype)

<table>
<thead>
<tr>
<th>Stage</th>
<th>Activity</th>
<th>Artefact</th>
<th>Format</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Identification</td>
<td>Requirement gathering</td>
<td>Existing business model</td>
<td>BMC (as-is)</td>
</tr>
<tr>
<td>Business modelling</td>
<td>Business requirement</td>
<td>Proposed business model</td>
<td>BMC (to-be) Structured narratives</td>
</tr>
<tr>
<td></td>
<td>Business service catalogue</td>
<td>Proposed business model</td>
<td>BMC (to-be) Structured narratives</td>
</tr>
<tr>
<td>2. Design</td>
<td>Service Modelling</td>
<td>Service model</td>
<td>Service Blueprint</td>
</tr>
<tr>
<td>Process Modelling</td>
<td>Process model</td>
<td>Process model</td>
<td>BPMN</td>
</tr>
<tr>
<td>Software Modelling</td>
<td>Software model</td>
<td>Software model</td>
<td>UML</td>
</tr>
<tr>
<td></td>
<td>Software Service Catalogue</td>
<td>Software Service Catalogue</td>
<td>Structured table</td>
</tr>
</tbody>
</table>

As an initial refinement of the prototype, the activities in case studies are grouped into four activities: (1) Understanding the service context, (2) defining the service concept, (3) designing the business service, and (4) designing the software services. Each activity is
operationalized with output artefacts and its suggested format. This structure is summarized and illustrated in figure 4.3.

<table>
<thead>
<tr>
<th>Stage</th>
<th>1. IDENTIFICATION</th>
<th>2. DESIGN</th>
</tr>
</thead>
<tbody>
<tr>
<td>Activity</td>
<td>1.1 Understanding Service Context</td>
<td>1.2 Defining Service Concept</td>
</tr>
<tr>
<td>Artfact</td>
<td>1.1.1 Requirement List</td>
<td>1.2.1 Business Model</td>
</tr>
<tr>
<td>Tool (Format)</td>
<td>Survey and Observation Results</td>
<td>Business Model Canvas (BMC)</td>
</tr>
</tbody>
</table>

Figure 4.3: Service Engineering Framework (1st iteration)

The identification stage is the first stage as a business side analysis in which potentials for new services are identified and proposed with analysis of existing condition of the organization. The design stage, as the second stage, consists of both the business design and technical design processes. The services are designed or redesigned in this stage with regard to the existing condition of the organization.

This initial framework is considered to be sufficiently covering five core aspects of service engineering identified in the previous section: (1) Business Model, (2) Service Model, (3) Interaction Model, (4) Process Model, and (5) Software Model.

### 4.2.1 Identification Stage

In the identification stage, potential new business service for a customer is identified and defined. A common motive of anew service innovation is
to improve customer value. In that sense, the service identification stage must consist of steps to identify services with high customer value.

A common identification technique for service innovation is the questionnaire method. The technique is commonly used during a market research activity. The use of questionnaire is part of true requirement investigation, as the requirement is defined based on actual interaction with prospective customers [13]. An analysis from the questionnaire data might instigate a new service innovation. For example, the questionnaire result might suggest an improvement in the firm key activities. Therefore, a service innovation can be introduced in the back-end to enhance the actual service activities.

Another common technique of true requirement investigation is the observation. Observation is a method to capture the information based on a real-world situation. The observation process can also include a questionnaire, or interview activities. During the observation, the researcher directly visits or experiences the site of the actual business process and customer transactions performed.

New service innovations are frequently driven by implementing new IT components. An assessment for IT potential is therefore valuable in the identification stage. The latest trend in IT, such as cloud computing offering, could serve as an important element in the service innovation.

The Business Model Canvas (BMC) [85] format is proposed in this stage as the artefact format and also as the analysis and modelling tool to identify potential components to improve in defining a service. The nine
blocks of the BMC are the analytical bases for improving the firm value (figure 4.4). The potential of improvement might arise from the customer segment, revenue stream, cost structure, value proposition, or in key activities.

![Figure 4.4: BMC context in Service Engineering][84]

The identification stage might also be assisted by the definition "as-is BMC", to be analysed into the proposed "to-be BMC". This new BMC represents the proposition of service innovation. For example, the improvement decision might reside in the value proposition block. In this case a new service is proposed for the customers, in the form of direct customers facing services.

The “to-be BMC” serves as the starting point in service design. Each component of BMC eventually forms components of service design: Resource model, Process Model, Product Model and Marketing Plan. The "to-be BMC" is complemented with a summary list of the service innovation idea. Both components should be formalized as part of a business service identification document.
4.2.2 Design Stage

The design stage of the proposed framework is divided into two sub stages (activity): Business-service design, and software-service design. The goal of the first activity, the business-service design, is to create the design of the service defined from analysis in the identification stage. The second activity, software-service design, is the sub-stage to elaborate the design of software component structure using SOA approach and methodology.

During service process design, the service blueprinting (SBP) format (figure 4.5) [88] is suggested to visualise a series of service interactions between the service provider and consumer within the whole cycle of the business service. The format defines five layers for service interaction: physical evidence, customer action, on-stage contact employee, back-stage contact employee, and support process.

The actual focus for the interaction model is only in the two top most layers. The detailed process beyond the *line-of-visibility* is provided in the
process model. Business Process Model and Notation (BPMN) is used to elaborate the supporting process in the context of service provision. As required, a more detailed process abstraction can be further elaborated to produce the most atomic abstraction of activities.

The objective of the business service design sub-stage is to have an overview of the service. The result from this process will be used as a reference to identify the role of the software service during the interaction, and during the execution of the supporting process. Some automated feasibilities in interactivity and operation serve as the basis for devising software-services in the subsequent activity of software-service design.

In the software-service design activity, a SOA methodology guides the creation of services in an IT context, i.e. software service. A SOA methodology commonly involves an identification activity followed by a design activity. The purpose of identification is to identify the candidate services, while the purpose of the design stage is to define the services specification, such as the service contract and choreography.

In the proposed framework, some parts of SOA service identification have already been performed during the previous stage, in the form of to-be BMC, service blueprint and BPMN. These artefacts are the starting point for Service identification for SOA methodology.

In the case of adopting Thomas Erl’s MSOAM [24], the first step of the methodology, ‘Business Model Alignment’, is simplified by employing the results of BMC analysis. The same idea is also applicable to ‘Business
Requirement Definition’ and ‘Decompose Business Process’, in which the Interaction Model and Process Model artefacts can be used.

A clear delineation should be made regarding the ‘Software-Service Design’ (activity 2.2). While SOM generally does not limit the analysis on the software-service alone, it will only elaborate a subset of operation interactivity with feasible software enablement. Therefore, manual services are defined only until the ‘Business Service Design’ (activity 2.1) to be further elaborated in anon-IT manner, while the software-services are elaborated through the ‘Software-Service Design’.

4.3 Case Studies

To assess the feasibility and its sufficiency, the first version of the framework was applied in three separate service projects as research case studies.

4.3.1 Case Study 1 – *Citizen Registry*

The first case study is focussed on improving the performance of business services in a local governments’ citizen registration office. The project scope covers the aspects from customer facing business-services through to its implementation as software-services.

For understanding the service context, the case study conforms to the framework by initiating a qualitative approach to the requirement
process. In this process, a list of high-level directives, problems, and opportunity were compiled (table 4.2). The directives are collected from documented narratives defining the services mandate. The high-level problems are defined from business process analysis which measures a performance gap between the current service situations and mandated service performance. Service performance is measured from both the conformance to transparency principles and service time. The opportunities list is drawn from the directives and identified problem as a list of high-level items for improving the performance of business service.

<table>
<thead>
<tr>
<th>KPI</th>
<th>Rethink</th>
<th>Redesign</th>
<th>Retool</th>
</tr>
</thead>
<tbody>
<tr>
<td>Complaints management procedures</td>
<td>Rethink of complaints management issues pursuant to Rule Minister PAN - RI 13 2009</td>
<td>Redesign-complaint management through information systems</td>
<td>Routine evaluation of complaints and settlement issues</td>
</tr>
<tr>
<td>Percentage number of complaints can be resolved</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Complaints management refers to the Minister PAN - RI No. 13 of 2009 in order to improve the quality of service</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Implementation of the ISM Survey</td>
<td>Rethink of the achievements and the recommendations of the citizen satisfaction index survey.</td>
<td>Business process redesign propose for all kinds of services</td>
<td>Evaluation of citizen satisfaction survey on a regular basis to determine the level and recommendations that should be followed up.</td>
</tr>
<tr>
<td>ISM Survey conducted refers Kepmenpan 25. 2004</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>The average scores obtained ISM Follow-up of the ISM survey</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Electronic information system services</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Delivery of information public services to the citizen</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Disclosure of public service information</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 4.2: Key Performance Indicator Analysis (Case Study 1)

This first stage produced a proposed service innovation, formalized in the format of BMC. The produced BMC highlights the innovation in three of its component: (1) Activity, (2) Value, and (3) Channel. The service innovation is also accompanied with a catalogue of targeted business-service, along with the improvement criteria.
As a direct customer facing service, the design stage is initiated with a definition of customer interaction model. The interaction model is formalized with Service Blueprinting diagram to illustrate customer touch points with the registration office. The back-end processing is the elaborated in BPMN notation (figure 4.6), in conformance with the interaction model,

![BPMN Diagram](image)

Figure 4.6: BPMN of the ‘to-be’ business process (Case Study 1)

For the software service design activity, the case study partially adopted SOMA [47] as its SOA methodology. In producing the software service candidates, two approaches were combined. The first approach composed a Goal Service Model matrix which derives service candidates from mandated goals and criteria. The second approach decomposed business process into candidate operations and services. The two results are consolidated and analysed to produce the (software) service catalogue.
Entering the software design, the candidate services were functionally grouped as service components. The usage structure of the software services was also formalized in a use case diagram and use case scenario. Afterward, the supporting data structure was composed as class diagrams. Finally, as exemplified by SOMA, all the components were combined into SOA Architecture (Figure 4.7), in a layered manner, from
the business service to physical data assets. Figure 4.8 summarizes the framework adoption in this case study.

4.3.2 Case Study 2 – Citizen Relationship Management

The second case study targeted a Citizen Relationship Management system in a local government. The project emphasizes the aspects of service goal and service value under the transparency mandate related to an ‘Open Government’ initiative. As in the first case study, the service under study contains a direct customer interaction in the form of a citizen report, complaint or query.

Similar to the first case study, the project commenced in a qualitative manner of requirement engineering by compiling a high-level list of directive, problem and opportunity to understand the service context and current situation. Afterward, a SWOT analysis is performed to organize the qualitative results into feasible opportunities of service innovations.

The intended service innovation was first formulated in a BMC format. The improvement potential was identified in the components of (1) value, (2) customer relationship, and (3) channel. The proposed service innovation is composed into a business-service catalogue with a list of mandated service features. The catalogue and features were formatted into a matrix of the SOMA Goal Service Model to produce a list of business sub-services.
In the interaction design, both the existing and future interaction patterns are modelled using the Service Blueprinting format. The future interaction pattern was designed to adopt the features mandated from the identification stage. Detailed business processes were elaborated using BPMN (figure 4.9). Each new process feature was cross-referenced with an item in the Goal Service Model matrix to justify its introduction.

