Towards a resilient service oriented computing based on ad-hoc web service compositions in dynamic environments

Wenbin Li

N° d'ordre 2014ISAL0032
Année 2014

Thèse

Towards a Resilient Service-Oriented Computing based on Ad-hoc Web Service Compositions in Dynamic Environments

Présentée devant
L'institut national des sciences appliquées de Lyon

Pour obtenir
Le grade de docteur

Formation doctorale : Informatique

École doctorale : École Doctorale Informatique et Mathématiques

Par
Wenbin LI
(Maîtrise en Informatique)

Soutenue le 27 Mars 2014 devant la Commission d'Examen

Jury

CAUVET Corine	Professeur, Université d'Aix-Marseille	Raporteur
FRONT Agnès	Maître de Conférences, HDR, Université Pierre Mendès	Raporteur
MALEFANT Jacques	Professeur, Université Pierre et Marie Curie	Président
EXPOSITO Ernesto	Maître de Conférences, HDR, INSA de Toulouse	Examinateur
BADR Youakim	Maître de Conférences, HDR, INSA de Lyon	Directeur de Thèse
BIENNIER Frédérique	Professeur, INSA de Lyon	Co-directrice de Thèse

Laboratoire de recherche: Laboratoire d'Informatique en Image et Systèmes d'information (LIRIS)

Abstract

The Service-Oriented Computing (SOC) promotes assembling software components into loosely-coupled networks of services, to create flexible, agile applications and business processes that span organizations and computing platforms. Due to the distributed and asynchronous nature of Web services, the Web service composition process plays an important role in achieving SOC. In dynamic environments by which contextual information such as Web service properties and composition requirements often change, the composition process is thus affected and, consequently, should be able to adapt composite applications to changes at design time and runtime. Unfortunately, current Service-Oriented Architecture (SOA) and Web service composition approaches lack of the ability to deal with continuous and unpredictable changes. Building resilient service-oriented architectures that are adaptable to endogenous and exogenous changes in dynamic environments reveal a drastic challenge to current composition processes. In addition, current composition processes provide a limited support for business users to specify their requirements in business languages to automatically compose business processes (i.e., composite services). By such, the gap between business requirements and composition requirements related to Web services increases the complexity of developing adaptable SOA-based applications and processes in dynamic environments.

To overcome these challenges, we introduce the concept of Resilient Service-Oriented Computing (rSOC) to construct resilient SOA-based applications driven by business requirements in dynamic environments. To this end, the resilient SOA is defined as a set of models that affect and are affected by each other, and relies on a model-to-model transformation approach to ensure SOA adaptability and evolution. In this thesis, we particularly focus on two models: *a three-level composition requirement model* and *a Web service composition model*, to establish the foundation for a resilient SOA as follows: firstly, composition requirements are modeled in three levels, i.e., *business-centric, capability-focused* and *rule-driven*. Particularly the *business-centric requirement model* provides business users with a structured natural language to specify requirements; secondly, a two-phase requirement transformation process builds the *rule-driven Web service composition requirement model* from the business-centric requirement model as set of composition rules, expressing multi-objective constraints that affect the composition process and its dynamic environment; thirdly, *an ad-hoc Web service composition approach* is introduced to flexibly construct composite services without predefined composition plans. Particularly, composition rules generated in composition process may affect other model(s) in the resilient SOA, such as composition requirement model, and recursively invoke the model-to-model transformation approach to replan the ad-hoc Web service composition approach.

Keywords: Web service – service-oriented computing – service-oriented architecture – service composition – composition requirement – service capability – resilient – ad-hoc – dynamic environment

Résumé

L'informatique orientée services (SOC) favorise l'assemblage de composants logiciels indépendants pour créer des applications et processus métier flexibles et agiles. Le processus de composition des services Web joue un rôle important dans la réalisation des architectures orientées services (SOA). Dans les environnements dynamiques dans lesquels des informations contextuelles changent souvent, le processus de composition est souvent affecté pendant les phases de conception et d'exécution. Ce processus devrait par conséquence être en mesure de s'adapter aux changements en temps de conception et exécution. A présent, les architectures orientées services et les mécanismes automatiques de composition de services Web ne parviennent pas à faire face aux changements continus et imprévisibles. Construire des architectures orientées services qui s'adaptent aux changements dans des environnements dynamiques révèle un défi pour les processus de composition de services Web. En outre, les processus de composition actuelles offrent un support limité pour les utilisateurs professionnels de spécifier leurs exigences métier afin générer automatiquement les processus métiers (services Web composites). Par cela, l'écart entre les exigences fonctionnelles et non-fonctionnelles au niveau métier et les exigences techniques liées aux mécanismes de composition de services Web augmentent la complexité du développement d'applications ou de processus métier adaptés aux environnements dynamiques.

Pour remédier à ces défis, nous introduisons le concept de résilience appliqué à l'informatique orientée services (nommé SOC résilient) afin de construire des applications et processus métier dynamiques en respectant les exigences métier dans des environnements dynamiques. La SOA résilient est conçue comme un ensemble de modèles qui affectent, et sont affectées par, d'autres modèles. Dans cette thèse, nos contributions, qui se concentrent en particulier sur *le modèle d'exigence de composition* et *le modèle de composition des services Web*, composent trois parties principales: tout d'abord le modèle de d'exigence est modélisé aux trois niveaux, i.e., métier, capacité, et règle. Particulièrement, le modèle de d'exigence métier offre aux utilisateurs un langage structuré à base de langage naturelle pour spécifier les processus métier; d'autre part, un processus de transformation dérive par transformation un modèle d'exigence de composition à base de règles. Chaque règle représente un ensemble de contraintes multi-objectives concernant différentes variables liées au processus de composition et à son environnement dynamique. Troisièmement, une approche ad-hoc de composition des services Web a été développée pour construire de services composites sans plans de composition prédéfinis dans des environnements dynamiques. L'approche de transformation de modèle-à-modèle génère les règles qui sont récursivement utilisées pour modifier ces modèles et ensuite replanifiés une composition ad-hoc de services Web.

Mots-Clés: service Web – informatique orientée services – architecture orientée services – composition des services – exigence de la composition – capacité de service – résilient – ad-hoc – environnement dynamique

Acknowledgements

First of all, I would like thank my supervisor Professor Youakim Badr and my co-supervisor Professor Frédérique Biennier for the invaluable discussions and advice throughout the course of this research, without which this dissertation would not have been achieved. Working with them during the three-and-a-half years was an exciting adventure.

I own my thanks to Professors Corine Cauvet and Agnès Front for spending their time in reviewing my dissertation and giving me the constructive suggestions.

I am grateful to Professors Jacques Malefant and Ernesto Exposito for their participation in this jury.

I would also like to thank my colleges at INSA Lyon, Francis Ouedraogo, Yong Peng, Ziyi Su, and Wei Zuo, for the pleasant time that we passed together.

In particular, I express my sincere thanks to my parents. Their unwavering support and encouragement have brought me to the completion of this dissertation. You raise me up, to more than I can be.

Finally, special thanks to all the people whose names are not listed here for helping me with this dissertation: to the ones who know they did, and to the ones who do not.

Table of Contents

Abstract ..... *i*

Résumé ..... *ii*

Acknowledgements ..... *iii*

Chapter 1 General Introduction *1*

1.1 Background 1

1.2 Research Challenges 6

1.3 Research Contributions 8

1.4 Motivation Scenario 14

1.5 Thesis Outline 18

Chapter 2 State of the Art *21*

2.1 Service-oriented Computing Preliminaries 21

2.2 A Glance at Web Service Composition Approaches 29

2.3 Overview of Requirement Specifications in Web Service Composition 59

2.4 Managing Requirements 64

2.5 Executive Summary 69

Chapter 3 A Three-level Requirement Model for Ad-hoc Web Service Compositions *73*

3.1 Introduction 74

3.2 Related Work 76

3.3 General Process for Crisis Management 78

3.4 Three Levels of Composition Requirements 79

3.5 The Business-centric Requirement Model 82

3.6 The Capability-focused Requirement Model 88

3.7 The Rule-driven Web Service Composition Model 92

3.8 Conclusion 100

Chapter 4 Two-level Requirement Transformations *103*

4.1 Introduction 104

4.2 The Requirement Transformation Framework 105

4.3 The Capability Matching Process 109

4.4 The Association Discovery Process 117
4.5 Conclusion..... 123
Chapter 5 Service Farming: An Ad-hoc Web Service Composition Approach 125
5.1 Introduction 126
5.2 The Service Model 129
5.3 Service Farming Composition Algorithm 132
5.4 The Dynamic Reconfiguration of Composite Web Services 145
5.5 Conclusion..... 151
Chapter 6 Implementation Architecture 153
6.1 Implementation Architecture 153
6.2 Experiment Results 157
6.3 Conclusion..... 162
Chapter 7 Research Perspectives..... 165
7.1 Research Contributions 165
7.2 Future Research Trends..... 168

Table of Figures

Figure 1.1 A General View of the Resilient SOC Paradigm 10
Figure 1.2 The Thesis Organization 20
Figure 2.1 SOA with Web Services 23
Figure 2.2 WSDL Document Representation 26
Figure 2.3 SOAP Message Structure 27
Figure 2.4 OWL-S Structure 28
Figure 2.5 ESB Architecture Pattern 29
Figure 2.6 A Conceptual Model of Automated Web Service Composition 33
Figure 2.7 Requirement Engineering Process 60
Figure 3.1 The Conceptual rSOC with Requirement Model 75
Figure 3.2 A General Crisis Management Process 78
Figure 3.3 The UML class diagrams of the Business-centric Requirement Model 85
Figure 3.4 The UML Class Diagram of the Capability-focused Model 90
Figure 3.5 Composite Services as Binary Trees 95
Figure 3.6 Structure Rules as Binary Trees 97
Figure 4.1 General Framework of Composition Requirements Transformation 106
Figure 4.2 Concept Matching between Different Requirement Models 108
Figure 4.3 The Capability Matching Process (Part 1) 115
Figure 4.4 The Capability Matching Process (Part 2) 116
Figure 4.5 The Association Discovery Process (Part 1) 122
Figure 4.6 The Association Discovery Process (Part 2) 123
Figure 5.1 The Conceptual rSOC with Service Farming 128
Figure 5.2 The Service Farming Flowchart 133
Figure 5.3 The Service Planting Stage 134
Figure 5.4 The Service Growing Stage 135
Figure 5.5 The Algorithm of Constructing $c_i \overset{m}{=} r_{s.p}$ 136
Figure 5.6 Service Growing: Constructing Candidate Composite Services 137
Figure 5.7 Service Growing: Constructing Entire Composite Services 138
Figure 5.8 The Service Harvesting Stage 139
Figure 5.9 The Utility Calculation 141
Figure 5.10 K-means Clustering Example 142
Figure 5.11 The Structure Rule Enrichment 143
Figure 5.12 The Service Evaluating Stage 144
Figure 5.13 The Service Substitution Algorithm 147
Figure 5.14 Service Substitution Example 148
Figure 5.15 The Web Service Composition Replanning 150
Figure 5.16 Composition Replanning Example 151
Figure 6.1 The Resilient SOA Framework 154
Figure 6.2 General Implementation of Service Framing Manager 155

Figure 6.3 Class Diagram of Service Farming Manager 156
Figure 6.4 Optimal Utility Increase 160
Figure 6.5 Average Time for Each Cycle by Varying k 161
Figure 6.6 Average Time for Each Cycle by Varying T 161
Figure 6.7 Scalability Evaluation 162

Table of Tables

Table 2.1 Comparison of Automated Composition Approaches 40
Table 2.2 A Global View on Web service Composition Approaches 58
Table 3.1 Major Features of SBVR Meta-model 84
Table 3.2 Business-Centric Requirement Examples 87
Table 3.3 Domain for a Dynamic Environment 88
Table 3.4 Web Service Capability Instances based Capability Model 93
Table 3.5 The Rule-driven Web Service Requirement 100
Table 4.1 Notation and Operator Legends 109
Table 4.2 Matching Relation of Capability Matching Principle 1 110
Table 4.3 Capability Matching Principle 1 Example 111
Table 4.4 Matching Relation of Capability Matching Principle 2 111
Table 4.5 Capability Matching Principle 2 Example 111
Table 4.6 Matching Relation of Capability Matching Principle 3 112
Table 4.7 Capability Matching Principle 3 Example 112
Table 4.8 Matching Relation of Capability Matching Principle 4 113
Table 4.9 Capability Matching Principle 4 Example 113
Table 4.10 Matching Relation of Capability Matching Principle 5 114
Table 4.11 Capability Matching Principle 5 Example 114
Table 4.12 Matching Relation of Capability Matching Principle 6 115
Table 4.13 Association Discovery Principle 1 Example 118
Table 4.14 Association Discovery Principle 2 Example 119
Table 4.15 Matching Relation of Association Discovery Principle 3 119
Table 4.16 Association Discovery Principle 3 Example 120
Table 4.17 Matching Relation of Association Discovery Principle 4 120
Table 4.18 Association Discovery Principle 4 Example 121
Table 4.19 Association Discovery Principle 5 Example 122
Table 5.1 Summary of the Set of Web Services and the Set of Composition Rules 129
Table 5.2 QoS Aggregation Examples 131
Table 5.3 Utility Calculation Example 141
Table 5.4 Clustering Example 143
Table 5.5 Structure Rule Enrichment Example 144
Table 5.6 Service Evaluating Example 145
Table 5.7 Calculation of QoS Constraints in Run Time 150
Table 6.1 Web Service Capability Instances 157
Table 6.2 Business-centric Requirement 158
Table 6.3 Service Farmimng Running Example 159
Table 6.4 The Service Farming Performance based on Different Cases 159

Chapter 1

General Introduction

Background 1
Research Challenges 6
Research Contributions 8
Motivation Scenario 14
Thesis Outline 18

1.1 Background

Service-Oriented Computing promotes assembling application components into a loosely coupled network of services, to create flexible, dynamic business processes and agile applications that span organizations and computing platforms [PAPA03]. This is achieved through a service-oriented architecture (SOA), which is an architectural style that guides all aspects of creating and using business processes throughout their life-cycle, packaged as services. SOA defines and provides guidelines that allow different applications to exchange data and participate in building business processes, independent from operating systems and programming languages, underlying applications [BRWI09]. Based on the SOA, Web services have became the most popular technology to support interoperable machine-to-machine interaction and achieve service-oriented computing over distributed Internet due to its distributed and asynchronous nature.

According to W3C definition [FERR04], a Web service is "*a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL). Other systems interact with the Web service in a manner prescribed by its description using SOAP messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards.*".

Due to their distributed and asynchronous nature, Web services can be easily reused since they are loosely-coupled and platform-independent software components. The Web Service Composition (WSC) refers to the process of reusing existing atomic services and logically re-combining them into composite services. The WSC provides an open and standards-based approach for connecting Web services together in order to build business processes. To this end, Web service standards are thus designed to reduce the complexity of composing Web services, hence reducing time and costs, and increasing overall efficiency [KHAL03][WEER05].

Generally, Web service composition approaches can be classified into two categories: static Web service composition approaches and dynamic Web service composition approaches, concerning the design time and runtime when Web services are composed [FOST04]. Static Web service composition approaches mainly construct composite Web services at the design time and then execute composite Web services (i.e., business processes) at runtime, whereas dynamic Web service composition approaches compose and execute Web services at runtime [FOST04]. Accordingly, composition environments in which Web services are composed and executed, can also be divided into two categories, i.e., static composition environments and dynamic composition environments [MBKG09], according to contextual information, which may influence the Web service composition process and / or the execution of composite Web services. Contextual information refers to any information, events, variations or changes that may occur during design and runtime, such as the availability of Web services, the composition requirements, changes in Quality of Service, known as QoS, (e.g., price, reputation, etc), degradation in system recourses (CPUs, memory storages, and bandwidth), just to mention a few. In static environments, contextual information rarely change whereas they seriously impact the composition process at design time and during the runtime in dynamic environments since contextual information continuously changes over time. In this context, contextual information may be generated by new Web services that are added to service provider registries, unavailability of Web services in composite services due to maintenance or overload reasons, or changes in QoS attributes related to Web services.

In practice, many business domains or cases exist and assume that the composition environment is a static, which shows their limits and drawbacks regarding their adaptation and evolution. In fact, information related to Web services may change while composite Web services are designed or executed. By such, contextual information changes may invalidate predefined Web service compositions. Dynamic Web service composition approaches deal dynamic environments and should be able to

adapt the Web service composition process and the Web service execution to environment changes and seek to minimize user interventions in order to provide most appropriate composite services and satisfy user's requirements. As a matter of fact, the SOA fails to support adaptable applications and adaptable business processes in dynamic environments since it is unabile to deal with continuous and often unpredictable changes [RONA09] through adaptable and dynamic Web service compositions. Most of current dynamic Web service composition approaches require predefined general composition plans to construct composite Web services [HAAM08]; the general composition plan indicates the execution order of different actions (usually represented as abstract Web services). The composition plan is mainly defined by analyzing Web service functional properties, matching their inputs and outputs' messages. Dynamic Web service composition approaches usually discover and select concrete Web services for each action in the composition plan and then integrate them following the composition plan. Based on the composition plan, appropriate Web services are selected with respect to their non-functional properties to construct the composite Web service. The late binding term is often used to indicate the selection of Web services based on their non-functional properties at runtime and before invoking constituent Web services [CHMT10]. However, we highlight two shortcomings in current dynamic Web service composition approaches [HAAM08]: firstly, when constructing a composite Web service, a predefined composition plan without considering Web service non-functional properties may not provide the best way to compose services with regard to QoS properties. As a result, the composite service constructed upon the predefined composition plan may not provide a global optimal QoS value in the composite Web service since the composition plan does not provide a global optimal composition solution; secondly, these approaches cannot deal with some business cases where the composition plan can only be partially defined or cannot be defined in advance. The crisis management is an example of business applications where crises often occur abruptly, so the exact actions to respond to a crisis cannot be precisely identified before a crisis occurs and contextual information cannot be identified in advance and collected from the crisis environment (e.g., witnesses, police reports, weather conditions, etc.). In this context, building the crisis management response process based on Web services to support complex crisis situations require a particular Web service composition approach without predefined composition plans in dynamic environments. Moreover, rebuilding a crisis management process when new contextual information are collected from witnesses, for example, requires that the Web service composition process should be updated if necessary. In such dynamic environment, de-

veloping ad-hoc Web service composition processes without predefined composition plans and adapting them to contextual changes reveal a particular challenge, as extending the SOA and current Web service composition approaches to achieve an "resilient" service-oriented computing in which the SOA adapts and evolves in response to internal and external changes in dynamic environments.

In order to refer to the challenge, we introduce the resilient service-oriented computing (rSOC) concept as "*a service oriented computing paradigm that flexibly constructs adaptable SOA-based applications and adaptable business processes without predefined composition plans dealing with continuous and unpredictable environment changes*". To this end, the resilient service-oriented computing should establish foundations to ensure SOA adaptability and its evolution in dynamic environments.

In our view, an adaptable Web service composition process is the keystone to establish the resilient service computing. The ultimate objective of the Web service composition is to construct composite services to satisfy user's requirements, which does not only include functional requirements, but also non-functional requirements as well. Functional requirements describe "*what a composite service should do*" and define the functionalities of the composite service to be constructed; whereas non-functional requirements describe "*how a composite service should do*" and specify the expected criteria on composite service's attributes and all other requirements that are excluded from functional requirements. Non-functional properties of a single Web service or a composite Web service is modeled as Quality of Service (QoS), which may include properties such as performance, price, reputation, reliability, scalability, capacity, robustness, exception handling, accuracy, integrity, accessibility, availability, interoperability, security, and network-related attributes, etc [ZBDK03]. Nowadays, more and more Web services are published by different providers with same functionalities and different QoS values (e.g., price, reputation, response time, etc.). Web service composition approaches thus should be able to compose Web services satisfying particularly business requirements while maximizing business user's satisfaction by providing composite services with better global QoS properties aggregated from atomic Web service QoS properties. Constructing composite Web services and meeting requirements imposed by users on both functional and non-functional properties is proved to be a NP-hard problem [MBKG09] and AI-complete problem [OHLK06].

Yet another disadvantage of Web service composition approaches is that users should be familiar with technical backgrounds in order to compose Web services (e.g., Workflow modeling, composition languages and

techniques, etc.). Most of all, existing Web service composition approaches highly rely on the fact that users' composition requirements formally describe necessary actions and constraints on functional and non-functional properties to discover the set of relevant Web services: on one hand, existing Web service composition approaches often rely on different formalisms to specify requirements on Web service properties and thus require competencies related to Web service standards (e.g., XML, UDDI, SOAP, WSDL, OWL, etc.). For examples, Web service composition approaches based on the Artificial Intelligence (AI) [LIZD10] require that users define goals and rules in terms of formalized domain states, whereas Web service composition approaches based on workflows [SFKM11] require users' to specify control flows by means of logic operators and data flows. On the other hand, existing requirement specification languages which are independent from any specific Web service composition approaches (e.g., Business Process Execution Language for Web Services (BPEL4WS) [MMVL05] or Business Process Model and Notation (BPMN) [SKTB12]) remain too technical for business users. By such, Web service composition approaches still focus on the technical level as they require domain-specific knowledge on Web services in to specify requirements. As a result, they fail to provide business users with natural language-based notations or means to express requirements at the business level and describe business objectives to be achieved for discovering and composing Web services together.

In addition to adaptable Web service composition processes in dynamic environments, a resilient service oriented computing should be driven by business requirements in order to establish useful foundations upon which adaptable SOA-based applications and adaptable business processes can be build to deal with real world applications.

For the before-mentioned reasons, our research strategy has particularly focuses on the Web service composition process and business requirements to achieve a resilient service-oriented computing. By such, constructing resilient processes from Web services driven by business requirements and adaptable to contextual changes in dynamic environments provide the following benefits:

1) Allow business users to specify business requirements from which both Web service functional and non-functional composition requirements can be deduced prior to the Web service composition process;

2) Construct composite services to satisfy business users' initialized requirements and maximize global composite Web service QoS values without predefined composition plans;

3) Build resilient SOA-based applications that are adaptive to internal and external environmental changes at design time and runtime in dynamic environments.

1.2 Research Challenges

The lack of support in the SOA for contextual changes and business users requires extending the SOA with requirement model at the business level and adaptable Web service composition approaches to deal with contextual changes in dynamic environment. In such context, we summarize our main research question as:

"*How to achieve a resilient service-oriented computing based on an ad-hoc Web service composition process driven by business requirements in order to build adaptable composite Web services without predefined composition plans?*

To the end, how to adapt composite Web services accordingly to endogenous and exogenous changes that may occur within and outside the dynamic composition environment at design time and runtime?"

This research question is challenging, as business users are not familiar with Web service technical details, and paradoxically, the resilient SOC and Web service composition approaches are expected to deal with business requirements and automatically compose Web services together. During the Web service composition process, Web services are expected to be composed without predefined composition plans; when several possible composition solutions exist at the same time, Web service composition approach should also provide the optimal solution to maximize business users' satisfaction. Finally composite Web services are expected to be automatically adapted in response to changes in dynamic environments, when contextual information changes.

To answer the main research question, we identify a research strategy to answer three interrelated sub-questions as follows.

1) How to specify business requirements based on business languages, such as structured English based languages, and to which extent business users can specify adaptable Web service composition with business languages?

In order to provide users with a business language-based specification to build adaptable business processes and consequently specify composition requirements, the main question is how to model composition requirements from a business perspective to meaningfully capture different categories of business requirements at different abstraction levels. Since the Web service composition process is driven by composition constraints on

Web service properties and automatically manipulated by machines, the challenge is to keep a balance between machine accessibility (e.g., technology support, machine readable, …) and human understandability (e.g., expressiveness of goals, human readable, …).

Deriving composition constraints from natural languages (e.g., English, French, etc.) without semantical and syntactical restrictions leads to misunderstandings and ambiguity [SOMM07]; Conversely, technical details related to Web services should be avoided when specifying business requirements in order to ensure that business users can directly express their requirements without technical concerns and background.

In dynamic composition environments, contextual information related to the composition process change overtime. Changes in the environment do not only influence Web service composition processes but also require flexible mechanisms to adapt current composite Web service to changes. In the case that composition requirements change at design time and/or at runtime, Web services should be discovered based on updated requirements. This leads to causal relationships among business requirements, composition requirements, and contextual changes at design time and runtime.

2) How to bridge the gap between business requirements expressed by business languages at the business level and composition requirements manipulated by Web service composition approaches at the technical level?

As mentioned before, current composition approaches still focus on the technical level and require domain-specific knowledge on Web services to specify composition requirements; in order to provide business users with business languages such as natural based languages to specify their business objectives, composition requirements should be modeled with respect to natural language-based specifications and then should be connected with composition requirements manipulated by Web service composition approaches. This leads to a gap between business requirements based on natural languages describing business objectives, and composition requirements at the technical level describing Web service properties and other technical details.

Bridging the gap between requirements at the business level and the Web service level is challenging due to two reasons: requirements at the business level and requirements at the technical level are defined in different languages and scopes, thus a requirement transformation process is required to derive from business requirements to technical requirements to be used by an adaptable Web service composition process; secondly, business and Web service requirements are defined from two different perspectives;

business requirements are specified from a "request" perspective, which describes what business users seek as business objectives, while technical-level requirements are defined from an "offer" perspective, which describes what Web services can provide (i.e., functionalities, utilities, etc…). Roughly speaking, the transformation from a business model, consisting of requirements from a request perspective to a Web service composition model, consisting of technical requirements from an offering perspective, leads to a business model to Web service composition model transformation.

3) *How to construct ad-hoc composite Web services without composition plans in dynamic environments satisfying multiple constraints while maximizing global QoS property values?*

As mentioned in the previous paragraph, different constraints should be also taken into account alongside business requirements when composing Web services in dynamic environments. For examples, constraints, which are issued from different perspectives and reflecting endogenous and exogenous changes, may include control-flow constraints to describe expected execution orders between/among different Web services, QoS constraints to describe expected values on Web service QoS attributes, dependency constraints to describe implicit dependency relationships (e.g., Web service substitution, Web service recommendations, etc.) between Web services and so on. Constraints in dynamic environments also include the availability of Web services, the system recourses (e.g., CPUs, memory stores, and bandwidth), etc. However, constructing composite services while meeting multiple constraints is proved to be a NP-hard problem [MBKG09]. Dealing with multiple constraints also makes the Web service composition process difficult to be specified in advanced and leads to an ad-hoc composition process that composes *on-the-fly* Web services without predefined composition plans. The absence of an explicit composition plan offers the advantage of finding optimal global QoS of the composite services that maximize business users' satisfaction and adapt composite Web services to different contextual changes.

1.3 Research Contributions

The foremost objective of our research contributions is to provide foundations to build a resilient service-oriented paradigm (rSOC) to flexibly construct resilient SOA-based applications based on an ad-hoc Web service composition process driven by business requirements in dynamic environments. A general view of the rSOC is presented in Figure 1.1. The main

idea behind the rSOC is to consider the SOA as a set of models and then applying the General System Theory [SKYT05] to establish causal dependencies between models. From a system perspective, each model is a fine-grained element since it focuses on specific changes (e.g., QoS, security, fault tolerance, etc). Each model also affects and is affected by sibling models through cybernetic loops such as feedback loops and control loops. In our research activities, we particularly develop two models: the *ad-hoc Web service composition model*, which is a central pillar in the rSOC and plays a central role by ensuing the adaptability of the SOA through on-the-fly Web service compositions, and the *three-level requirement model*, which is also another pillar in the rSOC by extending the focus of the SOA to businesses level from Web service technical requirements. All endogenous and exogenous changes in the dynamic environment that occur during design time and runtime can also be captured as models, consisting of variables to observe and control with respect to multiple constraints. As a result, the rSOC can be viewed as a closed system of models, and each of the models depends on and affects each other. Since models are considered fine-grained elements, the cybernetic loops show dependencies between them and assume that they are static. However, the requirement model and the ad-hoc composition model are particularly dynamic and evolve in response to changes (e.g., new business logic or recompose Web services). The business requirement model mainly guides the generation of the ad-hoc Web service composition model and, consequently, controls the ad-hoc composition process. Reciprocally, the ad-hoc Web service composition model may affect the requirement model during the composition process by relying on sibling endogenous/exogenous models (e.g., fault tolerance model or security model) to respectively propose new functionalities that might interest the business users or suggest non-functional requirements (e.g., business security objectives). In order to ensure the evolution of models when they affect each other, the rSOC follows the dynamic data-driven application system concept (DDDAS), which unifies applications and measurement processes[DARE09]. The DDDAS concept entails " *[…] the ability to incorporate dynamically data into an executing application simulation, and in reverse, the ability of applications to dynamically steer measurement processes*"[DARE05]. Similarly, the rSOC entails the ability of the three-level requirement model to dynamically steer the ad-hoc composition process through Web service composition model, and in the reverse, the ability of the Web service composition model to dynamically incorporate requirements into the requirement model. In order to ensure these abilities, rSOC relies on model-to-model transformations.

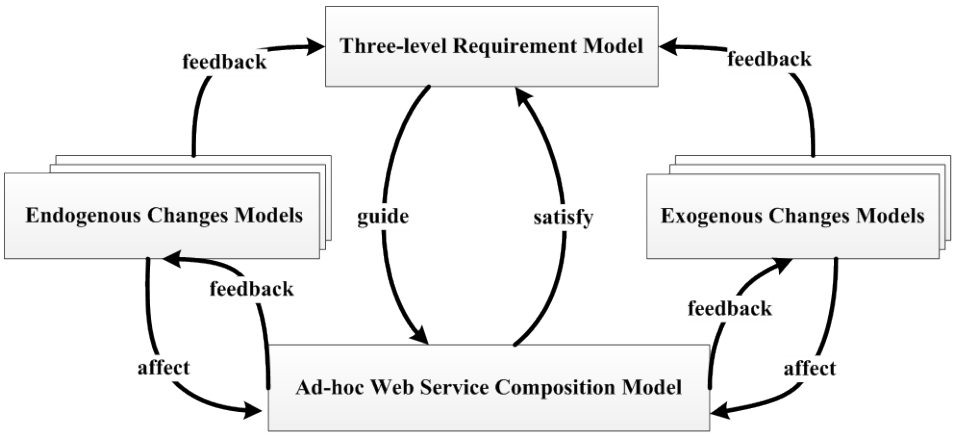

Figure 1.1 A General View of the Resilient SOC Paradigm

Based on the rSOC paradigm in Figure 1.1, we propose a resilient Service-Oriented Architecture (rSOA) consisting of four parts:

1- A three-level requirement model to support composition requirement specifications based on either structured natural languages at the business level or Web service composition languages at the technical level.

2- An ad-hoc Web service composition approach to compose Web services without predefined composition plans while responding to contextual information.

3- Endogenous changes models to represent contextual changes that occur within the composition environments at design time and runtime.

4- Exogenous changes models to represent contextual changes that occur outside the composition environments at design time and runtime.

Based on rSOA, firstly business users are expected to specify the business requirements following a business requirement model to guide the ad-hoc Web service composition approach, which takes into account all constraints derived from the business requirements; when contextual changes occur, either the Web service composition approach automatically deals with changes without users' interventions, or business requirements are updated by business users to guide the Web service composition process in response to endogenous and exogenous changes, which can be represented as models.

The endogenous changes may include:

1) *Available Web services*. The set of available Web services may evolve as new Web services emerge from time to time and old Web services are replaced, or existing Web services temporarily disconnected due to maintenance or overload.
2) *Web Service Properties*. Web Service functional and/or non-functional properties changes mainly focus on QoS attribute values such as the price of invoking a Web service is aug-

mented or Web service functionalities such as a Web service does not satisfy anymore user requirements.

3) *System Recourses*. System recourses supporting successful implementation of the composition process concerns CPUs, memory storages, bandwidth, and etc.

The exogenous changes may include:

1) *Web Service Composition Requirements*. Expected business requirements can change during the composition process since users simply express new needs or provide additional requirements to tune the service composition.
2) *Business Logic*. When the business logic changes, the previous composition process may become no longer valid since it does not respect the new business logic.
3) *Security Concerns*. Security attacks from outside of the composition environment will also influence or invalidate the composition process. In addition, security vulnerabilities can also be considered as endogenous changes.

In this thesis, our contributions focus on the three-level requirement model, the ad-hoc Web service composition model and their model-to-model transformation process prior to ad-hoc Web service compositions. We are particularly interested in how to model requirements for ad-hoc Web service compositions at different levels, how to transform business requirements into Web service composition requirements, and how to flexibly construct composite service to satisfy requirements in dynamic environments. It is worth noting that we do not cover formal presentations for endogenous and exogenous changes models but we advocate that these models can affect the Web service composition processes through multiple constraints (e.g., substitution constrain rules, security constrain rules, etc.).

In order to achieve a resilient service-oriented computing based on ad-hoc Web service compositions in dynamic environments, we firstly apply a structured natural language based requirements to capture business users' needs in terms of objectives, and develop a composition requirement model including three levels to support both business users and technical users. Secondly, we propose a transformation approach that transforms structured English language-based business requirements to technical composition requirements by use of an intermediate model, namely the Web service capability model; we introduce our ad-hoc Web service composition approach in dynamic environments to compose Web services driven by transformed technical requirements and adapt to different changes. In addition to the general view of the resilient Service Computing paradigm, our contributions in this thesis mainly consist of the following.

The Three-level Requirement Model: we model composition requirement in three levels namely business-centric requirement, capability-focused requirement, and rule-driven Web service requirement, allowing either technical or business users can specify their composition requirements and consequently compose and execute Web services to satisfy their requirements. The reference model takes into account three main constraints influencing Web service composition process: control flow constraints, QoS constraints, and dependency constraints.

Business-centric requirement is modeled based on Semantics of Business Vocabulary and Business Rules (SBVR), which is a structured English and rule-based language to facilitate the collaboration among customers, business experts, IT specialists, and developers. SBVR defines semantics for vocabulary and rules in a domain of interest. Business-centric requirement is specified by business users to describe the business objectives, functional requirements and non-functional requirements to explicitly provide a model of formal logic so requirements can be understood by both computer programs and other users.

Capability-focused requirement is specified based on a proposed Web service capability model. The semantics of a capability is captured via a combination of action-verb and noun pairs, a set of attributes, describing required resources, current states, and new states, which reflect changes in the real world effects, and particularly, business objectives to achieve by applying the capability. An objective can be achieved by one or more capabilities each of which is semantically interlinked to other capabilities by means of composition relationships. In order to capture different levels of abstractions, the capability model is decomposed into three layers: the capability objective layer, the capability profile layer and the inter-capability composition layer.

Rule-driven Web service requirement is modeled based on Web service operations and different categories of composition rules, which are correspondent to three categories of constraints introduced before. Composition rules are formalized for the direct use of future Web service composition process.

The Two-level Requirement Transformation: we present a top-down composition requirement transformation process to transform business-centric requirements to capability-focused requirement, and finally to rule-driven Web service requirements. This approach consists of two processes based on semantically keyword matching, i.e., the *capability matching* process and the *association discovery* process.

Capability matching transforms business-centric requirements to capability-focused requirements in order to derive available Web service

capability instances to satisfy users' business-centric requirements. The association discovery process discovers all Web services that are associated with the capability instances from the previous step as well as derives different composition rules ready to be used by Web service compositions. As our Web service capability model includes a capability objective layer to describe business objectives to achieve by applying the capability, Web service capability instance created based on capability model is consequently connected to users' business objectives described in business-centric requirements, while all Web services are associated with at least one capability instance created by domain experts or service providers, capability instances are also connected with Web service profiles. In the whole transformation process, capability model is used as an intermediate model to bridge the gap between composition requirements based on SBVR and composition requirements in terms of composition rules.

The concept of capability denotes what an action does in terms of changes in the state of entities that are involved in an interaction [MLMB06]. The purpose of using capability is to describe what services can do without needing to know all the details and, consequently, allows users to discover and invoke services that perform particular kinds of operations in particular contexts. A cornerstone of the SOA is that capabilities consolidate Web services and business processes [MLMB06].

The Ad-hoc Web Service Composition Approach driven by Rules: we propose a novel Web service composition approach, called *Service Farming*, which infers in a reasonable time the optimal composite service. Our approach is ad-hoc in the sense that it composes Web service without predefined composition plan by simultaneously selecting atomic services and inferring the composition patterns between the selected services so as to provide the best ways to compose services together with regard to QoS. Here, the optimal composite service refers to the composite service that satisfies all constraints imposed by the transformed rule-driven Web service requirements, as well as possesses the largest value of Web service utility. Service utility refers to the total satisfaction received by a user from consuming a service, and in our approach, it is estimated based on QoS values before service execution and calculated by a preference-based utility calculation method. Multiple constraints are structured in terms of composition rules to guide the composition process.

In order to adapt to dynamic composition environment, we provide our service farming approach with two key measures, i.e., Web service substitution and composition replanning to ensure optimal composite service is successfully executed when contextual changes occur.

We develop a prototype to validate the resilient SOA. Based on a motivation scenario of a train crash, we illustrate how business-centric requirement is specified and gradually transformed to rule-driven Web service requirements; and then our implemented Service Farming approach is applied to compose Web services together to satisfy users' initialized business-centric requirements, we also discuss relations between the performance and the values of different parameters in our approach.

1.4 Motivation Scenario

In order to explicitly present our composition requirement model, requirement transformation process, and ad-hoc Web service composition approach, we illustrate several SOA applications in dynamic environment. We then present a motivation scenario of a train crash and make assumptions to show how we can apply ad-hoc Web service composition in dynamic environments to achieve resilient SOC.

1.4.1 Dynamic Environment

Web service composition environments, in which Web services are composed and executed, can be divided into two categories according to the contextual information influencing composition process: static composition environment and dynamic composition environment. In static Web service composition environments, contextual information influencing the composition process rarely changes and they are tolerated or neglected; while in the dynamic composition environments, contextual information changes over time and should be handled to adapt the Web service composition accordingly. In practice, assuming that Web service composition environments are static is very reductive and somehow unrealistic due to the endogenous and exogenous changes that may occur during design time and runtime.

We present hereafter several dynamic environment examples where changes may occur during the composition processes. In response to changes, Web service composition approach should be able to transparently adapt to environment changes with minimum business user interventions and to provide most appropriate composite services that satisfy user's requirements. In the following, we illustrate some examples of dynamic environments that are often used in application of real cases.

1) *Pervasive Environments*: Pervasive environment is a global communication environment where a diffuse computing allows

computers and other mobile smart devices to automatically recognize and locate each other; while the pervasive computing allow users to access every information using every device and over every network at every time [IBMO09]. This environment is dynamic as personal computers and other handheld devices are largely widespread everyday and take a large part of information systems; On the other hand, the data information in the environment may be distributed over large areas through networks that range from a world-wide network like the Internet to local peer-to-peer connections like for sensors and are updated overtime [GRLP10]. For example, smart phone has become more and more popular in daily life, services delivered via phones are more often composed on the fly instead of following a predefined composition plan to satisfy users requirements since the available applications and other devices change overtime in the environment and thus requires an ad-hoc Web service composition approach.

2) *Case Handling Environments*: Case handling is a paradigm to support flexible and knowledge intensive business processes [AAWG05]. Unlike workflow management, which uses predefined process control structures to determine what should be done during a workflow process, case handling focuses on what can be done to achieve a business goal. In case handling, the knowledge worker in charge of a particular case actively decides on how the goal of that case is reached, and the role of a case handling system is assisting rather than guiding the worker in doing so. Apparently, in case handling system a task is accomplished via the interaction between workers and systems, the action to be proceeded or the resources to be selected is determined at the last moment, and a predefined process doesn't exist. In this context, an ad-hoc service composition approach can also be helpful to construct composition and create the entire process with predefined composition plans.
3) *Ad-hoc Collaboration Environments:* Ad-hoc collaboration is defined in [KOCK09] as "short-term, on demand, spontaneous, and task-specific collaboration". In ad-hoc collaboration environment, a task is collaboratively achieved by teams, however the typical characteristic of ad-hoc collaboration is that no regularly time is scheduled for the collaboration among individuals, while the collaboration is rather established based on real time task progress and environment information. The task

progress highly depends on all individuals in the team while the performance of individuals tends cannot be precisely predicted, and thus the ad-hoc collaboration environment are highly dynamic due to the collaborations established in real time, and an ad-hoc Web service composition approach is able to create the collaboration in an ad-hoc way.

4) *Crisis management Environments:* A crisis management is the whole of processes, means and organizations, which allow a group of actors to be prepared when a crisis appears[DEVL06]. A crisis usually causes negative facts and the objective of crisis management is to control, reduce and recover from negative facts caused by a crisis (e.g., fire, injuries, buildings collapses). The crisis management environment is dynamic as a crisis often follows an evolutionary process and new facts and events may occur and thus change the environment. In addition, crisis often occurs abruptly and processes to manage them cannot be precisely defined in advance. The crisis management environment is thus dynamic and the environment reflects the changes varying from requirement changes, processes changes, to resources changes. Crisis management requires an ad-hoc Web service composition approach since a crisis occurs abruptly and a general composition plan cannot be precisely defined before a crisis occurs.

1.4.2 A Glance at the Crisis Management as a Dynamic Environment

A crisis is any event that is, or expected to lead to, an unstable and dangerous situation affecting an individual, group, community or whole society, which is deemed to be negative changes in the security, economic, political, societal or environmental affairs, especially when they occur abruptly, with little or no warning [EHSA12]. The crisis management is generally divided in four main steps in [TRBP11]:

- Crisis Prevention, which aims at reducing the probability of crisis appearance;
- Crisis Preparation, which aims at defining the actors' abilities, identifying and preparing the means for crisis response;
- Crisis Response, which aims at solving problems caused by the occurrence of a crisis;
- Crisis Re-establishment, which aims at restoring the subpart of the world affected after the crisis.

In our research work, we support step 2, 3, and 4 with dynamic Web service compositions: when a crisis occurs, how to model crisis management requirements by using a business language, how to identify necessary actions to manage crisis based on requirements, how to carry out different actions in simultaneous and cooperative ways in order to response and re-establish from the crisis, and how to reconfigure crisis management processes when crisis environments change.

1.4.3 Motivation Scenario: Train Crash Crisis Management

We introduce a motivation scenario of a crash of trains with negative facts are caused by the crisis, such as:

1) Injured Victims: drivers and passengers are hurt;
2) Caused Fire: a big fire is caused;
3) Interrupted Electricity: the electricity supply is interrupted in the crisis spot;
4) Damaged Railway: the railways in the crisis spot are damaged.

In order to manage this crisis by using ad-hoc Web service compositions, we make the following assumptions:

1. The crisis management process is regarded as the execution of composite Web services.

2. Different actions, which build up the crisis process, are regarded as the execution of Web services.

3. The adaptation to changes is achieved by reconfiguring composite Web services.

Based on these assumptions, our objective is to rapidly identify appropriate Web services and compose them and execute the resulting composite services in order to reduce and resolve negative facts. We identify three main features, which make our motivation scenario challenging when using existing Web service composition approaches as follows:

1) **Dynamicity**: A crisis is often an evolutionary process, and extra internal or external events may change the crisis environment overtime. Consequently crisis management processes need to be reconfigured. The environment changes include: negative facts, such as local transport accidents, explosions, etc.; new actions to do, such as a fire brigade is occupied by another mission; new strategy to make in order to manage the crisis, and so on.

2) **Responsiveness**: In order to avoid the continuous expansion of negative facts caused by the crisis, the crisis management processes should be created within a short time;

3) **Abruption**: Crisis often occur abruptly and thus appropriate actions and general composition plans to manage crisis cannot be precisely defined before the crisis occurs;

Generally, crisis management processes are manually configured, and then reconfigured when crisis domain changes. This requires the officers' high familiarity of crisis domain, functionalities of all actions that to be undertaken in addition to technical background to specify their requirements and decisions before building crisis management processes. Assuming crisis management officers are not experts in specifying their crisis management requirements based on technical language specification and business processes. An alternative solution is to help them to express their requirements in a structured English language to focus on business-centric requirements to describe crisis management objectives such as "manage train crisis in Paris" while taking into account some necessary actions, such as "fire should be extinguished", "victims should be assisted", and "the total budget should be less than 20000 Euros" etc. Given such kind of requirements, officers expect that Web services that support crisis tasks are automatically discovered and consequently composed to satisfy requirements.

In the following chapters, we extend SOA with different models to achieve resilient service-oriented computing in order to build ad-hoc process and adapt it to contextual changes. The resilient SOA is such applied to solve the following questions in the context of Web service compositions:

1) How officers are allowed to express their business-centric requirements in a structured natural language?
2) How to automatically discover Web services and derive constraints on/among Web services based on users business-centric requirements?
3) How to construct and execute composite services to satisfy multiple constraints imposed by the business-centric requirements?
4) How to adapt to composite Web services to the contextual changes?

1.5 Thesis Outline

As illustrated in Figure 1.2, the dissertation is structured as follows:

Chapter 1 describes our research motivations, research challenges and presenting our major contributions.

Chapter 2 presents the state-of-the-art and covers research background related to our problematic. Firstly, we provide preliminaries on SOA and Web services, and then we respectively discuss current approach-

es for Web services composition, requirement specification and requirement transform.

Chapter 3 introduces our reference model for Web service composition requirements in three different levels: business-centric requirement, capability-focused requirement, and rule-driven Web service requirements.

Chapter 4 presents our top-down and two-level requirement transformation process that transform business-centric requirements to capability-focused requirements and finally to rule-driven Web service requirement for the direct use of our ad-hoc Web service composition approach.

Chapter 5 presents our ad-hoc Web service composition approach, called Service Farming, to construct composite services driven by transformed rule-driven Web service requirements while maximizing users' satisfaction; we also present how we adapt our Web service composition approaches into dynamic composition environment by introducing two key dynamic reconfiguration measures: service substitution and composition replanning.

Chapter 6 illustrates the technical architecture and the implemented prototype to compose Web services based on the train crash motivation scenario, and also present experimental results.

Chapter 7 summarizes our work and outlines future research directions.

Some contributions in our thesis are disseminated in the following international conferences:

-Wenbin LI, Youakim BADR, and Frederique BIENNIER, "Towards Natural-like Requirement based Web Service Composition" The 25th International Conference on Software & Systems Engineering and their Applications (ICSSEA 2013), Télécom ParisTech, Paris, France, November, 2013
-Wenbin LI, Youakim BADR, and Frederique BIENNIER, "Improving Web Services Composition with User Requirement Transformation and Capability Model" The 21st International Conference on Cooperative Information Systems (CooPIS, OTM 2013), Graz, Austria, pp.300-307, September, 2013
-Wenbin LI, Youakim BADR, and Frederique BIENNIER, "Towards a Capability Model for Web Service Composition" IEEE 20th International Conference on Web Services (IEEE ICWS 2103), Santa Clara, U.S.A., pp.609-610, June, 2013
-Wenbin LI, Youakim BADR, and Frederique BIENNIER, "Service Farming: An Adhoc and QoS-aware Web Service Composition Approach" The 28th Symposium On Applied Computing (ACM SAC 2013), Coimbra, Portugal, pp. 750-756, March, 2013
-Wenbin LI, Youakim BADR, and Frederique BIENNIER, "Digital Ecosystems: Challenges and Prospects" The 2012 ACM International Conference on Management of Emergent Digital EcoSystems (ACM MEDES 2012), Addis Ababa, Ethiopia, pp.117-122, October, 2012

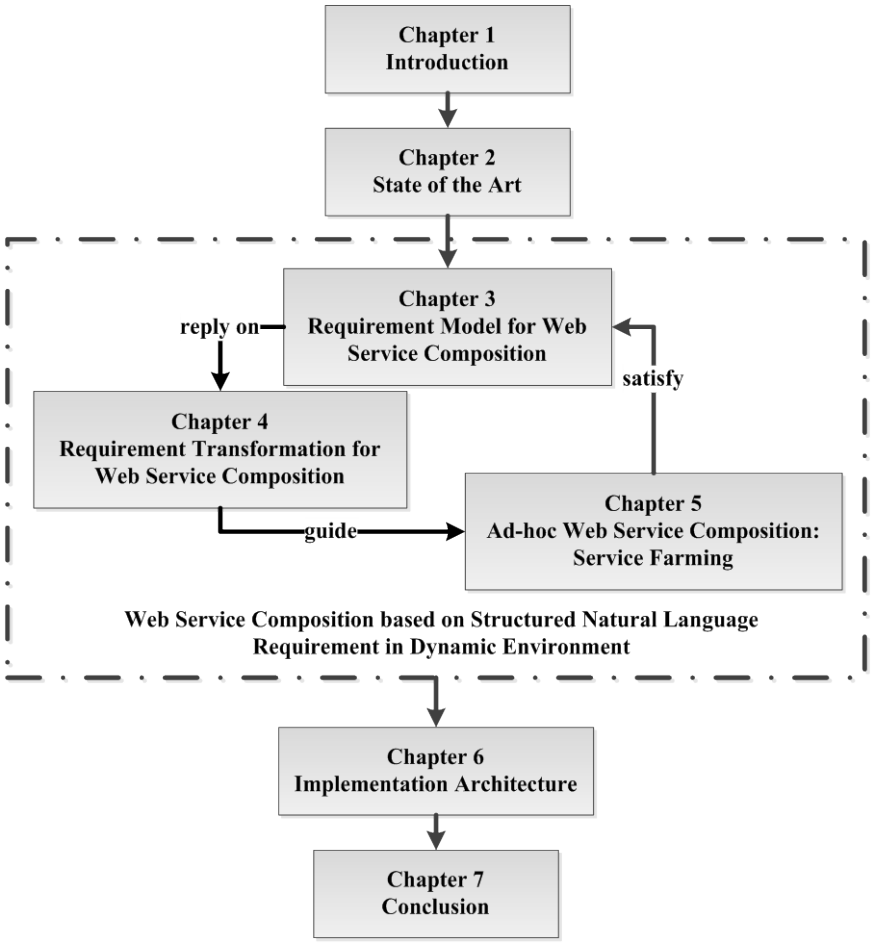

Figure 1.2 The Thesis Organization

Chapter 2

State of the Art

Service-oriented Computing Preliminaries 21
A Glance at Web Service Composition Approaches 29
Overview of Requirement Specifications in Web Service Composition 59
Managing Requirements 64
Executive Summary 69

Abstract: Web service composition approaches have received abundant research attention to support business-to-business or enterprise application integration. Many approaches are proposed to compose Web services together to satisfy users' requirements in different contexts. This chapter seeks to synthesize the state-of-the-art regarding Web service composition approaches focusing on ad-hoc and dynamic perspective. From different points of view, Web service compositions can be classified as manual, semi-automated and automated composition; syntactic composition and semantic composition; static composition and dynamic composition. In this chapter, we firstly introduce the background of Web services and Service-oriented Architecture (SOA), and then we investigate different Web service composition approaches, in the following, we examine current techniques and models to specify and transform requirements for Web service compositions. At last, we elucidate how these models and approaches are not appropriate for an ad-hoc Web service composition approach driven by business requirement specification to construct composite services in dynamic environment without abstract composition plans.

