# DYNAMIC WEB SERVICES COMPOSITION


A DISSERTATION SUBMITTED TO THE SCHOOL OF COMPUTER SCIENCE AND ENGINEERING OF THE UNIVERSITY OF NEW SOUTH WALES IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

Liangzhao Zeng

August 2003

‘I hereby declare that this submission is my own work and to the best of my knowledge it contains no materials previously published or written by another person, nor material which to a substantial extent has been accepted for the award of any other degree or diploma at UNSW or any other educational institution, except where due acknowledgement is made in the thesis. Any contribution made to the research by others, with whom I have worked at UNSW or elsewhere, is explicitly acknowledged in the thesis.

I also declare that the intellectual content of this thesis is the product of my own work, except to the extent that assistance from others in the projects design and conception or in style, presentation and linguistic expression is acknowledged.’

Liangzhao Zeng
August 8, 2003

# Acknowledgements

Many people have contributed, directly or indirectly, to the successful completion of this thesis. I would like to thank the following:

I would like to thank my supervisors. I have a great debt of gratitude with my supervisor, Dr. Boualem Benatallah. I own him most of what I have learned about how to do research. I also thank him for his encouragement, for his patience, and for teaching me the meaning and the importance of doing research in a "professional" way. My thanks also go to my co-supervisor, Dr. Anne Ngu. She gave me the blueprint of my research, as well as helped me find and focus on an exciting research topic.

Many thanks go to my colleagues when I worked in IBM T.J. Watson research center. In particular, I would like to thank Dr. Henry Chang. He has been my manager, mentor and friend. His vision, guidance, and experience helped me learn a lot, not only on how to conduct research, but also on how to be a real researcher. Dr. Jen-yao Chung gave me valuable directions on how to investigate and tackle real problems. Dr. Jayant Kalagnanam taught me linear programming. My thanks also go to Dr. Hui Lei and Dr. Jun-Jang Jeng for many fruitful discussion and valuable comments. They both treat me like an elder brother.

Furthermore, I would like to thank Dr. Marlon Dumas and Dr. Fethi Rabhi. It was wonderful time when worked with them on papers. From them, I learned the recipe for good papers.

Special thanks to my parents, Qingren Zeng and Jingxia Zhu, my sister Fangfei Zeng and my brothers Liangyi Zeng and Liangyuan Zeng who always encouraged and supported me throughout my studies.

# Contents

# List of Tables

# List of Figures

# Chapter 1

# Introduction

With ever developing globalized markets, today's business organizations operate their business in a globalized economy. In order to be more competitive in such dynamic economic environments, business organizations need to streamline their business processes. With the proliferation of the Internet and the wide acceptance of e-commerce, an increasing number of Web services are being offered. At the same time, the composition of Web services has gained a lot of momentum as a means to support business process automation. Our research is motivated by the need to facilitate the creation and execution of composite services.

This chapter is organized as follows. In Section 1.1, we outline the research issues that we tackle in this thesis. In Section 1.2, we summarize our solutions and in Section 1.3 we present the structure of this thesis.

## 1.1 Problem Statement

Web services represent a new generation of web application. They are self-contained, self-describing, modular applications that can be published, located, and invoked

across the Internet. Web services perform functions, which can be anything from simple operations to complicated business processes [70]. Once Web services are deployed, they can be aggregated into *composite services*. Indeed, process-based composition of Web services emerged as the technology of choice for integrating heterogeneous and loosely coupled applications across the Internet and organizations. When deploying business process management technologies to support dynamic Web service composition, there are three major issues that need to be addressed, namely: (i) Dynamic process schema creation, (ii) Dynamic Web service selection, and (iii) Service execution monitoring and change management.

### 1.1.1 Dynamic Process Schema Creation

One of the fundamental assumptions in most production workflow management systems (WFMS) [37, 47, 60] is that process schemas are static and predefined. In order to automate business processes, designers need to understand business processes and use modelling tools to chart process schemas. When a particular business process needs to be enacted, a process instance is created from a process schema. In this approach, the designer is required to explicitly define the tasks that compose business processes and specify relationships among them. However, it is impractical to compose services in many application domains. For example, it is extremely difficult, if not impossible, to predefine all composite service schemas for research and development (abbr R&D) processes in the automobile industry since R&D business processes are very dynamic. To meet the constraints and opportunities posed by new technologies, new markets, and new laws, business processes must be constantly redefined and adapted. However, this does not mean there are no business rules that govern R&D processes; it does not mean that planning for R&D processes is impossible. Indeed, there is a need to have an approach that enables automatic generation of process schemas customized to an organization's environment, business policies and business objectives.

### 1.1.2 Dynamic Web Service Selection

Composite services' *quality of service* (QoS, e.g., execution duration and execution price, etc.) is a key factor to satisfy end-user requirements. When composing existing Web services to execute business processes, the number of composed Web services may be large and the number of Web services to choose from may be even larger. Although some Web services may have similar functionalities, they may offer different QoS properties. In this context, it is a big challenge to optimize the QoS of composite services by selecting component services from a large number of Web services. Existing service selection solutions adopt mainly a *local dynamic selection approach*, meaning that they assign an individual task of the composite service to a component service, one at a time [8, 23, 38]. Such an approach is not appropriate to cater for global constraints and preferences. For example, the global constraints can be the minimization of the overall duration of the execution of the composite services, or the satisfaction of a given budget constraint. There is a need to have a *quality driven approach* to select and compose Web services.

### 1.1.3 Service Execution Monitoring and Change Management

Web services perform their functionalities in an autonomous way. They operate in a highly dynamic environment as new services may become available at any time, and existing services may be removed, become temporarily unavailable, offer better QoS properties, withdraw advertised QoS properties, etc. Moreover, enterprises are changing constantly: entering into new markets, introducing new products and restructuring themselves through mergers, acquisitions, alliances and divestitures. Runtime modification of business processes is necessary to meet changes in application requirements, technologies and business policies. This calls for adaptive composition techniques in which composite services will dynamically adjust their operations to respond rapidly

to exceptions (e.g., non-availability of a selected component service) and opportunities (e.g., availability of a new component service offering better QoS properties than existing ones, or emergence of a new business procedure offering a better approach to achieve a business goal).

## 1.2 Solution Overview

In this thesis, we present a dynamic Web service composition framework called $DY_{flow}$: $DY$namic intelligent $flow$. The contributions of this thesis comprise three major parts: rule-directed process schema generation, quality-driven service selection and adaptive service composition. Furthermore, a prototype has been developed to demonstrate the feasibility of the approaches proposed in this thesis.

### 1.2.1 Rule-directed Process Schema Generation

We propose an approach that supports rule-directed process schema generation [98, 102, 103]. In our approach, business objectives are declaratively defined, the process schemas are generated on demand for composite services, and composite services can be re-configured at runtime in order to adapt to changes. The features of our approach are:

- *A set of business rule templates*. We propose a set of business rule templates for modelling business policies. Traditionally, business rules are hard code into business processes. We argue that business rules should be independent of individual business processes. They can be re-used to generate different composite service process schemas [102].
- *A rule inference mechanism that combines backward-chain and forward-chain inference for dynamic process schema generation*. We propose a rule infer-

ence mechanism to dynamically generate process schemas for composite services. This is different from traditional business process modelling techniques where the business rules are implicitly codified in the process schemas (e.g., data and control flow constraints).

### 1.2.2 Quality-driven Service Selection

We propose an approach to enable quality driven Web services composition [97, 99, 100]. In the framework, QoS of Web services is evaluated by an extensible multi-dimensional quality model. The selection of Web services is driven by optimizing QoS of composite services. The salient features of our approach are:

- A *Web services quality model*. We propose an extensible multi-dimensional Web services quality model. The dimensions of this model characterize non-functional properties that are inherent to Web services in general: *execution price*, *execution duration*, *reputation*, *reliability*, and *availability*.

- *Quality-driven service selection*. We propose a global planning approach to select Web services. In this approach, quality constraints and preferences are assigned to composite services rather than to individual tasks within a composite service. Service selection is then formulated as an optimization problem and a linear programming method is used to compute optimal service execution plans for composite services. Experimental results show that the proposed service selection strategy can efficiently create optimal execution plans for composite services.

### 1.2.3 Adaptive Service Composition

We propose an approach that supports adaptive service composition. It is able to react to exceptions occurred in component services at runtime. At the same time, it also provides facilities to allow runtime modifications on composite services. The salient features of our approach are:

- *Adaptive service composition*. We propose an adaptive service composition approach in which composite services continuously monitor the behavior of their components and adapt themselves to appropriately react to run-time exceptions (e.g., component service failures, violation of QoS constraints) [101]. The adaptive behavior of services is centered around the concepts of *service coordinators* and *control tuple spaces*.

- *Handling unexpected exceptions*. We proposed an approach that handles unexpected exceptions by runtime modification on composite services [103]. During the execution of a composite service, the service composition checks the consistency between business policies and the process schema of the composite service. In addition, the adaptive service composition manager can automatically incorporate newly added business rules when there is a need to refine the composite services at run time.

## 1.3 Thesis Structure

This thesis is structured as follows. In Chapter 2, we introduce some basic concepts and definitions used in the thesis and survey some technologies that are related to Web service composition. In Chapter 3, we present details on dynamic process schema generation. In Chapter 4, we focus on quality-driven and dynamic service execution

planning. In Chapter 5, we discuss adaptive service composition. In Chapter 6, we present the prototype and experiment results. Finally, in Chapter 7, we provide the concluding remarks of the thesis and discuss some future work.

# Chapter 2

# Sate of the Art

This chapter gives an introduction to the research fields of service composition. Some concepts such as business process, workflow, etc., as well as some terminologies that will be used throughout this thesis are explained first. Then an overview of current service composition approaches is given. A further goal of this chapter is to review these research fields providing a broader context to the more specific issues that are discussed in the thesis.

This chapter is organized as follows: Section 2.1 introduces some background concepts and terminologies for this thesis. Section 2.3 provides a survey on service composition frameworks. Section 2.4 reviews some related research prototypes. Section 2.5 provides a brief evaluation of these frameworks and prototypes. Finally, the chapter is summarized in Section 2.6.

## 2.1 Background

In this section, some background concepts and terminologies that relate to research fields of service composition are introduced.

### 2.1.1 Business Process

A *business process* is a set of activities (also called tasks) which are performed collaboratively to realize a business objective or goal. Normally, business processes are owned by an organization; however, the activities involved can be conducted in different organizations. For example, the business process `claims processing` is owned by an insurance company. It consists of activities such as `claim receiving`, `customer checking`, `claim classification`, `lost analysising`, `auditing`, `payments & entitlements`, `claim settlement`, etc. It should be noted that these activities can be conducted in different organizations. For instance, the activity of `customer checking` is conducted by the insurance company, while the activity of `lost analysis` is conducted by a consultant company and the activity of `auditing` may require a licensed finance company to perform.

The *business process lifecycle* is depicted in Figure 2.1, which includes four steps from business process modelling to change management. The details of each step are given as follows:

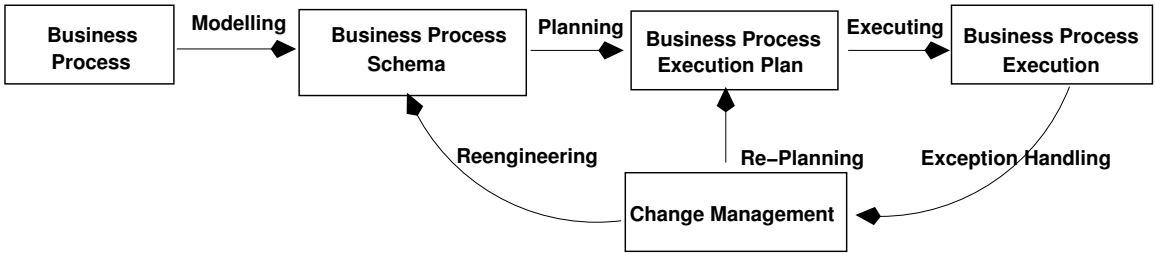



Figure 2.1: Business Process Lifecycle

- *Business Process Modelling.* Modelling the business processes results in a *process schema* (i.e., process model) that captures the three aspects of a business process: description of each activity in the business process, control flows and data flows among the activities.

- *Business Process Execution Planning.* Prior execution of a business process, it is necessary to conduct the planning first. Execution planning may include discovering the candidate participators (i.e., services and service providers), allocating the resources, and selecting appropriate participators, etc. Usually, there are multiple ways to execute the same business process. For instance, a task can be assigned to different service providers that offer different QoS properties. It is important to select appropriate participators based on certain quality criteria, such as *execution time* and *execution price*, so that the QoS of the business process is optimal.

- *Business Process Execution.* Executing business process means invoking services to execute the tasks and enable control and data flows among the tasks. The task may include *material* and/or *electronic* processes. Material processes comprise designing, manufacturing, transforming, measuring, assembling physical object, etc. An example of material process can be assembling cars in automobile industry. Electronic processes, on the other hand, involve software systems and humans to create and manage the information and transactions. The *material* process can have an *electronic* presentation, and can therefore be managed by the software system.

- *Change Management.* During the execution of business processes, it is always possible that some exceptions occur. For example, business conditions may change, which require modification of business processes; some allocated resources may become unavailable, users who expected to carry out the activities are absent, etc. These kinds of exceptions may require the re-engineering of the business processes, re-planning the execution, etc.

### 2.1.2 Workflow

Workflow is a technology that manages business processes automation [26]. The Workflow Management Coalition(WfMC) [88] defines workflow as "the automation of a business process, in whole or part, during which documents, information or tasks are passed from one participant to another for action, according to a set of procedural rules". In the following subsection, some basic concepts in workflow are introduced.

#### Workflow Schema and Workflow Instance

A *workflow schema* denotes the design of a workflow, which is a formal description of various aspects of a business process, such as the activities to be carried out, dependencies/relationships that exist among the activities (i.e., data flow and control flow among the activities). Usually, workflow schemas can be defined by scripting languages (e.g., XPDL [96]) or graph modelling tools (e.g., Statechart [41], Petri Net [10, 71, 82, 90], etc.). A comprehensive survey on workflow modelling can be found in [83]. An arbitrary number of *workflow instances* (i.e., cases) can be generated based on a workflow schema. Each workflow instance represents one individual enactment of a business process, using its own runtime data.

#### Workflow Management System

In [89], the WfMC defines *workflow management system* (WFMS) as:

> A system that defines, creates and manages the execution of workflows through the use of software, running on one or more workflow engines, which is able to interpret of the process definition, interact with workflow participants, and, where required, invoke the use of IT tools and applications.

Following this definition, *workflow management* involves the modelling of *workflow schemas* and enactment of *workflow instances*. Accordingly, a WFMS consists of two main functional components: a *buildtime* component and a *runtime* component. The buildtime component provides a workflow definition language, and together with appropriate tools, such as editors, browsers, compiler, etc., allow workflow designers to define workflow schemas. On the other hand, the runtime component supports the creation, planning and enactment of workflow instances according to workflow schemas. Also, a WFMS normally provides monitoring tools that allow users to keep the track of execution progress of workflow, as well as detect and handle the exceptions that occur during the execution of workflow instances. A WFMS is a system that enables the automation of business processes.

## 2.2 Overview of Service Composition

In the previous section, we introduced business processes and their automation. In this section, we will focus on a new approach which enables business process automation by service composition. In the first part of this section, we illustrate an example of composite service. Then we present a typical architecture of a service composition framework. After that, we identify some issues in service composition. These issues are used to compare different service composition frameworks and prototypes.