Figure 4.9: BPMN of the 'to-be' Business Process (Case Study 2)

Figure 4.10: Component Dependency Diagram (Case Study 2)
In the software service design, the project adopted the process decomposition approach of SOMA [47]. The decomposed process is combined with the Goal Service matrix to produce a software services catalogue. The software design also included a Use Case diagram, Use Case scenario, and Service Component dependency diagram (figure 4.10). As standardized in SOMA methodology, these components were then summarized in the SOA Architecture Diagram.

Figure 4.11 summarizes the framework adoption in this case study.

### 4.3.3 Case Study 3 – Accounting Information Service

Unlike the two previous case studies, this case study concerns less on direct customer interaction and more on inter-organization interaction. The service system under study is an internal accounting system which is targeted to be able to provide more transparent information services.

To understand the context, the project began with composing the business model into a BMC format. The high-level view of BMC does not sufficiently explore potential scope for service innovation. To elaborate
the context comprehension, the Component Business Model (CBM) [94] concept was adopted (figure 4.12). The CBM is able to highlight areas within the competency map of the accounting office with potential information services.

Figure 4.12: Component Business Model (Case Study 3)

The potential service scope was identified in the CBM. It drives the business process analysis in the targeted area which was later documented in a BPMN format. In a qualitative manner these potential scopes were filtered and prioritized based on the mandated directive principles. The resulting list was then defined as a service innovation opportunity.

A detailed service innovation proposition was then formalized into a business service catalogue along with its features. A to-be BMC was also created highlighting a modification on the components of (1) values and (2) channels. During the identification stage, the existing structure of application subsystems and its domain ontology were also assessed and
documented. The domain ontology is documented using an ER diagram (figure 4.13).

Business process design marked the beginning of the design stage. Detailed BPMN were created to highlight the modification of business processes. Introduction of new processes led to new operations assigned to the business actors, presented in Responsibility Assignment Matrix. The matrix defines actors assigned to each level of responsibilities: Responsible, Accountable, Consulted, and Informed (RACI), as presented in figure 4.14.
Accommodating the proposed functionality, new data architecture was also introduced in this stage. For the software service design, the case study followed MSOAM guidelines [11] in breaking down processes into operations, grouping operations into software service candidates, normalizing service candidates, and structuring service dependency in a layered approach. The MSOAM informal notation of software service is used throughout the design which simply stated the service name and the operations contained. To summarize the structure, a Service Architecture was defined to a logical arrangement of four service layers from business processes to physical data assets (figure 4.15).
UML style notation was also used in the design documentation. Sequence diagram were used to illustrate complex software service interactions (figure 4.16). To communicate the design of software service usage, service functionality was assigned to system actors in the form of a Use Case diagram for each subsystem (figure 4.17).
Figure 4.16: Sequence Diagram in Case Study 3

Figure 4.17: Use Case Diagram in Case Study 3

Figure 4.18 summarizes the framework adoption in this case study.

<table>
<thead>
<tr>
<th>IDENTIFICATION</th>
<th>DESIGN</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Understanding Service Context</strong></td>
<td><strong>Defining Service Concept</strong></td>
</tr>
<tr>
<td>As-is Business Model (CBM)</td>
<td>As-is BPMN</td>
</tr>
<tr>
<td>Directives, Opportunities (prioritized)</td>
<td>Capturing Software System Structure</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 4.18: Adapted Service Engineering Framework (Case Study 3)
4.3.4 Evaluation of Case Studies

Several observations can be drawn from the stages and activities undertaken in the case studies. Some additional aspects are introduced in the ‘Understanding Service Context’ activity in the ‘Identification’ Stage. The first case study demonstrated that examining a current business practice can be performed with a Business Process analysis technique, by documenting and analysing the process in BPMN format. This also demonstrates the implied implication of the selected ‘artefact’ format (i.e. metamodel) to the detailed ‘how’ aspect (i.e. ‘technique’ component) in the framework activities.

The third case study also shows that capturing a current business model with BMC is feasible, but it can be considered insufficient in some cases. A closer look at business unit capabilities with CBM provides beneficial insight to the potential area of service enablement.

The ‘Understanding Service Context’ sub-stage defines the scope for a service enablement. The scope can be qualitatively derived from the business mandate and identified problems in the current situation. The resulting list of enablement opportunity can be further narrowed into a priority based on identified important directives, as in the first and third case study, or based on an analysis technique such as with SWOT analysis.

The second activity of the identification stage is basically uniformed with BMC format for business modelling and a business service catalogue
for business-service modelling. In the second case study, an additional artefact for the business model is introduced to define and elaborate the ‘value’ concept brought to the business model from the service.

As an artefact, the business service catalogue is a list of enablement targets enriched with service feature descriptions mandated for the design stage. The second case study demonstrated that the list and its features can be linked to the mandated directives using a Goal Service Matrix. In a later stage, this matrix will be used to derive atomic operations to be implemented.

The second case study also showed that, before entering a design stage, an examination should be performed to understand the current software system structure. This activity can be done in parallel with the service concept definition activity.

For producing the ‘Process Model’ in the Design stage, the second case study suggests that interaction modelling can be omitted for certain type of service systems without direct customer-facing interaction, where service transactions are performed mechanically through system interactions. The second case study also introduced the use of the Responsibility Assignment matrix (RACI) to summarize the assignment of a process and operation to actors in the related business process diagram (BPMN).

The re-designed business process also derives some consequences for data entity modification. Therefore, a re-designing of data architecture can be performed in conjunction with ‘Process Model’ design, before
entering the ‘Software-Service Design’ activity. This variation conforms to the aspect of ‘ontology engineering’ in the broader context of service engineering [82].

The ‘Software-Service Design’ activity in the design stage produced a list of exposable software-services. This list is a result of combined analysis of process decomposition and process-operation grouping service goal referencing as demonstrated in the first and second case study. Additionally, software-services could be regrouped into service-components to be developed as software components in the software development stage.

There are various graphical styles available in abstracting a software-service: As a container with service name and operations (in UML class, and MSOAM abstraction), as use-case ellipse (in use case diagram), or as an UML component (in component diagram for software development).

A structured relation between software-services is provided in three manners: First, a UML-styled sequence diagram which shows the sequential execution of software service operations in a complex multi-service operation. Second, a dependency diagram which show the inter-calling relation between software services. Third, in a SOA architecture which summarizes all software-service dependencies in a layered format.

The importance of SOA architecture is shown by the fact that all of the case studies provided the diagram. The dependency diagram is used to elaborate service inter-relations in a bounded subsystem. It can also be
used to assess the grouping of software-services into software components. The sequence diagram is only occasionally used to illustrate the execution of complex orchestrated or composed software services.

In a technical level, a software-services catalogue lists services with its associated operations and parameter options. The catalogue is also accompanied with UML Use Case diagrams which specify the invoking right of a software-service to certain actors. The actor itself can be a human actor, e.g. customer or an automated process, i.e. application system.

Accommodating all the variants of framework adoption in the case study, a consolidated version of the framework prototype is visualized in figure 4.19. Due to the dominant existence of legacy application systems in case study 3, activities for capturing and analysing software systems, i.e. software architecture and data architecture, are embedded in the framework.

![Figure 4.19: Adapted Service Engineering Framework (Case Studies)](image-url)
Omitting activities related to software-engineering in both stages, the framework is simplified back to the service level context as the 2nd iteration of the proposed framework (figure 4.20).

<table>
<thead>
<tr>
<th>1. IDENTIFICATION</th>
<th>1.2 Defining Service Concept</th>
<th>2.1 Business Service Design</th>
<th>2.2 Software Service Model</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1 Understanding Service Context</td>
<td>1.2 Defining Service Concept</td>
<td>2.1 Business Service Design</td>
<td>2.2 Software Service Model</td>
</tr>
<tr>
<td>1.1.1 Business Directives</td>
<td>1.1.2 (as-is) Business Model</td>
<td>1.2.2 (to-be) Business Capabilities</td>
<td>2.1.1 (to-be) Interaction Model</td>
</tr>
<tr>
<td>1.1.2 (as-is) Business Model</td>
<td>1.1.4 Business Capabilities</td>
<td>1.2.2 (to-be) Business Model</td>
<td>2.1.2 (to-be) Process Model</td>
</tr>
<tr>
<td>1.1.3 (as-is) Business Process</td>
<td>1.1.5 Opportunity Priority</td>
<td>1.2.3 Business Service Model</td>
<td>2.1.3 Data Model</td>
</tr>
<tr>
<td>Business Model Canvas</td>
<td>Business Model Canvas</td>
<td>Business Service Catalog</td>
<td>2.2 Software Service Model</td>
</tr>
<tr>
<td>BPMN Business Process</td>
<td>Component Business Model</td>
<td>Service Blueprint</td>
<td>Business Model Canvas</td>
</tr>
<tr>
<td>List of Opportunity</td>
<td>UML Use Case</td>
<td>BPMN Business Process</td>
<td>UML Class Diagram</td>
</tr>
<tr>
<td></td>
<td></td>
<td>UML Class Diagram</td>
<td>UML Sequence Diagram</td>
</tr>
</tbody>
</table>

Figure 4.20: Service Engineering Framework (2nd iteration)

From the case studies, two additional concepts were emerged previously uncovered by the framework: business unit capability (component 1.1.4), and service value (component 1.2.2). Table 4.3 summarized these aspects of service engineering.

Table 4.3: Service Engineering Modelling Aspects (from Case Study)

<table>
<thead>
<tr>
<th>Modelling Aspect</th>
<th>Related Concept</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Business model</td>
<td>Business model value-net [31], service customer [32]</td>
</tr>
<tr>
<td>2 Capability model</td>
<td>Service-resource model[24], service entities [32]</td>
</tr>
<tr>
<td>3 Value model</td>
<td>Service value-net [31], service goal [32], service linkage[32]</td>
</tr>
<tr>
<td>4 Service model</td>
<td>Service-product[24]</td>
</tr>
<tr>
<td>5 Interaction model</td>
<td>Service input-output [32]</td>
</tr>
<tr>
<td>6 Process model</td>
<td>Service-process[24][32]</td>
</tr>
<tr>
<td>7 Software model</td>
<td>IT enabler[32]</td>
</tr>
</tbody>
</table>
4.4 Service Engineering with SoaML

As exercised in the ontological chapter, the existence of the SoaML specification is reused here as a comparison source with the tentatively produced framework.

4.4.1 Specification of SoaML Activity Flow

Some metamodels are introduced embedded with a formal methodological guidance on its usage, e.g. SOMF [59] and Archimate [95]. But other metamodel specification are introduced independently and detached from the usage guidance, e.g. UML [56]. SoaML is an example of the second type where the actual usage pattern within an engineering process is to be decided by the modeller.

SoaML specification explicitly declares that it supports both top-down and bottom-up approaches in SOA development. The key difference is in the departing point for service identification, which led to a flow of subsequence process. The specification states five service identification approaches facilitated by SoaML [60]:

1. **Top-down A**: Starting from the *Services Architecture*, as a community of interacting participants, to individual *Service Contracts*, as an interaction agreement toward further detailed service identification.
2. **Top-down B**: From organizing functions into a hierarchy of *Capability* to identify potential *Service Interfaces* that will expose the capability as a service.

3. **Top-down C**: From *Business Process* within a specific purpose to collect functional *Capabilities* and *Roles* related to *Participant*.

4. **Bottom-up A**: From assessing assets owned by *Participants*, as potential *Capabilities* to be exposed as a service.

5. **Bottom-up B**: From an identification of common data and data flows between parties, to be grouped into modules of services.