2.1 Service-oriented Computing Preliminaries

Service-Oriented Computing is an emerging cross-disciplinary paradigm for distributed computing that changes the way software applications are designed, architected, delivered and consumed. The SOC [HU-

SI05] promotes the idea of composing self-contained services into a loosely coupled agile software systems. The SOC utilizes services as fundamental elements to support the development of rapid, low-cost and reusable distributed applications even in heterogeneous environments. Services are self described, platform-agnostic computational elements, and perform functions, which can be anything from simple requests to complicated business processes.

The SOC relies on the SOA, which is a way of reorganizing software applications and infrastructure into a set of interacting services [PA-PA03]. The creation of SOA was inspired by the necessity to develop and support complex cross-enterprise information systems that can be quickly and cost-efficiently adapted to changes in the operational environment. In order to construct business applications of loosely-coupled, autonomous and reusable services, a list of principles has to be followed by services when implementing SOA [ERL08]:

Standardized service contract. Services adhere to a communication agreement (also called service contract) as defined by service description documents. Within the same service inventory, services use the same service description standards, such as Web Service Description Language (WSDL);

Service loosely-coupling. Service contract is not tightly coupled with customer requirements nor with service implementations. In this case the service contract can evolve without affecting service consumers and service implementations;

Service abstraction. Details in the service contract, services hide their internal logic;

Service reusability. The service logic is arranged so that to promote its reuse.

Service autonomy. Services have a high level of control of the underlying run-time execution environment;

Service statelessness. Services minimize resource consumption by deferring the management of state information when necessary;

Service discoverability. Services are equipped with communicative meta data by means of which they can be discovered by consumers;

Service composability. Services can be effectively composed into new services with the functionality of arbitrary complexity.

These principles make business applications much more flexible and significantly decrease the cost of their initial development and further support. Following these principles, the conceptual SOA model includes service consumer, service provider, and service registry to provide an inter-

action environment for consuming, publishing, and discovering services, and achieving certain goals [ZHAN08].

Figure 2.1 illustrates the implementation of SOA with Web services [DUSC05]. A Web service is a specific kind of service use the Internet as the communication medium and open Internet-based standards. A Web service is identified by a Uniform Resource iIdentifier (URI), whose public interfaces and bindings are defined and described using Extensible Markup Language (XML). Its definition can be discovered by other software systems. These systems may then interact with the Web service in a manner prescribed by its definition, using XML-based messages conveyed by Internet protocols.

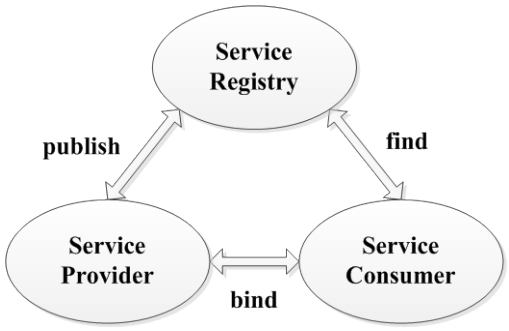

Figure 2.1 SOA with Web Services

In Figure 2.1, the service provider creates or simply offers Web services by describing Web service in a standard format i.e., XML, which in turn is XML and publish them in a central Service Registry. The service registry contains additional information about the service provider, such as address and contact of the providing company, and technical details about the Web service. The Service Consumer retrieves information from the registry and uses the service description obtained to bind to and invoke the Web service.

In general, Web service has the following features that make itself the best service model when developing SOA in heterogeneous environments [RAO04]:

Loosely Coupled: In software development, coupling typically refers to the degree to which software components/modules depend upon each other. Comparing with tightly-coupled components (e.g., Distributed Component Object Model (DCOM) [REF95], Common Object Request Broker Architecture (CORBA) [OBJE04]), Web services are autonomous and operate independently from one another. The loosely coupled feature enables Web services to locate and communicate with each other dynamically at runtime.

Universal Accessibility: Web services can be defined, described and discovered through the Web to enable an easy accessibility. Not only

users can locate appropriate Web services, but services can be described and advertised so that they are possible to bind and interact with each other.

Standards Languages: Web services are described by standards based on XML [WEER05]. Web services standards increase abstraction. Although the cores of Web services may implemented by different programming languages, the interface of Web services are described by uniform standard based XML languages.

Web services are considered to be self-contained, self-describing, modular applications and can be published, located, and invoked across the Web. Currently, they become the most prominent realization technology for SOA [TSPI02][CDKN02]. When composing Web services, two parts of Web service descriptions are especially important: Web service functional properties and Web service non-functional properties including QoS. Web service operations describe functional aspects of a Web service and are generally identified by operation name, input and output messages; whereas QoS is a broad concept that encompasses a number of non-functional properties of Web services, [ROPD06][MORD08][REYT07] define and represent different non-functional properties from the user's perspective. These properties apply both to atomic Web services and composite Web services and include information such as:

Price: The price that a service user has to pay for invoking the service.

Response time: The time interval between the moment when a service is invoked and the moment when it is finished.

Reliability: The probability that a request is correctly responded within the maximum expected time.

Availability: The probability that a service is available during the request. The availability is calculated by dividing the downtime by uptime (downtime ratio) and subtracting it from the maximum availability.

Reputation: The average ranking given to the service by end users according to their own experiences.

Although SOA has its advantage of enables the creation of applications that are built by combining loosely coupled and interoperable Web services, current SOA standard still has several drawbacks and thus require improvements.

1. Inability to deal with continuous and often unpredictable change [RONA09]. SOA has its standards to support describing, publishing and invocating Web services from different standards. However, no standard has been widely established to support dealing with exceptions occurred during invocating single or list of Web services.

2. Lack of support in business level. The development of service-oriented architecture depends on the implementation of standard technologies that will be introduced in the following. In order to compose Web services based on SOA, users are expected to be familiar with the technical standards: users should not only be able to specify the composition requirements based on technical languages, but also they are expected to deal with any exception during composition processes in the technical level. However, SOA lacks of appropriate models to support business users who are not familiar with technical details to use, which leads us to the assumption that Web services composition cannot be widely applied to business process design.

In order to develop resilient applications in dynamic environment, we need build rSOC based on current SOA standard and provide system with extra capabilities to deal with endogenous and exogenous change changes and adapt to business levels. In the following, we illustrate technologies to support Web service descriptions and implementations.

2.1.1 WSDL

The Web Service Description Language (WSDL) is an XML format for describing Web services as a set of endpoints operating on messages containing either document-oriented or procedure-oriented information [CDKN02]. Operations and messages are described abstractly, and then bound to a concrete network protocol and message format in order to define an endpoint. Related concrete endpoints are combined into abstract endpoints. WSDL is designed to allow description of endpoints and their messages regardless of what message formats or network protocols are used to communicate.

A WSDL document uses the following elements in the definition of Web services as shown in Figure 2.2:

Types: the date represented by a container for data type definitions using some type system (e.g., XML Schema);

Interface: the operations that can be performed, and the messages that are used to perform the operation;

Operation: an abstract description of an action supported by the service;

Binding: a concrete protocol and data format specification for a particular port type;

Endnote: a single endpoint defined as a combination of a binding and a network address;

Service: a collection of related endpoints.

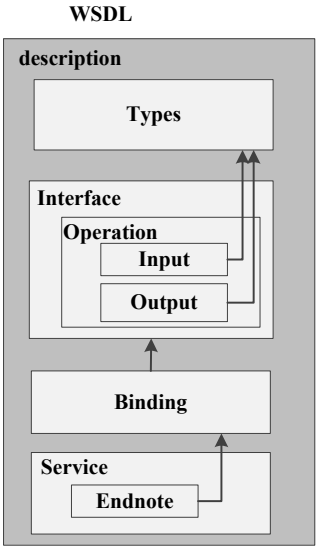

Figure 2.2 WSDL Document Representation

2.1.2 SOAP

The Simple Object Access Protocol (SOAP) is a protocol intended for exchanging structured information in a decentralized, distributed environment in order for services to be able to describe their capabilities, and to allow applications on the Internet to use those capabilities. By means of SOAP different object models can be bridged over the Internet and an open mechanism is provided that allows applications to communicate with each another [MMNJ07].

The basic elements of a SOAP message are depicted in Figure 2.3. A SOAP message is an XML document that is sent via a communication protocol, such as HTTP or SMTP. A message without attachments is made up of an envelope, an optional SOAP header and a SOAP body. The optional SOAPHeader element can include one or more headers, which carry metadata about the message (e.g. information regarding the receiving and sending parties). The SOAPBody element, which always follows the SOAPHeader element, contains the message content. In case of an error, SOAPBody carries fault and status information. A SOAP message can also include one or more attachments in addition to the SOAP part, which must contain XML content only.

In addition to the messages structure, SOAP also defines a processing model, error handling mechanisms, an extensibility model, an RPC convention and a protocol binding framework [WEER05].

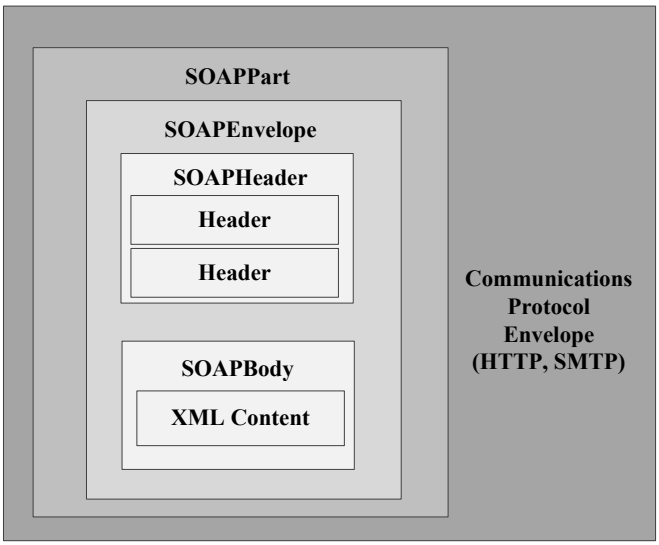

Figure 2.3 SOAP Message Structure

2.1.3 UDDI

The Universal Description, Discovery and Integration (UDDI) [Oas04] is a group of Web-based registries that expose information about a business or other entity. The UDDI provides extensions to the basic Web Services technologies by creating a registry of Web services. A UDDI registration consists of three components: White Pages, Yellow Pages, and Green Pages. White pages give information about the business supplying the service; Yellow pages provide a classification of the service or business, based on standard taxonomies; Green pages are used to describe how to access a Web Service, with information on the service bindings [TAYL04].

The UDDI enables service consumers to quickly, easily, and dynamically find and transact with Web services. It provides the following functionalities [KATT12]:

- Find Web services implementations that are based on a common abstract interface definition.
- Query Web services providers that are classified according to a known classification scheme.
- Issue a search for services based on a general keyword.
- Cache information about a Web service and then update at runtime.

2.1.4 OWL-S

The WSDL only provides syntactical description of Web service functionalities. Other researches attempt to annotate Web service with se-

mantic description. OWL-S is an ontology built on top of Web Ontology Language (OWL) for describing Semantic Web Services. It will enable users and software agents to automatically discover, invoke, compose, and monitor Web resources offering services, under specified constraints [MBHL04].

The OWL-S ontology has three main parts as shown in Figure 2.4: the service profile, the process model and the grounding [MPMB05]. The service profile is used to describe what the service does. This information is primary meant for human reading, and includes the service name and description, limitations on applicability and quality of service, publisher and contact information. The process model describes how a client can interact with the service. This description includes the sets of inputs, outputs, pre-conditions and results of the service execution. The service grounding specifies the details that a client needs to interact with the service, as communication protocols, message formats, port numbers, etc.

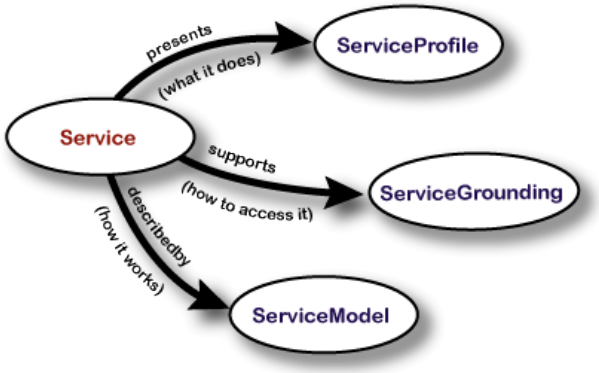

Figure 2.4 OWL-S Structure

2.1.5 Enterprise Service Bus Infrastructures

An enterprise service bus (ESB) is a software architecture model used for designing and implementing the interaction and communication between mutually interacting software applications in service-oriented architecture (SOA) [KABH04].

ESB acts as an intermediary between service consumers and service providers [PGPS11] as shown in Figure 2.5. Service consumers are designed to interact with the ESB and the ESB is configured to route and transform different kinds of request and response messages between service consumers and service providers. There are vendor products that implement many of the features described below in the supporting patterns and tactics section.

In order to support the design of our proposed rSOC concept, we introduce a work presented in [SAMI08] to achieve a resilient scatter-

gather ESB messaging design with message-driven beans to resolve the scatter-gather application integration problem when a requester sends an asynchronous request to a number of providers, which send their replies asynchronously to the requester.

Figure 2.5 ESB Architecture Pattern

In this work, one message-driven bean (MDB) receives a request message from a requester and publishes it to multiple providers, while another MDB aggregates the replies from the providers and sends the aggregate to the requester. The scatter-gather design resilience relies on Enterprise JavaBeans (EJB) and Java Message Service (JMS) provider runtime services, message delivery assured by the JMS provider, EJB transaction management, and automatic triggering of an MDB resulting from arrival of a message at the MDB-specified destination. The work focuses on the design and deployment of the scatter-gather components, and presents a good reference on how to resolve the common scatter-gather application integration problem.

2.2 A Glance at Web Service Composition Approaches

Web service composition has become an emerging development process to design and build complex inter-enterprise business applications out of existing Web services. In the Web service composition, services come in two flavors: atomic services and composite services. Atomic service is a single Web service that cannot be divisible and is readily available and deployable; composite services involve assembling existing atomic Web services and other Web composite services that access and combine information and functions from possibly multiple service providers.

When composing Web services, Web service discovery and Web service selection are generally two important processes prior to Web service composition. The Web service discovery aims at discovering specific Web services from Web service registries based on certain functional requirements. According to W3C consortium, Web service discovery is defined as the act of locating a machine-processable description of a Web service that may have been previously unknown and that meets certain functional criteria [FERR04]. The Web service selection is to select appropriate Web services from results of the Web discovery process based on non-functional requirements. The selected Web services are thus used to construct composite services.

Different approaches tackle the problem of Web service compositions, and can be classified into manual, semi-automated or automated compositions [CHBB07]; syntactic or semantic Web service composition [AGHS03][RASU05]; static or dynamic Web service composition [DUSC05]; in addition, many approaches particularly focus on QoS to derive and provide QoS-aware Web service compositions [ZBDK03] [STRU10] [HLXZ11][SHCH11].

In the following, we synthesize the state of the art in Web service composition approaches and we analyze them to which extent they can be applied to develop ad-hoc composition in dynamic environment. We also present and classify them, and conclude with their limitations in solving our research challenges related to developing resilient SOA.

2.2.1 Manual, Semi-Automated and Automated Composition Approaches

Web service composition approaches can be classified into three categories: manual, semi-automated and automated composition approaches [RASU05] from the respective of facilitating the composition process by reducing users' interventions in the design and runtime. Early works mainly contribute to manual and semi-automated Web service composition approaches, whereas recent researches mostly dedicate to automate Web service composition processes.

2.2.1.1 Manual Composition Approaches

The manual Web composition is the Web service composition process that is based on human intervention and deals with low-level programming and implementation issues. Most manual composition approaches [ADHW05][TASW03][BESD03] expect users to generate specific and executable workflow scripts to represent composite services and describe their execution order, either graphically or through declarative languages.

A workflow consists of a sequence connected tasks [MMVL05] and emphasis on the control flow. Composite services can be represented as workflows [CASS01], since they include sets of atomic services together with the control and data flows among Web services.

In order to manually build workflow or Web service-based processes, many process modeling languages have been proposed such as BPML, BPEL, etc. The Business Process Modeling Language (BPML) is a meta-language for describing business processes [ADHW05]. Basic activities for sending, receiving, and invoking services are available, along with structured activities that handle conditional choices, sequential and parallel activities, joins, and looping. The BPML also supports the scheduling of

tasks at specific times. The Business Process Execution Language for Web Services (BPEL4WS) is another low-level process modeling and execution language for Web service compositions [WADH03]. The BPEL4WS provides notations for describing interactions of Web services, and thus allows developers to implement composite Web services using simple monolithic blocks.

Many manual Web service composition approaches rely on graphical editors. Authors in [TASW03], for example, propose a graphical user interface, called Triana, allowing users to select required services from a toolbox and "drag-and-drop" them onto a canvas. Services are retrieved from the UDDI registry keywords search. Another graphical editor [CCMN04], called BPWS4J, has been proposed based on the BPEL-WS language to extend the capabilities with programming languages such as JAVA. BPWS4J integrates an Eclipse plug-in to allow the user to compose a graph at the XML level. This composed graph, along with a WSDL document for the composite service, is submitted to the execution engine.

Self-Serve [BESD03] is another state chart based editor that allows users to build workflows by locating required services by using the service builder. The service builder interacts with UDDI to retrieve service meta-data and then executes the state chart in a P2P based execution model.

Despite these works, the potential number of suitable Web services available online is extremely large, and is already beyond the human capability to deal with the whole Web service composition process manually. Additionally, manual composition approaches are usually time-consuming, error-prone and often not scalable processes. Therefore, constructing composite Web services with a semi automated or automated approaches is advantageous.

2.2.1.2 Semi-Automated Composition Approaches

The semi-automated service composition is a step forward than the manual service composition in the sense that it makes recommendations for Web service selection during the composition process [SIHP02]. Users then select appropriate services from a shortlist of recommended services and link them up in the desired execution order.

The myGrid framework [STRG03] allows users to compose, store and execute workflows. Users browse a registry and select a workflow template. This template is then submitted to the enactment engine which asks users to select actual instances of the workflow components, e.g., Web services.

In order to simplify the users' selection process, many research works seek to reduce the number of choices that users have to make when

composing Web services. For example, the semi-automated approach in [SIHP02] proposes recommended Web services by matching Web services' functional prosperities and filtering them based on non-functional properties when creating composite Web services. The composite service is then executed by invoking each individual selected Web service and passing data between them according to the control flow.

In order to facilitate the modification of existing compositions based on workflow, a framework is proposed in [CASH03] to provide assistance to users by recommending Web services meeting users' requirements. When users add Web service to the workflow, they start by creating a service template which indicates their intentions to extend the workflow functionalities. The service template is sent to the Web service discovery module, which returns a set of service object references that are ranked according to their degree of similarity with respect to the service template. Users then select the most appropriate Web services to accomplish their objectives. Furthermore, a knowledge-based advice system for service composition is proposed in [CSGT03], which uses domain knowledge and provides advice and guidance with respect to the selection, sequencing and configuration of services. It additionally relies on semantically enriched service descriptions to assist in the process of discovering available services. The ability to exploit service descriptions facilitates the composition specification process with respect to existing descriptions of Web resources.

The work presented in [DIPW08] reduces the number of choices that users have to make by restricting the overall set of Web services by ranking Web services so that the most desirable ones are presented first to use. Authors in [ZAPG09] propose a rule-based approach for the semi-automatic composition problem, giving end-user the control to guide the overall composition process. End-users build the composition flow by selecting constrained Web service types, called nodes. End-users connect them using a set of control and data flow connectors. The specified nodes will then be bounded to concrete Web service instances using a set of rule-based queries satisfying the associated constraints. When compared to current semi-automatic approaches, this approach is declarative, allows specifying both functional and non-functional requirements, and provides connectors that include both the data and control flow aspects.

An assistant goal oriented service composition approach is proposed in [ALPA10] for semi-automatic compositions where the composition process is gradually generated by using a declarative oriented generic composition plan. Users do to tell the system what to do but rather they establish and negotiate about goals and how to accomplish them. At each composition step, the system proposes to users which new services can be

added to the composition and which kind of actions can be taken, based on the current context, the composition objective and user preferences.

Although these representative semi-automated composition approaches solve some shortcomings of manual composition approaches, they are still not scalable as the user is expected to browse a registry to select appropriate services. In dynamic environment, semi-automated Web service composition approaches highly rely on users' intervention when contextual information changes. To that end, many approaches for this purpose have been proposed to fully automate Web service composition process.

2.2.1.3 Automated Composition Approaches

Compared to manual and semi-automated compositions, automated Web service compositions generate the entirely composite service without human involvement. The automated Web service composition is defined as "Given a query describing the composition goal and providing some inputs, design automatically a composite service from the available services such that if it is executed, it produces the required goal. [BABI12]" A conceptual model of automated Web service composition is shown in Figure 2.6 [RAIK12].

The composer automatically derives executable compositions from Web service descriptions and composition requirements. For different solutions, structures of inputs and outputs may widely vary. Service descriptions normally include specifications of service interfaces. Annotations such as non-functional property annotations [ZBND04], and semantic annotations [KVBF07], enable richer composition requirements [BKPP09]. Composition requirements express the users' expectations about the service composition. Finally, the outcome of the composer is an executable composite service.

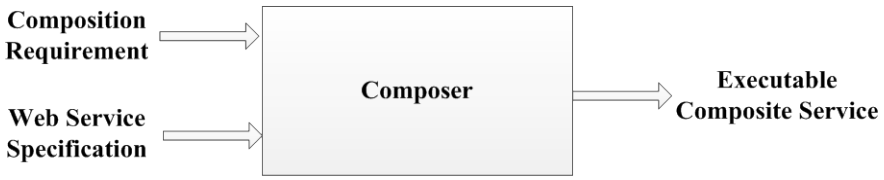

Figure 2.6 A Conceptual Model of Automated Web Service Composition

In the following, we cover automated Web service composition approaches according to the reasoning mechanism they exploit namely,

- Automated composition based on finite state machine;
- Automated composition based on based on situation calculus;
- Automated composition based on based planning domain definition language;
- Automated composition based on rule-based planning;

- Automated composition based on theorem proving;
- Automated composition based on hierarchical task network.

We conclude this section with a comparison of all these approaches and provide insights from requirement specification and ad-hoc composition in dynamic environment perspectives.

2.2.1.3.1 Automated Composition based on Finite State Machine

The FSM is a mathematical model of computation conceived as an abstract machine that can be in one of a finite number of states [CHU06]. The state of the machine is certain at a time, and it can change from one state to another through transition. A particular FSM is defined by a list of its states, and the triggering condition for each transition.

In [BCGL03], Web services to be composed are encoded as FSM on the basis of their exported behaviors (i.e., protocols). The user specifies a desired behavior of a composite service as a tree of actions which is, again, transformed into FSM. By analyzing all available services, the approach provides how to construct a composite service from available Web services. An extension of this work is presented in [BCGH05] allows for advanced control flow requirements relying on semantic-like annotations of Web services in terms of effects on the real world. It also addresses basic data flow requirements in terms of data pieces that services can receive and send. Another extension is proposed in [KAKS07] to compose Web services based on a graph search algorithm. Component services and composition requirements are thus modeled as directed graphs with rich semantic attributes. To make search more efficient, all services stored in the registry are joined into an aggregated graph representing collective behavior of a service system. Once a goal graph is specified, a search algorithm is used to find solution in the aggregation graph.

Composition approaches based on finite state machine require the composition requirements specified in terms of directed graphs with attributes but do not support requirement specification in business languages, and moreover, they do not provide solutions to deal with contextual information changes in dynamic environment.

2.2.1.3.2 Automated Composition based on Situation Calculus

Planning techniques are regarded as one of the most popular techniques to automate Web service composition process in current research. Planning is the process of thinking about and organizing the activities required to achieve a desired goal. It is proved in [CAST03] that the Web

service composition problem can be viewed as a planning problem in which state descriptions are ambiguous and operator definitions are incomplete.

One of the planning techniques used to automate Web service composition process is situation calculus. The situation calculus is a logic formalism designed for representing and planning about actions/tasks [LALE05]. The situation calculus represents changing scenarios as a set of first-order logic formulae. The basic elements of the calculus are: the actions that can be performed in the world; the fluent that describes the state of the world, and the situations.

McIlraith et. al. [NAMC02][MCIL02] adapt and extend the Golog language for automatic construction of Web services. Golog is a logic programming language built on top of the situation calculus. The authors address the Web service composition problem through the provision of generic processes and customizing constraints. The general idea of the work is that software agents could reason about Web services to perform automatic Web service discovery, execution, composition and inter-operation. User request and constraints are presented by the first-order language of the situation calculus. Each Web service is conceived as an action. A knowledge base provides a logical encoding of the preconditions and effects of the Web service actions expressed with the situation calculus. Based on deductive reasoning, a composite service is then constructed as a set of atomic services which connected by procedural programming language constructs (if-then-else, while and so forth).

The work in [ZHAO10] extends the semantic of OWL-S through semantic capabilities of the norm. Norm is also known as social norms which is the common rules of conduct and standards of every member in a social group or smaller groups. This work uses situation calculus to formally describe the norm in order to ensure the correctness of norm and the user correct expression of various scene constraints, and provide more accurate and subjective service in expression of the preferences. The situation calculus is applied to realize formal norm in order to ensure the accuracy of semantic Web service composition.

Web service composition approaches based on situation calculus expect composition tasks to be defined as a set of first-order logic formulae which are difficult to use for users who are not familiar with technical details. In addition, before constructing composite services there is still a considerable effort on modeling domain information such as the actions to perform, the states of the world, and the situations, which also increases the complexity of Web service composition.

2.2.1.3.3 Automated Composition based on Planning Domain Definition Language

Planning Domain Definition Language (PDDL) is a standard encoding language for planning tasks. It expresses mediators, possible actions and their structures, and action effects.

McDermott [MCDE02] present an approach to compose Web services based on PDDL. When composing Web services, an estimated-regression planner is used as a backward analysis of all possible actions to achieve a goal to guide a forward search through situation space. The advantage of this work is a proposed new type of knowledge, called value of an action, which persists and which is not treated as a truth literal. From Web service construction perspective, the value of an action enables users to distinguish the information transformation and the state change produced by the execution of the service. The information, which is presented by the input/output parameters are thought to be reusable, thus the data values can be reused for the execution of multiple services.

When a plan is generated, an approach for translating the produced PDDL plans to OWL-S descriptions of the final composite Web services is proposed in [FOLO03]. The result is a new Web service that can later be discovered and invoked or take part in a new composition.

A similar work is introduced in [BOZH09] to support composing Web services described by OWL-S based on PDDL. This work provides a solution on how to translate OWL-S process models to the PDDL actions so that PDDL can be applied to compose Web services.

PDDL is an encoding language for technical users to specify their requirements and construct composite Web services; however, they are not suitable for business users who specify their requirements in terms of business objectives and expect Web services to be composed following business languages based requirements.

2.2.1.3.4 Automated Composition based on Rule-based Planning

Rule-based planning refers to the planning process under guidance of rules, which are structured and used to make deduction or choices.

The work presented in [POFO02] constructs composite Web services using rule-based plan generation by modeling services by their preconditions and post-conditions. A Web service is represented in the form of a horn rule that denotes the post-conditions are achieved if the preconditions are true. To create a composite service, the service requester only

needs to specify initial and final states, and then the plan generation can be achieved using a rule-based expert system.

An similar approach is presented in [MEBE03] to generate composite services from declarative descriptions by specifying the operations to be performed through composition. This work uses composability rules to determine whether two services are composable. The composition approach consists of four phases. The specification phase enables descriptions of the desired compositions using a language called Composite Service Specification Language (CSSL). Second, the matchmaking phase uses composability rules to generate composition plans that conform to user's specifications. In the selection phase, if more than one plan is generated, the user selects a plan based on quality of composition (QoC) parameters (e.g. rank, cost, etc.). In the generation phase, a detailed description of the composite service is automatically generated and presented to the user. The main contribution of this method relies on the composability rules, because they define the possible Web service's attributes that could be used in service composition. Those rules can be used as a guideline for other Web service composition approaches.

The SECE (Sense Everything, Control Everything) is a platform for context-aware service composition based on user-defined rules and ontology descriptions of services [BEAS12]. The SECE creates user-defined rules based on the ontology descriptions of services to discover Web services and to issue more complex queries for service discovery and composition.

The advantage of Web service composition approaches based rule-based planning lies on its extensibility since composition approaches are able to take into account new constraints by introducing new categories of rules and accordingly updating composition algorithms. However, the introduced approaches do not provide any solution for dealing with contextual information changes in dynamic environment.

2.2.1.3.5 Automated Composition based Theorem Proving

Theorem proving refers to the proving of mathematical theorems by computer programs and deals with the development of computer programs that show that some statements are a logical consequence of a set of statements [NEWB01].

Waldinger [WALD01] elaborates the Web service composition from the theorem proving perspective based on automated deduction and program synthesis. In this work, agents cooperate with each other to answer a query, which is phrased as a theorem and the answer is extracted from a

proof. Available Web services and user requirements are described in a first-order language, and then constructive proofs are generated with a theorem prover. Finally, Web service composition descriptions are extracted from particular proofs.

The framework introduced in [PAFL11] for the formal verification of Web services composition is based on the proofs-as-processes paradigm and enables inference rules of Classical Linear Logic (CLL) to be translated into π-calculus processes. In this context, composition is achieved by representing available Web services as CLL sentences, proving the requested composite service as a conjecture, and then extracting the constructed π-calculus term from the proof. This work composes Web services by taking into account both functional and non-functional requirements.

Web service composition based on theorem proving provides contributions when composite services can be constructed from the exiting Web services and show how composite services can be constructed; however, these approaches lack the capacity to provide optimal composite Web services to maximize user's satisfaction when many possible composite Web services exist.

2.2.1.3.6 Automated Composition based on Hierarchical Task Network

Hierarchical task network (HTN) is an approach to automate planning in which the dependency among actions can be given in the form of networks. Planning problems are thus specified in HTN by providing a set of tasks, i.e., primitive tasks (i.e., actions that can be executed), compound tasks (i.e., complex tasks composed of a sequence of actions), and goal tasks (i.e., tasks satisfying conditions).

The SHOP2 planner, which is AI planner based on HTN, is applied to automatically compose Web services [SPWH04] by translating OWL-S description of Web services to SHOP2. In particular, most control constructs can be expressed by SHOP2 in an explicit way. The Web service composition is thus conducted by recursively decomposing goal tasks and searching compound tasks and primitive tasks satisfying the goal tasks.

Based on HTN and non-functional requirements, authors in [LIN08] present an automatic Web service composition approach satisfying user preferences as much as possible. Users express constraints over the states and specify state trajectory corresponding to the plan using a special preference language. The work mainly describes preferences augment service compositions and how they are mapped into a planning language for HTNs.

Authors in [KUXR09] consider that previous composition approaches based on HTN planning have not taken into account decompositions to a problem can lead to a variety of valid solutions. In [KUXR09], a model of combining a Markov decision process model and HTN planning is presented to address Web services composition. In this model, HTN planning is enhanced by decomposing a task in multiple ways and hence being able to find more than one plan, taking into account both functional and non-functional properties.

In general, the HTN is proved to be an effective planner to compose Web services, however, composition based on HTN planning is based on the notion of composite tasks that can be refined to atomic tasks using predefined formalism, which requires a great deal of pre-processing work in technical level.

2.2.1.3.7 Comparison and Limitations

Generally speaking, Web service composition based on planning firstly converts the composition problem into planning problem by transforming Web service descriptions and requirements into necessary planning domain descriptions, and secondly different planners are applied to generate planning solutions to satisfy the goal states. At last, these planning solutions are transformed into executable specifications like BPEL documents or other XML-based descriptions, and executed through the corresponding engines.

To compare different automated Web service composition approaches, we adopt the criteria introduced in [BCDD08] from four perspectives:

Domain Independence: The approach is not exclusive to a specific domain (e.g. travelling, automotive) but can be applied to any, allowing for the solution of a broad range of problems;

Partial Observability: The approach is able to reason on incomplete information;

Non-determinism: The approach deals with actions that may lead to different states depending on the values of some parameters (e.g. an if-else construct)

Scalability: The approach is able to solve real-world composition problems which often deal with a large number of services. A low scalability means that this particular category of approaches becomes less efficient as the number of associated services and/or their complexity rises. On the contrary, a high scalability ensures that there is support for large numbers of services and/or high levels of complexity.

Table 2.1 compares all automated composition approaches introduced in section 2.2.1.3 based on different categories.

Table 2.1 Comparison of Automated Composition Approaches

Automated Composition	Domain Independence	Partial Observability	Non Determinism	Scalability
FSM	Yes	Yes	Yes	Average
Situation Calculus	Yes	Yes	Yes	Average
PDDL	No	Yes	No	Good
Rule-based Planning	Yes	No	No	Varies
Theorem Proving	Yes	No	No	Good
HTN	Yes	Yes	Yes	Good

Based on Table 2.1, we conclude that automated composition approaches based on HTN planning is better than other planning based composition approaches since composition based on HTN planning has good scalability, partial observaiblity, non-determinism, and it can be domain independent.

Despite the large spectrum of Web service composition approaches, there are still limitations when these approaches are applied to support business requirements from business users and adapt to contextual changes in dynamic environments. We elaborate the limitations as follows:

1) Composition requirement specifications highly rely on users' technical background. Almost all beforementioned approaches requires that users' composition requirements are formally expressed in with technical or formal methods. For example, in HTN based composition, users' requirements need to be defined in the form of goal tasks which specify the exact expected state that a composite service should achieve; whereas in theorem proving based service compositions, composition requirements should be defined as theorems. As a matter of fact, users should be familiar with the composition domain and composition techniques. In addition, these approaches do not support business users to specify their requirements based on general business objectives in a natural language; additionally, since the goal (i.e., composition requirement) is expected to specifically describe the future state of the domain, this makes the transformation from business-centric requirements with general objectives to specific goal states difficult.

2) Web service composition requires pre-formalized domain description. Not only composition requirements should be formally defined following technical details or formal methods, other composition domain information should also be formalized according to different planners, such

as describing initial domain states, possible domain states, and different actions. Although automated service composition approaches significantly reduce the amount of manual work to construct composite services, a considerable effort remains necessarily when modeling the composition domain (e.g., service annotations, ontologies, goal abstract processes etc.). Moreover, current standard of Web service description cannot be directly adhered, which apparently increases the complexity of Web service compositions.

3) Web service composition process is vulnerable to dynamic environment changes. Although the before introduced approaches can be effective to compose Web services under certain conditions, they do not easily adapt to contextual changes in dynamic environments. When changes occur at rum time, existing composite Web services may be affected (e.g., disappearance of Web services participating in the composition). If dynamic changes happen frequently, redesigning composition requirements and accompanying specifications may become time consuming. As a result, these solutions are usually applicable to a very limited set of real world composition problems and cannot be largely adopted as general-purpose compositions.

4) Web service composition does not fully take into account multiple constraints. As introduced before, multiple constraints such as control flow constraints, QoS constraints, and dependency constraints, can influence Web service composition process, such as. Most of existing approaches provide composition solutions satisfying functional properties while few approaches take into account users' QoS constraints and QoS preference as well; They neglect, for example, dependency relations between/among Web services. In addition, control flow constraints are mainly considered by matching input/output messages or precondition/effect on Web services to be composed; however, they do not support composing Web services following users' personalized control flow constraints, which indicate the preferred execution order for certain specific Web services or Web services with the same operations.

2.2.2 Syntactic and Semantic based Composition Approaches

From a semantic perspective, Web service composition approaches can be classified into two categories: syntactic based Web service compositions and semantic based Web service compositions. Generally speaking, Web service XML-based standards (e.g., WSDL, UDDI, etc.) focus on syntax structures to facilitate machine processing; and are for syntax composition, whereas ontology-based standards (e.g., OWL-S, RDF, etc.) focus

on adding semantic information to Web service XML-based standards to facilitate users' manipulation.

2.2.2.1 Syntactic Web service Composition Approaches

The syntactic Web service composition refers to the composition process in which Web services to be composed are described based on syntactical languages, and the composition is thus proceeded based on syntactic discovery, selection and composition.

In tradition, Web services are described based on WSDL, outlining what input they expect and what output they return. WSDL is an XML based language, which provides syntactic description of Web services. BPML and BPEL4WS are also XML-based standards to support syntactically composing Web services into business processes. The work in [KHAL03] proposes a Web service composition approach based on BPEL4WS, this work analyzes the compositional aspects of the BPEL4WS and identifies multiple interaction patterns, and support Web service composition lifecycle management.

The syntactic rule based approach for Web service compositions in [PUHK06] derives a given desired input/output type from a collection of available types of Web services using a prescribed set of rules with costs. A solution based on dynamic programming is presented to solve the minimal cost composition. The focus point of this work is in the case when only input-output type information from the WSDL specifications is available.

Generally speaking, the core problem of syntactic Web service composition is the syntactical matching, where output parameters of a Web service can be used as input parameters of another Web service. To solve this problem, many automatic Web service composition algorithms based on AI planning techniques have been proposed [BOZH09][KUXR09] [ZHAO10] [PAFL11][BEAS12].

However, the syntactic Web service composition was designed primarily for machine and program interpretation and use, the inherent meaning of information cannot be understood for computers and applications. In order to entirely automate Web service discovery and composition, a framework that semantically describes Web services is needed for human semantic and facilitating Web service composition.

2.2.2.2 Semantic Web service Composition Approaches

To overcome problems of interpretability and interoperability in traditional Web service compositions, the semantic Web is proposed as a list of standards to bring machine-understandable and human-transparent

descriptions to existing data and documents on syntactic Web services [MOZE11]. Semantic Web means adding machine processable semantics to data. With the help of well defined semantics, machines can understand information and process it on behalf of human users.

In semantic Web services, information needed to select, compose, and respond to Web services can be encoded with semantic markup at the Web service description. Semantic Web service composition is thus the process of composing semantic Web services and promises for providing the most suitable composite services to satisfy users' requirements.

A key element to realize the Semantic Web is to develop a suitably rich language for encoding and describing the Web content. Such a language must have a well defined semantics, be sufficiently expressive to describe the complex interrelationships and constraints between the Web objects, and be amenable to automated manipulation and reasoning with acceptable limits on time and resource requirements. RDF, WSMO, OWL-S are current standards to support semantic Web service composition as follows:

The Resource Description Framework (RDF) is a foundation for processing metadata and provides interoperability between applications that exchange machine-understandable information on the Web [ALHE11]. RDF extends the linking structure of the Web to use URIs to name the relationship between things as well as the two ends of the link (this is usually referred to as a "triple"). Using this simple model, structured and semi-structured data can be mixed, exposed, and shared across different applications.

The WSMO is used for describing the semantics of Web services [FEDO05]. It consists of four parts namely goals, ontologies, mediators, and Web services. Goal defines the user desires. Ontologies define formal semantics for the terms describing data to achieve interoperability among other WSMO elements. Mediator is used to handle interoperability problems between different WSMO elements while Web service part describes the functional behavior, precondition, post condition, control flow of an existing deployed service.

Since the WSDL standard operates at the syntactic level and lacks the semantic expressivity needed to represent the requirements and capabilities of Web Services, WSDL-S is proposed as a new standard to semantically annotate WSDL interfaces and operations as well as XML Schema types, linking them to concepts in ontologies [AFMN05]. The annotation mechanism is independent of ontology or mapping languages. WSDL-S keeps the semantic model outside WSDL, making the approach independent from any ontology language. However, without describing how to use of

annotations in different languages, it is rather difficult to formally define requests, queries or matching between service requests and service descriptions. As a result, WSDL-S may be successful in annotating WSDL with semantic information, but does not offer any support for automated service discovery and composition.

The OWL-S is almost the most popular standard to describe semantic Web services and support semantic Web service composition, as it overcomes the limitations of WSDL-S and provides extra support for semantic based Web service discovery and composition. OWL-S [ALHE11] is a high level language used for describing Web services properties which builds on top of earlier languages such as RDF and RDF Schema. OWL-S consists of a set of ontologies designed for describing and reasoning over service descriptions. It consists of three parts namely service profile, process model and grounding. Service profile includes general information and describes what Web services will do; process model describes how the service will perform functionally; grounding describes how to access Web services. In a more detailed perspective, a composite Web service can be viewed as a process, which is specified by a subclass of Service Model called process ontology. A process enables users to automatically discover, invoke, compose and execute Web services under certain conditions [MBHL04].

In addition to the standards describing semantic Web services, many approaches are proposed to semantically compose Web services to satisfy users' requirements.

Since BPEL4WS provides only syntactic support for Web service composition, a bottom-up semantic Web service composition approach is proposed in [MAMC03] to adapt BPEL4WS for the Semantic Web. This work collects service profiles into a repository and exploits their semantics to query for partners. Then, it integrates semantic services descriptions querying into BPWS4J, which is an engine that implements a subset of the features defined in the BPEL4WS specification. Since the BPWS4J is not extensible, they construct a Semantic Discovery Service to work within BPWS4J's perspective as an aggregator of Web services. However, this work requires users' intervention and does not support the automated integration and composition of Web services functionalities.

The Internet Reasoning Service (IRS-III) in [DCHS04] is a framework which supports the creation, publication, composition and execution of semantic Web service according to the WSMO ontology. D. Sell et al. introduce in [SHDM04] a graphical tool that supports users by defining dynamic compositions in IRS-III and recommending goals according to the context at each step of the composition process.

Roughly speaking, the IRS Web service composition is suitable for representing a service description as a process in OWL-S. However in IRS-III, the notion of goal refers to a general description of a problem and can be solved by different Web services. A goal describes a problem to be solved and represents the knowledge required for matching the problem to a set of Web service descriptions. In order to integrate OWL-S into IRS-III, the work introduced in [HDMC04] explain how ontologies describing a service in OWL-S specification (that uses the Process Model for modeling and describing Web services) are mapped to the WSMO ontology (that uses the notion of goal) and translated to be used by IRS-III. This work extends the potential of IRS-III in the sense that the separation of goals and Web services makes the Web service composition process more flexible.

Although such approaches provide solutions for semantic Web service composition, they still require user' specific knowledge of domain ontology to specify composition requirements, i.e., goals, and focus only on constructing composite services to satisfy users' functional requirements.

The Active Semantic Web Service (ASWS) [CHSA06] allow causal users to reason about their actions at runtime in order to compose autonomously and automatically Web services. In this work, the user interacts with an ASWS mediator that behaves as an agent asking for a set of requirements to search for the appropriate services that provide as a result one or more of the needed requirements. When ASWS receives the required action definition, it processes the next requirement. As a consequence, the user interacts with the mediator service that collects actions that enables it to satisfy user's requirements. This work provides a new means to help business users to specify their requirements and then automatically compose Web services to satisfy their requirements; however, the requirement specification process involves repeated interactions between users and the system (i.e., ASWS).

The Processes with Adaptive Web Services (PAWS) in [ACMP07] deploys the annotated Business Process Execution Language (BPEL) process with local and global QoS constraints on individual Web services or composite services. Through negotiations, the service retrieval module finds the best service that has the required interface and does not violate constraints on each task in the process. Multiple candidate services are selected for each task and for each process; only one candidate service is executed by the BPEL engine. The PAWS also allows faulty services to be replaced with other candidate services and provides the recovery actions to undo the results of the faulty services.

In [ZNBP08], a goal-directed service composition and optimization framework is presented. The Web service composition is expressed as

a goal directed problem that takes three inputs, namely domain specific composition rules, description of business objectives and description of business assumptions. The initial step in the composition process relies on a backward chaining where the composition rule creates a chain backwards from the business objectives until the initial state is reached and there are no more rules. The second step relies on a forward chaining, where some additional services are added to the composition schema produced during the first step to complete it. These steps deal with the control flow aspects of the composition. The final step relies on a data flow interface to add the data flow to the composition. The entire Web service composition process is supported by service ontology which specifies a common semantic model and defines concepts that are used by all participants in the process.

Roughly speaking, the idea of semantic Web services is to describe Web service interfaces semantically in a machine-readable manner, thus enabling automatic Web service discovery and composition. Compared to syntactic Web service composition approaches, semantic Web service composition has apparently its advantage in combining data and Web services from different sources without losing their meaning, interpreting meanings of Web services' input and output messages, and providing machine readable and human understandable composition solutions.

2.2.3 QoS-aware Web Service Composition Approaches

The Web with available Web services constitutes a growing and dynamic changing environment; more and more Web services with same functional attributes and similar QoS are published every day. How to discover and select component Web services from a list of service candidates with same operations and different QoS has become a key issue in Web service composition context. While some Web service composition approaches construct only composite services based on requirements on functional properties [GBNA08][LIN08] [KBBG08][KASE09], alternative approaches seek to provide composition solution satisfying functional requirements while taking into account composite services' QoS. These approaches are known as QoS-aware Web service compositions.

In the presence of multiple Web services with overlapping or identical functionality, users unavoidably discriminate Web service offerings based on their QoS. When constructing composite services, the QoS of the resulting composite service is a determinant factor to ensure user's satisfaction, based on QoS aggregation methods that calculate QoS values of composite services. For example, a QoS aggregation method using workflow patterns is introduced in [JARM04] to determine the QoS of a Web service

composition by aggregating the QoS dimensions of individual services. This allows verifying whether a set of services selected for composition satisfies QoS requirements for the whole composition. The QoS aggregation builds upon abstract composition patterns, which represent basic structural elements, like sequence, iterative, or parallel execution. This work models a Web service composition as a graph, and transforms it into a graph of composition patterns based on structural elements and QoS properties of individual Web services. A different approach of deriving QoS of composite services is proposed in [JAMG05] by following an aggregation approach and well-known workflow patterns defined in [DTKB03]. The authors analyze workflow patterns for their suitability and applicability to composition and they derive a set of composition patterns. Additionally, they define a simple QoS model consisting of execution time, cost, encryption, throughput, and uptime probability and QoS aggregation formulas for each pattern. The computation of the overall QoS is then realized by performing a stepwise graph transformation, that identifies patterns in a graph, calculates its QoS according to aggregation functions. The composition is repeated until the graph is completely processed and only one single node remains. The main disadvantage of these methods for QoS aggregation is that they do not consider the measurement scales of different QoS attributes when calculating the aggregated QoS values. For example, certain QoS attribute may be measured by different measurement scales, and thus requires different aggregation methods.

Since QoS preferences become one of the most crucial criteria for composing Web services, users may have different requirements and preferences regarding QoS of Web services. The QoS-aware approaches are therefore required to maximize the QoS properties of composite service by taking into account users' constraints and preferences.

The QoS-aware middleware for Web service composition in [ZBND04] presents a multi-dimensional QoS model consisting of price, duration, reputation, success rate and availability attributes. Each service is associated with a quality vector containing all QoS attributes for each operation of a service. The basic idea is to split a composition into multiple execution paths based on their notations that a composition is specified using a state chart diagram. The user is required to define an execution plan expressing for every task in the composition, a service exists that implements the operations required for that task. The QoS-aware composition lies in two steps: the local optimization and the global optimization. During the local optimization, the system tries to find all candidate Web services that implement the given task. Each service is assigned a quality vector and user defined scores for different quality constraints. These constraints are then

used to compute a score for each candidate service. Based on multiple criteria decision making, a service is chosen when it fulfills all requirements and has the highest score. The global optimization comes with integer programming that has variables, an objective function and a set of constraints as an input. The optimization problem is then solved using an integer programming solver.

Authors in [BRPB08] present Amadeus, a holistic service-oriented environment for QoS-aware Grid workflows. Amadeus considers user requirements, in terms of QoS constraints, during workflow specification, planning, and execution. Workflows and associated QoS constraints are specified at a high level using intuitive graphical notations. A user specifies the workflow with a UML-based Grid workflow modeling and visualization tool by composing predefined workflow elements. For each workflow element different properties may be specified that indicate the user's QoS requirements. A set of QoS-aware service-oriented components is provided for workflow planning to support automatic constraint-based service negotiation and workflow optimization.

QoS-aware composition is considered as one of the most important crucial issues for Web service composition since it determines the final performance of the resulting composite services and is directly related to the user satisfaction. Most of QoS-aware composition approaches especially those before mentioned in this section, work in static composition environment. Assuming that QoS properties are stable is very reductive. QoS properties are often subject to changes. In the next section, we discuss Web service composition from static and dynamic perspectives.

2.2.4 Static Composition and Dynamic Composition Approaches

As introduced in Chapter 1, composition environments, in which Web services are composed and executed, can be divided into two categories according to contextual information influencing the composition process: static composition environment and dynamic composition environment. In the static environment, contextual information in composition process rarely changes while in the dynamic composition environment, contextual information influencing composition process changes over time.