### 2.2.1 Example

In this example, we use an application from the finance domain. Assume that there is a collection of *component services* provided by finance industry vendors (i.e., service provider), which enable a variety of functions such as market analysis, security[1]

[1] A security is the basic unit being traded which can be a stock, option, future etc.

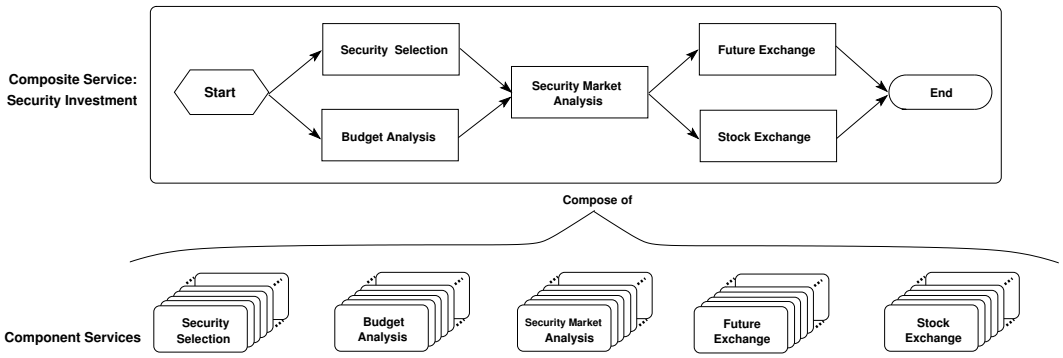



Figure 2.2: Service Composition: A Running Example

exchange, and investment decision-making. An example of a composite service is illustrated in Figure 2.2. This service is part of an integrated investment-making business process, consisting of five component services, which selects the most profitable security to be traded, performs an analysis of the budget available, determines the most appropriate type of market on which to conduct the trade, then executes the trade on a given exchange. For each task, the composite service selects a component service to execute it.

### 2.2.2 Architecture of a Service Composition Framework

The Internet is emerging as the platform for organizations to provide their services[2]. In this new platform, process-based service composition framework has emerged as a key solution for managing business processes. The architecture of *service composition framework* is depicted in Figure 2.3, which includes *component services*, *composite services* (usually, composite services are associated with service composition engine), and *service brokers*.

1. **Component Service**

   Component services are own by service providers, which are software applications for specific needs. In order to participate in a composite service, a service

[2]In remainder of the thesis, we use term *Web service* , *e-service* and *service* interchangeably

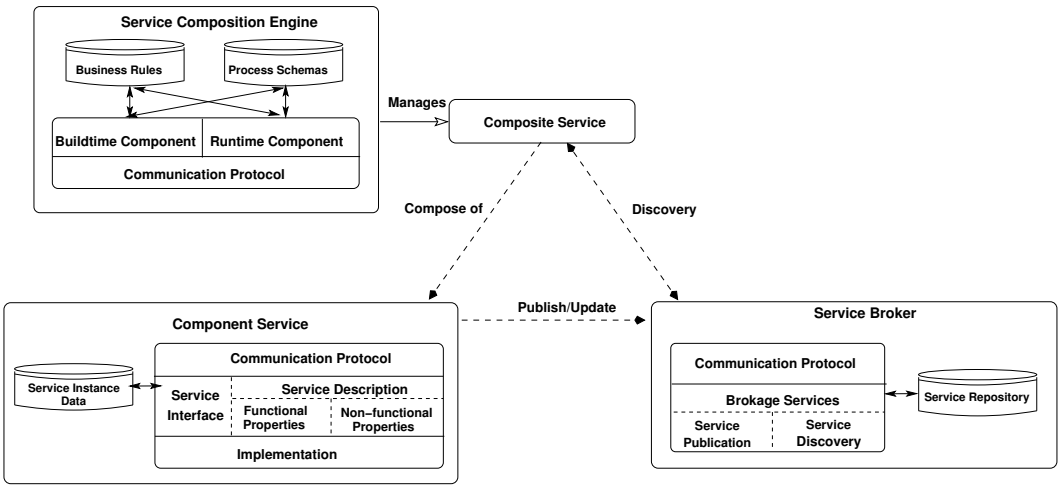



Figure 2.3: Architecture of Service Composition Framework

provider needs to advertise the service to the service broker. The components of a service may include: *communication protocols*, *service interface*, *service description* and *implementation*. Communication protocols (e.g., HTTP, RMI, JMS (Java Message Service), etc.) are used for exchanging messages between the component services, composite services, and service brokers. It is possible that each service uses different proprietary network protocols. For example, vendor `A` that provides `Market Analysis` services may allow HTTP access, while vendor `B` that provides `Future Exchange` service may accept JMS messages. The service composition framework needs to provide a common communication protocol that allows services to communicate with each other. Usually, protocol translators should be used to translate messages between services' own network protocols and a common protocol.

A service interface provides the access point to invoke a service. Usually, service interfaces are bound with communication protocols. Service description includes two major parts: functional and non-functional properties. Another component in a service is the implementation. Different service providers can have totally different approaches to implement services. For example, one service provider may user JAVA to implement the service, while another service

provider may implement the service on the mainframe using COBOL. Service wrappers can be used to provide the mapping between the implementation and the service interface.

2. **Composite Service**

   A composite service represents a business process. It consists of a collection of component services. Often, composite services are supported by the *Service Composition Engine* that selects and orchestrates the component services. The *Service Composition Engine* may consist of: *communication protocols*, *build-time component* and *runtime component*. The buildtime component defines or generates process models, while the runtime component executes composite services, including locating, selecting, and orchestrating component services.

3. **Service Broker**

   The service broker is a computation entity that acts as a mediator between component and composite services. Again, it also requires communication protocols to exchange information with composite services and component services. The key component of a service broker is brokage services. The basic brokage service may include a naming service that consists of service publication and discovery. Usually, the service broker maintains a service repository that stores the Meta-data of services.

### 2.2.3 Issues

In the view of business process management, composite services are similar to workflows. Both of them aim to enable the flow of business processes. As composite services operate in an open and dynamic environment, issues to be addressed may include: *Service Description*, *Service Advertisement & Discovery*, *Service Provisioning*, *Business Process Modelling*, *Service Selection* (i.e., execution planning), *Composite*

*Service Orchestration* and *Adaptive Composition*.

- Service Description. In order to participate in composite services, service providers first need to specify properties of a service as service descriptions. Typically, a services has functional and non functional properties. The functional properties specify what the service can do (e.g., operations, input and output parameters), while non-functional properties include the service of quality, transaction, conversation, security, etc. It is important for service composition frameworks to provide languages that specify both functional and non-functional properties.

- Service Advertisement & Discovery (A&D). To be noticed and selected by composite services, component services need to be advertised first. In the meantime, composite services need to discover the component services that can match the task execution requirements (both functional and non-functional). Usually, there is a service broker that enables service advertisement and discovery.

- Composite Service Modelling. Typically, composite services represent complex business processes, which may include the activities that the services consists of (i.e., the *tasks* of the composite service) and interactions among the tasks (e.g., control-flow, data-flow, and transactional dependencies). These composite service models can be hard-coded, or scripted as process schemas. Obviously, the approach that adopts process schemas provides more flexibility than a hard-coded approach. It should be noted that process schemas can be pre-defined, or generated on demand.

- Service Selection (execution planning). Once the process schema is defined, the composite service needs to select component services to execute it. The

number of services providing a given functionality may be large and constantly changing. Consequently, approaches where the development of composite services requires the identification at design-time of the exact component services to be composed are inappropriate. The runtime selection of component services during the execution of a composite service has been put forward as an approach to address this issue. Service selection can be local optimal or global optimal. In the local optimal mode, for each task in composite services, a component service is selected that aims for optimizing the QoS of executing a task, without considering the QoS of composite services. On the contrary, in the global optimal mode, the task assignment aims for optimizing the QoS of composite services.

- Composite Service Orchestration. When a composite service has an underlying process schema, it is necessary to have a composition engine to orchestrate the execution of composite services. In practice, there are two different approaches: Centralized and Peer-to-Peer. In the centralized mode, the composition engine is responsible for invoking component services. It executes control flows and enables message exchange among the component services. In the peer-to-peer mode [9], the composition engine deploys some fragment of data and control flow into the component services. At runtime, the component services coordinate with each other to execute the data and control flow.

- Adaptive Composition. Web environment is highly dynamic. New Web services may become available at any time. Existing Web services may be removed, become temporarily unavailable, offer better QoS properties, withdraw advertised QoS properties, etc. On the other hand, business environments also highly dynamic, which requires composite services dynamically to change its business goal, operations, etc. Therefore, two kinds of exceptions may occur during the execution of composite services: expected and un-

expected. The expected exceptions are related to execution results of a task. For example, the hotel room can not be booked since the price is changed. The unexpected exceptions are related to the inconsistencies between process schemas that are used to execute composite services and business processes in real world. Therefore, there is a need for adaptive composition techniques in which composite services will dynamically adjust their operations to respond rapidly to exceptions (e.g., non-availability of a selected component service) and opportunities (e.g., availability of a new component service offering better QoS than existing ones).

## 2.3 Service Composition Frameworks

In this section, we survey some industry and organization (i.e., standard body) efforts on service composition.

### 2.3.1 Service Composition By Programming

Service composition can be done using general-purpose programming languages. In this approach, services are provided via an Application Programming Interface (API) of a programming language, for example, C API. In addition to an API, service providers also need to set up network communication protocols so that service requestors can invoke the API, for example, TCP/IP sockets. Service providers provide API manuals to describe services. Usually, since the service broker is absent from this approach, the service requestors have to understand the API manuals that are given by the service providers and manually select the appropriate services. If the selected services are supported by different communication protocols, then service requestors need to setup different network communication channels to assess these services. For example, if each component service of the composite service `Security`

`Investment` is supported by different protocols, then the developer needs to handle five different protocols. At the same time, the business logic of composite services must be hard-coded. Any modifications on business logic require re-compiling of the programs. For example, when service requesters want to replace a component service `Market Analysis` (provided by vendor A) in the composite service `Security Investment` with new one (provided by vendor B), it is necessary to manually modify and re-compile the programs. The major shortcomings of this approach are: it is tightly coupled, lacks adaptability, and not easily scalable. In order to overcome these problems, people propose service composition frameworks to facilitate creation and execution of composite services. In the following subsections, we survey some service composition frameworks and research prototypes.

### 2.3.2 Component-based Middleware

A *Software Component* is a physical packaging of executable and published interface [45]. The concept of software components has been around virtually since the beginning of software, it has now evolved as a framework to support component (i.e., service) composition. Szyperski [80] provides the following definition: "A software component is a unit of composition with contractually specified interface and explicit context dependencies only. A software component can be deployed independently and is subject to composition by third parties". Currently, there are several components-based middleware that have gain commercial support, such as DCOM, CORBA, and EJB. In the following subsection, we take CORBA as an example to illustrate how the component-based middleware supports service composition.

CORBA is the acronym for Common Object Request Broker Architecture, which is defined by Object Management Group (OMG) [73]. It is an open, vendor-independent architecture and infrastructure that computer applications use to work together over networks.

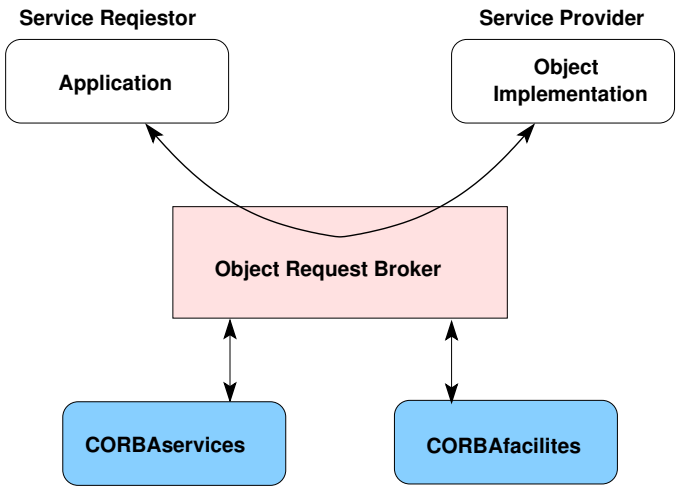



Figure 2.4: CORBA architecture

CORBA proposes a *Reference Model* (see Figure 2.4), which consists of the following components: *Application Objects*, *Object Request Broker*,*CORBAservices* and *CORBAfacilities* [27].

Application Objects (i.e., services) are individual units of running software that combine functionality and data. Services that objects provide are given by interfaces, where interfaces are defined by OMG's Interface Definition Language (IDL). According to our service composition model, IDL is the standard language for defining service interfaces.

Object Request Broker (ORB) is the service broker in CORBA, which enables objects to transparently receive and response to requests in a distributed environment. It is the foundation for building applications from distributed objects and for interoperability between applications in hetero- and homogeneous environments.

CORBAservices (i.e., Object Services) is a collection of services (interfaces and objects) that support basic functions for using and implementing objects. Services are necessary to construct any distributed application and are always independent of application domains. For example, the *Life Cycle Service* defines conventions for creating, deleting, copying, and moving objects; it does not dictate how the objects are imple-

mented in an application.

CORBAfacilities (i.e., Common Facilities) is a collection of services that many applications may share, but which are not as fundamental as the Object Services. For instance, a system management or electronic mail facility could be classified as a Common Facility.

In a summary, the CORBA provides an infrastructure to support communications among service providers and service requestors, which hides the underlying complexity of network communication protocol from the service composition developers. CORBA provides IDL to define service interfaces. At the same time, it provides naming and trader services which allow service providers to register services and service requestors to discover services. However, IDL only defines the syntactic of services, it does not provide support for semantic description of services. Again, the semantic description (especially the non-functional properties) of services is given by API manuals and the participants of a certain composite service need to agree on a predefined interface. Another problem is that the ORB is not only responsible for service registry and discovery, but also manages the invocation of services. It can be a bottleneck for the execution of composite services. Other drawbacks of CORBA-based framework include scalability and availability issues. In a summary, CORBA is suitable for statically composed services among a small number of organizations. However, it provides little or no support for management of the changes in component services' interface. The component services in the composite services are tightly coupled. Hence, it is not suitable when the number of component services in composite services is very big and business processes are highly dynamic and extend across multiple organizations.

### 2.3.3 W3C Web Service Framework

Web services are information, software packages, software processes and computation resources delivered over the Internet using XML messages (e.g., SOAP message), which serve a particular set of user needs. They are self-contained, self-describing components that can be published, located, and used across the Internet. The rapid growth of the Internet's functionality and bandwidth has created tremendous opportunities for organizations of any size to adopt Web services to diversify their businesses and become truly global. Web services (e.g., order procurement, customer relationship management, finance, accounting, human resources, supply chain and manufacturing) have built very high expectations for organizations to establish deeper relationships with partners. Widely available and standardized Web services make it possible to realize just-in-time Business-to-Business Interoperability (B2Bi) by composing Web services.

The Web services architecture (see Figure 2.5) is centred around protocols and standards proposed by W3C (World Wide Web Consortium) [85] and related organizations. W3C was created in October 1994 to lead the World Wide Web to its full potential by developing common protocols that promote its evolution and ensure its interoperability. W3C has around 500 member organizations from all over the world and has earned international recognition for its contributions to the growth of the Web.

This architecture describes the principles behind the next generation of e-business architectures. It presents a logical evolution from the object-oriented system to the service-oriented system. In the following, we will introduce the essential protocols and components that construct this framework.