![SoaML Top-Down Approaches Artefacts Flow](image)

**Figure 4.21: SoaML Top-Down Approaches Artefacts Flow**

Adhering to the service engineering context of this research, in which a service is not proposed from its underlying asset but instead
grounded on specific goals in a certain process context, the top down approach is more appropriate to be adopted for the Service Engineering Framework.

Interpreting and combining the top down approach, the sequential dependence between artefacts can be visualized as a SoaML artefacts flow in figure 4.21. Regrouping the SoaML artefacts produced a stereotyped flow of the service engineering process. Figure 4.22 shows the regrouping with activities and artefacts numbering correlated to the framework in figure 4.23.

Replacing UML roles in the previous framework (figure 4.20) with SoaML artefacts (figure 4.22) produces a modified framework (figure 4.23). A simplification of the framework, by omitting software-engineering activities, produces the third iteration of the framework, as presented in figure 4.24.
Figure 4.23: Incorporating SoaML Artefacts into Framework

Figure 4.24: Service Engineering Framework (3rd iteration)
4.4.2 Case Study with SoaML

The three preliminary case studies use UML in modelling the software service. UML adoption is fairly sufficient for modelling a simple software service system. But as also evident in the previous case studies, UML lacks standardized notations to represent the service concepts. To verify the usability of SoaML as a service modelling language in a complex service system, a larger case study is performed.

The case study operates in the larger context of implementing a smart campus initiative within a university-wide scope. The project covers a complex system of services involving multiple participants, including students, lecturer, administrative staff of various business units, and external providers.

Figure 4.25: Package Diagram of Project Domains (Smart Campus)
The project is divided into six domains (figure 4.25). The project directives, i.e. service objectives, are defined for each domain before detailing the domain into several service systems.

In total, there are 18 sub-systems to be detailed as service-systems under the 6 domains (figure 4.26). This boundary of service systems serves as a container for individual software services to be introduced and defined in the design stage.
In general, the project follows framework guidance:

- Business directives are captured as business goals (table 4.4), to derive service requirements to be meet as service goals (table 4.5). Both artefacts take the form of a structured narrative presented in a form of a table.

- Business Model Canvas (BMC) is used to assess the business context, as a basis for (software) services introduction. BMC is produced both in ‘as-is’ version during identification stage and in ‘to-be’ version in the design stage.

- BPMN diagram is used to capture the existing business process during the identification stage and defined during the design stage to elaborate on the process supporting the proposed business model.

- Service Blueprint is used in parallel with BPMN but only in the design stage to highlight the interactions processes expected from the consumer for each service system.

  Occasionally, a high level view of interaction between a participant and the targeted service system is provided in the form of a context-diagram, which could take the form as a Data Flow Diagram (DFD) style context diagram as suggested in [62], or as a UML use case.
Table 4.4: Sample of Business Goals (Smart Campus)

<table>
<thead>
<tr>
<th>Domain</th>
<th>Service Strategy and Objectives</th>
</tr>
</thead>
<tbody>
<tr>
<td>Smart Learning</td>
<td>1) improve collaborative learning between lecturers and students;</td>
</tr>
<tr>
<td></td>
<td>2) provide equal opportunities for all students to study without being limited by distance and time;</td>
</tr>
<tr>
<td></td>
<td>3) support self learning in accordance with the correct path;</td>
</tr>
<tr>
<td></td>
<td>4) evaluate learning competency achievement (self assessment).</td>
</tr>
<tr>
<td>Smart Management</td>
<td>1) face recognition to avoid crime or to record people who are in a particular area;</td>
</tr>
<tr>
<td></td>
<td>2) trace the movement or mobilization of people to know the distribution of humans at certain times;</td>
</tr>
<tr>
<td></td>
<td>3) smart cards to provide parking permits and non-cash transactions;</td>
</tr>
<tr>
<td></td>
<td>4) recording attendance on a teaching and learning activity.</td>
</tr>
</tbody>
</table>

Table 4.5: Sample of Service Goals (Smart Campus)

<table>
<thead>
<tr>
<th>Services Systems</th>
<th>Services Systems Requirements</th>
</tr>
</thead>
<tbody>
<tr>
<td>Smart Learning Management</td>
<td>a. should be able to meet the needs of recording data relating to teaching activities, including the course syllabus, meeting schedule, student attendance, project group creation, and the provision and execution of tasks.</td>
</tr>
<tr>
<td></td>
<td>b. should also be able to manage and set the main supporting data of teaching process.</td>
</tr>
<tr>
<td>Personalized Learning System</td>
<td>a. should be complemented by the concept of adaptive learning</td>
</tr>
<tr>
<td></td>
<td>b. providing feedback on achieving current learning outcomes in intelligent learning environments.</td>
</tr>
<tr>
<td></td>
<td>c. may include: learning objectives, plans, learning maps, learning activities, competence levels, performance or achievement of learning outcomes, and reflections</td>
</tr>
</tbody>
</table>

The main difference with the first cycle of case studies is in the use of SoaML to replace UML’s role in service modelling. For illustrative purposes several artefact samples are presented here.

Figure 4.27: SoaML Capability Diagram (Smart Campus)
The project adopted SoaML’s “Top Down B” and “Top Down C” approaches in which the service identification departs from the business process[77], and optionally via a capabilities set collected from the process (figure 4.27). The produced SoaML’s capability diagrams serve as templates for service candidates, and service interfaces.

![Figure 4.28: SoaML Service Interface Diagram (Smart Campus)](image)

Services are defined with SoaML’s Service Interface as a composite usage of an atomic Interface (figure 4.28). SoaML Interface lists the available operations available to be invoked by partnering participants. The interaction protocol, which specifies the sequence of invoked operations, is presented in UML’s sequence diagram within the service interface definition. Additionally, descriptive rules for service engagement are presented within SoaML’s service contract diagram, e.g. minimal number of items ordered.

These service components, i.e. interface and service interface, are collected into a service container as a SoaML’s participant diagram (figure 4.29). In the participant diagram, the service (SoaML’s service
interface) becomes a participant port, and the interface becomes a spoke or a socket protruding from the port.

Finally, the whole service components are tied and collected into SoaML’s service architecture, covering participants, and services involved (figure 4.30).

4.4.3 SoaML Case Study Evaluation

SoaML may have weakness in its ambiguity and overlapping of service abstraction, but compared to the use of UML, the superiority of
SoaML is evident during the case study. A comparison between the two is provided in table 4.6.

Table 4.6: Comparison of SoaML and UML for Service Modelling

<table>
<thead>
<tr>
<th>SOA Concept</th>
<th>SoaML Representation</th>
<th>UML Representation</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Service</td>
<td>Interface, Service Interface</td>
<td>&lt;&lt;service&gt;&gt; stereotype of class or interface</td>
</tr>
<tr>
<td>2. Atomic Service</td>
<td>Interface</td>
<td>&lt;&lt;service&gt;&gt; stereotype</td>
</tr>
<tr>
<td>3. Composite Service</td>
<td>Service interface, Service contract</td>
<td>Usage dependency of &lt;&lt;service&gt;&gt;</td>
</tr>
<tr>
<td>4. Service Architecture</td>
<td>Service Architecture</td>
<td>(High level) Use Case diagram</td>
</tr>
<tr>
<td>5. Element</td>
<td>Participant, Component</td>
<td>&lt;&lt;entity&gt;&gt; stereotype, Component</td>
</tr>
<tr>
<td>6. Endpoint</td>
<td>Port in Participant.</td>
<td>Port in component</td>
</tr>
<tr>
<td>7. Capability</td>
<td>Capability, Participant behaviour (as a set of operation and method)</td>
<td>Class operation and method</td>
</tr>
<tr>
<td>8. Service interface</td>
<td>Interface, Service Interface</td>
<td>Interface</td>
</tr>
<tr>
<td>9. Service Contract</td>
<td>Service Contract</td>
<td>-</td>
</tr>
<tr>
<td>10. Role</td>
<td>Role (in Service Contract, Service Interface and Service Architecture)</td>
<td>Actor specialization, or actor to use case association line</td>
</tr>
<tr>
<td>11. Information Type</td>
<td>Message Type</td>
<td>Data type</td>
</tr>
</tbody>
</table>

Despite its complexity, SoaML is considered to be better in abstracting SOA concepts, as partially reflected in [63]. Elaborating point 3, 4, 5, 9 and 11 of the comparison table, SoaML can be considered superior due to the lack of several features in UML:

- SoaML provides better traceability in tying the related design components during the definition of a service composition (service interface and interface), its interaction protocol behaviour (sequence diagram behaviour of service), toward the definition of service
components (participant) through the service architecture (participant and service).

- SoaML facilitates explicit separation between a service interface and the service implementation. Consequently, the concept of ‘role’ within a service and ‘role-binding’ during service interactions can be clearly presented. A particular participant can be abstracted to assume more than one role within the scope of a service architecture.

- SoaML is able to represent a service system involving multiple provider roles, in the form of number of roles defined in the service interface and service contract diagram.

Considering these features, SoaML can be suggested as a metamodel for complex service system, which is able to accommodate certain types of business models with various assumable roles and multiple service providers.

While ‘Service Architecture’ is part of SoaML diagrams, it is actually more applicable for the non-IT context, i.e. non-software service. Service architecture visualises the interrelation between participants within a service community, adopting a certain business model for each participant. Therefore, its usage in the framework is placed in the ‘Business Service Design’ activity (activity 2.1, artefact 2.1.1).
The same assessment can also be given to ‘Service Contract’ and ‘Capability’. Service contract is mostly derived from descriptive business rules and agreement in conducting service interactions, while capability relates to intrinsic capacity owned by a participant. Unless the rules and capacity relate to automatic features of a software service, they reside outside the boundary of ‘Software-Service Design’ activity.

Service architecture is useful in a business context to define the scope of the analysis and design. Certain participants may be defined to be an external party, which is beyond managerial control of the service system, but adheres to a mutually agreed collaboration rule and protocol. SoaML actually provides a way to define an external participant in the form of a participant with a dashed-line boundary [60].

The key differentiator between a manual service and software service is the type of Participant involved. For software-services, both interacting participants are software component. Consequently, all participants defined in the ‘Software-Service Design’ activity must be implemented as software components in the implementation.

This chapter has defined a proposed and tested framework of ‘service engineering’. In the next chapter, the framework will be verified with the ontologies of the third chapter in terms of its coverage sufficiency. An identification of a stereotyped format for an artefact is also drawn. The next chapter explores the metamodel landscape to identify potential format for each framework’s artefacts.
Chapter 5
Modelling in Service Engineering

5.1 Metamodel Exploration

This chapter provides format elaboration for each artefact from activities in the Service Engineering Framework. The artefact is essentially a model representing partial aspects of the service system under study. The model is built based on a specific modelling language convention, i.e. metamodel.

Figure 5.1: Research Flow and Content Map (Chapter 5)

Figure 5.1 illustrates the content of this chapter with regard to research activities flow. The chapter is divided into three parts. The first
part explores the current landscape of metamodel in the objective of collecting a palette of potential metamodels to be included in the framework. The second part examines the framework in terms of artefacts coverage to the ontology to assess the completeness of the framework and characterize the metamodel suitable for the artefacts. Finally, the third part combines the previous two results to enrich the framework with alternative potential metamodel.

5.1.1 Defining Model

Models hold a significant role in software engineering throughout its evolution. Since the adoption of flowchart [96], the modelling approach has shifted on several occasions; from the structured programming paradigm [97], to object-oriented, with Unified Modelling Language (UML) [98], and more recently toward a service-oriented approach [99]. The importance of modelling in software engineering is further strengthened by the emergence of Model Driven Engineering [100], in which modelling becomes the core of the engineering activity, and the coding is mainly performed mechanically with automated machine assistance, theoretically making the process becomes more efficient and adaptive to changes.

