APPROVAL SHEET

Title of Dissertation: Automatic Service Search & Composability Analysis in Large Scale Service Networks

Name of Candidate: Yunsu Lee

Doctor of Philosophy, 2015

Thesis and Abstract Approved: _____

Yun Peng
Professor
Department of Computer Science and
Electrical Engineering

Date Approved: _____

Curriculum Vitae

Name: Yunsu Lee.

Degree and date to be conferred: Ph.D., December 2015.

Secondary education: Baejung High School, Pusan, Korea, 1994.

Collegiate institutions attended:

Seoul National University, Bachelor in Electrical Engineering, 2000.

Major: Computer Science.

Professional Publications:

Lee, Y., & Peng, Y. (2013). A Framework for Developing Manufacturing Service Capability Information Model. In *Advances in Production Management Systems. Sustainable Production and Service Supply Chains* (pp. 325-333). Springer Berlin Heidelberg.

Lee, Y., Vujasinovic, M., & Ivezic, N. (2013). Use Case Analysis for Standard Manufacturing Service Capability Model. In *Advances in Production Management Systems. Sustainable Production and Service Supply Chains* (pp. 361-369). Springer Berlin Heidelberg.

Kulvatunyou, B., Wallace, E., Ivezic, N., & Lee, Y. (2014). Toward Manufacturing System Composability Analysis: A Use Case Scenario. In *Advances in Production Management Systems. Innovative and Knowledge-Based Production Management in a Global-Local World* (pp. 658-666). Springer Berlin Heidelberg.

Kulvatunyou, B., Lee, Y., Ivezic, N., & Peng, Y. (2015). A framework to canonicalize manufacturing service capability models. *Computers & Industrial Engineering*, *83*, 39-60.

Kulvatunyou, B., Ivezic, N., Lee, Y., & Shin, J. (2014). An analysis of OWL-based semantic mediation approaches to enhance manufacturing service capability

models. *International Journal of Computer Integrated Manufacturing*,*27*(9), 803-823.

Kulvatunyou, B., Ivezic, N., & Lee, Y. (2014). On enhancing communication of the manufacturing service capability information using reference ontology. *International Journal of Computer Integrated Manufacturing*, *27*(12), 1105-1135.

Shin, J., Kulvatunyou, B., Lee, Y., & Ivezic, N. (2013). Concept Analysis to Enrich Manufacturing Service Capability Models. *Procedia Computer Science*,*16*, 648-657.

Professional positions held:

Guest Researcher at National Institute of Standards and Technology, Gaithersburg, MD, USA, April 2011 – Present.

Summer Intern at Clark & Parsia, Washington D.C., USA, June 2013 – August 2013.

Co-founder and Chief Engineer at Torpedo Inc., Seoul, Korea, June 2004 – March 2011.

Software Engineer at Innodigital Inc., Seoul, Korea, April 2001 – May 2004.

ABSTRACT

Title of Dissertation: AUTOMATIC SERVICE SEARCH & COMPOSABILITY ANALYSIS IN LARGE SCALE SERVICE NETWORKS.

Yunsu Lee, Ph.D. Computer Science, 2015

Directed By: Yun Peng, Professor,
Department of Computer Science and
Electrical Engineering

Currently, software and hardware system components are trending toward modularized and virtualized as atomic services on the cloud. A number of cloud platforms or marketplaces are available where everybody can provide their system components as services. In this situation, service composition is essential, because the functionalities offered by a single atomic service might not satisfy users' complex requirements. Since there are already a large number of available services and significant increase in the number of new services over time, manual service composition is impractical.

In our research, we propose computer-aided methods to help find and compose appropriate services to fulfill users' requirement in large scale service networks. For this purpose, we explore the following methods. First, we develop a method for formally representing a service in term of composability by considering various functional and non-functional characteristics of services. Second, we develop a method for aiding the

development of the reference ontologies that are crucial for representing a service. We explore a bottom-up-based statistical method for the ontology development. Third, we architect a framework that encompasses the reference models, effective strategy, and necessary procedures for the services search and composition. Finally, we develop a graph-based algorithm that is highly specialized for services search and composition. Experimental comparative performance analysis against existing automatic services composition methods is also provided.

Keywords:

Service composition, AI planning, AND/OR graph search, heuristic search, ontology development

AUTOMATIC SERVICE SEARCH & COMPOSABILITY ANALYSIS IN LARGE SCALE SERVICE NETWORKS

By

Yunsu Lee

Dissertation submitted to the Faculty of the Graduate School of the
University of Maryland, Baltimore County, in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
2015

Acknowledgements

Over the past five years I have received great support and encouragement from a number of individuals.

First and foremost, I would like to express my gratitude to my advisor Dr. Yun Peng for his endless advice, support, and encouragement throughout my Ph.D. study. He is very enthusiastic researcher who always looks for novel approaches and motivates me toward innovations.

I would also like to express special thank to Dr. Nenad Ivezic and Dr. Boonserm (Serm) Kulvatunyou. Thanks to them, I could get a great opportunity to work at NIST. They never put me down or told me I could not do it, always encouraging me, and supporting me. Especially, I really enjoyed the regular meetings with them on every Friday afternoon. That was really exciting and constructive discussion. I have a feeling that I'll never again meet such a wonderful person like them.

I would also like to acknowledge, with profound thanks, the support and encouragement I have been receiving from Simon Frechette who is a program manager at NIST.

I especially would like to thank Dr. Hyunbo Cho who led me to a whole new world of science and technology. He made me realize that the world is a big place and there are still lots of things waiting to be explored.

During my Ph.D. study, it was very fortunate to have chances to learn a lot from the brilliant professors at UMBC. I have learned the fundamentals of the semantic web and

artificial intelligence from Dr. Tim Finin which became solid foundations of my research. Dr. Milton Halem's service oriented computing class motivated me to choose my research topic. Dr. Yelena Yesha's classes improved my knowledge on database and operating system that was a great help when I implement the framework for my research.

I would like to say thank you very much to my dear friends, Dr. Kilwon Moon, Dr. Jongyoon Ha, Dr. Jaehun Lee, Dr. Seungjun Shin, Dr. Jaekark Choi, Dr. Jungyub Woo, Dr. Eunju Kim, Dr. Yena Kim, Dr. Duckbong Kim, Dr. Jeonghoon Ha, Dr. Youngjong Lee, and Dr. Jaewook Kim. The wonderful time with them was always what keeps me going.

Sincerely, I would like to thank my parents, parents-in-law, brother, brother-in-law, sister, and sisters-in-law from the bottom of my heart, for their support, understanding, and endless patience.

Last but not least, I would like to say I love you to my wife, Minjung Cho and my lovely three sons, Chano, Chanseo, and Bokdung. Bokdung is a birth name of my third son to be born soon. In such a hardscrabble life as a foreign student, my family gave me the strength and direction necessary to stay balanced and happy.

Finally, I would like to acknowledge the support provided by NIST Grants #70NANB9H9145, #70NANB12H214, #70NANB13H154, and #70NANB14H269.

Table of Contents

List of Tables

List of Figures

Chapter 1. Introduction

1.1 *Motivation*

Currently, software and hardware system components are trending toward modular atomic services (rather than large monolithic applications) on the cloud. This trend enables companies to quickly respond to disruptions and the customer's changing needs by composing services to meet the new requirement. This is supported by the openings of cloud platforms or marketplaces where virtually anybody can provide software and hardware system components as a service [IFTTT 2015, Zapier 2015, Programmable Web 2015, and IBM BlueMix 2015]. The open access of such marketplaces means that the number of available services, sometimes referred to as 'Apps', is poise to be exploding.

In such situation, computer-aided services composition is essential. For instance, Zapier is a user-friendly services composition platform. It provides a simple rule-based structure to connect different services. For example, a user can write a rule to detect an event from a Customer Relationship Management (CRM) service which then triggers a email service to send the event to a particular address as a consequence. Despite such simple services composition structure, it is more important to figure out what services to use (the functionality of the service) and what services are compatible to each other. For that

Zapier provides only a simple categorization of the services. Therefore, it is a challenge for the user to manually figure out a service or a set of services that meet both the functional and non-functional requirements, and then check whether the platform supports composition of the identified services. In an enterprise business environment, the composition task is typically more complex than the simple email trigger example above. Therefore, a computational aid is necessary for making services search and composition more efficient and effective in such an open cloud service environment.

1.2 *Problem Statement*

There are two primary problems in developing a computational aid for services search and composition.

1) Currently available computer interpretable service description is insufficient.
2) There is no automatic method that is best suitable for the services composition problem.

The next two subsections provide more detailed backgrounds of these two problems.

1.2.1 Insufficient service description

Typically, services have functional and non-functional characteristics. Currently, several standards and efforts exist to describe the characteristics of services. WSDL [W3C 2001] is one of the standards used by many software vendors. However, the standard focuses

more on the non-functional characteristics such as the transfer protocol specification that are essential for measuring compatibility between different services. The standard offers no more than inputs and outputs of the service and their formats in term of the functional characteristics.

Prior works such as OWL-S [W3C 2004b] and WSMO [W3C 2005] have been proposed to enhance the functional semantics of web services descriptions. Although these works could be a strong basis for service description development, there are some limitations. For example, the service profile in the OWL-S is used to describe the service's functionality (what the service does), but the information is primary for human reading rather than computer interpretation. OWL-S provides a process model that consists of sets of inputs, outputs, pre-conditions and results of the service execution. These sets may be used to represent the service's functionality in a computer interpretable manner, but those are insufficient, because same or similar sets of the characteristics may represent different functionalities depending on the context of service usage or service provider's intention. Another limitation is a lack of provision for describing domain-specific characteristics of services that can be critical when composing services. In order to address the issues, the existing specifications for service descriptions should be extended and it is necessary to investigate existing function modeling/representation theories to precisely represent functions of services.

1.2.2 No suitable automatic approach for services composition

Although some existing works [Lin et al. 2012, Yan et al. 2012, and Hatzi et al. 2013] have formulated the services composition problem as a classical planning problem, the assumptions made do not fit well with the open marketplace of services environment. In those works, services were considered as operators and user's requirements were represented by the initial and the goal conditions. Then, the problem is to find a set of services that can transition the initial condition to the goal condition by applying input/output and pre/post-conditions of the services. However, there are some issues when applying the classical planning methods to the services composition problem.

Firstly, there are differences in characteristics of the problem elements. A classical planning problem typically has a small number of actions (e.g., moving blocks), a large number of objects (e.g., hundreds of blocks). On the other hand, services composition problem typically deals with a large number of actions (huge number of services on the cloud), a limited number of objects (e.g., getting a flight, renting a car, and registering a hotel). In addition, the classical planning problem does not deal with the situation where there are a large number of same or similar services (e.g., hundreds of travel agent services).

The other issue is that classical planning problem and consequently its approaches were designed out-of-the-box to deal with the interleaving between sub-plans to avoid the Sussman anomaly [Nils 2001]. For services composition problem, however, interleaving

condition can be checked and prevented a priori. Thus, applying classical planning problem solving approaches to the services composition problem is usually computationally too expensive [Oh et al. 2008]. Therefore, a highly specialized method for services composition problem that scales better with the number of action is required, rather than compiling the services composition problem into a more complex planning problems such as Satisfiability, Constraint Satisfaction, and Integer Linear Programming.

1.3 *Research Objectives and Methods*

The main objective of this research is to develop a framework for computer-aided services search and composition in an open cloud services marketplace environment. Such framework will scale well with the large number of services available and take into account not only functional requirements but also non-functional requirements that include both technical (e.g., security, reliability) and non-technical (e.g., cost, vendor preferences) characteristics. To realize such framework, four sub-objectives are needed as followings:

1) Architect the framework that provides a high-level design of components, strategy, and procedure for the services search and composition. The components within the architecture shall assist the user in discovering and composing services in a large-scaled cloud services repository (i.e., open cloud marketplace) and shall have the flexibility to deal with various aspects of functional and non-functional user requirements.

2) Develop a method for representing a service's functionality.

- Define composability: investigate and adapt definitions from existing works.
- Formalize functional and non-functional characteristics: investigate and adapt the notions of functional and non-functional requirements from the requirement engineering discipline.
- Develop a functional representation model: investigate and adapt the models from the function modeling and functional representation theories works; and develop use cases to validate the developed functional representation model.

3) Develop a method for aiding the functional representation ontology development.

- Design and implement an efficient method for identifying appropriate functional and non-functional characteristics of software and hardware services.
- Assess the feasibility of the method by utilizing a large quantity of complex functionality information as a data source.

4) Develop a specialized algorithm for services search and composition.

- Design and implement a specialized algorithm for services search and composition.
- Provide a formal proof for the correctness of the algorithm
- Provide a time complexity analysis.

- Provide an experimental comparative performance analysis against existing automatic services composition methods.

1.4 *Organization of the Dissertation*

In Chapter 2, we provide a formal definition of the problem that we are going to solve. In Chapter 3, we provide a review of the literatures that are closely related to our research. In the first part of the Chapter 3, we present details of the open cloud architecture which is a background of our research. In the next parts, we provide extensive review of existing service description, ontology development, and automatic service composition methods and discuss benefits and limitations of the existing works. In the subsequent chapters, we provide our approaches for the topics that are reviewed. In Chapter 4, for the service description, we analyze functional and non-functional characteristics of services by investigating existing function modeling and representation theories. In addition, we present our own function and service representation method. In Chapter 5, we provide a computational method for the ontology development. We provide the result of the experiments to validate the method. Chapter 6 presents a novel framework that encompasses necessary components and strategies for the service search and composition. In Chapter 7, we provide a graph-based service search and composition algorithm. We experimentally show the benefits of our algorithm against existing AI-Planning algorithms. Chapter 8 concludes the research as well as future research directions.

Chapter 2. Problem Definition

In this Chapter, we provide a formal definition of the service search and composition problem. Other necessary definitions such as service, service request, and composition cost are also provided.

Definition 2.1 (Service). A service *S* has six sets of parameters: $\boldsymbol{S} = (F, I, O, Pre, Post, Prop)$, where

$\boldsymbol{F} = \{F_1, F_2, F_3, \dots\}$ is a set of functions that are provided by the service.

$\boldsymbol{I} = \{I_1, I_2, I_3, \dots\}$ is a set of inputs that are consumed by the service *S*.

$\boldsymbol{O} = \{O_1, O_2, O_3, \dots\}$ is a set of outputs that are produced by the function *S*.

$\boldsymbol{Pre} = \{Pre_1, Pre_2, Pre_3, \dots\}$ is a set of pre-conditions that should always be satisfied prior to the execution of the service *S*. If any of the pre-conditions defined in the service *S* is violated, the result of the execution of the service *S* may or may not carry out its intended work.

$\boldsymbol{Post} = \{Post_1, Post_2, Post_3, \dots\}$ is a set of post-conditions that are effects after the execution of the service *S*.

Prop = {$Prop_1$, $Prop_2$, $Prop_3$, … } = a set of properties that represents non-functional characteristics.

Definition 2.2 (Service Request). A user who wants to invoke a service has a service request ***R*** = {R_i, R_g}, where

$\boldsymbol{R_i}$ = {I_{c1}, I_{c2}, I_{c3}, … } is a set of initial condition parameters.

$\boldsymbol{R_g}$ = {G_{c1}, G_{c2}, G_{c3}, … } is a set of goal condition parameters.

Definition 2.3 (Service Search Problem). Given a service request *R*, the service search problem is to find a service *S* that satisfies:

$$R_i \supseteq (I \cup Pre) \cap R_g \subseteq (O \cup Post)$$

If there is no single service satisfying the service request *R*, the user may want to compose multiple services to satisfy the service request *R*. This problem then becomes a *Services Composition Problem*.

Definition 2.4 (Services Composition Problem). Given a service request *R*, the services composition problem is to find a set of services {S_1, S_2, S_3, … , S_k} that satisfies:

$R_i \cup \sum_{j=1}^{k}(S_j.O \cup S_j.Post) \supseteq R_g \cup \sum_{j=1}^{k}(S_j.I \cup S_j.Pre)$, where

$S_j.I$, $S_j.Pre$, $S_j.O$, and $S_j.Post$ represent input, pre-condition, output, and post-condition of service S_j respectively.

The set of services satisfying the above condition is said to be *functionally satisfying R*.

In practice, two services can be composed even though the format of the output from one service does not exactly match with the format of the input to the other service. For example, the services can be composed by using an adapter that converts the output format of one service to the input format of another.

Services have a variety of characteristics other than their inputs and outputs. For instance, an information service may have various characteristics such as data formats, protocols, encryption/digital-signature algorithm, message compression algorithm, and industry standards, etc. Each characteristic is further characterized such as SHA256 or SHA128 as the supported digital-signature algorithm, and JSON or XML as the supported data format. Thus, other important characteristics should be considered when composing different services. The cost required to align different characteristics is referred to as *Composition Cost*. Finding a set of services that has the minimum composition cost is referred to in this work as *Minimum Composition Cost Problem*.

Definition 2.5 (Composition Cost). Services *S1* and *S2* have property sets *Prop1* and *Prop2* respectively that are relevant to the services composition. There are two types of penalties including a default and user-defined penalty. User can choose any properties and assign any penalty values to the properties. The default penalty will be assigned to the properties that do not have any user-defined penalty.

Given the user-defined penalties $\{u_1, \ldots u_k\}$ on the set of properties $\{Prop_1, \ldots Prop_k\}$, the composition cost C is defined by:

$C = Prop_{default} * d_p + \sum_{i=1}^{k} u_i$, where $Prop_{default}$ is a set of properties that do not have an user-defined penalty and d_p is a default penalty value, and u_i will be 0 when the properties between the two services do match.

Definition 2.6 (Minimum Composition Cost Problem). Given the user's service request R, the minimum composition cost problem is to find a set of services $S = \{S_1, S_2, S_3, \ldots, S_k\}$ functionally satisfying R with the total composition cost minimized. Assume that there are m edges $(e_1, e_2, \ldots, e_m)$ between services in S. Then, the total composition cost, $C = \sum_{n=1}^{m} c(e_n)$, where $c(e_n)$ is the composition cost on the edge e_n.

Chapter 3. Related Works

In this chapter, the works that are closely relevant to our research are presented as follows. First, we present a brief review of the emerging open cloud architecture including the infrastructure as service (IaaS), platform as service (PaaS), and software as service (SaaS) [Ivezic et al. 2014]. Second, we review existing specifications for service description. And then, we discuss about their limitations. Third, we present existing model development methods, specifically the ontology development methods, for the service modeling and discuss the limitations of the existing methods. Forth, a state of the art review on existing services composition methods is provided. We investigated two major streams of the automatic service search and composition methods including AI Planning-based service composition approaches and graph search-based composition approaches. We discuss the benefits and shortcomings of the existing works. Finally, we present researches on the function representation theories, and discuss how these works can help enhance the service description as well as the automation of services composition.

3.1 *Open Cloud Architecture*

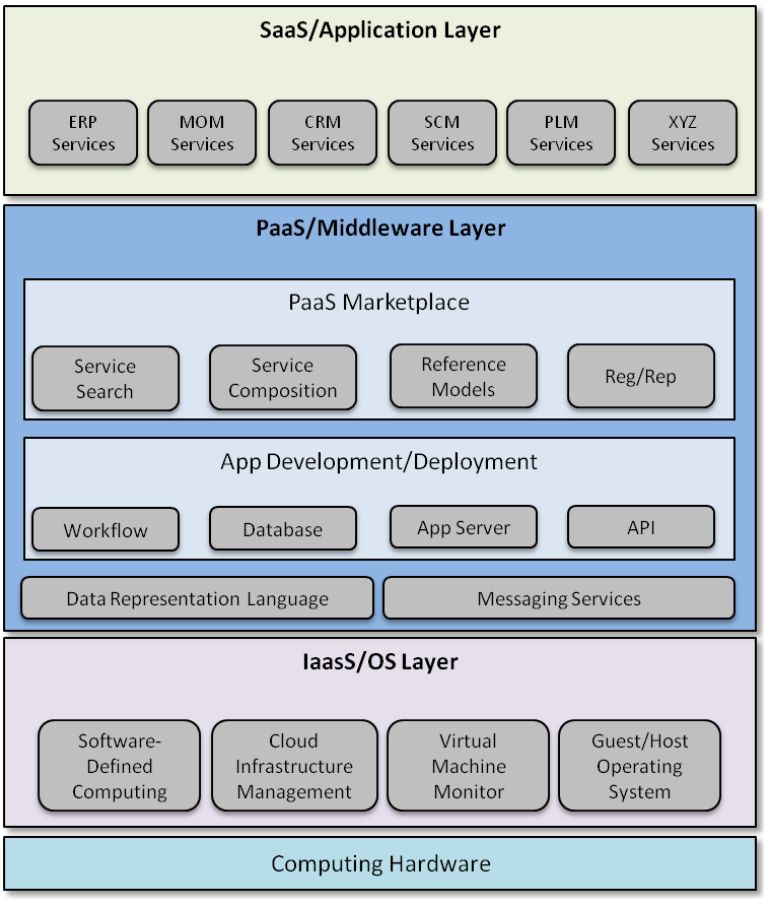

Figure 3-1 Open Cloud Architecture

Figure 3-1 above illustrates a high-level view of the open cloud architecture specifically for manufacturing enterprises. A number of open standards and open sources that support the open cloud architecture are also presented in [Diaz 2013]. Typically, the cloud computing architecture is categorized into three different layers: Infrastructure-as-a-Service (IaaS); Platform-as-a-Service (PaaS); and Software-as-a-Service (SaaS) [Liu et al.

2011]. The IaaS layer provides computing resources by a virtualization of computing hardware in a scalable manner. Physical or virtual machines, firewalls, and data storage are some of the examples of the computing hardware. The PaaS layer is to provide computing platforms such as operating system, database, and web application server etc. The PaaS layer connects the IaaS layer with the SaaS layer. In the SaaS layer, service providers can develop and deploy software and hardware services on the basis of the PaaS layer. The services in SaaS layer provide domain specific functionalities to the cloud user. In the next sub-sections, we discuss details of each layer and provide the target layer of our research.

3.1.1 Infrastructure as a Service (IaaS)

Computing resources (e.g., processor, data storage, network devices, and other fundamental computing resources) are provided in the IaaS layer through virtualization. The IaaS layer enables a scalable/dynamic provision of the computing resources.

OpenStack [OpenStack. 2015] is one of the open source community that provides an open architecture for the IaaS layer. The OpenStack has a number of infrastructure related projects specifically for the *Software Defined Computing and Cloud Infrastructure Management*. The *Software Defined Computing* component allows access to the computing resources through the Application Programming Interfaces (APIs). For the management of the resources, specific APIs are provided by the *Cloud Infrastructure Management* component. ISO 19831 – Cloud Infrastructure Management Interface (CIMI)

[ISO/IEC 19831] recently completed the standardization of the IaaS API. The OpenStack framework also supports several open source virtual machines such as VirtualBox [VirtualBox. 2015] and KVM (Kernel-based Virtual Machine) [KVM 2015].

3.1.2 Platform as a Service (PaaS)

The PaaS layer connects the computing resources in the IaaS and the services in the SaaS layer, as shown in Figure 3-1.

The *Data Representation Languages* enables information sharing, and the *Messaging Services* components provide communication capabilities. For the *Data Representation Languages,* in order to address some of the limitations in the Extensible Markup Language (XML) [W3C 2006], the Resource Description Framework (RDF) [W3C 2004a] and JavaScript Object Notation (JSON) [ECMA International 2013] have emerged. For the *Messaging Services,* new standards, such as Open Data (OData) protocol [OASIS 2014a], Message Queue Telemetry Transport (MQTT) [OASIS 2014b], and Advanced Message Queuing Protocol (AMQP) [ISO/IEC 2014] are developed primarily for the following: for the compatibility work with existing standards, such as the Web Services [W3C 2002] and OPC Unified Architecture [OPC Foundation 2006]; and for the connectivity from the lower device level to the higher business level.

The PaaS layer also provides a platform for the development and deployment of services, called *Application Development/Deployment*. Workflow, database, and application server

are examples of the platform. OASIS Cloud Application Management for Platforms (CAMP) [OASIS 2014c] and Topology and Orchestration Specification for Cloud Applications (TOSCA) [OASIS 2013] are two promising standards for the portability of cloud applications across the platforms. CAMP provides 1) a service-oriented API specification for the management of the application deployment life-cycle, and 2) a specification for the description of the PaaS-layer components that are required by an application. TOSCA has a larger scope than CAMP. TOSCA describes all the resources required by an application and the processes needed to deploy. In addition, TOSCA describes the provision of the whole application as well as its life-cycle management.

The PaaS Marketplace allows PaaS users to access platform services to meet their particular requirements. The Service Search and Composition component assists in managing the complexity of services compositions by providing multi-criteria decision support for the user to match requirements with capabilities. The Reference Model Management component assists the community with the evolution and other life-cycle-management aspects of the reference model. The same set of marketplace-management functionalities in the PaaS layer will also support the SaaS Marketplace. The diversity of manufacturing domains, where a functional characteristic may be specific to an industry or even product type, makes these functionalities even more important in the SaaS layer.

The marketplace is the area where more research and new standards are needed to fully realize the open architecture. This belief is supported by several, recent publications discussed in the next section.

3.1.3 Software as a Service (SaaS)

The cloud-based SaaS layer continues to evolve. The ability, which quickly builds up a new business value, accelerates the on-going evolution. The ability can be achieved by dynamically composing software services that are currently being operated. “An API economy.” represents well this ability [Diaz 2013]. For the API economy, the SaaS Marketplace enables the information and application functions accessible as discrete services.

Standards for data-level interoperability are important to enable the service composition. In addition, new standards that take into account the functional and non-functional level interoperability are also needed. These new standards for functional and non-functional level interoperability would enable to specify functional and non-functional requirements as well as allow for effective search and composition of services. Although the standards in this area are underdeveloped [VDE Association for electrical, electronic & information technologies. 2014], there exists some research results that might be a basis to build upon. For example, the NIST SIMA Reference Architecture Activity Models [Barkmeyer 1999] provides an integrated and hierarchical view of manufacturing enterprise functions. OAGIS provides a basis for the Enterprise Resource Planning (ERP) and Supply Chain

Management (SCM) Functions [Murray 2011 and Gerald et al. 2001]. ISA-95 and ISA-88 [ANSI/ISA 2010] provide a foundation for the Manufacturing Operation Management (MOM) Functions [Younus et al. 2010]. And, the PLM (Product Lifecycle Management) Services standard provides a basis for PLM and Digital Manufacturing (DM) Functions [CIMdata 2010 and CIMdata 2011]. The telecommunications industry has developed a functional model that provides a four level decomposition of functions in telecommunications enterprises [CISCO 2009]. Such a functional model allows user to effectively specify customer requirements as well as identify providers' service capabilities for rapidly designing and configuring systems.

The Smart Manufacturing Working Group at the OAGi [OAGi 2014] tries to address the requirement for the functional specification standards specifically for the manufacturing domain based on some of existing standards such as the Process classification framework [American Productivity and Quality Center 2014], the Supply Chain Operation Reference (SCOR) model [Supply Chain Council 2012], and ISA-95.

The importance of standards for functional and non-functional requirements' specifications is also found in some of the research initiatives [Acatech 2013 and European Commission 2013]. They investigate new methods for developing, adopting, and managing reference models for the functional and non-functional requirements' specifications and the bottom-up-based model development method appeared recently in both industry and academia [Acatech 2013 and European Commission 2013].

3.2 *Service Description Method*

3.2.1 OWL-S and WSMO

The rise of Internet computing resulted in Web-based interface definition languages. W3C WSDL (Web Services Description Language) [W3C 2001] is a predominant one. Basic semantics and structure of WSDL are similar to that of programming language APIs. A WSDL function, however, does not necessarily tie to a source code function – allowing it to represent the functionality. WSDL also allows for richer description via XML Schema specification of the input and output and via structured annotation on any information element. However, WSDL does not standardize any semantics of the annotation. The strength of WSDL is in the standardized semantics of its transfer protocol specification that is essential for measuring compatibility.

Several efforts have been proposed to enhance the semantics of Web services descriptions. Prior works in this area include OWL-S [W3C 2004], SAWSDL [W3C 2007], and WSMO [W3C 2005]. SAWSDL (Semantic Annotations for WSDL and XML Schema) enhances WSDL and associated XML Schema semantics by adding attributes to WSDL entities that point to concepts in a semantically rich ontology. SAWSDL does not define any additional semantics to describe functionalities; it only provides a mechanism to link to additional semantics.

OWL-S and WSMO (Web Services Modeling Ontology) are similar in their efforts to define upper ontologies for service descriptions (called Service Profile in OWL-S and Capability in WSMO). Both of them rely on a similar set of elements that describe 1) pre and post conditions associated with information used and produced by a service and 2) pre and post conditions associated with the states of world before and after the execution of the service. Further study about what kinds of world states are essential to describe functionality is necessary. Both efforts allow for a detailed description of functionality by specifying a process via choreography or orchestration. Such provision needs to be evaluated for complexity at the time of composability analysis. Semantic links between the service and ontological concepts of functionalities as in SAWSDL may be sufficient.

WSMO defines a Goal concept in addition to the Capability concept. SAWSDL also informally describes this notion as a Web service request. In WSMO, a Goal is described by the post condition. The functional description in the Goal expresses requirements and is used for matching/searching a Capability. If the post condition in the Goal matches the one in the Capability, then the service is relevant. Such notion of Goal and Capability matching is part of the composability analysis.

While OWL-S and WSMO define upper ontologies for service descriptions, they lack provision for describing functionalities or domain-specific characteristics of the capability. For example, a particular order processing service may be able to process several order types including the new-item outbound order, return order, and consumer

replacement order while another order processing service may only be able to process a new-item outbound order. While some practitioners may view this as constraints on input to the service, it is not always possible to express such characteristics as a condition on the input (i.e., the input schema may not contain an order type element). These sorts of differences in services must be known to identify the component services that are composed to perform a desired higher-level, business functionality. One objective of our research work is to develop a shared ontology for manufacturing software (and hardware) services that describe such kind of characteristics of functionality.

3.3 *Approaches for Ontology Development*

3.3.1 Summary of existing works

Since an ontology can be created by unifying/merging existing models, methodologies to create unified database views are relevant to the ontology creation. Hayne and Ram (1990) and Navathe et al. (1986) have provided the methodologies to create unified database view. Mapping is one of the most difficult tasks in unifying/merging existing ontologies. Automated or semi-automated ontology merging and mapping technologies are provided by PROMPT (Noy and Musen 2003) and Chimaera (McGuiness et al. 2000). We have found that PROMPT does not perform well when encountering with structural conflicts. Shvaiko and Euzenat (2011) have summarized algorithms to suggest mappings and

indicated that one of the open issues is to identify correspondences between classes and properties, i.e., when dealing with structural conflicts.

Jones et al. (1998) have summarized the main ontology engineering activities and identified the need for guidance on ontology reuse. Staab et al. (2001) have presented guidance for building ontologies either from scratch, reusing other ontologies as they are, or re-engineering them. Pinto et al. (2004) has suggested a distributed ontology engineering process. These ontology engineering approaches can be applied to any ontology development activity. There are some approaches to use the *Ontology Design Pattern* (ODP) for the ontology development. The NeOn project has delivered an initial and significant report on ontology development using ODPs. Ontology evolution management is also important task in the ontology development. Noy and Klein (2004) have characterized the causes of evolution of ontologies, including changes in the domain, changes in conceptualization, and changes in the explicit specification. Flouris et al. (2008) have summarized related works for the ontology change management.

3.3.2 Challenges in Ontology Development

One of the most challenging activities when unifying/merging existing ontologies is to resolve semantic/schematic conflicts between different ontologies. Sheth and Kashyab (1992) have identified various types of schematic differences between semantically similar objects from the relational databases perspective (i.e., objects are tables). Park and Ram (2004) have characterized these differences into two broad categories, namely the

data-level and schema-level conflicts. Most of these conflict types can be extended to OWL-based models. Data-level conflicts are differences in data domains caused by the multiple representations and interpretations of similar data. Data-level conflicts are applicable to representation of values of OWL data properties. Types of data-level conflicts relevant to our work include data-representation conflicts, data-unit conflicts, and data-precision conflicts. Data-representation conflicts occur when the semantically same values are represented differently such as "05/08/2012" and "May-08-2012". The data-unit conflicts occur when the same quantities are represented with differing units, e.g., "2 inches" and "5 centimeters". Data-precision conflicts occur when different scaling is used, e.g., when continuous values between 0 and 100 are used to indicate qualities vs. when discrete scales like low, medium, high is used. The schema-level conflicts are subcategorized into naming conflicts, entity-identifier conflicts, schema-isomorphism conflicts, generalization conflicts, aggregation conflicts, and schematic discrepancies. Naming conflicts are the cases where two semantically identical concepts are named differently (synonyms); or, when two semantically different concepts are named the same (homonyms). Naming conflicts are applicable to OWL classes and properties as they have names. Entity-identifier conflicts can occur when differing primary keys are used for the same entity in different databases. This can occur in OWL when multiple class instances (individuals) with different URIs refer to the same individual. Isomorphism conflicts are the cases where two semantically identical concepts are modeled with differing set of attributes and also different number of

attributes, e.g., *Supplier(ID, GeneralPhone, SupportPhone)* and *Supplier(ID, Phone), Address(Line1, Line2, Zip)* and *Address(Street, City, State, Zip)*. Isomorphism conflicts are applicable to OWL classes in the sense that they can have differing set of properties. Generalization conflicts are the cases where objects/classes subsume one another, e.g., *Student(ID, Name)* subsumes *GraduateStudent(ID, Name)*. Generalization conflicts are applicable to OWL classes and properties particularly when two models have different subsumption hierarchies. Aggregation conflicts are the cases when a property of a class is an aggregation of properties from multiple instances of another class. For example, the *MonthlyProduction(ID, Month, Year, Item, Quantity)* is an aggregation of the *DailyProduction(ID, Date, Item, Quantity)*. The schematic discrepancies are the cases where information is modeled using differing constructs – table name, attribute name, and attribute value. In OWL, the information about a supplier providing a *CNC Machining Service* may be modeled using a class declaration axiom (a supplier is a type of *CNCMachiningService* class), an object property assertion (e.g., the supplier has an object property pointing to an instance of *CNCMachiningService* class or the supplier has an object property pointing to a *CNCMachiningService* instance of a *ManufacturingService* class), or a data property assertion (e.g., the supplier has a string-based property *providesService* pointing to *CNCMachiningService*, the supplier has a boolean property *isCNCMachiningServiceProvider* with the value true).