- SOAP (Simple Object Access Protocol) [78] is a network protocol that enables communication among Web services. Actually, SOAP was initially created by Microsoft and later developed in collaboration with Developmentor,

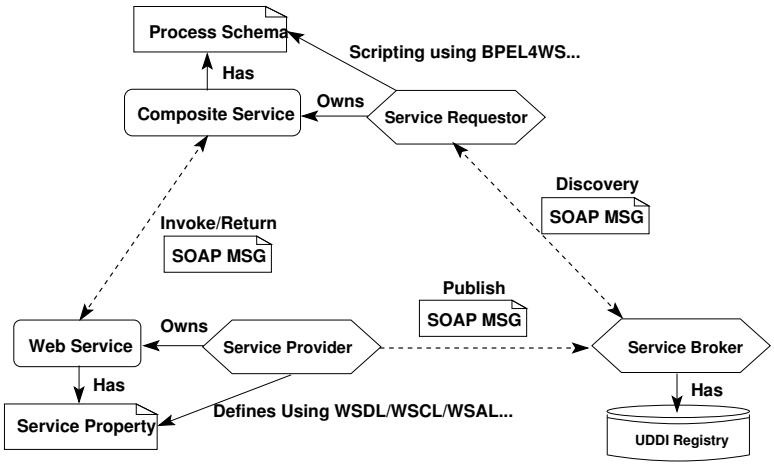



Figure 2.5: Web Service Architecture

IBM, Lotus and Userland. SOAP is an XML based protocol for messaging and remote procedure calls (RPCs). Rather than defining a new transport protocol, SOAP works on exiting network transport protocols, such as HTTP, SMTP, MQSeries, FTP, etc. The transport element in the protocol is a SOAP message, which is an XML document with a very simple structure: an XML element with two child elements: the header and the body. Both the header and the body elements are themselves XML nodes. In addition to the basic message structure, the SOAP specification also defines a model that dictates who should process messages and how recipients should process SOAP messages. From the view of a service composition framework, SOAP can be used as a common communication protocol for the framework.

- WSDL (Web Service Description Language) [93] is an XML format for describing Web services as a set of endpoints (i.e., *interfaces*) operating on messages containing either document-oriented or procedure-oriented information. The interfaces are described abstractly, and then bound to a concrete network protocol and message format. WSDL is extensible to allow description of endpoints and their messages regardless of what message formats or network protocols are used to communicate. From the view of a service composition

framework, WSDL can be considered as a common language that defines service interfaces.

- WSLA Language (Web Service Level Agreement language) [52, 61] is proposed by IBM. It is targeted at defining and monitoring SLAs (e.g., Service of Quality) for Web services. A WSLA document defines assertions of a service provider to perform a service according to agreed guarantees for IT-level and business process-level service quality parameters such as response time and throughput, and measures to be taken in case of deviation and failure to meet the asserted service guarantees, for example, a notification of the service customer. The assertions of the service provider are based on a detailed definition of the service parameters including how basic metrics are to be measured in systems and how they are aggregated into composite metrics. WSLA agreement focuses on non-functional properties of Web services, which complements service interfaces. From the view point of a service composition framework, WSLA language can be used as the common language that specifies the QoS property and agreement.

- WSCL (Web Services Conversation Language) [92] is proposed by HP. WSCL specifies the XML documents being exchanged by Web services, and the allowed sequencing of these document exchanges. WSCL conversation definitions are themselves XML documents and can therefore be interpreted by Web services infrastructures and development tools. From the view point of a service composition framework, WSCL can be used as a common language that specifies the conversation properties of services.

- WSCI (Web Service Choreography Interface) [91] is codeveloped by BEA Systems, Intalio, SAP AG, and Sun Microsystems. It is an XML-based interface description language that describes the flow of messages exchanged by a Web Service participating in choreographed interactions with other ser-

vices. It describes the observable behavior of a Web Service. This is expressed in terms of temporal and logical dependencies among the exchanged messages, featuring sequencing rules, correlation, exception handling, and transactions. From the view of a service composition framework, WSCI is similar to WSCL, which can be used as a common language that specifies the conversation properties of services.

- BPEL4WS (Business Process Execution Language for Web Services) [12] is proposed by IBM, Microsoft and BEA for defining business processes. BPEL4WS represents a convergence of the ideas in the XLANG [95](structural constructs for processes) and WSFL [94](support for graph oriented processes) specifications. Both XLANG and WSFL are superseded by the BPEL4WS specification. BPEL4WS supports the modelling of two types of business processes: *executable* and *abstract* processes. An abstract, (not executable) process is a business protocol, specifying the message exchange behaviour between different parties without revealing the internal behaviour for any one of them. An executable process, specifes the execution order between a number of *activities* constituting the process, the partners involved in the process, the messages exchanged between these partners, and the fault and exception handling specifying the behaviour in cases of errors and exceptions. The basic activities defined in BPEL4WS include: invoking a Web service operation; terminating Web service instance, etc. From the view of a service composition framework, BPEL4WS is a language that allows service requestors to specify composite services.

- UDDI (Universal Description, Discovery and Integration) [81] is a service registry specification proposed by an industry consortium led by IBM, Microsoft and Ariba. The UDDI is a platform-independent, open framework for describing services, discovering businesses, and integrating business services

using the Internet, as well as an operational registry that is available today. It proposes a unified and systematic way to publish and discover a Web service. UDDI provides two basic specifications that define a service registry's structure and operation [29]:

- A definition of the information to provide about service and how to encode it; and
- A query and update API for the registry that describes how service information can be accessed and updated.

From the view point of a service composition framework, UDDI can be considered as a service broker.

### 2.3.4 DAML-S Framework

DAML-S [30] is a DAML+OIL ontology for Web services developed by a coalition of researchers [2], under the auspices of the DARPA Agent Markup Language (DAML) program. It attempts to provide an ontology, within the framework of DAML, for describing the properties and capabilities of Web services in an unambiguous, computer-interpretable form. It focuses on enabling agent technologies for automated Web service discovery, execution, interoperation, composition and execution monitoring.

The DAML-S ontology describes a set of classes and properties, specific to the description of Web services. The upper ontology of DAML-S comprises the *service profile* for describing service advertisements, the *process model* for describing the actual program that realizes the service, and the *service grounding* for describing the transport-level messaging information associated with execution of the program. The *service grounding* is akin to the Web Service Description Language (WSDL) in the W3C framework.

In DAML-S the process model provides a declarative description of the properties of the Web-accessible programs. It conceives each program as either an atomic or composite process. It additionally allows for the notion of a simple process, which is used to describe a view, abstraction or default instantiation of the atomic or composite process to which it expands. An atomic process is a non-decomposable Web-accessible program. It is executed by a single (e.g., http) call, and returns a response. It does not require an extended conversation between the calling program or agent, and the Web service. In contrast, a composite process is composed of other composite or atomic processes through the use of control constructs. These constructs are typical programming language constructs such as sequence, if-then-else, while, fork, etc. that dictate the ordering and the conditional execution of processes in the composition.

### 2.3.5 BPMI Framework

BPMI.org (the Business Process Management Initiative) [13] is a non-profit corporation that empowers companies of all sizes, across all industries, to develop and operate business processes that span business partners over the Internet. It focuses on promoting and developing Business Process Management (BPM) through the establishment of standards for process design, deployment, execution, maintenance, and optimization. The open specifications defined by BPMI.org include the Business Process Modeling Language [14] (BPML), and the Business Process Query Language (BPQL). These two languages aim for enabling the standard-based management of e-Business processes with forthcoming Business Process Management Systems (BPMS), in much the same way Structured Query Language (SQL) enabled the standards-based management of business data with off-the-shelf Database Management Systems (DBMS).

BPMI.org considers an e-Business process conducted between two business partners as made of three parts: a public interface and two private implementations (one for each partner). The public interface is common to the partners and is supported by protocols

such as ebXML [33]. The private implementations are specific to each partner and are described in any executable language. BPML is one such language.

Once developed, the private implementation of an e-Business process must be deployed on a platform that will actually execute it. For this purpose, BPMI.org defines BPQL, a standard management interface, for the deployment and execution of e-Business processes. Furthermore, BPQL relies on UDDI in order to provide a standard way to register, advertise, and discover the public interfaces of e-Business processes.

### 2.3.6 ebXML Framework

ebXML [33] is an international initiative established by United Nations Centre for Trade Facilitation and Electronic Business (UN/CEFACT) and OASIS. The proposed architecture can be viewed as a service composition framework. It includes the following components:

1. A standardized business *Messaging Service* framework that enables the interoperable, secure and reliable exchange of messages among trading partners. From the view of a service composition framework, it is a common communication protocol for the framework.

2. A standard mechanism (Collaboration Protocol Profile - CPP) for describing a *Business Process* (i.e., service) and its associated information model such as company profiles. From the view of a service composition framework, it enables service descriptions.

3. A mechanism (ebXML Registry) for registering and storing *Business Process* and *Information Meta Models* so they can be shared and reused. From the view of a service composition framework, it can be considered as the service broker.

4. A mechanism (Collaboration Protocol Agreement - CPA) for describing the execution of a mutually agreed upon business arrangement which can be derived from information provided by each participant. From the view of a service composition framework, it specifies the agreement among business partners.

5. A metamodel (ebXML Business Process Specification Schema - BPSS) for specifying a collaboration (i.e., composite services) between business partners. An ebXML *Business Process Specification* contains the specification of business transactions and the choreography of business transactions into collaboration.The *Business Process Specification* is therefore incorporated with or referenced by an ebXML trading partner CPP and CPA. Each CPP declares its support for one or more roles within the Business Process Specification. Further technical parameters are then added to these CPP profiles and CPA agreements resulting in a full specification of the run-time software at each trading partner.

Figure 2.6 depicts a sequence of steps to establish and conduct business collaborations between two trading partners using ebXML architecture.

In this scenario, Company A is a service provider, while Company B is a service requestor. In the service register phase, Company A becomes aware of an ebXML Registry that is accessible on the Internet (see Figure 2.6, step 1). Then, Company A, after reviewing the contents of the ebXML Registry, decides to build and deploy its own ebXML compliant application (see Figure 2.6, step 2). Custom software development is not a necessary prerequisite for ebXML participation. ebXML compliant applications and components may also be commercially available as shrink-wrapped solutions. Company A then submits its own business profile information (including implementation details and reference links) to the ebXML Registry (see Figure 2.6, step

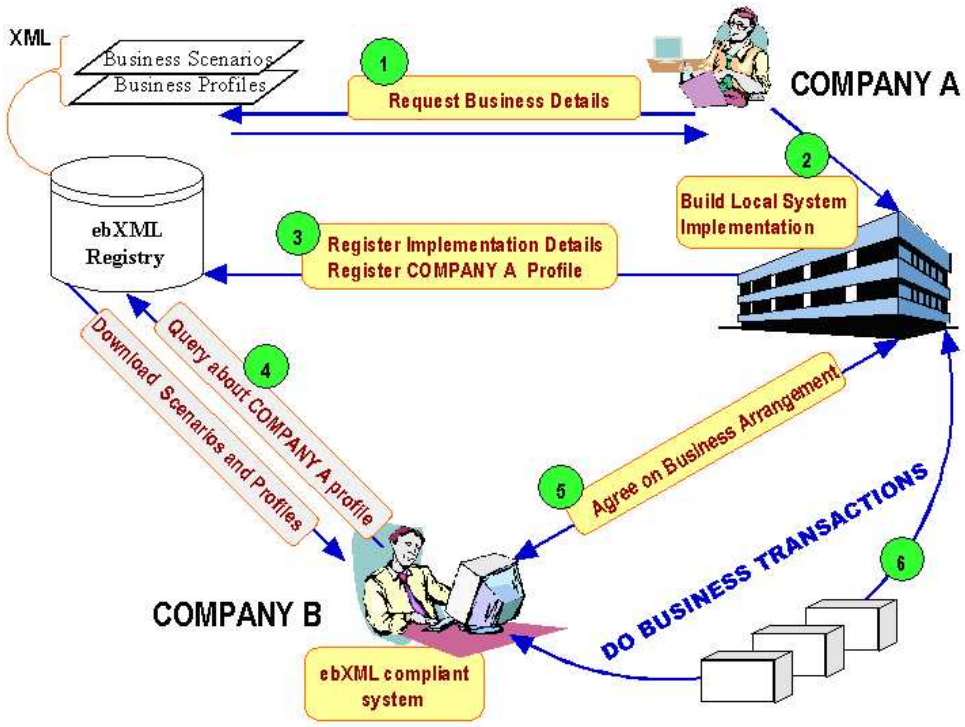



Figure 2.6: A High Level Overview of the Interaction of Two Companies Conducting a Business Process Using ebXML [33]

3). The business profile submitted to the ebXML Registry describes the company's ebXML capabilities and constraints, as well as its supported business scenarios. These business scenarios are XML versions of the business processes and associated information bundles (e.g. a sales tax calculation) in which the company is able to engage. After receiving verification that the format and usage of a business scenario is correct, an acknowledgment is sent to Company A (see Figure 2.6, step 3).

In the service discovery phase, Company B discovers the business scenarios supported by Company A in the ebXML Registry (see Figure 2.6, step 4). Company B sends a request to Company A stating that they would like to engage in a business scenario using ebXML (see Figure 2.6, step 5). Company B acquires an ebXML compliant shrink-wrapped application.

In the service provisioning phase, before engaging in the scenario Company B submits a proposed business arrangement directly to Company A's ebXML compliant soft-

ware Interface. The proposed business arrangement outlines the mutually agreed upon business scenarios and specific agreements. The business arrangement also contains information pertaining to the messaging requirements for transactions to take place, contingency plans, and security-related requirements (see Figure 2.6, step 5). Company A then accepts the business agreement. Company A and B are now ready to engage in a business process using ebXML(see Figure 2.6, step 6).

It should be noted that the ebXML specifications are not limited to the above scenario. More complicated service composition scenarios, such as three or more service providers conducting business using shared business processes, are also supported.

## 2.4 Research Prototype

In this section, we present some research projects related to business process integration and service compositions.

### 2.4.1 CMI

CMI (Collaboration Management Infrastructure) [4, 38] is a platform to manage collaboration process. It supports service modelling, service broker, dynamic service selection and service integration (composition). The *Collaboration Management Model* (CMM) is used to combine the business process and service management in a single model. CMM consists of a Core Model (CORE) and several specialized models. CORE provides a common set of process model primitives that constitute the basis for all extensions. These primitives include constructs for defining resources, roles and generic state machines. The CMM extensions include *Coordination Model* (CM), *Service Model* (SM) , and *Awareness Model* (AM). The *Coordination Model* provides primitives for coordinating participant and automating process management, which are

used by the service requestor to specify and manage composite services. The *Service Model* is used by service providers to support service interfaces, service activities and related wrapper processes, and service contracts. The *Awareness Model* allows the monitoring of process related events, and the authorized composition and delivery of such events only to process participants that need to perform their roles.

The CMI components include the user tools and engines. User tools in CMI are organized into to separate clients for participants and designers. A CMI participant is a human or program individual involved in a collaboration process. A designer creates and maintains CMM process, awareness, and service specifications. Participants interact with the CMI enactment system using CMI client tools for participants. The client tools subsume the tools defined by the Workflow Management Coalition. In particular, participant tools include a worklist tool, an awareness information viewer, and a process monitoring tool. The CMI worklist divides the activities that each participant is eligible to perform into mandatory or optional work items. The awareness information viewer maintains a participants awareness event queue and displays awareness events to him/her. The CMI enactment system contains several engines that implement the CMM. The CORE Engine implements the primitives of the CORE. It is used by the other engines that are responsible for the CM, SM and AM. The Coordination Engine implements the CM by mapping CM primitives to process fragments of a WfMS, i.e., IBM FlowMark, and orchestrates them at process execution time. The Awareness Engine that implements AM and the Service Engine implements SM.