The last two decades witnessed an unprecedented proliferation of modelling languages, or metamodels. Domain disciplines, and industrial practitioners, propose metamodels for a specific purpose or with general
applicability in mind. A consortium for standardized metamodels, the Object Management Group (OMG), has published no less than 70 model-related specifications since their first UML specification in 1989 [101].

In its most generic form, a model is defined as a “simplified view, or abstraction, of reality” [102]. More formally, ‘model’ is defined as:

- “a set of statements about a system under study” [103].
- “a set of formal elements describing something being developed that can be analysed using various methods” [104].

From these definitions a model can be summarized as an abstraction format, representing an underlying object. A model is typically required for two reasons. First, the object might be too complex to comprehend in its original size and details. Second, the concept might not be discernible, either as an abstract concept or is not yet materialized.

The model definitions imply the purpose and activities surrounding a model, i.e. studying and developing. In this context, the “bridge-model”,

![Analysis-Synthesis Bridge Model](image_url)

Figure 5.2: Analysis-Synthesis Bridge Model[105]
as visualized in figure 5.2 summarizes the roles of models in engineering [105]. Model is based on an object, and the model can represent the object in various periods: past state, current state ("as-is"), or a future state ("could-be", "to-be").

An engineering process can be seen as a transition from problem domain to solution domain [61]. During the analysis phase, the “as-is” models from the “problem domain” are composed to gain an understanding of the current situation, to identify the problems, to examine the needs, and to decide upon a specific requirement. During the design phase, the “to-be” models from “solution domain” are created in which a future state or a specific solution is proposed. During these phases, model serves as a communication and collaboration tool among involved parties, within a project team or with the stakeholders.

Besides serving as a communication and collaborating tool, a model can also serve as a verification and validation tools during the early engineering phases, e.g. prototyping in agile forms of software engineering. In some cases, the models are gradually elaborated in detail toward later stage of design to pinpoint a specification and to avoid interpretation variance during the implementation phase.

### 5.1.2 Model Taxonomy

As an abstraction tool, model can be composed in various format, e.g. textual, or graphical format [106]. Textual model presents its information
as a sequence of characters. Narrated description of a situation and declarative languages (e.g., OWL, RDF) are some examples of textual modelling. A graphical model on the other hand, uses a spatial arrangement of graphic and text elements to convey the information. Graphical models can also be categorized as two-dimensional models, while the linear representation of textual model as one-dimensional.

Graphical models possess several advantages compared to a textual model [106]. First, a diagram conveys its information in a concise manner, while still retaining some degree of precision. Second, a diagram facilitates a faster cognitive comprehension compared to a textual description. This is due to its parallel presentation format, compared to the serial presentation of textual form. Third, graphical information typically can be cognitively processed and retained more efficiently due to the nature of the human brain.

On the other hand, textual representation generally possesses a higher expressive power compared to a graphical form. Textual form has a wider flexibility for varying expressions due to less restrictive vocabulary options. Also, due to the similarity to a natural language, the learning curve for a textual language is relatively shorter compared to a graphical language. For a graphical model, the understanding of the notation vocabulary is a necessity. Despite these disadvantages, a person normally prefers graphical representation over textual form due to its efficiency and conciseness [106].
5.1.3 Metamodel Anatomy

A clear distinction should be made between a model and a modelling language. To represent an underlying object, a model is created based on a specific modelling language, i.e., a metamodel specification. The specification sets the convention for creating and interpreting a model.

![Figure 5.3: Metamodel components][107]

Figure 5.3 illustrates components of a metamodel and their relation in functioning as a language, which are [107]:

- **Abstract syntax**, which defines metamodel conceptual coverage by defining a set of covered concepts and its relational structure, i.e., metamodel ontology.

- **Concrete syntax**, which declares a library of available representational forms, such as elements, primitives, or notations, along with the combination rules.

- **Semantics**, which provide interpretative translation, by relating each element (or combinations of elements) in the syntax to a meaning.
The concrete syntax is an important reference point for creating or understanding a model. The concrete syntax also differentiates graphical metamodel with its textual counterpart. Textual metamodels define its syntax in ‘word’ forms while the graphical metamodel uses visual forms, collected into a Graphical Concrete Syntax (GCS). GCS is theoretically specified in three parts [106]:

- **Graphical symbols**, specifying a library of notational symbols, including the option for embedding textual information for a notation label.

- **Compositional rules**, which defines the notation combinatorial rules and its feasible nested structure.

- **Visual semantics**, which provides a mapping between graphical symbols to the elements of the abstract syntax to define a meaning to a symbol, or a group of symbols.

Considering the superiority of a graphical over textual metamodel, this research focuses on the graphical metamodel. It should be noted that the difference between textual and graphical metamodels is not always clear. A narration written in natural language can obviously be categorized as a textual model, but some formalized declarative languages occasionally offer an option to arrange its elements in a spatial structure, e.g., partitioned box, associative line. In this case, exception is made to include these special types of textual metamodels.
5.1.4 Building Metamodel Landscape

In pursuing a cross-disciplinary engineering initiative, as in ‘service engineering’, a structured collection of relevant metamodel options might be beneficial. Several previous attempts have been made to collect metamodel offerings, but mostly limited to certain specific domains, e.g. service-oriented system [54], business process [108], or enterprise [109].

A structural map, as a navigational tool, to understand the relation between metamodels and their position in the modelling landscape is produced in this research. Detailed comparisons between metamodels has been discussed in other studies, but they were mostly performed between two metamodels, e.g. [110][111][112]. Only a few studies examine the relation between more than three metamodels, e.g. [113].

Metamodels collected in this chapter were gathered from a literature survey conducted in four disciplines: software engineering, e.g. [114], system engineering, e.g. [115], enterprise architecture, e.g. [116][117] and service engineering. Two interrelated sub domains were covered from the service engineering domain: business-side service engineering, e.g. [26], and informatics service engineering, e.g. [53]. Several additional disciplines were also inevitably traversed during the lateral exploration of metamodels, namely the Business Process Management and Financial Accounting.

Due to the selection process during the exploration, an exhaustive list of metamodel is not claimed in this chapter. An effort has been made
to ensure the representativeness of selected metamodels by cross-checking to additional sources. Informal emphasis was made toward a more recent and academically popular metamodel. A certain degree of emphasis was also made toward service engineering-related metamodels.

Rather than adopting a pre-defined structure for mapping the metamodel, an exploratory approach is taken, in which a structure was derived from the metamodel set [118][119]. Each selected metamodel is coded with two attributes: ‘scope’ and ‘keywords’. Scope relates to the type of underlying object commonly represented in metamodel usage. Keywords are assigned to provide certain characteristics to a metamodel based on its central themes.

5.1.5 Metamodel Landscape

In literatures, the term ‘model’ and ‘metamodel’ are often used loosely. Due to its strong correlation [120], metamodel often introduced with a prescriptive engineering method, forming a ‘framework’. In this case the name of the metamodel could be shared with its method counterpart. In other cases, the terms are also used interchangeably with a ‘reference model’ (or ‘conceptual’ model), in proposing an ontological structure (i.e., grouping and relating concepts) of a domain, akin to an abstract syntax definition.

To be included in the collection, the source is verified for the existence of a graphical concrete syntax specification. In most cases, a
formal definition is omitted, and the specification is presented in the form of examples. While this approach conveys the concepts and conventions in a practical term, the lack of formalism risks of multiple interpretations.

The exploration produces a list of 53 selected metamodels. At the beginning, the collection process strives for completeness, by covering both older and newer metamodels. Older metamodels are included not only for historical reasons, but also due to the fact that these metamodels have never been formally fully retired and are potentially still used by certain communities. The approach later shifted toward comprehensiveness by omitting metamodels decided to be redundant or too minor.

The ‘scope’ coding is drawn from several options of source: from stated objective in the original introduction, from explanatory example given, or from case studies performed. Two initial simplified codes for ‘scope’ are used: “software” and “business”. A model is either used to represent a software system (including its underlying IT components), or an (organizational) business system.

Later, a new type of metamodel is emerged that interchangeably cover both the software and business aspects of an organization. A third ‘scope’ is then introduced to categorize them as “enterprise” metamodels. A fourth minor scope also arises from certain type of metamodels which are not specifically tied to the two initial scopes, but rather to a generic definition of a system, covering even a physical system. This type of metamodel are then coded with ‘system’ label in their scope.
Some observed metamodels can be categorized as a ‘family’ of language, by offering several types of diagram to cover multiple aspects of its underlying object, e.g., class, component, collaboration diagram in UML. Asterix (*) symbol is added after the ‘scope’ code for metamodels with more than two types of diagrams.

The ‘keyword’ codes are assigned to describe the characteristic of a metamodel. It is arbitrarily derived from the core theme, important concept, or special unique aspect represented by the metamodel. To be useful as a grouping mechanism, the assignment of keywords is coordinated among metamodels to limit the variance of a keyword. Up to three keywords are coded to each metamodel, stated in descending order based on its importance.

To convey a structure of the metamodel landscape, the coded metamodels are presented both in tabular and graphical format under a pre-ordered arrangement [121]. To form a continuous structure, the tabular presentation is sorted in the order of the ‘scope’, followed by the ‘keyword’. As listed in table 5.1, the sorting is arranged from abstract concepts to more concrete concepts.