Preserving consistency is one of the most challenging activities in all ontology development methodologies. Vujasinovic et al. (2013) have shown that the inconsistent ontology design may cause additional efforts in the ontology activities such as ontology refactoring, or querying. An ontology developer typically encounters alternative solutions such that a certain concept can be modeled by introducing a separate class, or by introducing a property. For instance, to distinguish between *EDM Machining* and *CNC Machining*, an ontology developer may encode *EDM* and *CNC* as two separate subclasses of *Machining*. Alternatively the ontology developer could encode *EDM* and *CNC* as two different values of a property called *hasMachiningType*. In most cases, modeling conventions chosen in ontology development largely depend on the ontology developer's taste.

The management of changes after developing an ontology is also a challenging task. Noy and Klein (2004) have characterized the causes of evolution of ontologies, including changes in the domain, changes in conceptualization, and changes in the explicit specification.

3.3.3 Discussion

In order to address the issues in the ontology development, the ontology should be developed and evolved in a way that the outcome of the process is repeatable (i.e., the resulting reference model must identical, when started from the same initial conditions). In other words, while human intervention is essential in making choices at various levels

when developing or evolving an ontology, human inputs need to be taken into account within a controlled setting. Typically, an ontology development or evolution methodologies provide guidelines to assist the user in making choices from the high-level structure of the ontology, to the detail of specific concepts. However, the guidelines might not prevent various possible conflicts due to the differences of the perceptions, experiences, and understanding specific to each user. This will require a mixture of statistical and other computational methods that minimize the users' subjective judgment.

3.4 *Representing Function*

Enhancing service description with better semantics of functionality is necessary for more precise composability analysis. For that, existing works in function representation theories and function modeling research are investigated. Studies about function modeling and representation have been prevalent in the product design discipline.

First, definitions of the function or functionality[1] are investigated. Varying definitions are found as followed.

[1] In software engineer, 'function' is typically referred to the implementation, the code. Hence, the term 'functionality' should be used to refer to what the function (or an object) does. Since materials discussed below are borrowed from the product design discipline the term 'function' is used interchangeable with

- Faltings (1990): Function of a mechanical object is dependent on the way that motion and forces are transmitted through the contacts between parts.
- Chittaro and Kumar (1998): Function is a source of knowledge that abstracts behavior. Function of a component can be defined as operational (i.e., a relation between the input and output in the component) or purposive (i.e., a relation between the goal of a human user and the behavior of the component).
- Chakrabarti (1998): Function of an object is distinct from its behavior in that it is intentional rather than actual or expected, and proposes that there are two related but distinct views of function. In one view, it is at the same level of abstraction as behavior (intended behavior), while in the other it is at a higher level (purpose).
- Chakrabarti and Bligh (2001): Function is a description of the action or effect required by a design problem, or that supplied by a solution.
- Chandrasekaran and Josephson (2000): Device-centric function is the internal actions that a device should perform and environment-centric function is the effects that the device has on its environment.
- Deng (2002): Function can be semantically classified into two types: purpose function and action function. Purpose function is a description of the designer's

---

'functionality'. Notice that in the product design discipline the implementation is referred to as 'object' or 'device'.

> intention or the purpose of a design. It is thus abstract and subjective. It is teleological knowledge and is human oriented. Action function is an abstraction of intended and useful behavior that an artifact exhibits.

As Chandrasekaran (2005) stated, the various terms in the definitions are not clear as in the function definitions. For instance, it is not clear whether what one author means by behavior or action is exactly the same as that meant by another author. Crilly (2013) raised a critical issue that statements about functions can be interpreted in different ways such that function definitions are relative or subjective and they are just labels that people assign to things to reflect how they think about them.

However, we observed that the device and environment centric distinction (Chandrasekaran and Josephson, 2000) covers all the meanings in the different definitions, because the distinction clearly relates two common meanings of function. First off, it is clear that the device-centric function corresponds to the behavior of a given system (the term, system, here can be any of device, service, component or so on) stated "in terms of variables associated with internal structural elements." Thus, the device-centric function can cover the 'operational function' in Chittaro and Kumar (1998), the 'intended behavior' in Chakrabarti (1998), the 'action' in Chakrabarti and Bligh (2001), and the 'action function' in Deng (2002).

On the other hand, environment-centric function corresponds to the system's effect on the environment, stated "entirely in terms of elements external to the device." Thus, the

environment-centric function covers the ‘purposive function’ in Chittaro and Kumar (1998), the ‘purpose’ in Chakrabarti (1998), the ‘effect’ in Chakrabarti and Bligh (2001), and the ‘purpose function’ in Deng (2002). This is because the effect of the environment-centric function has a close relation with the purpose or role of the system. Chandrasekaran and Josephson (2000) introduced the concept, ‘Mode of Deployment,’ that enables the assignment of a specific context to function. For example, the effect of electric lamp is ‘room illumination’ when the lamp is placed (i.e., deployed) in a room with the switch turned on. Such condition for the effect is called ‘Mode of Deployment’. Thus, we can generalize the effect of the electric lamp as ‘illuminate something or somewhere’. Then, the effect can be exactly matched with the purpose of the electric lamp, because both the designer and user of the electric lamp have the purpose, ‘illuminate something or somewhere’. Chandrasekaran (2005) explains that the device-centric function is the mean to achieving the environment-centric function and this statement also implies that the environment-centric function has a close relation with the ‘purpose’.

It is observed that function definition is quite subjective. Similarly, statements about functions of service can be interpreted in different ways. Function and service are both subjective that is just labels that people assign to things to reflect how they think about them. Thus, semantics of function associated with a service description standard used today is ambiguous. It is typically represented with a single label. More logical

representation is needed. Subjective nature of service boundaries is also causing a problem when interpreting its functions. In the Section 4, details about this issue and its solution are discussed.

3.5 *Approaches for Automatic Services Composition*

In the past decade, a number of researches for automatic services composition (specifically web services composition) have appeared. Artificial Intelligence (AI) techniques, specifically AI Planning techniques, were popular for the automatic services composition. However, graph search methods were also used in the automatic services composition.

In the following sub-sections, an overview of AI Planning researches and their applications to the services composition works is provided. Limitations of these approaches presented. The graph search-based approaches are similarly reviewed.

3.5.1 AI Planning-based approaches

AI Planning approaches can be roughly categorized into two main streams. First one is domain-independent AI Planning approaches that try to solve general planning problem without reliance on domain-specific knowledge. Second one is domain-specific AI Planning approaches that directly use domain heuristics to solve domain specific problems.

3.5.1.1 Domain-Independent Planning

In practice, it is not feasible to develop domain-independent planners that work in every possible domain. Thus, typically, most of the domain-independent planning approaches such as classical planning make simplifying assumptions to restrict the set of domains. The followings are the most commonly used assumptions.

- Finite system: states, actions, and events are finite
- Fully observable
- Deterministic: each action has only one outcome
- Static (no exogenous events): no changes but the controller's actions
- Attainment goals: a set of goal states
- Sequential plans: a plan is a linearly ordered sequence of actions
- Implicit time: no time durations; linear sequence of instantaneous states
- Off-line planning: planner doesn't know the execution status

3.5.1.1.1 GraphPlan

GrpahPlan is a general-purpose planner for STRIPS-style domains [Blum and Furst 1997]. The operation of GraphPlan consists of two phases. In the first phase, a forward search is used to build a planning graph. In this phase, the GraphPlan extends a planning graph forward from the initial state until a necessary (but insufficient) condition for plan

exits. In the second phase, a regression search is performed to extract valid plan. In this phase, backward search is performed from the goal, looking for a correct plan.

3.5.1.1.2 Compilation-based Planning

Compilation-based Planning approaches try to solve planning problem by converting it into another generic planning problem such as Satisfiability, Constraint Satisfaction, and Integer Linear Programming. Typical procedure of the compilation-based planning approaches is as follows: 1) set the plan k-length bound, 2) encode the plan into one of the generic planning problems, and 3) solve the problem using an off-the-shelf solver for the generic planning problem. If the solution is found, then the plan is decoded into the original problem. Otherwise, repeat the procedure after incrementally increasing the length bound until the plan is found. Figure 3-2 below show the general procedure of the compilation-based Planning.

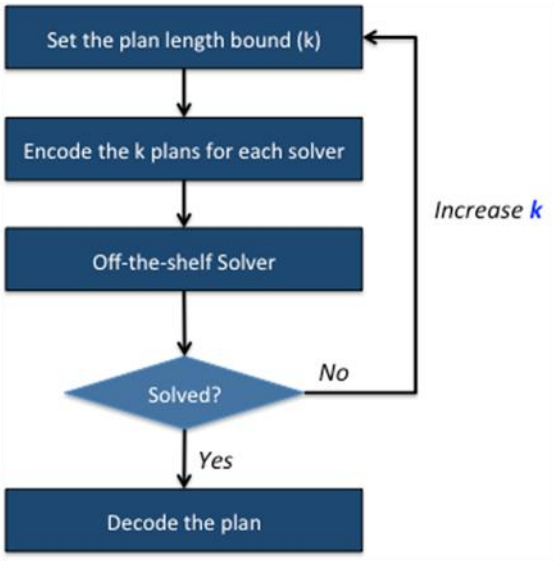

Figure 3-2 General Procedure of compilation-based Planning

3.5.1.1.2.1 Planning as Satisfiability (SAT)

Planning as Satisfiability translates classical planning problems into satisfiability problems, and solves them using highly optimized SAT Solvers such as Davis-Putnam, Local search, and GSAT [Kautz and Selman 1992].

BlackBox [Kautz and Selman 1999] unifies the planning as satisfiability framework with the plan graph approach to STRIPS planning. It builds the planning graph until all goals appear non-mutex, backward relevant analysis to remove irrelevant actions/facts, and encode the remaining graph as SAT.

3.5.1.1.2.2 Planning as Constraint Satisfaction Problem (CSP)

The satisfiability problem can be roughly thought of as certain forms of the constraint satisfaction problem [Do and Kambhampati 2001]. CSP is considered as a better substrate than either SAT or ILP due to its rich structure and the flexibility to represent different types of constraints procedurally.

Planning problems can be fully casted as a constraint satisfaction problem (CSP). The basic modeling units are constraints and variables. A constraint is an entity that restricts the values of variables. In order to use efficient solving techniques, most search frameworks use only a restricted scenario. In propositional satisfiability, constraints are restricted to propositional formulas, which constrain variables to a Boolean domain. In integer linear programming, linear inequalities can be applied to restrict numerical variables. Constraint programming is the most general framework with no restriction on the types of constraints, although usually only variables with finite domains are considered.

3.5.1.1.2.3 Planning as Integer Linear Programming (ILP)

In spite of the general applicability of a SATplan, the propositional representations used in SAT solvers also have some inherent limitations that is it is impossible to incorporate numerical constraints [Vossen et al. 2000]. For instance, converting a boolean linear inequality into a propositional representation may require an exponential number of clauses. Numerical constraints (such as capacity and durational constraints) however do arise in many practical, real-world domains, and the ability to incorporate these

constraints would therefore significantly enhance the power of domain-independent planners.

3.5.1.2 Domain-Specific Planning

Domain-specific planning (DSP) is also known as configurable planning. DSP exploits one or a few planning recipes that are specific to a particular type or domain of problems. For example, a recipe for traveling to a distant destination may be 1) buying a ticket for the fly from the local airport to the remote airport, 2) taking a public transportation to the local airport, 3) flying to the remote airport, and 4) taking a public transportation to the final destination. Such recipe or recipes narrow down the search space which is as opposed to the domain-independent planning that considers every combination of transportation modes, providers, and routes. Hierarchical Task Nework (HTN) Planning is a domain-specific planning. It divides the problem into tasks (activities) rather than goals and methods to decompose tasks into subtasks. Figure 3-3 below shows an example of HTN planning.

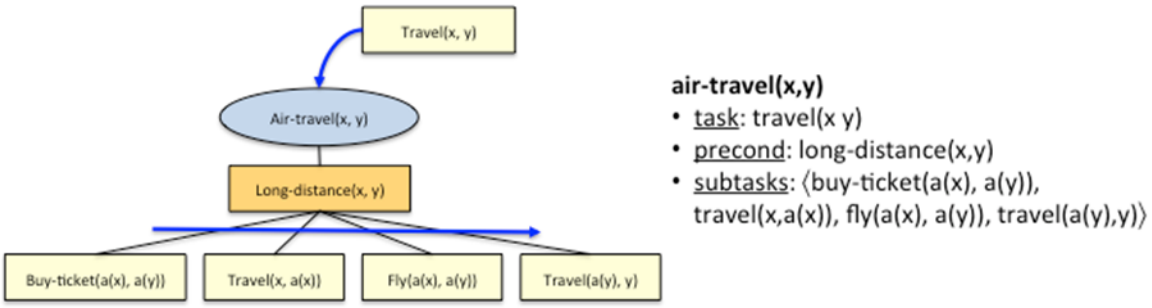

Figure 3-3 Example of HTN Planning

HTN Planners provide a construct to encode a recipe as a collection of methods and operators. Each recipe provides a standard way to solving a certain problem. As a result, the planning system doesn't necessarily have to repeatedly derive solutions, every time it solves a problem. However, disadvantage of the HTN Planning is writing a knowledge base can be more complicated than just writing classical operators.

3.5.1.3 Discussion

Intuitively, classical planning approaches can be the solution for automatic services composition, which explains why a number of services composition approaches are relying on them. However, we argue that the classical planning approaches may not be the best way for automatic services composition.

First, the problem complexity of the classical planning approaches is typically very high. For instance, in the case of GraphPlan, the forward search requires polynomial time while the regression search requires exponential time. SAT problems have been proven to be a NP-Complete. CSP and ILP are also NP-Complete in general. A number of other NP-Complete problems are expressible as CSP (e.g., propositional satisfiability). 3SAT can be reduced to ILP. Thus, exhaustive search method does not work for the problems.

Local search methods that do not systematically search the whole search space may find solutions quickly on average. Local search methods start with a complete assignment of a value to each variable and try to iteratively improve this assignment by improving steps,

by taking random steps, or by restarting with another complete assignment. Despite the efficiency of local search methods, they do not guarantee that a solution will be found even if one exists. In the case of the HTN Planning, writing a knowledge base can be more complicated than just writing classical operators.

Classical planning has also been designed to deal with problems with different characteristics than those of the services composition problem. The classical planning problems generally have a small number of actions (e.g., moving block) and a large number of objects (e.g., hundreds of blocks). Hence, the planner focuses more on finding an appropriate order of operators to achieve a goal. In addition, the classical planning problem and consequently its approaches were designed out-of-the-box to deal with the interleaving between sub-plans to avoid Sussman anomaly [Nils 2001]. Consequently, classical planning problem solvers spend their computational capacity to validate the interleaving problem even when it is unnecessary.

On the other hand, services composition problems generally deal with a large number of actions (huge number of services on the cloud) and a limited number of objects (e.g., registering one hotel). Hence, a services composition problem solver should focus more on finding appropriate services that are composable and less on the order of services. Moreover, interleaving condition can be checked and prevented a priori in the service composition problems rather than having to validate the whole plan as in the case of classical planning problems. Therefore, the services composition problem (specifically,

Web Services) is a kind of information gathering problem [Kwok and Weld 1996] where new information is gathered by executing a service, which results in output information that is fed into another service. Each resulting output used to trigger subsequent services is immutable (by other services).

3.5.2 Graph-based Planning Approaches

Graph-based planning approaches can be applied to services composition problems. Services, initial states, and goal states can be modeled as vertices, while input and output can be modeled as edges between vertices. This can be done vice versa. Graph search algorithms find path – a set of valid edges connecting the initial state to the goal state. [Hashemian and Mavaddat 2005, Oh et al 2005, and Zhang et al. 2003]

It is straightforward to construct an adjacent list or a matrix to represent a service network graph and obtain the shortest path from the source to the goal vertex using existing well-known shortest path finding algorithms such as the Bellman–Ford Algorithm [Bellman 1956, Ford 1956, and Moore 1959].

However, there are some limitations in the existing works. First, the existing works only support a graph with single input and single output per vertex as shown in Figure 3-4. Therefore, the result of the existing shortest path finding methods is always a linear path from the source vertex to the goal vertex. In addition, the existing methods only work with a single cost model and hard constraint associated with each of the edges. In services

composition problems multiple cost models, hard constraint, as well as software constraint are present to represent the complex characteristics of services. To that effect, the existing shortest path finding methods cannot deal with these additional parameters.

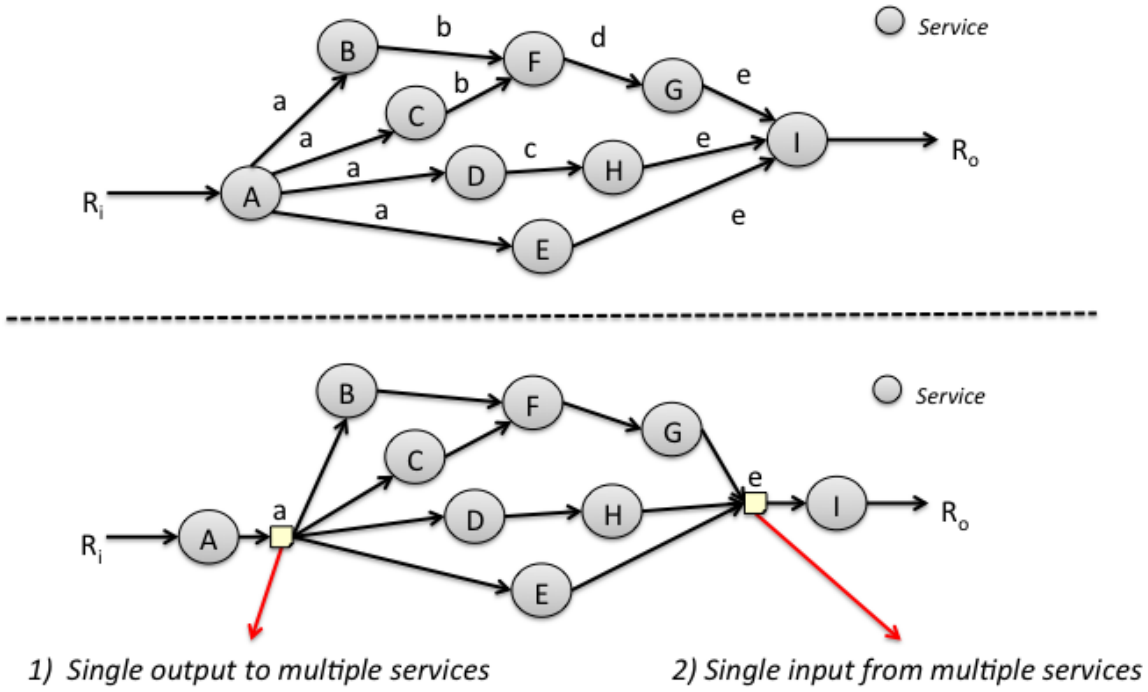

Figure 3-4 Issues in existing graph-based approaches

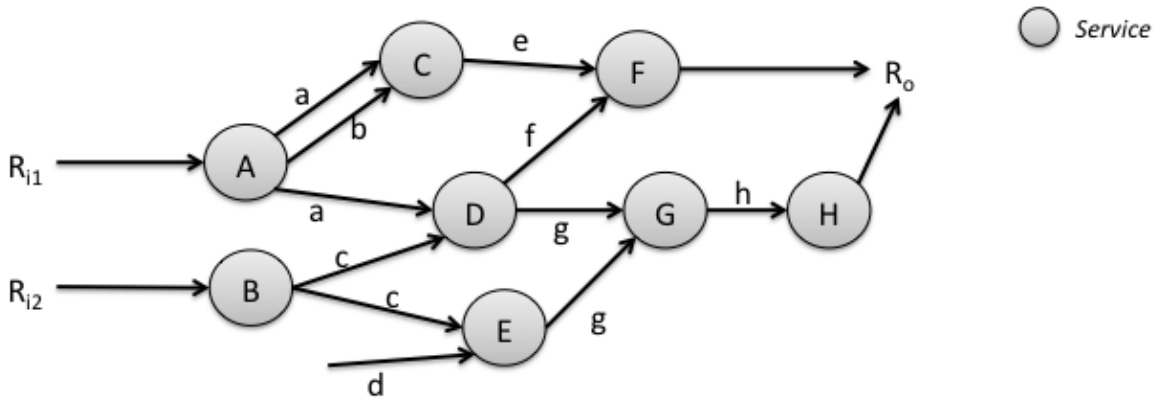

Figure 3-5 more realistic service network

Figure 3-5 illustrates the complexity of service network that have multiple inputs and outputs on several vertices. It can be seen that a linear path cannot take the initial state to the goal state. The planning objective is to find non-linear, shortest plan with multiple alternatives within the polynomial time.

3.5.3 Conclusion

Services composition problems can be viewed as a classical planning problem. Services can be considered as operators and the problem is to find a sequence of services that transits the initial condition to the desired condition through matching outputs and post-conditions of one service with inputs and pre-conditions of the next service. The main issue with classical planning is the assumption that interleaving of operators between different sub-plans cannot be checked before choosing the next operator (or service) and consequently wasting computational resource to validate the whole plan at every

planning step. However, interleaving can be checked and prevented before selecting the next service in the services composition problem. In addition, the services composition problem needs to focus more on choosing the right operator and less on the ensuring the right sequence of operators. For these reasons, a planner for the services composition problem should be tailored to optimize for these different characteristics

Graph based planning approaches can be more suitable for the services composition problem from the computational complexity perspective. However, current existing works were not designed to address the complex relations between today's services.

In order to address this issue, characteristics of services need to be formally expressed in the service description. Therefore, function representation theories, which can provide the basis for a service description specification, are investigated next.

Chapter 4. Function and Service Representation Method

Before developing services composition method, one of the most important questions is what services composition means. That is, when we say “Services X and Y are composable”, what must be satisfied? For example, we may hear the following statements - “In injection molding, clamping unit and injection unit are composable, because clamping unit pushes the mold halves together and exerts sufficient force to keep the mold securely closed while the material is injected by the injection unit.”, “An MTConnect client service and predictive model building service are composable, because MTConnect client service can provide process information of the machine to build a predictive model.”, “A Create ECO service and a Validate ECO service are composable, because the output of the Create ECO service can be consumed by the Validate ECO service.”, or “Two different ebMS (ebXML Messaging Service) are composable, because both follow a common standard specification”. More generally, these statements are of the form “services X and Y are composable”, by which it is variously meant that X and Y are functionally composable, non- functionally composable, and/or are based on same standard specification, and so on.

These different statements illustrate that the definition of services composition differs depending on types of services (e.g., hardware or software) and aspects of composability (e.g., functional or non-functional).

The diversity of services composition definitions appears in existing works. Some of the definitions are generally similar in concept, but often differ in level of details or scope depending on approaches used to solve the services composition problem. For instance, Zeng et al. (2003) considered five generic quality criteria (execution price, duration, reputation, reliability, and availability) for service composition. On the other hand, Oh et al. (2008) just focused on matching input and output parameters between services and user's initial and goal states. Milanovic and Malek (2004) suggested that services composition must satisfy connectivity, non-functional quality-of-service properties, correctness, and scalability.

The objective of this chapter is to analyze what must be considered, to make different services composable, whether the condition differs in different types of services (software or hardware) as well as what various aspects of composability are (functional or non-functional).

4.1 *Functional Characteristics*

Functional composability deals with functional characteristics of the services. In software engineering, identifying functional requirements is essential to development of any software systems. The functional requirements would be manifested as functional characteristics of the developed software systems. We can extend this notion to services as well, because every service is created to satisfy a certain requirement and the

requirement can be manifested as functional characteristics. Thus, identifying functional requirements would help identify services. We investigated existing works in requirement engineering research area.

4.1.1 Functional Requirement

According to Glinz (2007), in requirement engineering research area, there is a broad consensus on the definition of the term, functional requirements, in two main threads. The first thread emphasizes function. Suzanne and James (1999) stated that a functional requirement specifies "a function that a system (...) must be able to perform", while Sommerville (2004) stated that functional requirement specifies "what the system should do". They used different terms, but commonly stated that functional requirement specifies function. The second thread emphasizes behavior. Anton (1997) stated that functional requirements "describe the behavioral aspects of a system". And Davis (1993) stated that "those requirements that specify the inputs (stimuli) to the system, the outputs (responses) from the system, and behavioral relationships between them; also called functional or operational requirements." According to IEEE 830 (IEEE 1998), functional requirements should define the fundamental actions required to process the inputs and generate the outputs.

There are various terms that are not clear in the functional requirement definitions. For instance, it is not clear what the 'function' means in Suzanne and James (1999) and what

the ‘behavioral aspects’ means in Anton (1997). Thus, it is necessary to clarify the meaning of function and behavior to identify the functional characteristics in more detail.

4.1.2 Function as Behavior and Effect

For a clear definition of function and behavior, we investigated existing works in function representation research areas and found that there are a number of definitions in existing works as described in the section 3.4.

As Chandrasekaran (2005) stated, “the various terms in the definitions are not clear as in the functional requirement definitions.” For instance, it is not clear whether one author means by behavior or action is exactly the same as meant by another author. Crilly (2013) raised an issue that statements about functions can be interpreted in different ways such that function definitions are relative or subjective and they are just labels that people assign to things to reflect how they think about them. The relativeness or subjectiveness would hinder objective representation of functional characteristics. For example, let’s take Deng (2002)’s definition to capture a purposive functional characteristics of ‘electric motor’. Then, what is the purpose of the electric motor? It has a wide variety of purposes and is found in clocks, drills, fans, fridges, hair dryers, vacuum cleaners, hard disk drives, DVD players, and industrial equipment including lathes, mills, and so on. Since the purpose of ‘electric motor’ depends on the context or the perspective taken, we cannot define a single, absolute definition for that.

However, it is observed that the device-environment distinction (Chandrasekaran and Josephson, 2000) covers all the meanings in the different definitions, because the distinction clearly relates two common meanings of function. Let's take a look at Chandrasekaran and Josephson' definition. It is clear that the device-centric function corresponds to the behavior of a given system (The term, system, here can be any of device, service, component or so on.), stated "in terms of variables associated with specific structural elements". Thus, the device-centric function might cover 'operational function' in Chittaro and Kumar (1998), 'intended behavior' in Chakrabarti (1998), 'action' in Chakrabarti and Bligh (2001), and 'action function' in Deng (2002). Although the terms vary, they commonly mean 'objectively observable behavior'.

On the other hand, environment-centric function corresponds to the system's effect on the environment, stated "entirely in terms of elements external to the device." Thus, the environment-centric function might cover 'purposive function' in Chittaro and Kumar (1998), 'purpose' in Chakrabarti (1998), 'effect' in Chakrabarti and Bligh (2001), and 'purpose function' in Deng (2002). That's because the effect of the environment-centric function has close relation with the purpose or role of the system. Chandrasekaran and Josephson (2000) introduced the concept, 'Mode of Deployment', that enables to assign specific context to function. For example, the effect of electric lamp is 'room illumination', if the lamp placed in room and switch is turned on. The condition for the effect called 'Mode of Deployment'. Thus, we can generalize the effect of the electric

lamp as ‘illuminate something or somewhere’. Then, the effect can be exactly matched with the purpose of the electric lamp, because both the designer and user of the electric lamp have the purpose, ‘illuminate something or somewhere’. Chandrasekaran (2005) explains that the device-centric function is the mean to achieving the environment-centric function and this statement also implies that the environment-centric function has a close relation with purpose.

Crilly (2013) argued that a system can provide many effects (or satisfy many purposes), but not all of which are necessarily functions. In the motor example, for instance, the motor does not just convert the electrical energy into the rotational energy; it also generates heat and noise. The different outputs of the electric motor would cause different effects on its environment and satisfy different purposes. Crilly (2013) stated that “Depending on the perspective taken, it is variously argued that a device’s functions are restricted to the roles that it was intended to play (e.g. by a designer), is used to play (e.g. by someone able to operate it), has been selected for playing (e.g. by market forces), and so on.” Therefore, the effect or purpose can be derived from either a designer or user. However, the distinction is not crucial for our objective, because identifying the effect as an important characteristic of function is sufficient for the objective of our research.

4.1.3 Functional Characteristics for Services Composability

In the previous section, we have identified that behavior and effect are important characteristics of function. In this section, we discuss the relation between the two characteristics and composability.

4.1.4 Behavior: Input and Output

As discussed in the previous section, the term ‘behavior‘ means something objectively observable without any context. Then, what can we objectively observe in the system? Let’s go back to again the electric motor example. What we can objectively observe is that when the electric motor is fed with the electrical energy, it generates a rotational kinematics. It also generates other things like noise and heat, but let us focus on the rotational kinematics. Then, the electrical energy corresponds to input of the electric motor and the rotational kinematics corresponds to output of the electric motor. Figure 4-1 below shows simple modeling of the electric motor. The electric motor has a function, called *convert*, which has *electrical energy* as an input and *rotational kinematics* as an output.

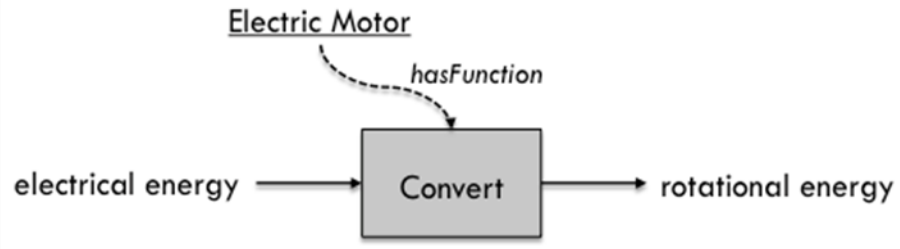

Figure 4-1 Composability with input and output

The input and output are important characteristics for services composability, because, in general, the output of one service must be matched with the input of the other service to be composed. Thus, any system to be composed with the electric motor must have the electrical energy as an output or the rotational kinematics as an input. In some cases, the matching may not be exact. For instance, in the case that the output is subsumed by the input, we can still say that the two systems are composable. It is clear that the input and output are important characteristics of composability.

4.1.5 Effect: Pre-condition and Post-condition

Another important characteristic of function is the effect. The effect is what a system should have on its environment. Let's take a look at the following example shown in Figure 4-2.

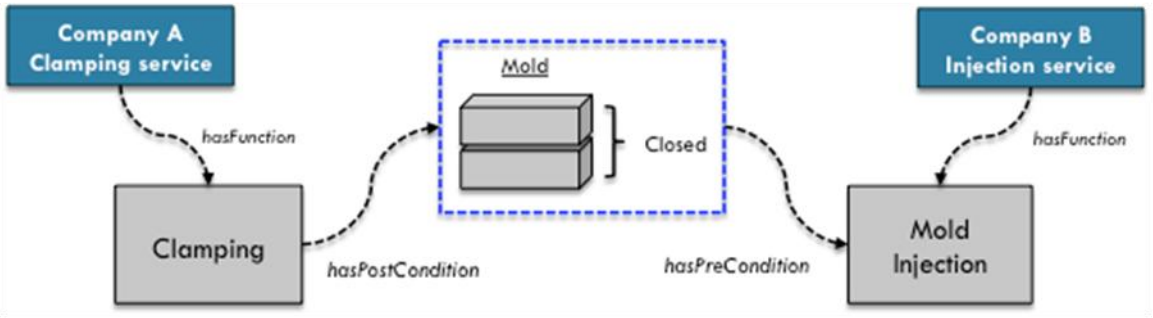

Figure 4-2 Composability with pre-condition and post-condition

Company A's Clamping Service has a function called *Clamping* and *Company B's Injection Service* has a function called *Mold Injection*. Each service also has input and output, but let's just focus on the effect, specifically the effect of *Clamping*. The effect of Clamping is to push each of mold halves together and exerts sufficient force to keep the mold securely closed for the *Mold Injection*. The mold halves are entirely external to the Clamping, thus, according to Chandrasekaran and Josephson (2000), that's environment-centric function of the Clamping and the effect of the function is 'Mold – Closed'. Since, prior to the injection of the material into the mold, the two halves of the mold must first be securely closed, the effect, 'Mold – Closed' is prerequisite to perform the *Mold Injection* function. Thus, the effect is also very important when we determine whether two services are composable.

In order to define more appropriate term for the effect, if an effect must be present before performing a function, we call the effect as Pre-condition, and if an effect occurs after execution of a function, then we call the effect as Post-condition.

4.2 *Non-functional Characteristics*

Similar to Section 4.1, this section discusses first about non-functional requirements and then identifies non-functional characteristics.

4.2.1 Non-functional Requirement

Glinz (2007) summarized several definitions for non-functional requirement from the software engineering discipline. The followings are part of those definitions.

- Davis (1993): The required overall attributes of the system, including portability, reliability, efficiency, human engineering, testability, understandability, and modifiability.
- IEEE 610.12 (IEEE 1990): The standard distinguishes design requirements, implementation requirements, interface requirements, performance requirements, and physical requirements.
- IEEE 830 (IEEE 1998): The standard defines the categories of functionality, external interfaces, performance, attributes (portability, security, availability, reliability, Maintainability), and design constraints.
- Jacobson et al. (1999): A requirement that specifies system properties, such as environmental and implementation constraints, performance, platform dependencies, maintainability, extensibility, and reliability. A requirement that specifies physical constraints on a functional requirement.
- Mylopoulos et al. (1992): "... global requirements on its development or operational cost, performance, reliability, maintainability, portability, robustness, and the like. (...) There is not a formal definition or a complete list of nonfunctional requirements."