In CMI, the business processes are defined as state machine based models, which are executed by a workflow engine. The adaptive processes are enabled by the notion of *Repeated Optional Dependency*, which includes two parts: *repeated optional creator* and *repeated optional terminator*. They handle the situation when both the exact invocation place in the control flow path of the process and the number of invocations cannot be specified beforehand.

### 2.4.2 eFlow

eFlow [21, 22, 23] is a platform that supports specifying, enacting and monitoring of composite services. In eFlow, a composite service is described as a process schema that consists of other basic or composite services. A composite service is modelled by a graph (the flow structure), that defines the order of execution among the nodes in the process. It may include *service*, *decision*, and *event* nodes. Service nodes represent the invocation of a basic or composite service, where a service node specification includes a service selection rule. The rule is executed by an eFlow component called a *service process broker*, which enables the selection of the appropriate service and service provider. Decision nodes specify the alternatives and rules controlling the execution flow. Event nodes enable service processes to send and receive several types of events.

eFlow architecture consists of the *Service Process Composer*, *Service Process Engine*, *Service Process Broker*, *Service Operation Manger* and *Migration Manger*. Composite services are specified through the *Service Process Composer*, which enables the definition and modification of process schemas. Services are enacted by the service process engine, which is composed of the *Scheduler*, the *Event Manager*, and the *Transaction Manager*. The scheduler contracts the *Service Process Broker* in order to discover the actual service (and service provider) that can fulfill the requests specified in the service node definition. The scheduler eventually contracts the provider in order to execute the service. The *Event Manager* monitors event occurrences, by detecting temporal, data and workflow events, and by receiving notifications of application-specific events by external application. The *Transaction Manager* enforces transactional semantics by enabling compensation of *transaction regions*, where the transactional region identifies a portion of the process graph that should be executed in an atomic fashion. If for any reason the part of the process identified by the transactional region cannot be successfully completed, then all running services in the region are aborted and completed ones are compensated, by executing a service-specific compensating action. The *Ser-*

*vice Operation Manger* logs the execution details and provides correction assistance if desired. It is the component through which authorized users can monitor running service executions or retrieve information about completed service execution. The eFlow also includes a *Migration Manager* that handles dynamic changes to process instances. When migrating process instances, it suspends instances to be migrated, verifies that the migration can be actually performed, generates an execution state for the instance in the new schema, and finally informs the engine that the execution of the migrated instance can be resumed.

### 2.4.3 IE (Internet Enterprise)

IE (Internet Enterprise) [43, 49, 68] is an infrastructure for enabling virtual enterprise. In the IE, service providers can enable its business to be programmatically accessible on the Internet (becomes an e-service). The architecture treats e-services as workflow participants in Internet wide workflow automation applications. Enabled by a scalable brokering service and an inter-organizational workflow facility, services offered by autonomous service providers can be composed into an "Internet Workflow" (i.e. composite services). There are three core components in the infrastructure:

1. BizBuilder [58], an e-service framework that is used to created E-services. In BizBuilder, e-services are presented using WSDL. The non-functional properties are specified as constraints using the Constraint-Based Requirement Specification Language [79]. These constraints on e-services can be constraints on the service attributes, or constraints on the input attributes and service attributes of the operations. The E-services Adapter is the core component that allows the underlying service (e.g., service implemented using Java) to be accessed on the Internet using standard protocol and format.

2. Sangam [44], a scalable, hierarchical brokering community based on UDDI,

which is responsible for service advertising and discovering.

3. A dynamic workflow engine that uses e-services as entities, to create and enact workflow models (i.e., process schema).

Collectively, these three components empower the Internet as a "public enterprise", where virtually any person or organization can design workflow models, and hence create new businesses, using available, competing e-services. At runtime, the workflow engine accomplishes the dynamic service binding with the help of the broker server, which performs the constraint-based [79] service provider selection, where the constraints restrict the selection of proper e-service providers for e-service requests.

### 2.4.4 METEOR

METEOR [18, 69] is a workflow management system that enables QoS management and service composition. A QoS model that allows for the description of nonfunctional aspects of workflow component is developed, where the QoS model is composed of four dimensions: time, cost, reliability, and fidelity. It has a mathematical model to compute overall QoS of workflow, which can be used to estimate, predict, and analyze the QoS of production workflow. In order to support service composition, Web services are described by extended DAML-S. The Web service specification composed of syntactic description, operational metrics (e.g., QoS dimensions), and semantic information. The matching between the service template (i.e., task in composite service) and service object is based on syntactic, operational and semantic similarity functions. The proposed architecture provides registry and discovery services which allow service providers to advertise services and service requestors to locate services. At runtime, composite services are executed by the workflow management system. Runtime exceptions are handled by a case-base reasoning mechanism.

### 2.4.5 SELF-SERV

SELF-SERV [8, 9, 77] is a platform supporting the rapid composition and scalable orchestration of Web services. It proposes a declarative language based on Statechart for composing Web services. The concept of *service community* is considered as the architect for the composition of a potentially large number of dynamic services. Service communities are essentially containers of alternative services. They describe the capabilities of a desired service without referring to any actual provider. Actual providers can register with any community of interest to offer the desired service. At run-time, the community is responsible for selecting the service offer that best fits a particular user profile in a specific situation. Another feature of SELF-SERV is the *peer-to-peer* service execution model, whereby the responsibility of coordinating the execution of a composite service is distributed across several peer software components called *coordinators*. Coordinators are attached to each involved service. They are in charge of initiating, controlling, monitoring the associated services, and collaborating with their peers to orchestrate the service execution. The knowledge required at runtime by each of the coordinator in a composite service (e.g. location, peers, and routing policies) is statically extracted from the service's Statechart and represented in a tabular form. In this way, the coordinators do not need to implement any complex scheduling algorithm.

The SELF-SERV adopts a layered architecture (i.e., service, communication, directory, and user layers) to provide support for discovery, creating and deploying Web services:

- Service Layer. It consists of a collection of composite services and containers. It features a class ContainerWrapper that defines methods for invoking operations provided by containers and collecting the outputs of an invocation. When a user, an application program, or a state coordinator invokes an operation of a container, the ContainerWrapper object corresponding to

this container invokes the corresponding scoring service. The scoring service takes as input the containers selection policy and the list of members registered with the container, returning the identifier of one of the members. The service layer also provides two classes, StateCoordinator and InitialCoordinator, that constitute the runtime environment required to perform peer-to-peer orchestration; service providers must install the classes before participating in a composite service. The StateCoordinator class implements methods for receiving, processing, generating, and dispatching control flow notifications according to a given routing table. In addition to these functionalities, the InitialCoordinator class implements methods for invoking an operation of a composite service and collecting the results of the invocation.

- Conversation Layer. It provides support for standardized interactions among services. For example, it allows business partners to share their external business processes according to a specific B2B standard, such as Electronic Data Interchange (EDI), RosettaNet, or cXML, which defines common formats and semantics for messages (such as request for quote or purchase order) and business process conversations (such as chronology of message exchanges). The conversation layer consists of a set of predefined service templates for various B2B standards.

- Directory Layer. It consists of a set of directories that store metadata about services and containers. Metadata directories contain information that describes the meaning, categories, properties, capabilities, location, and access information of the available services. The user layer of Self-Serv uses the metadata to locate, browse, and query services and containers. Service providers can advertise metadata in service directories such as UDDI or ebXML registries.

- User Layer. It provides access to the service composition environment through three main components: the *service discovery engine*, the *service builder*, and

the *service deployer*. The service discovery engine facilities the advertisement and location of services and their operations. The service builder allows the developer to create and configure composite services and service containers. The service deployer generates and deploys routing tables for every state in the composite service state chart to enable peer-to-peer service orchestration.

## 2.5 Comparison of Service Composition Frameworks and Prototypes

In this section, we compare the different service composition frameworks and Prototypes. The comparison is conducted based on the issues we illustrated in Section 2.2.3. In table 2.1, key service composition frameworks are compared. Currently, most of the framework efforts focus on enabling the inter-operations among services. A comprehensive survey on Business-to-Business interaction can be found in [67]. In table 2.2, some service composition prototypes are compared. One of the current trends is to provide flexible service composition solutions that can offer optimal execution result of composite services.

## 2.6 Summary

In this chapter, we first explained some basic concepts in business processes and then gave a brief survey of existing and ongoing service composition frameworks and research projects. The current research trend in service composition is to provide a scalable, low setup and development cost, flexible, and adaptive solution. It is however so vital to consider how to optimize QoS of composite services. In the next chapter we will present our solution for dynamic service composition.

Table 2.1: Service Composition Frameworks

| | *Service Description (functional)* | *Service Description (non-functional)* | *Service A&D* | *Composite Service Modelling* | *Service Selection* | *Composite Service Orchestration* | *Adaptive Composition* |
|---|---|---|---|---|---|---|---|
| *Programming* | API manual | Not Addressed | Not Addressed | Hard-code in programs | Manual Selection | Hard-code | Hard-code |
| *CORBA* | IDL | Not Addressed | IDL, Naming and trading services | Not Addressed | Not Addressed | Not Addressed | Not Addressed |
| *W3C Framework* | WSDL | WSCL or WSCI for conversation | UDDI | BPEL4WS | Not Addressed | Not Addressed | Application and excepted exception handling |
| *DAML-S* | Service Profile | Service Profile | Not Addressed | Process Model and Ontology | Not Addressed | Not Addressed | Not Addressed |
| *BMPI* | BPML | Not Addressed | UDDI | Not Addressed | Not Addressed | BPMS | Expected exception handling |
| *ebXML* | CPP | CPA for service agreement | ebXML Repository Not Addressed | BPSS | Not Addressed | Not Addressed | Application and excepted exception handling |

Table 2.2: Service Composition Prototypes

| | *Service Description (functional)* | *Service Description (non-functional)* | *Service A&D* | *Composite Service Modelling* | *Service Selection* | *Composite Service Orchestration* | *Adaptive Composition* |
|---|---|---|---|---|---|---|---|
| *CMI* | Service model | Ontology, service model | Service broker | Pre-defined state machine based model | Service selection policies | IBM FlowMark, centralized | Expected exception handling by repeated optional dependency |
| *eFlow* | e"speak | e"speak | Service broker | Pre-defined state machine based model | Service selection rule | Service operation manger, centralized | Expected and unexpected exception handling |
| *IE* | WSDL | Service constraints | UDDI | Pre-defined process schemas | Constraint based selection, local optimization | Workflow engine,centralized | Expected exception handling |
| *Meteor* | Service profile(DAML-S) | Ontology, QoS Model | Not Addressed | Pre-defined process schemas | Syntactic, operational and semantic similarity functions | Workflow engine, centralized | Case-based reasoning exception handling |
| *SELF-SERV* | WSDL | QoS Model | UDDI | Pre-defined Statecharts | Local optimization | Service coordinator, peer-to-peer | Not Addressed |
| $DY_{flow}$ | WSDL | Service ontology, QoS Model | UDDI | Generated Statecharts | Local or global optimization | Composition engine, centralized | Peer-to-Peer exception handling, component, expected and unexpected exception handling |

# Chapter 3

# Generating Process Schemas for Composite Services

The process-based composition of Web services is emerging as a promising approach to automating business process within and across organizational boundaries [4, 21, 8]. In this approach, individual Web services are federated into composite Web services whose business logic is expressed as a process model. Business process automation technology such as workflow management systems (WFMSs) can be used to choreograph the component services. However, one of the fundamental assumptions of most WFMSs is that workflow schemas are static and predefined. Such an assumption is impractical for business processes that have an explosive number of options, or dynamic business processes that must be generated and altered on the fly to meet rapid changing business conditions. For example, business processes for managing product lifecycle need to be generated and modified at runtime. In this chapter, we describe a rule inference framework, where end users declaratively define their business objectives or goals and the system dynamically composes Web services to execute business processes.

This chapter is organized as follows. In Section 3.1, we first identify some research

issues and then outline our solutions for these issues. In Section 3.2, we introduce a real world example, to be used for further illustration of our approach; the running example stems from a product lifecycle management case study, which provides motivations for our work. Section 3.3 gives an overview of the proposed process schema generator. Section 3.4 describes the details of rule inference algorithms for dynamic process schema generation. Finally, we discuss related work in Section 3.5 and provide a summary of this chapter in Section 3.6.

## 3.1 Introduction

Web services technologies are emerging as a powerful vehicle for organizations that need to integrate their applications within and across organizational boundaries. In particular, the process-based composition of Web services is gaining considerable momentum as an approach for the effective integration of distributed, heterogeneous, and autonomous applications [6]. In this approach, applications are encapsulated as Web services and the logic of their interactions is expressed as a process model. This approach provides an attractive alternative to hand-coding the interactions between applications using general-purpose programming languages [7].

The wide availability and standardization of Web services make it possible to compose basic Web services into *composite services* that provide more sophisticated functionality and create add-on values [24]. For example, a composite service can provide a high-level financial management service that uses individual payroll, tax preparation, and cash management Web services as components. The process model underlying a composite service identifies the functionalities required by the services to be composed (i.e., the *tasks* of the composite service) and their interactions (e.g., control-flow, data-flow, and transactional dependencies). Technologies for modelling and executing business processes are generally referred to as WFMSs.

Our work is motivated by the requirements of dynamic Web services composition. Web environments are highly dynamic, Web services can appear and disappear around the clock. So, approaches that statically compose services are inappropriate. Business process automation technology such as production WFMS can be used to choreograph the component services. However, one of the fundamental assumptions in most production WFMSs [37, 47] is that process schemas are static and predefined. In order to automate business processes, designers need to understand the processes and use modelling tools to chart process schemas. When a particular business process needs to be enacted, a process instance is created from a process schema. In this approach, the designer is required to explicitly define the tasks that compose business processes and specify relationships among them. However, it is impractical to compose services in many application domains. For example, it is extremely difficult, if not impossible, to predefine all composite service schemas for research and development (abbr R&D) processes in the automobile industry since R&D business processes are very dynamic. To meet the constraints and opportunities posed by new technologies, new markets and new laws, business processes must be constantly redefined and adapted. However, this does not mean there are no business rules that govern R&D processes; it does not mean that planning for R&D processes is impossible too. Indeed, there is a need to have a system that enables automatic generation of process schemas customized to an organization's environment, business policies and business objectives.

At the same time, since the global economy is volatile and dynamic, organizations are changing constantly: entering into new markets, introducing new products and restructuring themselves through mergers, acquisitions, alliances and divestitures. Moreover, most composite services are long running, representing complex chains of tasks. Organizations that executing composite services may need to change business processes to adapt the changes in application requirements, technologies, business policies, conditions and constraints. Consequently, runtime modification of composite services is

necessary to meet these changes. However, in most process modelling techniques, such runtime modification is a manual procedure, which is time-consuming and costly to enforce. In order to reduce the cost and provide fast responses to these changes, there is a need to automate the runtime modification of composite service schemas.

In this chapter, we present the design and implementation of a process schema generator: In the generator, business objectives (i.e., business goals) are declaratively defined, and the process schemas are generated on demand. The salient features are:

- *A set of business rule templates*. We propose a set of business rule templates for modelling business policies. Traditionally, business rules are hard code into business processes. We argue that business rules should be independent of individual business processes. They can be re-used to generate different composite service schemas [102].
- *A rule inference mechanism that combines backward-chain and forward-chain inference for dynamic process schema generation*. We propose a rule inference approach to support dynamic service composition. This is different from traditional business process modelling techniques where the business rules are implicitly codified in the process schemas (e.g., data and control flow constraints).