Graphical representation is created by focusing on the relationship between metamodels (figure 5.4). Relationship is drawn from three perspectives: (1) similarity, (2) evolution and (3) compatibility. Similarity is based on the coding assigned to metamodels. Similarity is visualized by positioning similar metamodels close to each other around a tag of shared code.
<table>
<thead>
<tr>
<th>Name</th>
<th>Curator</th>
<th>Year</th>
<th>Scope</th>
<th>Keywords</th>
<th>Ref</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. GoalML (Goal Modelling Language)</td>
<td>-</td>
<td>2014</td>
<td>Enterprise*</td>
<td>goal, actor, structure</td>
<td>[122]</td>
</tr>
<tr>
<td>2. MEMO (Multi-perspective Enterprise Mod)</td>
<td>-</td>
<td>2011</td>
<td>Enterprise*</td>
<td>goal, process, organization</td>
<td>[123]</td>
</tr>
<tr>
<td>3. KAS (Kowli, Acquisition in Automated Spec.)</td>
<td>-</td>
<td>1991</td>
<td>Enterprise</td>
<td>goal, structure</td>
<td>[124]</td>
</tr>
<tr>
<td>4. GRL (Goal Oriented Requirement Lang.)</td>
<td>ITU-T</td>
<td>2003</td>
<td>Enterprise*</td>
<td>goal, structure, distribution</td>
<td>[125]</td>
</tr>
<tr>
<td>5. &quot;i&quot; (Distributed Intentionality)</td>
<td>-</td>
<td>1995</td>
<td>Enterprise</td>
<td>goal, structure, distribution</td>
<td>[126]</td>
</tr>
<tr>
<td>6. UML (Unified Enterprise Mod. Lang.)</td>
<td>Interop-NoE</td>
<td>2002</td>
<td>Enterprise</td>
<td>goal, structure, ontology</td>
<td>[127]</td>
</tr>
<tr>
<td>7. ODM (Ontology Definition Metamodel)</td>
<td>OMG</td>
<td>2009</td>
<td>Enterprise*</td>
<td>ontology, concept, rule</td>
<td>[128]</td>
</tr>
<tr>
<td>9. SoaML (Service oriented archit. Mod. Lang.)</td>
<td>OMG</td>
<td>2009</td>
<td>Enterprise*</td>
<td>service, architecture</td>
<td>[60]</td>
</tr>
<tr>
<td>10. SOMF (Service Oriented Modelling Framework)</td>
<td>Meth. Corp.</td>
<td>2008</td>
<td>Enterprise*</td>
<td>service, architecture</td>
<td>[59]</td>
</tr>
<tr>
<td>12. E-BMM (Enterprise BMM)</td>
<td>-</td>
<td>2009</td>
<td>Enterprise</td>
<td>strategy, business, application</td>
<td>[130]</td>
</tr>
<tr>
<td>13. BMM-IT* (Business Motivation Model extension)</td>
<td>-</td>
<td>2013</td>
<td>Enterprise</td>
<td>strategy, structure, alignment</td>
<td>[131]</td>
</tr>
<tr>
<td>14. BMM (Business Motivation Model)</td>
<td>OMG</td>
<td>2008</td>
<td>Business</td>
<td>strategy, goal, influencer</td>
<td>[132]</td>
</tr>
<tr>
<td>15. b-SOAML (Business SOAML)</td>
<td>-</td>
<td>2012</td>
<td>Business</td>
<td>architecture, service</td>
<td>[133]</td>
</tr>
<tr>
<td>16. BMM Cube (Business Model Cube)</td>
<td>NEFFICS</td>
<td>2013</td>
<td>Business</td>
<td>model, innovation</td>
<td>[134]</td>
</tr>
<tr>
<td>17. BMC (Business Model Canvas)</td>
<td>-</td>
<td>2010</td>
<td>Business</td>
<td>model, innovation</td>
<td>[85]</td>
</tr>
<tr>
<td>18. S-BMC (Service Business Model Canvas)</td>
<td>-</td>
<td>2014</td>
<td>Business</td>
<td>model, innovation, service</td>
<td>[135]</td>
</tr>
<tr>
<td>20. SDBM (Service Dominant Business Model)</td>
<td>-</td>
<td>2014</td>
<td>Business</td>
<td>model, innovation, service</td>
<td>[136]</td>
</tr>
<tr>
<td>21. CBM (Component Business Model)</td>
<td>IBM</td>
<td>2005</td>
<td>Business</td>
<td>capability, structure</td>
<td>[94]</td>
</tr>
<tr>
<td>22. CM (Capability)</td>
<td>-</td>
<td>2011</td>
<td>Business</td>
<td>capability, structure</td>
<td>[137]</td>
</tr>
<tr>
<td>23. VDML (Value Definition Modelling Lang)</td>
<td>OMG</td>
<td>2015</td>
<td>Business*</td>
<td>value, capability</td>
<td>[138]</td>
</tr>
<tr>
<td>24. e3-value (Economic e-Commerce Value)</td>
<td>-</td>
<td>2001</td>
<td>Business</td>
<td>value, exchange</td>
<td>[139]</td>
</tr>
<tr>
<td>25. Value net</td>
<td>-</td>
<td>2002</td>
<td>Business</td>
<td>value, flow</td>
<td>[140]</td>
</tr>
<tr>
<td>26. ServiceML (Service Modelling Lang.)</td>
<td>SINTEF</td>
<td>2013</td>
<td>Business*</td>
<td>service, goals, interaction</td>
<td>[141]</td>
</tr>
<tr>
<td>27. SJML (Service Journey Modelling Lang.)</td>
<td>SINTEF</td>
<td>2013</td>
<td>Business</td>
<td>service, interaction, customer</td>
<td>[142]</td>
</tr>
<tr>
<td>28. CJM (Customer Journey Map)</td>
<td>IDEO</td>
<td>1999</td>
<td>Business</td>
<td>service, interaction, customer</td>
<td>[89]</td>
</tr>
<tr>
<td>29. PCN (Process Chain Network)</td>
<td>-</td>
<td>2012</td>
<td>Business</td>
<td>service, interaction, value</td>
<td>[143]</td>
</tr>
<tr>
<td>30. SBP (Service Blueprint)</td>
<td>-</td>
<td>1984</td>
<td>Business</td>
<td>service, process, interaction</td>
<td>[68]</td>
</tr>
<tr>
<td>31. POA (Possession-Ownership-Availability)</td>
<td>-</td>
<td>2015</td>
<td>Business</td>
<td>process, value, exchange</td>
<td>[144]</td>
</tr>
<tr>
<td>32. Value stream</td>
<td>-</td>
<td>1998</td>
<td>Business</td>
<td>process, value</td>
<td>[145]</td>
</tr>
<tr>
<td>33. SBVR (Semantics of Business Vocab. &amp; Rules)</td>
<td>OMG</td>
<td>2008</td>
<td>Business</td>
<td>ontology, rule</td>
<td>[146]</td>
</tr>
<tr>
<td>34. TDM (The Decision Model)</td>
<td>KPI</td>
<td>2009</td>
<td>Business</td>
<td>decision, rule</td>
<td>[147]</td>
</tr>
<tr>
<td>35. DMN (Decision Model and Notation)</td>
<td>OMG</td>
<td>2015</td>
<td>Business</td>
<td>decision, rule</td>
<td>[148]</td>
</tr>
<tr>
<td>36. CMMN (Case Management Model and Notation)</td>
<td>OMG</td>
<td>2009</td>
<td>Business</td>
<td>case, structure, state</td>
<td>[149]</td>
</tr>
<tr>
<td>37. BPDM (Business Process Def. Metamodel)</td>
<td>OMG</td>
<td>2008</td>
<td>Business</td>
<td>process, collaboration</td>
<td>[150]</td>
</tr>
<tr>
<td>38. BPM (Business Process Model and Notation)</td>
<td>OMG, ISO</td>
<td>2004</td>
<td>Business</td>
<td>process, flow</td>
<td>[151]</td>
</tr>
<tr>
<td>40. REA (Resource-Event-Agent)</td>
<td>-</td>
<td>1982</td>
<td>Business</td>
<td>process, structure, ontology</td>
<td>[153]</td>
</tr>
<tr>
<td>41. UML (Unified Modelling Lang)</td>
<td>OMG, ISO</td>
<td>1997</td>
<td>Software*</td>
<td>object, function, behaviour</td>
<td>[56]</td>
</tr>
<tr>
<td>42. YAWL (Yet Another Workflow Lang.)</td>
<td>YAWL Found.</td>
<td>2002</td>
<td>Software</td>
<td>process, flow</td>
<td>[154]</td>
</tr>
<tr>
<td>43. DFD (Data Flow Diagram)</td>
<td>-</td>
<td>1974</td>
<td>Software</td>
<td>process, flow, data</td>
<td>[97]</td>
</tr>
<tr>
<td>44. IFML (Interaction Flow Modelling Lang.)</td>
<td>OMG</td>
<td>2015</td>
<td>Software</td>
<td>interface, flow,user</td>
<td>[155]</td>
</tr>
<tr>
<td>45. SC (Structured Chart)</td>
<td>-</td>
<td>1988</td>
<td>Software</td>
<td>flow, structure</td>
<td>[156]</td>
</tr>
<tr>
<td>46. ERD (Entity Relationship Diagram)</td>
<td>-</td>
<td>1976</td>
<td>Software</td>
<td>data, structure, ontology</td>
<td>[157]</td>
</tr>
<tr>
<td>47. SysML (Systems Modelling Lang)</td>
<td>OMG, ISO</td>
<td>2006</td>
<td>System*</td>
<td>block, functional</td>
<td>[158]</td>
</tr>
<tr>
<td>48. IDEF (Integration Definition)</td>
<td>KBSI</td>
<td>1995</td>
<td>System*</td>
<td>function, data, process</td>
<td>[159]</td>
</tr>
<tr>
<td>49. Flowchart</td>
<td>ASME</td>
<td>1947</td>
<td>System</td>
<td>process, flow</td>
<td>[160]</td>
</tr>
<tr>
<td>50. Statechart</td>
<td>-</td>
<td>1987</td>
<td>System</td>
<td>state, transition</td>
<td>[161]</td>
</tr>
<tr>
<td>51. Petri net</td>
<td>-</td>
<td>1962</td>
<td>General</td>
<td>flow, queue</td>
<td>[162]</td>
</tr>
</tbody>
</table>
The evolutive relationship is based from the existence of influence from an older metamodel in the creation of a new metamodel. In some cases, the new metamodel made the influencing metamodel obsolete. In other cases, the influencing metamodel is only taken as an inspiration from which both metamodels survive and compete. The influencing relation might be explicit, i.e. stated by the creator in the original proposal, or implicit, i.e. assumed or implied by a third-party source.

The evolutive relation is visualized by a directed line toward the influenced metamodel. An explicit influence is symbolized with a solid line, while the implicit with a dashed line. In both cases, the evolutive relationship suggests an intersection of concept coverage, another form of characteristic similarity.
The third relationship, compatibility, is taken from a conception that a pair of metamodels is compatible and complementary. This claim can be identified from the metamodel’s original source or from a separate proposition. A compatibility relation is visualized by a thick shaded line. A compatibility relationship infers a degree of distinction between the pair of metamodels.

5.1.6 Metamodel Grouping

Positioning the metamodels based on its codes and relations provides an opportunity to group the metamodels based on its similarity. An insight for metamodel grouping emerges from correlating the tabular and graphical presentation, as presented in figure 5.5. From this insight, seven metamodel stereotype groups are defined. Each group is named according to the dominant code in the group: (1) goal, (2) enterprise, (3) business model, (4) service, (5) process, (6) software, and (7) system.

Group 1: Goal Metamodels

The group is characterized by emphasize on the concept of “business goals”. Some of the group metamodel (e.g. KAOS, i*, GRL, GoalML) were proposed within the context of requirement engineering activity during a specific software engineering project. But since these metamodels departed from the business goals and their distribution within the organisational structure, they fall into an “enterprise” scope and “goal” group.
Figure 5.5: Metamodel Grouping
The group is dominated by metamodels produced from the IT discipline departing from software requirement engineering, as exemplified by the pair of prospective dominating metamodels: GoalML and MEMO. MEMO is quite popular and still evolving in collecting other metamodels, including GoalML. MEMO is actually a metamodel proposed with an “enterprise” context in mind. Its strong emphasize on the “goal” concept puts it in this group. This actually illustrates an important relationship between enterprise modelling and goal modelling.

**Group 2: Enterprise Metamodels**

This group is the first among three that takes its label from its coded “scope”. Despite the claim that the enterprise engineering field is an amalgamation of information science and organization science [163], the group is largely dominated by products from the first academic discipline. In a moderate pace, the group is quite prolific in producing metamodels. More enterprise metamodel specifications can be expected from competing frameworks, such as CIMOSA, GERAM, ISO-19439 and DEMO[164].

The group is characterized by its wide coverage, traversing both business and technical (software) aspect of an organization. While not always detailed, it can encompass multitude of aspects, to include potentially all other aspects within the modelling continuum. Due to this, the group tends to produce ‘big’ metamodels, with a vast library of notations and views.
Some enterprise metamodels exclude (goal) motivational aspects, e.g. ARIS, the original Archimate, while others include them (e.g. UEML, Archimate version 2.1 onward). The group also includes the service-oriented metamodels, i.e. SOMF and SoaML, with a limited popularity.