- Suzanne and James (1999): A property, or quality, that the product must have, such as an appearance, or a speed or accuracy property.
- Wiegers (2003): A description of a property or characteristic that a software system must exhibit or a constraint that it must respect, other than an observable system behavior.

As Glinz (2007) indicated, there are not only terminological, but also major conceptual discrepancies in these definitions. However, it is not necessary to have a single clear definition of non-functional requirement to meet the objective of this research. It is more important to identify non-functional characteristics that should be considered for composability. It can be seen that the above definitions commonly used the following terms: *Property*, *Attribute*, *Quality*, *Constraint*, and *Performance*. In the next section we use these terms to extract non-functional characteristics.

4.2.2 Non-Functional Characteristics for Composability

We may roughly classify non-functional requirements based on the above definitions into two types – *constraint-related* and *quality-related*. Constraint-related non-functional requirements typically mean requirements that a system must satisfy, while quality-related non-functional requirements usually means a determinant factor to ensure customer satisfaction. *Constraint* related concepts such as 'Physical constraints' in Jacobson et al. (1999), and 'constraint that system must respect' in Wiegers (2003) are examples of the constraint-related non-functional requirements. Although IEEE 610.12

(IEEE 1990) does not use the term, *Constraint*, but 'interface requirements' could be a type of *Constraint* in terms of composability, because interface is the means of connecting different services and therefore services' interfaces must be compatible for the services to be composed. On the other hand, *Quality* and *Performance* related definitions can be the quality-related non-functional requirements. Some definitions related to *Property* and *Attribute* cannot be clearly classified into *Constraint* or *Quality*. For example, in IEEE 830 (IEEE 1998), *Attributes* are a collection of some part of qualities, except performance and constraints, while Davis (1993) defined that every non-functional requirement is *Attribute* including constraints. *Property* also has some issues such that Jacobson et al. (1999) considered *Constraint* as *Property*, while Wiegers (2003) excluded *Constraint* from *Property*. Thus, our attempt is to capture underlying meanings of specific characteristics rather than directly following the definitions to how they classify these characteristics into constraint or quality.

For our objective, the constraint-related non-functional requirements seem to be better than quality-related ones. That is because our focus is non-functional characteristics that must be satisfied to enable composition of different services. Then, what constraints should be considered in the service composition?

4.2.3 Constraints on Input

Chandrasekaran and Josephson (2000) stated that function can be defined in terms of behavioral constraints that we wish to be satisfied under certain conditions and the

behavioral constraint is any constraint on the behaviors of an object or on an object configuration. An example of the behavioral constraints they have provided is “the value of output voltage is greater than 5 volts”.

This example just shows a predicate defined on behavior. The constraints consist of variables and values. For instance, *output voltage* is a variable and *5 volts* is a value of the variable. One interesting observation here is that, at this point, it is not clear whether the behavioral constraint in the example is relevant to composability or not. That’s because the constraints above just describe how the device itself operates in a certain condition or in general.

In another example provided by Chandrasekaran and Josephson (2000), “if the input voltage is above 5, the output voltage is to be a sinusoid.” This example shows a predicate defined on behavior in a certain condition. In this example, we can figure out that the variable and value in the condition are relevant to composability, because the behavioral constraints impose a restriction on the input, and this implies that only the services that satisfy the restriction (e.g., the value of output voltage is greater than 5 volts) can be composed with the system.

What we observe here is that constraints on input are closely relevant to composability. Then, how about the constraints on output? It is apparent that, in order to be relevant to composability, the constraints on output should be paired with variables of the other systems to be composed. Otherwise, the constraints are just for the system itself.

4.2.4 Looks Quality, but Constraints

One important issue found is that the boundary between *Constraint* and *Quality* is blurry, because some of *Quality* related requirements can be *Constraint* when a customer requires a specific level of quality. For example, 'accuracy' in Suzanne and James (1999) is related to *Quality*. Suppose that a customer requirement is "accuracy must be higher than 99%". Then, the accuracy is not *Quality* related requirement any more, but becomes *Constraint.*

This can be observed in service composition as well. For example, let's go back to the Injection Molding example such that *Company A* provides a *Clamping* service and *Company B* provides an *Injection* service. The *Company A*'s clamping service has a non-functional characteristic, *Clamping force*. It is not definitely clear whether the *Clamping force* is *Constraint* or *Quality*, but it seems to be closer to *Quality* at this point. However, it would be apparent when composing the two services. As shown in Figure 4-3, in order to make the two services composable, the *Clamp force, X* must be greater than the *Injection pressure, Y* of *Company B*'s *Injection* service. That's because the *Clamping* service has to push the mold halves together and exert sufficient force to keep the mold securely closed while the material is injected by the *Injection* service. Thus, in the perspective of composability, *Clamp force* (*Injection pressure* as well) should be considered as *Constraint* rather than *Quality*, as a result, it becomes very important non-functional characteristic of *Clamping* service.

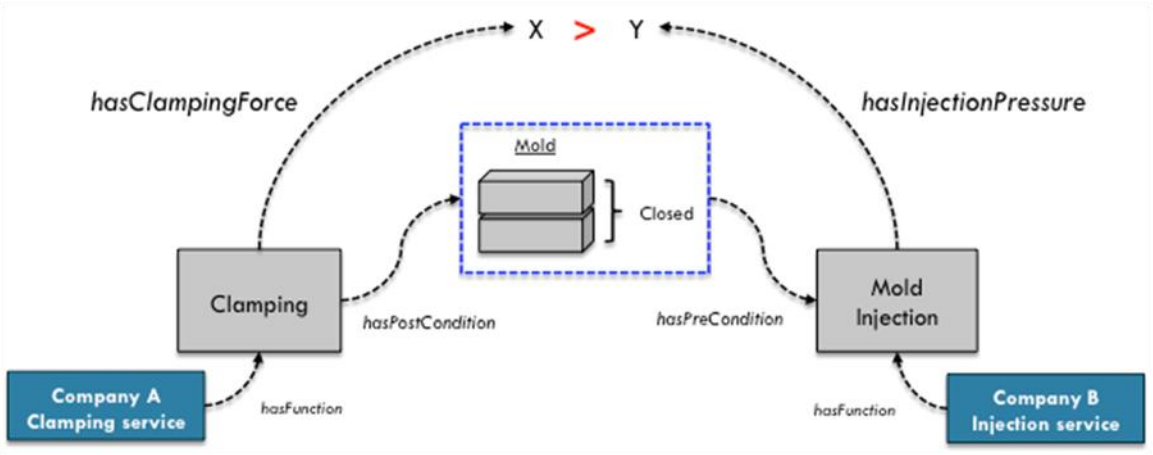

Figure 4-3 Looks quality, but actually constraint in composition

Therefore, we should take into account not only *Constraint* related characteristics, but also some of *Quality* related characteristics that can be transferable to *Constraint* when to be composed.

4.3 *Function and Service Representation*

In previous sections, we identified necessary functional and non-functional characteristics that are relevant to composability. In this section, we provide a formal representation of a function and service based on what we identified.

4.3.1 Resource and State Ontology

The Resource and State ontology defines the concepts and relationships used to describe and represent the input, output, pre-condition, and post-condition. Specifically, the Resource ontology is used to represent an artifact that is consumed or produced by the

function. External artifacts that may affect the operation of the function also can be represented by the Resource ontology in the form of pre/post-condition. The State ontology is used to represent the state of the artifact. The role of the Resource and State ontology is to help reduce the ambiguities that may exist on the terms used in describing the input, output, and pre/post-condition. The Resource and State ontology would be the basic building blocks for inference techniques as we model the ontology using OWL.

In practice, the Resource and State ontology can be very complex or very simple. It depends on how complex the domain is. Some of domains may only need to use even small set of concepts, and mostly focus on the logic while some of domains may need more complex set of concepts with complex reasoning procedures.

How to develop the Resource and State ontology is out of scope of our research. We assume that there exists the Resource and State ontology and all the input, output, and pre/post-conditions are described by using the ontology.

4.3.2 Representing a Function

A function *F* has five sets of parameters:

$F = \{I, O, Pre, Post, Prop\}$, where

$I = \{I_1, I_2, I_3, \dots\}$ = a set of inputs that are consumed by the function *F*.

- Input parameter I_i = {*Resource*, *State*} is a pair of a *Resource* and *State* where *Resource* and *State* are concepts (e.g., owl:class) defined in the resource and state ontology respectively.
- *Resource* is mandatory but the *State* is optional. The State is specified only when there exists constraint on the input, i.e., the input must have a specific state.
- In order to invoke the function *F*, all input parameters must be provided.

$O = \{O_1, O_2, O_3, \ldots\}$ = a set of outputs that are produced by the function *F*.

- Output parameter O_j = {*Resource*, *State*} is a pair of a *Resource* and *State* where *Resource* and *State* are defined in the resource and state ontology respectively.
- The *Resource* is mandatory but the *State* is optional. The State is specified only when the output has a specific state.

$Pre = \{Pre_1, Pre_2, Pre_3, \ldots\}$ is a set of pre-conditions that are predicates that must always be satisfied in order that the execution of function *F* yields the specified outputs and post-conditions. If any of the pre-conditions defined in the function *F* is violated, the result of the execution of the function *F* may or may not produce the specified outputs and post-conditions.

- Pre-condition parameter Pre_k = {*Resource*, *State*}

- Both the *Resource* and *State* are mandatory.
- Note that the pre-condition is to describe the necessary condition on the external artifact not on the input.

Post = {$Post_1$, $Post_2$, $Post_3$, … } is a set of post-conditions that are effects after the execution of the function *F*.

- Post-condition parameter $Post_l$ = {*Resource*, *State*}
- Both the *Resource* and *State* are mandatory.
- Note that the post-condition is to describe the effect on the external artifact not on the output.

Prop = {$Prop_1$, $Prop_2$, $Prop_3$, … } = a set of properties that are characteristics other than the input, output, pre-condition, and post-condition.

- The set of properties does not have specific values, but just list of properties.
- Specific values will be specified when instantiated by a service.

4.3.3 Representing a Service

A service *S* has six sets of parameters:

S = {*F, I, O, Pre, Post, Prop*}, where

F = {F_1, F_2, F_3, … } is a set of functions

$I = \{I_1, I_2, I_3, \ldots\}$ is a set of inputs that are consumed by the service *S*.

- Input parameter I_i = {*resource*, *state*} = a pair of the instance of *Resource* and the instance of *State*.
- The *resource* is mandatory but the *state* is optional.
- *Resource* and *State* are defined in the resource and state ontology respectively.
- In order to invoke the service *S*, all input parameters must be provided.

$O = \{O_1, O_2, O_3, \ldots\}$ is a set of outputs that are produced by the service *S*.

- Output parameter O_j = {*resource*, *state*} = a pair of the instance of *Resource* and the instance of *State* where *Resource* and *State* are defined in resource and state ontology respectively. The *resource* is mandatory but the *state* is optional.

$Pre = \{Pre_1, Pre_2, Pre_3, \ldots\}$ is a set of pre-conditions that are predicates that must always be satisfied prior to the execution of the service *S*. If any of the pre-conditions defined in the service *S* is violated, the result of the execution of the service *S* may or may not carry out its intended work.

- Pre-condition parameter Pre_k = {*resource*, *state*} = a pair of the instance of *Resource* and the instance of *State* where *Resource* and *State*
- Both the *resource* and *state* are mandatory.

Post = {$Post_1$, $Post_2$, $Post_3$, … } is a set of post-conditions that are effects after the execution of the service *S*.

- Post-condition parameter $Post_l$ = {*resource*, *state*} = a pair of the instance of *Resource* and the instance of *State*
- Both the *resource* and *state* are mandatory.

Prop = {$Prop_1$, $Prop_2$, $Prop_3$, … } = a set of properties that are characteristics other than the input, output, pre-condition, and post-condition.

Chapter 5. Ontology Development Method

This chapter provides a model development method that can be applied to the development of function, service, resource, and state model that are described in the previous section. Our model development method consists of several steps including *data preprocessing*, *feature frequency analysis*, *feature selection*, *pattern abstraction*, *pattern specification*, and *evaluation* as shown in Figure 5-1. In this chapter, we will present how this method can be used to develop the function model by identifying various functional characteristics and important features of manufacturing functions.

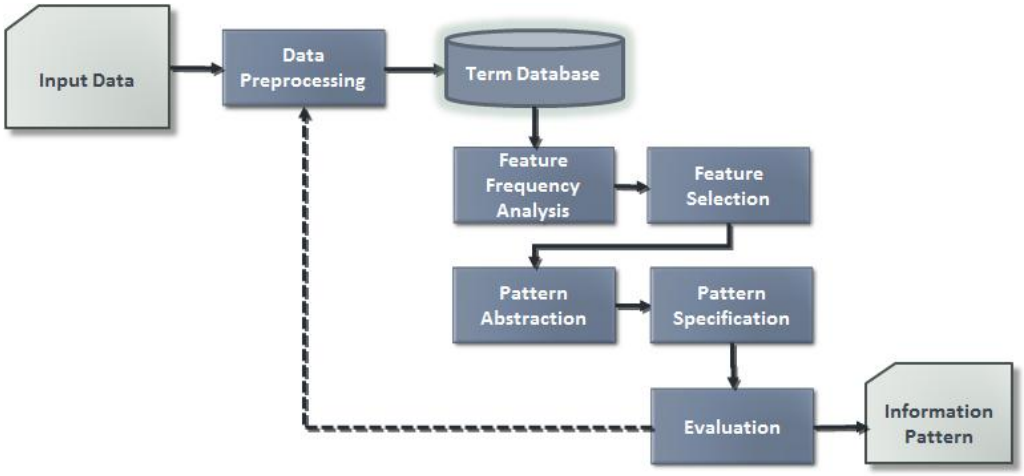

Figure 5-1 Workflow of the inductive information pattern identification

We collected the input data from manufacturers' web sites that advertise their manufacturing service capabilities. Specifically, the target input data is a manufacturing service capability description that is presented in a tabular form.

Many collected terms may have misspells and small deviations from each other. So, syntactical harmonization of the terms is needed to eliminate redundancy. We call this

process the *data preprocessing* and it should be done whenever new input data feeds into the function model development method against all terms which are already preprocessed. All terms and its relations are stored into the *Term Database* after the *data preprocessing*.

The *feature frequency analysis* is a statistical analysis to identify the frequency of features across all descriptions of a certain concept. This analysis provides us a number of important statistical information. (e.g., which features are most commonly used to describe CNC machining function? how many service providers are using a tolerance as important feature of their EDM function?)

The *feature selection* is to find a subset of relevant functional characteristics and features to define a certain function. The *feature selection* is also known as variable selection, attribute selection or variable subset selection. This is needed to find good features to describe each function after eliminating redundant or irrelevant features. Redundant features are those which provide no more information than the currently selected features, and irrelevant features provide no useful information in any context.

The *pattern abstraction* is to identify frequently repeated similar versions of general theory to describe functions. That is, commonly used features in different functional descriptions will be identified by the *pattern abstraction*. For instance, if we find same feature from two different functional descriptions, CNC machining and EDM machining, we can come up with a new upper concept (e.g., machining) of these two different concepts and assign the same features to the new upper concept. Thus, this step helps determine vertical decision boundaries in the model.

The *pattern specification* is to determine horizontal decision boundaries in the feature set which separate features belonging to different functions. This step is based on the probability distribution of the patterns belonging to each function.

After the *pattern abstraction* and *pattern specification*, we can locate each feature into functions vertically and horizontally within a function hierarchy. Vertical location is to move more general features to upper layer in the taxonomy hierarchy, and to move more specific features to lower layer in the taxonomy hierarchy. Horizontal location is to reorganize features to distinguish the functions in the same level.

Finally, the *evaluation* step is to evaluate the generated function model with a classifier. The dotted line from the evaluation to the *data preprocessing* means that the processes may iterate based on the result of the evaluation.

More detailed descriptions of each step will be presented in the following sub-sections.

5.1 ***Input Data***

The advertised manufacturing service capabilities from the manufacturers' web sites are mostly in unstructured/textual descriptions or in semi-structured, tabular form. In our work, the target of the input data is a functional description that is presented in a tabular form. This is because we have found that the tabular form is used by almost all manufacturing supplier's web pages to convey the manufacturing supplier's capabilities explicitly and concisely. And also, the tabular form provides a uniform format such that tokenization into terms and related terms are possible. Note that each input data consists of a concept name and its features. Figure 5-2 shows an example of the input data. We have implemented a HTML Parser to extract terms in tabular form. Additional terms and concepts may exist in textual descriptions as well, but they are rare and few and will not be considered in this research.

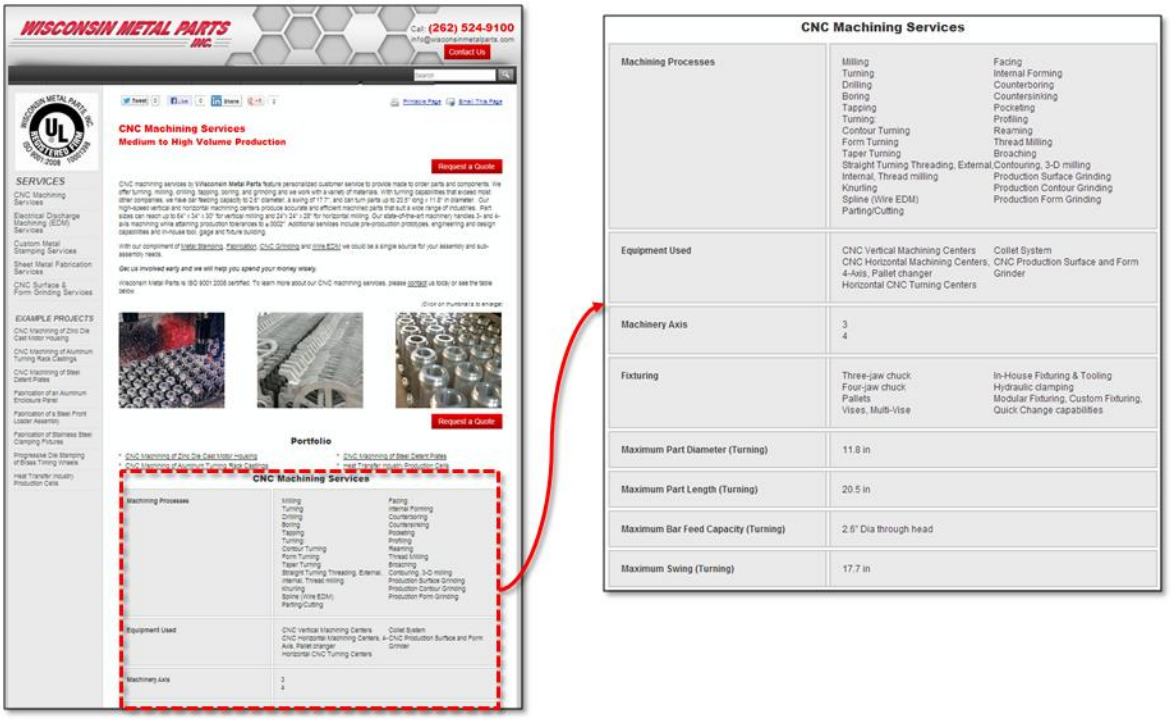

Figure 5-2 An example of the input data

We have implemented a toolkit for our research. Figure 5-3 below shows the *Term Collection* module in the toolkit. The module collects terms from manufacturing supplier's web page by identifying and parsing the HTML table which contains a functional description. The toolkit also provides a convenient user interface for manual cleanup of extracted data where the parser tool cannot cleanly tokenize. This includes removal of remaining special HTML symbols, special character encodings, and decomposition of long sentences into one or more terms. All collected terms can be stored into a relational database or text file (e.g., Microsoft Excel or plain text).

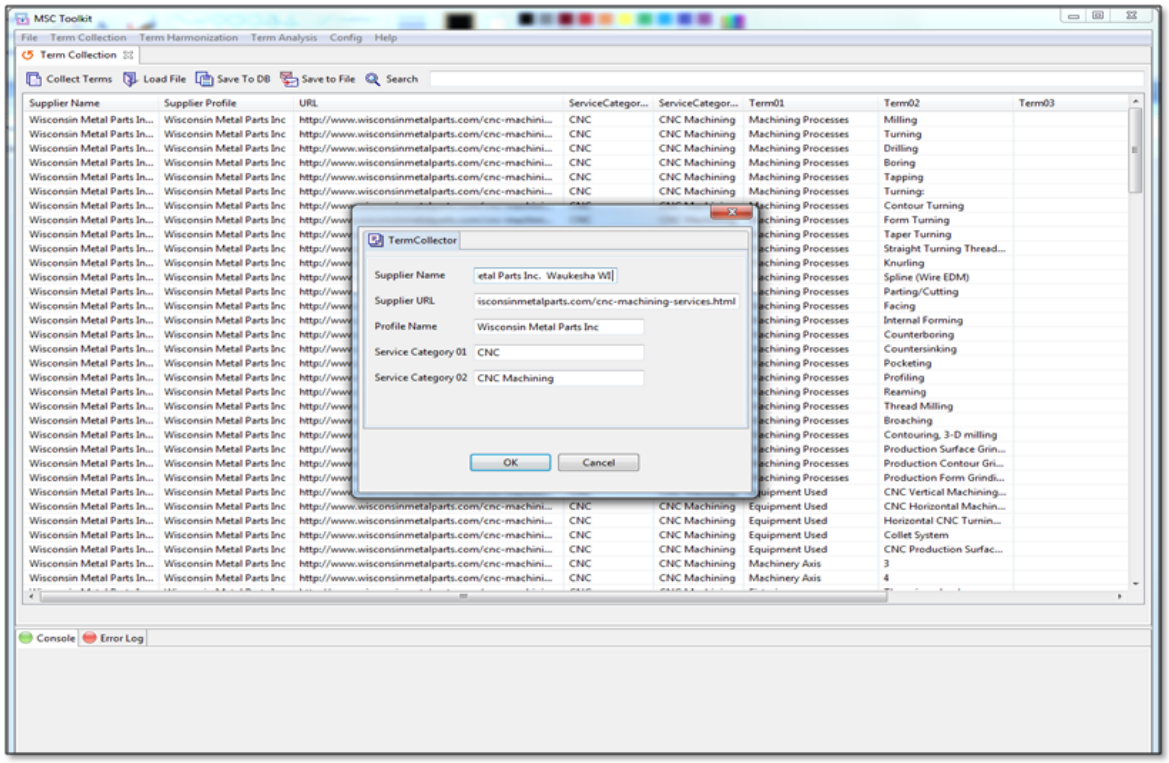

Figure 5-3 Screenshot of web content parsing

Data in all columns in a table are treated in the same manner as terms while their future conceptualization could be as high-level concepts, instances, or properties. Relationships between terms in each row are treated as related while their future conceptualization could be generalization, specialization, instance of, property of, etc. Specific conceptualizations are the future standardization task of this research. Figure 5-4 shows how the contents in HTML table are interpreted.

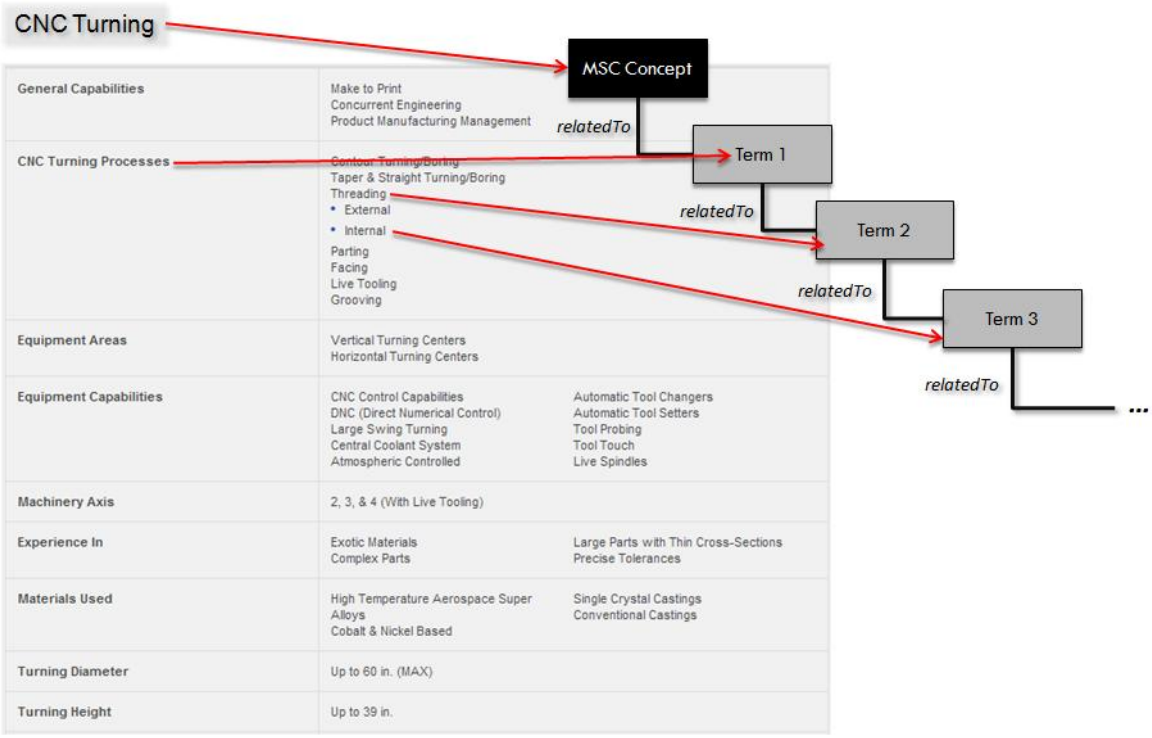

Figure 5-4 Organization of the terms

The first term typically represents a taxonomical concept (e.g., CNC Turning). The interpretation of these terms and their relationships may be that a term is a characteristic/property of the preceding term (the preceding term of term 2 is term 1), an instance of the preceding term, or a property value of the preceding term. This depends on specific situation in each particular row. The terms are left in their original form to preserve the various ways in which they are represented or conceptualized. For example, a diameter capability of a turning service capability description is called ‘Diameter’ by one supplier but it is called ‘Max Diameter’ by another. The associated value term is specified as ‘5 in.’ by one supplier while it is specified as ‘up to 5 in’ by another.

5.2 *Data Preprocessing*

Many collected terms have misspells and small deviations from each other; therefore, data preprocessing should be performed that is largely focused on syntactical harmonization. The primary purpose is to eliminate redundancy for better subsequent analysis. For example, only single term between 'Maximum Tolerance' vs. 'Maximum Tolerances' and between 'In House' vs. 'In-House' is kept. Limited sets of terms are designated as alternatives of another. Only the most obvious ones receive such designation. The reason is that we want the semantic analysis to be a separate step where additional domain knowledge is taken into account. And also, we want to preserve the various ways in which concepts are expressed to be rationalized in the semantic analysis step. In this case, one of the terms within each set of alternative synonym terms is assigned as a preferred term. This is not to say that the preferred term is a standard term; it is just the current convention used to organize the term set for easier analysis.

The *data preprocessing* is a semi-automated procedure supported by the lexical/semantic similarity measure. Lexical similarity measure can be done by the followings.

- Normalization: Typically, labels of ontology artifacts are often concatenations of several terms (e.g., HighVolumeProductionEDM), some of which are abbreviations (e.g., EDM), short-hands (e.g., doc for doctor or for document), or word variations in prefix and suffix (e.g., process vs. processes). Normalization is

to obtain the words in their standard forms from labels by resolving these issues. Techniques such as tokenization, lemmatization, and elimination can be applied. Algorithms may need to be developed to treat normalization issues specific to MSC information. A MSC information specific dictionary or thesaurus may also be helpful.

- String-based similarity measure: This is to measure similarity of two words based on how similar their character strings are. Techniques such as N-Gram, Edit Distance can be used with various metrics such as Jaccard coefficients, Levenshtein's metric, Precision and Recall, F-measure (Baeza-Yates and Ribeiro-Neto, 1999). Soundex (2015), a technique that computes the phonetic similarity between words from their corresponding soundex codes may also be considered.
- Meaning-based word similarity measure: This is to deal with synonyms, hyponyms and hypernyms which have the same or similar meaning of a given word. Measures such as Edge-counting (Rada et al, 1989), weighted distance, and information content based (Resnik, 1995) may be applied based on WordNet or some manufacturing specific thesaurus.
- Combined label similarity: This is to obtain the label-label similarity by combining similarities of individual words in the labels. Both simple methods of weighted sum and more principled methods based on national language processing such as noun-phrase analysis will be considered.

Figure 5-5 shows the data preprocessing module in the toolkit. For the lexical similarity measurement, we have implemented N-Gram and Edit Distance algorithm.

After each similarity measure, a human user goes through the result to identify redundancy or establish alternative relationship. The term set is then updated. Pair wise similarity is then calculated again excluding those terms that have already been identified. This is followed by the human review of the result again. This process recurs until no more term is identified as redundant or alternative to some other terms. The toolkit provides all these functionalities with a convenient user interface.

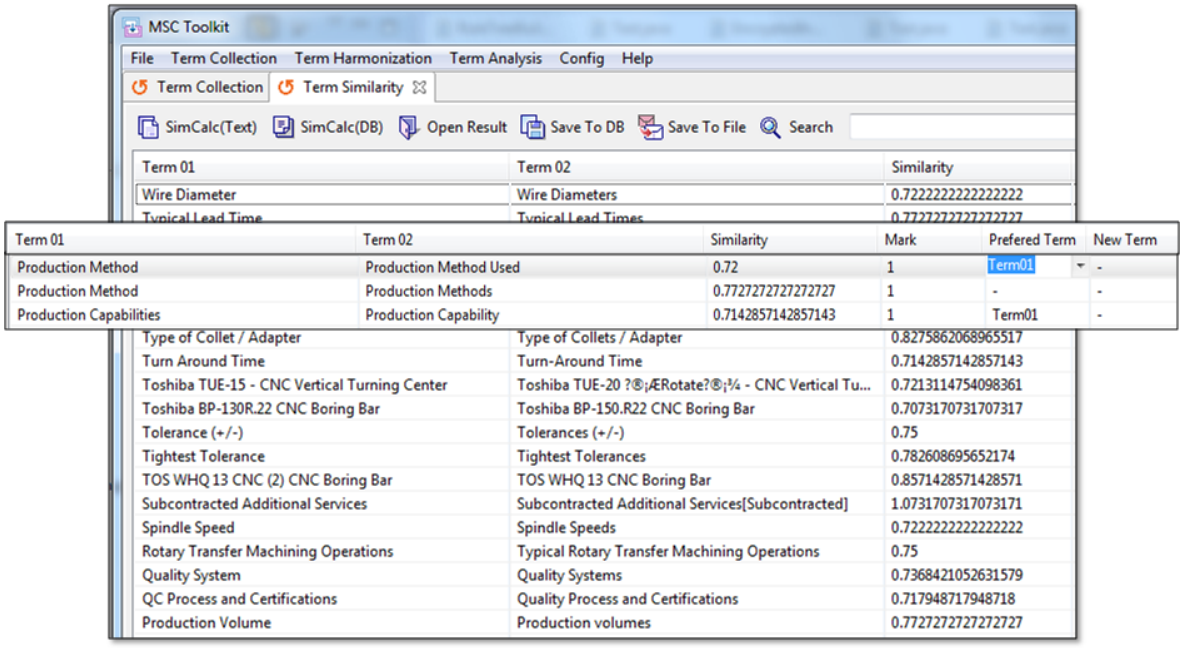

Figure 5-5 Screenshot of data preprocessing

5.3 *Term Database*

All collected terms are stored into permanent storage, called the *Term Database*, after data preprocessing. Figure 5-6 shows the Entity-Relationship diagram of the Term Database.

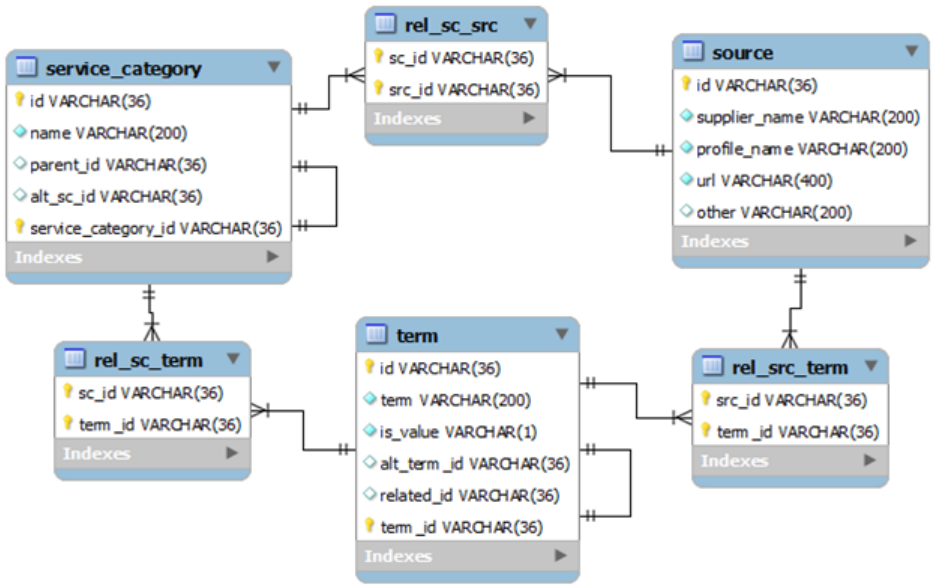

Figure 5-6 Entity-Relationship diagram of the Term Database

As shown in Figure 5-6, the *Term Database* consists of three main tables, and three relation tables to connect the main tables. The three main tables are *service_category*, *source*, and *term*. The three relation tables are *rel_sc_src*, *rel_src_term*, and *rel_sc_term*. The *service_category* table is to store service categories defined by manufacturing suppliers. The hierarchy of the service categories can be encoded by using the *parent_id* attribute. The *source* table is to store manufacturing supplier's information such as

manufacturing supplier's name and web page URL which contains the manufacturing capability information. The *source* table enables to trace the owner of the terms in the *Term Database*. The *term* table is to store manufacturing capability related terms which are extracted from the tabular form in the supplier's web page. The generic and alternative relations between terms can be encoded by using the *related_id* and *alt_term_id* attributes.