In composition environments, Web service composition approaches can also be classified into two categories concerning the design time and runtime during which Web services are composed and executed, namely static Web service composition and dynamic Web service composition. The main difference between static and dynamic compositions stems from the point of time at which a concrete atomic Web service is integrated into the

composition. Static approaches select services during the design time; whereas the dynamic approaches select services during the run-time.

According to [FOST04], "*Static Web service compositions are known at design time and are bound to a composition at design time. Dynamic Web service compositions are one or many compositions in which Web services are not known at design time, and which are discovered or their properties resolved based upon a criteria process set at run time.*"

2.2.4.1 Static Web service Composition Approaches

The before mentioned composition approaches, especially those are based on process modeling standards such as BPEL4WS and BPML, and Web service composition approaches that are based on AI planning are mainly static composition approaches, as they lack of proper support for contextual information and environmental changes, when constituting Web services suddenly become unavailable during the execution of composite services, the Web service composition should be regenerated.

In addition to the composition approaches based on modeling standards and AI planning technique introduced before, we cite Bea WebLogic [REF10] and Microsoft Biztalk [REF13A] as examples of static Web service composition, which are two platforms that compose Web services. BizTalk Server, which is both an application server and an application integration server, provides business process automation while Bea WebLogic is an application server to develop, compose, deploy and manage Web services and applications in large distributed environments. Microsoft Biztalk enables the automation of business processes, through the use of adapters which are tailored to communicate with different software systems.

Most of composition approaches work fine in static composition environments where Web services involved in compositions or other contextual information such as requirements not or rarely change. However, static Web service compositions are not flexible in the sense that they are not adaptive to the runtime changes such as service providers publish newer services, or services are replaced by other ones, current Web service compositions become inconsistent. In that case, it is unavoidable to reconstruct the composite service. Service composition approaches should be able to dynamically modify Web service compositions in simple and effective ways and adapt to dynamic environments.

2.2.4.2 Dynamic Web service Composition Approaches

Ideally, Web service composition should be able to transparently adapt to dynamic environment changes and to users' requirements with minimal or no human intervention. Dynamic Web service composition aims at overcoming the problems which are apparent in static Web service composition. It refers to compose complex services on the fly, in which services are composed at run-time and adapted to contextual information or environmental changes.

Most of existing dynamic Web service composition approaches rely on predefined abstract composition plans and service selection models at runtime. In this process, Web service composition and execution processes are interleaved in order to provide on the fly composition solutions.

An abstract composition plan includes a number of actions (usually represented as abstract Web services) to be achieved. It indicates the appropriate execution order of different abstract Web services by analyzing their functional properties. The selection phase aims at selecting atomic services for each abstract service and integrating them in the composition plan at runtime to construct specific composite service. The abstract composition plan is manually predefined by users [SING03] or automatically generated [YUZL07]. Dynamic composition approaches thus enable the automatic selection of atomic Web services at runtime and compose them to create executable composite services.

Some research efforts attempt to provide reference solutions for dynamic compositions. The approach in [MEHS00] proposes to build ad-hoc workflow by composing Web services. The general idea of ad-hoc workflow is to generate one template process in advance considering different possibilities, and then according to specific requirements, each composition task is derived from the template process that can be modified to meet different requirements.

The EFlow [CIJK00] is a platform presented by HP for the specification, enactment and management of composite services. A composite service is modeled by a graph that defines the order of execution among the nodes in the process. The graph is created manually but it can be updated dynamically. Three types of nodes are introduced (e.g., service, decision and event nodes). Service nodes represent invocation of atomic or composite services; decision nodes specify alternatives and rules controlling the execution flow; Event nodes enable service processes to send and receive several types of events. Although the graph is specified manually, the EFlow automatically binds the nodes with concrete services. The dentition of a service node contains a search recipe that can be used to query actual

service either at process design time or at runtime in order to provide dynamic composition solutions. However, the dynamic selection process only provides results satisfy functional requirements but they do not consider Web service non-functional properties.

The Self-Serv environment proposed in [BESD03] uses a P2P-based Web service composition model to enable the declarative composition of new services from existing ones, the dynamic selection of services, and peer-to-peer orchestration of composite service executions. The Web service composition is then manipulated by state chart, and the concept of service container is introduced to facilitate the composition of a potentially large and changing set of services: a container is a service that aggregates several other substitutable services that provide a common capability. During the composition process, Self-Serv postpones the decision of which specific service handles a given invocation until the moment of invocation. For each state (ST), Self-Serv generates a state coordinator, in which service providers are associated with the state ST hosts. At runtime, the coordinator of ST is responsible for 1) receiving notifications of completion from other state coordinators and determining from these notifications when to enter state, 2) invoking the service labeling ST, and 3) notifying the coordinators of the states that might need to be entered next that the service execution is complete.

A framework to construct composite Web services is proposed in [GRJA05] to compose Web services while taking into account both Web services' both functional and non-functional properties. An abstract composite model is firstly generated, containing all the necessary information that can be used in service discovery and selection. Suitable Web services are then handled by the discovery process based on the semantic descriptions matchmaking. At last, a finalized concrete composite service is obtained and executed. Another user requirements oriented dynamic Web service composition framework is introduced in [XQYB09], where dynamic Web service composition is preceded based on functional and non-functional requirements; With user requirements in OWL-S, the composition process firstly uses JSHOP2 technique [RAAN07] to generate an abstract service plan to satisfy user functional requirements. The Web service selection problem is then converted into a multi-objective optimization problem, using the multi-objective ant colony optimization (MOACO). The abstract service plan is concretized into a concrete workflow based on non-functional requirements, and then transformed into BPEL4WS.

In [KASE09] the authors present a composition framework allowing modeling and scheduling composite Web services under user constraints. The composition starts with a manual modeling of an abstract

workflow using a graphical interface depicting template of services which must be used. The framework finds concrete services to create a composite service fulfilling the user constraints. The advantage of this work is to compose Web services while considering the dependency relations between Web services. This work provides a good reference for modeling dependency relation between/among Web services.

A conceptual modeling approach for dynamic Web service composition is proposed in [GTSS11], in which the life-cycle for the dynamic composition of services is modeled in three phases, i.e., design, configuration and enactment phases. The design phase consists of modeling behavior and requirements using conceptual models. On the service provider side Web service interfaces and behaviors are modeled, and on the service user side requirements are specified by abstract service models. The configuration phase deals with the discovery, composition, selection and mediation of service candidates satisfying user requirements. In the enactment phase, an execution engine binds the abstract service model with selected services, executes the service calls and monitors the calls. During execution, it replaces failed service calls and performs auditing for QoS.

When an abstract composition plan is generated, the dynamic service selection is the important issue to deal with and the result of service selection directly affects the reusing and composition of services.

Authors in [SULY08] proposed an iterative selection algorithm for distributed environments which aggregate local optimum services to meet environment changes. The idea is to select one Web service from each group of Web service candidates independently while using a given utility function, values of different QoS criteria are then mapped to a single utility value and the service with maximum utility value is selected. The dynamic Web service selection algorithm based on markov decision process (MDP) in [FJWP10] considers the dynamic Web service composition problem as a dynamic optimization process, and the algorithm measures the credibility of services and the weight of QoS. By focusing on user's QoS preference, the preference based selection approach in [LASG07] determines the weight according to the maximum request utility for each composite service configuration, and combines declarative logic-based matching rules to overcome the lack of random selection of Web services. A multi-QoS based local optimal model of service selection (MLOMSS) is presented in [JIAN10] to provide important grounds to choose the best service, by using an ordinary utility function as a numerical scale of ordering local services. This work provides a solution to select Web service candidates based on users' preference and objective/goals.

These representative dynamic selection approaches are efficient in terms of computation time, but they cannot verify global QoS constraints for composite services, since Web services are selected individually and sometimes non-functional requirements cannot be satisfied.

In addition to these, man approaches seek to use local optimal strategy to filter candidate services for each abstract service, and then conduct service selection from the candidate services that have never been filtered out using global optimal policy to ensure non-functional requirements are satisfied both locally and globally.

In order to guarantee both local and global QoS constraints, an optimization approach for the composition of Web services is introduced in [ARPE06] to allow specifying QoS constraints both at local and global levels. The optimization is modeled as a mixed integer linear programming problem. In addition, Web service selection and Web services execution are interleaved. Authors in [LSYF09] propose a heuristic algorithm to optimize composition and service selection in order to meet the user's QoS expectations. The optimization is carried under two level constraints of global and local optimization, a mathematical model is given for each layer, using the convex hull frontier method to select Web services.

A QoS-based dynamic service composition approach for Web services with ant colony optimization is proposed in [ZCFJ10] to optimize the global utility function for QoS. If such a combination is found to form a path including Web services, the aggregated QoS of this path is regarded as optimal. The novelty of this work lies with the multi-objective optimal-path selection modeling for QoS-based dynamic Web service composition. Another work that combines ant colony algorithm and genetic algorithm together to provide dynamic Web service selection transforms the problem of selecting optimal Web services for composite Web service into selection of the optimal path in the weighted directed acyclic graph [YSLZ10] while the experimental results indicated the validity and more efficiency of this work.

Moreover, certain factors in the composition environment make the dynamic composition process tedious. In some case, many possible compositions exist instead of only one possible composition; the selection of compositions depends on the requirements and updated domain information. Two motivation scenarios are presented to describe this kind of problem [LKMC06][ZNBP08]. In this process, it is impossible to predefine every aspect of its composition schema due to the inherent complexity. It is extremely time consuming to enumerate all the possible options exhaustively. The control flow is determined by the data generated in real time. When abstract composition plan is difficult to generate in design time, this calls for dynamic generation of composite service plan based on domain infor-

mation. Authors in [LKMC06] propose an event-driven dynamic Web services composition for automation of business processes. Event-Condition-Action (ECA) rules are applied to generate the composition plan automatically and dynamically. Two types of ECA rules are defined as backward chain rule and forward chain rule. A backward chain rule specifies various pre-conditions (i.e. list of tasks that should be completed before a given task is initiated) for various tasks involved in the plan; while a forward rule is the form of an ECA rule where this work treats the completion of a task as an event and checks if the given condition is satisfied then executes the given action which is in turn execution of some other task. As an extension of this work, the dynamic composition and optimization proposed in [ZNBP08] couples the abstract composition plan and quality-driven selection approach in one single framework to ensure that the generated composite services comply with business rules. This approach can perform process adaptation and optimization in compliance with organization's business rules systematically. Compared to other dynamic composition approaches, this approach is able to produce composite services rooted in a rich process model. Nevertheless it still requires a predefined abstract process model as a reference to consider all possible compositions, and then generates or extracts an executable composite service from the predefined process model by selecting and integrating appropriate Web services. However, it is difficult to consider in practice all possible actions and their connections to provide a predefined abstract composition plan, and requires that users are familiar with composition domain, functionalities and connections of all domain actions.

Although many approaches deal with the dynamic Web service composition problem in the way of "selecting services after generating composition plan", we summarize the limitations of these approaches as follows:

1) Constructing composite services by selecting atomic services based on an abstract composition plans may limit the global QoS of composite services, because predefined composition plans without considering specific QoS profile may not provide the best way to aggregate QoS of atomic services;

2) These approaches lack of support to construct composite services without complete abstract composition plans, as in certain cases (e.g., to manage a crisis that occurs abruptly) a composition plan cannot be precisely predefined.

In sum, Web service composition approach which is able to adapt to dynamic environment while provide composition solutions with optimal QoS remains an open challenge.

2.2.4.3 Adapting Composition Approaches to Dynamic Environments

In addition to dynamic Web service composition approaches, some static Web composition approaches can be extended to deal with dynamic environment changes at runtime. In this context, we mainly focus on two research trends: Web service substitution and composition replanning.

The Web service substitution is the process of replacing failing constituting Web services in service compositions with new Web services; while the composition replanning is the process of re-adjusting and updating the Web service composition during runtime. As a result, some common Web service composition approaches have been extended and adapted to dynamic environment to deal with the following changes:

1) Constituting Web services become unavailable during the execution of composite services. Unavailable service case can be caused by different unforeseen reasons such as Web service overloaded, or disconnected for maintenance, etc.

2) Constituting Web services properties change/update during the execution of composite services. Changes in Web service properties may affect functional properties and QoS properties: when a Web service's functional property changes, the Web service becomes incompatible resulting composite service may no longer satisfy user's requirements; when a Web service's QoS changes, the global QoS of the resulting composite service is also updated, and consequently the resulting composite service is no longer satisfying user's non-functional requirements.

3) Any other endogenous/exogenous changes introduced in Section 1.3.

Although Web service substitution and composition replanning may not be effective when dynamic environments (i.e., available services, services properties) intensively changes occur frequently (e.g., every second), Web service substitution and composition replanning are able to adapt to the environment changes in most of the real cases. In addition, in order to provide efficient Web service substitution and composition replanning solutions when composition environment changes, monitoring mechanisms are necessarily required to monitor the execution of composite services and properties of all constituting Web services. When changes occur, the composition system is expected to detect and recompose Web services.

A similarity network is presented in [CHER13] for semantic Web services substitution. In the network, nodes are Web services operations and links join similar operations. The networks are built from a model that represents similarity relationships between Web services operations functionalities. Two operations are similar if they share common features re-

garding their input and output parameter sets. Four similarity measures based on the comparison of input and output parameters values of Web services operations are presented, and a set of functions that represent different degrees of similarity between operations are introduced. However, this works considers only functional properties when substituting a Web service but does not consider QoS properties.

A Web service substitution based on preferences over non-functional attributes is presented in [SABH09], where a preference networks is used for representing and reasoning about preferences over non-functional properties. Two variants of the service substitution problem, namely, context-insensitive and context-sensitive substitution are identified: the former assumes that the preferred substitution can be identified independent of the context, i.e., the other constituents in the composition, whereas the latter takes into account the context of the substitution. The Web service substitution is preceded by computing preferred substitution using a TCP-net model [BRDS06] of preferences over nonfunctional attributes.

In addition to these, authors in [VARB08] propose a semantic-based registry to implement a dynamic substitution of services in highly unpredictable contexts. A framework to reduce the complexity of service substitution is presented in [ATZI09], which organizes available services into groups and scales up with groups instead of services. Authors in [CCGP09] provides another runtime adaptation approach to bind each abstract service to a set of atomic services instead of one single atomic service, and selecting the most appropriate service to meet broader range QoS requirements.

Web service substitution cannot be effective in certain cases, however, in some other cases, substitution of unavailable/updated Web services can cause that the actual QoS values of new constructed composite service are against global QoS requirements; and in other cases, contextual changes may require the reconfiguration of Web service compositions, such as business logic change or users' decision change and thus the previous compositions are no longer valid. To avoid this kind of situations, it is necessary to reconsider and regenerate the service composition, i.e., to re-select the Web services to compose and to remake the composition patterns by which they are composed. Most of composition replanning approaches are based on AI planning techniques [HILM04][YAPZ10], as introduced in automated composition section, and are applied to regenerate the composite service when its execution is suspended. An OWL-S service composition planner is proposed in [KLGE05] called OWLS-Xplan, that allows for fast and flexible composition of OWL-S services in the semantic Web. Web

services and domain descriptions that are specified in the planning domain description language PDDL 2.1, and invokes an efficient AI planner Xplan to generate a service composition plan sequence that satisfies a given goal. Xplan extends an action based Fast Forward-planner with a HTN planning and re-planning component. These replanning approaches share the same limitations as composition based on AI planning.

A QoS-aware replanning approach is presented in [CPEV05] to trigger replanning opportunities during composite service execution. Replanning is triggered as soon as it is possible to predict that the actual service QoS will deviate from the initial estimates, and then the part of the service workflow that still has to be executed, will be determined and replanned by remaking the bindings between abstract and concrete services. The triggering algorithm proposed permits an early activation of the replanning, so to prevent risks as soon as possible. A proxy architecture is used to enable dynamic binding and replanning. Another replanning mechanism is proposed in [BSRH07] to adapt the execution plan to the actual behavior of already executed services by a dynamic service selection at runtime, and thus ensure that the QoS requirements will still be met. The replanning problem is modeled as an optimization problem, which is solved by a heuristic. However, these replanning approaches are still preceded from the perspective of selecting appropriate Web services and integrating them into a predefined workflow, and again have the limitation that abstract composition plan may limit the global QoS of composite services, which may influence users' satisfaction.

In conclusion, comparing to dynamic Web service composition, Web service substitution and composition replanning can enable static composition approach to be adaptive to dynamic environment changes in most cases, however, in order to maximize user's satisfaction, a more effective replanning approach is still required to regenerate composite services without predefined abstract composition plan and provide optimal replanning results with regard to QoS values.

2.2.5 A Brief Conclusion

Table 2.2 summarizes our Web service composition classification and gives a global view on the representative approaches introduced in this section.

Table 2.2 A Global View on Web service Composition Approaches

	Manual	Semi-automated		Automated	
		Static	Dynamic	Static	Dynamic
Syntactic	[BESD03] [KHAL03] [TASW03] [WADH03] [CCMN04] [ADHW05] [ACMP07](*)	[CASH03] [SIHP02](*) [ZAPG09](*)	[STRG03] [ALPA10]	[MCIL02] [KAKS07] [ZHYA08] [MEBE03](*) [PUHK06](*) [CASA12](*)	[CIJK00] [MEHS00] [BESD03] [HILM04] [CPEV05](*) [SULY08](*) [LSYF09](*) [ZCFJ10](*) [YSLZ10](*)
Semantic	[MAMC03]	[DIPW08](*)	[SHDM04] [CHSA06] [XCPM06](*)	[BCGH05] [MEWE06] [BOZH09] [ZHAO10] [PAFL11] [ZBND04](*) [WRGS07](*) [BRPB08](*) [LIN08](*) [ALRI09](*) [KUXR09](*) [BEAS12](*)	[LKMC06] [KAKG07] [GRJA05](*) [LASG07](*) [VARB08](*) [ZNBP08](*) [SABH09](*) [XQYB09](*) [YAPZ10](*) [GTSS11](*) [CHER13](*) [ZIVB13](*)

(*) means that this approach is a QoS-aware approach

From Table 2.2 we can see most of manual composition approaches are static and syntactic approaches; early researches on semi-automated and automated Web service composition also focuses on statically and syntactically composing Web services, and then with semantic Web and dynamic composition environment become more important, the research focus point has gradually switched from syntactic, static composition to semantic, dynamic composition; recent researches mainly focus on providing solutions for automated, dynamic, semantic and QoS-aware Web service composition. However, as introduced before, current automated dynamic Web service composition approaches rely on general composition plans to compose Web services and require users' high technical background to specific composition requirements, in this context, an ad-hoc Web service composition driven by business requirements to construct composite services without composition plan in dynamic environment is urgent needed to promote the popularity and adaptability of Web service composition. After survey-

ing the related work on Web service composition, in the following we investigate the related work on requirement models for Web service composition.

2.3 Overview of Requirement Specifications in Web Service Composition

A requirement is a singular documented physical and functional need that a particular product or process must be able to perform. The requirement is defined in [YOUN01] as "a statement that identifies a necessary attribute, capability, characteristic, or quality of a system in order for it to have value and utility to a user". Composition requirement refers to the constraints imposed on Web service composition process, and needs to be satisfied by the composition result. There are many different ways to document requirements. One common way is to use textual descriptions only; other ways to document requirements include use cases and customized document templates; for certain systems (e.g., Web service composition systems), requirements may even be documented as formal specifications.

In this section, we firstly introduce the general concept of requirement engineering, and then we analyze and survey current composition requirement models that are needed to initialize Web service composition process.

2.3.1 Requirement Engineering

Engineering refers to the creation of cost effective solution to practical problems by applying scientific knowledge [SHAW90]; while requirements engineering (RE) is defined in [NUEA00] as the branch of software engineering concerned with the real world goals for functions of and constrains on the software systems. It is also concerned with the relationship of these factors to precise specifications of software behavior and their evolution overtime and across software families.

The activities involved in requirements engineering vary widely, depending on the type of system being developed and the specific practices of the organization(s) involved. The typical activities for requirements engineering are introduced in [ZAVE97] including:

- Requirements inception;
- Requirements identification: to identify new requirements;
- Requirements analysis and negotiation: to check requirements and resolving stakeholder conflicts;

- Requirements specification: to document the requirements in a requirements document;
- System modeling: to derive models of the system, often using a notation such as the Unified Modeling Language;
- Requirements validation: to check that the documented requirements and models are consistent and meet stakeholder needs;
- Requirements management: to manage changes to the requirements as the system is developed and put into use.

These activities are sometimes presented as chronological stages although, in practice, there is considerable interleaving of these activities. Figure 2.7 presents a general process of requirement engineering.

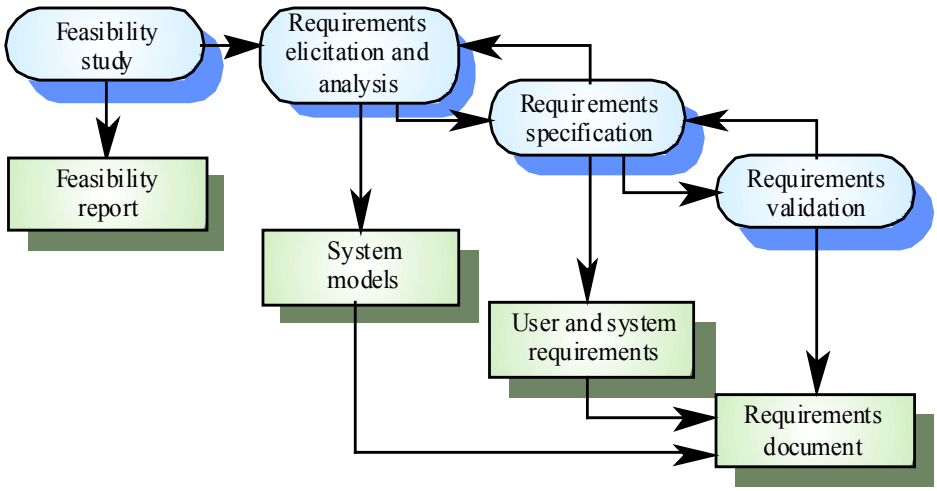

Figure 2.7 Requirement Engineering Process

There is a significant number of RE tools with different features for different requirement engineering purposes and activities. Here we briefly present several representative tools to support requirement engineering.

Giorgini et al. [GMMZ05] present a tool called ST-Tool for the design and verification of functional and security requirements, as security must be dealt with early on during the requirements phase. ST-Tool kernel has an architecture comprised of three major parts: Editor Module, Graphical-layer Manager Module, and Data-layer Manager Module. Editor module allows designers to edit secure models as graphs where nodes are actors and services, and arcs are relationships; graphical-layer manager module aims to manage graphical objects, and it supports goal refinement by associating a goal diagram with each actor; data-layer manager module is responsible for maintaining data corresponding to graphical objects.

Gregoriades and Sutclie [GRSU05] described a method and a tool called System Requirements Analyze (SRA) with which to validate non-

functional requirements in complex socio-technical systems. This tool can validate system reliability and operational performance requirements using scenario-based testing. Scenarios are transformed into sequences of task steps and the reliability of human agents performing tasks with computerized technology is assessed using Bayesian Belief Network (BN) models. The tool tests system performance within an envelope of environmental variations and reports the number of tests that pass a benchmark threshold. Moreover SRA is able to diagnose problematic areas in scenarios representing pathways through system models, assists in the identification of their causes, and supports comparison of alternative requirements specifications and system designs. It is suitable for testing socio-technical systems where operational scenarios are sequential and deterministic, in domains where designs are incrementally modified so set up costs of the BNs can be defrayed over multiple tests.

An empirical research to explore the use of mobile RE tools in practice is introduced in [MSGO06]. A mobile scenario tool Mobile Scenario Presenter (MSP) is presented to discover requirements directly in the user's work context. The MSP allows its user to discover and document requirements systematically in the workplace using structured scenarios. The MSP user walks through scenarios of future system behavior and observes current system behavior at the same time. The results from 3 evaluation studies demonstrate that these tools can support workplace requirements discovery and documentation.

Hall [6] presented the motivations for and problems with large scale scenarios, and a method called LSS (Large Scale Scenario), which uses automated and semi-automated techniques for description, maintenance and communication, with the use of large scale scenarios in RE. LSS helps acquire, represent, and use large scale scenarios in a way that is practical computationally and in human effort. Two application domains are used to illustrate the approach: live military training instrumentation and electronic mail servers and demonstrates the practical and beneficial use of LSS in architectural modeling of a complex, real-world system design.

A Requirement Engineering (RE) tool is presented in [JIEB07] containing a knowledge base to support RE process development and selection of RE techniques. The tool is built based on the Framework for Requirements Engineering pRocess dEvelopment (FRERE). The major merits of the tool over others is that the tool uses knowledge representation to manage the knowledge of the RE process and its technique, thus assisting development of the most suitable RE process for a software project.

Software development tools are presented in [LEPV10] which can be synchronized with RE tools for requirements collaborative access pur-

pose. This work suggests use collaboration tools all along the product life cycle to let developer work together, stay together, and achieve results together in software development process. The collaboration tools introduced in this work include version-control systems, trackers, build tools, modelers, knowledge centers, communication tools, and Web 2.0 applications.

Since requirement is a large subject and covers different topics, in our work, we focus only on requirement specification and provide models to help user specify composition requirements. Generally speaking, requirement specification methods can be categorized as:

- Structured natural language: standard forms, templates, decision tables;
- Program description languages: abstract features to specify requirements by defining an operational model;
- Requirements specification languages: special purpose languages with tool support;
- Graphical notations;
- Mathematical specifications.

2.3.2 Requirement Models in Web Service Compositions

In this section, we investigate composition requirement models of Web service composition approaches from three main categories of Web service composition approaches based on their techniques: Web service composition based on Workflow, Web service composition based on AI planning, and model-driven Web service composition.

We have presented the Web service composition approaches based on workflow [CASH03][BSRH07][BRPB08] and Web service composition based on AI planning [PEER05][STVV11] earlier in this chapter.

In Web service composition approaches based on workflow, users' requirements are mainly specified in terms of workflow that indicates necessary actions to be preceded and their logic execution order. When constructing/executing a composite service, atomic Web services are selected and invoked correspondent to each action defined in the workflow, and then the composite service is accordingly executed following the predefined execution orders. The specification of composition requirement in workflow-based approaches highly relies on the users' familiarity with each composition component' function as it requires users to describe all necessary actions and all possible execution relations among these actions.

Alternatively, Web service composition approaches based on AI planning or other automated composition approaches requires users to specify their composition requirements in different technical languages with re-

spect to different used planning/heuristic techniques; moreover, composition approaches based on AI planning requires pre-formalized domain information, and the domain information includes the descriptions of initial domain state, possible domain states, and different actions that can be performed.

Model-driven approaches for Web service composition approaches use models to describe user requirements, information structures, abstract business processes, component services and component service interactions. The models are independent of executable composition specifications, but can be transformed into executable composition specifications. The common of model-driven approach is to model the service orchestration by using a formal model such as UML activity diagram and then specify a transformation to executable workflow models, such as BPEL [DUHO01]. The work in [SKGS04] uses UML activity models to define compositions. In this work, existing WSDL specifications are transformed into UML models, which in turn are arranged to create new service composition; the composition can then be transformed into executable code such as WS-BPEL, and deployed on a workflow engine. The approach introduced in [GRJA05] entails separation of the fundamental composition logic from particular composition specifications (e.g., BPEL and BPML) in order to raise the level of abstraction. UML is used to provide a high level of abstraction, and to enable direct mapping to other standards, such as BPEL4WS. The Object Constraint Language (OCL) is used to express business rules and to describe the process flow. Business rules can be used to structure and schedule service composition, and to describe service selection and service bindings. The process of service composition development consists of service definition, scheduling, construction, and execution. The specifications of composition requirements in these approaches are mainly based on different UML diagrams: UML class diagrams are used to represent the state parts of compositions, i.e., Web service interface, QoS properties; while UML activity diagrams are used to represent the behavior parts, which describe the composition operations, interactions of component Web services, and control flow. Similar to composition approaches based on workflow, the specification of composition requirement still relies on the users' familiarity with each composition component' function and their composition relations; and the composition approaches share the same limitations as composition approaches based on workflow.

Web service composition approaches require users to specify composition requirements in a formal technical language, in order to compose Web services based on natural requirements, transformation approach-

es are required to transform natural requirements to formal requirements that are be directly used by Web service composition approaches.

Since most of composition approaches do not support requirement specification based on natural or structured natural language, we then survey the requirement transformation approaches that transform natural or structured natural language to other formats of technical languages.

2.4 Managing Requirements

Model transformation is one of the basic principles of Model Driven Architecture (MDA). To build a software system, a sequence of transformations is performed, starting from requirements and ending with implementation. Requirement transformation is the process of generation of a target requirement from a source requirement, according to a transformation description.

In this section, we firstly introduce the concept of Model Driven Architecture and Model Driven Development (MDD), and then present general requirement transformation approaches, at last we investigate the transformation approaches for Web service composition requirements.

2.4.1 Model Driven Architectures and Model Driven Development

MDA is an approach to using models in software development. The MDA is a specification that provides a set of guidelines for structuring specifications expressed as models. Using the MDA methodology, system functionality may first be defined as a Platform-Independent Model (PIM) through an appropriate domain specific language. Given a platform definition model such as CORBA, .Net etc., the PIM may then be translated to one or more Platform-Specific Models (PSM) for the actual implementation, using different domain specific languages, or a general purpose language such as Java, C#, etc. The translations between the PIM and PSMs are normally performed using automated tools, like model transformation tools [SISO09]. During the model transformation, the MDA prescribes certain kinds of models to be used, how those models may be prepared and the relationships of the different kinds of models.

Generally, the MDA follows four main principles as follows [BEBG05]:

1. Models expressed in a well-defined notation are a cornerstone to understanding systems for enterprise-scale solutions.

2. The building of systems can be organized around a set of models by imposing a series of transformations between models, organized into an architectural framework of layers and transformations.
3. A formal underpinning for describing models in a set of meta models facilitates meaningful integration and transformation among models, and is the basis for automation through tools.
4. Acceptance and broad adoption of this model-based approach requires industry standards to provide openness to consumers, and foster competition among vendors.

In order to give a concrete idea of the MDA concept, we illustrate two examples based on the MDA as follows:

Rewriting Systems: The theory of rewrite systems is of relevance to many areas of computer science, such as in the syntax description for programming languages and in the derivation of distributed algorithms. A rewrite system is a set of entities of type X→Y, where → is the so-called rewrite operator and X, Y text strings formed with constants taken from some alphabet. A entity is applied to a given string S in the usual way by replacing zero or more occurrences of the production's left-hand side in S with its right-hand side [RÖNN96].

Grammar Compilers: A grammar compiler is a computer program (or set of programs) that transforms source code written in a programming language (the source language) into another computer language (the target language, often having a binary form known as object code) [WONN07]. The most common reason to use grammar complier is to transform source code is to create an executable program.

2.4.2 General Requirement Transformation

The requirement transformation can be generally divided into three categories: transformation based on traceability [YUBL10], transformation based on behavior trees [DROM03], and transformation based on model transformation [AYWO09]. Requirements traceability refers to the ability to define, capture and follow the traces left by requirements on other elements of the software development environment and the trace left by those elements on requirements. Traceability is the ability to link requirements to corresponding analysis and design models, code, test cases, and other software artifacts. Requirement Behavior Trees are used to capture all the fragments of behavior in each individual natural language requirement by a process of rigorous, intent-preserving and vocabulary-preserving translation. The translation process can uncover a range of defects in origi-

nal natural language requirements. Model transformation is defines as the automatic generation of a target model from a source model, according to a transformation definition that consists of a set of transformation rules that together describe how a model in the source language can be transformed into a model in the target language. For each category, authors in [JASA12] list approaches to derive UML use cases and activity diagrams from other formalized requirements. This survey provides a good reference of general methodologies for requirement transformation, however, these approaches are designated to transform formalized requirements into other formalized requirements; and they do not provide solutions for structured natural requirement transformation. A systematic review is presented in [YUBL11] to examine existing works that transform textual requirements into analysis models. An analysis model is a description of what a system is required to do functionally, and aims to be less ambiguous and more correct and consistent than textual requirements [BRDU10]. As the direct automatic transformation of natural language requirements to analyze models is very difficult due to the inherent ambiguities of natural language, most of the research [GREM04][SAMK04][FKMS07] achieve a manual or semi-automated transformation and requires user's intervention during the transformation process; other automated methods [MICH96][SLFE04][SASO05] are only effective on natural requirements defined by given patterns.

Due to the difficulty on automating natural requirement transformation, alternative researches work on structured natural requirements (instead of natural requirements) based on SBVR, and provide transformation solutions. Authors in [AFBA11] present a solution that transforms SBVR requirements into UML class models; while authors in [RAPH08] proposes a more powerful method that transforms requirements based on SBVR into a set of UML diagrams, which includes Activity Diagram, Sequence Diagram, and Class Diagram. The focus point of these works is to derive UML diagrams from requirements based on SBVR and support future software development; however, they are not suitable for requirement transformation in the context of Web service composition: on one hand, as in the context of Web service composition, instead of developing new Web services to satisfy the initialized requirements, existing Web services with constraints are expected to be discovered and to be composed to satisfy the initialized requirements; on the other hand, although certain composition approaches take into account composition requirements in terms of UML diagrams, they share the same limitations with composition approaches based on workflow, and require user's technical background to specify the requirements.

2.4.3 Requirement Transformation for Web service composition

In the context of Web services, requirement transformations play an important role in Web service composition approaches, as they provide casual or business users with an appropriate requirement language to specify their needs and accordingly transform their requirements into requirement convenient to discover and compose Web services. In addition, Web service composition can be regarded as one special case of requirement transformation, as it connects business process and Web service together and transforms individual Web services into business process.

A framework for performing dynamic service composition is illustrated in [LÉSP08]. This framework contains four basic components as semantic analyzer, composition factory, property aggregator, and matcher to compose Web services by exploiting the semantic matchmaking between service inputs and outputs. Especially, this framework can take as input both natural language and formalized requirements: the natural language requirements are expected to be transformed into formalized ones using a built-in Semantic Analyzer, however, the transformation solution is not provided in the framework.

A Web service composition approach from natural language requirement is introduced in [BCVM06]. This work describes an approach to derive formal specifications of Web Service compositions on the basis of the interpretation of informal user requests expressed in restricted natural language. Each user request is processed against a natural language vocabulary that includes lexical constructs designed to convey the operations' semantics, in order to recognize and extract fundamental functional requirements implied by the request, and associate them to entries of known service operations. In addition, the request interpreter extracts from the request the overall service logic, expressed in terms of a set of modular templates describing control and data flow among the selected operations. The result is a composition specification that associates on demand each user request to a new composed service. That specification is formal and can thus be transformed in an executable flow document for a target service composition engine. This is an interesting work in Web service composition domain to compose Web services based on natural request; the idea of this work is to ask users to describe the whole composition based on natural language, and then to transform the composition in natural language to formal language for execution. However, this work still cannot solve our research problem which is to compose Web services based on business objectives instead of describing exact service composition; given user's busi-

ness-centric requirements containing business objectives in structured natural language, our objective is to discover what kind of Web services can satisfy the business objectives and how they are composed to satisfy user' business-centric requirement. Requirement transformation is indispensable to solve our research problems.

A model-driven based approach for Web service composition is introduced in [CAMS06]. Their paper presents MIDAS, a framework that proposes multi-level modeling of service compositions: composite services are specified using computation independent models, which are mapped (transformed) into platform independent models and in turn are again mapped into platform specific models. In MIDAS, platform specific models are represented using WS-BPEL. This work presents a good reference for Web service composition and requirement transformation, however, this paper only presents a general architecture but without any implementation details.

In addition, Web service discovery can be regarded a special case of transformation solution for composition requirement, as the objective of Web service discovery is to derive existing Web services from users' requirements in other forms.

Authors in [MUCH12] classify Web service discovery into two categories as syntactical discovery based on keywords and semantic discovery based on ontology concerning the language by which Web services are described. As counterpart to discovering Web services based on functional properties, approaches in survey [PHKH12] attempt to discover Web services based on functionality and QoS to satisfy both functional and non-functional requirements. A number of discovery approaches are introduced in to facilitate service discovery process, which can be generally summarized into three categories from the technologies used in Web service discovery process.

The first one is to discover services by enhancing requirement based on metadata. the approach presented in [PAAB07] expands user's requirement by combining ontologies and latent semantic indexing. The requirement vector is build according to the domain ontology while the description vector is built by extracting features from WSDL profiles. The discovery process is achieved by matching the description vectors against requirement vectors. Authors in [GCBG10] propose a Web service discovery approach for BPEL process. In this approach, user requirements are expressed as a service behavior model. BPEL specification is transformed to a behavior graph using flattening strategy, and the matching is transformed to graph matching problem.

The second category is to add additional tools in traditional discovery framework. A push model for Web service discovery is presented in [NAQA08] where service requesters are provided with service notification prior to discovery. The discover process is divided into two phases in a limited time period, i.e., subscription phase that allows user to subscribe specific requirements, and notification phase that newly registered Web services are added to match with user's requirements. Authors in [APRT06] present the VitaLab system to discover Web services based on indexing using hash table. They implement indexing on WSDL descriptions which are parsed using Streaming API for XML. The hash table maintains the mapping from each requirement into two lists of service names for request and response respectively, to get a list of services that satisfy a particular requirement.

The third one is to minimize total search area in service discovery process. Web service discovery based on keyword clustering and concept expansion is suggested in [ZZMX08], where similarity matrix of words in domain ontology is calculated and is used for semantic reasoning to find matching service. Through classification and subsumption of concept, the computing complexity is reduced. Authors in [WEHH10] propose to divide search in three layers by applying filters at each layer and thus minimizing search area. Three layers for service matching are service category matching, service functionality matching and quality of service matching.

These approaches are effective to discover Web services from user's requirements but they do not support to discover constraints on these Web services. By discovering Web services prior to the composition, the composition approaches still requires users' further manual specification of constraints on/among Web services. In addition, specifying constraints on Web services is closely related to WSDL profiles and QoS ontological-based descriptions, which make discovery processes inconvenient to causal users and their requirements in a real business context. Unfortunately, we do not find any work answering our research question which provides automated transformation solution to derive both Web services and multiple constraints from composition requirements that are defined in natural/structured natural languages.

2.5 Executive Summary

In this chapter, we organize and introduce the state of the art from three parts: Web service composition approaches, composition requirements models and composition requirement transformation approaches. For

each part, we introduce representative models or approaches in current researches, and then analyze and conclude their limitation. Consequently, we come out with the conclusion that an ad-hoc Web service composition approach driven by structured natural language requirement is required to compose Web services without predefined composition plans in dynamic environments.

Firstly, we analyze a large number of existing Web service composition approaches, and classify them into different categories from different views influencing composition process. From the view of composition automation degree, Web service composition approaches can be classified as manual, semi-automated and automated Web service composition; from the view of semiotics in Web service composition, Web service composition approaches can be classified as syntactic Web service composition and semantic Web service composition; from the view of composition agility, Web service composition approaches can be classified as static Web service composition and dynamic Web service composition; in addition, a large number of Web services with same functionalities and similar QoS profiles has become available on daily basis, QoS-aware composition approach is indispensable to compose Web services while taking into both functional and non-functional requirements.

Automated composition approach is able to automatically generate the entire composite service without human involvement, however, current automated composition approaches highly rely on users' technical background to specify composition requirement, and also requires preformalized domain description, which make them not suitable for business users who prefer to specify their composition requirement in terms of business objectives; a Web service composition approach driven by business requirements is thus required to automatically discover and compose Web service to satisfy users' initialized requirement. Moreover, dynamic composition environment imposes more constraints on Web service composition approaches, dynamic composition approaches are able to dynamically modify the Web service composition in a simple and effective way when composition environment changes, however, dynamic composition approaches requires predefined abstract composition plans to construct composite services, which may not provide the best composition result with regard to global QoS; alternatively, Web service substitution and composition replanning are able to adapt to the environment changes in most of the real cases, thus based on the two measures, common Web service composition approaches can be adapted to dynamic environment. A more effective replanning approach is still required to regenerate composite services without

predefined abstract composition plan and provide optimal replanning results with regard to QoS values.

In order to compose Web service driven by business requirements while overcoming the limitations in current researches, we propose an automated, semantic QoS-aware Web service composition approach in dynamic environment which will be introduced in detail in Chapter 5: in order to provide the optimal composition result with regard to QoS value, instead of selecting services after generating a composition plan, our approach simultaneously selects atomic services and infer their composition patterns to ensure that services are composed in the best way with regard to QoS values; in order to better adapt to changes in the environment, we apply a heuristic to compose Web services while provide our approach with Web service substitution and composition replanning mechanisms. Our Web service composition approach is able to take into account control flow constraints, QoS constraints, dependency constraints, as well as user's preference when composing Web services while provide an optimal Web service with regard to QoS values to satisfy users' requirements and maximize user's satisfaction.

And then, we analyze current composition requirement models that are needed to initialize Web service composition process; since most of composition approaches do not support requirement specification based on natural or structured natural language, we then survey the requirement transformation approaches that transform natural or structured natural language to formal technical languages. By analyzing existing requirement transformation approaches, we conclude that the requirements that are transformed in these approaches are actually defined from the same perspective by use of different languages or models; if requirements are defined from different perspectives, these approaches lack the ability to identify the connections between requirements from different perspectives, and thus cannot provide the transformation solution. In our research context, a requirement transformation solution is required to transform business-centric requirements in terms of business objectives to composition requirements directly manipulated by Web service composition describing which Web services can satisfy business objectives and how they are expected to be composed.

Aiming at overcoming different limitations introduced above, we present our contribution in the following chapters, an ad-hoc Web service composition approach driven by structured natural language requirements in dynamic environments.

Chapter 3

A Three-level Requirement Model for Ad-hoc Web Service Compositions

Introduction 74
Related Work 76
General Process for Crisis Management 78
Three Levels of Composition Requirements 79
The Business-centric Requirement Model 82
The Capability-focused Requirement Model 88
The Rule-driven Web Service Composition Model 92
Conclusion 100

Abstract: In order to provide current SOA with business level support to allow either business or technical users to express their requirement models, we build our resilient SOC by use of a three-level requirement model to focus on requirements at different levels, i.e., the business-centric requirement model, the capability-focused requirement model and the rule-driven Web service composition requirement model. Since our multi-level requirement approach relies on the capability model as an intermediate model for requirements transformation, we survey related works that attempt to model Web service capabilities and highlight their limits and drawbacks with respect to our requirement approach. We then develop the three levels of requirements by providing definitions and general specification formalisms. Since the ultimate goal of our requirement models is to guide the Web service composition process from casual user perspective, we develop an end-to-end requirement transformation process in the next chapter to transform business-centric requirements into capability-focused requirement model and finally generate rules to represent Web service requirements to be used in ad-hoc Web service composition approach in dynamic environments.

3.1 Introduction

A requirement is a singular documented physical and functional need that a particular product or process must be able to perform. A requirement is often defined as "a statement that identifies a necessary attribute, capability, characteristic, or quality of a system in order for it to have value and utility to a user" [YOUN01].

Many different ways to document requirements range from high-level abstract statements of services or system constraints to detailed mathematical specifications. The common ways to specify requirements are to use textual descriptions, UML use cases, customized document templates, or formal specifications [YUBL11].

Composition requirements refer to constraints imposed on the Web service composition process, and need to be satisfied by composite Web services. Generally, composition requirements do not only describe Web services to be composed, but also include constraints on/among these Web services (e.g., control flow constraints, functional and non-functional properties constraints, and dependency constraints, etc.).

Current SOA lacks of the support for business users who prefer to express the composition requirements in business language and expect to accordingly discover and select Web services. And consequently, most of existing Web service composition approaches [DUSC05] such as workflow techniques and Artificial Intelligence planning focus on "how-to-do" to compose Web services together and highly rely on the fact that users' composition requirements are defined in formal languages to describe necessary actions and set constraints on these actions. However, composition approaches still focus on the technical level and require domain-specific knowledge on Web services to specify requirements. Nevertheless, business users prefer to describe their business objectives and requirements to be achieved in natural-like languages such as SBVR rather than technical languages. The gap between user's business-centric requirements describing business objectives and composition requirements describing technical details and properties of Web services makes Web service composition unpractical in real context of usage.

In order to fill the gap between business-centric requirements and rule-driven Web service composition requirements, we introduce a Web service capability model as an intermediate model to connect business-centric requirements and rule-driven Web service composition requirements together, and based on the capability model we model composition re-

quirements in three levels: business-centric requirement, capability-focused requirement and rule-driven Web service requirement. Figure 3.1 presents the three-level requirement model within the context of our proposed rSOC.

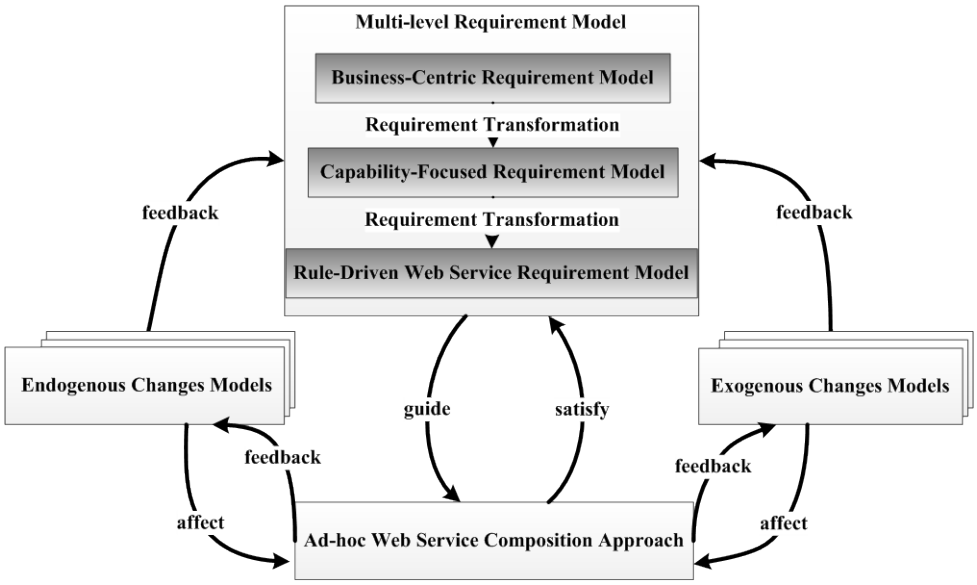

Figure 3.1 The Conceptual rSOC with Requirement Model

From Figure 3.1, we can see that the three levels of composition requirements are connected in a top-down way by requirement transformation processes. Any new requirement at the business level or any contextual changes occurred in the environment will be propagated and impacted the Web service composition model, which consequently guides an ad-hoc composition without any predefined composition plan. In this chapter, we introduce each requirement model and define formalisms to specify them. In Chapter 4 , we discuss the transformation approach and introduce the ad-hoc composition approach in Chapter 5 .

The concept of capability denotes what an action does in terms of changes in the state of known or unknown entities that are involved in an interaction. Our motivation use a capability model as intermediate model is that capability describes what Web services can do without describing all details related to functional and nonfunctional properties, and consequently the capability model enables to automatically discover and compose Web services that perform particular kinds of operations in particular contexts; A cornerstone of Service-Oriented Architecture is that capabilities consolidate Web services and business processes [MLMB06]. Nevertheless, service capability modeling or capability description raises a challenge on how to meaningfully capture different levels of abstractions from a functional perspective and keep the balance between machine accessibility (e.g., tech-

nology support, machine readable, …) and human understandability (e.g., expressiveness of goals, human readable, …).

In this chapter, we firstly survey the research work on capability model, and respectively introduce our three-level requirement model for ad-hoc Web service composition.

3.2 Related Work

Since we use capability model as an intermediate model for requirements transformation, we survey the related work on Web service capability modeling in this section.

A capability is a functional description of a Web service describing constraints on the input and output of a service through the notions of preconditions, assumptions, post conditions, and effects, and interfaces that specify how the service behaves in order to achieve its functionality [RBML06]. As an emerging field to model Web services, the concept of Web service capability has not drawn the research community attention as it deserves. Current approaches for capability modeling are in fact part of efforts for describing related concepts such as business processes, service descriptions and search requests [BHDZ12]. Thus most of current capability models are created only based on the functionalities and QoS of Web services, and are not practical to facilitate either requirement modeling or service discovery process. The difference between services and Web services are described in [KLPT04][PREI04] as:

Web services are means to find or buy services. For example, a user may want to travel from Innsbruck to Venice and he is looking for a service that provides this to him. The service may be provided by an airline or a train company. This is the service he is looking for. In order to find (and buy) the service he is accessing a Web service, which will not provide him the service to travel from Innsbruck to Venice (he needs a plane or a train) but it may help him to find this service [KLPT04]. A Web service is a computational entity, which is able, by invocation, to achieve a goal. A service, in contrast, is the actual value provided by this invocation [PREI04]. Based on this, Web service capability should be modeled as the real ability of a Web service instead of the operations a Web service can carry out.

Focusing on the existing Web service capability models, one popular theme to model Web service capability is based on the IOPE paradigm, in which the Web service capability is described as constraints on inputs and outputs through the notions of pre-conditions and effects [PKPS02]

[KVBF07]. By such, the Web service capability is modeled as a functional description of a Web service describing constraints on the input and output of a service through the notions of preconditions, assumptions, post conditions, and effects, and interfaces that specify how the service behaves in order to achieve its functionality. The capability is syntactically [PKPS02] or semantically [KVBF07] described based on ontology, which is part of a larger service ontology. However, the main focus point of these capability models are the operations that Web services can carry out from an offer perspective. In this context, they are not expressive enough to represent what Web services can really perform and to be connected with business-centric requirements defined from a request perspective.

Capability models based on case-frame are interesting alternatives to IOPE models since they describe abilities that services can perform. The case-frame based representations are one of the most expressive and flexible means to represent the capabilities of intelligent agents [GERH99] [WICK00]. The case frame provides a convenient way of structuring the description of what behaviors, actions or capabilities a service provides. Roughly speaking, the capability model is expressed as action verbs and informational attributes, structuring the description of what behaviors or actions a service provides.