## 3.2 A Motivating Example

In this section, we present an example of product lifecycle management in the automobile industry. Let us assume that an automobile company decides to build a new prototype of a sedan car. The chief engineer proposes to build the new prototype car by replacing the old `petrol engine` with a new `electric engine`. In order to achieve this business goal (i.e., replacing the engine), a sequence of tasks need to be

conducted, which includes: (1) new electric engine development; (2) new parts layout design; (3) parts design, development & manufacturing and (4) assembling and testing (see Figure 3.1).

1. **New Engine Development**

   The automobile company has two different alternatives to obtain a new electric engine: it can outsource the development of the electric engine to other companies or it can develop the electric engine in-house. If the cost of outsourcing is over $2000K, then the company will design and develop the electric engine in-house. If the company wants to outsource the electric engine, a suitable vendor needs to be selected first. If a domestic vendor is selected, then the national quality control procedure should be applied, otherwise, international quality control procedure needs to be applied. There are many alternative processes in each type of quality control procedure; the selection of processes depends on the requirements of the electric engine, budgets, and relationships with vendors, etc.

2. **Parts Layout Design**

   Replacing the engine of a sedan car will require system engineers to conduct parts layout design. This is because the new and old engines may have different 3D specifications. Some old parts may need to be replaced. New unique parts may be required by the electric engine. System engineers will first conduct a 3D layout evaluation to decide whether to have a complete new layout design or to re-use parts of the old layout design. The layout feasibility analysis has three possible alternatives: (1) change the electric engine's 3D specification; (2) change related parts' 3D specifications; or (3) change both engine and related part's 3D specifications. For each alternative, there is a set of optional processes; the selection of these processes is based on the results of a layout feasibility analysis. Within each process, there are many possible

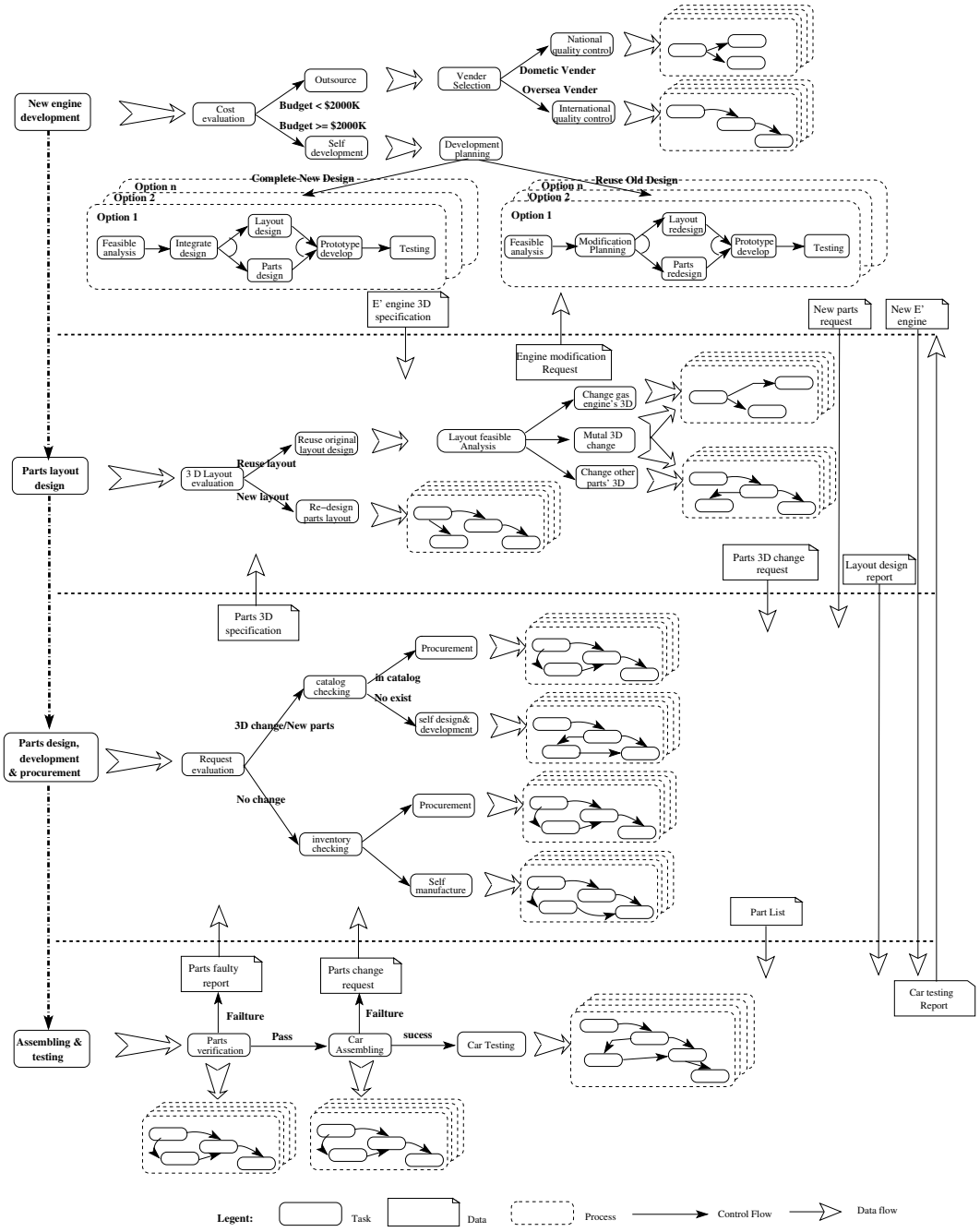



Figure 3.1: A Motivating Example

options for which tasks need to be invoked by the process, and many possible data flows and control flows among the tasks.

3. **Parts Design, Development & Procurement**

   Both the new electric engine and the layout design may require the design and the development of new parts for the sedan car. Three basic processes are involved to obtain a new part, namely (1) procurement, (2) self design & development, and (3) self manufacture. The decision on which processes should be adopted is dependent upon the result of request evaluation, catalog checking and inventory checking.

4. **Assembling and Testing**

   When the new electric engine and parts are ready, workshop engineers will verify them. Any faults on these parts will be reported to the related part-obtaining processes. If a part passes the verification, then workshop engineers can assemble the parts. Any problem on assembling will generate a part modification request. When workshop engineers finish the assembling of the car, the test engineers will conduct a sequence of tests, where each test will generate a report. The report may recommend some modifications of the electric engine or other parts. In each step, there are several options for tasks and control flow constraints. The final decisions depend on the type of parts, the budget, testing standards, etc. For example, different countries have different car safety standards. If the automobile company wants to sell the new car in a particular country, then testing based on that country's car safety standard needs to be used.

The process of developing a new sedan car by replacing the `petrol engine` with an `electronic engine` will take at least half a year with hundreds of new parts to be designed, developed and manufactured. There are an explosive number of tasks

and control flow relationships in this R&D product development process. When predefining process schemas, only the general knowledge (i.e., business rules) can be used. The schemes need to consider all the possible options. However, it is almost impossible to predefine process schemas for such an R&D product process since it is too complex and time consuming to exhaustively enumerate all the possible options. This calls for dynamic generation of composite service schemas based on both business rules and context information (such as user profiles that define users' role, preference, etc.). For example, during the R&D process, a user (i.e., engineer designer) is assigned to develop a new electronic engine under $2000K budget. In this case, a composite service can be generated based on current available services, requirement of the new electronic engine, her profile and the budget constraints, without exhaustively enumerating all the possible options in engine development. Another challenge is that this R&D product process is a long running process, during which many changes may occur. For example, during the enactment of this R&D product process, better quality materials or new batteries that can be used for the electric engine may become available. The R&D product process needs to adapt to these changes by modifying process schemes. In production workflows, such adaptations are very costly, especially when workflow schemas are complex. However, if the composite service schema is dynamically generated, adaptation can be done by re-generating composite service schemas. In the following sections, we will use this motivating example to illustrate how composite service's process schemas are generated in our framework.

## 3.3 Design Overview

In this section, some basic definitions and concepts in dynamic process schema generation are explained first. We then present the main characteristics of our approach.

### 3.3.1 Preliminaries

In this subsection, we introduce some important concepts and definitions that are used in process schema generation.

#### Service Ontology

A service ontology (see Figure 3.2) specifies a common language agreed by a community (e.g., automobile industry). It defines basic concepts and terminologies that are used by all participants in that community. In particular, a service ontology specifies a *domain* (e.g., `Automobile`, `Healthcare`, `Insurance`), a set of *synonyms*, which is used mainly to facilitate flexible search for the domain (for example, the domain `Automobile` may have synonyms like `Car`), and a set of *service classes* that are used to define the property of the services. A service class is further specified by its *attributes* and *operations*. For example, the attributes of a service class may include access information such as URL. Each operation is specified by its name and signature (i.e., inputs and outputs). Apart from the service classes that are used to describe functional properties of services, service ontology also specifies the *services quality model* that is used to describe non-functional properties of the services, e.g., execution duration of an operation. The services quality model consists of a set of quality dimensions (i.e., criteria). For each quality dimension, there are three basic components: what's the definition of criteria; which service elements (e.g., services or operations) it relates to; how to compute or measure the value of criteria. More details about the service quality model can be found in the next chapter.

The service ontologies are organized as a tree structure: the root service ontology can be used by all the communities; child nodes automatically inherit parent's properties such as service classes, and can extend service classes that are used in their own communities. In our framework, *business objectives*, *business rules*, and *process schemas*

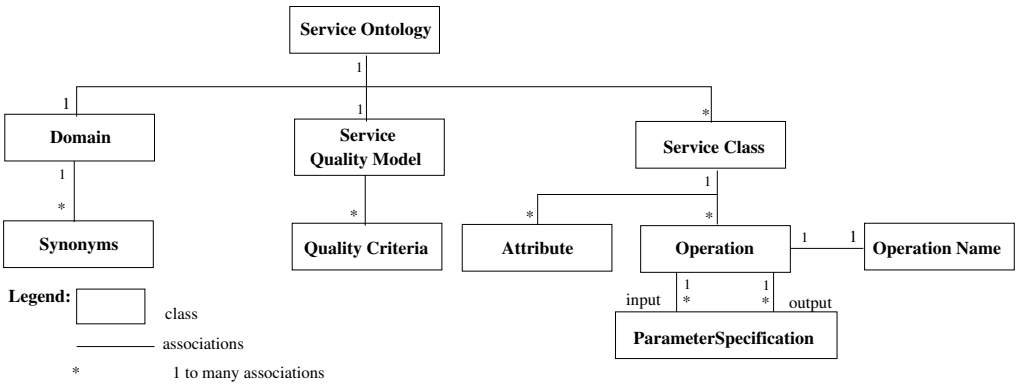



Figure 3.2: UML Class Diagram for Service Ontology

are specified over defined sets of service ontologies.

### Initial State and Business Objectives

An initial state represents an end user's starting point (i.e., initial task) of a business process, while business objectives represent goals (e.g., target tasks) that the user wants to achieve. We develop an XML schema based language that allows users to specify their initial state and business objectives. In our framework, both initial state and business objectives are defined in terms of a pre-defined service ontology. For both initial state and business objectives, users need to specify the operation name as defined in the service ontology. In initial state, users can provide the constraints on the input of the operation; while in the business objectives, users can provide constraints on both input and output of the operation. In Table 3.1, the XML document illustrates a business objective where a user wants to change a leaded petrol engine in a sedan car to an unleaded petrol engine.

### Business Rule Templates

Business rules are statements about how business is conducted, i.e., the guidelines and restrictions with respect to business processes in an enterprise. We propose using

```
< businessObjectives >
   <User Name="Gerg" Role="Chief Engineer" / >
   <targetTask Name="change engine for sedan">
   <ontology-service NAME="Automobile Engine" / >
      < operation NAME="changeEngine" >
         < data-constraint >
            <variable dataName="car" dataItemName="type" />
            <op value="="/>
            <value>"sedan"</value>
         </data-constraint >
         < data-constraint >
            <variable dataName="originalEngine" dataItemName="type" />
            <op value="="/>
            <value>"Leaded Petrol"</value>
         </data-constraint >
         < data-constraint >
            <variable dataName="newEngine" dataItemName="type" />
            <op value="="/>
            <value>"Unleaded Petrol"</value>
         </data-constraint >
      </operation>
   </targetTask >
</ businessObjectives >
```

Table 3.1: Business Objective

business rule templates to facilitate the description of business policies. There are two categories of business rule templates:

1. **Service Composition Rules.** Service composition rules are used to dynamically compose services. There are three types of service composition rules, namely backward-chain rules, forward-chain rules and data flow rules.

   - *Backward-chain rules* indicate preconditions of executing a task. A precondition can specify data constraints, i.e., some data must be available before the execution of the task. A precondition can also be a flow constraint, i.e., execution of the task requires other tasks to be completed beforehand. A backward-chain rule is specified using the following structure:

     **BACKWARD-CHAIN RULE** < `rule-id`>

**TASK** < task> **CONSTRAINT** <constraint>

**PRE-CONDITION** <pre-condition>

The following is an example of a backward-chain rule. This rule defines that if a user wants to conduct a costAnalysis task on a new part, where the part's type is a car engine, then the systemTest and clashTest tasks need to be completed first.

**BACKWARD-CHAIN RULE** bcr1

**TASK** costAnalysis

**CONSTRAINT** costAnaysis::partType== 'car engine'

**PRE-CONDITION** complete_task (systemTest) AND complete_task (clashTest)

- *Forward-chain rules* indicate that some tasks may need to be added or dropped as a consequence of executing a given task. Forward-chain rules are defined as ECA (Event-Condition-Action) rules:

**FORWARD-CHAIN RULE** <rule-id>

**EVENT** <event> **CONDITION** < condition>

**ACTION**< action>

In the following forward-chain rule, if task engineCostAnalysis is completed and the makingCost of the new part is greater than $2000K, then the task audit_2 needs to be executed after the task costAnalysis.

**FORWARD-CHAIN RULE** fcr1

**EVENT**

TaskEvent::complete_task(engineCostAnalysis)

**CONDITION**

`(engineCostAnalysis::makingCost>`

`$2000K)`

**ACTION** `execute_task(audit_2)`

- *Data flow rules* are used to specify data flows among tasks. Each task in a business process may have input and output parameters. For those tasks that require an input, data flow rules can be used to specify the data source. The general form of a data flow rule is given as follows:

  **DATA FLOW RULE** < `rule-id>`

  **CONSTRAINT** `<constraint>`

  **DATA-SOURCE TASK**

  < `task>` **DATA-ITEM**< `data-item>`

  **DATA-TARGET TASK** <

  `task>`**DATA-ITEM**< `data-item>`

  A data source can be a task or human users. For example, in the following data flow rule, task `designEngine`'s output `engineType` provides input for task `testEngine`'s `engineType`.

  **DATA FLOW RULE** `dfr1`

  **DATA-SOURCE TASK** `designEngine` **DATA-ITEM**

  `output::engineType`

  **DATA-TARGET TASK** `testEngine` **DATA-ITEM**

  `input::engineType`

2. **Service Selection Rules.** Service selection rules identify a particular algorithm or strategy to be used for choosing a Web service to execute tasks during the runtime. The general form of a service selection rule is as follows::

   **SERVICE SELECTION RULE** `<rule-id>`

**TASK**< task> **CONSTRAINT**< constraint>
**SERVICE-SELECTION**<
service-selection-method>

For each task, there is a set of candidate Web services that can perform the task. Currently, we adopt a Multiple Criteria Decision Making (MCDM) [59] approach to select Web services. However, methodologies other than MCDM, such as auction or negotiation, can also be adopted by the system to support selection of Web services. MCDM is a configurable decision model for quality-based Web services selection. This decision model takes into consideration some service quality criteria such as execution price, execution duration, etc. Our implementation of MCDM based Web service selection can be found in the next chapter. In the following service provider selection rule, if the task is `buildEngine` and the engine's weight is greater than 3000Kg, execution price is used as the criteria to select Web services.