5.4 *Feature Frequency Analysis*

Feature frequency is a similar to the document frequency in the *Information Retrieval* research. Feature frequency is used to select some terms, which are frequent across concepts in the term set. Feature frequency can be obtained by simply counting the number of a certain concept (e.g., EDM Machining) that contains a certain feature (e.g., Tolerance) at least once. Let *C* be the collection of a certain manufacturing concept in the term set. Let *F* be the collection of features (unique token) in the collection *C*. We define that the feature frequency, *FF*, as the ratio of the number of a certain concept containing the feature f_i to the total number of a certain concept.

$$FF = \frac{|\{c : f_i \in c\}|}{|C|} \qquad (1)$$

- $|C|$: total number of a certain concept in the term set
- $|\{c : f_i \in c\}|$: number of a certain concept where the feature f_i appears.

Figure 5-7 below shows simple example of the feature frequency. Suppose that there are three *Sinker EDM* concepts in the term set. The circles in the box represent features. For instance, *Sinker EDM 01* has 5 different features including *A*, *B*, *C*, *D*, and *E*. Common features across all three *Sinker EDM*s are represented by blue circle (e.g., A, *B*, *C*, and *D*). The light blue circle (e.g., *E*) represents common features in two different *Sinker EDM*s. The black circle (e.g., *M*) represents the features which are owned by only one *Sinker EDM*. The feature frequency of each features are represented at the right side of each features in the right box of the Figure 5-7. For instance, the feature frequency of *A* is 3/3 = 1, and the feature frequency of *E* is 2/3 = 0.3333 ≈0.3. Another example of the feature frequency, *Ram EDM*, is shown in Figure 5-8.

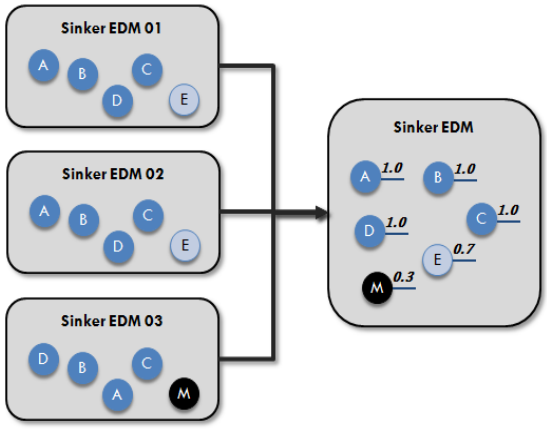

Figure 5-7 Feature frequency of Sinker EDM

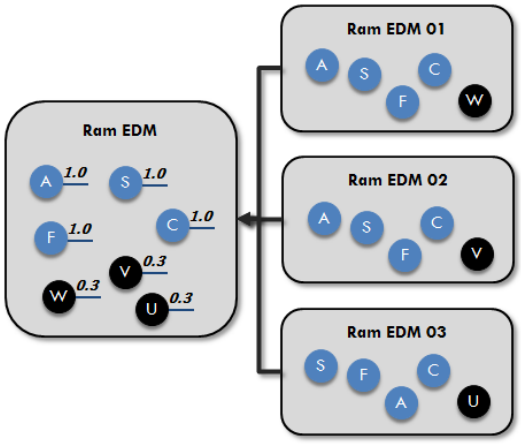

Figure 5-8 Feature frequency of Ram EDM

5.5 *Feature Selection*

Feature selection is the process of selecting a subset of a certain concept in the term set and using only this subset as features when defining a function. In our method, the main purpose of the feature selection is to increase classification accuracy by eliminating irrelevant features. An irrelevant feature is the one that, when added to the document representation, increases the classification error on new data. In the presence of many irrelevant features, learning models tend to over-fit and become less comprehensible. Feature selection is one of the effective means to identify relevant features for dimensionality reduction (Guyon and Elisseeff , 2003).

As stated in the Section 3, feature selection algorithms designed with different strategies broadly fall into three categories: filter, wrapper and embedded models. The filter model relies on the general characteristics of data and evaluates features without involving any

learning algorithm. The wrapper model requires a predetermined learning algorithm and uses its performance as evaluation criterion to select features. Algorithms with embedded model, e.g., C4.5 (Quinlan, 1993) and LARS (Efron et al, 2004), incorporate variable selection as a part of the training process, and feature relevance is obtained analytically from the objective of the learning model. Feature selection algorithms with filter and embedded models may return either a subset of selected features or the weights (measuring feature relevance) of all features. According to the type of the output, they can be divided into feature weighting and subset selection algorithms. Algorithms with wrapper model usually return feature subset.

Figure 5-9 below shows a simple example of the feature selection process with the two different concepts, *Sinker EDM* and *Ram EDM*. In the example, all concepts have *A* and *C* as their feature. The features *B*, *D*, and *E* exist only in the *Sinker EDM* and the features *S* and *F* exist only in the *Ram EDM*. Thus, these features are good for classifying the two different concepts. On the other hand, the features *A* and *C* exist both in *Sinker EDM* and *Ram EDM*. Thus, they provide no more information than the currently selected features (i.e., redundant). The features *M*, *U*, *V*, and *W* are irrelevant features which provide no useful information in any context.

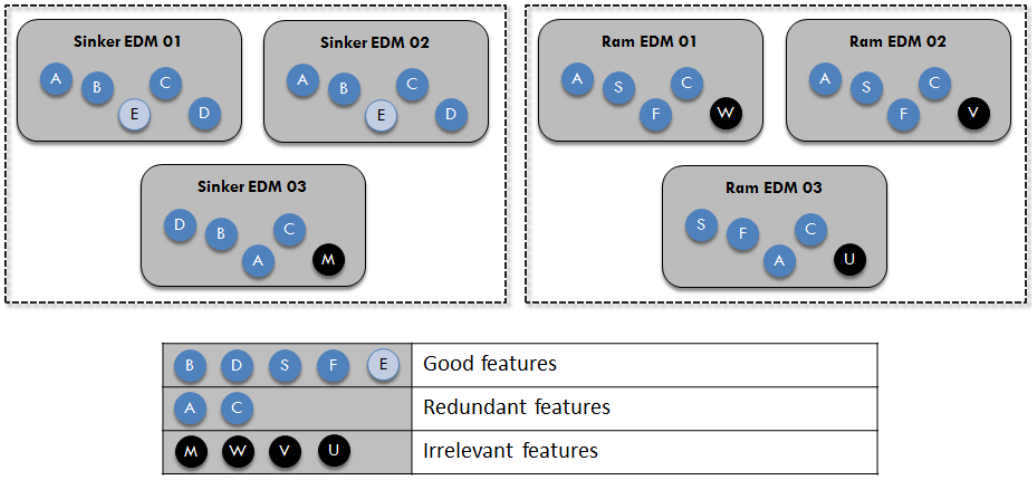

Figure 5-9 Example of feature selection

Since the filter model for feature selection is generally involve a non-iterative computation on the dataset, which can execute much faster than a classifier training session, the filter model has better performance than the wrapper and embedded model. And, the results of the filter model exhibit more generality, because filters evaluate the intrinsic properties of the data, rather than their interactions with a particular classifier. Thus, in our research, we will focus on the filter model.

In following sub sections, we briefly introduce the filter model feature selection algorithms used for our experiment.

5.5.1 Information Gain

Information Gain (Cover and Thomas, 1991) is a measure of dependence between the feature and the class label. It is one of the most popular feature selection techniques as it

is easy to compute and simple to interpret. *Information Gain* (IG) of a feature X and the class labels Y is calculated as

$$\mathrm{IG}(\mathrm{X},\mathrm{Y}) = \mathrm{H}(\mathrm{X}) - \mathrm{H}(\mathrm{X}|\mathrm{Y}). \qquad (2)$$

Entropy *(H)* is a measure of the uncertainty associated with a random variable. *H(X)* and *H(X|Y)* is the entropy of X and the entropy of X after observing Y, respectively.

$$\mathrm{H}(\mathrm{X}) = -\sum_{\mathrm{i}} \mathrm{P}(\mathrm{x}_\mathrm{i})\ \log_2\big(\mathrm{P}(\mathrm{x}_\mathrm{i})\big). \qquad (3)$$

$$\mathrm{H}(\mathrm{X}|\mathrm{Y}) = -\sum_{\mathrm{j}} \mathrm{P}(\mathrm{y}_\mathrm{i}) \sum_{\mathrm{i}} \mathrm{P}\big(\mathrm{x}_\mathrm{i}\big|\mathrm{y}_\mathrm{j}\big)\ \log_2\Big(\mathrm{P}\big(\mathrm{x}_\mathrm{i}\big|\mathrm{y}_\mathrm{j}\big)\Big). \qquad (4)$$

The maximum value of information gain is 1. A feature with a high *Information Gain* is relevant. *Information Gain* is evaluated independently for each feature and the features with the top-k values are selected as the relevant features. *Information Gain* does not eliminate redundant features.

5.5.2 Chi-square

Chi-square (Liu and Setiono, 1995) is used to assess two types of comparison: tests of goodness of fit and tests of independence. In the feature selection, it is used as a test of independence to assess whether the class label is independent of a particular feature. *Chi-square* score for a feature with r different values and C classes is defined as

$$X^2 = \sum_{i=1}^{r}\sum_{j=1}^{c}\frac{(n_{ij} - \mu_{ij})^2}{\mu_{ij}}, \qquad (5)$$

where n_{ij} is the number for samples with the i^{th} feature value. And

$$\mu_{ij} = \frac{n_{*j}\, n_{i*}}{n}, \qquad (6)$$

where n_{i*} is the number of samples with the i^{th} value for the particular feature, n_{*j} is the number of samples in class j and n is the number for samples.

5.5.3 CFS

CFS (Hall and Smith, 1999) uses a correlation based heuristic to evaluate the worth of features.

$$\text{Merit}_S = \frac{k\overline{r_{cf}}}{\sqrt{k + k(k-1)\,\overline{r_{ff}}}} \qquad (7)$$

Here $Merit_S$ is the heuristic "merit" of a feature subset S containing k features, and $\overline{r_{cf}}$ is the mean feature class correlation and $\overline{r_{ff}}$ is the average feature inter-correlation. $\overline{r_{cf}}$ is defined as follow.

$$\overline{r_{cf}} = \sum_{f_i \in S}\frac{1}{k}\sum(f_i, c) \qquad (8)$$

The mean feature-class correlation (numerator) is an indication to how easily a class could be predicted based on the feature. And the average feature to feature inter correlation (denominator) determines correlation between the features which indicates the level of redundancy between them. Feature correlations are estimated based on the information theory that determines the degree of association between features. The amount of information by which the entropy of *Y* decreases reflects the additional information about *Y* provided by *X* which is measured via *Information Gain*. Since, *Information Gain* is usually biased in favor of features with more values, symmetrical uncertainty is used. The symmetrical uncertainty is defined as:

$$\mathrm{SU}(\mathrm{X},\mathrm{Y}) = 2\left[\frac{\mathrm{IG}(\mathrm{X}|\mathrm{Y})}{\mathrm{H}(\mathrm{X}) + \mathrm{H}(\mathrm{Y})}\right], \qquad (9)$$

where $\mathrm{IG}(\mathrm{X}|\mathrm{Y})$, $\mathrm{H}(\mathrm{X})$ and $\mathrm{H}(\mathrm{X}|\mathrm{Y})$ are defined in the section 5.5.1.

CFS explores the search space using the *Best First* search. It estimates the utility of a feature by considering its predictive ability and the degree of correlation (redundancy) it introduces to the selected feature set. More specifically, *CFS* calculates feature-class and feature-feature correlations using symmetrical uncertainty and then selects a subset of features using the *Best First* search with a stopping criterion of five consecutive fully expanded non-improving subsets. Benefits of *CFS* are it does not need to reserve any part of the training data for evaluation purpose and works well on smaller data sets. It selects

the maximum relevant feature and avoids the re-introduction of redundancy. But, one of the shortcomings is that *CFS* cannot handle problems where the class is numeric.

5.6 *Pattern Abstraction*

Pattern abstraction is a process of identifying some set of common features in different concepts, and forming an upper concept on that basis. The *Feature Abstraction* might be formed by reducing some features of a concept, typically to retain only features which are relevant for a particular purpose. For instance, abstracting a *Sinker EDM* to the more general idea of an *EDM Machining* retains only the information on general *EDM Machining* attributes and process, eliminating the other characteristics of that particular *Sinker EDM*. Figure 5-10 below shows the relation between two different concepts based on the feature frequency analysis on the *Sinker EDM* and *Ram EDM*.

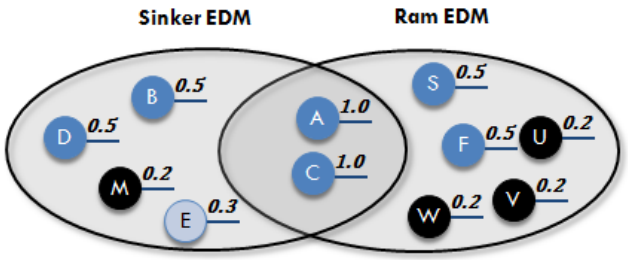

Figure 5-10 Feature frequency between Sinker EDM and Ram EDM

As shown in Figure 5-10 above, the two features *A* and *C* are common features in *Sinker EDM* and *Ram EDM*. The Feature *Abstractions* process reduces the features which are specific to *Sinker EDM* and *Ram EDM*, and forms an upper concept, *EDM*, with these

common features. Figure 5-11 below illustrates the *Feature Abstraction* process. In this example, since we illustrate the *Feature Abstraction* process using only two concepts, the common features are clearly identified. However, in practice, there would be hundreds of concepts to be analyzed for identifying some set of common features. Thus, we need to determine a threshold which indicates the decision boundary using the probability of commonality. For instance, suppose that we have 100 concepts, all concepts have feature *A*, 90 concepts have feature *B*, and 70 concepts have features *C*. If we determine the threshold as 80, only *A* and *B* are considered as common features in the concept set.

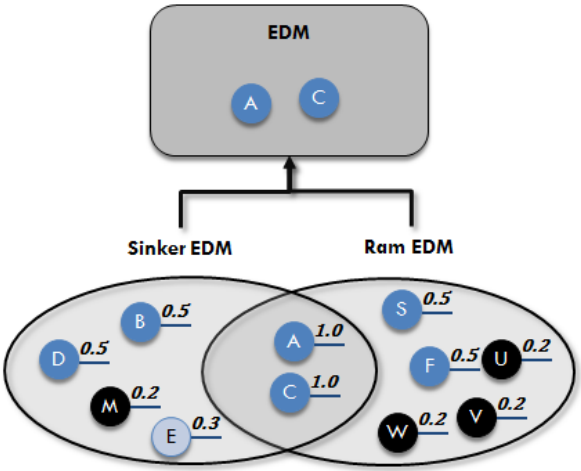

Figure 5-11 Example of feature abstraction from Sinker EDM and Ram EDM

Since there are a number of MSC concepts, how to group MSC concepts as a target of the pattern abstraction is the key research issue in the pattern abstraction step. The clustering techniques might help identify MSC concepts which have similar features with each other.

5.7 *Pattern Specification*

After *Feature Abstraction*, all common features in a set of concept are moved to upper concept. The *Pattern Specification* is to specify each concept in the set of concepts using the remaining features. As a result, those remaining features in each concept occupy compact and disjoint regions in a feature space. For instance, in Figure 5-12 below, the features *B*, *D*, *E*, and *M* are the remaining features in *Sinker EDM*, and the features *F*, *S*, *U*, *V*, and *W* are the remaining features in *Ram EDM* after the *Features Abstraction*.

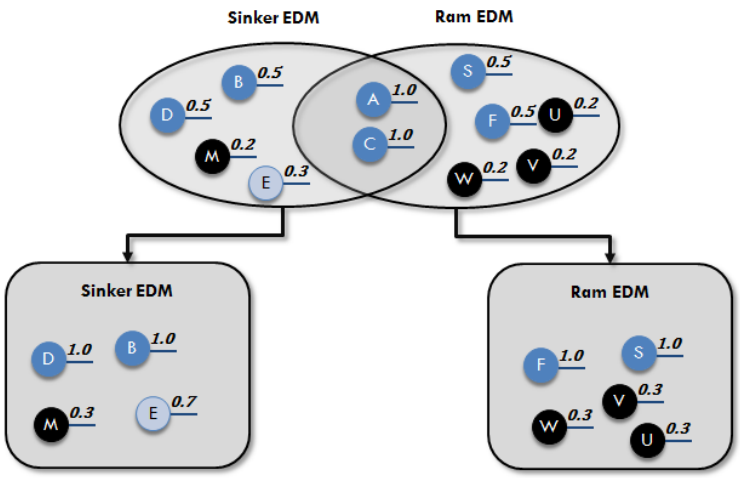

Figure 5-12 Example of feature specification from Sinker EDM and Ram EDM

Figure 5-13 below shows the resulting hierarchy after the *Feature Abstraction* and *Specification*. The sub-class relation between *EDM*, and *Sinker EDM* and *Ram EDM* is derived.

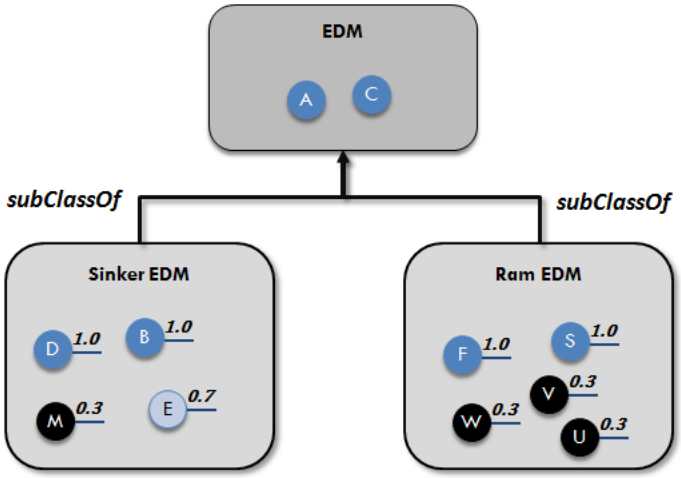

Figure 5-13 Resulting hierarchy after the feature abstraction and specification

Examples in this section illustrate the process of *Features Abstraction* and *Feature Specification* using simple example. However, in practice, hundreds or even thousands of concepts might be involved in these processes. Thus, we cannot expect this desired situation in reality. In order to address this issue, we adopt the statistical approach such that each concept is represented in terms of *d* features and is viewed as a point in a *d*-dimensional space. The objective of the *Feature Specification* is to establish such decision boundaries in the feature space. The decision boundaries can be determined by the probability distributions of the patterns belonging to each concept (Devroye et al, 1996).

5.8 *Evaluation*

The results from the previous sections can be evaluated by using a classifier with a test data set. The decision tree, support vector machine, or other classifier can be used for the evaluation. The decision *tree* (Breman et al, 1984), (Chou, 1991) can be trained by an iterative selection of individual features which are most salient at each node of the tree. The main advantages of the decision tree are its fast speed in computation, and the possibility to interpret the decision rule in terms of individual features (Jain et al, 2000). The disadvantage of the decision tree is a tendency of overtraining. However, that can be avoided by pruning (Mehta et al, 1995).

The support vector machine is primarily binary classifier developed by Vapnik (Vapnik, 1998). The support vector machines are supervised learning models with associated learning algorithms that analyze data and recognize patterns, used for classification. (Jain et al, 2000) summarized the decision function for a two-class problem derived by the support vector classifier.

$$D(x) = \sum_{\forall x_i \in S} \alpha_i \lambda_i K(x_i, x) + \alpha_0 , \qquad (10)$$

where $K(x_i, x)$ is a kernel function of a new pattern x (to be classified) and a training pattern x_i, S is the support vector set (a subset of the training set), and $\lambda_i = \pm 1$ the label of object x_i. The parameters $\alpha_i \geq 0$ are optimized during training by

$$min_{\alpha}(\alpha^{T}\Lambda K\Lambda\alpha + C\sum_{j}\varepsilon_{j}) \qquad (11)$$

Constrained by $\lambda_i D(x_i) \geq 1 - \varepsilon_i, \forall x_i$ in the training set. Λ is a diagonal matrix containing the labels λ_j and the matrix K stores the values of the kernel function $K(x_i, x)$ for all pairs of training patterns. The set of slack variables ε_j allow for concept overlap, controlled by the penalty weight $C > 0$. For $C = \infty$, no overlap is allowed. During optimization, the values all α_i would become 0, except for the support vectors which are the only ones finally needed.

5.9 ***Experiment***

A series of simulation experiments have been conducted to validate our ideas. For these experiments, we collected 28,798 terms and values from 811 manufacturing suppliers' web pages. The dataset has 158 manufacturing service categories. In this section, we present the analysis specifically on the *EDM* related manufacturing concept among 158 manufacturing service categories. In our data set, there are 14 *EDM* related concepts and the total number of features is 91 after the *data preprocessing* process. Figure 5-14 below shows the feature distribution of the 4 *EDM*s (*Ram EDM*, *Sinker EDM*, *Small Hole EDM*, and *Wire EDM*). All 91 features are encoded by a number from 1 to 91 and represented in x-axis. The y-axis represent the existence with the value 0 (not exist) or 1 (exist). As shown in Figure 5-14, different *EDM* concepts have different distribution of the features. The objective of this experiment is to illustrate each process in the proposed approach on

the actual data set based on only the feature existence. The actual values, which each feature has, are not considered in this experiment. That would be the next task of our future work.

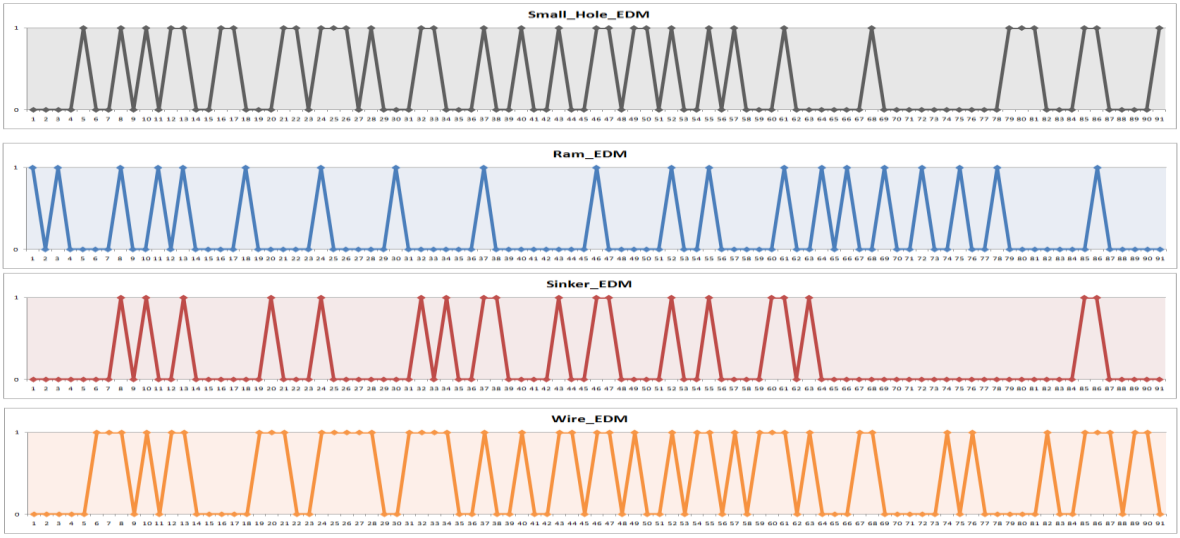

Figure 5-14 Distribution of feature existence

Table 5-1 below shows the features in the top 30th percentile among 91 features, and its frequencies. The feature frequencies are calculated using the formula (1). All *EDM* related concepts have *Standard* as their feature and *Intended Application*, *Industry Focus*, *Production Volume*, *Lead Time*, and *Material* also have high feature frequency. On the other hand, *Maximum Travel Width*, *Maximum Travel Height*, *Maximum Travel Length*, and *Maximum XY Travel* have low feature frequencies.

Table 5-1 Feature frequency under EDM service category

Features	FF	Features	FF	Features	FF

Standard	1.00	Additional Capabilities	0.64	Thickness	0.21
Intended Application	0.93	Equipment Capabilities	0.64	Surface Finish	0.21
Industry Focus	0.93	Material Thickness	0.50	Additional Services	0.21
Production Volume	0.93	Minimum Part Feature	0.36	General Machining Capabilities	0.14
Lead Time	0.86	Width	0.36	Hole Sizes	0.14
Material	0.86	Maximum Part Length	0.36	Taper Cuts	0.14
Process	0.79	Length	0.36	Other Equipment	0.14
Tolerances	0.79	Part Width	0.36	Certification	0.14
File Format	0.71	Advantage	0.29	Quality Control Process	0.14
Cutting Axis	0.71	Wire Diameter	0.21	Software Used	0.14

We have run three feature selection algorithms presented in the section 5.5. Table 5-2 below shows the result of *Information Gain* feature selection algorithm. In order to evaluate the features, we used *Ranker* which ranks attributes by their individual evaluations in conjunction with feature evaluators (*ReliefF*, *GainRatio*, *Entropy* etc). *Ranker* is only capable of generating attribute rankings. Since the total number of features in this experiment is not much, we did not set any threshold to discard lower ranking features. The features which does not exist in the table is 0, thus has 0 as the feature frequency value.

Table 5-2 Result of the information gain feature selection algorithm

Features	Ranker	Features	Ranker
Material Thickness	1	Maximum Part Length	0.94
Length	0.94	Part Width	0.94
Additional Capabilities	0.94	File Format	0.863

Width	0.94	Cutting Axis	0.863
Minimum Part Feature	0.94	Advantage	0.863
Equipment Capabilities	0.94		

Table 5-3 below shows the result of *Chi-square* feature selection algorithm. Same as the *Information Gain* feature selection algorithm, we used *Ranker* to evaluate the features and we did not set any threshold to discard lower ranking features.

Table 5-3 Result of the chi-square feature selection algorithm

Features	Ranker	Features	Ranker
Advantage	14	Length	13.99
Minimum Part Feature	14	Width	13.99
Material Thickness	14	Cutting Axis	13.99
Maximum Part Length	14	Additional Capabilities	13.99
Part Width	14	Equipment Capabilities	13.99
File Format	13.99		

Table 5-4 below shows the result of *CFS* feature selection algorithm. We have used *GreedyStepwise* to evaluate the features. *GreedyStepwise* performs a greedy forward or backward search through the space of attribute subsets and stops when the addition/deletion of any remaining attributes results in a decrease in evaluation. *GreedyStepwise c*an also produce a ranked list of attributes by traversing the space from one side to the other and recording the order that attributes are selected. We have set 0.4 as a threshold. Thus, the features with 0.4 or less are discarded.

Table 5-4 Result of the CFS feature selection algorithm

Features	Greedy Stepwise	Features	Greedy Stepwise
Length	0.793	Additional Capabilities	0.7302
Part Width	0.7846	File Format	0.6579
Advantage	0.7844	Maximum Part Height	0.5806
Cutting Axis	0.7837	Minimum Part Feature	0.5696
Width	0.7718	Cutting Area	0.4811
Equipment Capabilities	0.7691	Material Thickness	0.416
Maximum Part Length	0.7623	Work Piece Thickness	0.4159

How to choose the optimal feature selection algorithm is a challenging task. For the simplicity, we selected features using following formula.

$$Feature\ Set = \left(F_{InfoGain} \cap F_{ChiSquare} \cap F_{CFS}\right) \cup F_{ff>0.70} \qquad (12)$$

$F_{InfoGain}$, $F_{ChiSquare}$, and F_{CFS} mean the resulting features from the *Information Gain*, *Chi-Square*, and *CFS* respectively. $F_{ff>0.70}$ means the features that have the feature frequency greater than 0.70. In this experiment, we determine 0.70 as a threshold of the *feature frequency*. This means the features used by 70% of *EDM* descriptions are considered as common features of all *EDM*s. In practice, the threshold can be calibrated through recursive experiments as well as domain knowledge. Table 5-5 below shows the selected feature set.

Table 5-5 Select feature set from feature frequency and feature selection algorithm

Selected Features

Standard	Advantage	File Format
Intended Application	Minimum Part Feature	Length
Industry Focus	Material Thickness	Width
Production Volume	Maximum Part Length	Cutting Axis
Lead Time	Part Width	Additional Capabilities
Material	Tolerances	Equipment Capabilities
Process		

Based on the result of the *feature frequency* and *feature selection* above, the *pattern abstraction* and *pattern specification* can be done. Figure 5-15 illustrates one part of the resulting *pattern abstraction* and *pattern specification*. The red-fonted term represents the terms which already defined in upper service category.

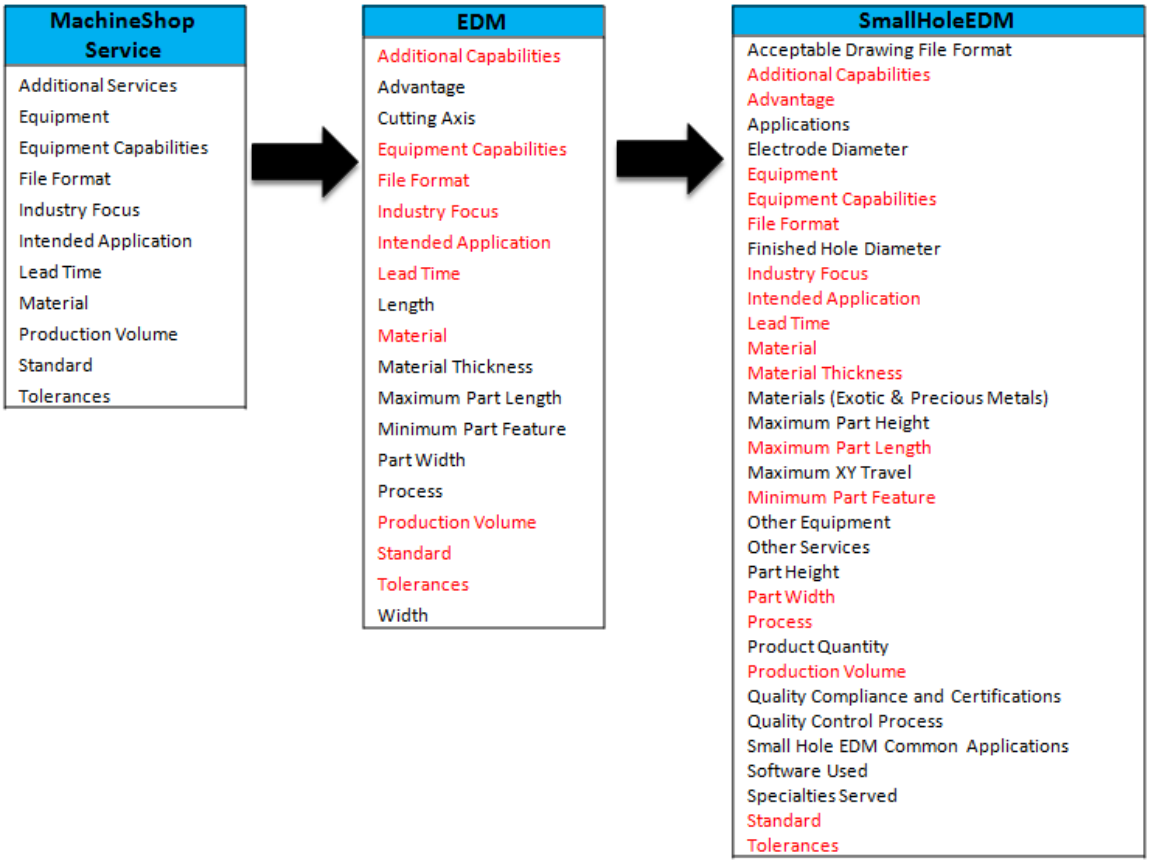

Figure 5-15 The result of the pattern abstraction and pattern specification

Figure 5-16 illustrates conceptual representation of the relationship from the upper concept, *MachineShopService* to the lower concept, *SmallHoleEDM*, after the resulting *pattern abstraction* and *pattern specification*. Some set of common features in different concepts moves up to the upper concept, and some specific features are retained in the lower concept.