A Web service capability model based on case-frame is introduced in [OAHE03] to describe what services can do. The capability model emphasizes on action verb and informational attributes to provide enough information for users to identify and locate alternative services without human intervention. A capability is described by an action verb that expresses what the capability does. To allow for the fact that different verbs may be used to express the same action, synonyms and ontological source are provided. The ability to provide alternatives to the primary verb assists similarity matching of capabilities. An extended capability model is proposed in [BHDZ12] where the Web service capability is modeled as an action verb and a set of attributes and their values supported by domain ontology. This capability model specifies three types of informational attributes depending on the source of their values, i.e., consumer or context attributes, provider (Pro) attributes, consumer then Provider attributes. In general, three sources are involved in a scenario of capability consumption. In addition, semantic links as generalization and specification relations are introduced between different capabilities. The meta model is implemented based on RDF and makes use of Linked Data to define capability attributes as well as their values.

Modeling Web service capability based on case-frames shows more expressiveness to describe the real ability of a Web service. However, we argue that describing service capabilities only based on action verb and attributes are oversimplified and causes ambiguity without explicit descriptions of the domain by which service capabilities are manipulated. A more expressive Web capability model is required to represent different levels of abstraction, to capture business-centric requirements, and derive potential Web services and constraints reflecting the business aspects that the capability can satisfy.

3.3 General Process for Crisis Management

In order to facilitate the presentation of our three-level requirement model for Web service compositions, we briefly introduce a motivation to present a general process of crisis management in a dynamic environment as shown in Figure 3.2.

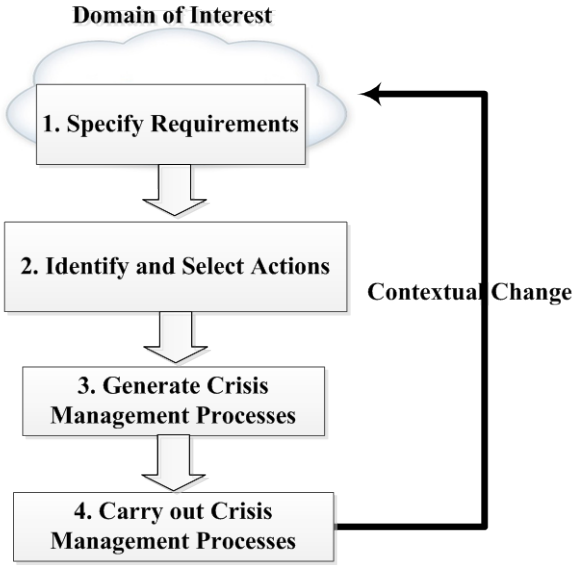

Figure 3.2 A General Crisis Management Process

As illustrated in Figure 3.2, when a crisis occurs, the first step is to specify crisis management requirements after assessing and evaluating the crisis domain; the second one is to identify and select necessary actions based on crisis management requirements; the third step is to build crisis management process, enabling different actions working together; the fourth step is to deploy the crisis management process to reduce and resolve negative events.

During the execution of the crisis management process, the domain is a dynamic environment and objective to changes that may occur inside or outside the crisis management process, such as other negative facts

are caused, or certain actions are not available due to other emergencies. In such environment, crisis management process needs to be reconfigured and adapted to contextual changes. Since current SOA does not support dealing with continuous and unpredictable changes, extending SOA with extra capabilities is required to build resilient applications in dynamic environments.

Based on this scenario, some actions examples are necessary to build up the crisis management process on the fly. Information collect in the context of crisis are often incomplete and the actions to perform to manage the train crash crisis are listed as follows:

1) Alert Public: to alert public about the crisis;
2) Evacuate Population: to evacuate the population around the crisis spot;
3) Transport Victim: to transport victims to the hospital or other medical institutions;
4) Assist Victim: to assist victims to receive medical treatment;
5) Extinguish Fire: to extinguish the fire on the crisis spot;
6) Recover Electricity: to recover the electricity supply on the crisis spot;
7) Clear Site: to clear the crisis site;
8) Repair Railway: to repair the railway on the crisis spot.

In the following sections, we firstly provide conceptual models of requirement and then illustrate examples based on our crisis management.

3.4 Three Levels of Composition Requirements

Since the ad-hoc Web service composition is the keystone to develop the resilient service oriented architectures adaptable to dynamic environments, we introduce a three-level requirement model to simultaneously capture requirements at business level and Web service level to allow either business or technical concerns to be expressed from different perspectives.

In this section, we model composition requirements in three levels with increasing technical details, i.e., **Business-centric Requirements**, **Capability-focused Requirements**, and **Rule-driven Web Service Requirements**. This model provides a reference for composition requirements modeling and allows either business or technical people expressing their composition requirements from different perspectives.

We introduce each level from its definition and general requirement specification methods in this section, and illustrate our proposed formal requirement model in the following sections.

3.4.1 The Business-centric Requirements (BCR)

Definition: the business-centric requirement model refers to business objectives and rules that are expressed in a natural structured language to describe users' business actions to be performed. Actions are concretized with composite Web services build on-the-fly and validate by users.

Requirement Specifications: the natural language is often used to express business activities and concerns. The natural language is any language which arises in an unpremeditated fashion as the result of the innate facility for language possessed by the human intellect [LYON91]. The natural language is typically used for communication, and can be spoken, signed or written. However, expressing Web service composition requirements based on natural language (NL) is difficult due to three main reasons [SOMM07]:

1) Specifications that writers use to express their intentions may not be exactly understood by readers. This leads to misunderstandings because of the ambiguity of natural language (synonyms, anonyms, ….);

2) Specifying natural language-based requirements is over flexible, as same concepts can be completely interpreted in different ways;

3) Requirements written in natural language are difficult to modularize (encapsulate), which makes tracing the effects of a requirements change difficult.

We specify the business-centric requirement model, which later will derive the Web service compositions, with the Semantics of Business Vocabulary and Business Rules (SBVR). The SBVR, as a standard of the Object Management Group (OMG), is intended to be formal and detailed natural language declarative descriptions of complex entities, such as businesses [HEND05]. It allows formalizing complex compliance rules, such as enterprise operational rules, security policies, standard compliance, or regulatory compliance rules. Our choice of the SBVR to model requirements is motivated by two main reasons.

1) The SBVR is a formal business rule language with a natural language interface, which implies that either casual users or domain expert can express their requirements with natural language-based expressions.

2) Compared to other semantic ontology languages or business rules (i.e., Description Logic and Semantic Web Rule Language), the SBVR provides structured natural language-based vocabularies to define rich business vocabularies and specify rules by reducing ambiguity caused by the natural language.

The business-focused requirement model specifies what a user wants to do with a structured English language based on the SBVR, and represents the primary actions by which they define the operative way to reach their objectives and perform their actions. The business-centric requirement model and its specifications are discussed in section 3.5.

3.4.2 The Capability-focused Requirements (CFR)

Definition: The capability-focused requirement model refers to Web service capability expressed based on what Web services are can do functionally without provide technical details and being less ambiguous and more consistent than textual requirement specifications.

Requirement Specifications: Currently Web services are described based on technical languages, such as WSDL and OWL-S, by which Web services are defined from the perspective of offering, i.e., what kind of operation Web services can carry out. Descriptions are only based on Web service operations and, unfortunately, these technical specifications cannot really describe the real ability that Web services can perform. The difference between Web service operations and Web service capabilities is extensively discussed in [KLPT04] as " *[...] Web services are means to find or buy services. For example, customer may want to travel from Innsbruck to Venice and he is looking for a service that provides this to him. The service may be provided by an airline or a train company. This is the service he is looking for. In order to find (and buy) the service he is accessing a Web service, which will not provide him the service to travel from Innsbruck to Venice (for this he needs a plane or a train) but it may help him to find this service*." Based on this example, the Web service operation "BookAirlineTicket" provides an access to the travel service, or the "capability of travelling".

In addition, current Web service description languages, such as WSDL, only focus on Web service operations (e.g., BookAirlineTicket) but they fail to present the real ability (e.g., capability of traveling) that a Web service can perform. The gap between user's objective descriptions (e.g., travel) and Web service's operation descriptions (e.g., BookAirlineTicket) requires to extend Web service descriptions with a capability model to describe the real ability that Web services can perform instead of describing their operations. As we have chosen to build the Web service capability model as an intermediate to fill the gap between business requirements and Web service composition requirements, we develop the Web service capability model to describe what Web services can do, what are their required resources, and their states, which reflect changes in the real world effects,

and what are the business objectives that Web services have to achieve [LIBB13A].

The capability-focused model and its specifications are introduced in Section 3.6.

3.4.3 The Rule-driven Web Service Requirements (WSR)

Definition: The rule-driven Web service requirement model refers to Web service composition constraints that are expressed based on functional and non-functional properties as well as multiple constraints among Web services.

Requirement Specifications: Since Web service requirements directly used to guide the Web service composition process, we specify on Web service requirements from two perspectives based on rules:

1) Requirements on Web service functional and non functional properties;

2) Multiple constraints on relationships among Web services participating in the composition process.

We develop an extensible Web service composition model based on a set of rules to specify multiple constraints on/among Web services (i.e., control flow constraints, Quality of Service constraints, and dependency constraints). Without loss of generality, new composition rules can be added and are directly took into account in the Web service composition approach presented in Chapter 6 .

The rule-driven composition model mainly consists of three categories of rules:

1) The structure rules define constrains on the control flow between Web services that must be respected by the Web service composition process.

2) The local and global rules define constrains on non-functional properties that participant Web services and composite Web services must be respected during the Web service composition process.

3) The dependency rules represent constraints on relationships that may exist between Web services.

The rule-driven requirement model and its specifications are introduced in Section 3.7.

3.5 The Business-centric Requirement Model

As before-mentioned, the business-centric requirement model is built based on a subset of the SBVR [FAYA13]. The SBVR defines semantics for vo-

cabulary and rules in a business domain of interest. It is a common formal rule-based language that facilitates the collaboration among customers, business experts, and IT specialists. Before introducing and specifying business requirements, we briefly introduce SBVR to illustrate its main characteristics.

3.5.1 Introducing SBVR

The SBVR defines structured-based natural languages for documenting semantics of business vocabularies, business facts, and business rules. It contains linguistic terms for conceptual modeling and captures expressions as formal logic structures. The SBVR vocabulary allows one to formally specify representations of concepts, definitions, instances, and rules of any knowledge domain with structured set of terms in natural language that reduce ambiguity cased by free text in natural languages.

Roughly speaking, SBVR consists of two parts, i.e., SBVR facts and SBVR rules. **SBVR facts** are defined based on *noun concepts* and *fact types* (i.e., simply relations among noun concepts) whereas **SBVR rules** are defined based on *modal operators*, *quantifiers*, *qualifiers* and SBVR *facts*. Table 3.1 summarizes SBVR concepts and provides some examples. In this table, we underline nouns and instances, italicize the *fact types*, and double-underline the modal operators, quantifiers, and qualifiers. In addition, SBVR supports standard logical operators such as “and”, “or”, “if”, and “not” just to mention a few.

The SBVR business vocabularies define noun concepts, fact types, and individual instances of both noun concepts and fact types;

Noun concepts describe terms, representing classes. Noun concepts form class hierarchies via subtype relationships, providing basis for subsumption reasoning. In the example of “An ambulance *is a kind of* vehicle.” Ambulance and vehicle are both noun concepts. The instance “ambulance” plays the role of “vehicle” in the SBVR rule “At least 20 vehicles should *transport* victims”.

Fact types describe identify relationships among one or more roles. Every fact type has a fixed number of roles, which are the kinds of things that can participate in the corresponding relationships.

Unary fact types capture aspects of a single instance, such as "Fire *is extinguished*".

Binary fact types capture relationships among two roles, such as "Fireman *extinguishes* fire".

Ternary and larger arity fact types are also possible, such as “Fireman *extinguishes* fire *before* electricity *is recovered.*”

Table 3.1 Major Features of SBVR Meta-model

Semantics of Business Vocabulary and Business Rules	
Business Vocabulary	**Business Rules**
Noun Concept: classes e.g., fireman, fire **Fact Types**: relationships among noun concepts e.g., fireman *extinguishes* fire **Instance** e.g., Paris *is* a Place	**Modal Operators** e.g., necessity, possibility, … **Quantifier** e.g., each, at least one, … **Qualifier**: additional fact types e.g., the date of the crisis

SBVR rules are defined based on SBVR facts with *modal operators*, *quantifiers*, *qualifiers*, and *condition*s as follows:

Modal operators are defined by the use of alethic and deontic modalities from the world of philosophy and logic [HALP06].

The alethic modalities include for example:

it is necessary that …

it is possible/impossible that …

They enable structural rules such as:

"It is possible that a victim *is assisted by* more than one doctor."

The deontic modalities include for example:

it is obligatory that …

it is permitted/forbidden that …

They describe behavioral rules such as:

"It is obligatory that at least 10 firemen *extinguish* fire."

The primary distinction between structural rules and behavioral rules is that structural rules define characteristics of a model itself and thus cannot be violated, whereas behavioral rules specify expectations of users with the understanding that such expectations are not always met.

Qualifiers are defined to qualify different roles in the fact type, such as "more than", "at least", etc. For example, a SBVR with qualifier is "It is necessary that total response time *is* less than 4h."

Quantifiers are defined to quantify different roles in the fact type, such as "of", 'the", etc. For example, of the SBVR with qualifier is "The place of the crisis *is* Paris".

In conclusion, SBVR facts are defined based on noun concepts and fact types (i.e., simply relations among noun concepts); while SBVR rules are defined based on SBVR facts and modal operators with optional quantifiers and qualifiers.

3.5.2 The Business-centric Requirement Model

We model user's business-centric requirements by using a subset of SBVR terms. Based on the motivation scenario as an example, we show how users express their business requirements. The business-centric requirement model (BR) includes three categories of constrains, namely Functional Requirements (**FR**), Non-functional Requirement (**NR**), and the Contextual information (**CTT**). The business-centric requirements are thus incorporates functional concerns, non-functional concerns and contextual information related to a specific domain of interest as follows: BR = FR ∪ NR ∪ CTT

Figure 3.3 shows the UML class diagram of the business-centric requirement model, including different categories of constraints expressed with SBVR terms and vocabularies.

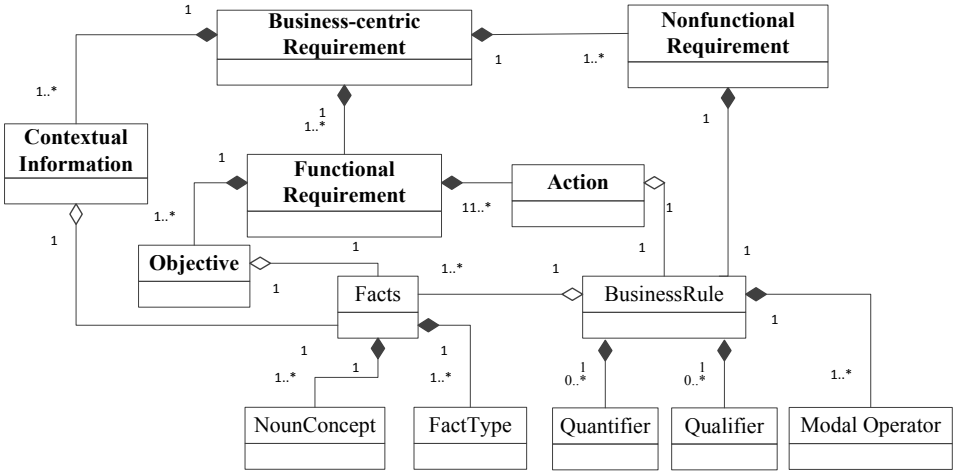

Figure 3.3 The UML class diagrams of the Business-centric Requirement Model

3.5.2.1 Functional Requirements

Functional requirements describe users' general objectives and necessary actions that are expected to be carried out. **FR** consist of two parts:

a) **Objective** descriptions include a list of ultimate objectives (**obj**) based on SBVR facts in order to describe users' general goals to be achieved, such as:

FR.Objective = {obj$_1$, obj$_2$, ..., obj$_n$}, where *obj$_i$* is one of the objectives to achieve.

Functional requirements can thus have one or more objectives. For example, in the crisis management crisis scenario, the objective , "*to manage train crisis*", can be described as: FR.Objective = {*Manage train crisis*}.

b) **Action** descriptions includes a list of actions to be performed in response to contextual information or internal/external events generated by the dy-

namic environments in order to support the objectives such as: FR.Action = $\{a_1, a_2, \ldots, a_n\}$, while a_j information generated by endogenous or exogenous changes in the environments and expressed in terms of SBVR fact types.

For example, in the crisis management crisis scenario, expected actions can be described in terms of "Fire must *be extinguished*", "Victims must *be assisted*", etc.By such, FR.Action = {"Fire must *be extinguished*", "Victims must *be assisted*."}

3.5.2.2 Non-functional Requirements

Non-functional requirements (NF) describe different constraints on/among actions (**nf**) that are expected to be carried out such as:

$NF = \{nf_1, nf_2, \ldots, nf_n\}$, where nf_i is a non-functional requirement expressed based on SBVR rules, specifying a list of constraints.

We introduce three categories of constraints such as :

a) **Control-Flow Constraints** which indicate the expected control relationship or execution order between different actions. They are also defined with SBVR rules and fact types such as "*before/after*", "*while*", and "*if ..., then ...*" such as:

i) The "before" and "after" fact types denote a sequential relationship between two different actions and indicate that an action is directly followed by the application of another action.

In this example, "It is obligatory that the victims *are assisted after* the victims *are transported*." The action *assist* is followed by the action *transport*.

ii) The "while" fact type denotes a concurrent relationship between two actions and indicates that the application of one action can be occurred with the application of another action.

In this example, "It is necessary that the victims *are being transported while* the fire *is being extinguished*, both actions, *transport* and *distinguish*, are performed concurrently.

iii) The "if…, then …" fact type denotes a condition execution order between two actions and indicates that the application of one action relies on the successful application of another action.

In this example, *If* the fire *is extinguished, then* the electricity *must be recovered*, the action *recover* should be accrued out whether the action ex*tinguish* is successfully performed.

b) **Property constraints** refer to expected properties or attributes associated with actions and they are defined based on SBVR rules and quantifiers, such as "more than" and "at least" to annotate actions with additional attributes:

nf_1: It is obligatory that at least 10 firemen *extinguish* fire

nf_2: It is obligatory that total response time *is* less than 4hours

nf_3: It is obligatory that the cost *of* assisting victims *is* less than 2000€.

c) **Dependency Constraints** indicate dependency relationships between actions. They refer to implicit relationship between different actions defined by means of SBVR operator such as "maybe" and "must not". For example, two actions A, and B can be related with dependency relationship patterns such as:

If *A* is selected, *B* may be selected.

If *A* is selected, *B* must not be selected.

3.5.2.3 Contextual Information

The contextual information (CTT) describes the domain of interest with facts and additional information such as date, location, etc. The set of contextual information, CTT, is expressed with SBVR facts such as:

CTT = {ctt_1, ctt_2, …, ctt_n}, where ctt_i denotes a domain state

The following examples express contextual information with SBVR notations:

ctt_1: Crisis place *is* Pairs.

ctt_2: Crisis date *is* 2013/03/01.

Table 3.2 provides an example of business-centric requirements related to the motivation scenario.

Table 3.2 Business-Centric Requirement Examples

FR.Objective	**obj_1**: *Manage* train crisis
FR.Action	**a_1:** Fire must be extinguished. **a_2:** Victims must *be assisted*. **a_3:** Railways must *be repaired*. **a_4:** Electricity must *be recovered*.
Non-functional Requirement	**nf_1:** It is obligatory that at least 10 firemen *extinguish* fire. **nf_2:** It is necessary that total response time *is* less than 4 hours. **nf_3:** It is necessary that the cost *of* assisting victims *is* less than 2000€. **nf_4:** It is obligatory that the electricity *is recovered after* the fire *is extinguished*.
Contextual Information	**ctt_1**: Crisis place *is* Pairs. **ctt_2**: Crisis date *is* 2013/03/01.

3.5.3 Domains of dynamic environments

Since SBVR is a domain specific language, the domain should be predefined by domain experts in order to support the specifications of business requirement based on SBVR as well as support later business requirement transformations into Web service composition requirements. In the motivation scenario, the specific crisis domain, i.e., the train crash crisis, should

be described by domain experts with SBVR terms independent from any particular crisis, place, time and different kind of actors.

To this end, we define the Domain, which mainly describes the dynamic environment by defining different concepts and their relationships. Table 3.3 illustrates the Domain definition for the train crash crisis environment with SBVR terms, fact types, rules and modalities.

Table 3.3 Domain for a Dynamic Environment

Noun concepts:
(**Crisis**) traffic crisis, accident, terrorist attack, train crash, fire, …
(Person) fireman, doctor, victim, repairman, electrician, policeman, …
(Organization) fire brigade, hospital, EDF, rail station, police station
(**Vehicle**) train, car, ambulance, plane, taxi, …
(**Facility**) railway, …
(**Location**) city, airport, school, hospital, …
(**Date**) MM/DD/YY
(**Properties**) response time, price, cost, …
Facts Types:
(**attributive**)
Fire brigade has firemen.
Hospital has doctors.
EDF has electrician.
Rail station has repairmen.
(**unary**)
A crisis is observed.
(**binary**)
Web service provides capabilities.
Capability responses to crisis.
Fireman distinguishes fire.
Doctor assists victims.
Repairman repairs railway.
Electrician recovers electricity.
Cleaner cleans site.
Concept Deviation:
A victim is an injured person.
A doctor is a person who assists victim.

3.6 The Capability-focused Requirement Model

As discussed early in this chapter, current capability models attempt to specify Web service capabilities in terms of they can do. Unfortunately, they fail to pro-

vide sufficient expressiveness to capture both user's business objectives in dynamic environments and facilitate the discovery of appropriate Web services with respect to business requirements imposed by business objectives. In this section, we introduce the three-layer capability-focused model to formalize business requirements terms of Web service capabilities and enable the discovery of Web services before building ad-hoc composite Web services in dynamic environments [LIBB13B].

The three-layer capability-focused model describes Web service capabilities from different perspectives, namely request, offer, and composition perspectives. By such, it plays an important role as an intermediate in the multi-level requirement models proposed in the context of resilient service-oriented architectures. The capability-focused model consists of three layers, corresponding to these perspectives namely: 1) the *capability objective layer*, which describes business objectives that Web service capabilities should achieve; 2) the *capability profile layer*, which describes capabilities as *action-verb* and *noun* pairs, as well as attributes, representing abilities that Web services can perform; 3) the *inter-capability composition layer*, which establish relationships among Web service capabilities to facilitate the generation of rules used at the rule-driven Web service composition model.

The Web service capability model, **CAP**, is defined as a tuple such as: CAP = <id, OL, PL, CL>,

where *id* denotes an identifier related to this Web service capability, *OL, PL* and *CL* are respectively the capability objective layer, the capability profile layer, and the inter-capability composition layer.

Capability instances, denoted as *cap*, are specific examples created following our proposed capability model. Capability instance are mainly created by service providers or domain experts, and they are associated with Web services. A capability instance can be associated with one or more Web services.

Figure 3.4 shows the detailed UML class diagram of the capability-focused model that we discuss its components in the following sections.

3.6.1 The Capability Objective Layer

Connecting business requirements and objectives specified by business users with Web service capabilities leads us to model Web service capabilities from the "request perspective" to describe what kind of objectives can be achieved by enabling Web service capabilities. An objective can thus be achieved by one or more capabilities each of which is semantically interlinked to other capabilities by means of *composition relationships* as described in the inter-capability composition layer.

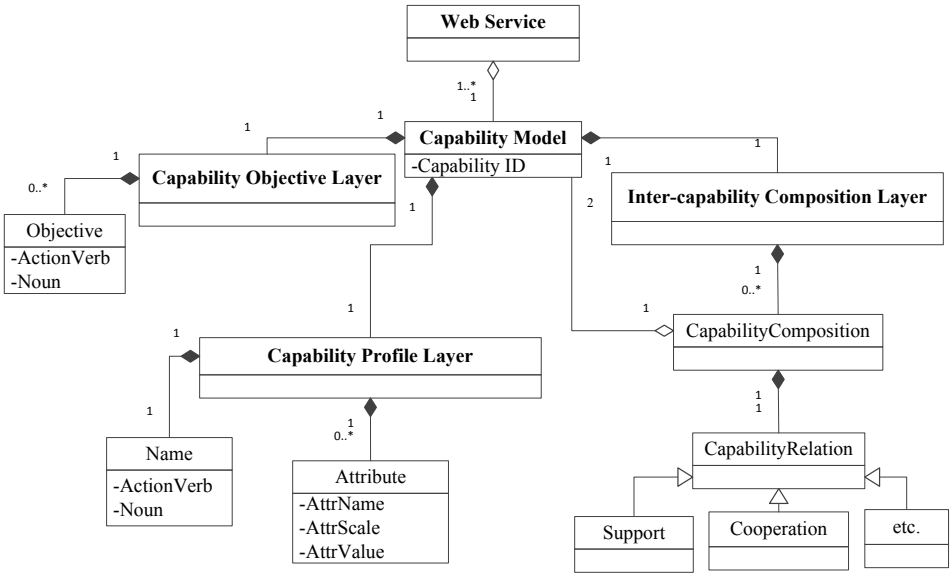

Figure 3.4 The UML Class Diagram of the Capability-focused Model

Roughly speaking, the capability objective layer consists of one or more objectives (**obj**) described by *ActionVerb* and a *Noun* pairs [OAHE03] expressed in SBVR terms. The ActionVerb concept is introduced in [OAHE03] to model Web service capabilities in a natural language in order to define what is the action being applied by Web services. In our context, the *ActionVerb* and *Noun* pairs define Web service capabilities from the request perspective to present which objectives that a capability can satisfy.

In order to facilitate the matching between objectives, and consequently between verbs and nouns, we associate domain specific ontologies to action-verbs and nouns to ensure semantic interoperability and share similar concepts with different names. The objective layer, *OL*, of the capability-focused model, CAP, is defined as:

CAP.OL = {obj_1, obj_2, …, obj_n}, where each objective obj_i is defined as obj_i = <$action\_verb_i$, $noun_i$>.

An objective can be associated with different capabilities while one capability can have achieve multiple objectives. For example, the objective of the "repair railway" capability, can be "manage accident", or "maintain facility". In this case, CAP.OL = {<manage, accident>, <maintain, facility >}.

3.6.2 The Capability Profile Layer

The capability profile layer is defined in terms of a capability *name* (**cname**) and a list of *attributes* (**attr**), describing the real ability and the business features that a Web service performs. The capability profile layer PL is defined as

$$CAP.PL = \{cname, attr_1, \ldots, attr_n\}$$

In a similar way to the capability objective layer, the capability name is defined as a pair of an ActionVerb and a Noun. However, the capability name describes the real ability that the Web service can perform whereas the capability objective describes the objective that the capability can satisfy.

Each attribute $attr_i$ is defined in terms of attribute name (**attr_name**) and attribute value (**attr_value**). The formal presentation of the capability name and attributes are:

$$cname = <action\_verb_i, noun_i>,$$
$$attr_i = <attr\_name_i, attr\_value_i>,$$

where $attr\_name_i$ is the name of attribute, and $attr\_value_i$ is the value of this attribute.

For example, a Web service capability to extinguish fire can provide maximum 40 firemen for one task at the same time, its capability profile layer is defined as CAP.PL = {<extinguish, fire >, <AvailableFiremen, 40>}.

3.6.3 The Inter-capability Composition Layer

In order to extend the Web service capability with new capabilities, we establish semantic links between existing Web service capabilities to "integrate" together. These links enable different capabilities to be "composed" in order to satisfy new business requirements. We add the inter-capability composition layer in the capability model. The inter-capability composition layer consists of different inter-capability compositions (**cc**) describing relationships between existing capabilities. The inter-capability composition layer is defined as:

$CAP.CL = \{cc_i, cc_2, \ldots, cc_n\}$,where each capability composition, cc_i, is defined as a tuple such as $cc_i = <cap_p, cap_q, rel, p>$,

The cap_p and cap_q denote two Web service capability *ID*; the *rel* is the type of binary relationship associating two capabilities; the *p* is the relationship property associated to the capability composition, which can be either *static* or *dynamic*.

The static property means that the capability composition is persistent and predefined by Web service providers or domain experts whereas the dynamic property means that the composition is automatically and temporarily generated based on user's business requirements.

We identify five types of composition relationships among pair of Web service capabilities as follows:

1) The *Cooperation* relationship indicates that two capabilities are requried to achieve a task, satisfying certain requriements. For exmaple, in order

to extinguish a big fire, the *extinguish fire* capability and *check building map* capability are required together.

2) The *Support* relationship indicates that the applicaiton of one capability depends on the application of another capabilty. For example, in order to assist victims to receive medical treatement, victims should be firstly transported to the medical treatement centers.

3) The *Condition* relationship describes conditionel situation among capabilities, which means that the execution of one capability in this relation depends on the sucessful execution of its previous capabilty. For example, the electricity should be recovered only after the fire is extinguished.

4) The *Promotion* relationship recommends that two capabilities should be applied together to satisfy a requirement since they provide extra benefits whether they combine together. For exmaple, the *transport victims* capability provided by one hospital is more efficient to be applied the *assist victims* capability that is provided by the same hospital.

5) The *Competition* relationship describes that two capabilities are somehow similar and one of them is enough to satisfy a business requirement or objective. For example, several capabilities to assist victim to receive medical treatments with different hospitals are some redundant and in competitive with each other.

Table 3.4 illustrates the capability-focused model with its layers through examples extracted from the motivation scenario. Each capability model is associated with one or more Web services and domain specific ontologies. Based on Web service capability model, the capability-focused model (CFR) for Web service compositions is modeled as a list of capability instances (cap_i) :

$$CFR = \{cap_1, cap_2, \ldots, cap_n\}$$

3.7 The Rule-driven Web Service Composition Model

The Web services composition process relies on requirements and constraints defined on functional and non-functional properties related to composite Web services and Web service participants in composite Web services. The composition process is the keystone upon which we build resilient service oriented architectures, which are capable of adapting and surviving to endogenous and exogenous changes. To this end, we focus on two fundamental characteristics of the composition process as being ad-hoc and adaptable to changes. In order to provide a flexible Web service composition process dedicated to ad-hoc compositions without predefined composition plans, we define a rule-driven composition approach, including a set of various rules to express these constraints. The rule-based approach also provides flexibility to easily extend the

rules or add new rules. It also provides adaptability to changes caused internally and externally by the dynamic environment in which Web services operate.

Table 3.4 Web Service Capability Instances based Capability Model

ID	Capability Objective Layer	Capability Profile Layer		Inter-capability Composition Layer
		capability name	attributes	
cap_1	{<response to, crisis>}	<alert, public>	{(AvailableRegion, France)}	
$cap_{2.1}$	{<manage, crisis>}	<evacuate, population>	{(AvailablePlace, France)}	
$cap_{2.2}$	{<manage, crisis>}	<evacuate, population>	{(AvailablePlace, Marseille)}	
cap_3	{<manage, crisis>, < rescue, people>}	<transport, victim>	{(MaxTransportNumber, 10)}	{cap_3, cap_4, Support, static}
cap_4	{<manage, crisis>, <rescue, people>}	<assist, victim>	{(MaxAssistNumber, 300)}	{cap_3, cap_4, Support, static }
$cap_{5.1}$	{<manage, crisis>}	<extinguish, fire>	{(AvailableFiremen, 40)}	{$cap_{5.1}$, $cap_{5.2}$, Cooperation, dynamic} {$cap_{5.1}$, cap_7, Support, static}
$cap_{5.2}$	{<manage, emergence>}	<put out, fire>	{(AvailableFiremen, 70)}	{$cap_{5.1}$, $cap_{5.2}$, Cooperation, dynamic} {$cap_{5.2}$, cap_7, Support, static}
cap_6	{<manage, crisis>}	<clear, site>	{(AvailableDate, 24/7)}	
cap_7	{<manage, accident>}	<recover, electricity>		{$cap_{5.1}$, cap_7, Support, static} {$cap_{5.2}$, cap_7, Support, static}
cap_8	{<manage, crisis>, <maintain, facility>}	<repair, railway>		

As we will explain in Chapter 5 , the composition approach hinges on heuristics and multi-objective constraints, which are best expressed in term of rules. Each rule is intended to focus on a specific aspect of the composition process and its environment (e.g., control flow constructs, QoS properties, business, information security, fault tolerance, monitoring, dependability, etc.). By such, rules connect the Web service composition process to various internal and external variables in the composition environment. These variables affect and are affected by each other. They can be collected together into different models with respect to their concerns such as business requirements, security objectives, tolerance policies etc. To this end, the rule-driven composition model is somehow connected to various models described by various variables to observe and update (e.g., business model, security model, tolerance model, …). Generally speaking, the rule-driven composition model is affected and affects all these models, which lead us to the fundamental idea of developing the resilient service oriented architecture. In sum, the rule-driven composition model is a central model upon which the ad-hoc composition and the adaptability to changes are

guaranteed through transformation mechanism between models based feedback and control loops.

In this section, we only focus on the Web service requirement model and the specification of its rules to set multiple constraints on Web service properties and among Web services participating in the composition. In Chapter 5 , we introduce the ad-hoc composition approach.

3.7.1 Specifying Web Service Requirements

We define different types of Web service (generally denoted by s_i) as follows:

1) **Abstract Service**: An abstract service $\mathbf{a_i}$ represents a task to be achieved.

Set of abstract service: $\mathbf{W_a}$ is denoted as the set of abstract services.

$$\mathbf{a_i} \in W_a,\ W_a = \{a_1, a_2, \ldots, \}$$

2) **Atomic Service:** An atomic service $\mathbf{t_i}$ represents a specific service to achieve a task.

Set of atomic service: $\mathbf{W_t}$ is denoted as the set of atomic services.

$$\mathbf{t_i} \in W_t,\ W_t = \{t_1, t_2, \ldots, \}$$

3) **Composite Service:** A composite service $\mathbf{c_i}$ is the atomic services composed in a logic order to achieve tasks.

Set of composite service $\mathbf{W_c}$: W_c is denoted as the set of composite services.

$$\mathbf{c_i} \in W_a,\ W_a = \{c_1, c_2, \ldots, \}$$

The set of Web services for composition is denoted as **W**, which is the union of W_a, W_t, and W_c: $W = W_a \cup W_s \cup W_c$. $\mathbf{s_i}$ denotes a generic service from W_a, W_t, or W_c.

Each Web service $\mathbf{s_i}$ is defined by its service profile (**SP**) consisting of Identifier(id), Input(I), Output(O), Operation(Op), and QoS(Q).

$$SP = <id, I, O, Op, QoS>,$$

where,

- *id* is the identifier of a Web service.
- $I = \{i_1, i_2 \ldots\}$, is a set of input messages $\mathbf{(i_i)}$ of a Web service.
- $O = \{o_1, o_2 \ldots\}$, is a set of output messages $\mathbf{(o_i)}$ of a Web service.
- $Op = \{op_1, op_2, \ldots\}$ is the set of operations $\mathbf{(op_i)}$ of a Web service. The operation of a Web service is defined as: given the required input messages, the execution of the service generates the correspondent output messages.
- $QoS = \{Q_1, Q_2, \ldots\}$ is the set of QoS attributes $\mathbf{(Q_i)}$ of a Web service.

3.7.2 Composition Patterns and Composite Services

In order to represent the composition relations between different Web services, we identify four types of composition patterns, which cover most of the structures specified by composition languages or workflow patterns [MMVL05][WADH02]:

1) **Sequence Pattern**: sequential execution of s_i and s_j, denoted by "$s_i \triangleright s_j$", which means s_j is executed after the successful execution of s_i.

2) **Parallel Pattern**: parallel execution of s_i and s_j, denoted by "$s_i \parallel s_j$", which means s_i and s_j are triggered and executed at the same time.

3) **Selection Pattern**: selective execution of s_i and s_j denoted by "$s_i \oslash s_j$", which means s_i or s_j is executed in a composite service.

4) **Iteration Pattern**: iterative execution of s_i, denoted by "$!s_i$", which means s_i is iteratively executed in the finite times.

The set of composition patterns is denoted as CP, CP = {$\triangleright$, $\parallel$, $\oslash$, !}.

In our context, we denoted a composite service as $\mathbf{c_i}$; each composite service c_i is regarded as a pair of atomic or composite services composed together following one composition pattern, and we use the binary tree model to represent and manipulate composite services. The reason why we choose binary tree model to represent composite services is to facilitate the service decomposition process which will be introduced in detail in chapter 6.

The formal expression of a composite service c_i is defined as: $c_i = \langle s_L, s_R, cp \rangle$, where s_L and s_R are the left and right sub-tree representing two services, *cp* indicates the composition pattern between two sub trees, $cp \in$ {$\triangleright$, $\parallel$, $\oslash$, !}. Figure 3.5 shows the binary tree presentation of three composite services.

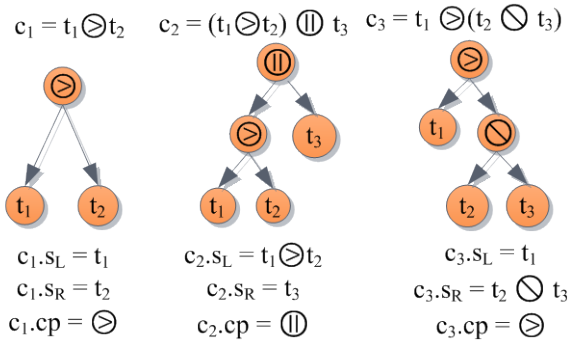

Figure 3.5 Composite Services as Binary Trees

3.7.3 Composition Rules

In order to model multiple constraints in the Web service composition process, we structure different constraints in terms of **Composition Rules**. We identify four categories of composition rules namely **Structure Rules**, **Constraint Rules**, **Dependency Rules** and **Matching Rules**.

3.7.3.1 Structure Rules

A structure rule represents constraints on the control flow in the Web service composition process. A structure rule consists of two Web services and their composition patterns to progressively construct potential composite services.

We use two subscripts to denote a structure rule, and a structure rule is denoted as $r_{s.i}$, where the subscript *s* denotes that this composition rule is a structure rule while the subscript *i* denotes the identifier of this structure rule. The set of structure rule is denoted as R_s,

$$R_s = \{r_{s.1}, r_{s.2}, \ldots, \}, r_{s.i} \in R_s$$

A structure rule $r_{s.i}$ is defined as a triple,

$$r_{s.i} = <s_l, s_r, cp>, r_{s.i} \in R_s, R_s \subseteq W \times W \times CP$$

where, s_l and s_r are the two sub services in the structure rule, $s_l \in W$, $s_r \in W$, and *cp* indicates the composition pattern between the two services, $cp \in CP$. We introduce an example of a structure rule $r_{s.1}$ as

$r_{s.1}$ = <ExtinguishFire_AWS, RecoverElectricity_AWS, ⊜>

The example given above means that the Web service to recover electricity should be sequentially composed with the Web service to extinguish fire.

We also use the binary tree model to manipulate structure rules as shown in Figure 3.6: the leaf nodes of a structure rule are abstract/atomic/composite services, while the inner and root nodes are composition patterns. Since we use the same model to represent composite services and structure rules, apparently we can 1) construct a composite service following a structure rule; 2) generate a structure rule following composite service.

We identify two sub types of structure rules as follows:

-Abstract Structure Rule: An abstract structure rule is a structure rule in which all of its constituent services are abstract services. An abstract structure rule is denoted as $\mathbf{r_{sa}}$.

-Concrete Structure Rule: A concrete structure rule is a structure rule a structure rule in which all of its constituent services are atomic or composite services. A concrete structure rule is denoted as $\mathbf{r_{sc}}$.

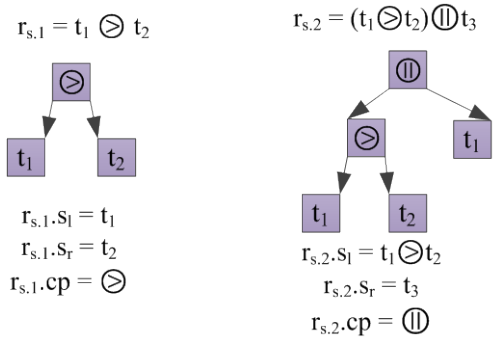

Figure 3.6 Structure Rules as Binary Trees

3.7.3.2 Constraint Rules

A constraint rule represents the constraints on the value of Web service QoS attributes in the Web service composition process. A constraint rule gathers QoS constraints related to individual atomic services and entire composite services.

We use two subscripts to denote a constraint rule, and a constraint rule is denoted as $r_{c.i}$, where the subscript *c* denotes that this composition rule is a constraint rule while the subscript *i* denotes the identifier of this constraint rule. The set of constraint rule is denoted as R_c,

$$R_c = \{r_{c.1}, r_{c.2}, \ldots, \}, r_{c.i} \in R_c$$

The set of constraint rule is composed as,

$$R_c \subseteq QoS \times P \times V,$$

where *QoS* is the set of QoS attributes to be constrained, *P* is the set of operators $P = \{=, \neq, >, \geq, <, \leq, \supseteq, \not\supseteq\}$, *V* indicates the set of the constraint values of QoS attributes.

We specify two sub types of constraint rules and individually give its formalization.

-Local Constraint Rules: a local constraint rule is a constraint rule which describes the constraint on a QoS attribute of single abstract service a_i or atomic service t_i. A local constraint rule is denoted as r_{lc}, and is defined as

$$r_{lc.i} = < s_i.Q_j, p, v >$$

where $p \in P$, $v \in V$, $s_i \in W_a \cup W_t$. $s_i.Q_j$ is the QoS attribute *j* of service s_i.
We introduce an example of a local constraint rule $r_{lc.1}$ as

$$R_{lc.1} = < AssistVictim.price, <, 200euros>$$

The example given above means that the price of the Web service to assist victims should be less than 200 euros.

- Global Constraint Rules: a global constraint rule is a constraint rule which describes the constraint on a QoS attribute of all composite services. A global constraint rule is denoted as r_{gc}, and is defined as

$$r_{gc.i} = < Q_j, p, v >,$$

where $p \in P$, $v \in V$. Q_j is the QoS attribute *j* of all composite services.

We introduce an example of a global constraint rule $r_{gc.1}$ as

$$r_{gc.1} = \text{<crisis\_response.response\_time, <, 2h>}$$

The example given above means that the response time of the composite service should be less than 2 hours.

3.7.3.3 Dependency Rules

The loosely-coupled is a fundamental property of the service oriented architecture and seek to increase reusability of Web services in different composition scenarios. Most of Web service discovery and Webs service selection assume that Web services are independent from each other and loosely coupled. However, this assumption is very reductive since implicit relationship may exist between Web services that are participated in previous composition (collaboration relationships) or have different functional or non-functional properties (competition relationships). Without loss of generality, Web services may depend on each other and provide new insight to discover, select and compose Web services. In addition, dependency among Web services may be imposed by the business requirements. Thus, there has been an increasing research interest in composing services based on social networks to provide service composition models. Dependency relations help to connect individual Web services together and select optimal Web service in the Web service composition process.

For these reasons, we introduce the service dependency rules (e.g., collaboration, substitution, exclusion, etc.) to connect services together and enable Web service compositions with new capabilities. A dependency rule represents the dependency constraints among Web services and is denoted as $r_{d.i}$, where the subscript *d* denotes that the rule is a dependency while the subscript *i* denotes the rule identifier. The set of dependency rules is denoted by R_d such as : $R_d = \{r_{d.1}, r_{d.2}, \ldots, \}$, $r_{d.i} \in R_s$;

Each dependency rule $r_{d.i}$ is defined as a triple

$$r_{d.i} = \text{<}t_l, t_r, dp\text{>}, r_{d.i} \in R_d, R_d \subseteq W_t \times W_t \times DP,$$

where t_l, t_r are two constituent atomic services in the dependency rule, $t_l \in W_t$, $t_r \in W_t$, and *dp* indicates the dependency relation between the two services, *DP* is the set of dependency relation, $dp \in DP$.

We identify three types of dependency relations namely **"Optimal Composed"** denoted as "⊞", **"Excluded"** denoted as "⊗", and "**Substituted**" denoted as "◉", hence, $DP = \{⊞, ⊗, ◉\}$.

-Optimal Composed (⊞): Two atomic services t_i and t_j have the dependency relation of "⊞" means that if t_i and t_j are composed together, they will together provide additional benefits (e.g. lower price, lower costs) or satisfaction.

-Excluded (⊗): Two atomic services t_i and t_j have the dependency relation of "⊗" means that t_i and t_j are excluded in a composite service.

-Substituted (◉): Two atomic services t_i and t_j have the dependency relation of "◉" means t_j can be substituted by t_i in a composite service.

Substituted dependency relation is special case of excluded relation. For each atomic service t_i in W_t, we calculate the QoS similarity between t_i and its excluded services to find the most similar atomic services as the substituted services of t_i. The calculation of QoS similarity is introduced in the Chapter 5 .

3.7.3.4 Matching Rules

The execution of Web services requires some benefits to be sacrificed (e.g., money to be paid) or some value to be satisfied (e.g. expected traveling date is inputted), which are defined as Web service input; and the execution of Web services also results in the availability of other benefits (e.g., users' satisfaction) or value (e.g., delivered products), which are defined as Web service output. The output of one service may be used or partially used as an input of another service.

A matching rule specifies constraints on the output of one Web service connected to the input of another Web service. A matching rule consists of two Web services to indicate that two Web service can be composed with each other by analyzing their input and output messages.

The set of matching is denoted as R_m such as $R_m = \{r_{m.1}, r_{m.2}, \ldots, \}$, where $r_{m.i}$ is a matching rule defined as

$$r_{m.i} = <s_l, s_r, CP`>,\ r_{m.i} \in R_m,\ R_m \subseteq W_t \times W_t,\ CP` \subseteq CP$$

where s_l, s_r are two constituent services in the matching rule, $s_l \in W$, $s_r \in W$, and CP` is the subset of composition pattern set CP which indicates in which pattern(s) the two constituent services can be composed together;

Matching rules are applied to determine whether two services can be composed together according to their input and output relations. In a matching rule $r_{m.i} = <s_i, s_j, CP`>$, if $CP` \neq \emptyset$, then the matching rule $r_{m.i}$ means that service s_j can be composed with service s_i according to their input and output relations, denoted by $\mathbf{M(s_i, s_j)}$.

The matching rules are not transformed from users' business-centric requirements but automatically generated based on the following principles:

Pinciple 1. $(s_i \oslash s_j) \wedge (s_i.O \supseteq s_j.I) \Rightarrow M(s_i, s_j)$;

Pinciple 2. $s_i \textcircled{\scriptsize ||} s_j \Rightarrow M(s_i, s_j)$;

Pinciple 3. $(s_i \oslash s_j) \wedge (s_i.O \supseteq s_j.I) \Rightarrow M(s_i, s_j)$;

Pinciple 4. $!s_i \Rightarrow M(s_i, s_i)$;

The set of composition rule is denoted R, and it is the union of set of structure rule, set of constraint rule, set of dependency rules and set of matching rules. And a composition rule from the set of composition rules is denoted as $\mathbf{r_i}$,

$$R = R_s \cup R_c \cup R_d \cup R_m,\ R = \{r_1, r_2, \ldots, r_n\},\ r_i \in R.$$

The Web service requirement (WSR) is modeled as a list of Web services, structure rules, constraint rules and dependency rules:

$$WSR = \{s_1, s_2, \ldots, s_m, r_1, r_2, \ldots, r_n\},\ r_i \in R_s \cup R_c \cup R_d$$

In Table 3.5, we present the context free grammar for the rule-driven Web service composition model.

Table 3.5 The Rule-driven Web Service Requirement

CompositionRules::= (CompositionRule, CompositionRules) | CompositionRules | ∅
CompositionRule ::= StructureRules | ConstraintRules | DependencyRules

StructureRules::= Service StructureOperator Service | Service StructureOperator StructureRule
StructureOperator:= ⊜ | ⫼ | ⦸ | !

DependencyRules ::= AService DependencyOperator AService
DependencyOperator::= ⊞ | ⊗ | ◉

ConstrainRules::= LocalConstraint | GlobalConstraint | ContextualConstraint
LocalConstraint::= AService.QoSAttribute ConstraintOperator Value
GlobalConstraint::= CService.QoSAttribute ConstraintOperator Value
ContextualConstraint::= Service.QoSAttribute ConstraintOperator Parameter
ConstraintOperator::= < | ⩽ | > | ⩾ | = | ⊆
Value::= Literal | Number Unit

Unit::= (Unit / Unit) | % | min | …
Service::= AService | CService
AService::= Literal
CService::= Literal
QoSAttribute::= Literal
Parameter::= Literal
Literal::= [A-Za-z0-9#] | [A-Za-z]
Number::= [0-9#]*

3.8 Conclusion

In order to flexibly build resilient application in dynamic environment, rSOC requires support in business level to model business requirements and

accordingly discover and compose Web services to satisfy the business requirements. However, current SOA lacks the capability to construct composite Web services driven by business requirements defined in natural-like language in terms of objectives. And consequently, most of current composition approaches highly reply on the fact that composition requirements are well formalized based on technical languages, but they fail to provide business users with a natural means to express their business objectives. In this chapter, we firstly survey the related work of Web service capability models and then present our requirement model for ad-hoc Web service composition.

Web service capability models are presented in two categories: capability models based on IOPE and capability models based on case-frame. Frame-based approaches to model Web service capability make a step beyond the classical IOPE paradigm, as they feature both of the business and functional characteristics in Web service description. However, describing service capabilities only based on action verb and attributes are oversimplified and may cause ambiguity without explicit descriptions of the domain by which service capabilities are manipulated. A more expressive Web capability model is required to capture business requirements, to represent different levels of abstraction, and to derive potential Web services and constraints reflecting the business aspects that the capability can satisfy.