Probably due to the coverage option range, the group still lacks a dominant metamodel. So far, OMG has not published a specification for this specific group. UEML was proposed academically to address the “Tower of Babel” situation in the metamodel offering[165]. But its development seems to have subsided in the last decade, only to be eclipsed by newer offerings such as MEMO. Archimate, which is based on the Open Group’s TOGAF framework, actually demonstrates a robust quality, but it is still not academically popular enough to be regarded as a dominant enterprise metamodel.

**Group 3: Business Model Metamodels**

This specific group is characterized by its attempt to convey a structure of “money-earning logic”. It typically covers customer segmentation, market positioning, product innovation, and relates to the internal supporting structure. Due to this fact, the metamodels proposed in this group were originally produced from non-computer science disciplines, i.e., business, marketing and management disciplines. OMG also contributed a partial coverage on this group with BMM specification, emphasizing on the organizational motivation aspect, including the “goal” aspect.
Despite its specificity, the group can be considered to be dynamic. The dominant metamodel BMC becomes the basis for many other metamodel propositions. A re-factoring of BMC was proposed as BMI-Cube. A new perspective from service-orientation extends BMC into s-BMC, SDBM and SOBM, which respectively: (1) Elaborates the component of customer and partner, (2) considers multiple parties involvement and (3) adds a service repository concept.

The group also covers a minor theme of business “capability” and its structure as exemplified by CM (Capability Map) and CBM (Component Business Model). The concept of “capability” itself is linked to the concept of “value”, “service” and “process”. With this fact, the group is expanding and linked with contribution from other domain disciplines, including service science and enterprise engineering.

**Group 4: Service Metamodel**

As the most fluid group, service metamodels stereotype actually a loose confederation of several concepts coverage, i.e. “value”, “interaction”, and “service”. The non-technical side is covered by the concept of “value-exchange” or “value-interaction”, which is linked with the “business model” group. In the technical side the concepts are linked to the “software-service” as in SOA approach.

The concept of “interaction” is covered from several perspectives. Interaction in the sense of “value-network” or “value-exchange” is depicted by value-network metamodels (e.g. e³-value). The interaction can
also be conducted in inter-organization level, (e.g., B-SoaML), in customer
level (e.g., Service Blueprinting, CJM, SJML), or both (e.g., ServiceML,
PCN).

The contribution to the group is shared between the non-technical
domain of Service Science (e.g., Service Blueprint) and Service Operation
(e.g., PCN) and the technical domain, such as enterprise engineering (e.g.,
B-SoaML). This group can thus be considered as another integrative
concept as in “enterprise”, but from a specific perspective of a “service”.

As a relatively recent metamodel, VDML specification was
envisioned as an integrative metamodel which cover both the concept of
“value” and “capability”. From its introduction by OMG in 2015, it can be
expected to become a future dominant metamodel, potentially eclipsing
other “value” and “capability” related metamodels. But VDML lack of
detailed customer level “interaction” coverage, as abstracted in Service
Blueprinting, PCN, CJM, and SJML, makes it incomplete to cover the
whole “service” group. From the OMG perspective, the concepts covered
by these customer level interactions might be alternatively produced from
“process” modelling, i.e. BPMN.

**Group 5: Process Metamodels**

This group is characterized by the concept of a “process” which involved
both the terms “activity” and “workflow”. The group shares its origin from
both computer science and management science [166]. REA is an example
of proposition from non-computer related disciplines, i.e. accounting, with
an added “ontology” perspective. EPC is one example of offering from computer science discipline. As one the oldest group in the structure, the group therefore holds a vast library of metamodel offerings.

The introduction of BPMN by OMG in 2004 stabilizes the dynamic aspect of the group which leads to the establishment of BPMN as the dominant standard for “process” modelling. An attempt by OMG to formalize and extend BPMN with BPDM proved to be unpopular which led to the folding of the BPDM concept into the next iterative version of BPMN. Later, a variant of process modelling was also introduced by OMG in the form of CMMN for a specific type of consultative process, i.e., “case”, which typically involves knowledge workers. Other specific variants of process modelling were also introduced by OMG to structuring a decision process as DMN. These last two metamodels highlight additional aspects covered by this group: “decision”, “rule” and “ontology”. The three OMG’s process model (BPMN, CMMN and DMN) are actually envisioned to complement each other during process modelling.

The “process” model can be viewed as the core element in the global continuum of the modelling concept. It is upward linked with goal, enterprise, service and business models, and downward linked to software models. This notion can also be observed from the fact that, BPMN is demonstrated to have a compatibility relationship with all other OMG metamodels.
**Group 6: Software Metamodels**

As the second stereotyped group named after its scope, the group has specific context in modelling a “software” system. The contributing domains are therefore specific from computer science and software engineering disciplines. With its long tradition of modelling, the domains evolved its engineering approach in a certain pace, along with introduction of improved set of specific modelling technique. Combining the factors of historical evolution and the multitude of software aspect, the group possess a myriad of metamodel options. The arrival of UML in the late 90’s altered this landscape.

While other older metamodels may still survive, UML has become the *de-facto* metamodel in Software Engineering. While its adoption in the industry might be debatable [167], UML offers two advantages over other metamodels. First, UML is formalized into a metamodel architecture by having a meta-metamodel parent: Meta-Object Facility (MOF), to which other OMG’s metamodels are aligned. This way, UML is theoretically convertible to any other metamodel based on OMG’s MOF. Secondly, UML is extendable, that is a new metamodel can be created based on it for a specific purpose by defining an UML profile. Some examples of this are the SoaML and SysML. While not formally stated, Archimate can also be observed to possess notational likeness with UML.
Group 7: System Metamodels

The last group also takes its name from the scope: “system”. The group holds together the older class of metamodels and often serves as the source of influence to other newer metamodels. The group consists of a general purpose metamodel commonly use to describe a system, in a generic context, often as a physical or a complex system [115].

Categorized within this group, IDEF is actually a family of metamodels, originally envisioned to range from IDEF0 to IDEF14, in covering enterprise wide concepts such as data, process, business and network. In its implementation only several of them are fully developed [168]. Despite its adoption by IEEE in 1999, IDEF has a very limited adoption beyond its original intended usage within United States Department of Defense (DoD).

The group can also be considered stable after the introduction of SysML by OMG. Based on International Council on Systems Engineering (INCOSE) proposal, SysML extends UML by simplifying and modifying it into seven types of diagram and introducing two new diagrams to produce a metamodel for physical system, e.g. machinery or aircraft.

It is worth to note that the boundary between these stereotyped groups is not a clear-cut line. Due to often interlinked concept coverage in each group, a metamodel can potentially be categorized to belong to two or more groups. Nevertheless, the proposed grouping structure produces
an identification of important concepts covered by each group and the relationship between the groups and the concepts.

5.1.7 Metamodel Proliferation Pattern

From these results, three evolutive patterns of metamodel proliferation can be observed. The first pattern is the introduction of an entirely new metamodel, on previously unchartered area or taking a unique perspective on modelling the underlying object, with relatively minimal influence from previously available metamodels. Examples of these pioneering metamodels are: Flowchart (1947), Service Blueprinting (1984), i* (1995), REA (2005), TDM (2009) and BMC (2010). The BMM (2008) is a rare example of original metamodel proposed by OMG. In a lesser degree, OMG’s SBVR (2008) can also be seen as an original metamodel, but two facts diminish its originality: It was built based on the MOF specification, and it has some influences from declarative languages (i.e., CL and OWL). These pioneering metamodels could also be seen as a departure point, upon which a group of similar metamodels flourished.

The second pattern is the introduction of a new metamodel which is explicitly based or implicitly inspired from previously available metamodels. The recent metamodel proliferation is due to metamodels produced based on this pattern. The new proposition can take the same name of the original metamodel, with version number, or with an entirely
new name. There are at least three motives underlying the introduction: (1) improving the detail quality, (2) enlarging the coverage, and (3) adapting to a specific context.


The third pattern is the integrative attempt in which different metamodels are formally collected to become a family. This pattern can be typically observed from OMG metamodels, e.g., UML (1997), BPMN (2004), ODM (2009), and VDML (2015). The integrative attempt can either adapt the source metamodel as it is (e.g., ODM), or add some modifications (e.g., VDML) to allow a better compatibility between its components. The pattern can also be seen outside of OMG metamodels, such as ServML (2012) in service modelling, UEML (2002) and MEMO (2011) initiatives in enterprise modelling.

These patterns illustrate the importance of visibility for a metamodel to be considered as a significant contribution to the competitive landscape of metamodel offering. For obvious reason, the early-pioneering metamodel has a better chance to become a significant
metamodel. But a lack of a continuous supporting interest could also make a metamodel irrelevant and obsolete. A certain level of academic discussion and, most importantly, industrial adoption are required for its survival. An adoption by an authoritative curator body (e.g., OMG, ISO) is an important factor in overseeing metamodel usage beyond its initial inception.

5.1.8 Discussion on Metamodel Landscape Structure

Some limitations are observed from the metamodel exploration results. First is the missing occurrence of supporting information system components, such as the abstraction of software modular structure (e.g., software service, software component), computing node (e.g., physical or virtualized server) and its connectivity pattern (e.g., network). A justification can be made that these components only become important in later stages of the design phase, toward implementation modelling. Nevertheless, these components could still be considered as important concepts due to the fact that these concepts are actually covered by some metamodels in software and enterprise modelling. To bring out these concepts, a more refined coding approach, toward individual diagrams level of a metamodel family (e.g. UML), is required. This refined approach is decidedly out-of-scope for this research.

The second limitation is observed from the odd recurrence of “ontology” in the structure. Ontology is mentioned in the three scopes:
software, business, and enterprise contexts. This suggests that the concept of “ontology” persists in multiple contexts. This notion is supported by both Zachman [116] and TOGAF [117] enterprise architecture framework. In this case, confronting the emerged results with a predefined formal structure could be considered for further improvement of the result. A richer structure might also emerge from analysing the metamodels in its individual diagram level.

5.2 Service Engineering Ontology and Artefact

To verify the completeness of framework coverage on the aspects of service system, a comparative triangulation is made between the produced service ontology with artefacts defined in the framework. This cross-referencing into the proposed metamodels also serves as a bridge to characterize the artefacts form.

The assessment is performed in four parts divided by sub-stage, i.e. activity, from the latest iteration of proposed framework (figure 5.6) : (1) Understanding Service Context, coded as activity 1, (2) Defining Service Concept, coded as activity 2, (3) Business Service Design, coded as activity 3, and (4) Software Service Design, coded as activity 4.
5.2.1 Artefacts in Activity 1 (Understanding Service Context)

Before proposing new or improved services, an understanding toward the context is required. The activity in the first part of the identification stage captures and analyses the existing situation of the environment.

The activity covers foundational aspects of an organization. As illustrated in figure 5.7, four existing-situation aspects are captured as artefacts in this sub-stage: (1) business directives, (2) business model, (3) business process, and (4) business capability. The framework proposes to capture the guiding business directives as a list of narratives, which can be presented in tabular format. The current business model is visualized with BMC while the business process with BPMN.
The currently owned and potentially owned capabilities of the organization are also examined and presented in a capability diagram. The analysis and modelling could be based on Component Business Model (CBM), which is based on organization structure, or based on SoaML capability diagram.

The result of this sub-stage should be an identification of opportunities to be pursued in provisioning a business service. The opportunity could be numerous. Therefore, a ranked list should be made based on combination of various factors such as feasibility, prospective gain and cost, or resources required.