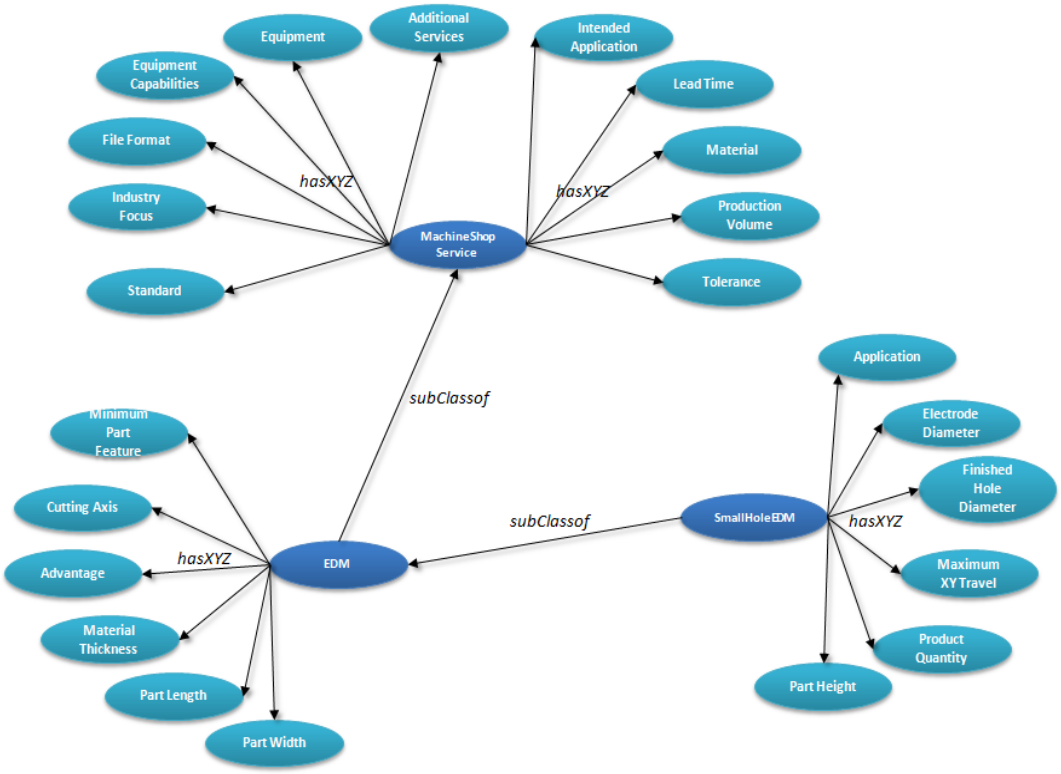

Figure 5-16 Conceptual representation of the relationship from the upper concept, MachineShopService to the lower concept, SmallHoleEDM

Chapter 6. Service Search and Composability Analysis Framework

This section describes the proposed service search and composability analysis framework. The proposed framework is outlined in Figure 6-1 below. The framework requires two essential components: the *Reference Models Repository* and *Service Registry & Repository*. The *Reference Models Repository* contains the resource, state, and function models as well as the instances of the models. The *Service Registry & Repository* contains service descriptions that are registered by service providers. A service description specifies the service's functional and non-functional characteristics using the concepts defined in the *Reference Models Repository.* If a required concept does not exist in the Reference Models Repository, the *Model Development & Evolution Framework* component (such as described in Ameri et al. 2015) is triggered. The *Model Development & Evolution Framework* is out of the scope of this research. However, the ontology development method in Chapter 5 can be a potential solution for the *Model Development & Evolution Framework.*

For effective strategy and procedure to service search and composability analysis the framework decouples the composition problem into two levels: function and service level composition. The benefits of this approach are also supported by the findings in Hassine

et al. (2006) and Baryannis and Plexousakis (2010). They have indicated that such problem decomposition enables reduction in computational complexity of the services composition problem and allows more flexibility by allowing the user to adjust the service-level solution when needed with having to re-computing the function-level solution.

The typical flow of activities in the framework is as follows. The first step is the *Requirements Formalization* in which user can represent his/her requirements using concepts in the reference models. Then, in the *Functional Design* step, the framework looks up the *Reference Models Repository* to assemble a sequence of functions that satisfies the requirement. Next, the framework looks up the *Service Registry & Repository* and retrieves a set of actual services that support each of the required functions. The final step is Compatibility Analysis in which the framework analyzes compatibilities between the retrieved services.

A prototype implementation of the framework has been developed to validate this design. Detail of each step and components is further described in the following subsections.

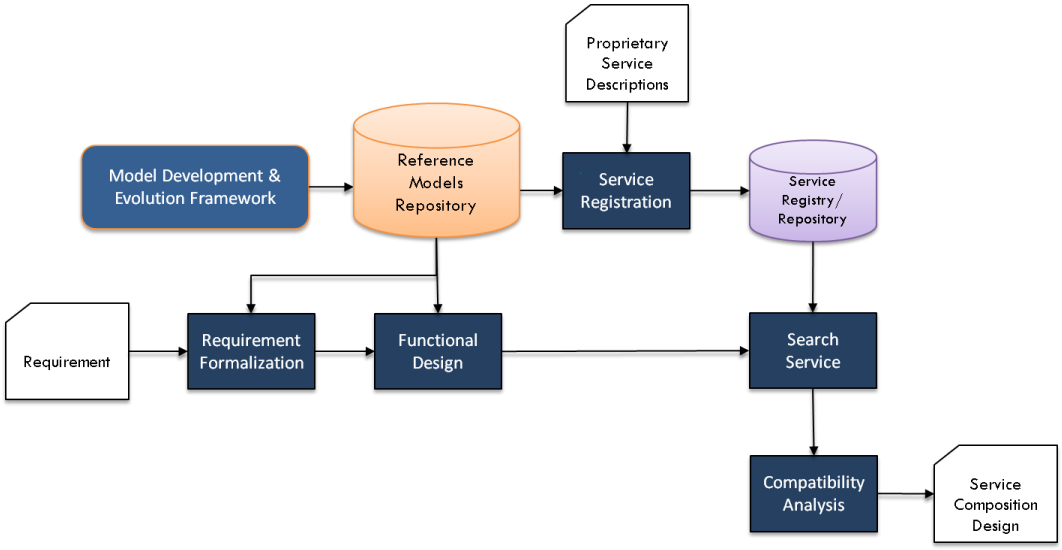

Figure 6-1 Composability Analysis Framework

6.1 *Reference Models*

Reference models for function, resource, state, and property as defined in section 4.3 have been implemented in OWL. The models have also been populated with domain specific concepts sufficient for illustrating the framework. These initial reference models coupled with the Model Development & Evolution Framework component would allow the models to grow and address the ever-growing domain-specific concepts. Figure 6-2 shows a simple illustration of the reference models.

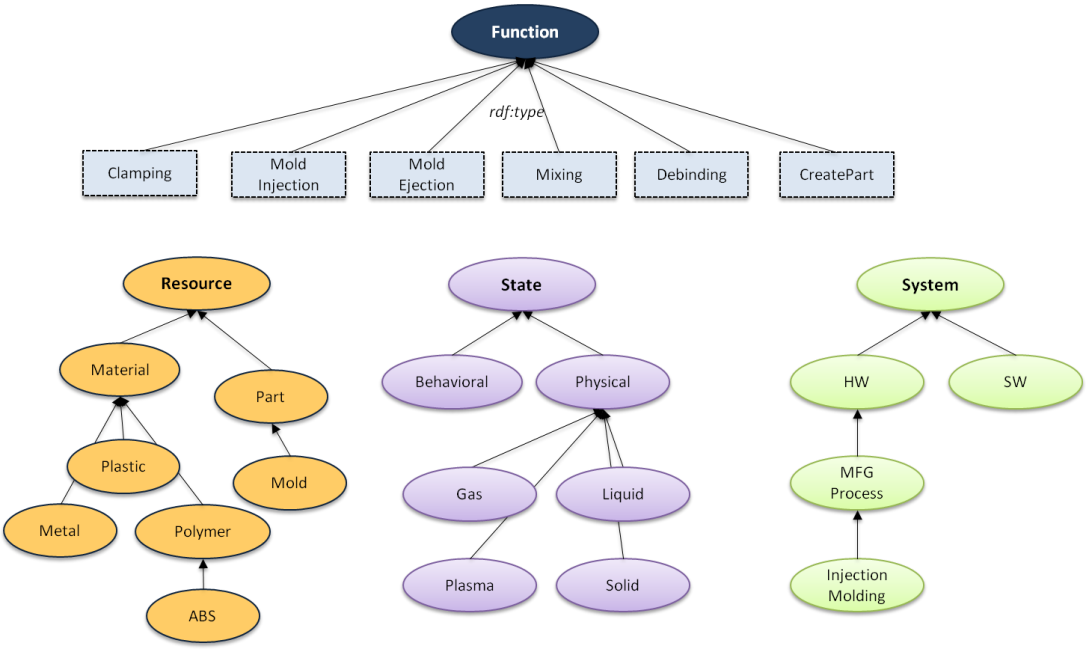

Figure 6-2 Reference Models

Figure 6-3 shows a snippet of the function models. The model follows the function representation method presented in the section 4.3.2.

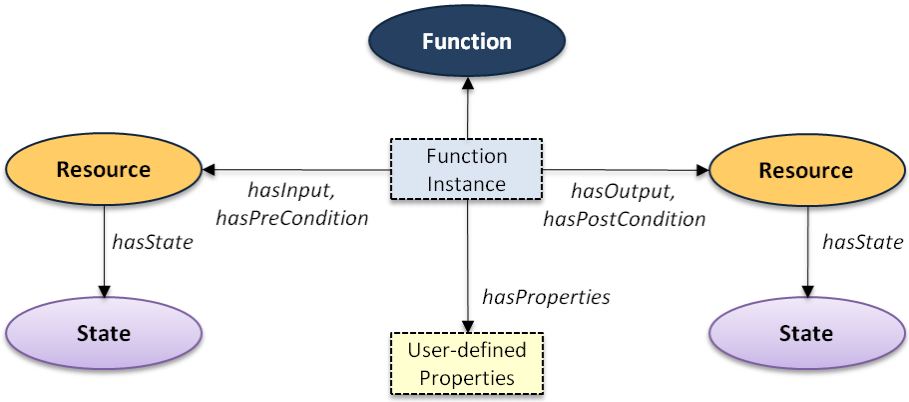

Figure 6-3 Reference Function Model

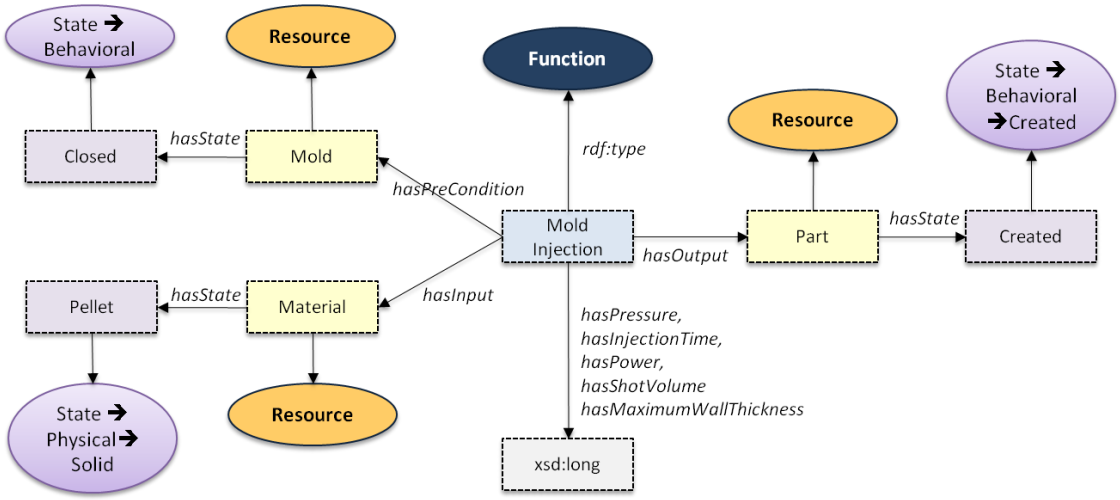

Figure 6-4 Function Instance

6.2 *Requirement Formalization*

This step is to formally encode user's requirements using concepts defined in the reference models. User's requirements can be represented by initial and goal condition, which are, as defined in Chapter 4, (resource, state) pairs. The initial condition is what the user currently has and the goal condition is what the user wants to have. Both the initial and goal condition can be represented by using the concepts in the resource and state model. The framework allows users to interactively configure any resources with different states. Figure 6-5 shows the user interface for encoding the user's requirements in the prototype system.

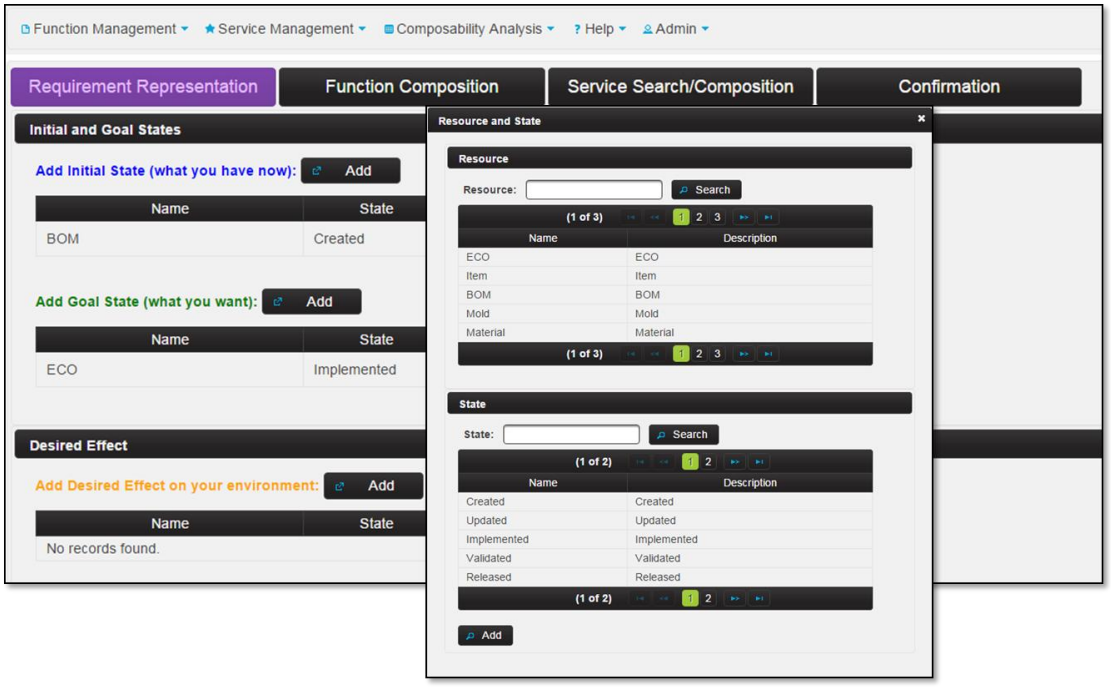

Figure 6-5 User interface for the requirement formalization

The initial and goal states can be specified by selecting *Resource* and their *States* in the resource and in the state reference model, respectively. For instance, consider the following user requirement: User has a new *BOM* (Bill of Materials) that have been *created* due to a design change, and the user wants to implement an *ECO* (Engineering Change Order) electronic documents whose information can be used to implement the necessary change in the manufacturing of the product. The user goal is to get a feedback from the manufacturing department that the necessary change has been successfully implemented which is indicated by the Engineering Change Order electronic document being in the *implemented* state. In this case, the initial condition is a '*BOM* in *created*

state', and the goal condition is '*ECO* in *implemented* state'. The user can encode these states by using the provided user interface that supports navigating and searching the reference models.

User can specify his/her own constraints when finding a set of functions to achieve the user's requirement. For instance, the user can specify a penalty for the total number of functions for the given requirement. The framework will add the specified penalty to the total cost in proportion to the number of functions in the result. For example, if the number of resulting functions is 4 and user specifies 1 as a penalty, the framework will add 3 to the total cost.

6.3 *Functional Design*

The framework looks up the function instances in the reference models to find a set of functions to satisfy the requirement specified in the *Requirements Formalization* step with consideration of the constraints. The result of the *Functional Design* is a directed graph that is constructed from function instances in the reference function model. The result is called as a *function level composition network* and represents a solution graph that has minimum cost. The method described in the next chapter is used to generate the *Functional Design*. A simple example of the result is shown in Figure 6-6.

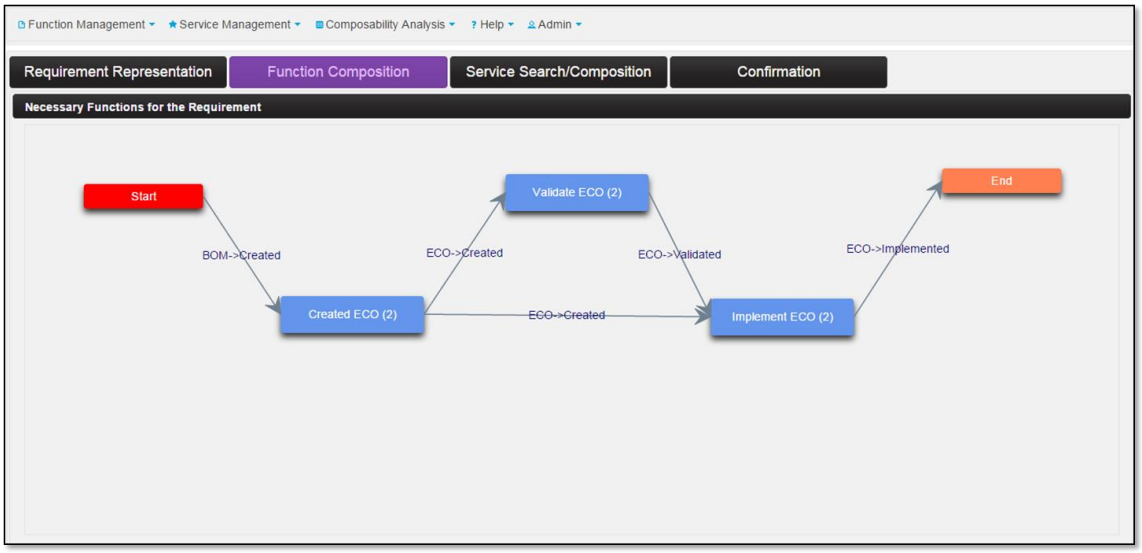

Figure 6-6 Example of Functional Design

This result consists of three functions including *Create ECO*, *Validate ECO*, and *Implement ECO* and their input /output and pre/post-condition relationships. After the user has selected a function-level plan, he/she moves on to the service search step, which is described in the next section.

6.4 *Service Search*

In the service search step, the framework searches for available services. We have implemented a virtual service registry & repository for our work. The result of the *Service Search* is called as a *service level composition network*. The *service level composition network* is a directed graph that is constructed from the services in the service repository. The *service level composition network* shows all possible

dependencies among the services that support the functions identified in the *Functional Design* step. Note that in the *service level composition network*, the dependency between services is determined by the characteristics of the functions supported by the services, not by the input/output and pre/post-condition of the services. Figure 6-7 illustrates an example of the *service level composition network*. Each service has a set of input and output. The function that is provided by the service is represented in the parenthesis under the name of the service. For example, the *Service A* supporting *Create ECO* function has *OAGIS 10.0 Standalone BOM* as an input and *OAGIS 10.0 Standalone ECO* as an output. The *Service B* supporting *Validate ECO* function has *SAP ECO v2.1* as an input and *SAP ECO Validation Res v2.1* as an output. Although the output of the *Service A* does not exactly match with the input of the *Service B*, the two services have dependency relationship, because the *Service B*'s function *Validate ECO* is dependent on the function, *Create ECO* that is provided by the *Service A*.

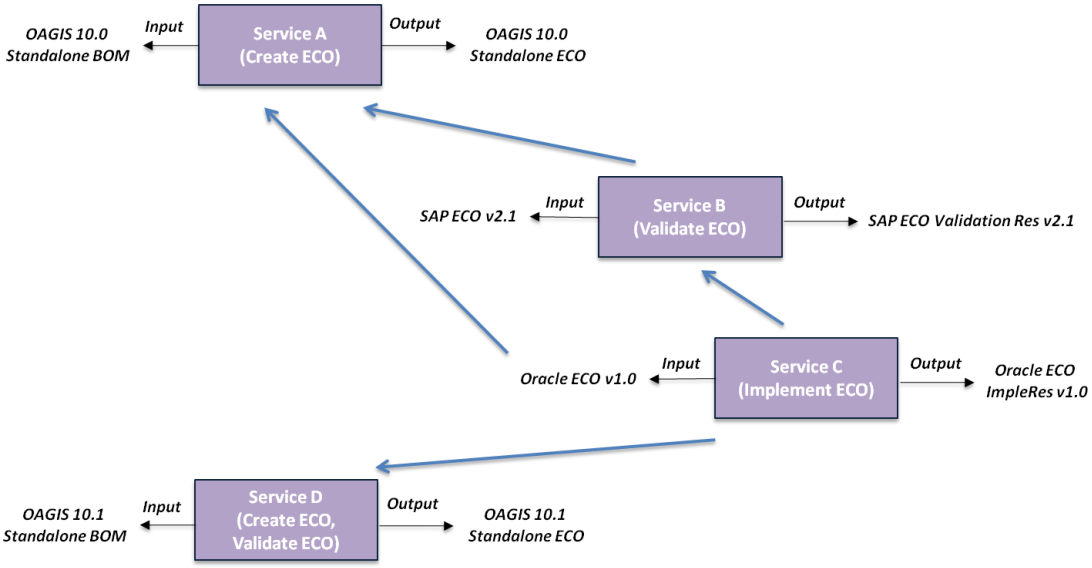

Figure 6-7. An Example of the service level composition network

Figure 6-8 below shows the result of service search and composability analysis with the given functional design in out prototype system. The upper part of Figure 6-8 shows the functional design and the middle part of Figure 6-8 shows identified services and their dependency that has minimum composition cost for the given functions. *Tibco ECO* service provides two functions including *Create ECO* and *Validate ECO*, and *ND Soft* service provides *Implement ECO* function. The bottom part of Figure 6-8 shows list of properties each service has and the result of the compatibility analysis.

Figure 6-8 Service search and composability analysis result

6.5 *Compatibility Analysis*

The framework assesses the compatibility between the services based on various non-functional characteristics of the services. The framework enables users to dynamically configure various constraints. For instance, assume that user A does not have an ability to handle I/O format mismatches between the services. That may lead the user A to increase

the I/O format mismatch penalty. On the other hand, user B has an expertise to handle I/O format mismatches, but does not have an ability to handle communication protocol mismatches. In this case, the user B may put higher penalty on the communication protocol mismatch. Figure 6-9 shows the user interface for constraint configuration for the service search and compatibility analysis.

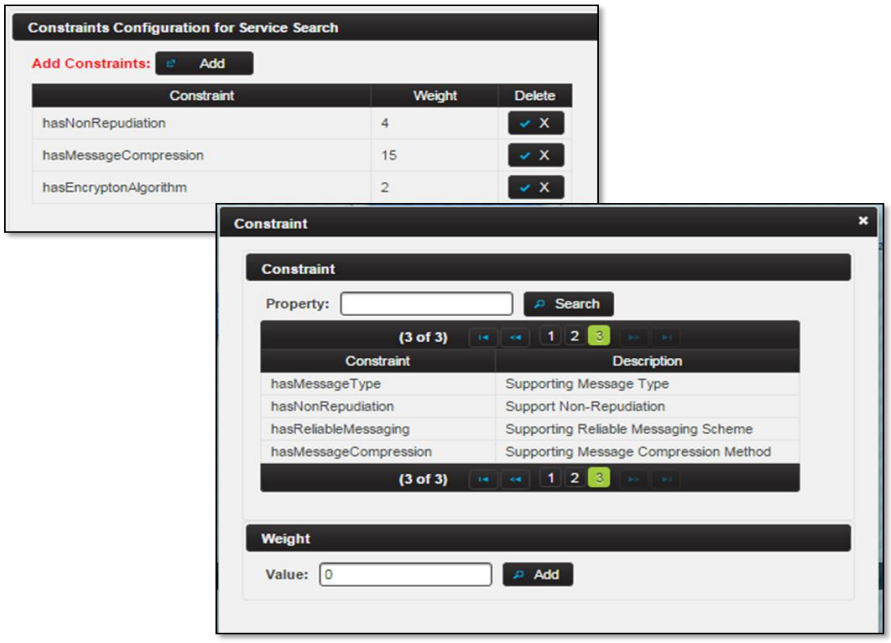

Figure 6-9 Constraint con figuration for service search and compatibility analysis

All these preference configurations will impact the set of services the framework selects for the best solution. Thus, the best solution totally depends on the users' preferences or priorities.

Chapter 7. Service Search and Composability Analysis Methods

In this chapter, we provide a graph-based method that is used for the service search and composability analysis framework described in the previous section. Specifically, the graph-based method is applicable to both of the *functional design* and *compatibility analysis* steps.

7.1 Problem Modeling

As presented in Chapter 6, the framework decouples the problem into two different levels of composition problems: function and service level composition. From the function and service representation in Chapter 4, both the function and service can be viewed as a vertex and their relationships through input/output or pre/post-condition can form edges between the vertices. The vertices and edges can form a graph, and we call the graph as a *Composition Network* (CN). In addition, the compatibility between the vertices can be quantified based on the constraints and penalty configured by a user as described in section 6.5. The quantified compatibility between the vertices can be used as a weight on the edge that connects the vertices.

The function and service search and composability analysis problem is to find a set of functions or services that can satisfy a given user's requirement while minimizing the expected cost that is required to compose the services. The user's requirement also can be modeled as vertices where the initial condition can be modeled as a source vertex and the goal condition can be modeled as a target vertex. Thus, the problem can be modeled as finding a set of vertices in the *CN* that are required to transit the initial condition to the goal condition while minimizing the sum of weights on the edges that connect these vertices.

7.1.1 Composition Network

A composition network (CN) is a directed graph that can be dynamically constructed from the function instances in the reference function model or service descriptions in the service repository. The CN represents all possible input/output and pre/post-condition relationships among the functions instances.

The CN may be viewed as a kind of multigraph [Balakrishnan and Ranganathan 2012] that has a directed edge. Like the multigraph, in the CN, edges from one vertex can have the same end vertices, that is, two vertices can be connected by more than one edge. Typically, multigraph has two distinct notions of edges. First one is edges without own identity. In this case, edge's identity is defined by the two vertices it connects. Thus, the same edge can occur several times between these two vertices. Second notion is edges with own identity. In this case, edges are primitive entities just like vertices.

The CN's notion of edges is a combination of the two notions in multigraph. That is the identity of the edge in the CN is determined by the two vertices it connects as well as the object that is transmitted by the edge. The object is called as *edge variable*. A simple example of the CN that is generated from the function instances is shown in Figure 7-1 below.

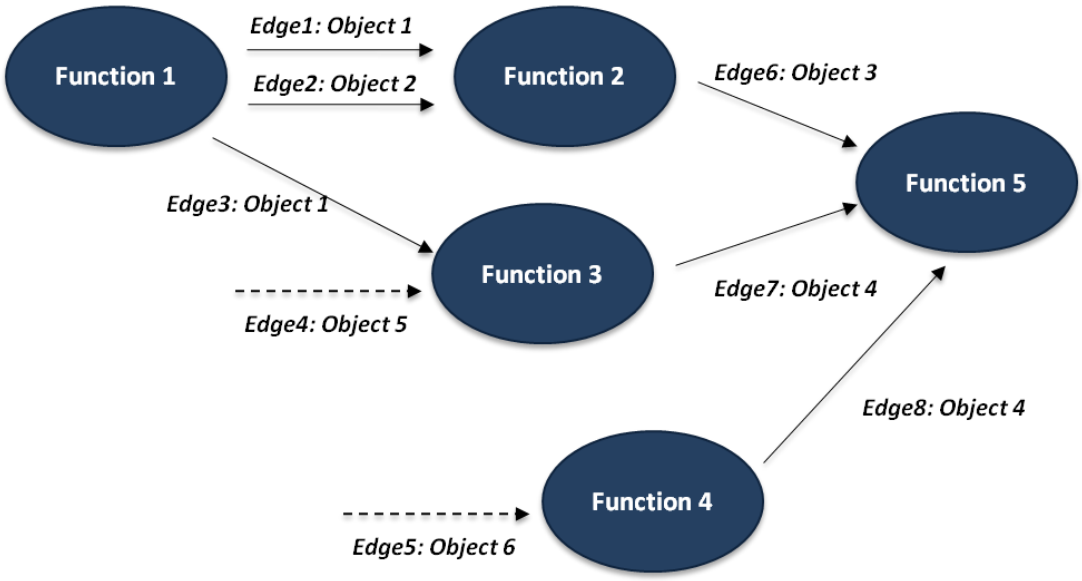

Figure 7-1 Example of Composition Network generated from the function instances

This CN consists of five vertices including *Function 1*, *Function 2*, *Function 3*, *Function 4*, and *Function 5* with their incoming/outgoing edges that are represented by the edge name and object. For instance, the *Edge 1*'s identity is defined by the two vertices *Function 1* and *Function 2* it connects together with the *object 1* that is transmitted by the edge.

Unlike the typical multigraph, the CN has an edge that does not have any source or target vertices. For example, the *Edge 4* in Figure 7-1 has target vertex *Function 3*, but does not have any source vertex. This may happen when a vertex has an input, but no other vertices have an output that is matched with the input.

In the CN, a vertex can be invoked if and only if all the inputs of the vertex are provided. For example, in the Figure 7-1, to invoke *Function 3*, *object 1* and *object 5* must be provided. We can say that the *Function 3* has a dependency to the *object 1* and *object 5*.

In addition to that, there exists logical relationship in CN. For instance, in order to invoke *Function 5*, *object 3* and *object 4* must be provided. Thus, the two objects are logically ANDed. On the other hand, the *object 4* can be provided by either *Function 3* or *Function 4*. That is, the *object 4* from *Function 3* and *Function 4* are logically ORed.

In order to develop service search and composability analysis method in the CN, it is necessary to formally represent the problem. In our research, we propose And/Or graph to accommodate the dependency as well as the logical relationship within the CN. The formal definition is presented in the Section 7.1.4.

7.1.2 AND/OR Graph

The use of AND/OR graphs for representing problems originated in the 1960's within the domain of Artificial Intelligence. Since then, it has spread to other fields, such as Operations Research, Automation and Robotics, where AND/OR graphs are nowadays

being used to represent cutting problems [Arenales and Morabito 1995], interference tests [Jiménez and Torras 1996], failure dependencies [Barnett and Verma 1994], robotic task plans [Cao and Sanderson 1998], and assembly/disassembly sequences [DeMello and Sanderson 1991].

An AND/OR graph can be seen as a generalization of a directed graph. It contains a number of vertices and generalized edges (or connectors) that connect the vertices. Each connector in an AND/OR graph connects a set of vertices to a single vertex. A connector is said to be an AND connector, if there is a logical AND relationship. A connector is an OR connector, if there is a logical OR relationship.

7.1.3 AND/OR graph representation based on CN

There might be a redundancy issue when we represent the CN as AND/OR graph. Figure 7-1 below shows how the *Function 2*, *3*, *4*, and *5* in Figure 7-1 can be modeled as AND/OR graph. As shown in the Figure 7-1, *Function 2* shows up in both of the conjunctions.

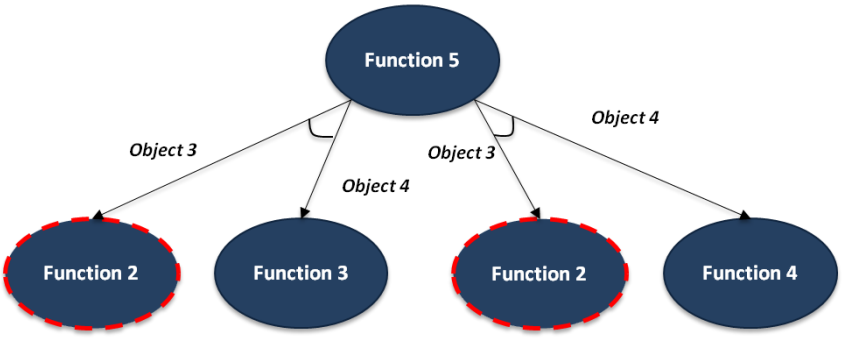

Figure 7-2 Example of redundant vertex representation in AND/OR graph

In order to address the redundancy issue, we propose the following representation method.

- A function or service vertex as an AND vertex.
- An object transmitted through an edge as an OR vertex.
- New AND vertex to represent user's goal condition in the service request defined in the Chapter 2. This vertex will be a start vertex in AND/OR graph.
- New OR vertex to represent user's initial condition in the service request defined in the Chapter 2. This vertex will be a terminating vertex in AND/OR graph.

Figure 7-3 below shows an exemplary composition network. User's initial and goal condition are modeled as gray ellipse and a possible set of functions to transit the initial condition to the goal condition are modeled as a blue rectangle.

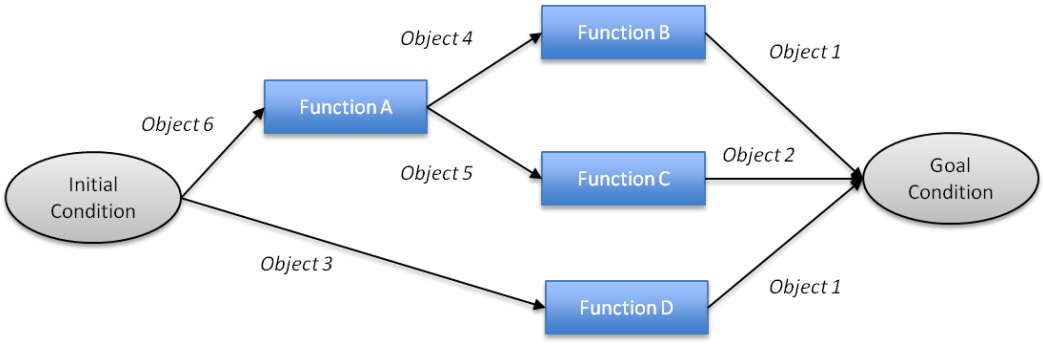

Figure 7-3 Exemplary composition network (CN)

The composition network in Figure 7-3 can be modeled as AND/OR graph as shown in Figure 7-4 according to the representation method above. The goal condition and initial

condition transformed into *Start* and *Terminal* vertex respectively. All the function vertices are represented as an AND vertex and the objects that are transmitted though edges are represented as an OR vertex.