We then model requirement for Web service composition in three different levels, i.e., business-centric requirement, capability-focused requirement and rule-driven Web service requirement allowing either technical or business users can specify their composition requirements and consequently compose and execute Web services to satisfy their requirements.

Business-centric requirements are modeled based on a structured English language, i.e., SBVR, to capture users' business objectives, expected actions and different constraints in a structured natural way. SBVR are intended to formalize the semantics of domain-specific vocabularies, business facts, and rules. User requirements represent the primary means by which they define the operative way to reach their objectives and perform their actions. Business-centric requirements incorporate functional concerns (i.e., concept nouns and verbs), non-functional concerns (i.e., rules and fact types) and contextual domain information specified with the formal use of natural language to explicitly provide a model of formal logic so requirements can be understood by computer programs.

As an intermediate level for the requirement transformation from business level, we model capability-focused requirements based on a Web service capability model to describe what a Web service can really provide.

The semantics of a capability is captured via a combination of an action-verb pair, a set of attributes, describing required resources, current states, and new states, which reflect changes in the real world effects, and, particularly, a business objective to achieve by applying the capability. An objective can be achieved by one or more capabilities each of which is semantically interlinked to other capabilities by means of capability relationships.

In the level of Web service requirement, we structured different kinds of constraints in Web service composition process in terms of composition rules, and give formal definition for each type of composition rule. All composition rules describe the constraints on/among Web services thus they can be directly taken into account when applying Web service composition approaches. Since all the constraints are formalized by in terms of rules, the set of categories of composition rules is easy to be enriched by structuring other factors (e.g., resources, mediation, etc.) that influence composition process, and by updating the composition algorithm, our approach is extensible to take into account of other constraints (composition rules) in service composition process.

The main values of our three-level requirement model for ad-hoc Web service composition are as follows: firstly it captures users' objectives in the business level and enables business objectives driven Web service composition; secondly, it extends current Web service descriptions by a capability model which describes what Web services can do without describing technical details; thirdly, it connects together composition requirements in business level and composition requirements in technical level to facilitate composition process driven by business requirements.

In the next chapter, we provide an end-to-end transformation solution to transform business-centric requirement to capability-focused requirement and finally to rule-driven Web service requirement, so that the composition requirement can be used by our Web service composition approach presented in Chapter 6 .

Chapter 4

Two-level Requirement Transformations

Introduction 104
The Requirement Transformation Framework 105
The Capability Matching Process 109
The Association Discovery Process 117
Conclusion 123

Abstract: Since different models describe endogenous and exogenous changes that may occur in the dynamic environment, ensuring the interconnectivity between them is mandatory to enable adaptability of the resilient service-oriented architecture in response to changes. The ability of models to affect and be affected by each other is also the most salient characteristic of the resilient service oriented architecture by comparison to dependable, autonomic or self-organizing architectures. The flexibility of compositing Web services without predefined plans is yet an important characteristic.

In order to connect models and guarantee the flexibility and adaptability characteristics, we develop a transformational approach between models. In our work, we focus on a business requirement model and a Web service composition model since they are the fundamental models upon which SOA-based applications and business processes are built. To this end, we have developed in the previous chapter the business-focused requirement model and the rule-driven Web service composition model. We have also introduced an intermediary model describing Web service capability in terms of SBVR business facts and rules. In this chapter, we discuss the transformational approach to derive rule-driven Web service requirement model from the business-focused requirement model. We particularly introduce the capability matching and association discovery processes to successfully transform business-centric requirements to capability-focused requirements, and finally to rule-driven Web service requirements. In fact, the capability matching process discovers all Web service capabilities that satisfy business-centric requirements, and if necessary, it dynamically gen-

erates capability compositions, whereas the association discovery process discovers all Web services that are associated with the Web service capabilities and automatically generates composition rules.

Without loss of generality, the ad-hoc Web service composition model can also be overloaded with requirements issued from other models (e.g., information security policy model, tolerance model, service level agreement model, …) since any requirement can be represented a constraint rule on Web services and among Web services. Since the ad-hoc Web service composition approach presented in next chapter applies a multi-objective and optimization techniques to find out optimal composite Web services, any additional rule can be further applied by modifying the composition logic.

4.1 Introduction

Business-centric requirements specified in structured natural language expressions cannot be directly used by most of Web service composition approaches. As result, a requirement transformation approach is required to transform structured natural language based requirements to formalized requirements that can be directly used by Web service composition processes, and accordingly composed together in order to satisfy business users' requirements.

The requirement transformation is the process of generating target requirements from source requirements, according to a transformation strategy. Current research work related to requirement transformations can be generally divided into two categories: Natural language-driven requirement transformation approaches translate free text requirements or semi-structured natural language based requirements into formalized and well-structured requirements whereas model-driven requirement transformation approaches transform requirements between different technical and well-structured models.

Concerning natural language-driven requirements transformation approaches, the direct automated transformation of natural language based requirements to formalized requirements is very difficult due to the inherent ambiguities of natural languages [GREM04][SAMK04][FKMS07] whereas manual or semi-automated transformations require often user's interventions during the transformation process or assisted patterns [MICH96][SLFE04][SASO05]. Alternative research work have focused on structured natural-based requirements such as SBVR, instead of natural language-based requirements) to provide controlled and valid transformation solutions (i.e., SBVR to UML diagrams) [AFBA11][RAPH08].

Most of these works remain efficient in developing new Web services to satisfy initialized requirements but inappropriate with multiple constraints and existing Web services which are expected to be discovered and composed in order to satisfy initialized requirements.

The emergence of the UML leads to the development of model driven architectures supported by model transformation and particularly the requirement transformation between different UML diagrams [YUBL10] [DROM03][AYWO09]. However, these approaches are designated to transform formalized requirements into other formalized requirements and they do not provide solutions for natural-like requirement transformation.

By comparing existing requirement transformation approaches, we observe that requirements that are transformed are actually defined from the same perspective, such as all requirements are defined from the perspective of Web service standards despite the usage of different languages or models (e.g., transformation from OWL-S to WSDL); if requirements are defined from different perspectives, such as business perspective and Web service technical perspective, these approaches lack the ability to identify connections between requirements in these perspectives, and consequently cannot provide obvious transformation mechanisms. For example, in our proposed composition requirement model presented in chapter 3, three level composition requirements are defined from three different perspectives (i.e., business objectives and Web service operations) ; they are rather than the same requirements (e.g., Web service operations) that are defined by different languages (e.g., WSDL, OWL-S). The divergence of perspectives and level of details raise a particular challenge for traditional requirement transformation approaches as how to transform requirements defined in different levels and from different perspectives.

To solve this challenge, we develop a requirement transformation framework to derive ad-hoc Web service requirement model from business-centric requirement models such as the business-centric requirement model. In our contribution, we discuss how to derive capabilities and Web services to satisfy business-centric requirement defined from the perspective of business objectives, and how business-centric requirements can be transformed to constrain capabilities and Web services.

4.2 The Requirement Transformation Framework

In this section, we introduce the general framework for transforming business models into Web service composition models in order to composite Web services in dynamic environments. The transformation is a top-down process consisting of two-phase processes (i.e., capability matching and as-

sociation discovery processes) to deduce Web services from business-centric requirements for a potential Web service composition. It identifies all constraints on functional and non-functional properties related to individual Web services and/or composite Web services. In addition, constraints on Web services are modeled as a set of composition rules to guide the ad-hoc Web service composition process. The transformation approach does not only discover Web services but it also deduces constraints on Web services from business-centric requirements.

The general framework of our requirement transformation is described in Figure 4.1 [LIBB13C] and includes a generic process of the following two steps:

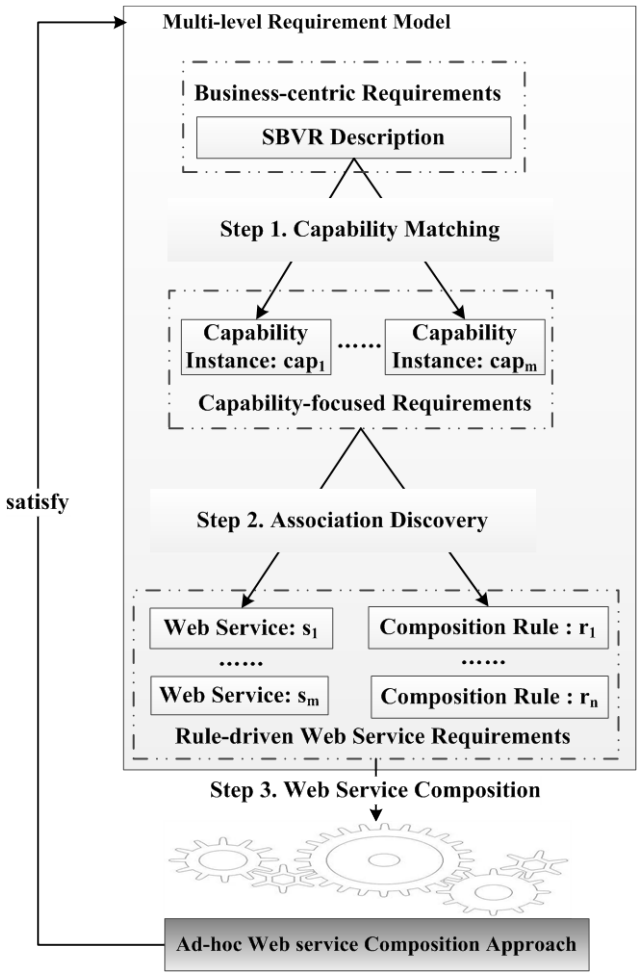

Figure 4.1 General Framework of Composition Requirements Transformation

Step 1. Capability Matching. The Capability matching process is applied to discover all capability profiles that satisfy user's business-centric requirements, in the meantime, additional capability compositions will be automatically generated in the inter-capability composition layer of

capability instances according to the information specified in business-centric requirements.

Step 2. Association Discovery. The Association discovery process discovers all Web services that are associated with capability profiles. Composition rules are automatically generated based on the capability composition as well as business-centric requirements.

Step 3. Web Service Composition. All discovered Web services and composition rules are used as the input of Web service composition approach to construct composite services satisfying users' business-centric requirements.

In the requirement transformation framework, one key operation is the *concept matching* between two concepts from requirements in different levels to decide whether they are related to each other or not. The concepts to be matched can be *action verbs* and *nouns* found in capability profiles, Web service *QoS attributes* as described by Web service profiles, etc. Our concept matching process extends the matching algorithm presented in [MEBA10] with semantic capabilities based on the domain ontology.

When matching the concept A against the concept B, we identify three cases that represent a successful matching between A and B:

1) Exact Match, which is denoted by **A=B**, means that the concepts A and B are exactly the same. For example, the SBVR fact type "extinguish" and the action verb "extinguish" in the capability profile layer, are semantically and syntactically similar (A = B).

2) Similar Match which is denoted by **A≈B** means that the two concepts A and B are synonyms,

For example, The SBVR fact type "put out" and the action verb "extinguish" in the capability profile layer are synonym (A ≈ B).

3) Subsumes Match, which is denoted by **A*sub*B**, means that the Concept B is super class of concept A in the domain ontology,.

For example, the SBVR noun concept "victim" is a sub concept of the action-verb "person" from the capability profile layer (**A*sub*B)**.

The result of matching concept is one of the following two cases:

Successful Matching: when A and B match each other according to any of the three above cases, denoted by $\mathbf{A} \overset{m}{=} \mathbf{B}$.

Failed Matching: when A and B do not satisfy any of the three above cases, denoted by $\neg(\mathbf{A} \overset{m}{=} \mathbf{B})$.

The general matching relations between different concepts from different multi-level requirement models are graphical illustrated in Figure 4.2.

.

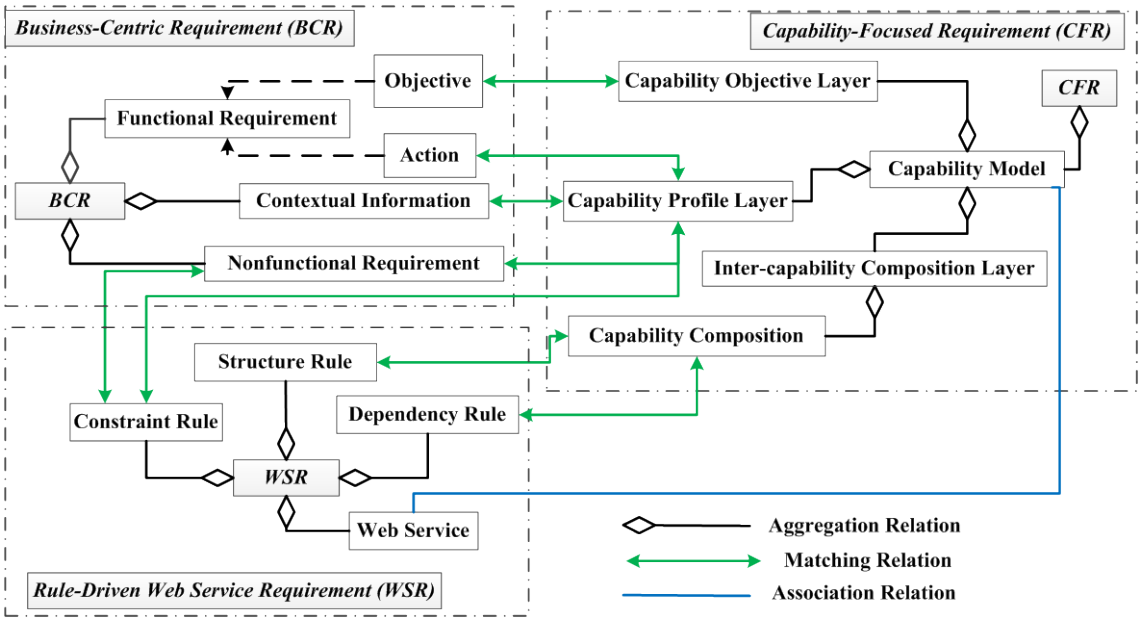

Figure 4.2 Concept Matching between Different Requirement Models

In Figure 4.2, the aggregation relation between two concepts describe one concept is constituent part of the other. The association relation between Web service and capability model describes the fact that Web services are associated with capability description to provide extra capability description. The matching relation between two concepts presents the *concept matching* between two concepts to decide whether they are related to each other or not.

In particular, the *matching relations* between different concepts ensure the connectivity among the business-centric requirement model, the capability-focused requirement model and the rule-driven Web service requirement model as described as follows:

- Functional requirements in the business-centric requirement model should be matched against the capability objective layer and the capability profile layer of capability model to discover capabilities that satisfy the objectives and action defined by functional requirements;

- Non-functional requirements should be matched against the profile layer of capability model to exclude the capability that do not satisfy non-functional requirements from the primary capability matching results, and then generate capability compositions at the inter-capability composition layer.

- Contextual information is also used to match against the capability profile layer of capability model to delete the capability instances that do not satisfy non-functional requirements from the primary capability matching results.

After discovering necessary capability instances, all Web services associated with the discovered capability instances are consequently discovered. Non-functional requirements and inter-capability compositions layer of capability model are also used to match against Web services in order to generate different categories of composition rules.

The transformation process is presented with additional details in the following sections. We introduce notations and operators in Table 4.1 to facilitate the presentation of the requirement transformation process.

Table 4.1 Notation and Operator Legends

Notation	**Meaning**
S	The set of all Web services in Web service registry;
s	A Web service;
CAP	The set of all capability instances in Web service registry;
cap	A capability instance;
S`	The set of derived Web services;
CAP`	The set of derived capability instances;
R`	The set of derived composition rules;
r_s	A structure rule;
r_c	A constraint rule;
r_d	A dependency rule;
Operator	**Meaning**
¬	Negation
^	Conjunction
∨	Disjunction
=>	If…, then….
A≝B	The matching between concept A and B is successful.(*)
⊨(condition)	The condition is satisfied.
⊭(condition)	The condition is not satisfied.
s_i►cap_j	Web service s_i is associated with capability cap_j.
t_j~a_i	The atomic service t_j is one of the discovered services to achieve the action represented by the abstract service a_i.

(*)To be more precise, we use NounConcept or FactType to represent all noun concepts or fact types in one sentence described based on the SBVR terms. For a concept A, the matching A≝NounConcept /FactType is successful when the concept A can be successfully matched against any of the noun concept/ fact type.

4.3 The Capability Matching Process

Business-centric requirements are defined from the "request" perspective based on business objectives while Web service capability is modeled from "offering" perspective, indicating what a Web service can really

provide. When transforming business-centric requirements to capability-focused requirements, the capability matching algorithm discovers available capability instances that satisfy user's objectives and actions as defined in business-centric requirements; when certain nonfunctional requirement defined in business-centric requirements cannot be satisfied, the capability matching process will try to compose capabilities to satisfy requirements.

In order to transform business-centric requirements to capability-focused requirements, the capability matching process matches different concepts from the business-centric requirement model and the capability model following six predefined principles.

Capabitliy Matching Principle 1.

Functional reuqirement of business-centric requirements (BCR) is used to match against capability objective layer (OL) and capability name in capability profile layer (PL) of capability instances in the Web service registry and all capability instances satisfying one objective and one action defined in functional requirements (FR) are discovered as the primary capability matching result.

Matching relations between functional requirements and the capability model is presented in Table 4.2. In the detail matching relation table, the two columns in the left and right respectively list two concepts to be matched; the *field* column in the table introduces the field where a concept is defined while the *concept* column introduces the name of the concept.

Table 4.2 Matching Relation of Capability Matching Principle 1

Business-Centric Requirement Model			Capability Model	
Field	Concept	Matching against	Concept	Field
BCR.FR.Objective	fact type		action verb	CAP.OL
BCR.FR.Objective	noun concept		noun	CAP.OL
BCR.FR.Action	fact type		action verb	CAP.PL.cname
BCR.FR.Action	noun concept		noun	CAP.PL.cname

For example, when giving a functional requirement specified as in Table 4.3, a capability instance $cap_{1.1}$ satisfying both one objective and one action of FR is discovered as shown in Table 4.3.

As illustrated in Table 4.3, the objective layer of the capability instance $cap_{1.1}$ can successfully match with the objective "obj_1: Manage train crisis" defined as functional requirement, while the capability name can successfully match the action "a_1: Fire must be extinguished." defined in functional requirement.

Capabitliy Matching Principle 2.

Non-functional requirement of business-centric requirements (BCR) are matched against the capability profile layer (PL) of capabilities and the capabilities dissatisfying rules specified in the nonfunctional requirements (NR) will be deleted from the primary capability matching result.

Table 4.3 Capability Matching Principle 1 Example

BCR.FR.Objective	obj_1: Manage train crisis	
BCR.FR.Action	a_1: Fire must be extinguished.	
Successful matching		
ID	Capability Objective Layer	Capability Profile Layer
		capability name
$cap_{1.1}$	{<manage, crisis>}	<extinguish, fire>

The detail matching relations from capability matching principle 2 is presented in Table 4.4.

Table 4.4 Matching Relation of Capability Matching Principle 2

Business-Centric Requirement Model			Capability Model	
Field	Concept	Matching against	Concept	Field
BCR.NR	noun concept		attribute name	CAP.PL
BCR.NR	quantifier		attribute value	CAP.PL

For example, when giving a non-functional requirement specified as shown in Table 4.5, a capability instance dissatisfying this requirement is found as shown in Table 4.5 and will be delted from the primary result.

Table 4.5 Capability Matching Principle 2 Example

BCR.NR	nr_1: It is obligatory that at least 40 firemen extinguish fire.		
Successful matching			
ID	Capability Objective Layer	Capability Profile Layer	
		capability name	attributes
$cap_{1.1}$	{<manage, crisis>}	<extinguish, fire>	{(AvailableFiremen, 30)}

As illustrated in Table 4.5, the attribute "AvailabileFiremen" in the capability instance $cap_{1.1}$ has successfully matched with the non-functional requirement "nr_1: It is obligatory that at least 40 firemen extinguish fire". However, the value of the attribute "AvailabileFiremen" is "30" and it does not satisfy the quantifier "at least 40", thus the capability instance $cap_{1.1}$ will be deleted from the primary capability matching result.

Capabitliy Matching Principle 3.

Contextual information (CTT) of business-centric requirements (BCR) are matched against capability profile layer (PL) of capabilities and the capabilities dissatisfying rules specified in nonfunctional requirements (NR) will be deleted from the primary result.

The detail matching relations from capability matching principle 3 is presented in Table 4.6.

Table 4.6 Matching Relation of Capability Matching Principle 3

Business-Centric Requirement Model			Capability Model	
Field	Concept	Matching against	Concept	Field
BCR.CTT	noun concept		attribute name	CAP.PL
BCR.CTT	instance		attribute value	CAP.PL

For example, when giving a context information "ctt_1: Crisis place is Pairs." as shown in Table 4.7, a capability instance $cap_{2.1}$ the attribute dissatisfying this requirement is found as shown in Table 4.7 and will be delted from the primary result.

Table 4.7 Capability Matching Principle 3 Example

BCR.CTT	ctt_1: Crisis place is Pairs.		
Successful matching			
ID	Capability Objective Layer	Capability Profile Layer	
		capability name	attributes
$cap_{2.1}$	{<manage, crisis>}	<evacuate, population>	{(AvailablePlace, Marseille)}

As illustrated in Table 4.7, the attribute "AvailablePlace" in the capability instance $cap_{2.1}$ has successfully matched with the contextual information "ctt_1: Crisis place is Pairs." However, the value of the attribute "AvailablePlace" is "Marseille" and it does not satisfy the instance "Pairs", thus the capability instance $cap_{2.1}$ will be deleted from the primary capability matching result.

Capabitliy Matching Principle 4.

For one non-functional requirement (NR) from business-centric requirements (BCR), if this non-functional requirement can succesful match againt other capability instances but no capability instance can satisfy this non-fucntional requirement, dynamic capability composition based on cooperation relation will be generated to satisfy this requirement by composing more capabilities; if composing more capabilities cannot satisfy this requirement, then eventually this requirement cannot be satisfied.

The detail matching relations from capability matching principle 4 is presented in Table 4.8.

Table 4.8 Matching Relation of Capability Matching Principle 4

Business-Centric Requirement Model			Capability Model	
Field	Concept	Matching against	Concept	Field
BCR.NR	noun concept		attribute name	CAP.PL
BCR.NR	quantifier		attribute value	CAP.PL

For example, when giving a non-functional requirement "nr_1: It is obligatory that at least 40 firemen extinguish fire." specified as shown Table 4.9, two capability instances $cap_{1.1}$ and $cap_{1.2}$ dissatisfying this requirement are found as shown in Table 4.9, and a capability composition is automatically generated to satisfy this non-functional requirement.

Table 4.9 Capability Matching Principle 4 Example

BCR.NR	nr_1: It is obligatory that at least 40 firemen extinguish fire.			
Successful matching				
ID	Capability Objective Layer	Capability Profile Layer		Inter-capability Composition Layer
		capability name	attributes	
$cap_{1.1}$	{<manage, crisis>}	<extinguish, fire>	{(Available-Firemen, 30)}	{cap1.1, cap1.2, Coopera-tion, dynamic}
$cap_{1.2}$	{<manage, crisis>}	<extinguish, fire>	{(Available-Firemen, 20)}	{cap1.1, cap1.2, Coopera-tion, dynamic}

As illustrated in Table 4.9, the attribute "AvailabileFiremen" in the capability instance $cap_{1.1}$ and $cap_{1.2}$ has successfully matched with the non-functional requirement "nr_1: It is obligatory that at least 40 firemen extinguish fire". However, the values of the attribute "AvailabileFiremen" of $cap_{1.1}$ and $cap_{1.2}$ are respectively "30" and "20" and neither of $cap_{1.1}$ or $cap_{1.2}$ satisfies the quantifier "at least 40". Thus a capability composition "{$cap_{1.1}$, $cap_{1.2}$, Cooperation, dynamic}" is automatially generated in the inter-capability composition layer of each capability instance, in which the $cap_{1.1}$ and $cap_{1.2}$ are composed by cooperation relation providing available 50 firemen to satisfy the non-functional requirement nr_1.

The detail matching relations from capability matching principle 5 is presented in Table 4.10.

For example, when giving a non-functional requirement "nr_1: It is obligatory that the electricity is recovered after the fire is extinguished." specified as shown in Table 4.11, two capability instances $cap_{1.1}$ and $cap_{3.1}$ successful matching this requirement are found, and a capability composition is automatically generated following this non-functional requirement.

Capabitliy Matching Principle 5.

If one non-functional requirement (NR) from business-centric requirements (BCR) defines control flow constraints by use of SBVR fact types: "*before/after*", "*while*", and "*if ..., then ...*", a dynamic capability composition will be generated:

1) A support capability composition will be generated if SBVR fact type "*before/after*" is used;

2) A cooperation capability composition will be generated if SBVR fact type "*while*" is used;

3) A condition capability composition will be generated if SBVR fact type "*if ..., then ...*" is used.

Table 4.10 Matching Relation of Capability Matching Principle 5

Business-Centric Requirement Model			Capability Model	
Field	Concept	Matching against	Concept	Field
BCR.NR	noun concept		noun	CAP.PL.cname
BCR.NR	fact type		action verb	CAP.PL.cname

Table 4.11 Capability Matching Principle 5 Example

BCR.NR	nr_1: It is obligatory that the electricity is recovered after the fire is extinguished.			
Successful matching				
ID	Capability Objective Layer	Capability Profile Layer		Inter-capability Composition Layer
		capability name	attributes	
$cap_{1.1}$	{<manage, crisis>}	<extinguish, fire>	{(Available-Firemen, 30)}	{$cap_{1.1}$, $cap_{3.1}$, Support, dynamic}
$cap_{3.1}$	{<manage, accident>}	<recover, electricity>		{$cap_{1.1}$, $cap_{3.1}$, Support, dynamic}

As illustrated in Table 4.11, the capability name "<extinguish, fire>" in the capability instance $cap_{1.1}$ has successfully matched with the non-functional requirement "nr_1: It is obligatory that the electricity is recovered after the fire is extinguished.", while the capability name <recover, electricity>" in the capability instance $cap_{1.1}$ has also successfully matched with the same non-functional requirement. A capability composition "{$cap_{1.1}$, $cap_{3.1}$, Support, dynamic}" is automatially generated in the inter-capability composition layer of each capability instance, in which the $cap_{1.1}$ and $cap_{3.2}$ are composed by support relation to satisfy the non-functional requirement nr_1.

Capabitliy Matching Principle 6.

If one non-functional requirement (NR) from business-centric requirements (BCR) defines dependency constraints by use of SBVR modal operators: "*maybe*", and "*must not*", a dynamic capability composition will be generated:

1) A promotion capability composition will be generated if SBVR operator "*maybe*" is used;

2) A competition capability composition will be generated if SBVR operator type "*must not*" is used.

The detail matching relations from capability matching principle 6 is presented in Table 4.12.

Table 4.12 Matching Relation of Capability Matching Principle 6

Business-Centric Requirement Model			Capability Model	
Field	Concept	Matching against	Concept	Field
BCR.NR	noun concept		noun	CAP.PL.cname
BCR.NR	fact type		action verb	CAP.PL.cname

Following the six principles, capability matching process firstly discover all capability instances satisfying the objectives and actions defined in functional requirements in business-centric requirement model; and then delete the capability instances that do not satisfy certain non-functional requirements or context information; at last dynamic capability compositions are automatically generated according to different constraints specified by non-functional requirements.

The detail capability matching process is formalized in Figure 4.3 and Figure 4.4.

Capability Matching Process
Input: BCR, CAP;
Step 1. CAP` = ∅;
Step 2. (Principle 1)
$\forall cap_i \in CAP$,
$\exists obj_j \in cap_i.OL$, $\exists obj_p \in FR.Objective$, $\exists a_q \in FR.Action$,
$(cap_i.obj_j.noun \overset{m}{=} FR.obj_p.NounConcept) \wedge (cap_i.obj_j.action\_verb \overset{m}{=} FR.obj_p.FactType) \wedge$
$(cap_i.cname.noun \overset{m}{=} FR.i_q.NounConcept) \wedge (cap_i.cname.action\_verb \overset{m}{=} FR.a_q.FactType)$
$\Rightarrow CAP` = CAP` \cup \{cap_i\}$;

Figure 4.3 The Capability Matching Process (Part 1)

Step 3. (Principle 4)

$\forall cap_i \in CAP`$, $\exists attr_j \in cap_i.PL$, $\exists nr_o \in NR$, $\exists quantifier \in nr_o$,
$(cap_i.attr_j.attr\_name \overset{m}{\equiv} nr_o.NounConcept) \wedge$
$(cap_i.attr_j.attr\_value \Vdash \neq (nr_o.quantifier))$
$\Rightarrow \exists cap_m \in CAP, \exists attr_p \in cap_m.PL, \exists cap_n \in CAP, \exists attr_q \in cap_n.PL,$
$(cap_m.attr_p.attr\_name \overset{m}{\equiv} nr_o.NounConcept) \wedge$
$(cap_n.attr_q.attr\_name \overset{m}{\equiv} nr_o.NounConcept)$
$\Rightarrow$ generate $cap_m.CL.cc_x = cap_m.CL.cc_y = <cap_m, cap_n, Cooperation, dynamic>$

Step 4. (Principle 2)

$\forall cap_i \in CAP`$,
$\exists attr_j \in cap_i.PL$, $\exists nr_o \in NR$, $\exists quantifier \in nr_o$,
$(cap_i.attr_j.attr\_name \overset{m}{\equiv} nr_o.NounConcept) \wedge (cap_i.attr_j.attr\_value \Vdash \neq (nr_o.quantifier))$
$\Rightarrow CAP` = CAP` - \{cap_i\}$;

Step 5. (Principle 3)

$\forall cap_i \in CAP`$,
$\exists attr_j \in cap_i.PL$, $\exists ctt_o \in CONTEXT$,
$(cap_i.attr_j.attr\_name \overset{m}{\equiv} nr_o.NounConcept) \wedge (cap_i.attr_j.attr\_value \Vdash \neq (NounConcept))$
$\Rightarrow CAP` = CAP` - \{cap_i\}$;

Step 6. (Principle 5)

$\forall cap_m \in CAP`$, $\forall cap_n \in CAP`$,
$\exists nr_o \in NR$,
$(cap_m.cname.noun \overset{m}{\equiv} NR.nr_o.NounConcept) \wedge$
$(cap_m.cname.action\_verb \overset{m}{\equiv} NR.nr_o.FactType) \wedge$
$(cap_n.cname.noun \overset{m}{\equiv} NR.nr_o.NounConcept) \wedge$
$(cap_n.cname.action\_verb \overset{m}{\equiv} NR.nr_o.FactType) \wedge$
$((\text{“after”} \in NR.nr_o.FactType) \vee (\text{“before”} \in NR.nr_o.FactType)$
$\vee (\text{“while”} \in NR.nr_o.FactType) \vee (\text{“if…then…”} \in NR.nr_o.FactType))$
$\Rightarrow$ generate $cap_m.CL.cc_x = cap_m.CL.cc_y = <cap_m, cap_n, rel, dynamic>$:
$(\text{“after”} \in NR.nr_o.FactType) \vee (\text{“before”} \in NR.nr_o.FactType) \Rightarrow rel = Support$;
$(\text{“while”} \in NR.nr_o.FactType) \Rightarrow rel = Cooperation$;
$(\text{“if…then…”} \in NR.nr_o.FactType) \Rightarrow rel = Condition$;

Step 7. (Principle 6)

$\forall cap_m \in CAP`$, $\forall cap_n \in CAP`$,
$\exists nr_o \in NR$,
$(cap_m.cname.noun \overset{m}{\equiv} NR.nr_o.NounConcept) \wedge$
$(cap_m.cname.action\_verb \overset{m}{\equiv} NR.nr_o.FactType) \wedge$
$(cap_n.cname.noun \overset{m}{\equiv} NR.nr_o.NounConcept) \wedge$
$(cap_n.cname.action\_verb \overset{m}{\equiv} NR.nr_o.FactType) \wedge$
$((\text{“maybe”} \in NR.nr_o.\ ModalOperators) \vee (\text{“must not”} \in NR.nr_o.ModalOperators))$
$\Rightarrow$ generate $cap_m.CL.cc_x = cap_m.CL.cc_y = <cap_m, cap_n, rel, dynamic>$:

Figure 4.4 The Capability Matching Process (Part 2)

4.4 The Association Discovery Process

When given the derived capability instances from capability matching process, association discovery process transforms the capability-focused requirement into rule-driven Web service requirement. Association discovery process discovers all Web services that are associated with capability instances from capability matching process, and generates different categories of composition rules based on capability compositions and non-functional requirement.

The association discovery process matches different concepts from business-centric requirement model, capability model and rule-driven Web service requirement model, and transform requirements following five association discovery principles.

Association Discovery Principle 1

For each discovered capability instance from capability matching process, all Web services that are associated with this capability are discovered as association discovery result; in the meantime, for all the atomic services that are associated with the same capability instance, an abstract service is generated to represent their similar operations.

For example, for a capability instance $cap_{1.1}$, three Web services t_1, t_2, t_3 are associated with this capability instance: $t_1 \blacktriangleright cap_{1.1}$, $t_2 \blacktriangleright cap_{1.1}$, and $t_3 \blacktriangleright cap_{1.1}$. The capability instance and Web service profiles are presented in Table 4.13, and t_1, t_2, and t_3 are all discovered as association discovery result, and an abstract service a_1 is generated. We get $t_1 \sim a_1$, $t_2 \sim a_1$, and $t_3 \sim a_1$.

Association Discovery Principle 2.

If a capability composition based on Support, Cooperation or Condition relation is defined in the inter-capability composition layer of a capability instance, a structure rule will be generated between all Web services that are associated with the two capabilities within the capability composition:

1) A structure rule with sequence composition pattern is generated if the capability composition is defined by use of Support relation;

2) A structure rule with parallel composition pattern is generated if the capability composition is defined by use of Cooperation relation;

3) A structure rule with selection composition pattern is generated if the capability composition is defined by use of Condition relation.

For example, two capability instances $cap_{1.1}$ and $cap_{1.2}$ have the same the inter-capability composition layer. Two Web services t_1 and t_2 are associated with $cap_{1.1}$ while one Web service t_3 is associated with $cap_{1.2}$. The capability instances and Web service profiles are presented in Table 4.14. Following association discovery principle 1, a structure rule with par-

allel composition pattern will be generated between all Web services associated with $cap_{1.1}$ and all Web services associated with $cap_{1.2}$, and the generated structure rules are:

$$r_{s.1} = <a_1, a_2, ⓘ>, t_1 \sim a_1, t_2 \sim a_1, t_3 \sim a_2$$

Table 4.13 Association Discovery Principle 1 Example

CAP ID	Capability Objective Layer	Capability Profile Layer		Inter-capability Composition Layer
		capability name	attributes	
$cap_{1.1}$	{<manage, crisis>}	<extinguish, fire>	{(AvailableFiremen, 30)}	
Associating with				
WS ID	Input	Operation Name	Output	QoS
t_1 ($t_1 \blacktriangleright cap_{1.1}$)	String: FirePlace	CallCommandCenter	Boolean: Confirmation	response_time = 20m
t_2 ($t_2 \blacktriangleright cap_{1.1}$)	String: FireLocation	SendFireBrigade	Integer: FiremenInTask	response_time = 10m
t_3 ($t_3 \blacktriangleright cap_{1.1}$)	String: FirePlace Integer: RequiredFiremen	RequireBrigade	Boolean: Confirmation	response_time = 15m

Association Discovery Principle 3.

If one nonfunctional requirement (NR) in of business-centric requirements (BCR) describes a property constraint on one capability, then a local constraint rule will be generated on all the Web services that are associated with these capabilities: the quantifier in NR is be used to generate constraint rule's expression while the QoS attribute which successfully matches against the noun concept in NR is the attribute to be constrained.

The detail matching relations between non-functional requirement, capability model and Web service model is presented in Table 4.15.

For example, Table 4.16 presents a non-functional requirement nr_1, a capability instance $cap_{1.1}$, and three Web services t_1, t_2 and t_3 that are associated with $cap_{1.1}$.

As illustrate in Table 4.16, the capability name of $cap_{1.1}$ "<extinguish, fire>" sucessfully matches the non-functional requirement "nr1: It is necessary that the response time of extinguish fire is less than 30m." while the noun concept "response time" in non-functional requirement nr_1 successful matches against the QoS attribute "response time" of Web services that are associated with the capability instance $cap_{1.1}$, i.e., t_1, t_2, and t_3. A local constraint rule is generated following association discovery principle 3 to constrain all Web services that are associated with capability

instance $cap_{1.1}$, which relate to the abstract service a_1. The expression in the constraint rule is genereated accroding to the quantifier of nr_1 "less than 30m", while the QoS attribute to be constrained is "response time", thus the generated constraint rule is:

$$r_{lc.1} = <a_1.\text{response time}, <, 30m>, t_1 \sim a_1, t_2 \sim a_1, t_3 \sim a_1$$

Table 4.14 Association Discovery Principle 2 Example

CAP ID	Capability Objective Layer	Capability Profile Layer		Inter-capability Composition Layer
		capability name	attributes	
$cap_{1.1}$	{<manage, crisis>}	<extinguish, fire>	{(AvailableFiremen, 30)}	{$cap_{1.1}$, $cap_{1.2}$, Co-operation, dynamic}
$cap_{1.2}$	{<manage, crisis>}	<extinguish, fire>	{(AvailableFiremen, 20)}	{$cap_{1.1}$, $cap_{1.2}$, Co-operation, dynamic}
Associating with				
WS ID	Input	Operation Name	Output	QoS
t_1 ($t_1 \blacktriangleright cap_{1.1}$)	String: FirePlace	CallCommandCenter	Boolean: Confirmation	response_time = 20m
t_2 ($t_2 \blacktriangleright cap_{1.1}$)	String: FireLocation	SendFireBrigade	Integer: FiremenInTask	response_time = 10m
t_3 ($t_3 \blacktriangleright cap_{1.2}$)	String: FirePlace Integer: RequiredFiremen	RequireBrigade	Boolean: Confirmation	response_time = 15m

Table 4.15 Matching Relation of Association Discovery Principle 3

Business-Centric Requirement Model			Capability Model	
Field	Concept	Matching against	Concept	Field
BCR.NR	fact type		action verb	CAP.PL.cname
BCR.NR	noun concept		noun	CAP.PL.cname
BCR.NR	noun concept		QoS attribute	SP.QoS

Table 4.16 Association Discovery Principle 3 Example

BCR.NR	nr1: It is necessary that the response time of extinguish fire is less than 30m.			
Successful matching				
CAP ID	Capability Objective Layer	Capability Profile Layer		Inter-capability Composition Layer
		capability name	attributes	
$cap_{1.1}$	{<manage, crisis>}	<extinguish, fire>	{(AvailableFiremen, 30)}	{$cap_{1.1}$, $cap_{1.2}$, Co-operation, dynamic}
Associating with				
WS ID	Input	Operation Name	Output	QoS
t_1 (t_1►$cap_{1.1}$)	String: FirePlace	CallCommandCenter	Boolean: Confirmation	response_time = 20m
t_2 (t_2►$cap_{1.1}$)	String: FireLoca-tion	SendFireBrigade	Integer: Fire-menInTask	response_time = 10m
t_3 (t_3►$cap_{1.1}$)	String: FirePlace Integer: Required-Firemen	RequireBrigade	Boolean: Confirmation	response_time = 15m

Association Discovery Principle 4.

If one nonfunctional requirement (NR) in of business-centric requirements (BCR) describes a property constraint on the expected business process, then a global constraint rule will be generated to constrain the composite services to be constructed: the quantifier in NR is be used to generate constraint rule's expression while the QoS attribute which successfully matches against the noun concept in NR is the attribute to be constrained.

The detail matching relations between non-functional requirement, capability model and Web service model is presented in Table 4.17.

Table 4.17 Matching Relation of Association Discovery Principle 4

Business-Centric Requirement Model			Capability Model	
Field	Concept	Matching against	Concept	Field
BCR.NR	noun concept		QoS attribute	SP.QoS

For example, Table 4.18 presents a non-functional requirement nr_1, and two Web services t_1, and t_2.

As illustrated in Table 4.18, the noun concept "total price" in non-functional requirement nr_1 "nr_1: It is necessary that the total price is less than 2000€." successful matches against the QoS attribute "price", a global constraint rule is generated following association discovery principle 4 to constrain all composite services. The expression in the constraint rule is genereated accroding to the quantifier of nr_1 "less than 2000€", while the

QoS attribute to be constrained is "price", thus the generated constraint rule is:

$$r_{gc.1} = <c.price, <, 2000€>$$

Table 4.18 Association Discovery Principle 4 Example

BCR.NR	nr_1: It is necessary that the total price is less than 2000€.			
Successful matching				
WS ID	Input	Operation Name	Output	QoS
t_1	String: FirePlace	CallCommandCenter	Boolean: Confirmation	price = 500€
t_2	String: VictimLocation String: Destination	SendAmbulance	Float: EstimatedTime	price = 100€

Association Discovery Principle 5.

If a capability composition based on Promotion or Competition relation is defined in the inter-capability composition layer of a capability instance, a dependency rule will be generated between all Web services that are associated with the two capabilities within the capability composition:

1) A dependency rule with *optimal composed* dependency relation is generated if the capability composition is defined by use of Promotion relation;

2) A dependency rule with *excluded* dependency relation is generated if the capability composition is defined by use of Competition relation.

For example, two capability instances $cap_{1.1}$ and $cap_{1.2}$ have the same inter-capability composition layer. The Web services t_1 is associated with $cap_{1.1}$ while the Web service t_2 is associated with $cap_{1.2}$. The capability instances i.e., $cap_{1.1}$ and $cap_{1.2}$, and Web service profiles i.e., t_1 and t_2 are presented in Table 4.19. Following association discovery principle 5, a dependency rule with excluded dependency relation will be generated between all Web services associated with $cap_{1.1}$ and all Web services associated with $cap_{1.2}$, and thus the generated dependency rule is:

$$r_{d.1} = <t_1, t_2, \otimes>$$

Following the five association discovery principles introduced above, association discovery process firstly discover all Web services that are associated with capability instances from capability matching process; and then if a capability composition is defined in the inter-capability composition layer of capability instances, a structure/dependency rule will be generated between the Web services that are associated with the two capabilities within the capability composition, while if NR describes a constraint on certain/all capabilities, then a constraint rule will be generated to

constrain all composite services. The detail association discovery process is formalized in Figure 4.5 and Figure 4.6.

Table 4.19 Association Discovery Principle 5 Example

CAP ID	Capability Objective Layer	Capability Profile Layer		Inter-capability Composition Layer
		capability name	attributes	
$cap_{1.1}$	{<manage, crisis>}	<extinguish, fire>	{(AvailableFiremen, 30)}	{$cap_{1.1}$, $cap_{1.2}$, Competition, static}
$cap_{1.2}$	{<manage, crisis>}	<extinguish, fire>	{(AvailableFiremen, 20)}	{$cap_{1.1}$, $cap_{1.2}$, Competition, static}
Associating with				
WS ID	Input	Operation Name	Output	QoS
t_1 (t_1►$cap_{1.1}$)	String: FirePlace	CallCommandCenter	Boolean: Confirmation	response_time = 20m
t_2 (t_2►$cap_{1.2}$)	String: FireLocation	SendFireBrigade	Integer: FiremenInTask	response_time = 10m

Association Discovery Process

Input: BCR, CFR, S;

Step 1. S` = ∅; R` = ∅;

Step 2. (Principle 1)

$\forall cap_i \in CAP`, \exists s_p \in S,$

$(s_p \blacktriangleright cap_i) \Rightarrow S` = S` \cup \{s_p\};$

$\forall s_i \in S`,$

generate $s_i \sim a_j \Rightarrow S` = S` \cup \{a_j\};$

Step 3. (Principle 2)

$\forall cap_i \in CAP`, \forall cap_j \in CAP`,$

$\exists cc_e \in (cap_i.CL \cap cap_j.CL), \exists s_p \in S`, \exists s_q \in S`,$

$(s_p \blacktriangleright cap_i) \wedge (s_q \blacktriangleright cap_j) \wedge (cc_e.rel = Support/Cooperation/Condition)$

$\Rightarrow$ generate $r_{s.m} = <s_p, s_q, cp>$:

a) $cc_e.rel = Support \Rightarrow cp = \circledS;$

b) $cc_e.rel = Cooperation \Rightarrow cp = \circledII;$

c) $cc_e.rel = Condition \Rightarrow cp = \oslash;$

$R` = R` \cup \{r_{s.m}\};$

Figure 4.5 The Association Discovery Process (Part 1)

Step 4. (Principle 3)

$\forall cap_i \in CAP`$, $\forall cap_j \in CAP`$,

$\exists cc_e \in (cap_i.CL \cap cap_j.CL)$, $\exists s_p \in S`$, $\exists s_q \in S`$,

$(s_p \blacktriangleright cap_i) \wedge (s_q \blacktriangleright cap_j) \wedge (cc_e.rel = Promotion/Competition)$

$\Rightarrow$ generate $r_{d.n} = <s_p, s_q, dp>$:

a) $cc_e.rel = Competition \Rightarrow dp = \otimes$;

b) $cc_e.rel = Promotion \Rightarrow dp = \boxplus$;

$R` = R` \cup \{r_{d.n}\}$;

Step 5. (Principle 4)

$\forall cap_i \in CAP`$, $\exists s_p \in S`$, $\exists q_x \in s_j.QoS$, $\exists nr_o \in NR$, $\exists quantifier \in nr_o$,

$(q_x \overset{m}{=} nr_o.NounConcept) \wedge (cap_i.cname.noun \overset{m}{=} nr_o.NounConcept) \wedge$

$(cap_i.cname.action\_verb \overset{m}{=} nr_o.FactType) \wedge (s_p \blacktriangleright cap_i)$

$\Rightarrow$ generate $r_{lc.m} = <s_p.q_x, nr_o.quantier>$;

$R` = R` \cup \{r_{lc.m}\}$;

Step 6. (Principle 5)

$\forall cap_i \in CAP`$, $\exists s_p \in S`$, $\exists q_x \in s_j.QoS$, $\exists nr_o \in NR$, $\exists quantifier \in nr_o$,

$(q_x \overset{m}{=} nr_o.NounConcept)$

$\Rightarrow$ generate $r_{lc.m} = <c.q_x, nr_o.quantier>$;

$R` = R` \cup \{r_{lc.m}\}$;

Output: $WSR = S` \cup R`$;

Figure 4.6 The Association Discovery Process (Part 2)

4.5 Conclusion

Connecting different models is a crucial step in developing resilient SOA to affect each model and dynamically update composite Web services (e.g., business processes) to changes. To this end, the transformation between requirement models allows flexibility and adaptability of the resilient SOA applications and business processes. Most requirement model transformation approaches have been developed to transform requirements defined from the same perspective with different languages. However, they do not provide efficient transformations when requirement models are defined from different perspectives (e.g., businesses, Web service capability, Web service operation, etc.).

In this chapter, we focused on our multi-level requirement models and particularly discuss the transformation of the business-centered requirement model into the Ad-hoc Web service composition model through the Web service capability model. We introduce the transformation framework to transform requirements defined from the business perspectives, Web service capability and Web service operations. The requirement trans-

formation framework consists of two processes, i.e., capability matching and association discovery processes, which are mainly driven by a list of predefined matching and discovery principles to match different concepts (e.g., action verb, fact type, etc.) from three-level composition requirement models so as to make links between business objectives and Web services.

In the two-phase transformation process, the capability matching process discovers capability instances that satisfy business objectives by matching concepts from business-centric requirement defined by SBVR and Web service capability instances, and also transforms different kinds of constraints into the form of capability compositions, which are dynamically generated following predefined principles.

Once capability instances are discovered and capability compositions are generated, association discovery discovers all Web services associated with the resulting capability instances and generates composition rules based on business-centric requirement and capability composition following predefined principles. The discovered Web services and generated composition rules form the rule-driven Web service requirements that are used as the input of Web service composition approaches, and guide the composition processes.

Based on our requirement transformation process, business-centric requirements describing business objectives with SBVR terms is transformed to capability-focused requirements describing Web services' real abilities based on Web service capability model, and further transformed to rule-driven Web service requirements that can be directly used by Web service composition approaches. The main contribution of this chapter is that we achieve a transformation between requirements that are defined from different perspectives in different languages, as business-centric requirements are defined from the perspective of request based on SBVR while the rule-driven Web service requirements are defined from the perspective of offer based on composition rules.

In the next chapter, we present the ad-hoc Web service composition approach in dynamic environments, called Service Farming, which constructs optimal composite services while taking into account all composition rules from rule-driven Web service requirements.

Chapter 5

Service Farming: An Ad-hoc Web Service Composition Approach

Introduction 126
The Service Model 129
Service Farming Composition Algorithm 132
The Dynamic Reconfiguration of Composite Web Services 145
Conclusion 151

Abstract: As an architectural style, the service-oriented architecture supports information systems by quickly developing new applications or new business processes, integrating existing self-contained and loosely-coupled components (i.e., services). The prominent characteristics of the SOA, such as agility, flexibility and reusability, are mainly ensured through the composition process that integrates existing services into new composite services or business processes. The composition process, as the core of the SOA foundation, reveals challenges as to how automate service compositions and update it to changes accordingly. As reported in the state of the art chapter, most of composition approaches that attempt to automate the dynamic composition processes require predefined abstract composition plans. This may cause that the aggregated QoS values of composites service are limited by the composition plan, since we will get different QoS aggregation results by applying the same set of atomic services into different composition plans. Our resilient SOA seeks to extend SOA capabilities, and particularly the composition process, to build adaptable applications or adaptable business processes without composition plans dealing with various changes in dynamic environments. The resilient SOA assumes that the composition process and internal/external changes in the dynamic environment can be specified with models that affect and are affected by each other. To this end, we build a rule-driven composition model based on which we develop an ad-hoc composition approach. In our review, these contribu-

tions establish the foundation towards a resilient SOA to build dynamic applications adaptable in dynamic environments.