**SERVICE SELECTION RULE** ssr1
**TASK** TaskEvent::start_task(buildEngine)
**CONSTRAINT** (bulidEngine::engine.weight >
3000Kg)
**SERVICE-SELECTION** service_selection(MCDM,
execution price)

In our framework, we expect domain experts to define the various business rules using the above rule templates. For example, domain experts in service outsourcing may define service selection rules, while domain experts in product life cycle may define service composition rules.

### Process Schema

A process schema of a composite service is defined in terms of service ontologies. It consists of a collection of generic tasks combined in certain ways. In our framework, we adopt Statechart [41] to represent the process schema. Statechart is the board extension of the conventional formalism of state machines and state diagrams with essentially three elements, dealing, respectively, with the notions of hierarchy, concurrency and communication. An example of process schema or a business process template for `making new part` is shown graphically in Figure 3.3. The choice of Statechart for specifying composite Web services is motivated by two main reasons: (i) Statechart have a well-defined semantics; and (ii) they offer the basic flow constructs found in contemporary process modelling languages (i.e., sequence, conditional branching, structured loops, concurrent threads, and inter-thread synchronization). The first characteristic facilitates the application of formal manipulation techniques to Statechart models, while the second characteristic ensures that the service composition mechanisms developed in the context of Statechart, can be adapted to other process modelling languages, like for example those that are being designed by Web services standardization efforts (e.g., BPEL4WS, BPML).

A statechart is made up of states (also called tasks) and transitions. In the proposed framework, the transitions of a Statechart are labelled with events, conditions, and assignment operations over process variables. States can be *basic* or *compound*. Basic states are labelled with an operation name in a service class of a service ontology. Compound states contain one or several statecharts within them. Specifically, compound states come in two forms: OR and AND states. An OR-state contains a single Statechart, whereas an AND-state contains several Statechart (separated by dashed lines) which are intended to be executed concurrently. Accordingly, OR-states are used as a decomposition mechanism for modularity purposes, while AND-states are used to express concurrency: they encode a fork/join pair. The initial state of a statechart is

denoted by a filled circle, while the final state is denoted by two concentric circles: one filled and the other unfilled.

There are two types of tasks in process schemas, namely an *atomic task* and *composite task*. The atomic task can be executed by a primary Web service, while a composite task is a sub-process that contains multiple primary Web services (i.e., composite services) to execute it. In our system, both the process schema for a whole business process and the sub-process schemas for the composite tasks in the business process are not statically predefined, they are dynamically generated at runtime.

It should be noted that tasks in process schemas are not bound with any Web services at definition time. Transitions represent the control flows in schemas and are defined by ECA (Event Condition Action) rules. The basic semantics of an ECA rule is as follows: when an event occurs, the condition is evaluated. If the condition evaluates to true, then the corresponding action is activated. Assume there is a transition between task $t_i$ and $t_j$, then the transition is denoted as $TS_{t_i \rightarrow t_j}$.

Each task in a process schema can be seen as having an input and an output. The input and output can be referenced in any of the conditions and actions in transitions. In addition to input and output, the conditions and the actions of a transition in a statechart may refer to other variables, namely *internal* variables. An internal variable can be used in , e.g., one of the condition part of a transition. To summarize, a variable appearing in the process schema can be: an input of a task, an output of a task, or an internal variable. The value of a variable can be: (i) requested from the user during the execution of the composite service, or (ii) obtained from the output of a task in the composite service. Accordingly, there are two forms of data flows in process schemas, i.e., data flow between two tasks, denoted as $DF_{t_i \rightarrow t_j}(d)$, which representing task $t_i$ provides data (i.e., variable) $d$ to task $t_j$; data flow between the user and tasks, denoted as $DF_{user \rightarrow t_j}(d)$, which representing user provides data $d$ to task $t_j$;

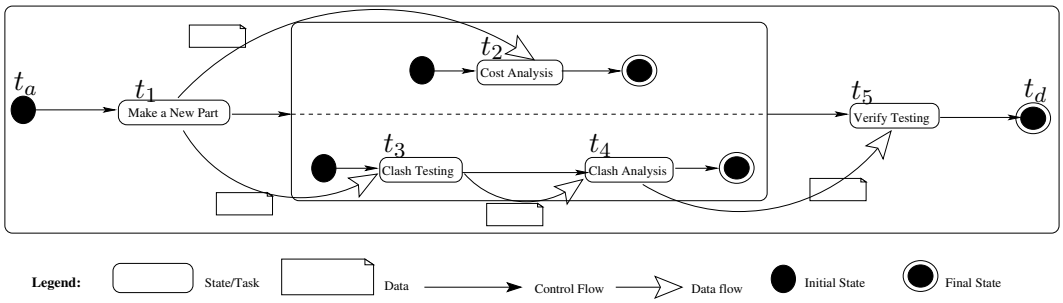



Figure 3.3: Defining Process Schema Using a Statechart

### Organizational Structure

In our framework, the organizational structure is used to glue the basic elements in an organization (real or virtual) such as *Department*, *Role*, *User*, *Business Rule*, and *Service* together (see Figure 3.4). An organization is subdivided into several departments that again may consist of other departments in a hierarchical structure. Each department is associated with a service ontology, a set of roles, users, and services. Every role is associated with a set of business rules. Every user is assigned one or more roles describing his/her context-dependent behavior and has his/her own *user profile*. The information in a user profile includes the user's personal information, roles in the organization, and preferences. In $DY_{flow}$, the user profile facilitates the generation and execution of composite services by providing the initial context information and users' preferences. When the system generates composite service schemas, personal information provides input for executing composite services, user's role information

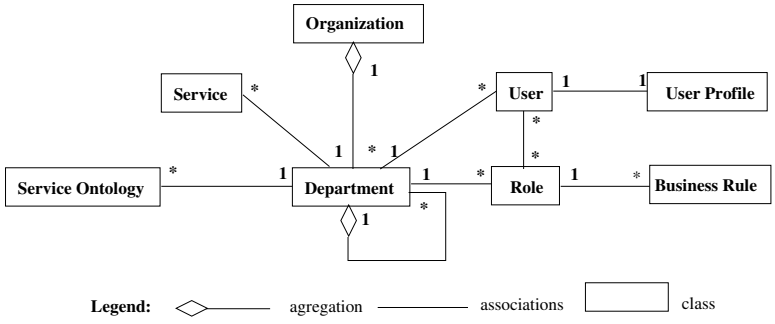



Figure 3.4: UML Class Diagram for Organizational Structure

**FORWARD-CHAIN RULE** `fcr3`
**EVENT**
`TaskEvent::complete_task(partPriceQuatation)`
**CONDITION** `(partPriceQuatation::price > orderBudgetLimitation)`
**ACTION** `execute_task(budgetApproval)`

Table 3.2: A Forward-chain Rule

is used to identify relevant business rules, and preferences are used to customize the business rules. For example, in the forward-chain rule `fcr3` (see Table 3.2), the variable `orderBudgetLimitation` in the condition statement is left uninstantiated. This value will be substituted by a preference (e.g., `orderBudgetLimitation = $20K`) in a user's profile. It should be noted that different users may have different constraints for `orderBudgetLimitation`.

### 3.3.2 Incremental Service Composition

Instead of using a single composite service schema to represent the whole business process, in our framework, we identify different levels of process schema. A process schema that is used to initiate a new business process is defined as the *top-level process schema*; a process schema that is used to execute a composite task in a process schema is defined as *task-level process schema* (i.e., sub-process schema). In addition, these two types of process schemas are organized in a hierarchical structure as shown in Figure 3.5. It should be noted that all the process schemas in the system are created on the fly based on user's business objectives using the business rules; the composition hierarchy is built from an initial top-level process schema which gets expanded into a composition hierarchy at runtime. In this subsection, we first give the procedure for process schema generation, then use the motivating example to illustrate how the process schema generator works.

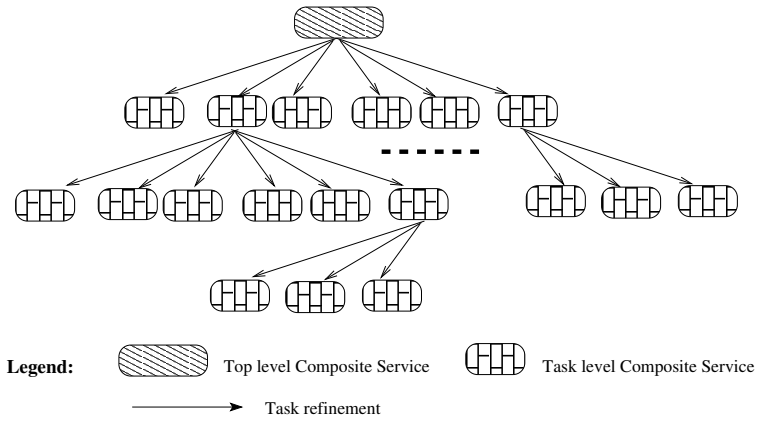



Figure 3.5: Composition Hierarchy

### Dynamic Process Schema Generation and Composite Service Execution

Service composition (both top-level and task-level) generation and execution in our framework involves three major steps (see Figure 3.6): process schema generation, process schema selection and composite service execution.

1. **Process schema generation.** We formulate the problem of process schema generation as a planning problem, which has three inputs:

   (a) A description of initial state and user's context (i.e., user profile),

   (b) A description of user's business objectives, and

   (c) A set of service composition rules (i.e., domain theory).

   It should be noted, in the input, the service composition rules are associated with the user's role that is specified in user profile. The associations between the business rules and roles are defined in organizational structure. The output

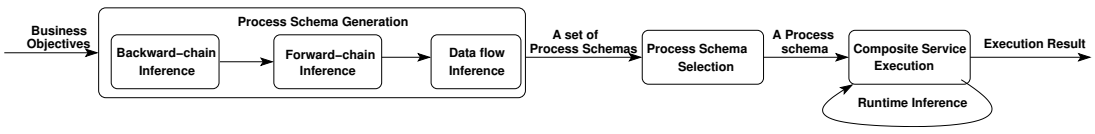



Figure 3.6: Process Schema Generation, Selection and Composition Service Execution

is a set of composite service schemas that can be instantiated and executed to achieve the business objectives. We propose a three-phase rule inference mechanism to generate composite service schemas. During the first phase, the backward-chain inference discovers all the necessary tasks (i.e., backward-chain task) that must be executed to achieve the business objective. The second phase consists of using the forward-chain inference to determine which tasks may potentially be involved as a consequence of executing tasks inferred in the previous phase. The final phase involves the data flow inference mechanism. Details about inference algorithms can be found in next section.

2. **Process schema selection.** In some cases, more than one process schema is generated by the three-phase inference mechanism. For example, if there are more than one backward-chain rules for the same task, then there may be multiple ways to achieve a business objective. In such case, the choice of process schema is based on a selection policy involving parameters such as total execution duration, and execution price, etc. By deriving an execution plan for the execution of the composite service based on the current available Web services, users are offered the opportunities to select one of the process schemas. Detailed discussion on execution planning can be found in the next chapter.

3. **Composite service execution.** In this step, the system starts executing the selected composite service. At the same time, forward-chain rules are re-applied at runtime to constantly monitor the state of the composite service execution by runtime inference. The runtime inference rules use a broad range of runtime events to drive the rule inference. This differs from the forward-chain inference performed at pre-execution time that assumes all the component services are able to complete the task and use start and termination events only.

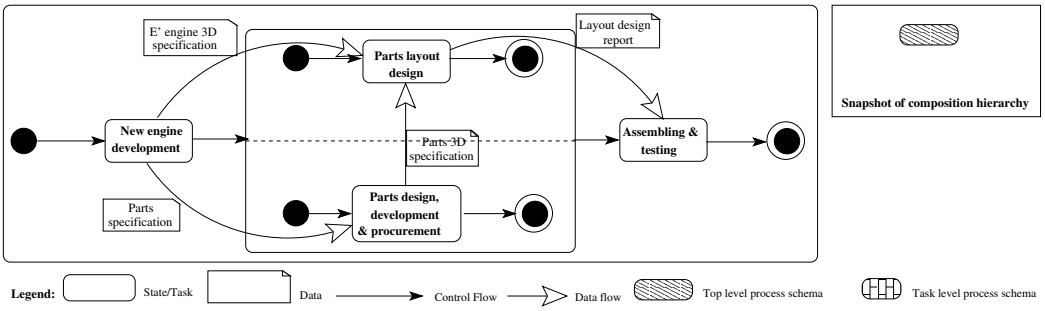



Figure 3.7: Top Level Process Schema for Replacing Engine

### Example

In this subsection, we use the running example to present a detailed scenario on how the process schema generator incrementally composes a composition hierarchy for an R&D product process. We assume that the business rule base has been built and is ready for rule inference.

- Step 1: Creating a top level process schema and starting the composite service instance. In this step, the chief engineer provides a description of his/her business objective (i.e., `replacing engine`) as input to the process schema generator. The process schema generator will locate the user's profile and appropriate business rules to generate an XML document that represents a statechart of service. The graphical presentation of the statechart is shown in Figure 3.7. After creating the top level composite service, the chief engineer will initiate the R&D product process. The task of `new engine development` will be assigned to an *engine designer*.

- Step 2: Creating task level process schema and executing the composite service instance. Assuming that an engine designer is assigned to execute the first task `new engine development` in the top level composite service, the process schema generator needs to generate a composite service for the engine designer to execute this task. Having the business objective and the

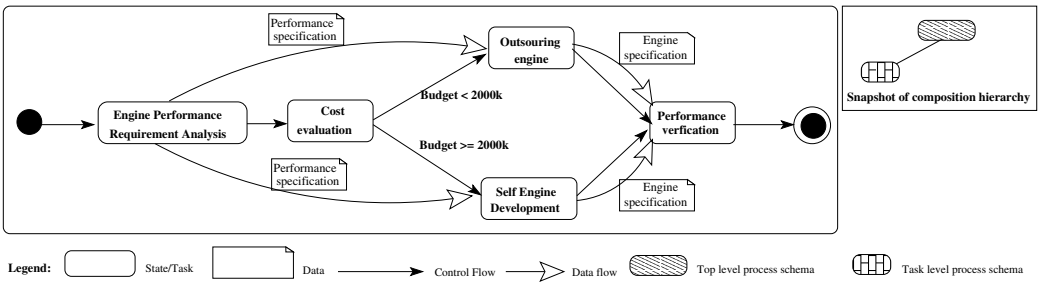



Figure 3.8: Task Level Process Schema for New Engine Development

engine designer's profile as initial context, the process schema generator can create a task level composite service schema as shown in Figure 3.8. It should be noted that, for a task in a composite service, either an elementary service is used to execute it, or the process schema generator creates a statechart to execute it. For example, for the task of `Cost Evaluation`, since there is no service composition rule for it, an elementary service is used to execute it. However, for the task of `Outsourcing Engine`, since there is a set of service composition rules, the process schema generator creates a composite service to execute it. In summary, whenever a composite task is initiated in a composite service (either top level or task level), a task level process schema will be created and that schema will be added to the composition hierarchy (see Figure 3.9).