In identifying the opportunity, external perspectives must also be incorporated. These external perspectives should capture the market opportunity and business partnership. In the framework, the combination of outward and inward-looking perspective is accommodated in the last
artefact, the opportunity list. The framework does not specify the standard format for the list, but it is usually in a narrative format produced from business analysis techniques, such as a SWOT or Value Chain Analysis. Any format should be acceptable as long as it helps the management to decide a specific opportunity to pursue.

5.2.2 Artefacts in Activity 2 (Defining Service Concept)

The second part of the identification stage is a business-side elaboration of the selection decisions made in the first part. The activity produces a high-level view of the service to be design and implemented.

Figure 5.8: Artefact Ontological Position in (Activity 1.2)

Figure 5.8 shows three (to-be) aspects to be covered: (1) Business Model, (2) Service Values and Goals, and (3) Business Service definition. The targeted business model is formalized as an artefact configuring the business concepts in terms of the BMC components, e.g. partnership, supporting activities, customer segment, channel, and others.
The service goal and value elaborate the value components of BMC by declaring the objectives and proposed values of targeted service, as directives in identifying service features and designing service processes. The format for service goals and value artefact should be a simple numbered table listing the objectives and values.

The final artefact of this sub-stage is a business service catalogue. This artefact is simply a formal catalogue of business services to be provided, in term of roles provided in the service with specific service features derived from service objectives and values. The table presentation of the artefacts could be combined with the list of service objectives and values to provide traceability between the service goals and features.

5.2.3 Artefacts in Activity 3 (Business Service Design)

The third activity delves into the design stage by detailing the mandates set by the previous stage. Four business-service aspects are covered, as described in figure 5.9: (1) Service Architecture, (2) Service Interaction, (3) Service Process, and (4) Business Ontology.
Service architecture visualizes a global collaboration relation between participants of the service community. The pairing roles of provider and consumer are a basic form of the architecture, but the relation might be connected with multiple service options. The artefact is particularly important for a service system which involves more than two parties or roles within the service scope. SoaML service participant diagram format is ideal to present this artefact, by relating the component of: participants, roles, and services.

Service interaction artefacts specify the touch point between a consumer and the providing participants throughout the cycle of service provision. The model focuses on the description of the process flow performed by consuming parties. The specification covers type of channel, interfacing mode, and specification of resource exchanged, e.g. document or information. Service Blueprinting (SBp) technique is suggested as a format for this artefact. Interaction rules, e.g. operational hours, pre-
requisite service states can be specified in the form of SoaML service contract, accompanying a SoaML service architecture.

The process model specifies the flow of activities, mostly in providing participants, covering through its collaboration with the co-providers. Special attention is given to the atomic abstraction of the activity tasks with interactivity feature: service-operation. These operations are the potential baseline for (software) service definition [77]. The artefacts are formatted in the de-facto format of business process metamodels: BMPN.

The fourth artefact to be produced lies in the ontology engineering context, in defining the business ontology model [53][82][10] as part of (service) ‘product model’ [31]. The ontology artefact should cover ontology components related to business models and service system, as a part of the whole business domain ontology. The artefact can be presented in UML class diagram.

5.2.4 Artefacts in Activity 4 (Software Service Design)

The fourth activity mirrors the activity in the ‘Business service design’ sub-stage. The difference is that the service elaborated in an IT context, i.e. software context, rather than in a business context as in the previous sub-stage. Figure 5.10 lists four aspects to be covered from the ontology at the software-service level: (1) Atomic service, (2) Composite service, (3) Service Detail, and (4) Service Information.
The *atomic service* specifies the design of a self-sufficient software service in terms of the service interface, contained operations and its underlying behaviour. The artefact is presented as diagrams of SoaML Interface.

The *composite service* describes the combined use of the atomic service in the form SoaML Service Interface which includes the behaviour in the form of a sequential arrangement of operations invocation. In the case of a composite service, it contains invocations to external services, i.e. services provided by other participants, the behaviour specification represents a software level collaboration-interaction with an external software component.

The *service detail* aspect gathers the software services into an abstracted form of software components with a service port, invoked and
invokable services into a SoaML Participant. All service behaviours are to be detailed in this container, to be implemented later as software components. These components serve as a representation of an interacting party, either as a consumer, as a provider or both.

Finally, the service information collects all of the information exchanged between services as operands and return values of invoked operations. SoaML message type diagram is defined for each service interaction transaction and cross-referenced with business information artefacts from previous activity to be standardized in maintaining consistency while at the same time facilitate message type reusability.

In general, the described triangulation between ontology and framework artefacts demonstrates a sufficiency of framework coverage in assessing service aspects and components, both in business and software perspective. In this stage, the prescribed metamodel of targeted artefacts is an open specification. The suggested metamodels are demonstrated to be sufficient for the case studies. But the dynamic nature of the metamodel landscape may offer alternative formats that might be better in capturing the modelling needs.

5.3 Alternative Service Engineering Metamodels

The first part of this chapter presents an emerging structure of a metamodel landscape from a grounded approach. While a clear differentiation between metamodel groups is not claimed, seven
stereotypes of metamodel are offered: (1) goal, (2) enterprise, (3) business model, (4) service, (5) process, (6) software, and (7) system.

The “service” perspective emerges as an alternative integrative approach, as in “enterprise” perspective, in traversing the context between “business” and “software” aspects. The “service” perspective covers the aspects of “business model”, “business capability”, “business interaction”, “value proposition”, “value exchange”, “customer interaction”, and “software service”, which is consistent with the produced Service Engineering Framework.

The observed proliferation pattern of metamodel projects its nature as an ever-dynamic landscape. Some cross-disciplinary initiatives might pursue an integrative universal metamodel. But a more pragmatic and feasible option is available in the form of specifying a metamodel stack [65][62], such as adopted by the framework of this research. As a set of originally unrelated metamodels, special care must be taken to ensure the traceability and translatability of the artefacts between stage and activity.

Examining the metamodel landscape, several newer metamodel propositions are worthy to be proposed as metamodel alternatives. The following section presents these potential metamodel alternatives divided into four sub sections: (1) OMG’s Business Motivation Model (BMM) [132], (2) OMG’s Value Definition Modelling Language (VDML) [138], (3) alternative Business Modelling, and (4) alternative Interaction Modelling. These alternatives are proposed here as potential options. For an actual
use, another set of case-study is required, which is beyond the scope of this research iteration.

5.3.1 Business Motivation Model

As the name implied, Business Motivation Model (BMM) [132] covers motivational aspects of a business case. It is situated on the strategic level of a business model by defining the drivers, the element and its interrelations for a business plan, without elaborating the detailed aspects of business process and business structure.

![Diagram of Business Motivation Model](image-url)

Figure 5.11: Sample of Business Motivation Model Artefact [169]

From the illustration in figure 5.11, BMM can be seen as an ontological structure for business motivation, which relates to two aspects: (1) Ends, defined as situations to be achieved, i.e. goals and objective, and (2) Means, as concepts adopted to achieve the ends, i.e. strategies, tactics, policies, and rules.

Not many published articles are found documenting BMM adoption for real world cases, but the recent update to the specification introduces metamodel notations for modelling purposes[169]. BMM metamodel can
be useful for structuring a business goal artefact in the framework. If needed, BMM is useful to document the traceability between the components of goals, objectives, strategies, tactics, policies, and rules, as a directive context for a service system.

5.3.2 Value Definition Modelling Language

Value Definition Modelling Language (VDML) [138] is a relatively recent metamodel to be introduced by OMG. It covers business concepts in terms of activities, roles, flows, participants and capabilities in a higher abstraction compared to BPMN. VDML is proposed as a modelling language for business analysis with focus on value creation and exchange, by combining external perspective on market opportunities with extended organisational capability structure.

Like UML and SoaML, VDML is actually a family of diagrams, which contains eight types of diagram: (1) Role Collaboration, (2) Value Proposition Exchange, (3) Activity Network, (4) Collaboration Structure, (5) Capability Library, (6) Capability Heat Map, (7) Capability Management, and (8) Measurement Dependency. The detail specification of these diagrams is out of scope for this chapter, but it suffices to identify the components relevant to the framework, with comparison to existing metamodels.

*Role collaboration* and *Value proposition exchange* describes service architecture, as a network of providers and consumers, in term of
participant role and value (potentially) exchanged. For this purpose, SoaML’s Service Architecture is decided to be sufficient to present the similar abstraction, with a compact abstraction of participants, roles and service interactions.

On the other hand, VDML is quite attractive to represent the concept of capability in the framework. Among three of its capability related diagrams, two are identified to have potential use in the framework: (1) capability Library and (2) capability management.

![Figure 5.12: Sample of VDML Capability Library [138]](image)

As can be seen in figure 5.12, the capability library provides a hierarchical structure of capabilities in an organization, which could be useful to replace the use of a Component Business Model (CBM) in the capability modelling in identification stage.

The capability management provides a graphical abstraction for ownership, dependency and exposition of capability within an organization or an extended organization (figure 5.13). It has similarity
with SoaML’s participant diagram in software-services but resides in the ‘capability’ context. VDML capability management diagram is also identified to be a potential metamodel for a capability model artefact, as it has the features to accommodate an extended organization and business model.

![Diagram](image)

Figure 5.13: Sample VDML Capability Management [138]

### 5.3.3 Business Modelling

Business modelling is a growing research area which flourished since the introduction of BMC in 2010. In the metamodel exploration, three business modelling languages are identified to be potentially relevant for the framework: (1) Service Dominant Business Model (SDBM) [136], (2) Business Model Innovation Cube (BM Cube) [134], and (3) Service BMC (S-BMC) [135].

SDBM offers a simple view of service business model with only four components [136]: (1) Service as the core element, (2) Management, representing the ‘how’ aspect of service access, analogous with relationship and channel components in BMC, (3) Benefit-Cost,
characterize the service value for specific participant in mostly financial context, and (4) Actor as providing or consuming participant of the service. All of the components are visualized as layers circling a specific service in the centre, as illustrated in figure 5.14.

Figure 5.14: Sample of Service Dominant Business Model (SDBM) [136]

SDBM strength lies in its simplicity in abstracting a service with multiple participants. But compared to the standard BMC, SDBM lacks the specification of activities and resources involved in the service provision. Despite its limitation, SDBM is still an attractive format as an early form of the service business model, describing a preliminary architecture of service participants.
BM Cube [134] is a reformulation of BMC in re-combining several of the components into seven components, forming the sides and the centre of a cube (figure 5.15), which are: (1) Value proposition, (2) user and customer, (3) internal value chain, analogous with BMC’s channel and activity, (4) competence, represents BMC’s resource, (5) networks, for BMC’s partner and channel, (6) relation, as in BMC’s relationship, and (7) value formula, combining BMC’s cost and revenue components.

The proposed cubical presentation has no practical benefit, but BM Cube represents a trend toward an Open Business Model [134], which features an explicit abstraction for an extended business network. The offered simplification of BM Cube might be useful for some cases, but its sufficiency for a real-world case still needs to be further examined.

Service BMC (S-BMC) is another reformulation of the BMC format by extending its usage for multi-party business models. In S-BMC, seven BMC components are spread vertically, and each participant’s perspective are specified as layers of these component.
Three perspectives are offered in its basic format: (1) customer perspective, (2) internal organization perspective, and (3) partner perspective (figure 5.16). Additional layers might be added for other business participant, as intermediaries either toward the customer or partner side (figure 5.17).

![Figure 5.16: Structure of Service BMC (S-BMC)[135]](image)

![Figure 5.17: Sample of S-BMC Artefact with five party layers [135]](image)

It can be observed that the three alternatives business model formats try to address BMC limitation in representing an extended organization, as a service system formed by multiple participants. BMC or BMI format is fairly sufficient for simple business cases with one
dominant providing participant. But to represent complex business model architecture, SDBM or S-BMC might be required.