In this chapter, we introduce the ad-hoc Web service composition approach, called Service Farming, which aims at constructing an optimal composite service in a reasonable time, and without predefined composition plans. Based on a set of rules, the composite services attempt to particularly satisfy business requirements while maximizing user's satisfaction (e.g., business requirement model) and to generally satisfy multiple constraints (e.g., models describing variable that may change in the environment). The composition is ad-hoc in the sense that it constructs composite services by simultaneously selecting atomic services and inferring the composition patterns between the selected services based on the set of rules in order to provide the best ways to satisfy constraints (e.g., QoS changes) in both static and dynamic composition environments.

5.1 Introduction

The Web service composition is achieved by reusing existing Web services and logically recombining them into composite services in order to create innovative Web services, often for a value that is higher than the sum of the value of the separate Web services.

In the process of Web service composition, the functional requirements should not only be satisfied, but the non-functional requirements imposed by users that must be taken into account as well. Functional requirements describe what Web services do while non-functional properties include other requirements, which are not directly related to the functionality provided by Web services. Based on non-functional requirements, users can differentiate Web services with similar functionalities. Functional requirements can also be regarded as which Web services should be executed; while non-functional requirement describe how they are executed: Non-functional requirements play an important role in discovering, selecting and composing Web services. However, constructing composite services while meeting both functional and non-functional requirements imposed by users is proved to be a NP-hard problem [MBKG09].

In dynamic Web service composition environments, constraints on/among different Web services, such as control flow constraints, property constraints, dependency constraints, and available Web services just to mention a few, change overtime. When contextual information changes occur at the design time or runtime, the Web service composition approach should be able to transparently adapt composite Web services to the environment changes with minimum user interventions. This context also raises

a challenge as how to compose Web services while satisfying users' requirements as well as adapting to changes in the dynamic environment.

Many Web service composition approaches proceed with two phase's process to construct composite services: composition planning and service selecting. Firstly, the composition planning phase aims at generating a composition plan with a number of actions to be achieved. Actions are usually represented as abstract Web services. The composition plan indicates the appropriate execution order of different abstract Web services by analyzing their functional properties and matching their inputs and outputs' messages [CAST03]. Secondly, the selection phase [ZBDK03] aims at selecting atomic services for each abstract service and integrating them in the composition plan to construct specific composite service. The second process is mainly driven by requirements that imply local constraints on Quality of Service (QoS) properties of atomic Web services and/or constraints concerning the global QoS properties of the resulting composite service. The global QoS values are calculated through aggregation of QoS values of constituent atomic Web services. This involves paying attention on QoS in composition planning phase as it will impact the global composition result. This requires an "on-the-fly" composition process to be able to contextualize the composition plans.

To overcome this challenge, we propose an ad-hoc service composition approach called "Service Farming": given a set of abstract services to be achieved and a list of already discovered atomic services for each abstract service, the service farming objective is to step by step construct composite services by identifying and refining a set of composition rules. The Service Farming approach is an ad-hoc composition in the sense that it simultaneously generates a composition plan and selects services to integrate them into the composition plan. The Service Farming infers the optimal composite service in a reasonable time. The optimal service refers to the composite service found by Service Farming that satisfies both functional and non-functional requirements, and possesses the largest value of service utility. Service utility denotes the total satisfaction received by users from consuming a service. In our approach, the utility is estimated based on QoS values and calculated by a proposed preference-based method. Within the context of the resilient service oriented architecture, Figure 5.1 presents the Service Farming approach as an ad-hoc Web service composition process to reflect changes or requirements imposed by models describing endogenous, exogenous changes as well as business requirements.

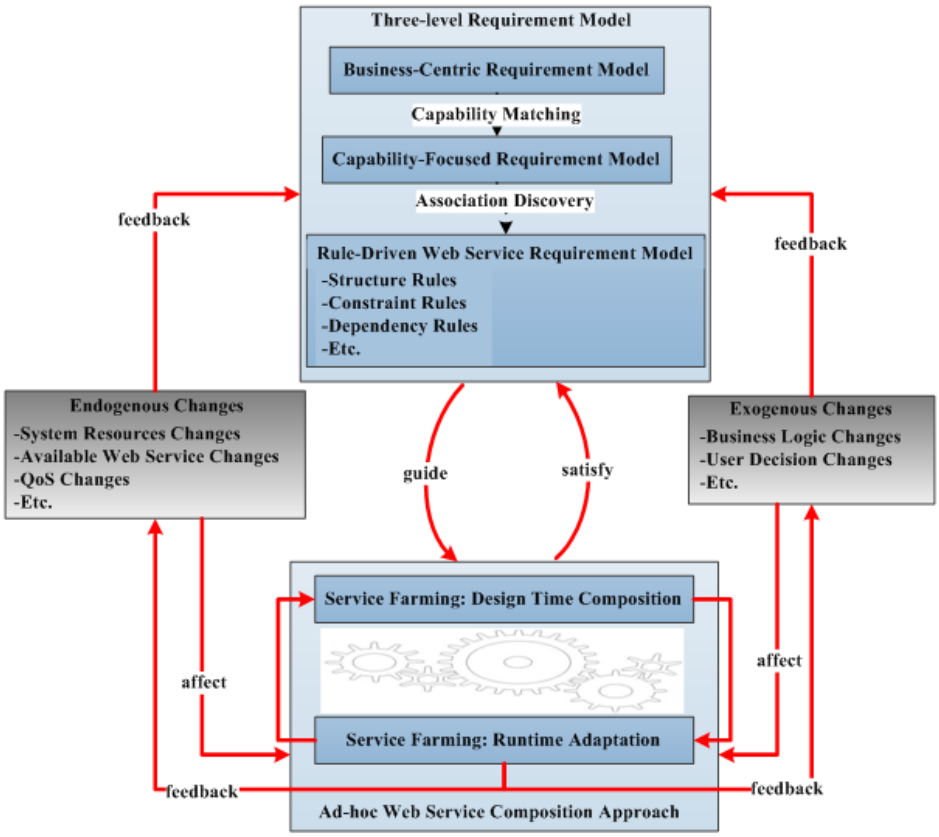

Figure 5.1 The Conceptual rSOC with Service Farming

As depicted in Figure 5.1 the Service Farming approach consists of two main parts: the design time composition and the runtime adaptation. When endogenous and exogenous changes occurs, if the changes do not update the users' requirements, the runtime adaption is activated to reconstruct new composite services adapted to contextual changes; if the changes influence users to update the composition requirements, business-centric requirements or rule-driven Web service composition requirements will be accordingly transformed and updated and thus used to guide the new Web service composition process to reflect responses to changes and updated requirements. In our contribution, we focus on how to compose Web services together and how to adapt to the endogenous and exogenous changes to satisfy users' requirements; we do not model endogenous and exogenous changes in the dynamic environment but we assume that different changes can be detected by system monitors and thus affect our Web service composition processes.

In the following section, we propose the service model to specify Web services and their functional and non-functional properties. We also introduce the QoS aggregation model and necessary formal notations before introducing the service farming composition approach.

5.2 The Service Model

The service arming approach (**SF**) is defined as a tuple:

$$SF = < W, R, QAM, CA >,$$

where *W* is the set of Web services for Service Farming, *R is the* set of composition rules, *QAM* is a QoS aggregation model, and CA is the composition algorithm that applies models and rules into a service set to construct composite services.

The definitions of the Web service set *W* and the composition rule set *R*, which are introduced in Chapter 3 summarized in Table 5.1.

Table 5.1 Summary of the Set of Web Services and the Set of Composition Rules

Notation	Meaning	Notation	Meaning
a	An Abstract Service	R	Set of Composition Rules
t	An Atomic Service	r_s	A Structure Rule
c	A Composite Service	R_s	Set of Structure Rules
s	A Generic Service	r_c	A Constraint Rule
W_a	Set of Abstract Services	R_c	Set of Constraint Rules
W_t	Set of Atomic Services	r_d	A Dependency Rule
W_c	Set of Composite Services	R_d	Set of Dependency Rules
W	Service Set for Service Farming	r_m	A Matching Rule
		R_m	Set of Matching Rules

We introduce QoS aggregation model and some other notations in this section to facilitate our presentation, and then illustrate detail service farming composition algorithm in the next section.

5.2.1 QoS Aggregation Model

The QoS attributes refer to a broad concept that encompasses a number of non-functional properties such as price, availability, reliability, and reputation [OSET02]. QoS properties are applied both to stand-alone Web services and to composite Web services as well. With the proliferation of Web services in business processes and enterprise application integration, QoS properties in Web services are becoming increasingly important from service providers and service consuming perspectives. In order to calculate the QoS of composite services, a QoS aggregation model is needed to take into account the fact that QoS involves multiple dimensions. That is why the QoS of composite services is determined by considering the QoS of its underlying component services.

The QoS of a Web service is represented as a vector of attributes,

$$QoS = <Q_1, Q_2, \ldots, Q_n>,$$

where n is the number of QoS attributes, Q_j is the QoS attribute j. Each QoS attribute is defined as a triple,

$$Q_j = <na_j, ms_j, q_j>,$$

where na_j is the name of this QoS attribute, ms_j is the measurement scale used to measure this QoS attribute, q_j is the value of this QoS attribute.

The QoS values of a service s_i is denoted as

$$s_i.Q = < q_{i.1}, q_{i.2}, \ldots, q_{i.n} >,$$

where $q_{i,j}$ is the value of the QoS attribute j of service s_i, $s_i \in W$.

In our work, we regard QoS as one part of Web service description, and the QoS profile of a Web service can be directly obtained from Web service description languages.

In order to calculate the QoS values of composite services, we develop the QoS aggregation model to derive QoS values of composite services based the QoS values of constituent atomic services. In Service Farming, we consider both QoS attributes and pay particular attention to their measurement scales in order to calculate QoS values of composite services by applying a pessimistic approach presented in [MBKG09]. In the pessimistic approach, the worst QoS values of all the possible executions of the composition are considered to determine the QoS values of a composite service. For example, to determine the response time of a composite service composed by two atomic services following the selection composition pattern, the longest response time is used to determine the composite service QoS value.

Scales of measurement refer to ways in which variables are defined and categorized [JACO99]. Different QoS attributes can be measured with respect to one of the following scales:

1. **Nominal Scale**. The nominal scale just assigns labels to QoS attributes and does not express relationships between values. For example, the value of QoS attribute "payment method" can be either "visa card" or "master card", both of them are payment methods. The relation between "visa card" and "master card" does not exist.

2. **Ordinal Scale**. QoS attributes of ordinal scale have a logical or ordered relationship to each other. This scale permits the measurement of degrees of difference. For example, the value of QoS attribute "security level" can be *low*, *middle*, or *high*. By such scale, the high security level is better than low security level. However, the specific amount of difference between high security level and low security level cannot be identified.

3. **Interval Scale**. For QoS attributes of interval scale, the distance between adjacent points are equal. However, the meaningful zero point does not exist in this scale. For example, the value of QoS attribute "rating point" of a has the scale values between "1"and "10". The difference between vale "1" and val-

ue "2" is the same as the difference between value "9" and value "10". Nevertheless, the value "0" does not exist.

4. **Ratio Scale**. QoS attributes of interval scale have a meaningful zero point. As a result, the distance between adjacent points is equal. For example, the value of QoS attribute "response time" of can be 0 ms, 10 ms, 20 ms.

In our aggregation model, we take into account of measure scales when aggregating QoS attributes by the Service Farming algorithm. In fact, the measurement scale of the same QoS attribute may be different depending on the service providers' descriptions of their Web service profiles. For example, the QoS attribute "security" can be roughly rated by *low*, *middle* or *high*, as an ordinal scale; it can also be defined from the perspective of authentication mechanism e.g., password, fingerprint, or voice, etc. which is a nominal scale. The same QoS attributes maybe measured in different scales and thus should be differently calculated during the aggregation process.

Since different QoS attributes requires different QoS aggregation methods we illustrate in Table 5.2 a representative QoS aggregation methods with respect to QoS attribute characteristics, measure scales and composition patterns. The left column represents the aggregate value of QoS attributes whereas the right side introduces the aggregation formulas for each composition pattern, here *k* represents the iterative times.

Table 5.2 QoS Aggregation Examples

QoS Attribute Name	Measure Scales	Composition Pattern Calculation Methods			
		Sequence	Parallel	Condition	Iteration
Payment(PA)	Nominal	$\bigcap_{i=1}^{n} pa_i$	$\bigcap_{i=1}^{n} pa_i$	$\bigcap_{i=1}^{n} pa_i$	$\bigcap_{i=1}^{n} pa_i$
Security (SE)	Ordinal	$\min(se_i)$	$\min(se_i)$	$\min(se_i)$	$\min(se_i)$
Rating Point (RP)	Interval	$\min(rp_i)$	$\min(rp_i)$	$\min(rp_i)$	$\min(rp_i)$
Response Time (RT)	Ratio	$\sum_{i=1}^{n} rt_i$	$\max(rt_i)$	$\max(rt_i)$	rt*k
Reliability (RE)	Ratio	$\prod_{i=1}^{n} re_i$	$\prod_{i=1}^{n} re_i$	$\min(re_i)$	re^k
Price (P)	Ratio	$\sum_{i=1}^{n} p_i$	$\sum_{i=1}^{n} p_i$	$\max(p_i)$	p*k

5.2.2 Notations for Web Service Composition based on Rules

In order to formalize service farming composition algorithm, we introduce the following notations:

The part of operator (⧐). The notation $\mathbf{t_j \Subset c_k}$ means that the atomic service t_j is one of the constituent atomic services of the composite

service c_k. We regard an atomic service as the constituent service of itself, denoted by $\mathbf{t_j \triangleright t_j}$, $a_j \in W_a$, $t_j \in W_t$, $c_k \in W_c$.

The exclusion operator ($\bowtie$). The notation $\mathbf{c_i \bowtie c_j}$ means that any of the constituent atomic services of c_i and any of the constituent atomic services of c_j do not have the same functional operation.

$\mathbf{c_i \bowtie c_j}$: $\forall t_p \triangleright c_i$, $\forall t_q \triangleright c_j$, $\nexists a_k \models ((s_p \sim a_k) \wedge (s_q \sim a_k))$, $s_p \in W_s$, $s_q \in W_s$, $a_k \in W_a$, $c_i \in W_c$, $c_j \in W_c$.

The completion operator ($\approxeq$). The notatio $\mathbf{c_i \approxeq W_a}$ means that each abstract service from the abstract service set have a correspondent discovered atomic service which is composed in composite service c_i.

$\mathbf{c_i \approxeq W_a}$: $\forall a_i \in W_a$, $\exists t_j \triangleright c_i \wedge t_j \models (t_j \sim a_i)$.

The compliance operator ($\triangleq$). $c_j \triangleq^{m} r_i$ means that the composite service c_j is constructed following the composition rule r_i, where $c_j \in W_c$, $r_i \in R$;

The branch operator (*in*). The notation $\mathbf{\mathit{in}(s_j, r_i)}$ means that the service s_j is a constituent service in the composition rule r_i, $s_j \in W$, $r_i \in R$;

5.3 Service Farming Composition Algorithm

The central idea of the Service Farming composition algorithm is to simultaneously select atomic services and infer their composition patterns following composition rules to ensure that services are composed in the best way with regard to QoS aggregation.

Driven by requirements or constraints issued from various models, at the design time, the Service Farming composition algorithm (CA) relies on four stages (i.e., Service Planting, Service Growing, Service Harvesting, and Service Evaluating), which are executed iteratively through cycles to construct composite services.; at the runtime, when endogenous/exogenous changes occur and requires reconfiguration of composition, two stages, i.e., service substitution and composition replanning are started to response to contextual changes. The flowchart of Service Farming composition algorithm is shown in Figure 5.2.

Service Farming algorithm starts with the input of the Web service requirements (WSR) that is transformed from user's business-centric requirement. At the end of each cycle, new composition rules are generated and thus used to guide the following composition cycles. The Service Farming algorithm ends until an optimal composite service is found.

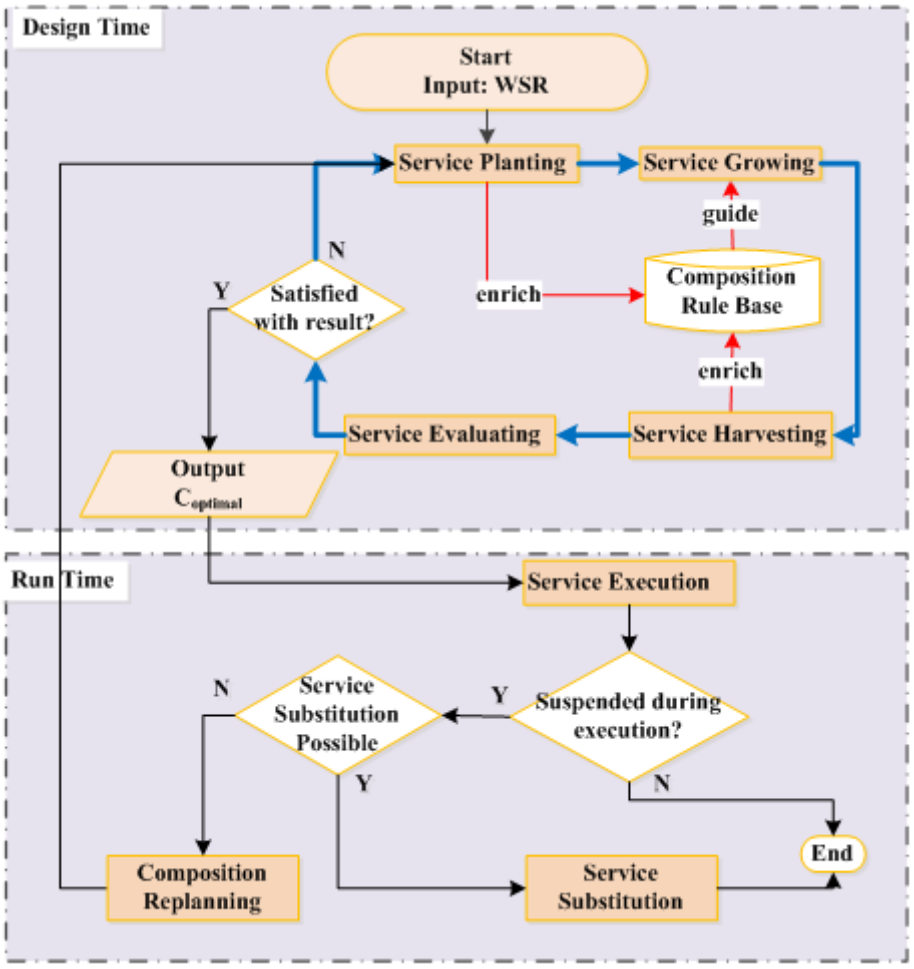

Figure 5.2 The Service Farming Flowchart

At the design time, the Service Planting stage focuses on the specification of composition rules according to WSR; the Service Growing stage constructs entire composite services by simultaneously selecting atomic services and their composition patterns, and the two selected Web services are composed following structure rules (if a structure rule exists indicating the composition pattern between the two selected Web services) or input and output relations; in Service the Harvesting stage, additional structure rules are generated by discovering knowledge represented by specific services and their composition patterns with higher QoS, so as to guide service composition process in the following cycles; Composite services are evaluated by comparing with results from previous cycles in Service Evaluating Stage. If an optimal composite service is found according to the stop condition, the optimal composite service will be deployed and executed; otherwise, the service composition process needs to be restarted over under the guidance of new enriched composition rules.

At the run time, when the execution of the optimal composite service is suspended due to contextual changes, e.g., one or more Web services are unavailable, the Service Farming algorithm will firstly try to substitute the unavailable Web service(s) by using Web services with similar functionalities and utility value; alternatively when contextual changes imply the modification of compositions, our algorithm conserves all information of the executed part of composite service, generates composition rules based on the executed part and starts over from the design time to reconstruct the unexecuted part of composite services.

In the following, we elaborate the service farming algorithm and illustrate its stages with additional details.

5.3.1 The Service Planting Stage

At the beginning of each cycle, the main input for the service farming algorithm is the rule-driven Web service requirements consisting of the derived set of Web services S` and the set of composition rules R` from requirement transformation processes. Two steps are carried out in the Service Planting stage: It firstly focuses on initializing the set of composition rules by combing the composition rules from rule-driven Web service requirement and the composition rules generated in previous cycles. It then eliminates all atomic services that do not satisfy local constraint rules. The process of service planting is formally presented in Figure 5.3.

Input: $WSR = S` \cup R`$
Step 1. Initialize set of composition rule
1.1. $R = R` \cup R$;
Step 2. Initialize Services Set
2.1. $\forall t_x \in S`$,
$t_x \Vdash (R_c) \Rightarrow W_t = W_t \cup \{S`\}$;
2.2. $\forall a_y \in W_a$, $W_a = W_a \cup \{a_y\}$;
2.3. $W_c = \emptyset$;

Figure 5.3 The Service Planting Stage

The set of composition rules R is initialized by combining the set of composition rules from rule-driven Web service requirements and previous cycles together; the set of abstract services W_a is from rule-driven Web service requirement; all atomic services issued from the rule-driven Web service requirements that respect constraint rules are added to the set of atomic services W_a; the set of composite services W_c is empty at the begin-

ning. When user's business-centric requirement changes that leads to the change of rule-driven Web service requirements, the services used to construct composite services will correspondently be updated; if user's requirements are updated during the composition process, the set of composition rules R used to guide the service farming process will also be updated to as to ensure that both functional and non-functional requirements are satisfied during the composition process.

5.3.2 The Service Growing Stage

The Service Growing stage focuses on constructing composite services satisfying both functional and non-functional requirements by incrementally applying composition rules to the set of atomic Web services Wt. Web services are randomly selected and composed following composition rules; in case of no correspondent rules indicating the selected Web services, the selected Web services are randomly composed in parallel or sequential pattern. The process of constructing composite services is formally depicted in Figure 5.4.

Step1. Construct candidate composite services
1.1. $\forall t_i \in W_t$, $W_c = W_c \cup \{t_i\}$;
1.2. $\forall r_{sa.p} \in R$, construct $c_i \triangleq^{m} r_{sa.p}$, $W_c = W_c \cup \{c_i\}$;
Step2. Construct entire composite services
$\forall s_i \in W_c$,
2.1. select $s_j \in W_c$, $s_j \Vdash ((s_i \bowtie s_j) \wedge M(s_i, s_j))$;
2.2. $\exists r_{c.q} \in R$, $r_{c.q} \Vdash (in(s_j, r_{c.q}) \wedge in(s_i, r_{c.q})) \Rightarrow$ construct $c_i \triangleq^{m} r_{c.i}$: $c_i = \langle r_{c.q}.s_L, r_{c.q}.s_R, r_{c.q}.cp \rangle$;
else => randomly construct $c_i = \langle c_i, s_j, cp \rangle$, $cp \in CP$`;
2.3. $W_c = W_c \cup \{ c_i \}$;
2.4. $c_i \approx W_a \Rightarrow$ go to step 3.1;
$\neg(c_i \approx W_a) \Rightarrow$go to step 2.1;
Step 3. Composite service elimination
3.1. $\forall c_i \in W_c$, $\neg(c_i \approx W_a) \Rightarrow W_c = W_c - \{c_i\}$;

Figure 5.4 The Service Growing Stage

The general idea behind the Service Growing stage is to construct composite services based on composition rules as following three steps:

Step 1. Construct candidate composite services. This step aims at constructing preliminary candidate composite services from atomic services. At the beginning of this step, all atomic service are regarded as candidate services; for each abstract structure rule in the composition rule set, we construct a composite service following the structure rule.

To be more precise, we then introduce the algorithm how to construct a composite service c_j following a structure rule $r_{s.p}$. For a concrete composition rule $r_{sc.p}$, we construct composite services by using the left and right sub services of $r_{sc.p}$ and their composition pattern indicated by $r_{sc.p}$. For an abstract composition rule $r_{sa.q}$, we recursively construct composite services corresponding to its left and right sub trees. The detail algorithm is introduced in Figure 5.5.

$r_{s.p}$ is a concrete composition rule, written $r_{rc.p}$.
Step 1. construct $c_i \overset{m}{=} r_{c.q}$: $c_i = < r_{c.q}.s_L, r_{c.q}.s_R, r_{c.q}.cp>$.
$r_{s.p}$ is an abstract composition rule, written $r_{sa.p}$.
Step 1. $\exists a_m \in W_a$, $\exists a_n \in W_a$, $(a_m \Vdash (in(a_m, r_{sa.p})) \wedge (a_n \Vdash (in(a_n, r_{sa.p}))$ => go to step 2;
else => recursively construct $c_k \overset{m}{=} r_{sa.p}.s_L$, $c_l \overset{m}{=} r_{sa.p}.s_R$, and construct $c_i = <c_k, c_l, r_{sa.p}.cp>$;
Step 2. randomly select $t_x \in W_t$, $t_x \Vdash (t_x \sim a_m)$;
Step 3. $\exists r_{sc.q} \in R$, $r_{sc.q} \Vdash (in(t_x, r_{sc.q})) \wedge (r_{sc.q}.cp = r_{sa.p}.cp) \wedge (\exists t_y \in W_t, t_y \Vdash ((t_y \sim a_n) \wedge in(t_y, r_{sc.q}) \wedge M(t_x, t_y))$ => construct $c_i \overset{m}{=} r_{sc.q}$;
else => $\exists r_{d.i} \in R$, $\forall t_y \in W_t$, $r_{d.i} \Vdash (in(t_x, r_{d.i}) \wedge in(t_y, r_{d.i}) \wedge (r_{d.i}.dp = \boxplus) \wedge M(t_x, t_y))$ => select $t_y \in W_t$, and construct $c_i = < t_x, t_y, r_{sa.p}.cp>$;
else => randomly select $t_y \in W_t$, $t_y \Vdash ((t_y \sim a_n) \wedge M(t_x, t_y))$, and construct $c_i = < t_x, t_y, r_{sa.p}.cp>$;

Figure 5.5 The Algorithm of Constructing $c_i \overset{m}{=} r_{s.p}$

This step is illustrated in Figure 5.6 through an example. As illustrated in Figure 5.6, an abstract structure rule $r_{as.1}$ exists as "$a_3 \oslash a_5$", and an atomic service $t_{3.1}$ is randomly selected as the Web service to achieve the operation represented by the abstract service a_3. If a concrete structure rule exists, e.g., $r_{cs.1}$, which indicates the composition pattern between the atomic service $t_{3.1}$ and another atomic service $t_{5.?}$ to achieve the operation represented by the abstract service a_5, a composite service will be constructed following the concrete structure rule $r_{cs.1}$; if not, the dependency rule set will be checked to see if a dependency rule exists, e.g., $r_{d.1}$, which indicates the optimal composed relation between the atomic service $t_{3.1}$ and another atomic service $t_{5.?}$ to achieve the operation represented by the abstract service a_5. If the dependency rule exists, we construct a composite service by using of the two constituent services in the decency rule following the composition pattern defined in the abstract structure rule $r_{as.1}$; If a dependency rule does not exist to indicate the optimal composed relation between the atomic service $t_{3.1}$ and another atomic service $t_{5.?}$, then an atomic service $t_{5.?}$ is randomly selected from all the atomic services achieving the op-

eration represented by the abstract service a_5, and the atomic service $t_{3.1}$ and the randomly selected atomic service are composed together following the composition pattern defined in the abstract structure rule $r_{as.1}$. At last the constructed composite service will be taken as a candidate service.

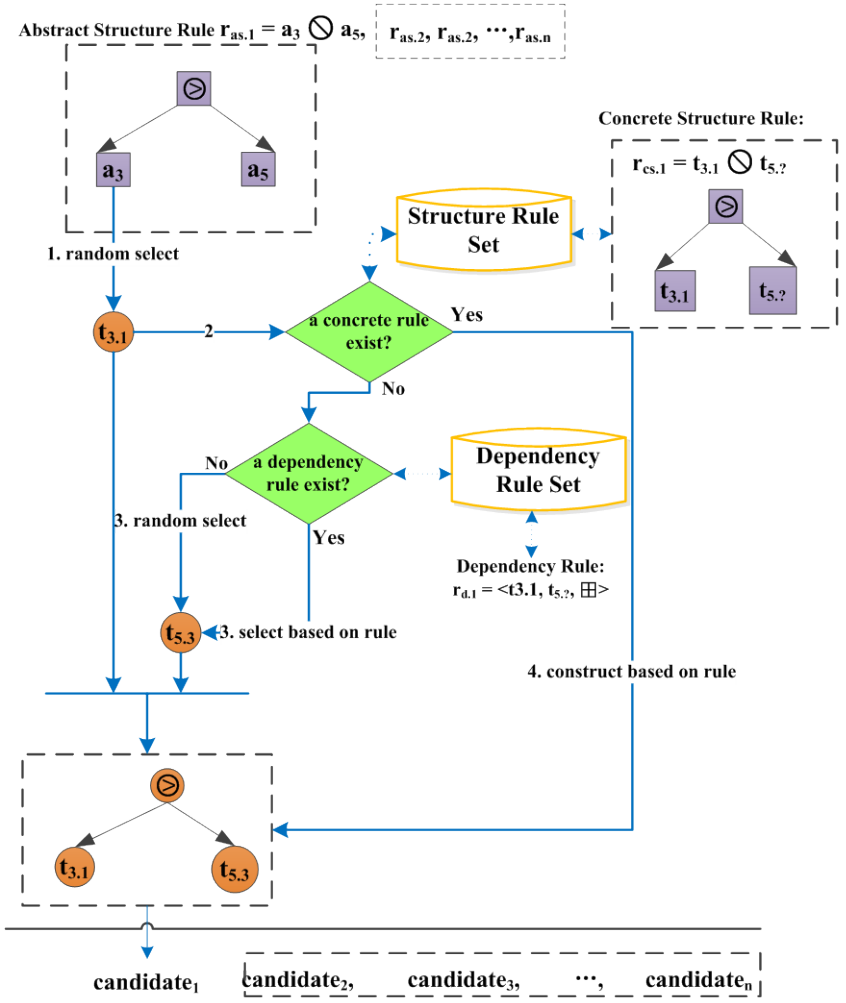

Figure 5.6 Service Growing: Constructing Candidate Composite Services

Step 2. Construct entire composite services. For each candidate service constructed from step 1, we randomly select another candidate service. If a concrete structure exists which indicates the composition pattern between the two selected services, then the two selected services are composed together following the concrete structure rule; otherwise, the two selected services are composed together following their input and output relations. This step repeats until all candidate services contain all required operations defined by abstract Web service set. This step is illustrated in Figure 5.7 through an example.

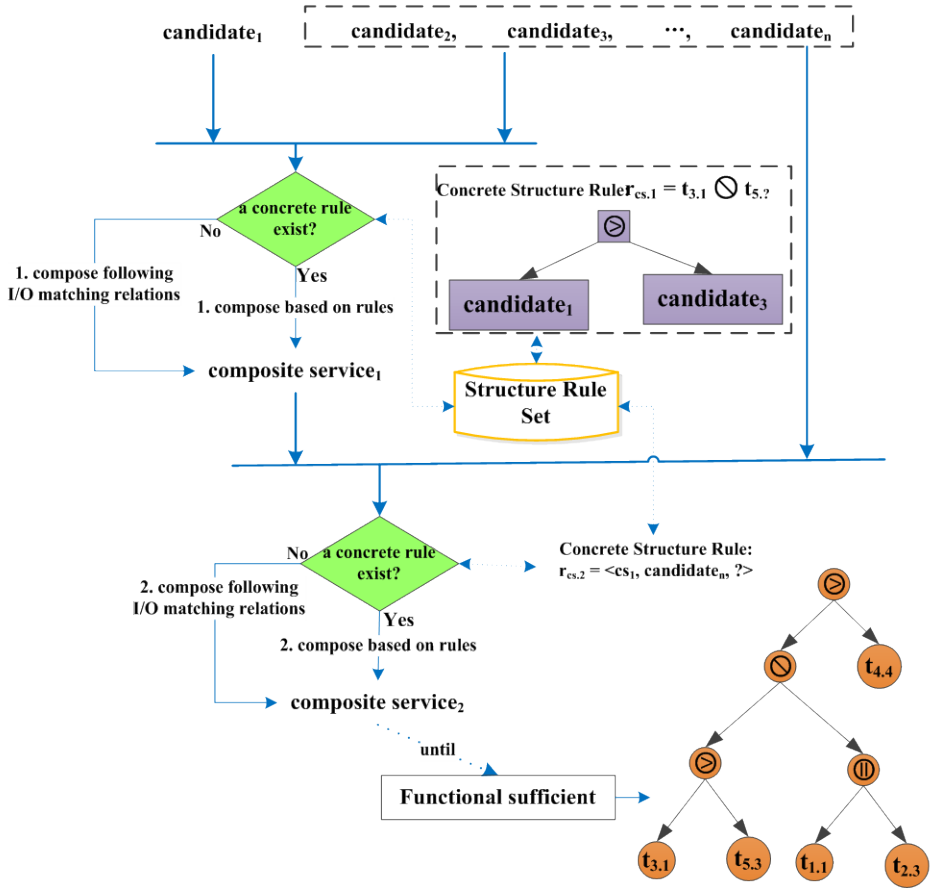

Figure 5.7 Service Growing: Constructing Entire Composite Services

Step 3. Composite service elimination. When step 2 ends, step 3 eliminates all composite services that are generated in the middle of constructing process and do not satisfy the functional requirement defined by abstract Web service set.

5.3.3 The Service Harvesting Stage

Based on the set of composite services (W_c) built by the Service Growing stage, the Service Harvesting stage consists of three big steps as follows; and for each big step, different small steps are proceeded as shown in Figure 5.8.

Step 1. Preference-based Utility Calculation. Service Harvesting stage calculates each service's utility by use of a proposed preference-based utility calculation method;

Step 2. Composite Services Clustering. Service Harvesting stage applies *k-means* cluster algorithm to divide W_c into k clusters based on composite services' QoS similarity;

Step 3. Structure Rule Enrichment. Service Harvesting stage decomposes all composite services from the cluster and enriches structure rule set R_s according to their appearance time.

Step 1.1. QoS Aggregation
Step 1.2. Composite Service Eliminate
Step 1.3. QoS Normalization
Step 1.4. Utility Calculation
Step 2.1. Cluster set of composite service
Step 2.2. Calculate average utility of each cluster
Step 2.3. Find the cluster with maximum average utility W_{max}
Step 3.1. Decompose composite services from W_{max}
Step 3.2. Count appearance time
Step 3.3. Generate structure rules

Figure 5.8 The Service Harvesting Stage

The central idea behind the Service Harvesting stage is that, after clustering, composite services in the same cluster have more QoS similarity than services in different clusters. Regarding the cluster with maximum average utility, we discover the reason why the composite services from the cluster with maximum average utility possess higher utility than the composite services from other clusters. We decompose these composites services into small pieces of composite services and count their appearance times. The composite services with higher appearance time are considered to represent the common characteristics (higher utility) of the cluster with the maximum average utility. In order to promote the appearance of these constituent atomic/composite services in the composition process, we enrich composition rules by use of the composite services with higher appearance time to guide service composition in the following cycles.

5.3.3.1 The Preference-based Utility Calculation

Utility refers to the total satisfaction received by a user from consuming a service. Apparently the higher the service's QoS is, the higher utility the service has. We proposed a utility calculation method to estimate service utility before execution based on Web service's QoS values and user preferences. This method is divided into four steps: QoS aggregation, Composite Service Elimination, QoS normalization and Utility Calculation.

QoS aggregation is carried out by use of the pessimistic model presented in section 6.2 to calculate the QoS values of composite services.

Once obtaining the QoS values of all composite services, composite service elimination deletes all composite services that do not satisfy global constraint rules.

QoS normalization aims at normalizing services' QoS values by transforming them into values between 0 and 1. $q_{i.j}$ represents the value of QoS attribute j of service s_i, and $q`_{i.j}$ represents its normalized value, $s_i \in W_t \cup W_c$. We use $\mathbf{q_{best.j}}$ and $\mathbf{q_{worst.j}}$ to represent the best and worst values for QoS attribute j among all services. For example, the QoS attribute "price", $q_{best}.price$, is the lowest price value among all services while $q_{worst}.price$ is the highest value. QoS normalization is preceded based on different measurement scale of QoS attributes following formulas 5.1 to 5.3.

For QoS attributes measured by nominal scale,

$$q'_{i.j} = 1 \qquad (5.1)$$

For each QoS attribute measured by ordinal scale, n is the number of values that this attribute can obtain; since the value of each attribute j on ordinal scale has its own rank, denoted as $rank_j$ (1^{st}, 2^{nd}, 3^{rd}, …) to represent the order to each other, the value of the attribute j of service i is firstly transformed from a ordinal value to a number, denoted as $q^*_{i.j}$: $q^*_{i.j} = n - rank_j + 1$, and then $q^*_{i.j}$ is transformed to a value between 0 and 1 following formula 5.2.

For example, for the QoS attribute "security" which has three values as *low*, *middle*, and *high*, we get $n = 3$, $v_{high} = 3$, $v_{middle} = 2$, $v_{low} = 1$, $q_{best} = 3$, $q_{worst} = 1$.

$$\mathbf{q'_{i.j}} = \begin{cases} \frac{|\mathbf{1} - \mathbf{q^*_{i.j}}|}{\mathbf{n-1}} & \textbf{if } \mathbf{q_{best.j}} \neq \mathbf{q_{worst.j}} \\ \mathbf{1} & \textbf{else} \end{cases} \qquad (5.2)$$

For QoS attributes measured by interval or ratio scale,

$$\mathbf{q'_{i.j}} = \begin{cases} \frac{|\mathbf{q_{worst.j}} - \mathbf{q_{i.j}}|}{|\mathbf{q_{best.j}} - \mathbf{q_{worst.j}}|} & \textbf{if } \mathbf{q_{best.j}} \neq \mathbf{q_{worst.j}} \\ \mathbf{1} & \textbf{else} \end{cases} \qquad (5.3)$$

Utility calculation is based on user's predefined preference on QoS attributes; all QoS attributes have the same preference if user do not indicates the QoS preference. A user's QoS preference is defined as a priority indication to n QoS attributes of all services: when service utility is calculated, the QoS attributes with higher priority will be added more weight than other QoS attributes with lower priority.

For QoS attribute q_i, the user may indicate the priority level (i.e., 1^{st}, 2^{nd}, 3^{rd}…), denoted as $\mathbf{p_l}.\mathbf{q_i}$, and a QoS attribute has the least priority if a user does not indicate its priority; the number of all priority levels are denoted as $\mathbf{p_n}$. For example, a user regards the price and response time are the first and second important attributes influencing his satisfaction when con-

suming a service, the other two attributes of availability and security have the lowest priority level. In this case, $p_n = 3$, p_l.price = 1, p_l.response_time = 2, p_l.availability = 3 and p_l.security = 3.

After normalization, the normalized QoS of s_i is denoted as $s_i.Q` = <q`_{i.1}, q`_{i.2}, ..., q`_{i.n}>$, $q`_{i.j} \in [0, 1]$, $s_i \in W_t \cup W_c$. The better the $q_{i.j}$ is, the more the value of $q`_{i.j}$ is close to 1.

The utility of service s_i is denoted as $\mathbf{s_i.u}$, and it is calculated by use of formula 5.4.

$$\mathbf{s_i.u} = \frac{\sum_{j=1}^{n}\left((p_n - p_l.q_{i.j} + 1) * q`_{i.j}\right)}{\sum_{j=1}^{n}(p_n - p_l.q_{i.j} + 1)} \quad (5.4)$$

The proposed preference-based utility calculation method is formalized in Figure 5.9; while an example of utility calculation is illustrated in Table 5.3.

Step 1.1. QoS Aggregation.
$\forall c_i \in W_c$, calculate $c_i.Q$.
Step 1.2. Composite Service Elimination.
$\forall c_i \in W_c$, $c_i \nvDash (Rc) \Rightarrow W_c = W_c - \{c_i\}$.
Step 1.3. QoS Normalization.
$\forall c_i \in W_c$, normalize $c_i.Q$ following formula 5.1, 5.2 and 5.3.
Step 1.4. Utility Calculation.
$\forall c_i \in W_c$, calculate $c_i.u$ following formula 5.4.

Figure 5.9 The Utility Calculation

Table 5.3 Utility Calculation Example

QoS Attributes	<	Security	ResponseTime	Reliability	Price	>
QoS Vector	<	High	32	97%	50	>
NormalizedQoS	<	1	0.7894	0.8924	0.5689	>
Utility	0.8126					

5.3.3.2 Clustering Composite Services

After calculating services utilities, we apply k-means algorithm to cluster the set of composite services W_c and find the cluster with maximum average utility, denoted as W_{max}. Three steps are carried out:

Step 2.1. Cluster W_c by using the k-means algorithm and getting *k* clusters;

Step 2.2. Calculate the average utility of each cluster;

Step 2.3. Find W_{max}.

Roughly speaking, the *K*-means provides a simple and efficient way to classify a set of data points into a fixed number of clusters [HAKP12]. Data points are characterized by their n-dimensional vector $<x_1, x_2, \ldots, x_n>$. The main idea of k-means is to define a centroid $c = <x_{c.1},$

$x_{c.2}$, …, $x_{c.n}$> for every cluster and to associate each data point, dp_i = <$x_{i.1}$, $x_{i.2}$, …, $x_{i.n}$> to the appropriate cluster by computing the shortest n-dimensional Euclidian distance between the data point and each centroid; And then the clusters' centroids are then recomputed and data points are re-associated to each cluster. The process repeats as introduced as follows:

Step 1. The algorithm arbitrarily selects *k* points as the initial cluster centers ("means").

Step 2. Each point in the dataset is assigned to the closed cluster, based upon the Euclidean distance between each point and each cluster center.

Step 3. Each cluster center is recomputed as the average of the points in that cluster.

Steps 2 and 3 are repeated until all clusters converge. The convergence normally means that either no observations change clusters when steps 2 and 3 are repeated or when the changes do not make a difference in the definition of the clusters.

The algorithm clusters the set of data points into k groups, where k is provided as an input parameter. Figure 5.10 illustrates an example of clustering process based on k-means algorithm.

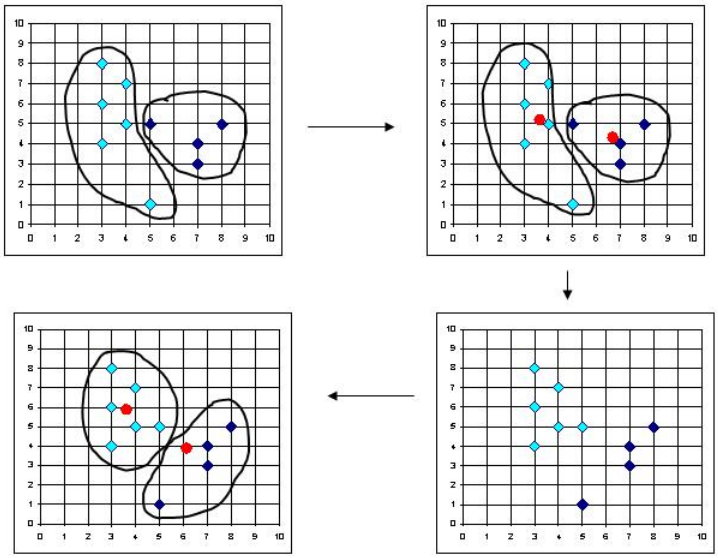

Figure 5.10 K-means Clustering Example

In our context, $\forall c_i \in W_c$, $s_i.Q` = <q`_{i.1}, q`_{i.2}, ..., q`_{i.n}>$ is regarded as the vector to apply cluster algorithm, and the distance formula 5.5 is used to measure QoS similarity between two services s_m and s_n:

$$\mathbf{D}_{(c_m,c_n)} = \sqrt{\sum_{j=1}^{n}(q`_{m,j} - q`_{n,j})^2} \qquad (5.5)$$

Table 5.4 illustrates an example of the composite service set before clustering, and the composite service clusters after clustering.

Table 5.4 Clustering Example

CompositeService	<	Security	ResponseTime	Reliability	Price	>
c_1	<	1	0.7894	0.8924	0.5689	>
c_2	<	0.7894	0.6805	0.6781	0.9857	>
…			…			
c_n	<	1	0.4738	0.9784	0.3493	>

⇊ Clustering ⇊

Cluster No.	Composite Services
1	{ c_1, c_3, …, c_i }
2	{ c_5, c_7, …, c_j }
…	{ c_2, c_4, …, c_m }
k	{ c_6, c_8, …, c_n }

5.3.3.3 The Structure Rule Enrichment

The structure rule set is continuously enriched in each cycle in order to guide service composition in further cycles. Structure rule enrichment consists of three steps as presented in Figure 5.11.

Step 3.1. Decompose composite services

$\forall c_i \in W_{max}$,

decompose c_i into repeatable set W_{temp}:

recursively construct: $c_1=c_i.s_L$, $c_2=c_i.s_R$, $c_3=(c_i.s_L).s_L$, $c_4=(c_i.s_L).s_R$,…;

Step 3.2. Count appearance time

$\forall c_j \in W_{temp}$, count c_j.at in W_{temp};

Step 3.3. Generate composition rules

3.1.$\forall c_j \in W_{temp}$, c_j.at>T => generate $r_{sc.p}$= <$c_j.s_L$, $c_j.s_R$, c_j.cp>;

3.2. R = $\{r_{sc.p}\} \cup R$;

Figure 5.11 The Structure Rule Enrichment

The notation **c_i.at** is used to represent the appearance time of a composite service c_i; while **T** is the parameter indicating the appearance time threshold. When the appearance time of a composite service c_i is more than T in the temporary and repeatable composite service set W_{temp}, a composition rule will be created based on the composite service c_i. An example of the structure rule enrichment process is illustrated in Table 5.5. In the given example, the appearance time threshold T is 12, while the appearance time of three composite services c_1, c_2 and c_3 are more or equal to 12, and

the set of composition rules are enriched by three structure rules created based on the composite service c_1, c_2 and c_3.

Table 5.5 Structure Rule Enrichment Example

W_{temp}	AppearanceTime	T=12	Results
c_1	15	⇉Save to R⇉	$R=R \cup \{c_1, c_2, c_3\}$
c_2	14		
c_3	12		
c_4	11		
…	…		
c_q	1		

5.3.4 The Service Evaluating Stage

Service Evaluating stage evaluates the constructed composite services by Service Farming and determines whether an optimal composite service is found.

The composite service with the highest utility in each cycle is denoted as $\mathbf{c_{max}.l_n}$, and its utility is denoted by $\mathbf{u_{max}.l_n}$, where $\mathbf{l_n}$ is the number of the service farming cycle.

From the first cycle to current cycle, the optimal composite service found by the Service Farming algorithm, denoted by $\mathbf{c_{optimal}}$, is defined as the composite service with the highest utility of all composite services in all cycles. The utility of the optimal composite service is denoted by $\mathbf{u_{optimal}}$.

The stop condition (**SC**) of Service Farming composition algorithm is that within a period of t service farming cycles, $u_{optimal}$ are not augmented any more:

$$SC: u_{max}l_n = u_{max}l_{n-1} = u_{max}l_{n-2} = \ldots = u_{max}l_{n-t} = u_{optimal} \quad (5.6)$$

This means that an optimal solution for the given composition problem is found out and the optimal utility cannot be improved anymore by our approach.

The Service Evaluating stage in cycle l_n is formalized in Figure 5.12, and an example of the service evaluating stage is presented in Table 5.6. In the given example, the utility of the optimal composite service $u_{optimal}$ is 0.8613 from cycle l_{n-t} to l_n, and within the period of t service farming cycles, $u_{optimal}$ are not augmented any more. In this case, Service Farming will stop and execute the optimal composite service $c_{optimal}$.

Step 1. Find $c_{max}.l_n$, $u_{max}.l_n$ and $c_{optimal}$.
Step 2. SC is true => output and execute $c_{optimal}$;
else =>go to Service Planting stage of cycle l_{n+1}.

Figure 5.12 The Service Evaluating Stage

Table 5.6 Service Evaluating Example

$Cycle(l_n)$	1	2	…	l_{n-t-1}	l_{n-t}	…	l_n
$U_{optimal}$	0.4854	0.5254	…	0.8429	0.8613	…	0.8613

5.4 The Dynamic Reconfiguration of Composite Web Services

In the dynamic composition environment, the execution of composite services may be interrupted due to endogenous and exogenous changes in the environment. For examples, these changes may include variations in Web service QoS, changes in business goals and requirements, detecting fault events at runtime, assessing security vulnerably, or employing new organizational decisions. Instead of considering changes as exceptions at runtime and interrupt the composite Web service's execution, we handle exceptions with alternative solutions ranging from self-managed solutions, such as Web service substitution as in fault tolerance, reconfiguring composite Web services such as selecting new Web services based on their QoS, or replanning the Web service composition process such as recomposing unexecuted constituent Web services in composite Web services or replanning the entire composite Web services, depending on the severity of changes. Normally, the execution of a composite service is interrupted and then cancelled when endogenous or exogenous changes occur; in our research, we assume that a framework exists to suspend the execution of composite services, to conserve all information regarding the part of composite service which is already executed, and able to recover the execution of composite service when an alternative composition is regenerated.

The general idea of our dynamic reconfiguration can be summarized as follows: when the execution of composite services is interrupted because of endogenous/exogenous changes of dynamic environment, the service farming algorithm firstly tries to substitute the services which are unavailable or change their functional/non-functional properties. On the other hand, when endogenous/exogenous changes, such as user decision or business logic changes, imply the re-composition of the composite services, the service farming algorithm conserves all information regarding the executed part of composite service, generates composition rules based on the executed part and starts over from the design time to re-construct the suspended part of composite service.