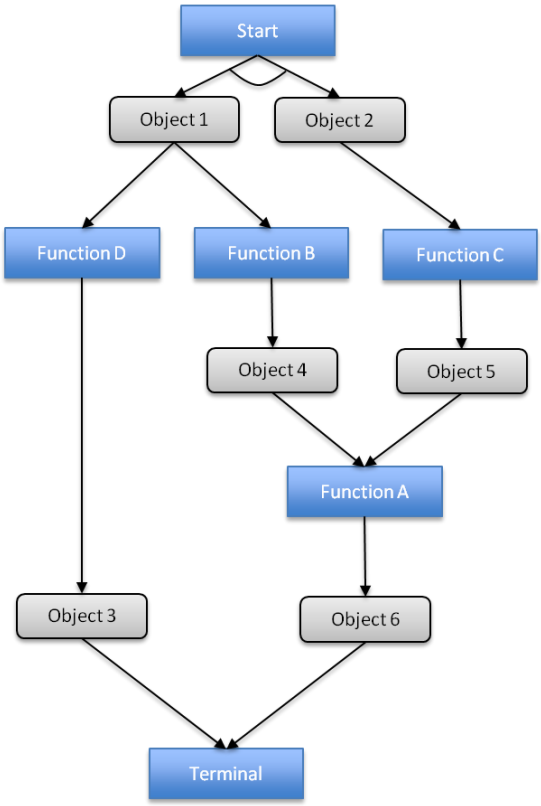

Figure 7-4 AND/OR graph to represent the CN in Figure 7-3

7.1.4 Problem Definition

The followings are formal definitions of the composition network, AND/OR graph, and problem.

Definition 7.1 (Composition Network). A composition network, *CN = (V, E, w)* is a weighted, directed graph, where *V* is a set of vertices, *E* is a set of edges, and w is a weight function *w : E -> ℝ*..

Each edge *e* consists of three variables including source vertex v_s, target vertex v_t, and object *o*:

$$e = \{ v_s, v_t, o \} .$$

The object *o* represents a resource and its state that is transmitted through the edge.

Definition 7.2 (User's Requirement). A user's requirement *Req* = {R_I, R_G} consists of a set of initial conditions R_I and a set of goal conditions R_G.

R_I consists of pairs of resource *r* and its state *s*:

R_I = { (r_1, s_1), ... (r_k, s_k) | *r* ∈ Resource defined in the resource model and *s* ∈ State defined in the state model } .

R_G also consists of pairs of resource *r* and its state *s*:

R_G = { (r_1, s_1), ... (r_j, s_j) | *r* ∈ Resource defined in the resource model and *s* ∈ State defined in the state model } .

R_I and R_G can be represented by start and terminal vertex respectively in the composition network. v_I is a vertex created from the set of initial conditions and v_G is a vertex created from the set of goal conditions. Note that v_I has R_I as an outputs and does not has any inputs while v_G has R_G as an inputs and does not have any outputs.

Definition 7.3 (AND/OR graph). An AND/OR graph, $AO = (V_{and}, V_{or}, E', w)$ is a weighted, directed graph, where V_{and} is a set of AND vertices, V_{or} is a set of OR vertices, E' is a set of edges, and w is a weight function $w : E' \rightarrow \mathbb{R}$.

V_{and} has (at least one) edges directed to OR vertices. The OR vertices are called the successors of the V_{and} and the edges have logical AND relationship such that all the OR vertices must be provided to achieve the V_{and}.

V_{or} has (at least one) edges directed to AND vertices. The AND vertices are called the successors of the V_{or} and the edges have logical OR relationship such that any one of the AND vertices enables to achieve the V_{or}.

Each edge $e' \in E'$ consists of two variables including source vertex v_s, and target vertex v_t:

$$e' = \{ v_s, v_t \}.$$

Definition 7.4 (Composition Network as an AND/OR graph). A Composition Network can be converted into an AND/OR graph by the following:

$v \in V$ in the composition network is converted into $v_{and} \in V_{and}$ in AND/OR graph.

$v_I \in V$ in the composition network is converted into start vertex in AND/OR graph.

$v_G \in V$ in the composition network is converted into terminal vertex in AND/OR graph.

e.o ($e \in E$) in the composition network is converted into $v_r \in V_{or}$ in AND/OR graph. Note that there must be a single OR vertex for each resource, even though there might exist multiple edges that have same object.

$e \in E$ in the composition network is converted into two edges $e_1 \in E'$ and $e_2 \in E'$ in AND/OR graph ($v_r \in V_{or}$):

$$e_1 = (e.v_t, v_r).$$
$$e_2 = (v_r, e.v_s).$$

Definition 7.5 (Solution Graph). Given an AND/OR graph *AO*, let *s* be the start vertex and *t* be the terminal vertex. The solution graph *sg* is a finite sub-graph of *AO* that satisfies the followings: *s* is a root of *sg*; for $\forall\, v \in V_{and}$, all of *v*'s immediate successors are in *sg*; for $\forall\, v \in V_{or}$, only one of v's immediate successors is in *sg*; and every maximal directed path start from *s* ends in *t*.

Definition 7.6 (Minimum Cost Solution Graph). Given an AND/OR graph *AO*, let *s* be a start vertex and *t* be a terminal vertex. The minimum cost solution graph is a solution graph with the minimum of the sum of the weights on the constituent edges.

7.2 Search Method

7.2.1 Overview

The AND/OR graph representation encompasses all possible ways to achieve the user's requirement. Since each possible ways corresponds to the solution graphs in the AND/OR graph, the selection of the best way can be viewed as a search problem.

Typically, such search problems require a criterion to compare which one is the best. In service search and composition problem, one possible method is to assign weights to the edges proportional to the difficulty of service composition. The difficulty of the service composition is quite subjective, because one user who is an expert in handling message type conflict may easily address the message type mismatch between the services, while the other one who has a specialty in security could handle the encryption algorithm mismatches. Thus, when quantifying the difficulty as a cost, we have to consider the various characteristics of the services as well as user's preferences.

When searching the AND/OR graph, precise cost estimation is also very important. If we can exactly estimate the cost of the minimal cost path from any vertices to the goal vertex,

then we only need to expend the vertices on the optimal solution path. Thus, in that case, no extra work will be performed for the solution search.

In our work, we propose a method for admissible cost estimation that never overestimates the cost of reaching the goal. The cost estimation method starts from finding all possible sub-graphs that satisfy user's requirement by forward searching the composition network from the initial to the goal condition vertex. Thus, as a byproduct of the admissible cost estimation, we can reduce the search space by screening out unnecessary vertices from the composition network.

7.2.2 Composition Network Pruning and Cost Estimation

Before the composition network pruning and cost estimation, it is necessary to generate a composition network first. For the functional design in section 6.3, the composition network is generated from the function instances in the reference function model, while for the service search in section 6.4, the composition network is generated from the service repository. Figure 7-5 below shows an example of the generated composition network. For the example presented in this chapter, let's assume that the composition network is generated from the function instances. Each rectangle in Figure 7-5 represents a function vertex and the label on each edge represents the object transmitted through the edge. *Object 1* and *2* are the initial conditions and *Object 6* and *7* are the goal conditions.

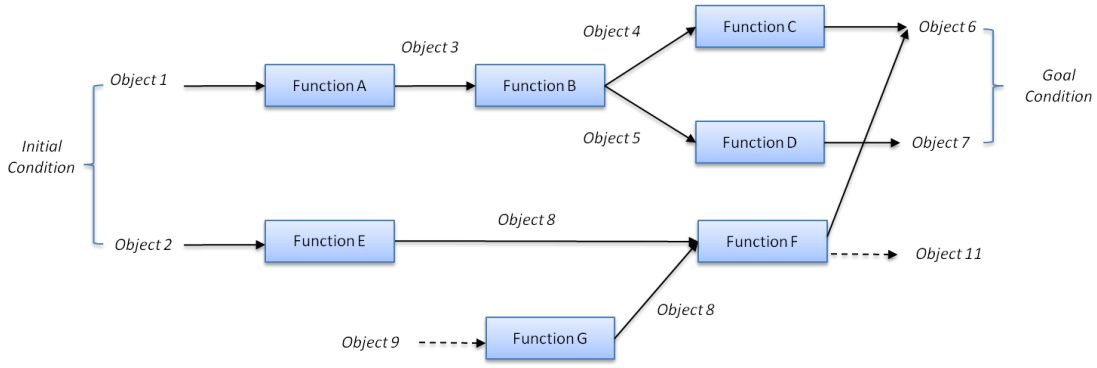

Figure 7-5 Composition Network

After the composition network generation, new vertices are created to encode the initial and goal conditions as vertices. Figure 7-6 below shows the composition network with newly created vertices.

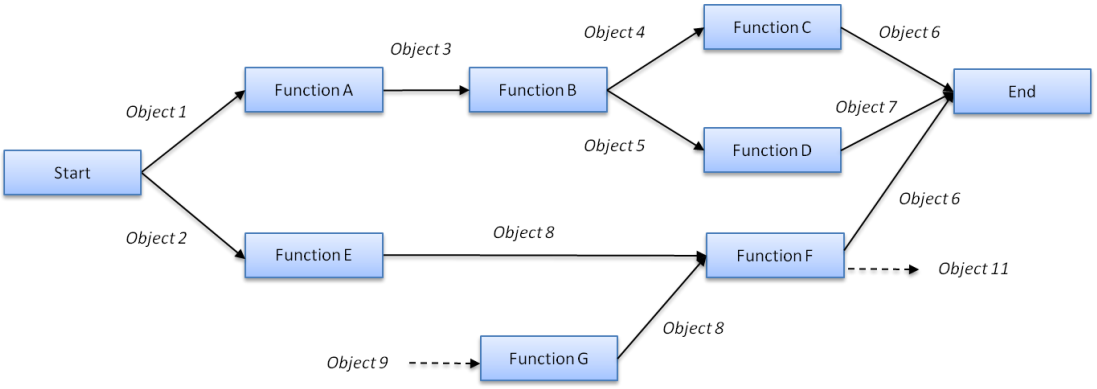

Figure 7-6 Composition Network with source and target vertices

And then, the vertices in the graph are topologically sorted to impose a linear ordering on the vertices. If the composition network has a path from the source vertex to the target vertex, then the source vertex must precedes in the topological sort. After the topological

sorting, we can pass over just once over the vertices in the topologically sorted order. After the topological sorting, all vertices are initialized. And then, as we process each vertex, each edge that leaves the vertex is relaxed. After the first relaxation, unnecessary vertices and edges will be screened out and we will get the cost estimation. However, there is a possibility that the estimated cost is overestimated. We can address the overestimation issue through relaxation again on the pruned composition network. Some of the procedures of the algorithm look similar with the *DAG-SHORTEST-PATHS* algorithm, but the specific methods and data structures are extended for the composition network. The extended methods are represented with asterisk in the pseudo code below. In next sections, we provide details of the extended procedures.

7.2.2.1 Notations

We use the following notations as well as the definitions in Section 7.1.4 to describe all the pseudo codes of the procedures.

- Vertex *v* has a set of inputs *I*, outputs *O*, incoming edges E_i, outgoing edges E_o, and cost *c* from a start vertex:

  $$v = (I, O, E_i, E_o, c) .$$

- Note that the inputs and outputs are explicitly defined in each vertex, while the incoming and outgoing edges are implicitly defined by the dependency between a given vertex and its adjacent vertices.

- *I* and *O* consists of pairs of resource *r* and its cost c_r:

$$\{ (r_1, c_{r_1}), \dots, (r_k, c_{r_k}) \mid r \in Resource\ in\ the\ resource\ model \} .$$

- E_i consists of source vertex v_s, resource *r*, and cost c_r:

$$E_i = \{ (v_{s_1}, r_1, c_{r_1}), \dots, (v_{s_j} r_k, c_{r_l}) \mid r \in Resource\ in\ the\ resource\ model \} .$$

- Some of the incoming edges may have the same resource with different source vertex. Thus, the cost c_r of the input is the minimum cost of the incoming edges that have *r* as a resource:

$$c_r = Min\ (\text{the costs of the incoming edges that has } r \text{ as a resource}) .$$

- The cost of vertex *v*, *v.c* is the sum of the costs of each input:

$$v.c = \sum_{i=1}^{k} c_{r_i} .$$

- E_o consists of target vertex v_t, resource *r*, and cost c_r:

$$E_o = \{ (v_{t_1}, r_1, c_{r_1}), \dots, (v_{t_m} r_k, c_{r_l}) \mid r \in Resource\ in\ the\ resource\ model \} .$$

- Each edge *e* consists of four variables including source vertex v_s, target vertex v_t, resource *r*, and cost c. The resource *r* represents an object that is transmitted through the edge:

$$e = (v_s, v_t, r, c) .$$

The pseudo code below shows the overall procedure of the composition network pruning and cost estimation method using the notation.

```
COMPOSITION-NETWORK-PRUNING-AND-COST-ESTIMATION (R_I, R_G)
1    CN = GENERATE-COMPOSIION-NETWORK(R_I, R_G) // CN in the definition 7.1
2    TOPOLOGICAL-SORTING* (CN)
3    VERTEX-INITIALIZATION* (CN)
4    for each vertex u in CN, taken in topologically sorted order
5        for each outgoing edges e_o ∈ u.E_o
6            RELAXATION* (e_o)
7    for each vertex u in CN, taken in topologically sorted order
8        if u.c = ∞, then remove u and its all edges from CN
9    for each vertex u in CN, taken in topologically sorted order
10       for each outgoing edges e_o ∈ u.E_o
11           COST-ADJUSTMENT* (e_o)
```

7.2.2.2 Topological Sorting

A topological sorting is to order all the vertices linearly. Suppose that the composition network has an edge (v1, v2). Then, v1 appears before v2 in the order after the topological sorting. One issue in the topological sorting is that if the graph contains a cycle, then linear ordering of the vertices is not possible. Therefore, we have to check whether the composition network contains a cycle or not. The cycle detection can be done by the depth first search. A directed graph is acyclic if and only if a depth-first search of the graph yields no back edges. If the function graph contains a cycle, then we have to make the composition network acyclic through the *STRONGLY-CONNECTED-*

VERTEX method. The *STRONGLY-CONNECTED-VERTEX* method transforms the vertices that form a cycle into one single vertex. Details of the method will be described in the section 7.2.2.5.

```
TOPOLOGICAL-SORTING* (CN)
1       while (DEPTH-FIRST-SEARCH(CN)) {
2               If cycle detected // 'back' edge found
3                       CN' = CREATE-STRONGLY-CONNECTED-VERTEX (CN)
4                       return TOPOLOGICAL-SORTING* (CN')
5               Else
6                       as each vertex is finished, insert it onto the front of a linked list
7                       return the linked list of vertices }
```

Figure 7-7 below shows the result of the topological sorting on the composition network presented in the Figure 7-6.

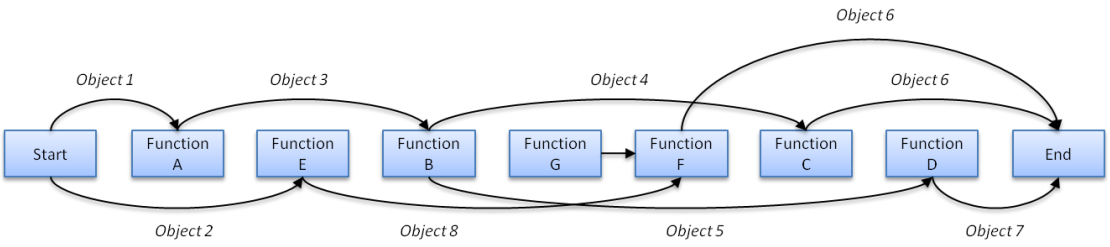

Figure 7-7 Topological Sorting

7.2.2.3 Vertex Initialization

We initialize the cost from source to each vertex, the local cost in the input set, and the incoming/outgoing edges of the vertex. All these initialized variables are defined in

section 7.1.4. The following pseudo code shows the vertex initialization procedure. The source vertex will have 0 as a cost. Figure 7-8 shows the result of the vertex initialization. The white boxes in the bottom represent the cost of each corresponding vertex above.

```
VERTEX-INITIALIZATION* (CN, s) // s is a start vertex
1       for each vertex v ∈ CN.V
2               for each vertex v' ∈ CN.Adj[v]
3                       set E_o in v and set E_i in v'
4               v.c = ∞
5               for each pairs of resource r and its distance c_r in the v.i
6                       c_r = ∞
7       s.c = 0
```

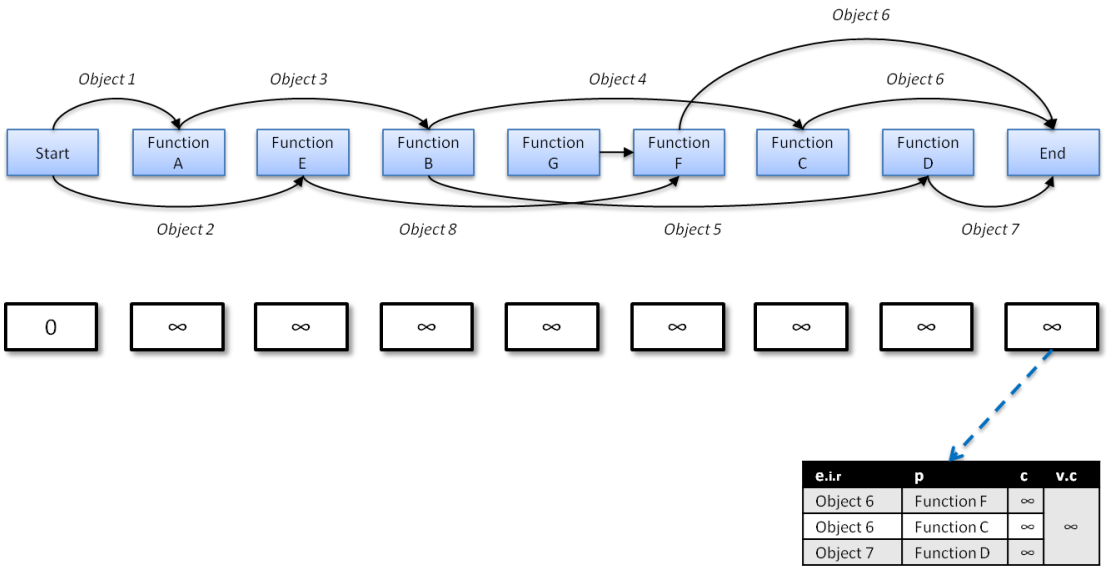

Figure 7-8 Vertex Initialization

7.2.2.4 Relaxation

The algorithm for composition network pruning and cost estimation uses the relaxation technique. Each vertex maintains an upper bound of the cost from source vertex. The upper bound of the cost is represented as c_t . The c_t of each vertex is initialized as ∞ in the vertex initialization step. The relaxation on an edge (u, v) checks whether the cost to v can be improved by going through u. If the cost can be improved, then c_t of v is updated. The details of the checking and updating procedures are described in the following pseudo code.

```
RELAXATION* (e_o)
1       v_s = e_o.v_s, v_t = eo.v_t
2       c_new = v_s.c + e_o.c
3       Get a local cost c_r of the resource in v_t.i using e_o.r as a key
4       if c_new < c_r
5       then c_r = c_new
6            update v.c with new c_r // v.c = Σ_{i=1}^{k} c_{r_i}
```

Figure 7-9 below shows an example of the relaxation process. There are two example vertices, *Vertex X* and *Y*. The upper boxes represent the vertex status before relaxation and the lower boxes represent the vertex status after relaxation. The table in the box shows incoming edge's resource ($e_i.r$), source vertex (p), local cost (c), and global cost (c_t) from start to the vertex. The *Vertex X* has two incoming edge resources *a* and *b*. Before relaxation, the local costs of *a* and *b* are 2 and ∞ respectively. Thus, at that time the

global cost c_t of the *Vertex X* is ∞ as a sum of the two values. Assume that after relaxation, the local cost of *b* is updated as 4 that is smaller than ∞, then that results in the update of the global distance as it improves the cost. The *Vertex Y* in the Figure 7-9 shows another example. The *Vertex Y* has three incoming edges and two of them have the same resource, *a*. Before relaxation, for the resource *a*, the costs through *Vertex A* and *Vertex C* are 5 and ∞ respectively. In this case, the relaxation algorithm takes the minimum cost edge for the input resource. Thus, the global cost to *Vertex Y* is 8 as a sum of 5 and 3. After relaxation, the local cost through *Vertex C* is improved to 2. Then again, the relaxation algorithm takes the minimum cost edge for the input resource, thus in this case, the edge through *Vertex C* is taken and that results in the update of the global cost as well.

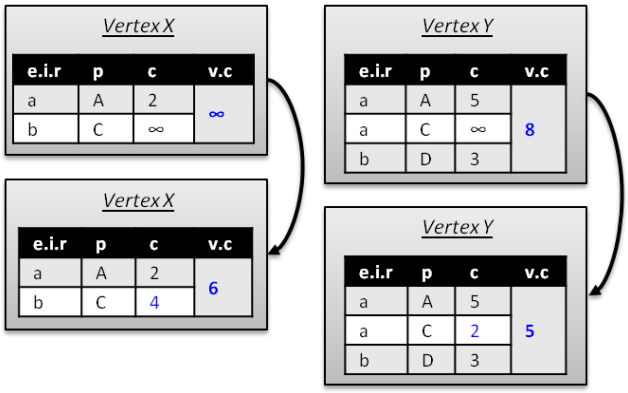

Figure 7-9 Relaxation Example

Figure 7-10 below illustrates the result of the relaxation. The number in the white boxes represents the cost from source to each vertex. As shown in the Figure 7-10, in this

example, there is one vertex that has ∞ as a cost. Thus, the vertex as well as the edges of the vertex will be eliminated from the result as shown in Figure 7-11.

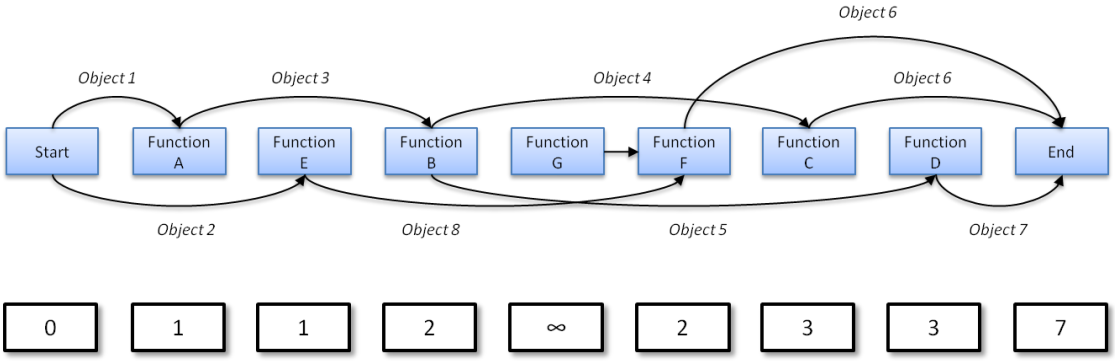

Figure 7-10 Result of the relaxation

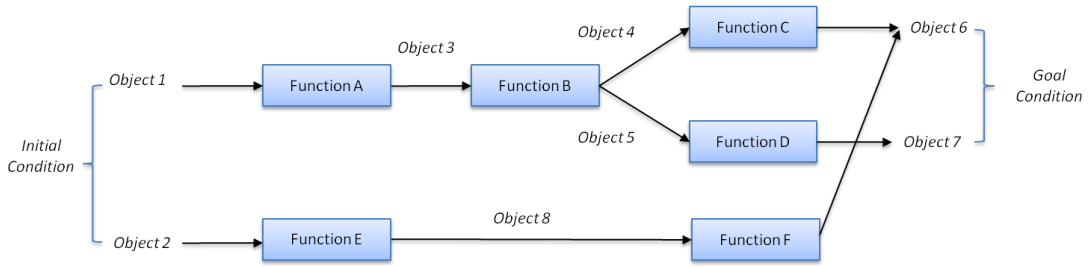

Figure 7-11 Result of composition network pruning

7.2.2.5 Cost-adjustment

The resulting cost of each vertex in previous section may be overestimated, specifically when there is a branch. For example, let's take a look at the example shown in Figure 7-12. For the simplicity, let's assume that all edge weights are 1. There are three paths from the start vertex to the end vertex. For the *Object 7*, there exists only one path while

for the *Object 6*, there are two different paths. The minimum cost path for the *Object 6* is the *Path 3* that has a cost 3. Since there is only one path for *Object 7*, if we just aggregate the minimum cost paths, then the *Path 2* and *Path 3* will be chosen and the total cost would be 7. However, we can reduce the total cost to 6 by choosing the *Path 1* instead of *Path 3* even though the cost of the *Path 1* is greater than the *Path 3*. Thus, the cost of the *End* vertex by the relaxation procedure is overestimated. In order to avoid the overestimation in the example, the cost up to *Function B* should be shared by the *Function C* and *D*. That is the cost of the shared path should not be added in both of the branching paths.

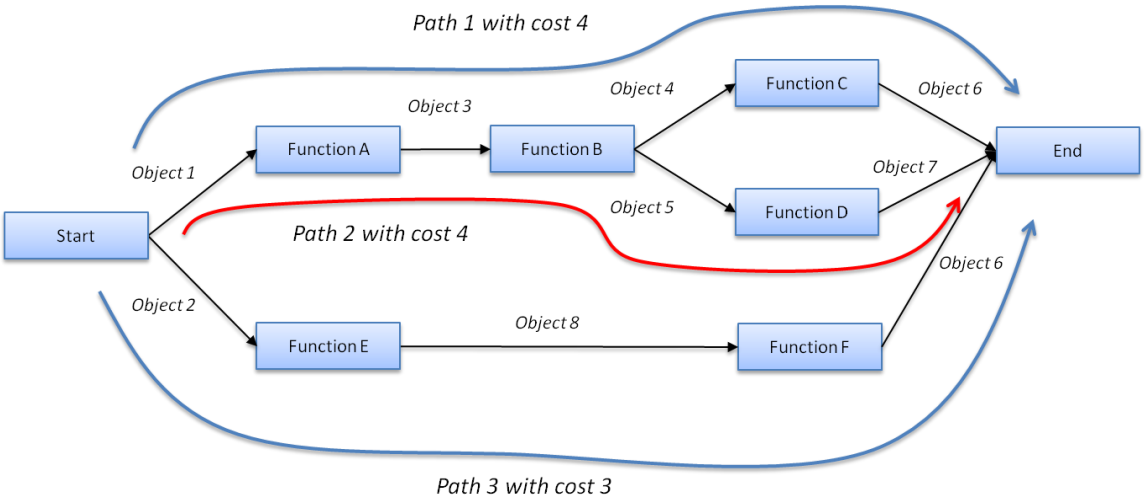

Figure 7-12 Cost over estimation when branch exists

To address this issue, we propose the following cost-adjustment method. The cost-adjustment divides the cost up to the precedent vertex by the degree of the outgoing

edges when relaxing the adjacent vertices of the precedent vertex. The pseudo code of the cost-adjustment is presented below.

COST-ADJUSTMENT* (e_o)

1 $v_s = e_o.v_s,\ v_t = e_o.v_t$

2 $c_{new} = \frac{v_s.c}{(outgoing\ degree\ of\ v_s)} + e_o.c$

3 Get a local cost c_r of the resource in $v_t.i$ using $e_o.o$ as a key

4 **if** $c_{new} < c_r$

5 **then** $c_r = c_{new}$

6 update $v.c$ with new c_r // $v.c = \sum_{i=1}^{k} c_{r_i}$

For example, when processing the *Function B* in the Figure 7-13, the *Function C* will be relaxed by the following way:

2 (cost up to *Function B*) / 2 (the outgoing degree of *Function B*) + 1 (edge weight) = 2

Figure 7-13 below shows the result of the cost-adjustment.

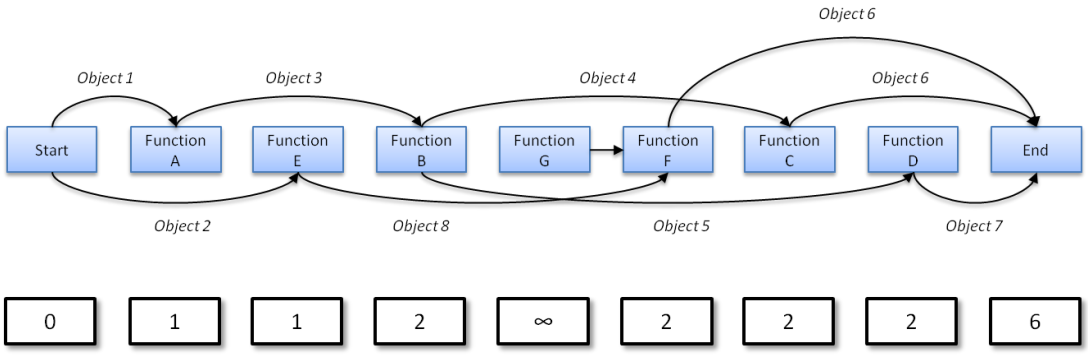

Figure 7-13 Re-relaxation result

The following definition, theorems, and corollary present how the cost-adjustment can guarantee that the estimated cost is always lower than or equal to the actual minimum cost.

Definition 7.7 (Branching Vertex). Let $G = (V, E)$ be a weighted, directed graph with weight function $w : E \rightarrow \mathbb{R}$. Any $v \in V$ that satisfies the followings is defined as a branching vertex:

The size of $v.E_o$ is greater than 1 such that v has at least two outgoing edges regardless of the object on the edges.

v has precedent vertices that provide all inputs that are required to invoke v.

v is not a start vertex.

Theorem 7.1 (Upper-bound without any branching vertices)

Let $G = (V, E, w)$ be a Composition Network. Assume that the graph is relaxed by *RELAXATION**. Let $G'(V', E', w)$ be the minimum cost solution graph of G that has $s \in V'$ as a start vertex and $v \in V'$ as a terminal vertex. Assume that G' does not have any

branching vertices, $v.c_t = \delta(s, v)$ for all $v \in V$ after the *COST-ADJUSTMENT*, where $v.c_t$ is the estimated cost of *G'* and $\delta(s, v)$ is the actual minimum cost of *G'*.

Proof Assume that v has n inputs, where $n > 1$. Then, $\delta(s, v) = c(sg_1) + c(sg_2) + \ldots + c(sg_n)$, where $c(sg_k)$ is the cost of the minimum cost solution graph for the input k of v. There is no overlapping vertices or edges between the solution graphs, because *G'* does not have any branching vertices. Thus, taking the minimum cost solution graph for each input of v always guarantees the total minimum cost of v as the RELAXATION method does. Since there is no branching vertex, COST-ADJUSTMENT is exactly the same with the RELAXATION. Thus, $v.c_t = \delta(s, v)$ for all v ∈ V

Theorem 7.2 (Upper-bound with one branching vertex)

Let $G = (V, E, w)$ be a Composition Network. Assume that the graph is relaxed by *RELAXATION**. Let $G'(V', E', w)$ be the minimum cost solution graph of G that has $s \in V'$ as a start vertex and $v \in V'$ as a terminal vertex. Assume that *G'* has only one branching vertex $u \in V'$ and u has n outgoing edges, where $n > 1$. Then, $v.c_t \leq \delta(s, v)$ for all $v \in V$ after the *COST-ADJUSTMENT*, where $v.c_t$ is the estimated cost of *G'* and $\delta(s, v)$ is the actual minimum cost of *G'*.

Proof *δ(s, v)* is the sum of weights on all $e \in E'$. Since there is no branching vertices in the precedent vertices of *u*, *δ(s, u)* = $u.c_t$ by Theorem 7.1.

Let *G''(V'', E'', w)* be the minimum cost sub-solution graph of *G'* that has *s* as a start vertex and *u* as a terminal vertex. Then, $E'' \subset E'$ and we can represent *δ(s, v)* by the following.

$\delta(s, v)$ = (the sum of the weights on $\forall\, e' \in E''$) + *X*, where *X* is the sum of the weights on $\forall\, e \in E' - E''$.

$$= \delta(s, u) + X$$

After the *COST_ADJUSTMENT*, add $\frac{u.c_t}{n}$ to the cost of each outgoing edges of *u* and set 0 for $\forall\, e \in E''$. Let E_o be the set of the outgoing edges of *u*.

If $(\forall\, e \in E_o) \in E'$, then

$$v.c_t = \left(\frac{u.c_t}{n}\right) * n + X = u.c_t + X = \delta(s, u) + X = \delta(s, v)$$

Else if $(\exists\, e \in E_o) \in E'$, then

$$v.c_t = \left(\frac{u.c_t}{n}\right) * k + X, \text{where k} < n$$

$$< u.c_t + X = \delta(s, u) + X = \delta(s, v)$$

Thus, $v.c_t \leq \delta(s, v)$. ■

Theorem 7.3 (Upper-bound with multiple branching vertices)

Let *G = (V, E, w)* be a Composition Network. Assume that the graph is relaxed by *RELAXATION**. Let *G'(V', E', w)* be the minimum cost solution graph of *G* that has $s \in V'$ as a start vertex and $v \in V'$ as a terminal vertex. Then, $v.c_t \leq \delta(s, v)$ for all $v \in V$ after the *COST-ADJUSTMENT*, where $v.c_t$ is the estimated cost of *G'* and $\delta(s, v)$ is the actual minimum cost of *G'*, and this invariant is maintained over any number of branching vertices in *G'*.

Proof We prove the invariant $v.c_t \leq \delta(s, v)$ for all vertices $v \in V$ by induction over the number of branching vertices in *G'*. Let the number of branching vertices in *G'* be *k*. For the basis ($k = 1$), $v.c_t \leq \delta(s, v)$ is certainly true by the Theorem 7.2.

For the inductive step, consider there are *n* branching vertices in *G'*. By the inductive hypothesis, $v.c_t \leq \delta(s, v)$ for all the number of branching vertices equal or less than *n* in *G'*.

Let $G_{n+1}(V_{n+1}, E_{n+1}, w)$ be the minimum cost solution graph of *G'* that has *s* as a start vertex and u_{n+1} as a terminal vertex, where u_{n+1} is the last branching vertex in topological order in *G'*. Let $U = \{u_1, \ldots, u_n\}$ be the other branching vertex in *G'*.

If $(\forall\, u\ \in U) \notin V_{n+1}$, then u_{n+1} does not have any precedent branching vertices. Thus, $v.c_t \leq \delta(s, v)$ by Theorem 7.2.

If $(\exists\, u\ \in U) \in V_{n+1}$, let u_m be the last branching vertex in G_{n+1} where 1 ≤ m ≤ n. Since G_{n+1} has the number of branching vertices equal or less than *n*, by the inductive hypothesis, $u_{n+1}.c_t \leq\ \delta(s, u_{n+1})$.