5.3.4 Interaction Modelling

Four interaction modelling formats are identified in metamodel exploration: (1) Process Chain Network (PCN) [143], (2) Service Modelling Language (Service ML) [141], (3) Service Journey Modelling Language (SJML) [142], and (4) Customer Journey Map (CJM) [89].

PCN is proposed in service operation management field as an attempt to improve Service Blueprinting (SBP) [143]. PCN focuses on the touch point by introducing three degree of interaction layer for each party, i.e. provider and consumer: (1) direct interaction, (2) surrogate interaction, and (3) independent processing.

![Figure 5.18: Sample of Process Chain Network (PCN) Artefact [143]](image-url)
The presentation has similar feature with SBP by defining participant activities in the interaction, but the elaboration is not only in the provider side but also accommodated in the consumer side (figure 5.18). PCN presentation also provides an abstraction of business-process networks, by aligning series of interactions for multiple service participants (figure 5.19). In this sense, PCN can be seen as an alternative improvement of SBP for multiple participants’ interaction, without separate components for exchanged artefacts such as physical evidence in SBP.

ServiceML [141] was proposed in a similar manner with VDML, as a family of diagrams collecting representations of ‘service’ concepts. Five types of diagram are defined, as summarized in figure 5.20:
1. **Needs model**, as a diagram relating customer needs with required service features.

2. **Service Architecture**, as a simplification of SoaML’s Service Architecture connecting participant and service (contract).

3. **Actor Network**, as a detailed version of Service Architecture with participant role and individual flow of sequential interaction.

4. **Service Journey Map** (SJM), as a graphical representation of a series of touch-points experienced by customer throughout the cycle of service provision.

5. **Service Experience Journey Map**, a similar form of SJM with colour-code representing expected customer (emotional) experience.

![Figure 5.20: Diagrams in ServiceML](image)
Each of these diagrams has usage potential for the framework. The needs model may be used to relate service goals (artefact 1.2.2) and service features in business catalogue (artefact 1.2.3). Service architecture simplification, which was also introduced as part of b-SoaML [133], might replace SoaML’s format for artefact 2.1.1. The details of Actor Network is more appropriate in the later stage, such as for accompanying the Interaction model (artefact 2.1.2).

The actual interaction model is offered in the form of SJM. The diagram focuses on the touch-points from the consumer side and a suitable format for the interaction model (artefact 2.1.2). This interaction model describes a ‘service path’ [23], as a series of service encounters while at the same time reflect service states. The omission of supporting back end activities avoids a coverage redundancy with the process model (artefact 2.1.3).

![Diagram](Figure 5.21: Sample of Service Journey Modelling Language (SJML) [142])

SJML [142] and CJM [89] share a similar abstraction with ServiceML’s SJM in representing a series of touch-points experienced by the customer. An example of SJML diagram is provided in figure 5.21.
Both SJML and CJM enhanced the touch-point visualization by providing notations symbolized the interaction mode, e.g. via telephone, via email, or via website.

CJM is actually a generic label given to types of diagram often used in service design. Without a single curator role to set the usage standard, the enhancement and extension of CJM produces various variants. One of CJM extensions introduces the motivational and emotional aspect of customers, extended from the pre-service encounter throughout to the post-service encounter [170][171].

Due to the clear origin of SJML, the modelling language experiences a more controlled enhancement. A more recent version of SJML adopts the multi-participant feature of service interactions, as demonstrated in figure 5.22 [172]. SJML is therefore an important alternative to be adopted for interaction modelling in the framework.

Figure 5.22: SJML abstraction for multi-participant interaction [172]
An important feature of these new streams of interaction modelling is in its specification of touch-point type, i.e. manual or software. It therefore might serve an important role in the framework: to differentiate between manual services and software services. Only services identified to be of the software type are required to be processed toward the software-service design (activity 2.2 in figure 5.6).

A summary of the additional format for the framework artefacts is presented in the figure 5.23. The proposed alternative metamodels are offered as a palette of options. The actual usefulness and usability of these alternatives still required further examination.

<table>
<thead>
<tr>
<th>1. IDENTIFICATION</th>
<th>2. DESIGN</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1 Understanding Service Context</td>
<td>2.1 Business Service Design</td>
</tr>
<tr>
<td>1.2 Business Model (4ds)</td>
<td>2.1.1 Service Architecture Model</td>
</tr>
<tr>
<td>1.3 Business Opportunity Priority</td>
<td>2.1.2 Service Goal &amp; Value (3b-ba)</td>
</tr>
<tr>
<td>1.4 Business Capabilities</td>
<td>2.1.3 Process Model</td>
</tr>
<tr>
<td>1.5 Business Service Design (3b-ba)</td>
<td>2.2 Composite Service</td>
</tr>
<tr>
<td>1.6 Business Process</td>
<td>2.2 Service Information Model</td>
</tr>
<tr>
<td>1.7 Business Model Canvas</td>
<td>2.2.3 Service Information Model</td>
</tr>
<tr>
<td>1.8 Opportunity List</td>
<td>2.2.4 Service Detail Model</td>
</tr>
</tbody>
</table>

Figure 5.23: Metamodels Options for Service Engineering Framework

This chapter describes an exploration toward a modelling concept as an analysis and design artefact. Despite the recent demand for agility in the business landscape, a model is still expected to hold an important role
in the analysis and design process [173]. The use of metamodels might be minimized in certain areas, but its usage pattern suggests the existence of a communication and collaboration space where models will persist, or even taking a central role, such as envisioned by the Model Driven Engineering approach [100].

In summary, this chapter provides an elaboration of verification process for the produced ‘service engineering’ ontologies and framework. This chapter also explores the metamodel landscape, resulting with additional metamodel alternatives to be considered into the produced framework. This new proposition opens new venues for future research works.
Chapter 6
Conclusion

6.1 Summary

The main goal of this research was to compose a consolidated framework for devising a service system from a business perspective toward a software perspective labelled as the General Service Engineering Framework (GSEF). In devising the framework, the research objectives were formalized into four research questions:

- **RQ1**: What concepts as components should be covered in SvE?
- **RQ2**: What are the activities performed during a SvE?
- **RQ3**: What artefacts should be produced in SvE?
- **RQ4**: In what format should the artefact be presented?

Figure 6.1 summarizes the produced artefacts as response to these questions within the context of the research structure. The first research question (RQ1) is examined with ontological approach, discussed in the third chapter. As a part of the framework, a service engineering ontological basis is proposed (figure 3.12 and table 3.22). The ontology specifies components relevant within the context of service engineering and defines relationship among the concepts.
The ontology covers both the business and informatics aspects of service engineering, i.e., (1) Business Capability, (2) Business Model, (3) Service Value, (4) Interaction Model, (5) Process Model and, (6) Software-Service Model. The software-service ontology (figure 3.14) were drawn from the informatics domain, while the generalized ontology of a service
system (figure 3.12) was built from both the business management the information systems domains.

The rest of the research questions (RQ2, RQ3, and RQ4) are explored throughout chapter four. Combining the response for these questions, a GSEF is produced composed from three components derived from research questions: (1) Stages and activity, (2) Artefacts, and (3) Format or metamodel. The produced framework was empirically evaluated with a series of case studies and theoretically validated with the previously produced ontology. In its final version, the framework specifies four main activities with sixteen types of artefacts to produce (figure 4.24).

Additional examination regarding the fourth research question (RQ4) is discussed in the metamodel exploration of the fifth chapter. A structured landscape of metamodel offerings from both industrial and academic circles was composed in the chapter (figure 5.5). The landscape is a product of exploring potential metamodels for representing an artefact. The exploration enriches the suggested artefact format from the original eighteen formats to thirty alternatives of metamodel (figure 5.23).

In summary, the works within this research produced three classes of contributing artefacts: (1) Service Engineering Framework, (2) Service Engineering Ontology, and (3) Metamodel Landscape. These artefacts are listed in table 6.1.
<table>
<thead>
<tr>
<th></th>
<th>Class</th>
<th>Artefact Name</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Framework</td>
<td>GSEF, first iteration (prototype)</td>
<td>Figure 4.3</td>
</tr>
<tr>
<td>2</td>
<td>Framework</td>
<td>GSEF, second iteration (evaluated)</td>
<td>Figure 4.19</td>
</tr>
<tr>
<td>3</td>
<td>Framework</td>
<td>GSEF, third iteration (validated)</td>
<td>Figure 4.24</td>
</tr>
<tr>
<td>4</td>
<td>Framework</td>
<td>GSEF, fourth iteration (proposed)</td>
<td>Figure 5.23</td>
</tr>
<tr>
<td>5</td>
<td>Ontology</td>
<td>Generals Service Ontology</td>
<td>Figure 3.12, table 3.22</td>
</tr>
<tr>
<td>6</td>
<td>Ontology</td>
<td>Software Service Ontology</td>
<td>Figure 3.14, table 3.24</td>
</tr>
<tr>
<td>7</td>
<td>Metamodel Landscape</td>
<td>MetamodelLandscape Structure</td>
<td>Table 5.1, figure 5.4</td>
</tr>
<tr>
<td>8</td>
<td>Metamodel Landscape</td>
<td>MetamodelLandscape Grouping</td>
<td>Figure 5.5</td>
</tr>
</tbody>
</table>

### 6.3 Limitations and Directions for Future Work

Several limitations are acknowledged from the research scope. First, the framework is not yet extended into the activity following design stage, i.e., implementation and evaluation stages. While some case studies were performed throughout the implementation stage, these activities are not captured and embedded into the framework, due to the original intended scope. This decision was made to reduce the complexity of the work by limiting the framework in a ‘platform-independent’ context.

A grounded conceptual work must be established before answering the implementation issues, which is inherently technology-specific. This framework scope limit is the first venue for available future work. A research challenge in this direction is to define a general abstraction capturing the variances exist in a ‘platform-specific’ context.
The second limitation is in the performance evaluation aspect of the framework. The research only evaluates the feasibility feature of the framework, via case studies, while the coverage sufficiency is validated with ontological comparison. A future work is available in terms of defining performance criteria to measure the efficiency and effectiveness of the framework and its components. Performance measurements might lead to further component modification to improve the overall performance of the framework, e.g. clarity, simplicity or traceability.

As a general framework, the produced GSEF serves as a template for further modifications and enhancements. The intentional limitation is the omission of a specific ‘technique’ in the framework. Future works is available in the form building a full-prescriptive service engineering method by specifying the ‘how-to’ technique for each activity. This venue can be undertaken based on a specific situational context or requirements.

Metamodel exploration is another potential venue to be pursued. Some metamodels can be considered to be a ‘better’ metamodel than others. The measuring criteria should be defined, which among others will be a trade-off between the expressive power, i.e. accuracy, and simplicity, i.e. comprehensibility [102]. For a stack of metamodels, the traceability and cross-translatability features will also be the criteria candidates.
Bibliography


178


[84] C. Ehrenhöfer and E. Kreuzer, “The role of business model design in the service engineering process: a comparative case study in the field of cloud computing to join service engineering with business model design,” in SRII Global Conference (SRII), 2012 Annual,


[119] M. B. Miles and A. M. Huberman, Qualitative data analysis: An expanded sourcebook. sage, 1994.


[149] OMG, “Case Management Model and Notation (CMMN),” *Object


[166] W. M. P. Van Der Aalst and W. M. P. Van Der Aalst, “Business