In the following section, we elaborate the service substitution and the composition replanning as two stages that make the ad-hoc Web service composition approach respectively respond to endogenous changes (e.g., Web service unavailable) and exogenous changes (e.g., business logic).

5.4.1 The Service Substitution Stage

The Web service substitution is the process of replacing a Web service by another Web services with same functionality and similar QoS properties to ensure fault tolerance. When substituting Web services, among the Web service candidates that offer desired functionalities, the Web service composition process might select a Web service candidate over others based on its non-functional attributes that best improves the overall composition.

The Web service substitution process proceeds with the following principles to choose candidate Web services for substitution:

Principle 1. The Web service for substitution provides same functionalities as the Web service to be substituted.

Principle 2. The Web service for substitution has similar QoS profile as the Web service to be substituted.

Principle 3. The Web service for substitution has higher utility values than other candidate Web services.

Principle 4. The Web service for substitution satisfies both local and global constraint rules.

For the optimal composite service $c_{optimal}$, we denote W_u as the set of atomic services to be substituted in $c_{optimal}$, and $c`_{optimal}$ is used to represent the new composite service to be executed after substituting all atomic services from W_u. For each $t_i \in W_u$, we use Wt_i to represent the set of all available Web services that provide the same functionalities as t_i, thus we have,

$$\forall t_i \in W_u,\ \forall t_j \in Wt_i,\ \exists a_m \in W_a,\ a_m \models ((t_i \sim a_m) \wedge (t_j \sim a_m)).$$

In order to satisfy the four substitution principles, we introduce the *substitution coefficient* (S_c) to quantify the similarity degree between two atomic services that provide the same functionality following formula 5.7.

$$S_c(t_i,t_j) = \frac{t_j.u}{D(t_i,t_j)} \qquad t_i \in W_u,\ t_j \in Wt_i \tag{5.7}$$

where the $D(t_i, t_j)$ is the distance between of t_i and t_j QoS profiles, and the $t_j.u$ is the utility of the atomic service t_j. The higher the $S_c(t_i,t_j)$ is, the more the two services are globally similar. We develop the Service Substitution algorithm in Figure 5.13 to integrate the Web service substitution with the context of ad-hoc Web service composition.

Input: Wu $\neq \emptyset$
Step 1. Create Substitution Set
$\forall t_i \in W_u$,
1.1. $Wt_i = \emptyset$;
1.2. $\exists a_m \in W_a, \exists t_j \in W_t, \ t_j \models ((t_i \sim a_m) \wedge (t_j \sim a_m)) \Rightarrow Wt_i = Wt_i \cup \{t_j\}$;
Step 2. Calculate Substitution Coefficient
$\forall t_i \in W_u$,
2.1. $\forall t_j \in Wt_i$, calculate $S_c(t_i, t_j)$;
2.2. Find $t`_j$, $\forall t_j \in Wt_i$, $t`_j \models (S_c(t_i, t`_j) \geq S_c(t_i, t_j))$;
2.3. Generate $r_{d.i} = <t_i, t`_j, \circledcirc>$
Step 3. Evaluate c`$_{optimal}$
$\forall t_i \in W_u$,
3.1. $\exists r_{d.i} \in R, \exists t`_j \in W_t, t`_j \models (in(t_i, r_{d.j}) \wedge in(t`_j, r_{d.j}) \wedge r_{d.j}.dp = \circledcirc)$ => substitute $t_i$ by $t`_j$, and construct $c`_{optimal}$.
3.2. $c`_{optimal} \models (R_c)$ => execute c`optimal.
else =>go to composition replanning stage.

Figure 5.13 The Service Substitution Algorithm

Three steps are proceeded to substitute services, which are introduced as follows:

Step 1. Create Substitution Set. For each atomic service to be substituted, we find all atomic services with same functionality as candidate services for substitution.

Step 2. Calculate Substitution Coefficient. For each candidate service for substitution, we calculate its substitution coefficient, and choose the atomic service with the highest value of substitution coefficient for substitution.

Step 3. Evaluate Composite Service. Before substituting, we calculate if the new generated composite service satisfies constraint rules: if yes, we proceed the substitution process and execute the new generation composite service; otherwise, we go to the composition planning stage to recompose services to response to changes.

Figure 5.14 illustrates a service substitution example based on our proposed resilient service-oriented computing concept.

As shown in Figure 5.14, when an endogenous change occurs in which one atomic service in the composition result is no more available, service substitution stage starts to substitute the atomic service and construct a new composite service to response to the endogenous change and satisfy users' initialized requirements.

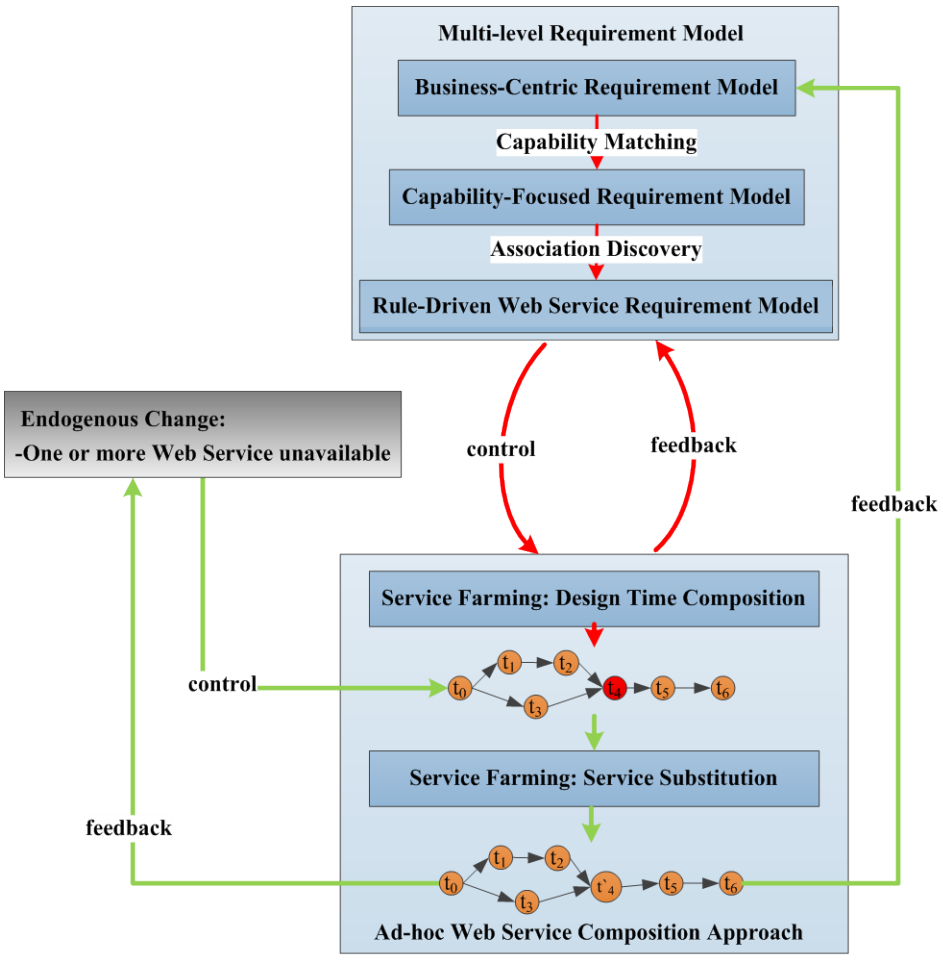

Figure 5.14 Service Substitution Example

5.4.2 The Composition Replanning Stage

The service substitution within the ad-hoc Web service composition approach handles the exception case in which a Web service is substituted. In some situations, when business logic or users' decision change, simple substitutions of services cannot provide effective solutions to response to changes. In this case, when the execution of composite service is suspended because composition requirements have changed, the unexecuted part of composite service needs to be reconstructed while preserving the result of the executed Web services. This re-composition is challenging as, without user's intervention, new constraints and new requirements used to construct the unexecuted part of composite service need to be derived from on the initialized constraints and requirements used in the previous Web

service composition process. In addition, combining the executed part and the new constructed part should satisfy user's initialized requirements.

Roughly speaking, when the $c_{optimal}$ is constructed at the design time, it is executed at the runtime to satisfy business requirements. In the case that the execution of $c_{optimal}$ is suspended and cannot be solved by simply substituting services, we extend the ad-hoc Web composition algorithm to develop a run-time adaptation mechanism to replan the unexecuted part of the constructed optimal composite service.

We denote the executed part of $c_{optimal}$ as $\mathbf{c_{pre}}$, and the suspended part of $c_{optimal}$ as $\mathbf{c_{post}}$. The c_{pre} and c_{post} are both composite services. The problem of the replanning Web service composition attempts to construct the c_{post} based on the $c_{optimal}$ and the c_{pre} and satisfying user's requirements. According to W_a, W_t, R_s, and R_c used in the process of constructing the $c_{optimal}$, we respectively denote $W`_a$, $W`_t$, $R`_s$, and $R`_c$ as the set of abstract services, the set of atomic services, the set of structure rules and the set of constraint rules used in the process of constructing the c_{post}, while R_d remains the same as originally derived from users' business-centric requirements.

The composition replanning stage can be divided into five steps as follows: 1) jointly integrate input messages of $c_{optimal}$ and output messages of c_{pre} as the input messages of the c_{post}; 2) generate $W`_a$, $W`_t$; 3) generate $R`_s$, and $R`_c$; 4) apply the Service Farming composition algorithm to find an optimal result for c_{post}; 5) integrate the output messages of c_{pre} and c_{post} as the final output messages of the composite Web service. Finally the output of composition replanning stage is the optimal composite service by composing c_{pre} and c_{post}. The service substitution stage of the service farming composition algorithm is formalized in Figure 5.15.

By using the same QoS attributes example introduced in Table 5.2,

Table 5.7 illustrates how to generate new global constraint rule $r`_{gc.i}$ for composition replanning from the initialized global constraint rule $r_{gc.i}$. The left and middle columns show the constrained QoS attributes and operator of $r`_{gc.i}$, which are same to $r_{gc.i}$; while the right column indicates how to calculate the constraint value of the $r`_{gc.i}$ based on the constraint value of $r_{gc.i}$.

Figure 5.16 illustrates a composition replanning example based on our proposed resilient service-oriented computing concept.

Step 1. Generate $I.c_{post}$

$I.c_{post} = I.c_{optimal} \cup O.c_{pre}$;

Step 2. Generate $W`_a$ and $W`_s$

2.1. $\forall a_i \in W_a$, $(\nexists t_i \rightarrow c_{pre} \models (t_i \sim a_i)) \Rightarrow W`_a = W`_a \cup \{a_i\}$;

2.2. $\forall t_i \in W_t$, (s_i is unavailable) $\Rightarrow W_t = W_t - \{t_i\}$;

2.3. $\forall t_i \in W_t$, $(\exists a_j \in W`_a \models (t_i \sim a_j)), \Rightarrow W`_t = W`_t + \{t_i\}$;

Step 3. Generate R`s and R`c

3.1. $\forall r_{as.i} \in R_s$, $(\nexists t_i \rightarrow c_{pre} \models (t_i \sim a_j \wedge in(a_j, r_{as.i})) \Rightarrow R`_s = R`_s \cup \{r_{as.i}\}$;

3.2. $\forall r_{cs.i} \in R_s$, $(\nexists t_i \rightarrow c_{pre} \models (in(t_i, r_{cs.i})) \Rightarrow R`_s = R`_s \cup \{r_{cs.i}\}$;

3.3. $\forall r_{lc.i} \in R_c$, $(\nexists t_i \rightarrow c_{pre} \models (in(t_i, r_{lc.i})) \Rightarrow R`_c = R`_c \cup \{r_{lc.i}\}$;

3.4. $\forall r_{gc.i} \in R_c$, generate $r`_{gc.i}$ following table 6.3, $R`_c = R`_c \cup \{r_{gc.i}\}$;

Step 4. Construct c_{post}

4.1. Initialize composition rule base $R` = R`_s \cup R`_c \cup R_d \cup R_m$;

4.2. Apply Service Farming based on $W`_a$, $W`_s$, and R, and generate cpost

Step 5. Generate $O.c_{post}$

$O.c_{post} = O._{pre} \cup O._{cpost}$

Output: $c_{optimal} = <c_{pre}, c_{post}, \otimes>$

Figure 5.15 The Web Service Composition Replanning

Table 5.7 Calculation of QoS Constraints in Run Time

$r`_{gc.i}.Q_j = r_{gc.i}.Q_j$	$r`_{gc.i}$.operator	$r`_{gc.i}$.value
Payment(PA)	$r_{gc.i}$.operator	$r_{gc.i}$.value
Response Time (RT)	$r_{gc.i}$.operator	$r_{gc.i}$.value - c_{pre}.rt-sf.rt
Availability (AV)	$r_{gc.i}$.operator	$r_{gc.i}$.value/c_{pre}.av
Reliability (RE)	$r_{gc.i}$.operator	$r_{gc.i}$.value/c_{pre}.re
Price (P)	$r_{gc.i}$.operator	$r_{gc.i}$.value - c_{pre}.p

From Figure 5.16 we can see that, when an exogenous change occurs in which user decision changes and thus requires the reconfiguration of Web service composition, composition replanning stage starts to conserve all information regarding the executed part of composite service c_{pre}, and start over from the design time to re-construct the unexecuted part of composite service c_{post}, and finally the two parts of composite service c_{pre} and c_{post}, are taken as the final composition result to response to the exogenous change and satisfy user's updated requirements.

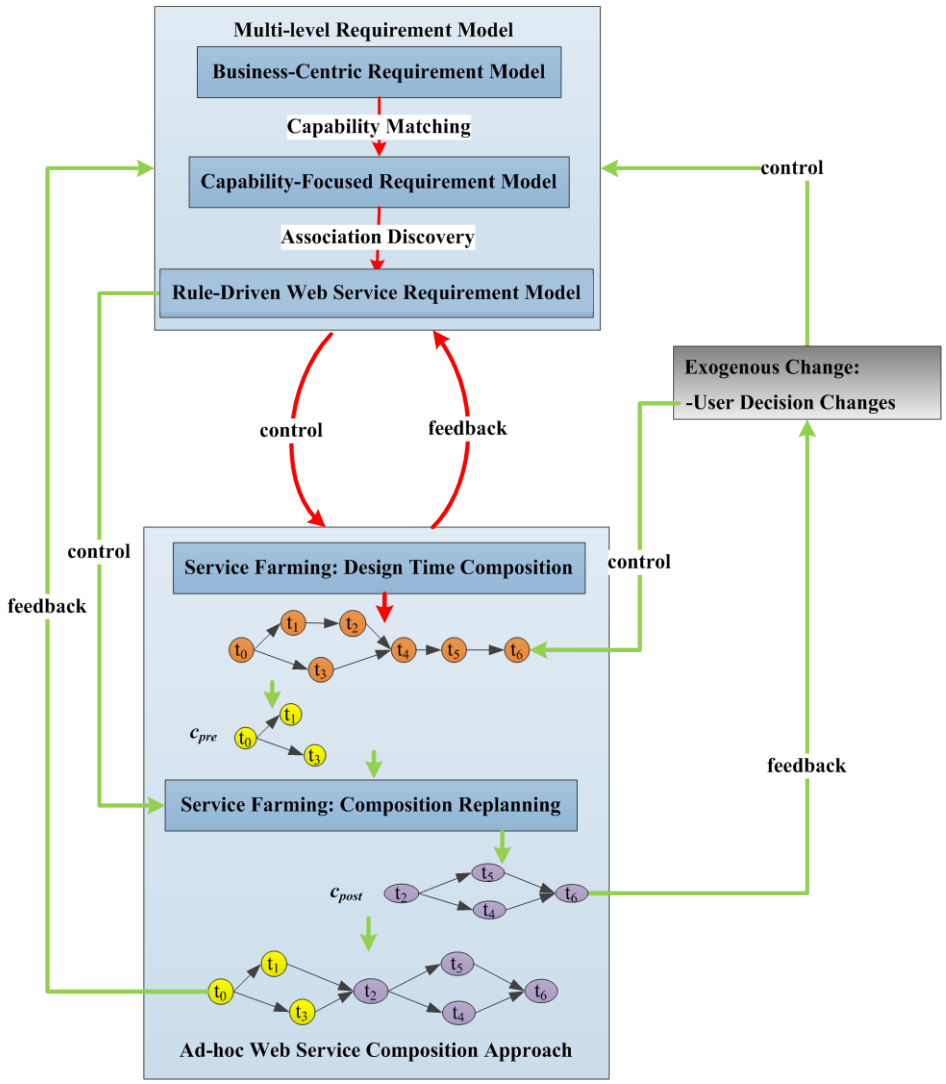

Figure 5.16 Composition Replanning Example

5.5 Conclusion

In this chapter, we focus on the ad-hoc Web service composition approach within the framework of the resilient service-oriented architecture. Based on the rule-driven Web service composition model, the Service Farming composition algorithm adapts the composition process to both static and dynamic environments. The Service Farming aims at, in a reasonable time, constructing optimal composite services meeting multiple constraints, particularly business requirements, while maximizing user's satisfaction. Instead of selecting atomic services based on a predefined composition plan, the Service Farming relies on a heuristic to identify and

refine a set of composition rules through several cycles. The composition process is achieved by simultaneously selecting atomic services and inferring composition patterns between selected services following composition rules.

The ad-hoc composition approach ensures that Web services in static environment can be composed in an optimal way with regard to constraints on QoS values. In addition, it ensures that the composite Web service is adaptable to endogenous and exogenous changes in dynamic environments while satisfying multiple constraints (e.g., business re requirements). The Service Farming provides several advantages such as:

1. Ad-hoc composition: The composition process is achieved by simultaneously selecting atomic services and inferring composition patterns to ensure that services are composed in the best way with respect to the QoS aggregation. Since Web services are usually composed by discovering and selecting services based on predefined composition plans, the aggregated QoS values of composite services are thus calculated by following the composition plan. This may lead to not having the best way to aggregate QoS values of composite services because composition plans are generated without considering QoS profiles of specific atomic services.

2. Flexibility: Instead of giving a complete composition plan, by means of the Service Farming approach, user is allowed to simply define a set of abstract services without composition plans or even with partial or whole composition plan(s). In these cases, the Service Farming approach is able to flexibly construct optimal composite services.

3. Extensibility: The Service Farming approach is a rule-based approach which specify different internal and external variables in the environment with various rules, and applies them to guide the service composition process. The categories of rules can be easily enriched by new rules to reflect constraints in new models (e.g., security rules, resources rules, mediation rules, …) and consequently influence the ad-hoc composition process. Since the ad-hoc Web service composition algorithm is based on multiple constraints, new rules can be easily considered and processed without updating the algorithm logic and structure.

Chapter 6

Implementation Architecture

Implementation Architecture 153
Experiment Results 157
Conclusion 162

Abstract: In this chapter, we illustrate the development of the resilient service oriented architecture based on models that affect and are affected by each other to ensure adaptability and flexibility of SOA-based applications or business processes in dynamic environments. In addition, we present the technical architecture to implement the prototype of the ad-hoc Web service composition approach based on users' requirements. The prototype has been developed in Java and used to conduct experiments and measure its performance.

6.1 Implementation Architecture

In order to demonstrate the feasibility of our contribution, we have developed a modular framework and implemented as illustrated in Figure 6.1, comprising different modules: the business-centric requirement manager, the rule-driven Web service requirement manager, the capability matching manager, the association discovery manager, the ad-hoc Web service composition manager and the C-WSDL Web service registry.

The business-centric requirement manager is specified with a customized version of the SBVR visual editor [REF13B] developed by the OPAALS project to build SBVR-based Knowledge. In our prototype, the capability matching manager implements the transformation process to deduce from business-centric requirements expressed with SBVR terms to the set of capability instances, while the association discovery manager implements the transformation from capability instances to rule-driven Web service requirements. The principles guiding the transformation processes in the capability matching manager and the association discovery manager are

realized by the Query/View/Transformation (QVT) language, which is a standard language for model transformation defined by Object Management Group [REF11]. Based on the C-WSDL registry introduced in [MTYB04], we have developed the C-WSDL registry to store Web service WSDL profiles as well as their associated capability instances. The Web service composition manager is developed based on our work on Web service composition [LIBB13D], and finally the composite service is represented based on BEPL to be executed.

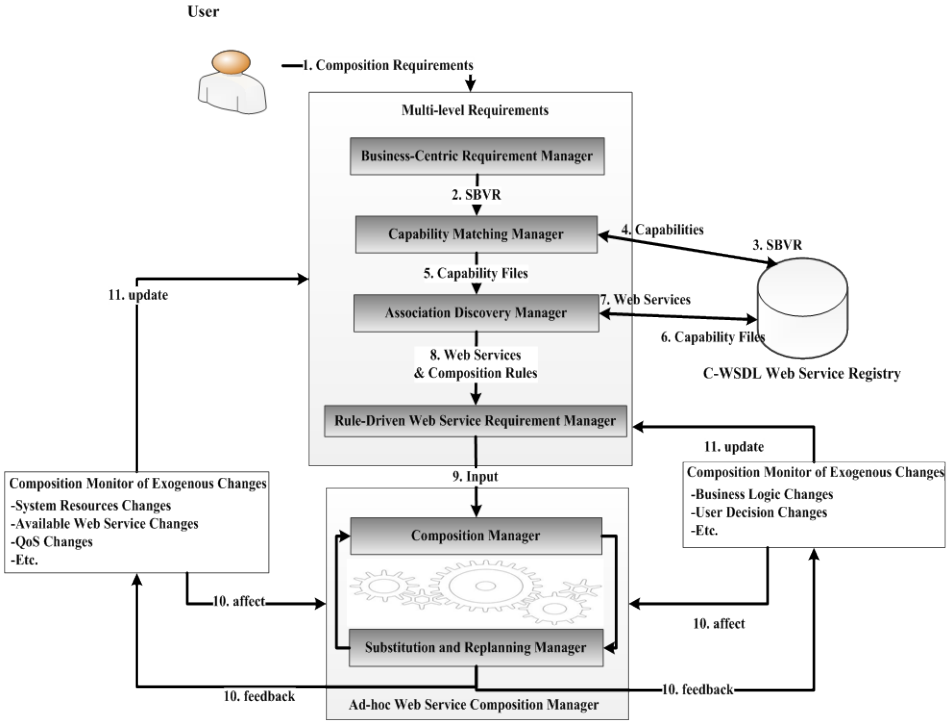

Figure 6.1 The Resilient SOA Framework

In this prototype, when business users express the business-centric requirement based on SBVR, these requirements are firstly sent to business-centric requirement manager. The requirement manager analyzes and selects useful SBVR patterns (e.g., Fact Type, Noun, etc.) and sends them to the capability matching manager to derive appropriate capability instances from the C-WSDL registry by following capability matching principles. The association discovery manager uses the derived capability instances to discover Web services and generate composition rules following the association discovery principles. At last, Web services and composition rules are forwarded to the composition manager to compose services, satisfying users' business-centric requirements. The composition monitors simply detect contextual changes at design/runtime during the composition process. The substitution and replanning manager adapts composite ser-

vices to contextual changes; the business-centric requirement manager and rule-driven requirement manager update the composition requirements if users change their business requirements. New updated requirements are thus used to guide a new composition process.

As illustrated in Figure 6.2, the module that implements the Web service composition manger consists of five packages: the Service Farming Registry, the Service Model, the Binary Tree Model, the Composition Rule Model, and the Composition Pattern. The Service Model represents the Web service descriptions with functional and non-functional properties, while the Service Farming Registry specifies different category of Web services, i.e., atomic service, abstract service and composite services and provides additional information related to the service farming algorithm such as normalized QoS, and service utility. The Composition Rule Model specifies and manipulates different categories of composition rules; The Binary Tree Model is mainly used as a flexible data structure to process composite service and composition rules. The composition pattern package implements the clustering algorithm and provides methods to decompose composite services and enrich the set of composition rules.

All these packages are connected through interfaces to increase modularity and are manipulated by the Service Farming Manger to implement the ad-hoc Web service composition process. The packages are general illustrated in Figure 6.2; while their implementation details are briefly illustrated in the UML class diagram in Figure 6.3.

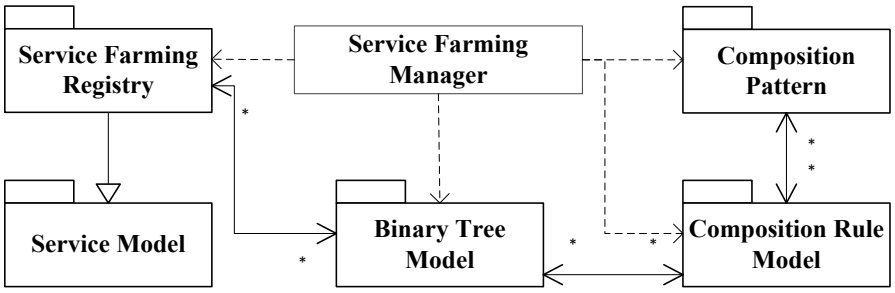

Figure 6.2 General Implementation of Service Framing Manager

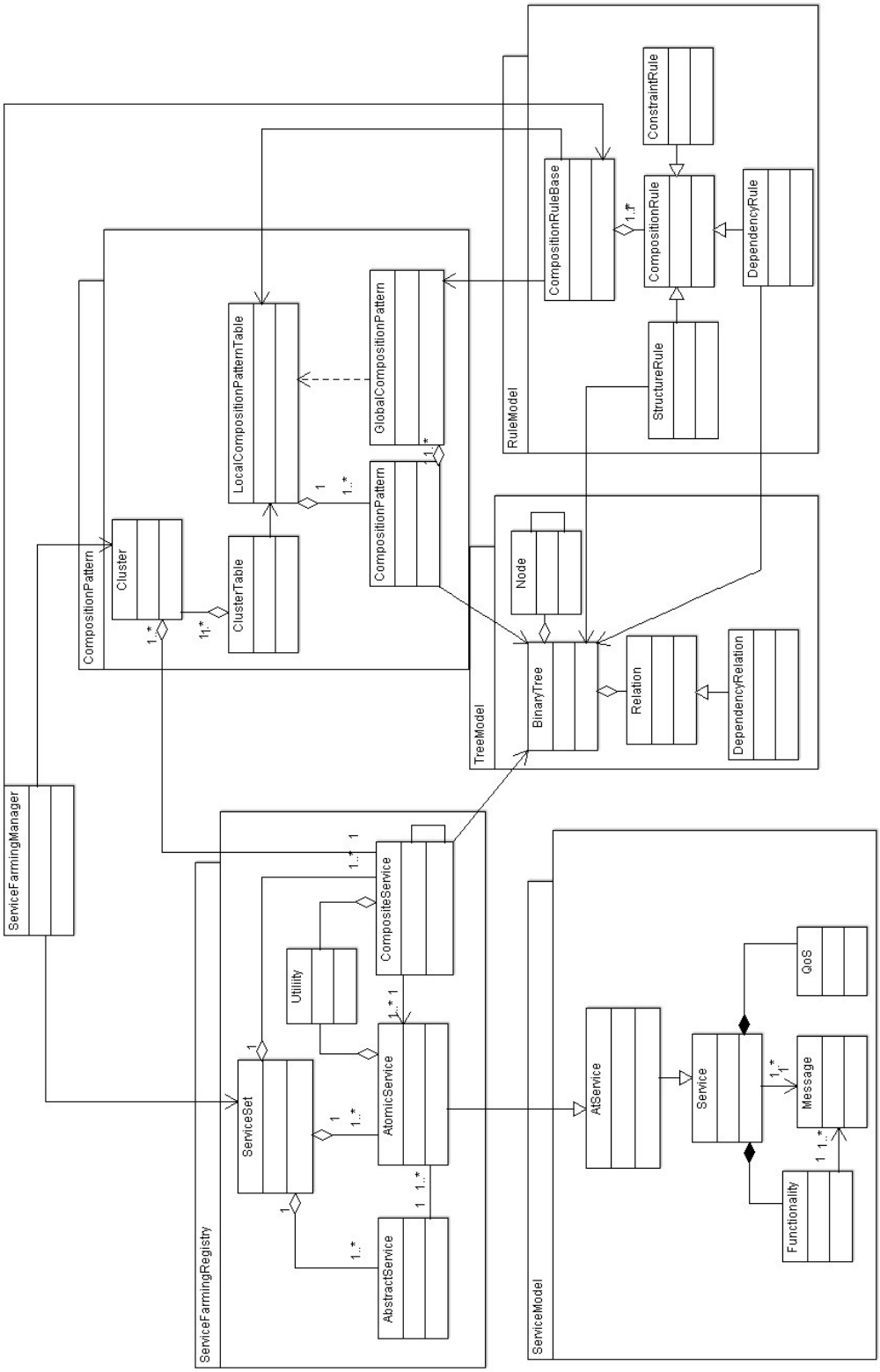

Figure 6.3 Class Diagram of Service Farming Manager

6.2 Experiment Results

To illustrate the resilient SOA framework and the implemented modules, we have manually created several capability instances based on our train crash crisis scenario (see Table 6.1) and associated Web services with capability instances in the C-WSDL Web service registry. Based on this scenario, the business-centric requirements are roughly illustrated in Table 6.2.

Table 6.1 Web Service Capability Instances

ID	Capability Objective Layer	Capability Profile Layer		Inter-capability Composition Layer
		capability name	attributes	
cap_1	{<response to, crisis>}	<alert, public>	{(AvailableRegion, France)}	
$cap_{2.1}$	{<manage, crisis>}	<evacuate, population>	{(AvailablePlace, France)}	
$cap_{2.2}$	{<manage, crisis>}	<evacuate, population>	{(AvailablePlace, Marseille)}	
cap_3	{<manage, crisis>, < rescue, people>}	<transport, victim>	{(MaxTransportNumber, 100)}	{cap_3, cap_4, Support}
cap_4	{<manage, crisis>, <rescue, people>}	<assist, victim>	{(MaxAssistNumber, 300)}	{cap_3, cap_4, Support}
$cap_{5.1}$	{<manage, crisis>}	<extinguish, fire>	{(AvailableFiremen, 40)}	{$cap_{5.1}$, $cap_{5.2}$, Cooperation} {$cap_{5.1}$, cap_7, Support}
$cap_{5.2}$	{<manage, emergence>}	<put out, fire>	{(AvailableFiremen, 70)}	{$cap_{5.1}$, $cap_{5.2}$, Cooperation} {$cap_{5.2}$, cap_7, Support}
cap_6	{<manage, crisis>}	<clear, site>	{(AvailableDate, 24/7)}	
cap_7	{<manage, accident>}	<recover, electricity>		{$cap_{5.1}$, cap_7, Support} {$cap_{5.2}$, cap_7, Support}
cap_8	{<manage, crisis>, <maintain, facility>}	<repair, railway>		

The prototype is successfully used to illustrate the train crash crisis scenario by specifying business-centric requirements and capabilities, and gradually generating the set of composition rules. The composition manager thus constructs composite services to satisfy the initialized business-centric requirements.

Focusing our ad-hoc composition approach, we performed a series of experiments on a Dell Laptop with an Intel Core 2 Quad processor at 2.53 GHz, 4 GB RAM, running Microsoft Windows 7. We also integrated

in our prototype the "Weka" Data Mining Package to perform *k-means* cluster algorithm. Our experimentations consist of sets of different numbers of abstract services (denoted as N_a), and different numbers of atomic services (denoted as N_t). For each abstract service, we manually create a number of atomic services by defining their QoS profiles with four QoS attributes (i.e., response time, price, availability, and reliability). The performance of Service Farming is evaluated based on the motivation scenario in Chapter 1 .

Table 6.2 Business-centric Requirement

FR.Objective	**obj$_1$**: *Manage* train crisis
FR.Action	**i$_1$:** Public must *be alerted*. **i$_2$:** Population must *be evacuated*. **i$_3$:** Victims must *be transported*. **i$_4$:** Victims must *be assisted*. **i$_5$:** Fire must be extinguished. **i$_6$:** Electricity must *be recovered*. **i$_7$:** Crisis site must *be cleared*. **i$_8$:** Railways must *be repaired*.
NF	**nf$_1$:** It is obligatory that at least 10 firemen *extinguish* fire. **nf$_2$:** It is necessary that total response time *is* less than 4h. **nf$_3$:** It is necessary that the cost *of* assisting victims *is* less than 2000€. **nf$_4$:** It is obligatory that victims *are transported before* victims *are assisted*. **nf$_5$:** It is obligatory that the electricity *is recovered after* the fire *is extinguished*.
Contextual Information	ctt$_1$: Crisis place *is* Pairs. ctt$_2$: Crisis date *is* 2013/03/01.

In order to provide a direct presentation of how Service Farming works to compose Web service through iterative cycles, we illustrate, in Table 6.3, a running example of Service Farming when given different values regarding different parameters.

For each 10 Service Farming cycles, we present the values of the current cycle number, the execution time, the initialized structure rule in the current cycle, the new generated structure rule in current cycle, and the maximum utility of all composite services found by Service Farming. From Table 6.3, we can see the evolution of these values, such as the structure rules used to guide the composition for each cycle is gradually enriched and the maximum utility of composite services found by Service Farming increases through iterative cycles.

Table 6.3 Service Farmimng Running Example

l_n	Execution Time (ms)	Initialized r_s Number	Generated r_s Number	Max_Utility
1	0	1	0	0.6043252
10	40	35	3	0.6519924
20	115	87	5	0.6652112
30	228	133	4	0.6817733
40	228	178	3	0.7103989
50	379	223	4	0.7226891
60	470	265	4	0.7474910
70	681	308	2	0.7515423
80	870	335	3	0.7609771
90	1020	365	2	0.7609771
100	1220	394	3	0.7609771
Abstract Service Number = 6, Atomic Service Number = 50, Constraint Rule Number =4, Dependency Rule Number = 2, k = 5, $T_{threshold}$ = 5, t = 50;				

In order to test the flexibility of Service Farming approach to construct composite services with or without predefined composition plans. We defined three different sub-cases by initializing different number of composition rules derived from user's requirements, which are shown in Table 6.4. We set N_a = 6, N_t = 30, k = 5, t = 50, T = 5 and carry on experimentations on the three cases.

Table 6.4 The Service Farming Performance based on Different Cases

Case No.	Initialized Structure Rule(s)	Cycle	Time(ms)
Case 1	{TransportVictim_AWS ⊜ AssistVictim_AWS}	85	1560
Case 2	{TransportVictim_AWS ⊜ AssistVictim_AWS, ExtinguishFire_AWS ⊜ RecoverElectricity_AWS}	60	508
Case 3	{(((AlertPublic_AWS⊜ (TransportVictim_AWS ⊜ AssistVictim_AWS)) ⏸ (ExtinguishFire_AWS ⊜ RecoverElectricity_AWS)) ⊜ ClearSite_AWS}	12	120

The right two columns of Table 6.4 show the cycles and execution time that service farming takes to find the optimal composition. Figure 6.4 presents the increase of $u_{optimal}$ value on three different cases. Given different composition rules, the Service Farming approach successfully finds optimal composite services following user requirements. Starting from the Case 1, the increase of composition rules completeness (number of abstract services that are included in composition rules) will significantly reduce the execution time for finding optimal solutions since the composition patterns between services that are indicated in composition rules are fixed. In the case 3 the composition rule with all abstract services shows a complete

predefined composition plan, the problem of service composition becomes only a problem of service selection based on an abstract composition plan. Trough this experiments, Service Farming shows its ability to flexibly construct the optimal composite services under the guidance of different initialized structure rules.

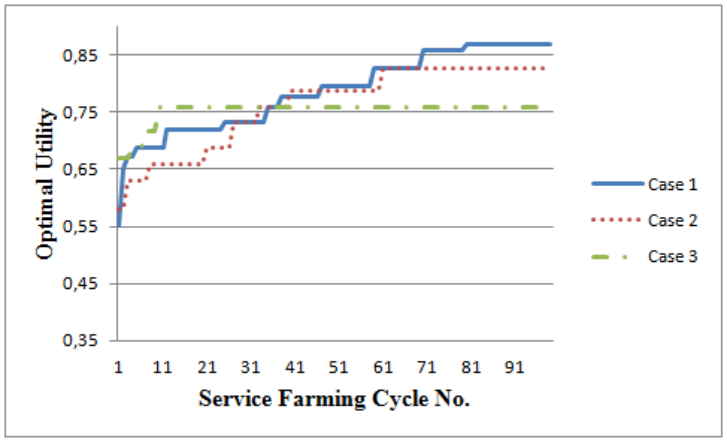

Figure 6.4 Optimal Utility Increase

Focusing on the case 1, we carry out experimentations to evaluate the influence of different parameters on the Service Farming performance. During our experiments, we have noticed that the repeated experiments on the same set of requirements and the same set of services produce different processes of optimal utility increase, but the execution time or number of cycles that Service Farming takes to find optimal services are in a limited range. This is because at the beginning, our approach composes services based on a random selection process, and this makes the composition results in the several initial cycles differ from one to another; as Service Farming is carrying on, the composition rules is gradually enriched, this random process gradually turns into a utility promoted process to help find the optimal solution.

Considering the parameter *t*, when given a large value of t, e.g., *t* = 200, Service Farming is able to find the best solution of all the composition possibilities, however the execution cost is quite long; on the contrary, if t is given a small value, e.g., *t* = 10, the execution time is reduced but the optimal composite service found is probably a sub optimal solution.

Figure 6.5 and Figure 6.6 respectively show the average execution time that each Service Farming cycle takes when given different values of the parameters k and T. These figures present that the latter cycles take more time than the previous cycles, as the number of composition rules in latter cycles is greater than the number of rules in previous cycles, and the composition approach takes more time in latter cycles on querying the increased set of composition rules. As for the parameter of cluster number k, the smaller the value of cluster number k is, the more time each cycle takes.

This is because the number of composite services constructed in each cycle is fixed, a smaller value of cluster number k will distribute more services in each cluster, and thus the number of composition rules generated in each cycle increases, and in a certain range, the smaller value of k will reduce the total cycles that service farming takes to find the optimal solution; on the other hand, in order to discover the representative constituent composite services that make a cluster possess higher average utility than other clusters, the value of k should not be too small, or else, even the composite services with relative low values of QoS are decomposed. Concerning the value of T, when given a small value, each cycle takes more time but the total execution time Service Farming takes to find optimal solution is reduced, as the composition rule is enriched more rapidly; on the other hand, when T is very small, the optimal solution is probably a local optimal composite service, as constituent composite services with lower appearance time are also enriched as composition rules. As a conclusion, the value of the parameters k and T should be adjusted through experiments according to the number of Web services to composed in order to ensure the good performance of Service Farming.

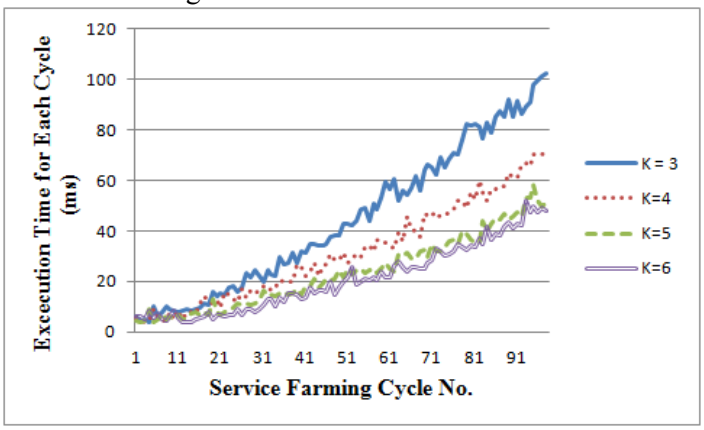

Figure 6.5 Average Time for Each Cycle by Varying k

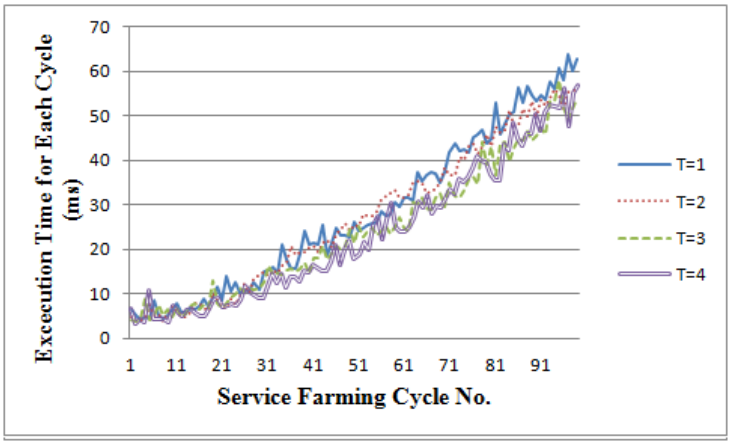

Figure 6.6 Average Time for Each Cycle by Varying T

Moreover, in order to test the scalability of Service Farming, we apply the Service Farming on different N_a and N_t without any initialized composition rules to test its performance by analyzing the execution time that Service Farming takes to find an optimal composition solution. The results are shown in Figure 6.7.

The results in Figure 6.7 are obtained by varying the value of N_a from 7 to 10, and the value of atomic service number for each abstract service, denoted as N_s, from 5 to 20. The execution time of our service farming increases along with the rise of N_a and N_s, while the increase trend is approximately exponential which are similar to most of composition approaches based on AI planning. By focusing on the execution cost with same N_s but different N_a, the higher the N_a is, the more time Service Farming takes, as N_a reflects the complexity of selecting a composition pattern between services, which is often more complex than selecting different services for a fix composition plan.

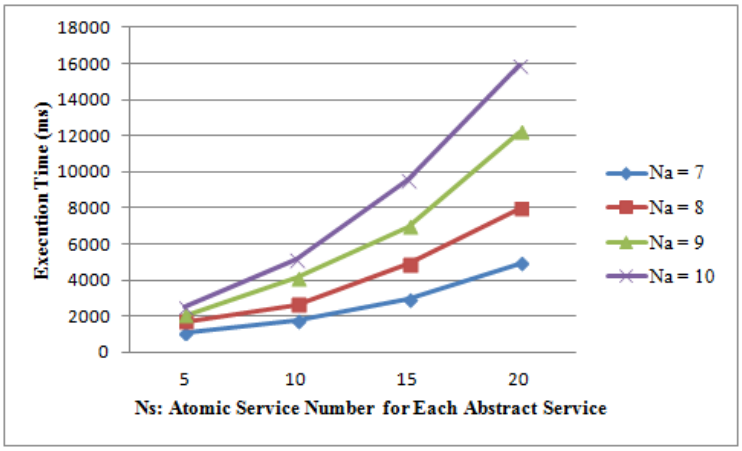

Figure 6.7 Scalability Evaluation

6.3 Conclusion

In this chapter, we have presented the technical architecture of the resilient service oriented framework and focused on the implementation of a prototype to test our proposed ad-hoc Web service composition approach. We at first specify user's requirements with SBVR terms following the business centric requirements. The prototype gradually transforms SBVR-based requirements into specific Web services and composition rules. At last, the Service Farming is applied to compose Web services together. Our experiments demonstrate the effectiveness of the proposed Web service composition approach in dynamic environments and the flexibility to compose Web services with or without predefined composition plans. We also examined

the performance of Service Farming through a series of experiments by varying the number of Web services to show its scalability. Finally, we discuss the influences of different parameters' values on the composition results and provide the strategy to set different parameters in order to quickly construct optimal composite services.

Chapter 7

Research Perspectives

7.1 Research Contributions

Nowadays many enterprises publish their applications functionalities on the Internet using Web services. This new generation of applications allows greater efficiency and availability for businesses. In many cases, a single service is not sufficient to respond to the user's request and thus Web services should be combined together to achieve a specific requirement that cannot be satisfied by a single Web service. Composing Web services while meeting multiple constraints (e.g., control flow constraints, QoS constraints, dependency constraints, etc.) from users' requirements has been proved as a NP-hard problem [MBKG09] and AI-complete problem [OHLK06].

Most automated dynamic Web composition approaches require a predefined abstract composition plan to compose Web services together by selecting specific Web service for each task in the composition plan and integrating them. However, in many cases of dynamic environments, a composition plan cannot be predefined in advance. An ad-hoc Web service composition is thus needed to build adaptable composite Web services in response to endogenous and exogenous changes that may occur in a dynamic environment without a predefined composition plan.

Throughout this thesis, we deal with the main research question as "*How to achieve a resilient service-oriented computing based on an ad-hoc Web service composition process driven by business requirements in order to build adaptable composite Web services without predefined composition plans? To the end, how to adapt composite Web services accordingly to endogenous and exogenous changes that may occur within and outside the dynamic composition environment at design time and runtime?"*. To answer this question, our contributions seek to achieve a resilient service oriented computing by extending current SOA with a set of models, representing requirements and changes that influence the Webs service compositions. In the resilient service-oriented architecture, models affect and are affected by each other. To this end, we particularly focus on the ad-hoc Web service composition model and the three-level requirement model. To ensure con-

nectivity between these three models and flexible Web service composition processes, we develop a two-phase requirement transformation process. We summarize the advantages of our contributions as follows.

1) The Three-level Requirement Model for Ad-hoc Web Service Composition

Business-centric requirements are modelled based on the SBVR, a formal business rule language with a structured natural language interface that allows business users to specify composition requirements in terms of business objectives, functional requirements and non-functional requirements.

Capability-focused requirements are modelled based on the Web service capability model in terms of Web service capability instances. We extend Web service descriptions with the capability model to describe what a Web service really does instead of its operations. The Web service capability model describes the Web services as a list of objectives to achieve, a list of attributes representing Web service abilities, and links to other capabilities by means of composition relationships. Web service capability instances are thus created by domain experts or service providers. Web service capability connects Web services with business objectives that Web services can achieve.

Rule-driven Web service requirements are modeled based on Web service operations and different types of rules (i.e., structure rules, constraint rules, and dependency rules) to represent multiple constraints influencing the Web service composition process. At this level, rule-driven Web service requirements are able to be directly used by the proposed ad-hoc Web service composition approach.

The value of this part of our contribution stems from providing a structured natural language for business users' to specify their composition requirements and consequently compose and execute Web services that satisfy the requirements; secondly, it allows either business users or technical users' to specify composition requirements based on different requirement models. The Web service composition process can thus be initialized from any level of the composition requirements by simply using our requirement transformation process.

2) The Two-level Requirement Transformation for Ad-hoc Web Service Composition: Capability Matching and Association Discovery

We develop the capability matching process and the association discovery process to transform business-centric requirements into capability-focused requirements, and transform capability-focused requirements to rule-driven Web service requirements. The requirement transformation pro-

cess is semantically based on keywords matching between different levels of requirement models.

The capability matching process focuses on deriving available Web service capability instances from users' business-centric requirements by matching business-centric requirements and existing Web service capability instances. The association discovery process then discovers all Web services that are associated with the capability instances from capability matching results to find appropriate Web services. In addition, the Association discovery process creates composition rules based on capability compositions as described by Web service capability instances.

Since existing requirement transformation approaches are only able to transform requirements defined from the same perspective in different language, the value of this contribution aims at providing an approach to transform requirements defined from different perspectives in different languages, as in our composition requirement models, business-centric requirements are defined from the business perspective, capability-focused requirements are defined from the Web service ability perspective , and rule-driven Web service requirements are defined from the Web service operation perspective.

3) The Ad-hoc Web Service Composition Approach: Service Farming

The ad-hoc Web service composition approach aims to construct Web services in dynamic environments without predefined composition plans by simultaneously selecting atomic services and inferring the composition patterns between the selected services in order to provide the best ways to compose services with regard to QoS.

The ad-hoc Web service composition approach consists of four stages (i.e., Service Planting, Service Growing, Service Harvesting, and Service Evaluating) that are executed iteratively through cycles to construct composite services at design time. In each cycle, Service Farming composes Web services by randomly selecting atomic services and inferring their composition patterns following composition rules; It ends when an optimal composition result is generated. At the runtime, when the composite Web service's execution is interrupted because of endogenous or exogenous changes, the ad-hoc composition approach conserves all information of the executed part of the composite service, generates composition rules based on the executed part and starts over from the design time to re-construct the unexecuted part of composite services.

The main value of this contribution is that the composition process can compose Web services in an ad-hoc way in dynamic environments without predefined composition plans. Instead of selecting specific Web

service after generating a general composition plan, the ad-hoc composition is able to ensure that Web services are composed in the best way with regard to QoS and user's satisfaction is maximized.

In our work we present an end-to-end solution on how to compose Web services driven by structured natural requirement in dynamic environment without predefined composition plans: users firstly specify the composition requirements flowing our requirement model for ad-hoc Web service composition; and then the requirement transformation is applied to transform requirements in business-level to requirements in technical-level which can be directly used by our ad-hoc Web service composition; at last Service Farming is used to construct optimal composite Web services to satisfy users' requirements. We implement our end-to-end Web service composition approach and develop a prototype to validate it. The experimental results introduced in Chapter 6 present the promising results.

7.2 Future Research Trends

The foremost aim of my research work is to establish foundations for achieving resilient service oriented computing to develop resilient SOA-based applications in dynamic environments. Nevertheless, the following research topics can be studied in the future to enhance the resilient SOA framework:

1. Extending the resilient service oriented architecture with bi-directional transformation between exogenous and endogenous models and the ad-hoc Web service composition approach. In our work, we only provide two situations, i.e., service substitution and composition replanning, to ensure composite services are successfully respond to changes in dynamic environments. However, when dynamic environment changes frequently, the two measures may be applied many times in one composition task, and which would greatly reduces the efficiency of Web service composition process. In the future work, we will investigate how to interleave the Web service composition and Web service execution process by dividing one big composition task into several small composition tasks; and thus for each small composition task, we respectively construct a sub composition solution, and then send it for execution while generating a sub composition solution for the next composition task. In addition, for different change model, we will provide more mechanisms to adapt composition results to changes.