- Step 3: Returning execution result to upper level process. When the task level composite service `outsourcing engine` is completed, the flow control

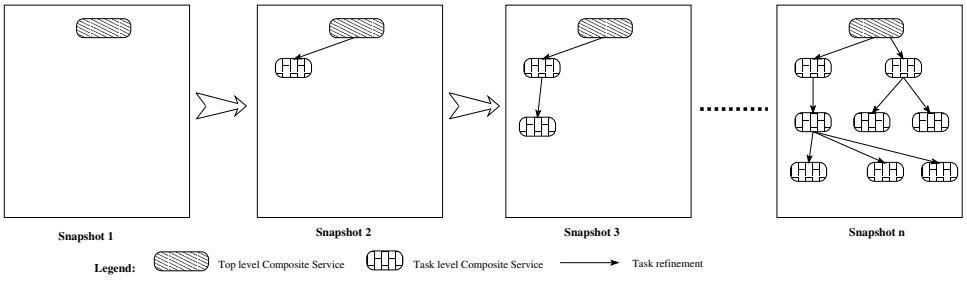



Figure 3.9: Snapshots for Composition Hierarchy

returns to the level above it, which is the `new engine development`. The execution result of the composite service will be used in the upper level composite services to execute data flows and control flows. The R&D product process is completed when the execution of the top level composite service is finished.

The above scenario shows that the system only creates the necessary composite service schemas for the R&D product process. It does not enumerate all the possible tasks, control flows and data flows. Instead of using a single, large, one-level schema to represent the whole R&D product process, we use a composition hierarchy that consists of multiple nested composite services. This modular approach allows distinct processes to be encapsulated in a composite service. This representation is more scalable and makes it easy to implement runtime modification on composite services.

To illustrate the viability of this architecture, and show how to incrementally generate composition hierarchies, we have implemented an automobile R&D application. We start the application with about 100 business rules. The application incrementally composes a composition hierarchy to manage the `replacing engine` R&D product process (see Section 3.2), where the composition hierarchy consists of 15 process schemas (120 tasks). The application shows that the system can efficiently create process schemas to support R&D processes using the rule inference approach. More details about the implementation can be found in Chapter 6.

## 3.4 Rule Inference for Process Schema Generation

As previously mentioned, the creation of process schemas are achieved through a combination of backward-chain inference, forward-chain inference, and data flow inference. In this section, we first present our approach on creating and updating business rules, then give the details on rule inference algorithms for process schema generation.

### 3.4.1 Creating and Updating Business Rules

When users add new rules or update existing rules, the system needs to check whether there are conflicts among the rules. We distinguish two different types of conflict: absolute conflict and partial conflict. For example, there are two forward-chain rules $fcr_1 = E|C|A$ and $fcr_2 = E^{'}|C^{'}|A^{'}$, if $A = \neg A^{'}$, $E = E^{'}$ and $C = C^{'}$, then $fcr_1$ and $fcr_2$ have an absolute conflict; while, if $A \subset \neg A^{'} or A \supset \neg A^{'}$, $E = E^{'}$, $C \neq C^{'}$ and $C \bigcap C^{'} \neq \emptyset$, then $fcr_1$ and $fcr_2$ have a partial conflict. In our framework, the business rule base accepts partial conflicts and rejects absolute conflicts since partial conflicts do occur in business policies. For example, the following two rules (i.e., $bcr_2$ and $fcr_2$ ) have a partial conflict since when `partType` is `engine`, $fcr_2$ drops task `orderPart` while $bcr_2$ adds task `orderPart`. Such a conflict indicates that there are business policies which do not allow for the simultaneous ordering of a new engine and the making of the same new engine.

### 3.4.2 Backward-chain Inference

In this subsection, we introduce the backward-chain inference algorithm (see Algorithm 1). When a system receives a user's objective, it must determine the tasks that need to be accomplished to achieve the business objective. This is done as follows:

1. It searches the backward-chain rules for the target task stated in the business objective. Then, starting at the target task, rules are used to infer backward-chain tasks that are components of the generated process schema. The algorithm recursively infers all the backward-chain tasks and the inference will stop when there are no new backward-chain tasks found or the initial state is reached.

2. Some backward-chain rules may have `AND` or `OR` operators in the action part.

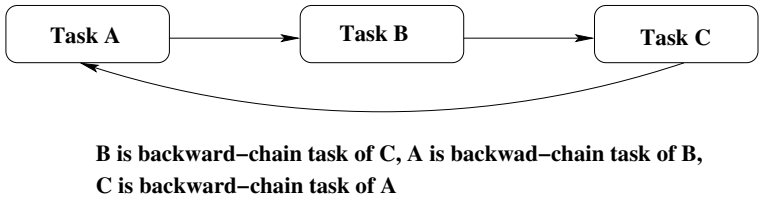



Figure 3.10: Cyclic Graph

In the case of an `AND` operator, both of the tasks are added into the process schema. In the case of an `OR` operator in the action part, more than one process schema is generated.

3. If the inferred task is already present in the process schema, the algorithm detects a cycle in the generated process schema (see example in Figure 3.10). The detection of a cycle indicates that there is a conflict among backward-chain rules, which causes the target task to be unreachable. In such a case, the process schema will not be able to achieve the business objective and thus no process schema is generated.

An example of a fragment of process schema generated by two backward-chain inference rules is illustrated in Figure 3.11. In this example, the target task is `Create New Part`. Based on the backward-chain rule $r_{b1}$, a `Clash Analysis` task and `BOM Rollup` task are added into the process schema as backward-chain tasks. Since a `Clash Analysis` task also has a backward-chain rule $r_{b2}$, a `Clash Design` task is added into the process schema as the backward-chain task of the `Clash Analysis` task.

### 3.4.3 Forward-chain Inference.

A process schema that is generated by the backward-chain inference is not complete, since new tasks may be added to the process schema at runtime depending on the result

**Algorithm 1:** Backward-chain Inference

**input** : Initial State $s$, Target Task $t$, Backward-chain rules
**output** : $\mathbb{W} = \{ W_1, W_2, ..., W_n \}$, where $W_i$ is a process schema.
**begin**

Task Pool $TP = \{t\}$
$\mathbb{W} = \emptyset$ ; $T = \{t\}$ ; $\mathbb{W} = \mathbb{W} \cup \{T\}$; $W = T$
$P$ is a set of 2-tuple $\langle TP, W \rangle$; $P = \{\langle TP, W \rangle\}$
**for** *each* $\langle TP_i, W_i \rangle$ *in* $P$ **do**
**while** $TP \neq \{s\}$ **do**
Current Task Pool is $TP$; Current schema is $W$
Get a task $t$ from $TP$ and $t \notin s$;
$TP = TP - \{t\}$
$BCR$ is a set of backward-chain rules for $t$
**if** $BCR \neq \emptyset$ **then**
**for** *each rule* $r$ *in* $BCR$ **do**
**if** *there is not* $OR$ *operator in* $r$*'s action* **then**
$t'$ is the backward-chain task in $r$
**if** $t' \notin t.BCTasks$ **then**
$t.BCTasks = t.BCTasks \cup \{t'\}$; $t.BCRules = t.BCRules \cup \{r\}$
$W = W \cup \{t'\}$; $TP = TP \cup \{t'\}$
**if** $t' \in W \wedge$ *There is a cyclic graph in* $W$ **then**
$\mathbb{W} = \mathbb{W}$ - { $W$ }; **remove**$\langle TP, W \rangle$ from $P$;
**else**
**for** *each element in* $OR$ *operator* **do**
**if** $A$ *is the first Element in* $OR$ *operator* **then**
$t'$ is the backward-chain task in $A$
**if** $t' \notin t.BCTasks$ **then**
$t.BCTasks = t.BCTasks \cup \{t'\}$
$t.BCRules = t.BCRules \cup \{r\}$
$W = W \cup \{t'\}$; $TP = TP \cup \{t'\}$
**if** $t' \in W \wedge$ *There is a cyclic graph in* $W$ **then**
$\mathbb{W} = \mathbb{W}$ - { $W$ }; **remove**$\langle TP, W \rangle$ from $P$;
**else**
$TP^{'} = TP$; $W^{'} = W$; $\mathbb{W} = \mathbb{W} \cup \{W^{'}\}$
**add** $\langle TP^{'}, W^{'} \rangle$ into $P$;
$t'$ is the backward-chain task in $A$
**if** $t' \notin t.BCTasks$ **then**
$t.BCTasks = t.BCTasks \cup \{t'\}$
$t.BCRules = t.BCRules \cup \{r\}$
$W^{'} = W^{'} \cup \{t'\}$; $TP^{'} = TP^{'} \cup \{t'\}$
**if** $t' \in W^{'} \wedge$ *There is a cyclic graph in* $W^{'}$ **then**
$\mathbb{W} = \mathbb{W}$ - { $W^{'}$ }; **remove** $\langle TP^{'}, W^{'} \rangle$ from $P$;

**end**

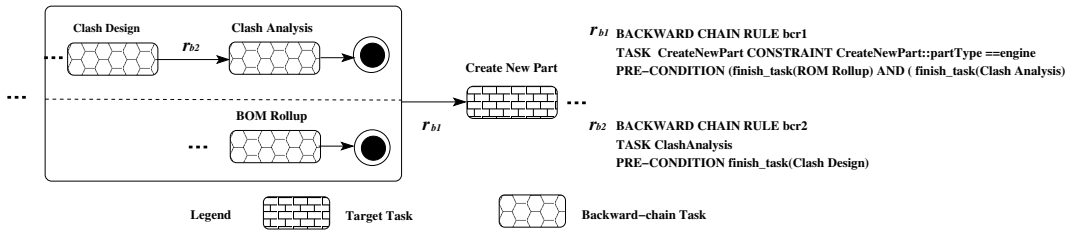



Figure 3.11: Backward-chain Inference

of the execution of certain tasks. In this subsection, we introduce a forward-chain inference algorithm (see algorithm 2) to discover additional tasks that may be added to process schemas. In the algorithm, for each backward-chain task $t$ in the process schema $W$, we assume that $R$ is the set of forward-chain rules that are triggered by the event of completing task $t$. For each rule $r$ in $R$, the condition part is represented as a *condition tree*. The condition tree is used to evaluate the condition part. The definition of the condition tree is as follows:

**Definition 1** (*Condition Tree*). $CT$ is a Condition Tree of the condition $C$ in rule $r$, if $CT$ has two types of nodes: parent nodes and leaf nodes. Parent nodes represent the logical operators in the condition $C$. Each atomic condition in $C$ is represented by two kinds of leaf nodes that are connected by an `AND` node. One of leaf node represents availability of data while the other represents the constraints on the data. □

The evaluation of a condition tree is done as follows: When the data is available, this leaf node is assigned a `true` value; otherwise it is assigned an `uncertain` value. The semantics of expression that involves an `uncertain` value is given in Table 3.4.3. Constraints on the data are expressed by a simple atomic condition, for example, `budget> $1000`. The condition tree is evaluated from leaves to parent, until the root node. The result of the root node presents the value of the condition that the tree presents.

**Example 1** (*Condition Tree*). Figure 3.12 gives the condition tree of condition $C$, where $C$ = $((Role = TestEngineer) \wedge (partName = Engine))$ $\vee \neg$ ( $(Role =$

| Operator | Expression | Result |
|---|---|---|
| AND | True $\wedge$ Uncertain | Uncertain |
| | False $\wedge$ Uncertain | False |
| | Uncertain $\wedge$ Uncertain | Uncertain |
| OR | True $\vee$ Uncertain | True |
| | False $\vee$ Uncertain | Uncertain |
| | Uncertain $\vee$ Uncertain | Uncertain |
| NOT | $\neg$ Uncertain | Uncertain |

Table 3.3: Operation Result on Uncertain

$SystemEngineer) \wedge ((Budget > \$1000)))$. $\square$

If the condition tree evaluates to `true`, then the action in a forward-chain rule is enabled. At the same time, the algorithm checks whether there are conflicts among the rules. There are two types of conflicts: a conflict between a forward-chain rule and a backward-chain rule; or a conflict between two forward-chain rules. For example (see Figure 3.14 case 1), backward-chain rule $r_b$ adds task $t_i$ into process schemas, while forward-chain rule $r_f$ drops task $t_j$ from process schemas. If the condition in $r_f$ is true, then there is a conflict between the forward-chain rule and the backward-chain rule. In another example (see Figure 3.14 case 2), forward-chain rule $r_{f1}$ adds the task $t_j$, while forward-chain rule $r_{f2}$ drops the task $t_j$. Here there is a conflict between $r_{f1}$ and $r_{f2}$. Such conflicts indicate that the target task is unreachable in the process schema. In this case, the inference procedure terminates and the process schema is abandoned. If there

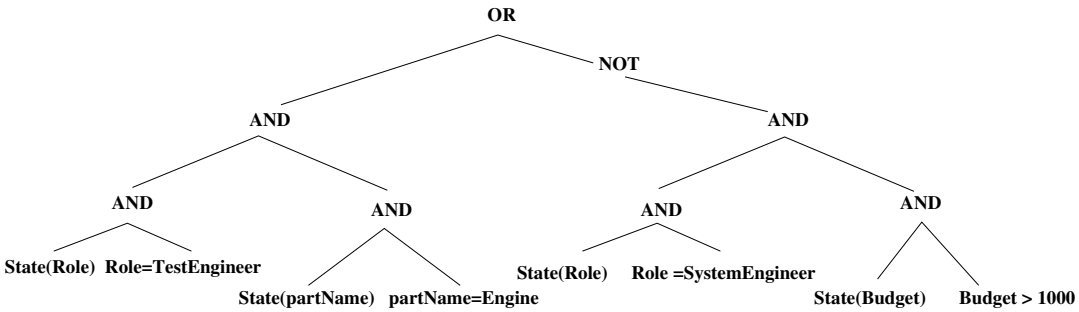



Figure 3.12: Condition Tree

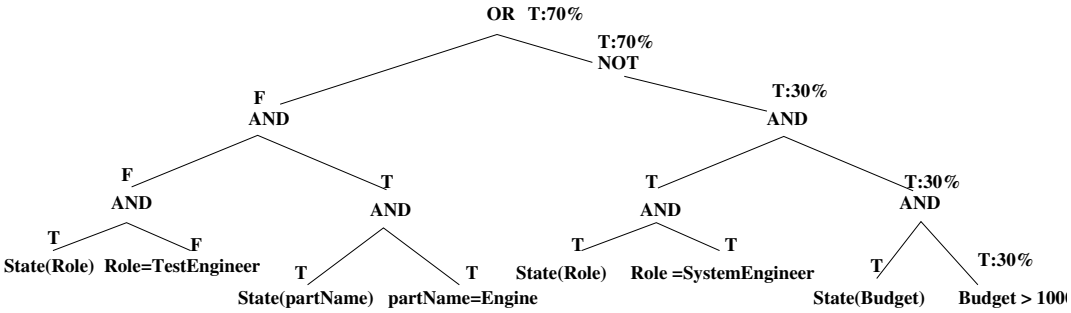



Figure 3.13: Annotated Condition Tree

exists a cyclic graph in the generated process schema, no process schema is generated and the inference procedure terminates as well.

In a situation where the condition tree is evaluated to `uncertain`, the algorithm conducts a non-deterministic inference. The basic idea behind a non-deterministic inference is to use past task execution results to determine the value of condition tree. A description of the non-deterministic inference is given as follows:

1. It identifies all the data required by the rule conditions. If we assume all the data required by the rule $r$ is $D$ and if current available data is $D_c$, then $D_u(D_u = D - D_c)$ is the set of unknown data that is required by the rule $r$.