Since $E_{n+1} \subset E'$, we can represent $\delta(s, v)$ by the following.

$\delta(s, v)$ = (the sum of the weights on $\forall\, e'\ \in\ E_{n+1}$) + *X*, where *X* is the sum of the weights on $\forall\, e\ \in\ E' - E_{n+1}$.

$\geq\ \delta(s, u_{n+1})\ +\ X$

$\geq\ u_{n+1}.c_t\ +\ X$

After the *COST-ADJUSTMENT*, add $\frac{u_{n+1}.c_t}{j}$ to the cost of each outgoing edges of u_{n+1}, where *j* is the outgoing degree of u_{n+1} and set 0 for $\forall\, e\ \in\ E_{n+1}$ Let E_o be the set of the outgoing edges.

If $(\forall\, e\ \in\ E_o) \in E'$, then

$$v.c_t\ = (\tfrac{u_{n+1}.c_t}{j}) * j\ +\ X = \ u_{n+1}.c_t + \ X \leq\ \delta(s, u_{n+1})\ +\ X = \ \delta(s, v)$$

Else if $(\exists\, e\ \in\ E_o) \in E'$, then

$$v.c_t = (\frac{u_{n+1}.c_t}{j}) * k + X, \text{where k} < j$$

$$< u_{n+1}.c_t + X \leq \delta(s, u_{n+1}) + X = \delta(s, v)$$

Thus, the invariant is maintained. ■

The following corollary proves that the estimated cost resulting from the cost-adjustment process is an admissible heuristic for searching AND/OR graph that is converted from a given *CN*. Thus, the estimated cost can be used as a heuristic function for the search algorithm in section 7.2.3.

Corollary 7.1

The estimated cost resulting from the cost-adjustment process is an admissible heuristic for AND/OR graph search.

Proof An admissible heuristic is used to estimate the cost of reaching the goal state in an informed search algorithm. In order for a heuristic to be admissible to the search problem, the estimated cost must always be lower than or equal to the actual cost of reaching the goal state. By the Theorem 7.1, 7.2 and 7.3, the estimated cost is always lower than or equal to the actual cost. Thus, the estimated cost resulting from the cost-adjustment process is an admissible heuristic. ■

7.2.2.6 Cycle Detection and Resolution

Strongly connected component of a directed graph is a maximal set of vertices C such that for every pair of vertices u and v in C, vertices u and v are reachable from each other. Therefore, if a directed graph has a strongly connected component, then the graph contains a cycle. The strongly connected component can be detected by using the depth first search. Once the strongly connected components are identified, we can resolve the cycle issue by replacing all vertices in the strongly connected components with new vertex. The detailed procedure and example are shown in the following pseudo code and Figure 7-14 below.

CREATE-STRONGLY-CONNECTED-VERTEX (*CN*)

1 Compute $u.f$ for each vertex u by calling **DEPTH-FIRSR-SEARCH**(CN) // $u.f$ is the finishing time of u

2 compute CN^T // $CN^T = (V, E^T)$, where $E^T = \{(u, v) : (v, u) \in E \text{ of } CN\}$

3 call **DEPTH-FIRSR-SEARCH**(CN^T) in order of decreasing $u.f$

4 output the vertices of each tree in the depth-first forest formed in line 3 as a separate strongly connected vertex

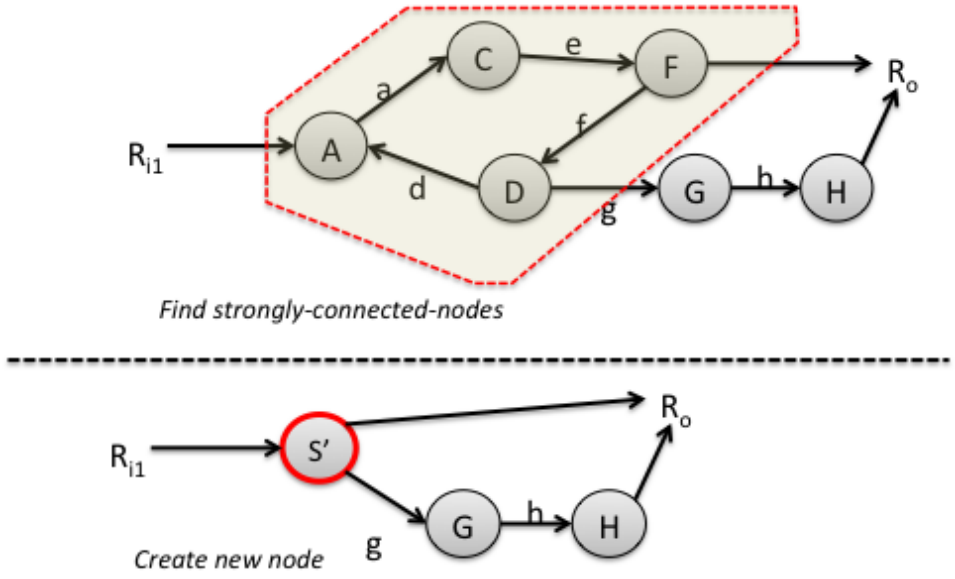

Figure 7-14 Cycle Detection and Resolution

7.2.2.7 Transformation to AND/OR graph

After the composition network pruning and cost estimation, the composition network is transformed into AND/OR graph for search. Figure 7-15 below shows the transformed AND/OR graph. Each AND vertex has an estimated cost that is represented in the parenthesis.

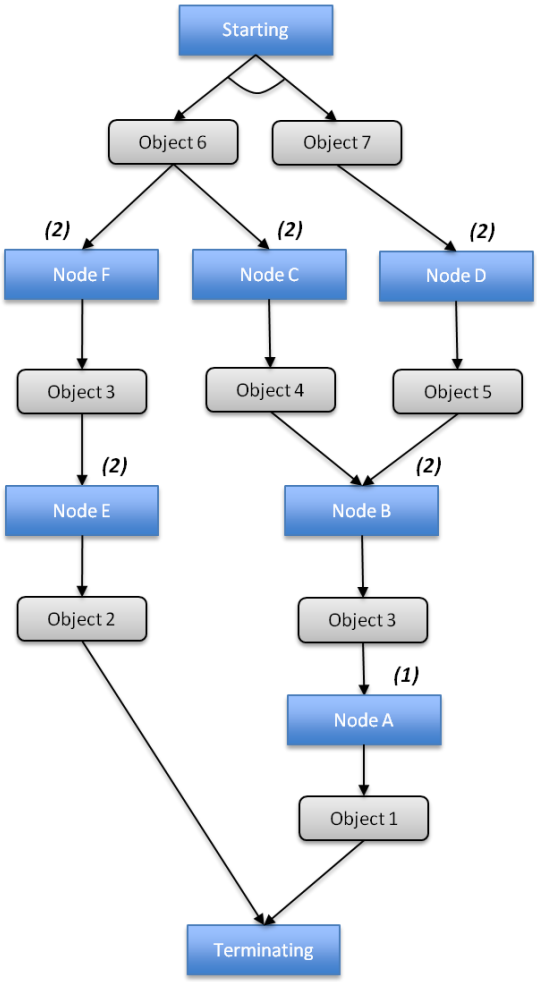

Figure 7-15 Resulting AND/OR graph

7.2.3 Search Algorithm

Nilsson (1971) introduced the AND/OR graph, or A/O graph for short, and A/O graph search problem for the first time, and since then various types of AND/OR graph search methods have been proposed.

AND/OR graph search methods can be categorized in multiple ways depending on the criterions. There are two mainly different approaches including top-down and bottom-up. In the top-down approaches, the graph search begins at the start vertex. The start vertex corresponds to the goal condition in our AND/OR graph representation. And then, the top-down search explores its child vertices, until it reaches the terminating vertex that represents the initial condition in our case. Martelli and Montanari (1978), Chakrabarti et al. (1989), and Chakrabarti (1994) are examples of top-town search algorithms.

On the other hand, in the bottom-up approaches, the search starts from the terminating vertex that represents the initial condition and explores the ancestor vertices until reaching the start vertex. Martelli and Montanari (1973) and Chakrabarti (1994) are examples of the bottom-up approaches.

AND/OR graph search algorithms can also be categorized into explicit-graph search and implicit-graph search methods. The implicit graph is a graph whose vertices or edges are not represented as explicit objects in a computer's memory, but rather are determined algorithmically from some more concise input. The explicit-graph search uses an explicit data representation for the vertices and edges of an AND/OR graph, while the implicit-graph search uses rules to represent them. AO* is an example of an implicit-graph search method.

Finally, the AND/OR graph search methods can be classified as admissible and inadmissible. Admissible algorithms guarantee that an optimal solution will be found, if

one exists. Inadmissible algorithms cannot guarantee that the solution found is an optimal solution.

Our objective is to develop admissible search method to find the minimum cost solution graph beginning from the start vertex and leading to the terminating vertex. In the previous section, the composition network pruning and cost estimation was a kind of implicit-graph search, while the AND/OR graph search in this section is a kind of explicit-graph search, because we already have a set of candidate vertices with explicit relations between the vertices and the estimated costs.

7.2.3.1 Notation

We follow the standard notation and definitions stated in Mahanti and Bagchi (1985).

- *G* is the entire problem graph that results from the composition network pruning.
- All vertices *u* in *G* has a finite set of successors *S(u)*.
- All vertices *u* in *G* have an estimated cost *h'(u)* that results from the cost estimation. This estimate will be used to guide the search and reduce the number of expanded vertices.
- All arc *(u, v)* in *G* has a fixed cost $c(u, v) > 0$.
- *P(u)* denotes the set of predecessors of vertex *u*.
- For any vertex *u* in *G*, *D(u)* denotes a solution graph.

- The subgraph of *G* that is generated up to a certain point is called the explicit graph G'. A cost function *h(v, G)* on each vertex *v* in *G* is defined as following way:
- h(v, G) = greatest lower bound {h(v, D(v)) | D(v) is a solution graph with root v in G}, where, for a vertex v in D(u),
  - *h(v, D(u)) = 0*, if *v* is a terminating vertex
  - *h(v, D(u)) = c(v, v') + h(v', D(u)),* if *v* is an OR vertex and *v'* is *v*'s immediate successor in *D(u)*
  - h(v, D(u)) = $\sum_{i=1}^{k}(c(v, vi) + h(vi, D(u)))$, if v is an AND vertex with immediate successors v1, v2, …, vk in D(u)

The result of the composition network pruning and cost estimation screens out unnecessary vertex and edges. Thus, there must be a solution from any vertex in *G*. That is for any vertex *u* in G, *h(u, G)* ∈ ℝ.

7.2.3.2 Algorithm

Our AND/OR graph search algorithm proceeds in a top-down fashion, where each vertex expansion step is followed by a bottom-up cost revision like all AO∗ algorithms [Pearl 1984]. The following pseudo code describes the procedure of out algorithm.

SSCA-AND/OR-Graph-Search (*G*, *s*) // *G* is an AND/OR graph and *s* is a start vertex

1 **If** s is a terminal vertex, **Then** label s SOLVED

2 create *G'* and add s to *G'*

```
3        While s is not SOLVED
4            choose any unsolved successor vertex u below s; expand u generating all its
immediate successors S(u);
5            for each v ∈ S(u) not in G'
6                add v to G'
7                If v is a terminal vertex, label it SOLVED
8                Else compute h(v)
9                    If h(v) > h'(v), Then relaxation to the predecessors and BREAK
10           for each v ∈ S(u) in G'
11               relaxation to the predecessors assuming h(v) = 0
12           for any v ∈ S(u)
13               If v is AND vertex and v has predecessors other than S(u),
14                   Then relaxation to the predecessors assuming h(v) = 0
15           Re-compare the cost of immediate successor vertex u below s, and set the
minimum cost as SOLVED
```

The outer loop of the algorithm implements the top-down growth of G', while the inner loop carries out the bottom-up cost revision. The estimated costs are revised from the expanded vertex up along marked arcs as well as the other arcs if there exists an AND vertex on the path. This revision process may change the cost of the successor vertices below the start vertex that may leads find to an alternative, more promising paths.

The procedure is similar to the AO* algorithm. The main difference is the cost revision process. Like AO* algorithm, our algorithm also propagates its new cost back up through the graph, if current vertex has been labeled SOLVED or its cost was just changed. In addition to that, if current path reaches to the terminating vertex and there exist any AND

vertex on the path that has another predecessor paths, then our algorithm updates the cost of the vertices on the other predecessor paths assuming the cost of the current AND vertex is 0. This cost revision process is necessary, because what we try to achieve is to find a minimum cost subgraph, not just a path, and there might be a shared path in a subgraph as we presented in section 7.2.2.5.

Mahanti and Bagchi (1985) has proven that, if the cost estimation is admissible (i.e., $h'(u) \leq h(u, G), \forall n \in G$), then AO* like algorithms terminate by either finding a minimum-cost solution graph rooted at *s* or else returning h(s) = ∞. In our case, once we have the AND/OR graph from the composition network, there must be a solution. And, as already proved in section 7.2.2.5, our cost estimation method guarantees $h'(u) \leq h(u, G), \forall n \in G$.

7.3 Experiment

In the last of the previous section, we presented theoretical time complexity of our service search and composition algorithm. In this section, we compare the performance of our algorithm (SSCA) and other prominent existing AI planners in terms of effectiveness and computational efficiency.

7.3.1 Existing AI Planners and search strategies for the experiment

We chose *OptaPlanner* (V6.0.1) and *BlackBox* (V4.5) for the performance comparison. The *OptaPlanner* is a lightweight, embeddable planning engine based on a constraint satisfaction solver [OptaPlanner 2015]. Since the *OptaPlanner* provides various sophisticated optimization heuristics and algorithms, it enables us to compare the performance of our algorithm with various combinations of optimization heuristics. Throughout our experiments, we use *Tabu Search* [Glover 1989], *Hill Climbing* [Gent and Walsh 1993], and *Simulated Annealing* [Davis 1987] as optimization heuristics and algorithms for the constraint satisfaction. All of the aforementioned optimization heuristics and algorithms are kind of a *Local Search* method. The *Hill Climbing* (HC) is a mathematical optimization technique which belongs to the family of local search. It is an iterative algorithm that starts with an arbitrary solution to a problem, then attempts to find a better solution by incrementally changing a single element of the solution. If the change produces a better solution, an incremental change is made to the new solution, repeating until no further improvements can be found. The *Hill Climbing* method can get stuck on local maxima. To avoid that, the *Tabu Search* (Tabu) maintains a list of k previously visited states, and prevents the search from revisiting them. The *Tabu Search* works like the *Hill Climbing*, but it maintains a tabu list to avoid getting stuck in local optima. The tabu list holds recently used objects that are taboo to use for now. Move that involve an object in the tabu list, are not accepted. The tabu list objects can be anything related to

the move, such as the planning entity, planning value, move, solution, etc. The *Simulated Annealing* (SA) exploits the analogy between how metal cools and freezes into a minimum-energy crystalline structure and the search for a minimum (or maximum) in a general system. SA can avoid becoming trapped at local minima. SA uses a random search that accepts changes that increase objective function f, as well as some that decrease it. SA uses a control parameter *T*, which by analogy with the original application is known as the system "temperature". *T* starts out high and gradually decreases toward *0*. A "bad" move from A to B is accepted with a probability, $e^{-(\mathrm{f(B)} - \mathrm{f(A)/T})}$. The higher the temperature, the more likely it is that a bad move can be made. As *T* tends to zero, this probability tends to zero, and SA becomes more like hill climbing. If *T* is lowered slowly enough, SA is complete and admissible.

Typically the *Local Search* methods need to start from an initialized solution. The Optaplanner also provides various methods (called construction heuristics) to generate the initialized solution. In our experiments, we use *First Fit* algorithm [REF] for the construction heuristics.

The *Blackbox* is a planning system that combines best features of *Graphplan*, *SATPLAN*, and new randomized systematic search engines. The *Blackbox* converts problems described in *STRIPS* [Fikes and Nilsson 1972] into Boolean satisfiability (SAT) problems, and then solving the problems with existing satisfiability engines. The front-end of the *Blackbox* employs the Graphplan system [Blum and Furst 1995] and for the SAT

problem, *Blackbox* applies the local-search SAT solver such as *Walksat* [Selman, Kautz, and Cohen 1994] and *Satz* [Li and Anbulagan 1997].

7.3.2 Evaluation Matrix, Assumption, and Test Environment

For the experiments, we use two evaluation metrics including execution time and total cost of a solution. The execution time is to measure how long each algorithm takes to find a solution. Since the test data is stored in a relational database, it is necessary to load the test data from the database and convert into specific format for each planner (e.g., conversion to PDDL for the *Blackbox*). The execution time is to measure the computational efficiency of each algorithm. Thus, we excluded such data preparation time and measured the time required purely inside of each planner. The time unit in this experiment is a millisecond.

The total cost of a solution is to measure the quality of the solution obtained. The total cost is the sum of the weights on edges in a solution. In practice, the smaller number of edges does not always guarantee the minimum cost, because the cost of edges varies depending on the degree of the composability of the two vertices to be connected. However, the *Blackbox* is a kind of optimal parallel planners that are designed to minimize the number of time steps, but not necessarily the number of actions and arcs between the actions. Moreover, in the case of the *Blackbox*, the objects in the effect of one action must be exactly matched with the objects in the precondition of the subsequent action while our algorithm and the CSP-based *Optaplanner* do not have such restriction.

Therefore, just for the performance comparison, we assume that all edges have a uniform cost and also we impose the restriction such that exact matching of the object is required. As a result, in our experiment, the algorithm that takes less execution time while having smaller number of edges in a solution will be considered better than others.

The local search methods in the CSP-based planners generally do not know when it finds the optimal solution. Thus, finding the optimal solution could take a large amount of time, if the search space of the problem is huge. Therefore, it is very important to notify the CSP-based planners when to stop execution.

Typically, termination of the local search methods can be based on a specific time bound. In addition, we can terminate the local search methods when the acceptable (or optimal) solution found. However, in this case, the user must inform the CSP-based planner of the acceptable (or optimal) solution in advance. In practice, the user does not know the optimal solution in advance. Thus, typically, if the user specifies the time bound, the planner tries to find the best solution until the time is up. However, for our experiment, the time bound method is not good, because the execution time is always the same with the specified time period and thus, we cannot exactly measure how long the algorithm takes to find the optimal solution.

Thus, in our experiment, we inform the CSP-bases planner of the optimal solution in advance so that the planner stops solving as soon as the optimal solution is found. Then, since there is no time bound, the planner always finds an optimal solution and we can

figure out the required time for finding the optimal solution. The required time can be used to compare the execution time with other planners. In addition, once we have the required time, we can answer various time bound related questions such as ‘can the planner find the optimal solution in 10 seconds?’.

The experiments were performed on Mac OS X version 10.9.5 with 3.5 GHz Intel Core i7 and 8GB 1600 MHz DDR3 RAM.

7.3.3 Experiment by varying the number of vertices in the data set

The first experiment is to analyze the correlation between the number of vertices and the performance of each planner.

7.3.3.1 Test Data

For the experiment, we have randomly generated the test data with the following way.

- Total 50 different objects are generated. The objects are used to describe the input and output of each vertex.
- Each vertex has at least one and at most three inputs and outputs. The number of inputs and outputs are randomly selected within the restriction.
- There is no duplication between inputs and outputs of each vertex. For example, if a vertex has an object as an input, then the object cannot be used as an output.

- For each of the test data set, we created user's initial condition and goal condition as well as solutions.

We have generated total 21 different sizes of test data set by varying the size of vertices as 10, 100 – 900, and 1,000 – 10,000. Each test data set has 4 possible solutions and 1 optimal solution. The optimal solution is a solution graph that has 3 vertices. Figure 7-16 below illustrates the solution graph and the optimal solution.

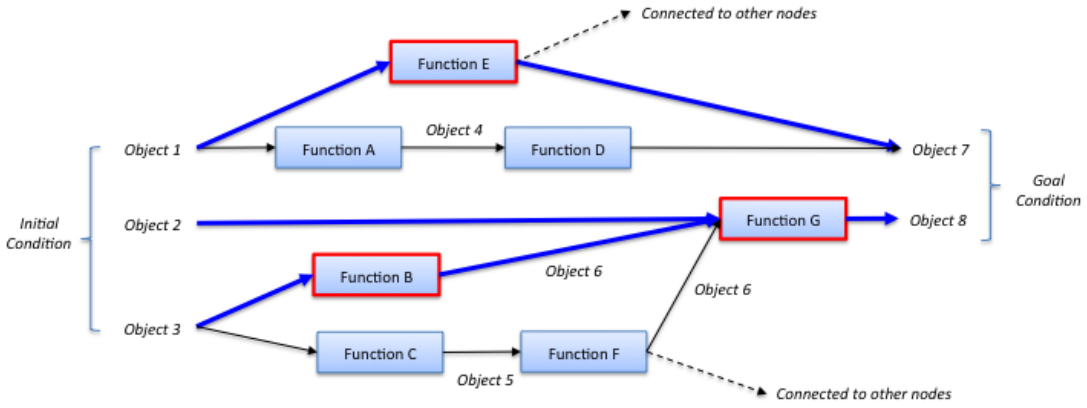

Figure 7-16 Solution subgraph and optimal solution

7.3.3.2 Result

Table 7-1 and Figure 7-17 below show the result of the first experiment. The bold font in the table represents the best performance and the underscore represents sub-optimal result. As shown in the Table 7-1 and Figure 7-17, the *SSCA* and the *Blackbox* outperform the CSP-based methods. Figure 7-18 below shows the performance comparison between the *SSCA* and *Blackbox*. The *Blackbox* produced suboptimal solutions when the number of

vertices is 200, 300, and 500. The number in the parenthesis right after the execution time in the Table 7-1 shows the number of resulting vertices (the optimal solution has 3 vertices).

Table 7-1 Test Result

# of Vertices	SSCA	Blackbox	CSP (Tabu)	CSP (SA)	CSP (HC)
10	**0.002**	0.007	0.037	0.032	0.027
100	**0.004**	0.013	0.089	0.078	0.050
200	**0.013**	0.020 (5)	0.079	0.094	0.190
300	**0.024**	0.026 (5)	0.093	0.091	0.297
400	**0.003**	0.004	0.098	0.134	0.148
500	0.055	**0.047 (4)**	0.096	0.138	0.187
600	**0.004**	0.004	0.096	0.177	0.218
700	**0.003**	0.005	0.109	0.220	0.231
800	**0.003**	0.006	0.113	0.316	0.457
900	**0.004**	0.004	0.115	0.107	0.300
1000	**0.002**	0.005	0.115	0.161	0.215
2000	**0.003**	0.006	0.390	0.952	0.489
3000	**0.002**	0.005	0.415	0.576	0.409
4000	**0.004**	0.007	0.489	0.445	0.850
5000	**0.005**	0.007	0.603	1.147	0.966
6000	**0.005**	0.008	0.601	0.340	1.542
7000	**0.003**	0.008	0.917	1.887	0.434
8000	**0.006**	0.010	1.237	0.926	1.001
9000	**0.005**	0.009	0.659	0.432	1.355
10000	**0.006**	0.009	0.664	2.351	0.590

The execution time in both *SSCA* and *Blackbox* is quite steady regardless of the increase of the number of vertices while the execution time in CSP-based planner is inclined to increase in proportion to the number of vertices. That is mainly due to the fact that both the *SSCA* and *Blackbox* identify solution candidates in the forward search phase by expanding a graph from the initial states until all goal states appear. Since, in this first experiment, all the test data set has a small number of vertices in the solution graph and also each vertex has at most 3 outputs, the number of the solution candidates is small, thus those are identified quickly and that significantly reduce the entire search space. On the other hand, in the case of CSP-based planners, there is no such pruning process, thus the execution time increases as the number of vertices is increased.

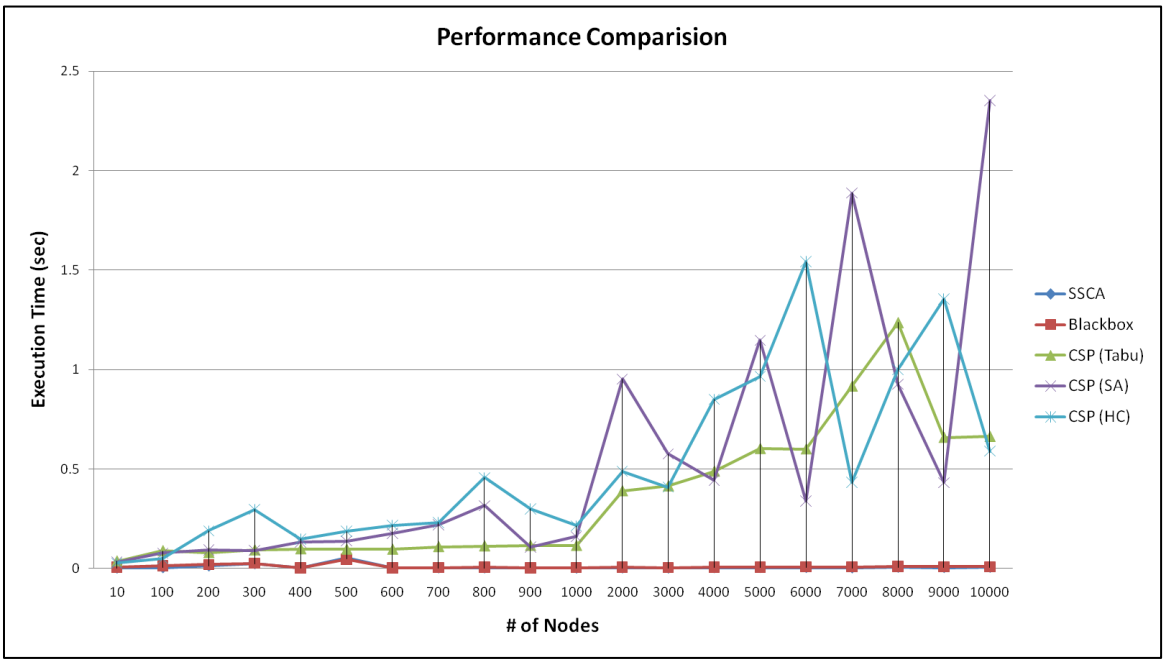

Figure 7-17 Performance comparison by varying the number of vertices

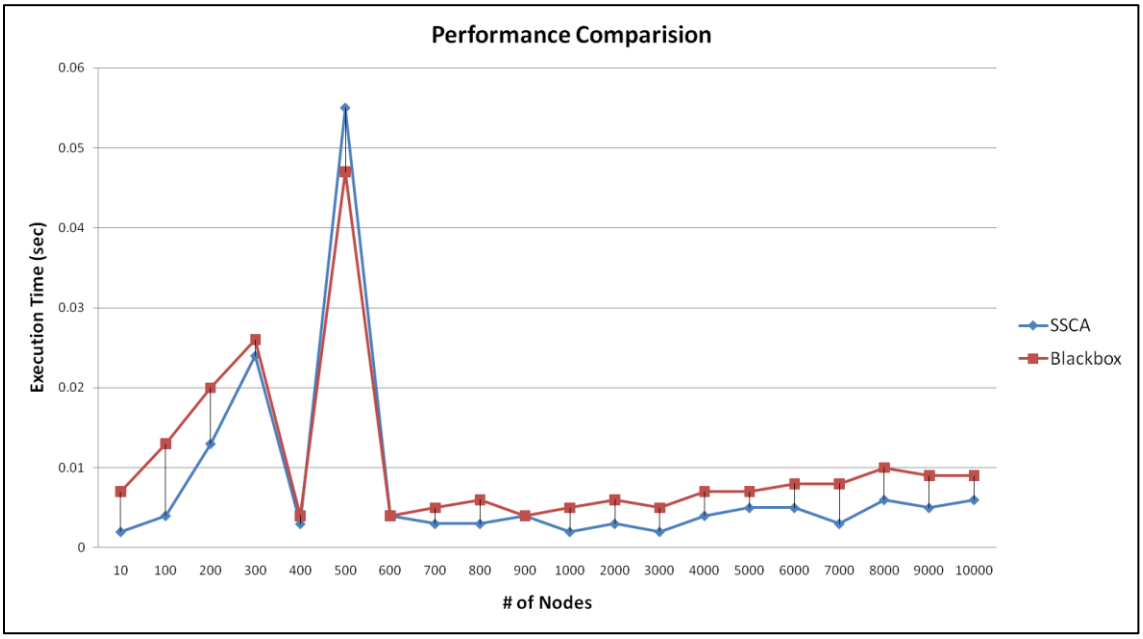

Figure 7-18 Performance comparison between SSCA and Blackbox

7.3.4 Experiment by varying the outgoing degree of vertices

From the result of the first experiment, we observe that the number of vertices is not critical in both of the *SSCA* and *Blackbox*. As described in the result of the first experiment, both the *SSCA* and *Blackbox* identify solution candidates in the forward search phase by expanding a graph from the initial states until all goal states appear. If each vertex has more outgoing degree, then more vertices are expanded in the forward search phase and it would require longer search time. Thus, the second experiment is to

analyze the correlation between the outgoing degree of vertices and the performance of each planner.

7.3.4.1 Test Data

For the second experiment, we have randomly generated the test data with the followings ways.

- We generated total 200 different objects to describe the input and output of each vertex.
- We have generated total 13 different test data sets by varying the outgoing degree of vertices as 1, 2, 3, 4, 5, 10, 15, 20, 25, 30, 35, 40, and 45. Outgoing degree 10 means that each vertex has 10 output.
- Each vertex has only one input.
- There is no duplication between inputs and outputs of each vertex. For example, if a vertex has an object as an input, then the object cannot be used as an output.
- For each of the test data set, we created user's initial condition and goal condition as well as solutions.
- Each test data set has a total of 1,000 vertices.
- Each test data set has 4 possible solutions and 1 optimal solution. The optimal solution is a solution graph that has 3 vertices as shown in Figure 7-16.

7.3.4.2 Result

Table 7-2 and Figure 7-19 below show the result of the second experiment. Same as the Table 7-1, the bold font in the table represents the best performance and the underscore represents sub-optimal result. As shown in the Table 7-2 and Figure 7-19, the *SSCA* outperforms the other methods. The *Blackbox* produced suboptimal solutions when the outgoing degree is 2 and 3. The number in the parenthesis after the execution time shows the number of resulting vertices in the suboptimal solution. And, the *Blackbox* didn't work at all when the outgoing degree is greater than 3.

Table 7-2 Test result by varying the outgoing degree of vertex

Outgoing degree	SSCA	Blackbox	CSP (Tabu)	CSP (SA)	CSP (HC)
1	**0.002**	0.002	0.161	0.183	0.262
2	0.024	**0.011** (4)	0.292	0.173	0.402
3	**0.032**	0.04 (4)	0.396	0.265	0.139
4	**0.041**	-	0.467	0.369	0.456
5	**0.043**	-	0.469	0.104	1.251
10	**0.124**	-	0.968	0.832	0.561
15	**0.176**	-	0.895	1.986	1.725
20	**0.201**	-	1.496	1.311	0.824
25	**0.285**	-	2.783	1.703	0.531
30	**0.368**	-	2.547	2.336	1.878
35	**0.381**	-	0.576	3.886	2.285
40	**0.433**	-	1.777	4.406	11.152

45	**0.462**	-	2.837	4.637	3.212

The execution time of the *SSCA* increased in proportion to the degree of vertices as shown in Figure 7-20. This result is quite apparent that when the *SSCA* expands a graph from the initial states until all goal states appear, the expandability of the graph is proportional to the outgoing degree of the already expanded vertices. Thus, the more outgoing degree each vertex has, the more vertices are expanded in the forward search and it would result longer search time. In the case of the CSP-based planner, the execution time is inclined to increase in proportion to the number of vertices, but in some cases, there exists sudden increase or decrease. The performance of the CSP-based planner depends on the initial solution generated by the construction heuristics. If the initial solution is closer to the goal solution, then we can get the goal solution relatively quickly, otherwise it would take longer.