2. Extending the business-centric requirement model based on the SBVR to provide users with natural means to specify other kinds of constraints in the Web service composition process, and accordingly provide transformation solutions. Since in our work we only apply a subset of

SBVR to represent users' business-centric requirements on control flow, QoS and dependency constraints, we can extend the business-centric requirement model by introducing other SBVR patterns to take into account other kinds of constraints, such as resource constraints to describe the consumed resources and take into account volatile resource environments when composing Web services.

3. Extending Web service capability model to provide richer capability objective and profile description. Although we extend current framecase based Web service capability model by using a pair of noun and action verb to describe the capability objectives and names, the expressiveness of this capability model is still limited to capture business objectives in richer context. We will further extend our proposed capability model to define complex objectives and profiles so as to connect Web services with richer business contexts.

4. Introducing new categories of composition rules to take into account multiple constraints. Since the ad-hoc Web service composition is rule based and directly driven by composition rules that are transformed from business-centric requirements. The set of categories of composition rules is easy to be enriched by structuring variables that may change (e.g., resources, mediation, etc.) and influencing the composition process. To this end, the composition approach should be updated since our approach is able to take into account of multiple constraints during the composition process, e.g., to introduce mediation rules to guarantee atomic services and exchange their input and output messages.

5. Carrying more experiments and benchmarks to study the characteristics of the Service Farming. The experiment results in our work are generated based on a preliminary motivation scenario of a crisis management, and future work should also carry more experiments in different scenarios and different set of Web services to compare their performance with other Web service composition approaches.

6. Verifying whether the set of capability instances from capability matching process can entirely satisfy or not the business objectives defined in business-centric requirements. In our work, we assume that after capability matching, the set of derived capability instances should be able to entirely satisfy the business objectives defined in business-centric requirements. As a future work, we will investigate the Intuitionist Linear Logic to proof whether a set of capabilities can entirely satisfy or not the business-centric requirements.

Bibliography

[AAWG05] W. M. Van der Aalst, M. Weske, and D. Grünbauer, "Case Handling: A New Paradigm for Business Process Support," *Data & Knowledge Engineering*, vol. 53, no. 2, pp. 129–162, 2005.

[ACMP07] D. Ardagna, M. Comuzzi, E. Mussi, B. Pernici, and P. Plebani, "PAWS: A Framework for Executing Adaptive Web-Service Processes," *IEEE Software*, vol. 24, no. 6, pp. 39–46, 2007.

[ADHW05] W. M. P. van der Aalst, M. Dumas, A. H. M. ter Hofstede, and P. Wohed, *Pattern Based Analysis of BPML (and WSCI)*. Brisbane: Queensland University of Technology, 2005.

[AFBA11] H. Afreena and I. S. Bajwab, "Generating Uml Class Models from Sbvr Software Requirements Specifications," in *Proceedings of the 23rd Benelux Conference on Artificial Intelligence*, 2011, pp. 23–32.

[AFMN05] R. Akkiraju, J. Farrell, J. Miller, and M. Nagarajan, *Web Service Semantics - WSDL-S*. University of Georgia Research Foundation, Inc., 2005.

[AGHS03] S. Agarwal, S. Handschuh, and S. Staab, "Surfing the Service Web," in *Proceedings of the 2nd International Semantic Web Conference*, 2003, vol. 2870, pp. 211–226.

[ALHE11] D. Allemang and J. A. Hendler, *Semantic Web for the Working Ontologist Effective Modeling in Rdfs and Owl*. Waltham, MA: Morgan Kaufmann/Elsevier, 2011.

[ALPA10] A. Albreshne and J. Pasquier, "Semantic-Based Semi-Automatic Web Service Composition," *Computer Department, Switzerland*, 2010.

[ALRI09] M. Alrifai and T. Risse, "Combining Global Optimization with Local Selection for Efficient Qos-Aware Service Composition," in *Proceedings of the 18th International Conference on World Wide Web*, 2009, pp. 881–890.

[APRT06] M. Aiello, C. Platzer, F. Rosenberg, H. Tran, M. Vasko, and S. Dustdar, "Web Service Indexing for Efficient Retrieval and Composition," in *Proceedings of the 8th IEEE International Conference on and Enterprise Computing, E-Commerce, and E-Services*, 2006, pp. 63–63.

[ARPE06] D. Ardagna and B. Pernici, "Global and Local QoS Guarantee in Web Service Selection," in *Business Process Management Workshops*, C. J. Bussler and A. Haller, Eds. Springer Berlin Heidelberg, 2006, pp. 32–46.

[ATZI09] D. Athanasopoulos, A. Zarras, and V. Issarny, "Service Substitution Revisited," in *Proceedings of the 2009 IEEE/ACM International Conference on Automated Software Engineering*, Washington, DC, USA, 2009, pp. 555–559.

[AYWO09] E. G. Aydal and J. Woodcock, "Automation of Model-Based Testing Through Model Transformations," in *Proceedings of the 2009 Testing: Academic and Industrial Conference - Practice and Research Techniques*, Washington, DC, USA, 2009, pp. 63–71.

[BABI12] P. Bartalos and M. Bieliková, "Automatic Dynamic Web Service Composition: A Survey and Problem Formalization," *Computing and Informatics*, vol. 30, no. 4, pp. 793–827, 2012.

[BCDD08] G. Baryannis, M. Carro, O. Danylevych, and S. Dustdar, *Overview of the State of the Art in Composition and Coordination of Services*. S-CUBE Consortium, 2008.

[BCGH05] D. Berardi, D. Calvanese, G. De Giacomo, R. Hull, and M. Mecella, "Automatic Composition of Transition-Based Semantic Web Services with Messaging," in *Proceedings of the 31st International Conference on Very Large Data Bases*, 2005, pp. 613–624.

[BCGL03] D. Berardi, D. Calvanese, G. D. Giacomo, M. Lenzerini, and M. Mecella, "Automatic Composition of E-services That Export Their Behavior," in *Proceedings of the 11th International Conference on Service Oriented Computing*, 2003, pp. 43–58.

[BCVM06] A. Bosca, F. Corno, G. Valetto, and R. Maglione, "On-the-Fly Construction of Web Services Compositions from Natural Language Requests," *Journal of Software*, vol. 1, no. 1, pp. 40–50, 2006.

[BEAS12] V. Beltran, K. Arabshian, and H. Schulzrinne, "Ontology-Based User-Defined Rules and Context-Aware Service Composition System," in *Proceedings of the 8th International Conference on the Semantic Web*, Berlin, Heidelberg, 2012, pp. 139–155.

[BEBG05] S. Beydeda, M. Book, and V. Gruhn, *Model-Driven Software Development*. Springer, 2005.

[BESD03] B. Benatallah, Q. Z. Sheng, and M. Dumas, "The Self-Serv Environment for Web Services Composition," *Internet Computing, IEEE*, vol. 7, no. 1, pp. 40–48, 2003.

[BHDZ12] S. Bhiri, W. Derguech, and M. Zaremba, "Web Service Capability Meta Model," in *Proceedings of the 8th International Conference on Web Information Systems and Technologies*, 2012.

[BKPP09] P. Bertoli, R. Kazhamiakin, M. Paolucci, M. Pistore, H. Raik, and M. Wagner, "Control Flow Requirements for Automated Service Composition," in *Proceedings of the 7th International Conference on Web Services*, 2009, pp. 17–24.

[BOZH09] Y. Bo and Q. Zheng, "A method of semantic web service composition based PDDL," in *Proceedings of the 2nd IEEE International Conference on Service-Oriented Computing and Applications*, 2009, pp. 1–4.

[BRDS06] R. I. Brafman, C. Domshlak, and S. E. Shimony, "On Graphical Modeling of Preference and Importance," *Journal of Artificial Intelligence Research*, vol. 25, no. 1, pp. 389–424, Mar. 2006.

[BRDU10] B. Bruegge and A. H. Dutoit, *Object-Oriented Software Engineering: Using Uml, Patterns, and Java*. Boston: Prentice Hall, 2010.

[BRPB08] I. Brandic, S. Pllana, and S. Benkner, "Specification, Planning, and Execution of Qos-Aware Grid Workflows Within the Amadeus Environment," *Concurrency and Computation: Practice and Experience*, vol. 20, no. 4, pp. 331–345, Mar. 2008.

[BRWI09] G. Briscoe and P. De Wilde, "Computing of Applied Digital Ecosystems," in *Proceedings of the International ACM Conference on Management of Emergent Digital EcoSystems*, New York, NY, USA, 2009, pp. 28–35.

[BSRH07] R. Berbner, M. Spahn, N. Repp, O. Heckmann, and R. Steinmetz, "Dynamic Replanning of Web Service Workflows," in *Proceedings of the 2007 Inaugural IEEE International Conference on Digital Ecosystems and Technologies*, 2007, pp. 211 –216.

[CAMS06] V. D. Castro, E. Marcos, and M. L. Sanz, "A Model Driven Method for Service Composition Modelling: A Case Study," *International Journal of Web Engineering and Technology*, vol. 2, no. 4, pp. 335–353, Jul. 2006.

[CASA12] D. Calvanese and A. Santoso, "Best Service Synthesis in the Weighted Roman Model," in *Proceedings of the 4th Central-European Workshop on Services and their Composition*, 2012, vol. 847, pp. 42–49.

[CASH03] J. Cardoso and A. Sheth, "Semantic E-Workflow Composition," *Journal of Intelligent Information Systems*, vol. 21, no. 3, pp. 191–225, 2003.

[CASS01] F. Casati, M. Sayal, and M.-C. Shan, "Developing E-Services for Composing E-Services," in *Advanced Information Systems Engineering*, Springer Berlin Heidelberg, 2001, pp. 171–186.

[CAST03] M. Carman, L. Serafini, and P. Traverso, "Web Service Composition as Planning," in *Proceedings of the 23rd International Conference on Automated Planning and Scheduling*, 2003, pp. 1636–1642.

[CCGP09] V. Cardellini, E. Casalicchio, V. Grassi, F. Lo Presti, and R. Mirandola, "Qos-Driven Runtime Adaptation of Service Oriented Architectures," in *Proceedings of the 7th Joint Meeting of the European Software Engineering Conference and the Acm Sigsoft Symposium on the Foundations of Software Engineering*, New York, NY, USA, 2009, pp. 131–140.

[CCMN04] G. Chafle, S. Chandra, V. Mann, and M. G. Nanda, "Decentralized Orchestration of Composite Web Services," in *Proceedings of the Alternate Track on Web Services at the 13th International World Wide Web Conference*, 2004, pp. 134–143.

[CDKN02] F. Curbera, M. Duftler, R. Khalaf, W. Nagy, N. Mukhi, and S. Weerawarana, "Unraveling the Web Services Web: An Introduction to Soap, Wsdl, and Uddi," *IEEE Internet Computing*, vol. 6, no. 2, pp. 86–93, 2002.

[CHBB07] K. S. M. Chan, J. Bishop, and L. Baresi, "Survey and Comparison of Planning Techniques for Web Services Composition," *Se-*

mantic Web Services and Web Process Composition, pp. 43–54, 2007.

[CHER13] C. Cherifi, "Similarity Network for Semantic Web Services Substitution," in *Proceedings of the 24th International Conference on Information Technologies*, 2013.

[CHMT10] P. Châtel, J. Malenfant, and I. Truck, "QoS-based Late-Binding of Service Invocations in Adaptive Business Processes," in *Proceedings of the 2010 IEEE International Conference on Web Services*, 2010, pp. 227–234.

[CHSA06] Y. Charif and N. Sabouret, "An Overview of Semantic Web Services Composition Approaches," *Electronic Notes in Theoretical Computer Science*, vol. 146, no. 1, pp. 33–41, 2006.

[CHU06] P. P. Chu, "Finite State Machine: Principle and Practice," in *Rtl Hardware Design Using Vhdl: Coding for Efficiency, Portability, and Scalability*, John Wiley & Sons, Inc., 2006, pp. 313–371.

[CIJK00] F. Casati, S. Ilnicki, L. J. Jin, V. Krishnamoorthy, and M. C. Shan, "Adaptive and Dynamic Service Composition in eFlow," in *Advanced Information Systems Engineering*, 2000, pp. 13–31.

[CPEV05] G. Canfora, M. Di Penta, R. Esposito, and M. L. Villani, "Qos-Aware Replanning of Composite Web Services," in *Proceedings of the 2005 IEEE International Conference on Web Services*, 2005, pp. 121–129.

[CSGT03] L. Chen et al., "Towards a Knowledge-Based Approach to Semantic Service Composition," in *Proceedinds of the 2nd International Semantic Web Conference*, 2003, pp. 319–334.

[DARE05] F. Darema, "Dynamic Data Driven Applications Systems: New Capabilities for Application Simulations and Measurements," in *Proceedings of the 5th International Conference on Computational Science - Volume Part II*, Berlin, Heidelberg, 2005, pp. 610–615.

[DARE09] F. Darema, "Characterizing Dynamic Data Driven Applications Systems (DDDAS) in Terms of a Computational Model," in *Computational Science–ICCS 2009*, Springer, 2009, pp. 447–448.

[DCHS04] J. Domingue, L. Cabral, F. Hakimpour, D. Sell, and E. Motta, "IRS-III: A Platform and Infrastructure for Creating WSMO-based Semantic Web Services," in *Proceedings of the Workshop on WSMO Implementations*, 2004, pp. 29–30.

[DEVL06] E. S. Devlin, *Crisis Management Planning and Execution*. CRC Press, 2006.

[DIPW08] M. DiBernardo, R. Pottinger, and M. Wilkinson, "Semi-Automatic Web Service Composition for the Life Sciences Using the Biomoby Semantic Web Framework," *Journal of Biomedical Informatics*, vol. 41, no. 5, pp. 837–847, Oct. 2008.

[DROM03] R. G. Dromey, "From Requirements to Design: Formalizing the Key Steps," in *Proceedings of the First International Conference on Software Engineering and Formal Methods*, 2003, pp. 2–11.

[DTKB03] W. M. P. Van Der Aalst, A. H. M. Ter Hofstede, B. Kiepuszewski, and A. P. Barros, "Workflow Patterns," *Distributed and Parallel Databases*, vol. 14, no. 1, pp. 5–51, Jul. 2003.

[DUHO01] M. Dumas and A. H. M. ter Hofstede, "UML Activity Diagrams as a Workflow Specification Language," in *Proceedings of the*

4th International Conference on the Unified Modeling Language, Toronto, Canada, 2001, pp. 76–90.

[DUSC05] S. Dustdar and W. Schreiner, "A Survey on Web Services Composition," *International Journal of Web and Grid Services*, vol. 1, no. 1, pp. 1–30, 2005.

[EHSA12] Ehsan Rasoulinezhad, "Evaluation of Countries Solution Against the U.s Economics Crisis Through a Multiple Attribute Decision Model (madm)," *Journal of Economics and International Finance*, vol. 4, no. 1, pp. 9–17, 2012.

[ERL08] T. Erl, *SOA: Principles of Service Design*. 2008.

[FAYA13] A. Fayoumi and L. Yang, "SBVR: Knowledge Definition, Vocabulary Management, and Rules Integrations," *International Journal of E-Business Development*, vol. 2, no. 2, pp. 70–76, Jan. 2013.

[FEDO05] C. Feier and J. Domingue, *WSMO Primer*. 2005.

[FJWP10] X.-Q. Fan, C.-J. Jiang, J.-L. Wang, and S.-C. Pang, "Random-QoS-Aware Reliable Web Service Composition," *Journal of Software*, vol. 20, no. 3, pp. 546–556, Mar. 2010.

[FKMS07] G. Fliedl et al., "Deriving Static and Dynamic Concepts from Software Requirements Using Sophisticated Tagging," *Data Knowl. Eng.*, vol. 61, no. 3, pp. 433–448, Jun. 2007.

[FOLO03] M. Fox and D. Long, "PDDL2.1: An extension to PDDL for expressing temporal planning domains," *Journal of Artificial Intelligence Research*, vol. 20, p. 2003, 2003.

[FOST04] H. Foster, "Behaviour Analysis and Verification of Web Service Compositions," Imperial College London, 2004.

[GBNA08] Y. Gamha, N. Bennacer, G. V. Naquet, B. Ayeb, and L. B. Romdhane, "A Framework for the Semantic Composition of Web Services Handling User Constraints," in *Proceedings of the 2008 IEEE International Conference on Web Services*, 2008, pp. 228–237.

[GCBG10] D. Grigori, J. C. Corrales, M. Bouzeghoub, and A. Gater, "Ranking BPEL Processes for Service Discovery," *IEEE Transactions on Services Computing*, vol. 3, no. 3, pp. 178–192, 2010.

[GERH99] A. T. Gerhard Wickler, "Capability Representations for Brokering: A Survey," *Knowledge Engineering Review*, pp. 1–70, 1999.

[GMMZ05] P. Giorgini, F. Massacci, J. Mylopoulos, and N. Zannone, "ST-tool: a CASE tool for security requirements engineering," in *Proceedings of the 13th IEEE International Conference on Requirements Engineering*, 2005, pp. 451–452.

[GREM04] P. Grünbacher, A. Egyed, and N. Medvidovic, "Reconciling Software Requirements and Architectures with Intermediate Models," *Software and Systems Modeling*, vol. 3, no. 3, pp. 235–253, 2004.

[GRJA05] R. Grønmo and M. C. Jaeger, "Model-Driven Semantic Web Service Composition," in *Proceedings of the 27th International Conference on Software Engineering*, 2005, p. 8–pp.

[GRLP10] Y. Gripay, F. Laforest, and J.-M. Petit, "A Simple (yet Powerful) Algebra for Pervasive Environments," in *Proceedings of the 13th International Conference on Extending Database Technology*, 2010, pp. 359–370.

[GRSU05] A. Gregoriades and A. Sutcliffe, "Scenario-Based Assessment of Nonfunctional Requirements," *IEEE Transactions on Software Engineering*, vol. 31, no. 5, pp. 392–409, May. 2005.

[GTSS11] G. Grossmann, R. Thiagarajan, M. Schrefl, and M. Stumptner, "Conceptual Modeling Approaches for Dynamic Web Service Composition," in *The Evolution of Conceptual Modeling*, R. Kaschek and L. Delcambre, Eds. Springer Berlin Heidelberg, 2011, pp. 180–204.

[HAAM08] P. Harshavardhan, J. Akilandeswari, and M. Manjari, "Dynamic Web Service Composition Problems and Solution -a Survey," *MES Journal of Technology and Management*, pp. 1–5, 2008.

[HAKP12] J. Han, M. Kamber, and J. Pei, *Data Mining: Concepts and Techniques*. Burlington, MA: Elsevier, 2012.

[HALP06] T. Halpin, "Business Rule Modality," in *Proceedings of the 11th Workshop on Exploring Modeling Methods in Systems Analysis and Design*, 2006.

[HDMC04] F. Hakimpour, J. Domingue, E. Motta, L. Cabral, and Y. Lei, "Integration of OWL-S into IRS-III," in *Proceedinds of the First AKT Workshop on Semantic Web Services*, 2004, pp. 1–4.

[HEND05] S. Hendryx, *Model-Driven Architecture and the Semantics of Business Vocabulary and Business Rules*. 2005.

[HILM04] M. W. Hilmar Schuschel, "Triggering Replanning in an Integrated Workflow Planning and Enactment System.," in *Proceedings of the 8th East European Conference*, 2004, pp. 322–335.

[HLXZ11] X. Han, Y. Liu, B. Xu, and G. Zhang, "A Survey on Qos-Aware Dynamic Web Service Selection," in *Proceedinds of the 7th International Conference on Wireless Communications, Networking and Mobile Computing*, 2011, pp. 1–5.

[HUSI05] M. N. Huhns and M. P. Singh, "Service-Oriented Computing: Key Concepts and Principles," *IEEE Internet Computing*, vol. 9, no. 1, pp. 75–81, 2005.

[IBMO09] N. Ibrahim and F. L. Mouel, "A Survey on Service Composition Middleware in Pervasive Environments," *International Journal of Computer Science Issues*, vol. 7, no. 4, pp. 1–12, 2009.

[JACO99] W. G. Jacoby, "Levels of Measurement and Political Research: An Optimistic View," *American Journal of Political Science*, vol. 43, no. 1, pp. 271–301, 1999.

[JAMG05] M. C. Jaeger, G. Mühl, and S. Golze, "Qos-Aware Composition of Web Services: An Evaluation of Selection Algorithms," in *Proceedings of the 2005 Confederated International Conference on on the Move to Meaningful Internet Systems*, R. Meersman and Z. Tari, Eds. Springer Berlin Heidelberg, 2005, pp. 646–661.

[JARM04] M. C. Jaeger, G. Rojec-Goldmann, and G. Muhl, "Qos Aggregation for Web Service Composition Using Workflow Patterns," in *Proceedings of the Eighth IEEE International Enterprise Distributed Object Computing Conference*, 2004, pp. 149–159.

[JASA12] N. Jazuli Bin Kamarudin, N. F. M. Sani, and R. Atan, "Transformation from Requirement into Behavior Design: A Review," in *Proceedings of the 2012 International Conference on Information Retrieval & Knowledge Management*, 2012, pp. 153–157.

[JIAN10] J.-Z. L. Jian-Qiang HU, "A Multi-QoS Based Local Optimal Model of Service Selection," *Chinese Journal of Computers*, vol. 33, no. 3, pp. 526–534, 2010.

[JIEB07] L. Jiang and A. Eberlein, "A Tool For Requirements Engineering Process Development," in *Proceedings of the 31st Annual International Computer Software and Applications Conference*, Beijing, China, 2007, pp. 319–325.

[KABH04] M. Keen et al., *Patterns: Implementing an SOA Using an Enterprise Service Bus*. IBM, International Technical Support Organization, 2004.

[KAKG07] R. Karunamurthy, F. Khendek, and R. H. Glitho, "A Business Model for Dynamic Composition of Telecommunication Web Services," *IEEE Communications Magazine*, vol. 45, no. 7, pp. 36–43, 2007.

[KAKS07] S. Kalasapur, M. Kumar, and B. Shirazi, "Dynamic Service Composition in Pervasive Computing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 18, no. 7, pp. 907–918, 2007.

[KASE09] E. Karakoc and P. Senkul, "Composing Semantic Web Services Under Constraints," *Expert Systems with Applications*, vol. 36, no. 8, pp. 11021–11029, Oct. 2009.

[KATT12] A. Kattepur, "Flexible Quality of Service Management of Web Services Orchestrations," Université Rennes 1, 2012.

[KBBG08] S. Kona, A. Bansal, M. B. Blake, and G. Gupta, "Generalized Semantics-Based Service Composition," in *Proceedings of the 2008 IEEE International Conference on Web Services*, 2008, pp. 219–227.

[KHAL03] R. Khalaf, "Service–Oriented Composition in BPEL4WS," in *Proceedings of the Twelfth International World Wide Web Conference*, Budapest, Hungary, 2003.

[KLGE05] M. Klusch and A. Gerber, "Semantic Web Service Composition Planning with Owls-Xplan," in *In Proceedings of the 1st International AAAI Fall Symposium on Agents and the Semantic Web*, 2005, pp. 55–62.

[KLPT04] U. Keller, R. Lara, A. Polleres, I. Toma, M. Kifer, and D. Fensel, *Wsmo Web Service Discovery*. 2004.

[KOCK09] N. Kock, Ed., *E-Collaboration: Concepts, Methodologies, Tools, and Applications*. IGI Global, 2009.

[KUXR09] C. Kun, J. Xu, and S. Reiff-Marganiec, "Markov-HTN Planning Approach to Enhance Flexibility of Automatic Web Service Composition," in *Proceedings of the 2009 IEEE International Conference on Web Services*, 2009, pp. 9–16.

[KVBF07] J. Kopecky, T. Vitvar, C. Bournez, and J. Farrell, "Sawsdl: Semantic Annotations for Wsdl and Xml Schema," *Internet Computing, IEEE*, vol. 11, no. 6, pp. 60–67, 2007.

[LALE05] G. Lakemeyer and H. J. Levesque, "Semantics for a Useful Fragment of the Situation Calculus," in *Proceedings of the 19th International Joint Conference on Artificial Intelligence*, San Francisco, CA, USA, 2005, pp. 490–496.

[LASG07] S. Lamparter, A. Ankolekar, R. Studer, and S. Grimm, "Preference-based Selection of Highly Configurable Web Services," in

Proceedings of the 16th International Conference on World Wide Web, New York, NY, USA, 2007, pp. 1013–1022.

[LEPV10] F. Lanubile, C. Ebert, R. Prikladnicki, and A. Vizcaino, "Collaboration Tools for Global Software Engineering," *IEEE Software*, vol. 27, no. 2, pp. 52–55, Mar. 2010.

[LÉSP08] F. Lécué, E. Silva, and L. F. Pires, "A Framework for Dynamic Web Services Composition," in *Emerging Web Services Technology, Volume II*, T. Gschwind and C. Pautasso, Eds. Birkhäuser Basel, 2008, pp. 59–75.

[LIBB13A] W. Li, Y. Badr, and F. Biennier, "Improving Web Service Composition with User Requirement Transformation and Capability Model," in *Proceedings of the On the Move to Meaningful Internet Systems: OTM 2013 Conferences*, 2013, pp. 300–307.

[LIBB13B] W. Li, Y. Badr, and F. Biennier, "Towards a Capability Model for Web Service Composition," in *Proceedings of the 2013 IEEE 20th International Conference on Web Services*, 2013, pp. 609–610.

[LIBB13C] W. Li, Y. Badr, and F. Biennier, "Towards Natural-like Requirement based Web Service Composition," in *Proceedings of the 25th International Conference on Software & Systems Engineering and their Applications*, 2013, pp. 9–15.

[LIBB13D] W. Li, Y. Badr, and F. Biennier, "Service Farming: An Ad-Hoc and Qos-Aware Web Service Composition Approach," in *Proceedings of the 28th Annual ACM Symposium on Applied Computing*, New York, NY, USA, 2013, pp. 750–756.

[LIN08] N. Lin, "Web Service Composition with User Preferences," in *Proceedings of the 5th European Semantic Web Conference*, 2008, pp. 629–643.

[LIZD10] X. Li, Q. Zhao, and Y. Dai, "A Semantic Web Service Composition Method Based on an Enhanced Planning Graph," in *Proceedings of the 2010 International Conference on E-Business and E-Government*, 2010, pp. 2288–2291.

[LKMC06] Z. Laliwala, R. Khosla, P. Majumdar, and S. Chaudhary, "Semantic and Rules Based Event-Driven Dynamic Web Services Composition for Automation of Business Processes," in *Proceedings of the 2006 IEEE Services Computing Workshops*, 2006, pp. 175–182.

[LSYF09] D. Liu, Z. Shao, C. Yu, and G. Fan, "A Heuristic QoS-Aware Service Selection Approach to Web Service Composition," in *Proceedings of the Eigth IEEE/ACIS International Conference on Computer and Information Science*, 2009, pp. 1184–1189.

[LYON91] J. Lyons, *Natural language and universal grammar*. Cambridge, U.K.: Cambridge University Press, 1991.

[MAMC03] D. J. Mandell and S. A. McIlraith, "Adapting Bpel4ws for the Semantic Web: The Bottom-up Approach to Web Service Interoperation," in *Proceedings of the 2nd International Semantic Web Conference*, D. Fensel, K. Sycara, and J. Mylopoulos, Eds. Springer Berlin Heidelberg, 2003, pp. 227–241.

[MBHL04] D. Martin, M. Burstein, J. Hobbs, and O. Lassila, *OWL-S: Semantic Markup for Web Services*. 2004.

[MBKG09] N. Ben Mabrouk, S. Beauche, E. Kuznetsova, N. Georgantas, and V. Issarny, "QoS-Aware Service Composition in Dynamic Ser-

vice Oriented Environments," in *Middleware 2009*, vol. 5896, J. M. Bacon and B. F. Cooper, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 123–142.

[MCDE02] D. McDermott, "Estimated-Regression Planning for Interactions with Web Services," in *Proceedings of the Fourth International Conference on Artificial Intelligence Planning Systems*, 2002, vol. 2, pp. 204–211.

[MCIL02] S. Mcilraith, "Adapting Golog for composition of semantic web Services," in *Proceedings of the 8th International Conference on Knowledge Representation and Reasoning*, 2002, pp. 482–493.

[MEBA10] G. Meditskos and N. Bassiliades, "Structural and Role-Oriented Web Service Discovery with Taxonomies in Owl-S," *IEEE Transactions on Knowledge and Data Engineering*, vol. 22, no. 2, pp. 278–290, Feb. 2010.

[MEBE03] B. Medjahed, A. Bouguettaya, and A. K. Elmagarmid, "Composing Web Services on the Semantic Web," *The VLDB Journal*, vol. 12, no. 4, pp. 333–351, Nov. 2003.

[MEHS00] J. Meng, S. Helal, and S. Su, "An Ad-Hoc Workflow System Architecture Based on Mobile Agents and Rule-Based Processing," in *Proceedings of the First International Conference on Parallel and Distributed Computing Techniques and Applications*, 2000.

[MEWE06] H. Meyer and M. Weske, "Automated Service Composition Using Heuristic Search," in *Business Process Management*, Springer, 2006, pp. 81–96.

[MICH96] L. Mich, "Nl-Oops: From Natural Language to Object Oriented Requirements Using the Natural Language Processing System Lolita," *Natural Language Engineering*, vol. 2, no. 2, pp. 161–187, Jun. 1996.

[MLMB06] C. MacKenzie, K. Laskey, F. McCabe, and P. Brown, *Reference Model for Service Oriented Architecture 1.0*. 2006.

[MMNJ07] Martin Gudgin, Marc Hadley, Noah Mendelsohn, and Jean-Jacques Moreau, *SOAP Version 1.2*. 2007.

[MMVL05] F. Moscato, N. Mazzocca, V. Vittorini, G. Di Lorenzo, P. Mosca, and M. Magaldi, "Workflow Pattern Analysis in Web Services Orchestration: The Bpel4ws Example," in *Proceedings of the First International Conference on High Performance Computing and Communications*, Berlin, Heidelberg, 2005, pp. 395–400.

[MORD08] O. Moser, F. Rosenberg, and S. Dustdar, "Non-intrusive Monitoring and Service Adaptation for WS-BPEL," in *Proceedings of the 17th International Conference on World Wide Web*, New York, NY, USA, 2008, pp. 815–824.

[MOZE11] R. Mohamad and F. Zeshan, "Semantic Web Service Composition Approaches_overview and Limitations," *Software Engineering and Computer Systems*, pp. 283–290, 2011.

[MPMB05] D. Martin et al., "Bringing Semantics to Web Services: The OWL-S Approach," in *Semantic Web Services and Web Process Composition*, J. Cardoso and A. Sheth, Eds. Springer Berlin Heidelberg, 2005, pp. 26–42.

[MSGO06] N. Maiden, N. Seyff, P. Grunbacher, O. Otojare, and K. Mitteregger, "Making Mobile Requirements Engineering Tools Usable and Useful," in *Proceedings of the 14th Ieee International Conference on Requirements Engineering*, 2006, pp. 29–38.

[MTYB04] Z. MAAMAR, S. TATA, K. YETONGNON, and D. BENSLIMANE, "A Goal-Based Approach to Engineering Capacity-Driven Web Services," *The Knowledge Engineering Review*, pp. 1–17, 2004.

[MUCH12] D. Mukhopadhyay and A. Chougule, "A Survey on Web Service Discovery Approaches," *Advances in Computer Science, Engineering & Applications*, pp. 1001–1012, 2012.

[NAMC02] S. Narayanan and S. A. McIlraith, "Simulation, Verification and Automated Composition of Web Services," in *Proceedings of the 11th International Conference on World Wide Web*, New York, NY, USA, 2002, pp. 77–88.

[NAQA08] F. Nawaz, K. Qadir, and H. F. Ahmad, "Semreg-Pro: A Semantic Based Registry for Proactive Web Service Discovery Using Publish-Subscribe Model," in *Proceedings of the Fourth International Conference on Semantics, Knowledge and Grid*, 2008, pp. 301–308.

[NEWB01] M. Newborn, *Automated theorem proving: theory and practice*. New York: Springer, 2001.

[NUEA00] B. Nuseibeh and S. Easterbrook, "Requirements Engineering: A Roadmap," in *Proceedings of the Conference on the Future of Software Engineering*, New York, NY, USA, 2000, pp. 35–46.

[OAHE03] P. Oaks, A. ter Hofstede, and D. Edmond, "Capabilities: Describing What Services Can Do," *Proceedings of the First International Conference on Service Oriented Computing*, pp. 1–16, 2003.

[OBJE04] Object Management Group, *Business Semantics of Business Rules Request For Proposal*. Object Management Group, 2004.

[OHLK06] S.-C. Oh, D. Lee, and S. R. Kumara, "A Comparative Illustration of AI Planning-Based Web Services Composition," *ACM SIGecom Exchanges*, vol. 5, no. 5, pp. 1–10, 2006.

[OSET02] J. O'Sullivan, D. Edmond, and A. Ter Hofstede, "What's in a Service?," *Distributed Parallel Databases*, vol. 12, no. 2–3, pp. 117–133, Sep. 2002.

[PAAB07] A. V. Paliwal, N. R. Adam, and C. Bornhovd, "Web Service Discovery: Adding Semantics through Service Request Expansion and Latent Semantic Indexing," in *Proceedings of the 2007 IEEE International Conference on Services Computing*, 2007, pp. 106–113.

[PAFL11] P. Papapanagiotou and J. D. Fleuriot, "A Theorem Proving Framework for the Formal Verification of Web Services Composition," in *Proceedings of the International Workshop on Automated Specification and Verification of Web Systems*, 2011, pp. 1–16.

[PAPA03] M. P. Papazoglou, "Service-Oriented Computing: Concepts, Characteristics and Directions," in *Proceedings of the Fourth International Conference on Web Information Systems Engineering*, 2003, pp. 3–12.

[PEER05] J. Peer, *Web Service Composition as AI Planning – a Survey*. Switerland: University of St. Gallen, 2005.

[PGPS11] Philip Bianco, Grace A. Lewis, Paulo Merson, and Soumya Simanta, *Architecting Service-Oriented Systems*. Carnegie Mellon University, 2011.

[PHKH12] R. Phalnikar and P. A. Khutade, "Survey of Qos Based Web Service Discovery," in *Proceedings of the 2012 World Congress on Information and Communication Technologies*, 2012, pp. 657 – 661.

[PKPS02] M. Paolucci, T. Kawamura, T. Payne, and K. Sycara, "Semantic Matching of Web Services Capabilities," *Proceedings of the First International Semantic Web Conference*, pp. 333–347, 2002.

[POFO02] S. Ponnekanti and A. Fox, "Sword: A Developer Toolkit for Web Service Composition," in *Proceedings of the 11th International WWW Conference*, 2002.

[PREI04] C. Preist, "A Conceptual Architecture for Semantic Web Services," in *Proceedings of the Third International Semantic Web Conference*, 2004, pp. 395–409.

[PUHK06] K. Pu, V. Hristidis, and N. Koudas, "Syntactic Rule Based Approach Toweb Service Composition," in *Proceedings of the 22nd International Conference on Data Engineering*, Washington, DC, USA, 2006, pp. 31–40.

[RAAN07] J. Ramírez and A. de Antonio, "Automated Planning and Replanning in an Intelligent Virtual Environments for Training," in *Knowledge-Based Intelligent Information and Engineering Systems*, B. Apolloni, R. J. Howlett, and L. Jain, Eds. Springer Berlin Heidelberg, 2007, pp. 765–772.

[RAIK12] H. Raik, "Service Composition in Dynamic Environments: From Theory to Practice," University of Trento, 2012.

[RAO04] J. Rao, "Semantic Web Service Composition Via Logic-Based Program Synthesis," Norwegian University of Science and Technology, 2004.

[RAPH08] A. Raj, T. V. Prabhakar, and S. Hendryx, "Transformation of Sbvr Business Design to UML Models," in *Proceedings of the 1st India Software Engineering Conference*, 2008, pp. 29–38.

[RASU05] J. Rao and X. Su, "A Survey of Automated Web Service Composition Methods," in *Proceedings of the First International Workshop on Semantic Web Services and Web Process Composition*, 2005, pp. 43–54.

[RBML06] D. Roman et al., "WWW: WSMO, WSML, and WSMX in a Nutshell," *Proceedings of the 1st Annual Asian Semantic Web Conference*, pp. 516–522, 2006.

[REF10] Oracle, *Deploying the BIG-IP System v10 with Oracle's BEA WebLogic*. 2010.

[REF11] Object Management Group, *Meta Object Facility (MOF) 2.0 Query/View/Transformation*. Object Management Group, 2011.

[REF13A] Microsfot, *What is BizTalk?* 2013.

[REF13B] OPAALS, "SBVR Visual Editor," *SourceForge*, 2013. [Online]. Available: http://sourceforge.net/projects/sbvrve/. [Accessed: 18-Nov-2013].

[REF95] Microsoft Corporation, *The Component Object Model Specification*. 1995.

[REYT07] S. Reiff-marganiec, H. Q. Yu, and M. Tilly, "Service selection based on non-functional properties," in *Proceedings of Non Functional Properties and Service Level Agreements in Service Oriented Computing Workshop*, 2007.

[RONA09] Ronald SCHMELZER, *Resilience: The Missing Word in the SOA Conversation*. 2009.

[RÖNN96] S. Rönn, "Invariants and Closures in the Theory of Rewrite Systems," *Formal Aspects of Computing*, vol. 8, no. 4, pp. 463–478, Jul. 1996.

[ROPD06] F. Rosenberg, C. Platzer, and S. Dustdar, "Bootstrapping Performance and Dependability Attributes of Web Services," in *Proceedings of the 2006 International Conference on Web Services*, 2006, pp. 205–212.

[SABH09] G. R. Santhanam, S. Basu, and V. Honavar, "Web Service Substitution Based on Preferences Over Non-Functional Attributes," in *Proceedings of the 2009 IEEE International Conference on Services Computing*, Washington, DC, USA, 2009, pp. 210–217.

[SAMI08] Samir Nasser, *Resilient Scatter-Gather ESB Messaging Design with Message-Driven Beans*. 2008.

[SAMK04] A. Salbrechter, H. Mayr, and C. Kop, "Mapping Pre-Designed Business Process Models to UML," in *Proceedings of the 2004 International Conference on Software Engineering and Applications*, 2004, pp. 400–405.

[SASO05] N. Samarasinghe and S. Somé, "Generating a Domain Model from a Use Case Model," in *Proceedings of the 14th International Conference on Intelligent and Adaptive Systems and Software Engineering*, 2005.

[SFKM11] Y. Syu, Y.-Y. FanJiang, J.-Y. Kuo, and S.-P. Ma, "Towards a Genetic Algorithm Approach to Automating Workflow Composition for Web Services with Transactional and QoS-Awareness," in *Proceeding of the 2011 2011 IEEE World Congress on Services*, 2011, pp. 295–302.

[SHAW90] M. Shaw, "Prospects for an Engineering Discipline of Software," *IEEE Software*, vol. 7, no. 6, pp. 15–24, Nov. 1990.

[SHCH11] Y. Shi and X. Chen, "A Survey on QoS-Aware Web Service Composition," in *Proceedings of the Third International Conference on Multimedia Information Networking and Security*, 2011, pp. 283–287.

[SHDM04] D. Sell, F. Hakimpour, J. Domingue, E. Motta, and R. C. S. Pacheco, "Interactive Composition of WSMO-Based Semantic Web Services in IRS-III," in *Proceedings of the First Akt Workshop on Semantic Web Services*, 2004.

[SIHP02] E. Sirin, J. Hendler, and B. Parsia, "Semi-automatic Composition of Web Services using Semantic Descriptions," in *Proceedings of the 1st International Workshop on Web Services: Modeling, Architecture and Infrastructure*, 2002, pp. 17–24.

[SING03] M. P. Singh, "Distributed Enactment of Multiagent Workflows: Temporal Logic for Web Service Composition," in *Proceedings of the Second International Joint Conference on Autonomous Agents and Multiagent Systems*, 2003, pp. 907–914.

[SISO09] Y. Singh and M. Sood, "Model Driven Architecture: A Perspective," in *Proceedings of the 2009 IEEE International Advance Computing Conference*, 2009, pp. 1644–1652.

[SKGS04] D. Skogan, R. Grønmo, and I. Solheim, "Web Service Composition in UML," in *Proceedings of the Eighth Ieee International*

Enterprise Distributed Object Computing Conference, 2004, pp. 47–57.

[SKTB12] T. Skersys, L. Tutkute, and R. Butleris, "The Enrichment of Bpmn Business Process Model with Sbvr Business Vocabulary and Rules," *Journal of Computing and Information Technology*, vol. 20, no. 3, pp. 143–150, 2012.

[SKYT05] L. Skyttner, *General Systems Theory Problems, Perspectives, Practice*. Singapore; Hackensack, NJ: World Scientific, 2005.

[SLFE04] K. Subramaniam, D. Liu, B. H. Far, and A. Eberlein, "UCDA: Use Case Driven Development Assistant Tool for Class Model Generation," in *Proceedings of the 16th International Conference on Software Engineering and Knowledge Engineering*, Banff, Canada, 2004, vol. 324329.

[SOMM07] I. Sommerville, *Software Engineering*. Pearson Education, 2007.

[SPWH04] E. Sirin, B. Parsia, D. Wu, J. Hendler, and D. Nau, "HTN planning for web service composition using SHOP2," *Web Semantics: Science, Services and Agents on the World Wide Web*, vol. 1, no. 4, pp. 377–396, 2004.

[STRG03] R. D. Stevens, A. J. Robinson, and C. A. Goble, "Mygrid: Personalised Bioinformatics on the Information Grid," *Bioinformatics*, vol. 19, pp. 302–304, 2003.

[STRU10] A. Strunk, "QoS-Aware Service Composition: A Survey," in *Proceedings of the IEEE 8th European Conference on Web Services*, 2010, pp. 67 –74.

[STVV11] T. G. Stavropoulos, D. Vrakas, and I. Vlahavas, "A Survey of Service Composition in Ambient Intelligence Environments," *Artificial Intelligence Review*, pp. 1–24, 2011.

[SULY08] S. Su, F. Li, and F. Yang, "Iterative Selection Algorithm for Service Composition in Distributed Environments," *Science in China Series F: Information Sciences*, vol. 51, no. 11, pp. 1841–1856, Oct. 2008.

[TASW03] I. Taylor, M. Shields, and I. Wang, "Distributed P2P Computing Within Triana: A Galaxy Visualization Test Case," in *Proceedings of the International Parallel and Distributed Processing Symposium*, 2003, pp. 8–15.

[TAYL04] I. J. Taylor, *Peers in a Client-Server World: A Modern Perspective on Peer to Peer and Grid Computing*. New York: Springer, 2004.

[TRBP11] S. Truptil, F. Bénaben, and H. Pingaud, "On-the-Fly Adaptation of Crisis Response Information System," in *Proceedings of the 2011 International Conference on Management of Emergent Digital EcoSystems*, New York, USA, 2011, pp. 114–121.

[TSPI02] A. Tsalgatidou and T. Pilioura, "An Overview of Standards and Related Technology in Web Services," *Distributed Parallel Databases*, vol. 12, no. 2–3, pp. 135–162, Sep. 2002.

[VARB08] Y. Vanrompay, P. Rigole, and Y. Berbers, "Genetic Algorithm-Based Optimization of Service Composition and Deployment," in *Proceedings of the 3rd International Workshop on Services Integration in Pervasive Environments*, New York, USA, 2008, pp. 13–18.

[WADH02] P. Wohed, W. M. P. van der Aalst, M. Dumas, and A. H. M. ter Hofstede, *Pattern Based Analysis of BPEL4WS*. Brisbane, Australia: Queensland University of Technology, 2002.

[WADH03] P. Wohed, W. M. P. van der Aalst, M. Dumas, and A. H. M. ter Hofstede, "Analysis of Web Services Composition Languages: The Case of BPEL4WS," in *Proceeding of the 22nd International Conference on Conceptual Modeling*, 2003, pp. 200–215.

[WALD01] R. Waldinger, "Web Agents Cooperating Deductively," in *Proceedings of the First International Workshop on Formal Approaches to Agent-Based Systems*, London, UK, 2001, pp. 250–262.

[WEER05] S. Weerawarana, *Web services platform architecture: SOAP, WSDL, WS-Policy, WS-Addressing, WS-BPEL, WS-Reliable Messaging, and more*. Upper Saddle River, NJ: Prentice Hall PTR, 2005.

[WEHH10] G. Wen-yue, Q. Hai-cheng, and C. Hong, "Semantic Web Service Discovery Algorithm and Its Application on the Intelligent Automotive Manufacturing System," in *Proceedings of the 2nd IEEE International Conference on Information Management and Engineering*, 2010, pp. 601–604.

[WICK00] G. Wickler, *Using Expressive and Flexible Action Representations to Reason about Capabilties for Intelligent Agent Cooperation*. Edinburgh, UK: The University of Edinburgh, 2000.

[WONN07] D. G. Wonnacott, "Attribute Grammars and the Teaching of Compiler Design and Implementation," *Journal of Computing Sciences in Colleges*, vol. 22, no. 3, pp. 121–127, Jan. 2007.

[WRGS07] Z. Wu, A. Ranabahu, K. Gomadam, A. P. Sheth, and J. A. Miller, "Automatic Composition of Semantic Web Services using Process Mediation," in *Proceedings of the 9th International Conference on Enterprise Information Systems*, 2007.

[XCPM06] M. Xu, J. Chen, Y. Peng, X. Mei, and C. Liu, "A Dynamic Semantic Association-Based Web Service Composition Method," in *Proceedings of the 2006 IEEE/WIC/ACM International Conference on Web Intelligence*, Washington, DC, USA, 2006, pp. 666–672.

[XQYB09] P. Xiaoming, F. Qiqing, H. Yahui, and Z. Bingjian, "A User Requirements Oriented Dynamic Web Service Composition Framework," in *Proceedings of the 2009 International Forum on Information Technology and Applications*, 2009, pp. 173–177.

[YAPZ10] Y. Yan, P. Poizat, and L. Zhao, "Repair vs. Recomposition for Broken Service Compositions," in *Service-Oriented Computing*, P. P. Maglio, M. Weske, J. Yang, and M. Fantinato, Eds. Springer Berlin Heidelberg, 2010, pp. 152–166.

[YOUN01] R. R. Young, *Effective Requirements Practices*. Addison-Wesley Professional, 2001.

[YSLZ10] Z. Yang, C. Shang, Q. Liu, and C. Zhao, "A Dynamic Web Services Composition Algorithm Based on the Combination of Ant Colony Algorithm and Genetic Algorithm," *Journal of Computational Information Systems*, vol. 6, no. 8, pp. 2617–2622, 2010.

[YUBL10] T. Yue, L. C. Briand, and Y. Labiche, "An Automated Approach to Transform Use Cases into Activity Diagrams," in *Modelling Foundations and Applications*, Springer, 2010, pp. 337–353.

[YUBL11] T. Yue, L. C. Briand, and Y. Labiche, "A Systematic Review of Transformation Approaches Between User Requirements and Analysis Models," *Requirements Engineering*, vol. 16, no. 2, pp. 75–99, 2011.

[YUZL07] T. Yu, Y. Zhang, and K.-J. Lin, "Efficient Algorithms for Web Services Selection with End-to-End Qos Constraints," *ACM Transactions on the Web*, vol. 1, no. 1, p. 6–es, May. 2007.

[ZAPG09] E. Zahoor, O. Perrin, and C. Godart, "Rule-Based Semi Automatic Web Services Composition," in *Proceedings of the 2009 World Conference on Services*, 2009, pp. 805–812.

[ZAVE97] P. Zave, "Classification of Research Efforts in Requirements Engineering," *ACM Comput. Surv.*, vol. 29, no. 4, pp. 315–321, Dec. 1997.

[ZBDK03] L. Zeng, B. Benatallah, M. Dumas, J. Kalagnanam, and Q. Z. Sheng, "Quality Driven Web Services Composition," in *Proceedings of the 12th International Conference on World Wide Web*, New York, NY, USA, 2003, pp. 411–421.

[ZBND04] L. Zeng, B. Benatallah, A. H. H. Ngu, M. Dumas, J. Kalagnanam, and H. Chang, "Qos-Aware Middleware for Web Services Composition," *IEEE Transactions on Software Engineering*, vol. 30, no. 5, pp. 311–327, 2004.

[ZCFJ10] W. Zhang, C. K. Chang, T. Feng, and H. Jiang, "QoS-Based Dynamic Web Service Composition with Ant Colony Optimization," in *Proceedings of the IEEE 34th Annual Computer Software and Applications Conference*, 2010, pp. 493–502.

[ZHAN08] L.-J. L. Zhang, "Eic Editorial: Introduction to the Body of Knowledge Areas of Services Computing," *IEEE Transactions on Services Computing*, no. 2, pp. 62–74, 2008.

[ZHAO10] J. Zhao, "Applications of Norm and Situation Calculus in the Semantic Web Service Composition," *Journal of Software Engineering and Applications*, vol. 03, no. 08, pp. 776–783, 2010.

[ZHYA08] X. Zheng and Y. Yan, "An Efficient Syntactic Web Service Composition Algorithm Based on the Planning Graph Model," in *Proceedings of the 2008 IEEE International Conference on Web Services*, 2008, pp. 691–699.

[ZIVB13] E. Ziaka, D. Vrakas, and N. Bassiliades, "Web Service Composition Plans in OWL-S," in *Proceeding of the 3rd International Conference on Agents and Artificial Intelligence*, 2013, pp. 240–254.

[ZNBP08] L. Zeng, A. H. H. Ngu, B. Benatallah, R. Podorozhny, and H. Lei, "Dynamic Composition and Optimization of Web Services," *Distributed and Parallel Databases*, vol. 24, no. 1–3, pp. 45–72, Oct. 2008.

[ZZMX08] J. Zhou, T. Zhang, H. Meng, L. Xiao, G. Chen, and D. Li, "Web Service Discovery Based on Keyword Clustering and Ontology," in *Proceeding of 2008 IEEE International Conference on Granular Computing*, 2008, pp. 844–848.