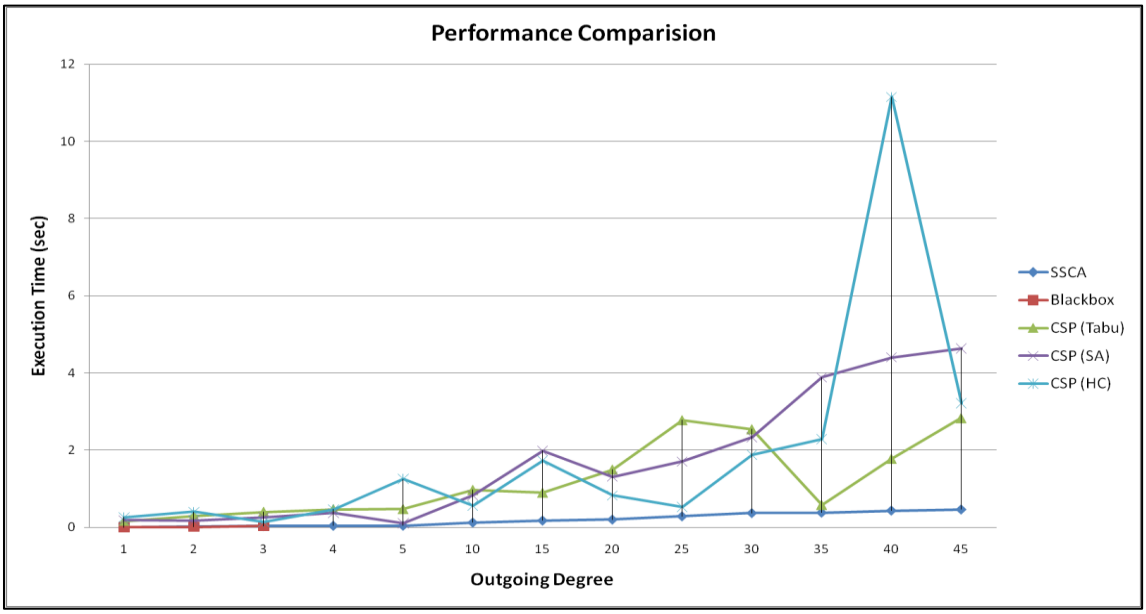

Figure 7-19 Performance comparison by varying the outgoing degree of vertices

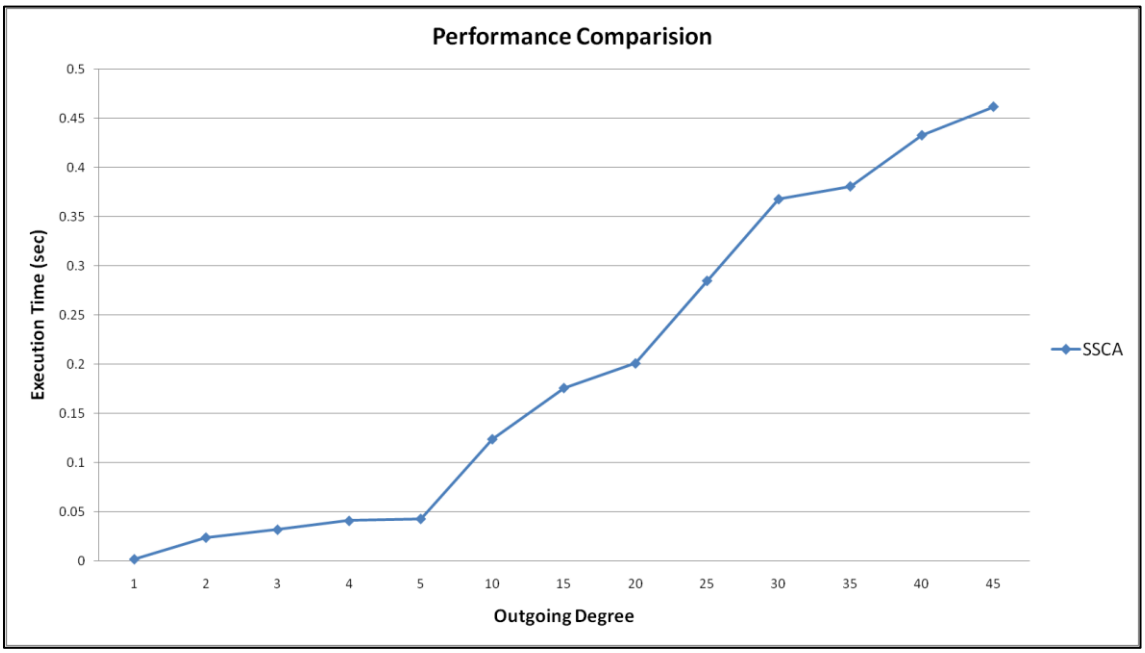

Figure 7-20 Execution time of SSCA

7.3.5 Discussion

The two experiments above show that *SSCA* is capable of solving the service search and composition problems better than other methods in terms of performance and scalability. Specifically, the *SSCA* shows a better performance than other methods when the number of vertices is huge and the composition network is more tense. The *Blackbox* performs well, but does not work at all when the outgoing degree of vertex is greater than 3. The CSP-based planners require much more execution time than the *SSCA*. This result implies that the other methods are not as scalable as *SSCA*.

The CSP-based planners always find optimal solutions, because we did not set the time bound. However, the CSP-based planners may not find the optimal solution, if we set the time bound lesser than the execution time in the Table 7-1 and Table 7-2.

We also observed that the *SSCA* did not blow up exponentially when the number of vertices increases and the composition network becomes dense. This implies that the *SSCA* is very scalable.

Chapter 8. Conclusion and Future Works

This section describes the summary of contributions and future works. In our research, we developed a computer-aided services search and composition methods in an open cloud services marketplace environment. In detail, we developed a method for formally representing a service in terms of composability by considering various functional and non-functional characteristics of services. The reference ontology is one of the most important components to formally represent a service. To come up with the reference ontology, we explored a bottom-up-based statistical method. After that, we architected a framework that encompasses the developed reference models, effective strategy, and necessary procedures for the services search and composition. Finally, we developed a graph-based algorithm that is highly specialized for services search and composition. The algorithm takes into account not only functional but also non-functional characteristics of services and also scales well with the large number of services available. Experimental comparative performance analysis against existing automatic services composition methods is also provided.

8.1 Summary of contributions

The followings are more details of the major contributions of our research.

8.1.1 Develop a method for representing a service's functionality

We analyzed what must be considered, to make different services composable, whether the condition differs in different types of services (software or hardware) as well as what various aspects of composability are (functional or non-functional). Based on the analysis, we developed a functional representation model by formalizing various functional and non-functional characteristics. For the functional and non-functional characteristics, we investigated and adapted the notions of functional and non-functional requirements from the requirement engineering discipline.

In addition to that, we also found that some of constraints on input or output are very important when composing different services. Moreover, we found that some of quality related characteristics can be transferable to constraints when services to be composed.

Thus, our functional representation takes into account various functional and non-functional characteristics of services as well as other very important constraints and quality related characteristics.

8.1.2 Develop a method for aiding the ontology development

We explored the bottom-up-based statistical ontology development method that minimizes the subjective judgment of ontology developers. Typically, human intervention is essential in making choices at various levels when developing or evolving an ontology. However, human inputs need to be taken into account within a controlled

manner to prevent various possible conflicts due to the differences of the perceptions, experiences, and understanding specific to each ontology developer. Our method is a mixture of statistical and other computational methods such that the ontology can be developed and evolved in a way that the outcome of the process is repeatable. Thus, the resulting reference ontology will be identical, when started from the same initial conditions.

8.1.3 Develop an effective composability analysis framework

We proposed the framework that provides a high-level design of components, strategy, and procedure for the services search and composition. We identified two essential components: the *Reference Models Repository* and *Service Registry & Repository*. The *Reference Models Repository* contains the resource, state, and function ontology that enables formal representation of services. The *Service Registry & Repository* contains service descriptions that specify the service's functional and non-functional characteristics using the concepts defined in the *Reference Models Repository.*

For effective strategy and procedure to service search and composition, we decoupled the composition problem into two levels: function and service level composition. Such problem decomposition enables reduction in computational complexity of the services composition problem and allows more flexibility by allowing the user to adjust the service-level solution when needed with having to re-computing the function-level solution.

The identified components within the framework shall assist the user in discovering and composing services in a large-scaled cloud services repository (i.e., open cloud marketplace) and shall have the flexibility to deal with various aspects of functional and non-functional user requirements.

8.1.4 Develop a specialized algorithm for services search and composition

We designed and implemented a highly specialized algorithm for services search and composition. We modeled the service search and composition problem into an AND/OR graph by considering the complexity of actual service network that have multiple inputs and outputs as well as logical AND/OR relations. Typically, graph search problems require a criterion to compare which one is the best solution. In service search and composition problem, one possible method is to assign weights to the edges. For the weight function, we assign the weight on the edges proportional to the difficulty of service composition. The difficulty of the service composition is quite subjective depending on users' preferences or expertise. For example, one user who is an expert in handling message type conflict may easily address the message type mismatch while the other one who has a specialty in security could handle the encryption algorithm mismatches. Thus, we considered the various characteristics of the services as well as user's preferences and expertise when quantifying the difficulty as a cost.

Finding the minimum cost solution graph in the AND/OR graph is NP-Complete. In order to address the intractable problem, we have to use an approximate algorithm like

heuristic search. The most important part of the heuristic search methods is to come up with an admissible heuristic that is used to estimate the cost of reaching the goal state in an informed search algorithm. In order for the heuristic to be admissible, the estimated cost must always be lower than or equal to the actual cost of reaching the goal state. We provided the cost estimation algorithm that guarantees that the cost is always lower than or equal to the actual minimum cost. We also provided a formal proof to validate the cost estimation algorithm.

8.2 Future works

As stated in Section 4, the proposed composability analysis framework has to rely on reference models for shared semantics of the relevant concepts. The framework aims at providing an efficient method for services composition that is applicable for a wide range of domains. For that it provides a core ontology that can be further extended with specific domain specific concepts. However, both domain-specific and domain-independent concepts may still need to be added and adjusted as new requirements may be encountered over many uses of the framework. The requirement for adaptability and continuous refinement of reference models is a basic premise for our future research.

In our current research, we explored a computational aid to help develop ontology. We hope that the method would enable the ontology to be developed and evolved in a way that the outcome of the process is repeatable by minimizing ontology developers'

subjective judgment. As presented in Section 5.9, we have conducted experiment on the method using about 800 actual service providers' service description in machining domain. We plan to apply the method to other domain such as information services on the cloud and continue to refine the method.

Our function representation method is primarily based on the operational function definition that is relatively objective than other function definitions. And then, we extended the operational function definition to consider various aspects of composability between services. However, there are other important aspects of function that are needed to be considered such as purposive function that define relation between the goal of a human user and the behavior of the service. Thus, we will consider other important aspects of function in our future works.

In our composability analysis framework, there might be several cases in which the user encounters a lack of concept in the reference function models. Firstly, the user may not find appropriate concept to represent his/her requirement in the Requirement Formalization step. This results in an addition of the new concept to the resource ontology. In the Functional Design step, there might not be a possible set of functions that satisfies user's requirement. This might be due to lack of appropriate functions in the function ontology or incorrect description of some function characteristics (e.g., incorrect description of pre- or post-conditions of the function). Either case may result in the function ontology evolution. Another model evolution case may take place when service

providers register their services into the Service Registry & Repository. This is the case when the function ontology is not rich enough to describe various characteristics of the service itself or there is no appropriate function to be referenced. Thus, we are planning to come up with a method to address a lack of concepts in the reference ontology.

In our composability analysis framework, we decoupled the composition problem into two different levels of problem: function level and service level. We presented the benefits of the decoupling, but there also exists some limitations of this approach. In general, optimization at each of the two levels does not guarantee the global optimization. As described in earlier chapters, in the function composition, the framework tries to identify the optimal set of functions for the user's requirement and then retrieve a set of services that support the identified functions. After that, the framework tries to find a set of services that have a minimum composition cost. However, when we retrieve the set of services through the necessary functions, we may miss some of the services that are in the actual global optimal solution. In an extreme case, although we identify necessary functions, there may not exist services that support those functions at all. In this case, how to backtrack and regenerate alternative function designs is very important issue. Addressing the limitations should be a top priority in our future works.

As stated in Chapter 2, our research focuses on minimizing composition cost by considering various functional and non-functional characteristics of services that are relevant to the composition. Typically, services have other important characteristics that

are relevant to the quality of service such as execution price, duration, reputation, reliability, and availability [Zeng et al. 2003]. In our future research, we plan to incorporate the quality of service related characteristics into our framework. Development of new cost scheme to combine the composition cost and the quality of service will be very important research topic.

Bibliography

[Acatech 2013] Recommendations for implementing the strategic initiative INDUSTRIE 4.0. 2013. Securing the future of German manufacturing industry.

[Ameri et al. 2015] Ameri, F., Kulvatunyou, B., & Ivezic, N. (2015). A Formal Process for Community-Based Reference Model Evolution for Smart Manufacturing Systems. In Advances in Production Management Systems: Innovative Production Management Towards Sustainable Growth (pp. 30-38). Springer International Publishing.

[American Productivity and Quality Center 2014] Process classification framework V. 6.1.1.

[ANSI/ISA 2000] ANSI/ISA-95.00.01-2000, Enterprise-Control System Integration Part 1: Models and Terminology

[ANSI/ISA 2010] ANSI/ISA-88.00.01-2010 Batch Control Part 1: Models and Terminology.

[Anton 1997] Anton, A. I. (1997). Goal identification and refinement in the specification of software-based information systems.

[Apache 2015] Apache Hadoop, available online at http://hadoop.apache.org, accessed September 2015.

[Arenales and Morabito 1995]M. Arenales, R. Morabito, An AND/OR-graph approach to the solution of two-dimensional non-guillotine cutting problems, European J. Oper. Res. 84 (1995) 599–617.

[Baeza-Yates and Ribeiro-Neto 1999] R. Baeza-Yates, B. Ribeiro-Neto, Modern Information Retrieval. New York, NY: ACM Press, Addison-Wesley, 1999.

[Barkmeyer et al. 1999] SIMA Reference Architecture, Part 1: Activity Models. NISTIR 5939.

[Balakrishnan and Ranganathan 2012] Balakrishnan, R., & Ranganathan, K. (2012). A textbook of graph theory. Springer Science & Business Media.

[Barnett and Verma 1994] J.A. Barnett, T. Verma, Intelligent reliability analysis, in: Proc. 10th IEEE Conference on Artificial Intelligence for Applications, San Antonio, TX, 1994, pp. 428–433.

[Baryannis and Plexousakis 2010] Baryannis, G., & Plexousakis, D. (2010). Automated Web Service Composition: State of the Art and Research Challenges. ICS-FORTH, Tech. Rep, 409.

[Bellman 1956] Bellman, R. (1956). On a routing problem (No. RAND-P-1000). RAND CORP SANTA MONICA CA.

[BigData Graph Database 2015] BigData Graph Database, available online at http://sourceforge.net/projects/bigdata, accessed September 2015.

[Blum and Furst 1997] Blum, A. L., & Furst, M. L. (1997). Fast planning through planning graph analysis. Artificial intelligence, 90(1), 281-300.

[Breman et al. 1984] L. Breman, J.H. Friedman, R. A. Olshen, and C. J. Stone, Classification and Regression Trees, Wadsworth, California, 1984.

[Cao and Sanderson 1998] T. Cao, A.C. Sanderson, AND/OR net representation for robotic task sequence planning, IEEE Trans. Systems Man Cybernet.—Part C: Applications and Reviews 28 (2) (1998) 204–218.

[Chakrabarti et al. 1989] Chakrabarti, P.P., Ghose, S., Acharya, A., & De Sarkar, S.C. (1989). Heuristic search in restricted memory. Artificial Intelligence, 41(2), 197-221.

[Chakrabarti 1994] Chakrabarti, P.P. (1994). Algorithms for searching explicit AND/OR graphs and their applications to problem reduction search. Artificial Intelligence, 65(2), 329-345.

[Chakrabarti 1998] Chakrabarti, A. (1998, July). Supporting two views of function in mechanical designs. In Proceedings 15th national conference on artificial intelligence, AAAI (Vol. 98, pp. 26-30).

[Chakrabarti and Bligh 2001] Chakrabarti, A., & Bligh, T. P. (2001). A scheme for functional reasoning in conceptual design. Design Studies, 22(6), 493-517.

[Chandrasekaran and Josephson 2000] Chandrasekaran, B., & Josephson, J. R. (2000). Function in device representation. Engineering with computers, 16(3-4), 162-177.

[Chandrasekaran 2005] Chandrasekaran, B. (2005). Representing function: relating functional representation and functional modeling research streams. AIE EDAM, 19(02), 65-74.

[Chittaro and Kumar 1998] Chittaro, L., & Kumar, A. N. (1998). Reasoning about function and its applications to engineering. Artificial intelligence in engineering, 12(4), 331-336.

[Chou 1991] Chou, Philip A. Optimal partitioning for classification and regression trees. IEEE Trans. on Pattern Analysis and Machine Intelligence, 13(4):340–354, April 1991.

[Cover and Thomas 1991] T. M. Cover and J. A. Thomas. Elements of Information Theory. Wiley, 1991.

[CIMdata 2010] Teamcenter “unified” “Siemens PLM Software’s Next Generation PLM Platform”, A CIMdata White Paper.

[CIMdata 2011] Digital Manufacturing - “Enabling lean for more flexible manufacturing”, A CIMdata Report.

[CISCO 2009] Introduction to eTOM. White paper.

[Cloud Foundry 2015] available online at http://www.cloudfoundry.org, accessed September 2015.

[Crilly 2013] Crilly, N. (2013). Function propagation through nested systems. Design Studies, 34(2), 216-242.

[Data Mining Group 2014] Predictive Model Markup Language V. 4.2.1.

[Davis 1987] Davis, L. (1987). Genetic algorithms and simulated annealing.

[Davis 1993] Davis, A. M. (1993). Software requirements: objects, functions, and states. Prentice-Hall, Inc.

[DeMello and Sanderson 1991] L.S.H. DeMello, A.C. Sanderson, A correct and complete algorithm for the generation of mechanical assembly sequences, IEEE Trans. Robotics and Automation 7 (2) (1991) 228–240.

[Deng 2002] Deng, Y. M. (2002). Function and behavior representation in conceptual mechanical design. AI EDAM, 16(05), 343-362.

[Diaz and Ferris 2013] Diaz, A., Ferris, C. 2013. IBM’s Open cloud architecture.

[Do and Kambhampati 2001] Do, M. B., & Kambhampati, S. (2001). Planning as constraint satisfaction: Solving the planning graph by compiling it into CSP. Artificial Intelligence, 132(2), 151-182.

[ECMA International 2013] The Javascript Object Notation (JSON) data interchange standard.1st Edition. ECMA-404.

[Efron et al. 2004] Efron, B., Hastie, T., Johnstone, I., & Tibshirani, R. (2004). Least angle regression. The Annals of statistics, 32(2), 407-499.

[European Commission 2013] Factories of the Future: Multi-annual roadmap for the contractual PPP under Horizon 2020.

[Faltings 1990] Faltings, B. (1990). Qualitative kinematics in mechanisms. Artificial Intelligence, 44(1), 89-119.

[Fikes and Nilsson 1972] Fikes, R. E., & Nilsson, N. J. (1972). STRIPS: A new approach to the application of theorem proving to problem solving. Artificial intelligence, 2(3), 189-208.

[Flouris et al. 2008] Flouris, G., Manakanatas, D., Kondylakis, H., Plexousakis, D., & Antoniou, G. (2008). Ontology change: Classification and survey. The Knowledge Engineering Review, 23(02), 117-152.

[Ford 1956] Ford Jr, L. R. (1956). Network flow theory (No. P-923). RAND CORP SANTA MONICA CA.

[Gerald et al. 2001] Gerald, B., King, N., Natchek, D. 2001. Oracle E-Business Suite Manufacturing & Supply Chain Management.

[Glinz 2007] Glinz, M. (2007, October). On non-functional requirements. In Requirements Engineering Conference, 2007. RE'07. 15th IEEE International (pp. 21-26). IEEE.

[Glover 1989] Glover, F. (1989). Tabu search-part I. ORSA Journal on computing, 1(3), 190-206.

[Gent and Walsh 1993] Gent, I. P., & Walsh, T. (1993, July). Towards an understanding of hill-climbing procedures for SAT. In AAAI (Vol. 93, pp. 28-33).

[Guyon and Elisseeff 2003] I. Guyon and A. Elisseeff, "An introduction to variable and feature selection," J. Mach. Learn. Res., vol. 3, pp. 1157–1182, 2003.

[Hashemian and Mavaddat 2005] Hashemian, S. V., & Mavaddat, F. (2005, January). A graph-based approach to web services composition. In Applications and the Internet, 2005. Proceedings. The 2005 Symposium on (pp. 183-189). IEEE.

[Hassine et al. 2006] Hassine, A. B., Matsubara, S., & Ishida, T. (2006). A constraint-based approach to horizontal web service composition. In The Semantic Web-ISWC 2006 (pp. 130-143). Springer Berlin Heidelberg.

[Hatzi et al. 2013] Hatzi, O., Vrakas, D., Bassiliades, N., Anagnostopoulos, D., & Vlahavas, I. (2013). The PORSCE II framework: Using AI planning for automated semantic web service composition. The Knowledge Engineering Review, 28(02), 137-156.

[Hayne and Ram 1990] Hayne, S., & Ram, S. (1990, February). Multi-user view integration system (MUVIS): An expert system for view integration. In Data Engineering, 1990. Proceedings. Sixth International Conference on (pp. 402-409). IEEE.

[IBM BlueMix 2015] IBM BlueMix, available online at http://www.ibm.com/cloud-computing/bluemix/, accessed September 2015.

[IEEE 1990] IEEE Standards Coordinating Committee. (1990). IEEE Standard Glossary of Software Engineering Terminology (IEEE Std 610.12-1990). Los Alamitos. CA: IEEE Computer Society.

[IEEE 1998] IEEE Computer Society. Software Engineering Standards Committee, & IEEE-SA Standards Board. (1998). IEEE Recommended Practice for Software Requirements Specifications. Institute of Electrical and Electronics Engineers.

[IFTTT 2015] IFTTT, available online at https://ifttt.com/, accessed September 2015.

[ISO/IEC 1983] ISO/IEC 19831 Cloud Infrastructure management Interface (CIMI) Model and RESTful HTTP-based Protocol – An Interface for Managing Cloud Infrastructure.

[ISO/IEC 2014] ISO/IEC 19464: 2014. Advanced Message Queuing Protocol.

Ivezic, N., Kulvatunyou, B., & Srinivasan, V. (2014). On Architecting and Composing Through-life Engineering Information Services to Enable Smart Manufacturing. Procedia CIRP, 22, 45-52.

[Jacobson et al. 1999] Jacobson, I., Booch, G., Rumbaugh, J., Rumbaugh, J., & Booch, G. (1999). The unified software development process (Vol. 1). Reading: Addison-wesley.

[Jain et al. 2000] A. Jain, R. Duin, and J. Mao, "Statistical pattern recognition: A review," IEEE Trans. Pattern Anal. Mach. Intell., vol. 22, no. 1, pp. 4–37, 2000.

[Jiménez and Torras 1996] P. Jiménez, C. Torras, Speeding up interference detection between polyhedra, in: Proc. IEEE Internat. Conference on Robotics and Automation, Minneapolis, MN, Vol. 2, 1996, pp. 1485–1492.

[Jones et al. 1998] Jones, D., Bench-Capon, T., and Visser, P. (1998), Methodologies for Ontology Development, Proceedings of the IT&KNOWS Conference of the 15th IFIP World Computer Congress, 1998.[Kautz and Selman 1992] Kautz, H. A., & Selman, B. (1992, August). Planning as Satisfiability. In ECAI (Vol. 92, pp. 359-363).

[Kautz and Selman 1999] Kautz, H., & Selman, B. (1999, June). Unifying SAT-based and graph-based planning. In IJCAI (Vol. 99, pp. 318-325).

[Kernel Based Virtual Machine 2015] Kernel Based Virtual Machine, available online at www.linux-kvm.org, accessed September 2015.

[Kwok and Weld 1996] Kwok, C. T., & Weld, D. S. (1996, August). Planning to gather information. In PROCEEDINGS OF THE NATIONAL CONFERENCE ON ARTIFICIAL INTELLIGENCE (pp. 32-39)

[Liu and Setiono 1995] H. Liu and R. Setiono. Chi2: Feature selection and discretization of numeric attributes. In J.F. Vassilopoulos, editor, Proceedings of the Seventh IEEE International Conference on Tools with Artificial Intelligence, November 5-8, 1995, pages 388{391, Herndon, Virginia, 1995. IEEE Computer Society.

[Li and Anbulagan 1997] Li, C. M., & Anbulagan, A. (1997, August). Heuristics based on unit propagation for satisfiability problems. In Proceedings of the 15th international joint conference on Artifical intelligence-Volume 1 (pp. 366-371). Morgan Kaufmann Publishers Inc..

[Lin et al. 2013] Lin, S. Y., Lin, G. T., Chao, K. M., & Lo, C. C. (2012). A cost-effective planning graph approach for large-scale Web service composition. Mathematical Problems in Engineering, 2012.

[Liu et al. 2011] Liu, F., Tong, J., Mao, J., Bohn, R., Messina, J., Badger, L. and Leaf D. 2011. NIST Cloud computing reference architecture. NIST SP 500-292.

[Mahanti and Bagchi 1985] A. Mahanti, A. Bagchi, AND/OR graph heuristic search methods, J. ACM 32 (1) (1985) 28–51.

[Mark and Lloyd 1999]Mark A. Hall and Lloyd A. Smith. Feature selection for machine learning: Comparing a correlation-based filter approach to the wrapper, 1999.

[Martelli and Montanari 1978] Martelli, A., & Montanari, U. (1978). Optimizing decision trees through heuristically guided search. Communications of the ACM, 21(12), 1025-1039.

[Martelli and Montanari 1973] Martelli, A., & Montanari, U. (1973). Additive and/or graphs. In Proceedings of the Third International Joint Conference on Artificial Intelligence (IJCAI-1973), Stanford, California.

[McGuiness et al. 2000] McGuiness, D.L., Fikes, R., Rice, J., and Wilder, S. (2000), The Chimaera Ontology Environment, Proceedings of AAAI Conference, 2000.

[Mehta et al. 1995] M. Mehta, J. Rissanen, and R. Agrawal. MDL-based decision tree pruning. InProc. of the Intl. Conf. on Knowledge Discovery and Data Mining (KDD), 1995.

[Milanovic and Malek 2004] Milanovic, N., & Malek, M. (2004). Current solutions for web service composition. IEEE Internet Computing, 8(6), 51-59.

[Moore 1959] Moore, E. F. (1959). The shortest path through a maze. Bell Telephone System.

[Murray 2011] Murray, M. 2011. Material Management with SAP ERP, 3rd Edition.

[Mylopoulos et al. 1992] Mylopoulos, J., Chung, L., & Nixon, B. (1992). Representing and using nonfunctional requirements: A process-oriented approach. Software Engineering, IEEE Transactions on, 18(6), 483-497.

[Navathe et al. 1986] Navathe, S., Elmasri, R., & Larson, J. (1986). Integrating user views in database design. Computer, 1(19), 50-62.

[Nils 2001] Nils J. Nilsson. “Artificial Intelligence: a new synthesis". Morgan Kaufmann, San Francisco, CA, USA, 2001.

[Nilsson 1971] Nilsson, N. (1971). Problem solving methods in artificial intelligence. McGraw-Hill.

[Noy and Musen 2003] Noy, N.F. and Musen, M.A. (2003), The PROMPT suite: Interactive Tools for Ontology Merging and Mapping, International Journal of Human-Computer Studies, Vol. 59(6), December 2003, pp. 983-1024, ISSN 1071-5819, 10.1016/j.ijhcs.2003.08.002.

[Noy and Klein 2004] Noy, N. F., & Klein, M. (2004). Ontology evolution: Not the same as schema evolution. Knowledge and information systems, 6(4), 428-440.

[OAGi 2014] OAGi Integration Specification Release 10.1.

[OASIS. 2013] Topology and Orchestration Specification for Cloud Applications V. 1.0.

[OASIS 2014a] Open Data Protocol Version 4.0.

[OASIS 2014b] Message Queue Telemetry Transport V. 3.1.1.

[OASIS 2014c] Cloud Application Management for Platforms V. 1.1.

[Oh et al. 2005] Oh, S. C., On, B. W., Larson, E. J., & Lee, D. (2005, March). BF*: Web services discovery and composition as graph search problem. In e-Technology, e-

Commerce and e-Service, 2005. EEE'05. Proceedings. The 2005 IEEE International Conference on (pp. 784-786). IEEE.

[Oh et al. 2008] Oh, S. C., Lee, D., & Kumara, S. R. (2008). Effective web service composition in diverse and large-scale service networks. Services Computing, IEEE Transactions on, 1(1), 15-32.

[OPC Foundation 2006] OPC Unified Architecture Specification V. 1.0.

[OpenStack 2015] OpenStack, available online at http://www.openstack.org, accessed September 2015.

[OptaPlanner 2015] OptaPlanner, available online at http://www.optaplanner.org//, accessed September 2015.

[Park and Ram 2004] Park, J., and Ram, S. (2004), Information Systems Interoperability: What Lies Beneath?, ACM Transactions on Information Systems, Vol. 22(4), pp. 595-632.

[Pearl 1984] Pearl, J. (1984). Heuristics: intelligent search strategies for computer problem solving.

[Pinto et al. 2004] Pinto, H.S., Tempich, C., and Staab, S. (2004), DILIGENT: Towards a Fine-Grained Methodology for DIstributed, Loosely-Controlled and Evolving Engineering of Ontologies, Proceedings of ECAI 2004, 393-397, López de Mantaras, R., Saitta, L., Eds, IOS Press, Amsterdam.

[Programmable Web 2015] Programmable Web, available online at http://www.programmableweb.com/, accessed September 2015.

[Quinlan 1993] J. R. Quinlan. C4.5: Programs for Machine Learning. Morgan Kaufmann, 1993.

[Rada et al. 1989] R. Rada, H. Mili, E. Bicknell, and M. Blettner, “Development And Application Of A Metric On Semantic Nets,” IEEE Transaction on Systems, Man, and Cybernetics, 19 (1), pp. 17-30, 1989.

[Resnik 1995] P. Resnik, “Using Information Content To Evaluate Semantic Similarity In A Taxonomy,” In Proceedings of the 14th International Joint Conference on Artificial Intelligence, pp. 448-453, 1995.

[Russell and Norvig 2002] S.J. Russell and P. Norvig. “Artificial Intelligence: a modern approach". Prentice-Hall, Englewood Cli® s, NJ, USA, 2002.

[Selman et al. 1994] Selman, B., Kautz, H. A., & Cohen, B. (1994, October). Noise strategies for improving local search. In AAAI (Vol. 94, pp. 337-343).

[Sesame 2015] Sesame, available online at http:// rdf4j.org, accessed September 2015.

[Sheth and Kashyap 1992] Sheth, A. P., and Kashyap, V. (1992), So far (schematically), yet so near (semantically), Proceedings of the IFIP WG2.6 Database Semantics Conference on Interoperable Database Systems (DS-5, Lorne, Victoria, Australia, Nov. 16–20), Hsiao, D. K., Neuhold, E. J., and Sacks-Davis, R., Eds., 283–312.

[Shvaiko and Euzenat 2011] Shvaiko, P., Euzenat, J. (2011), Ontology matching: state of the art and future challenges, IEEE Transactions on Knowledge and Data Engineering, 99.

[Sommerville 2004] Sommerville, I. (2004). Software Engineering. International computer science series.

[Soundex 2015] Soundex, available online at http://www.archives.gov/research/census/soundex.html, accessed September 2015.

[Staab et al. 2001] Staab, S., Schnurr, H.P., Studer, R., and Sure, Y. (2001), Knowledge Processes and Ontologies, IEEE Intelligent Systems, 16(1), 26–34.

[Supply Chain Council 2012] Supply Chain Operation Reference Model Revision 11.0.

[Suzanne and James 1999] Suzanne, R., & James, R. (1999). Mastering the requirements process.

[Tan et al. 2010] Tan, W., Missier, P., Foster, I., Madduri, R., and Goble, C. 2010. A Comparison of Using Taverna and BPEL in Building Scientific Workflows: the case of caGrid. Concurrent Computing, 22(19), 1098-111

[The R Project 2015] The R Project for Statistical Computing, available online at http://www.r-project.org, accessed September 2015.

[Vapnik 1998] V. Vapnik. Statistical Learning Theory. Wiley, New York, NY, 1998.

[VDE Association for electrical, electronic & information technologies 2014] The German standardization roadmap, Industrie 4.0. Version 1.0.

[VirtualBox 2015] VirtualBox, available online at http://www.virtualbox.org, accessed September 2015.

[Vossen et al. 2000] Vossen, T., Ball, M., Lotem, A., & Nau, D. (2000). Applying integer programming to AI planning. The Knowledge Engineering Review, 15(01), 85-100.

[Vujasinovic et al. 2013] Vujasinovic, M., Ivezic, N., and Kulvatunyou, B. (2013), Towards Ontology Design Patterns for Domain Specific Ontology: A Manufacturing Service Capability Information Perspective, submitted to International Journal of Computer Integrated Manufacturing for publication.

[W3C 2001] W3C - World Wide Web Consortium (2001), Web Services Description Language (WSDL) 1.1, March 15 2001, available online at http://www.w3.org/TR/wsdl

[W3C 2002] W3C – World Wide Web Consortium (2002), Web Services, available online www.w3.org/2002/ws/Activity

[W3C 2004a] W3C - World Wide Web Consortium (2004b), Resource Description Framework (RDF): Concepts and Abstract Syntax, February 10, 2004, available online at http://www.w3.org/TR/2004/REC-rdf-concepts-20040210/.

[W3C 2004b] W3C – World Wide Web Consortium (2004), OWL-S: Semantic Markup for Web Services, November 22, 2004, available online at http://www.w3.org/Submission/2004/SUBM-OWL-S-20041122/

[W3C 2005] W3C – World Wide Web Consortium (2005), Web Service Modeling Ontology (WSMO), June 3, 2005, available online http://www.w3.org/Submission/WSMO/

[W3C 2006] W3C – World Wide Web Consortium (2006), Extensible Markup Language (XML) 1.1, August 16, 2006, available online at http://www.w3.org/TR/xml11/

[W3C 2007] W3C – World Wide Web Consortium (2007), Semantic Annotations for WSDL and XML Schema, August 28, 2007, available online at http://www.w3.org/TR/sawsdl/

[Wiegers 2003] Wiegers, K. E. (2003). Software Requirements: Practical techniques for gathering and managing requirement through the product development cycle. Microsoft Corporation.

[Yan et al. 2012] Yan, Y., Chen, M., & Yang, Y. (2012, March). Anytime QoS optimization over the PlanGraph for web service composition. In Proceedings of the 27th Annual ACM Symposium on Applied Computing (pp. 1968-1975). ACM.

[Younus, Muhammad et al. 2010] "MES development and significant applications in manufacturing-A review." Education Technology and Computer (ICETC), 2010 2nd International Conference on. Vol. 5. IEEE, (2010).

[Zapier 2015] Zapier, available online at https://zapier.com/, accessed September 2015.

[Zhang et al. 2003] Zhang, R., Arpinar, I. B., & Aleman-Meza, B. (2003, June). Automatic Composition of Semantic Web Services. In ICWS (Vol. 3, pp. 38-41).

[Zeng et al. 2003] Zeng, L., Benatallah, B., Dumas, M., Kalagnanam, J., & Sheng, Q. Z. (2003, May). Quality driven web services composition. In Proceedings of the 12th international conference on World Wide Web (pp. 411-421). ACM.