2. For each $d \in D_u$, the algorithm searches its data source and computes its *Frequency Table* (see Definition 2) that enables the system to postulate the likelihood that the condition is true or false. For example, $d \in D_u$, if $d$ is an output of task $t$ in composite service $W$, we assume $d$ is provided by task $t$. By applying the service provider selection rules on task $t$, the algorithm locates a service provider. By querying the service provider's past execution results that are logged in the system, a frequency table is generated.

3. Assuming that for each $d$ ( $d \in D_u$), $State(d)$= `true`, the algorithm evaluates the atomic conditions based on the frequency table, which generates an annotated condition tree as illustrated in Figure 3.13. In this annotated tree,

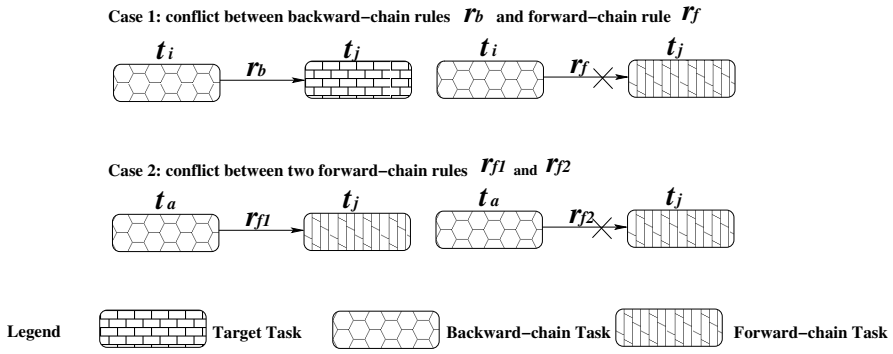



Figure 3.14: Conflict Between the Rules

the root node gives the probability that the condition tree is true.

4. If the probability of the enabling task is greater than a given threshold $\theta$ then the algorithm enables the action in the rule.

**Definition 2** (*Frequency Table*). FT is **Frequency Table** of data item $A$, if $FT = \{v_1, v_2, ...v_n\}$, where $v_i \in FT$, $v_i$ is a 2-tuple $\langle per, ran \rangle$, where $ran$ is the value range of the data item $A$, $per$ is the probability of data item $A$'s value in the range. □

**Example 2** (*Frequency Table*). FT is **Frequency Table** of data item `budget`, $FT = \{v_1, v_2, v_3\}$, where $v_1 = \langle 0.2, [1000, 1500] \rangle$, $v_2 = \langle 0.4, [1500, 2500] \rangle$, $v_3 = \langle 0.4, [2500, 3500] \rangle$ □

Figure 3.15 gives an example of forward-chain inference. In this example there is a forward-chain rule $r_{f1}$ for `Clash Analysis` task. Since the condition part is

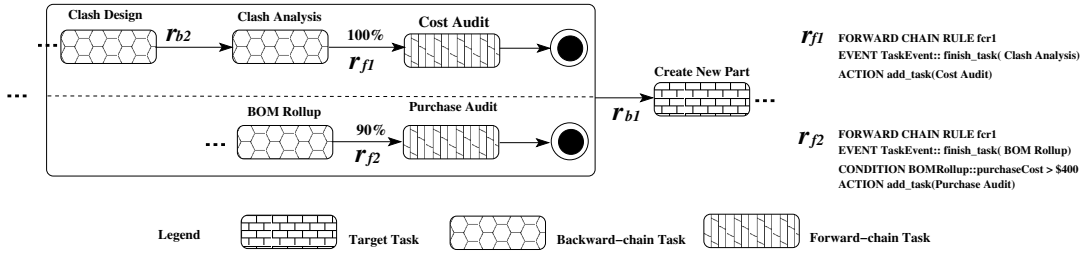



Figure 3.15: Forward-chain Inference

empty, the probability of adding `Cost Audit` task is 100%. For `BOM Rollup` task, the forward-chain rule's condition is `PurchaseCost>$400`. Based on past task execution results, if the probability of `PurchaseCost>$400` occurring is 90%, then the probability of enabling the rule is 90%. So, a `Purchase Audit` task is added into the process schema with probability of 90%.

### 3.4.4 Data Flow Inference

After the backward chain inference and the forward chain inference, tasks and control flows of a process schema $W$ are generated. The third or the final inference phase involves applying the data flow inference rules to discover the data flows among the tasks. If each task $t_i$ requires any input, data flow inference is invoked to discover the data sources. It is done as follows:

1. For each data item $d$ in task $t_i$'s input, look up the associated data flow rules in the rule base. Assuming that a set of data flow rules are associated with data item $d$ and a set of data sources $DS$ can be located. We identify the correct data source of data flows by checking the following four cases:

   - If a data source in $DS$ is the next task following task $A$ in process schema $W$, then the data source should be excluded from $DS$;
   - If a data source in $DS$ is not a task in process schema $W$, then the data source should be excluded from $DS$;
   - If a data source in $DS$ is a previous or a concurrent task of the task $t_i$ in the process schema $W$. Assume the data source is task is $t_j$, then we can denote a data flow as $DF_{t_j \rightarrow t_i}(d)$.
   - If a data source in $DS$ is a user who is assigned to a role $o$, then we can denote a data flow as $DF_{user(o) \rightarrow t_i}(D)$

**Algorithm 2:** Forward-chain Inference

**input** : A process schema $W$ generated by backward-chain reference
**output** : A process schema $W'$
**begin**

$W'$= $W$; Task Pool $TP$ = $W$

**for** *all* $t \in W$ **do**

$t.probability$ = 1

**while** $TP \neq \emptyset$ **do**

Get a task $t$ from $TP$; $TP$ = $TP$ - $\{t\}$; $R$ is set of all the forward-chain rules for $t$

**for** *each rule* $r$ *in* $R$ **do**

$CT$ is the condition tree of rule $r$; $t'$ is the task in action part of rule $r$

**if** *($CT$ is* `True` $\wedge$ *action is add_task()* $\wedge\ t' \notin t.FCTasks$*)* **then**

**if** *(($t' \in W'$) $\wedge$(There is a cyclic graph in $W'$))$\vee$(($t' \in W'.dropTask$) $\wedge$ ($t'.probability == 1$ ))* **then**

**Exit**

**if** *($t' \in W'.dropTask \wedge t'.probability != 1$)* **then**

**conflictWarning()**

$t'.probability = t.probability$

$t.RCTasks = t.RCTasks \cup \{t'\}$; $t.FCRules = t.FCRules \cup \{r\}$

$W'$ = $W' \cup \{t'\}$; $TP$ = $TP \cup \{t'\}$

**if** *($CT$ is* `True` $\wedge$ *action is drop_task())* **then**

$W.dropTask = W.dropTask \cup \{t'\}$; $t'.probability = t.probability$

**if** $t' \in W'$ **then**

There is a conflict between forward-chain rule and backward-chain rule; **Exit()**

**if** *($CT$ is* `Uncertain`*)* **then**

$D_u$ is the un-known data set required by rule $r$

**for** *each* $d \in D_u$ **do**

Locate the data source for $d$; Compute the possible value set $PV$ for $d$

Testing the condition tree $CT$ using $PV$

$per$ is the probability of enabling $CT$; $per' = per * t.probability$

**if** *( $per' \geq \theta \wedge$ action is add_task()$\wedge t' \notin t.FCTasks$)* **then**

**if** *( $t' \in W' \wedge$ There is a cyclic graph in $W'$)* **then**

**conflictWarning()**

**if** $t' \in W'.dropTask$ **then**

**conflictWarning()**

$t'.probability = per'$

$t.RCTasks = t.RCTasks \cup t'$; $t.FCRules = t.FCRules \cup \{r\}$

$W'$ = $W' \cup \{t'\}$; $TP$ = $TP \cup \{t'\}$

**if** *($per' \geq \theta \wedge$ action is drop_task())* **then**

$W.dropTask = W.dropTask \cup \{t'\}$; $t'.probability = per'$

**if** $t' \in T$ **then**

**conflictWarning()**

**end**

2. If the data flow inference can not generate a data flow for a data item $d$, and the data item $d$ is not available in process schema $W$'s input, then a notification message will be sent to the user to allow him/her to specify a data source for the data item.

3. The data flow inference also needs to detect if there are deadlocks in data flows among the concurrent tasks.

After the three-phases of rule inference, the complete process schema is generated. In some cases, a set of candidate process schemas is generated. For each process schema, the system uses service provider selection rules to select a candidate service provider for each of the tasks. Based on these service provider selections, the service composition manager can estimate the total execution time and execution price of the composite service. End users can then decide which process schema is to be executed to achieve the business objectives.

## 3.5 Related Work

There are several on-going research efforts in the workflow management area. In this section we review some related work on business rules specification in dynamic workflows, Web services standard, Web service composition, and planning in Artificial Intelligence.

In the WIDE project [25], a workflow management system is built to support distributed workflow execution. ECA rules are used to support exceptions and asynchronous behavior during the execution of distributed workflow instances. In the EvE project [39], ECA rules are used to address the problem of distributed event-based workflow execution, which is a fundamental metaphor for defining and enforcing workflow execution. Both WIDE and EvE predefine process schemas using the

ECA rules. However, our framework uses ECA rules to specify business rules and dynamically infer workflow schemas using those rules.

Some dynamic workflow management systems, such as [57] and [76], focus on the evolution of a statically defined business processes. Our framework differs from those systems in its ability to dynamically evolve process schemas via rule inference.

Decision Flow [46] focuses on providing a high level business process specification language with declarative semantics understood by users throughout an enterprise. It provides an algorithm for eager detection of eligible, needed or necessary tasks to support efficient execution of decision flow. However, a decision flow is predefined and the business rules are hard coded into the decision flow. ISEE [68] introduces events and rules to the business process model. This enables runtime modifications of the business processes. However, all the tasks in the business processes are predefined and the rules cannot be modified dynamically.

Our approach composes the business processes on demand immediately before execution and continuously adapts the process as events occur at runtime. Business rules are re-evaluated at runtime to ensure the optimal process schema is used at runtime.

Several standards that aim at providing infrastructure to support Web services composition have recently emerged including SOAP[78], WSDL[93], UDDI[81], and BPEL4WS[12]. SOAP defines an XML messaging protocol for basic service interoperability. WSDL introduces a common grammar for describing services. UDDI provides the infrastructure required to publish and discover services in a systematic way. Together, these specifications allow applications to find each other and collaborate using a loosely coupled, platform-independent protocol. BPEL4WS is the latest attempt to add a layer on top of WSDL for specifying business processes and business interaction protocols. By doing so, it extends the Web services interaction model and enables it to support business transactions. BPEL4WS and our framework are complementary to each other. The former provides a formalism for defining process schemas,

while the latter is concerned with how the process schemas may be derived from business rules. Although our prototype currently uses statechart to describe the resultant process schemas, it could easily switch to the BPEL4WS framework.

Early work in AI planning seeks to build control algorithms that enable an agent to synthesize a plan to achieve its goal [87]. Recently, AI planning has been used in information gathering and integration over the web [56, 65]. In [1], AI planning is adopted to enable the interaction of Web services, but it requires predefined activity diagrams (i.e., process schema) that enumerate all the possible interactions of Web services. Our framework uses planning algorithm for choosing the most optimal execution plan, but it does not require any predefined process schemas.

## 3.6 Summary

Our framework enables dynamic process schema generation by rule inference. The main features of our framework are:

- A set of business rule templates that model business policies.
- A rule inference approach that dynamically generates process schemas for composite services.

In order to validate the framework introduced in this chapter, the process schema generator has been implemented as a platform that provides tools for: (i) defining the business rules; and (ii) generating process schemas for composite services. Together, these tools provide an infrastructure for a new approach to composing Web services. More details on the implementation can be found in Chapter 6.

# Chapter 4

# Quality Driven Service Selection

In a process-driven service composition approach, individual Web services are federated into composite Web services whose business logic is expressed as a process model. The tasks of this process model are essentially invocations to functionalities offered by the underlying component services. Usually, several component services are able to execute a given task, with different levels of pricing and quality. In the previous chapter, we presented a framework that enables dynamic creation of process schemas. In this chapter, we assume a process schema is created for a composite service. We advocate that the selection of component services should be carried out during the execution of a composite service, rather than at design-time. In addition, this selection should consider multiple criteria (e.g., price, duration, reliability), and it should take into account global constraints as well as preferences set by the user (e.g., budget constraints). Accordingly, this chapter proposes a global planning approach to optimally select component services during the execution of a composite service. Service selection is formulated as a global optimization problem which can be solved using efficient linear programming methods. Experimental results show that this global planning approach outperforms approaches in which the component services are selected individually for each task in a composite service.

This chapter is organized as follows. Section 4.1 introduces the research issues and gives an outline of the proposed solution. Section 4.2 presents a service composition model and defines some key concepts used throughout the chapter. Section 4.3 defines the service quality criteria used for service selection and explains how the values of these quality criteria can be computed for a given service. Section 4.4 presents a local optimization based service selection approach. Section 4.5 formulates a global service selection problem and describes a linear programming method to efficiently solve it. Section 4.6 gives a brief comparison of both approaches. Finally, Section 4.7 discusses related work, and Section 4.8 summarizes this chapter.

## 4.1 Introduction

A Web service can be considered as a self-described application that uses standard Internet technologies to interact with other Web services. An example of a Web service is a SOAP-based interface to place bids in an auction house. Once deployed, services can be aggregated into *composite services*. An example of a composite service is a "Travel Planner" system that aggregates multiple *component services* for flight bookings, travel insurance, accommodation bookings, car rental, and itinerary planning, which are executed sequentially or concurrently.

The process model underlying a composite service identifies the functionalities required by the services to be composed (i.e., the *tasks* of the composite service) and their interactions (e.g., control-flow, data-flow, and transactional dependencies). Component services that are able to provide the required functionalities are then bound to the individual tasks of the composite services and invoked during each execution of the composite service.

The number of services providing a given functionality may be large and constantly changing. Consequently, approaches where the development of composite services

requires the identification at design-time of the exact services to be composed are inappropriate. The runtime selection of component services during the execution of a composite service has been put forward as an approach to address this issue [8, 23, 38]. The idea is that component services are selected by the composite service execution engine based on a set of quality criteria. However, previous approaches in this area have not identified a set of quality criteria (other than price and application-specific criteria) for selecting Web services. In addition, existing service selection approaches adopt a local selection approach, meaning that they assign a component service to individual tasks, one at a time. As a result, these approaches are not able to handle global user constraints and preferences, like for example, that the overall duration of the composite service execution should be minimized, or that a given budget constraint should be satisfied.

In this paper, we present a quality-driven approach to select component services during the execution of a composite service. The salient features of our approach are:

- A *Web services quality model*. We propose an extensible multi-dimensional Web services quality model. The dimensions of this model characterize non-functional properties that are inherent to Web services in general: *execution price*, *execution duration*, *reputation*, *reliability*, and *availability*.

- *Quality-driven service selection*. In order to overcome the limitations of local service selection outlined above, we propose a global planning approach. In this approach, quality constraints and preferences are assigned to composite services rather than to individual tasks within a composite service. Service selection is then formulated as an optimization problem and a linear programming method is used to compute optimal service execution plans for composite services. Experimental results show that the proposed service selection strategy significantly outperforms local selection strategies.

## 4.2 Web Service Composition Model

In this section, we first present the basic concepts of the adopted service composition model. We then give some definitions related to service execution planning.

### 4.2.1 Web Services

In order to participate in the composite service, service providers need to publish their Web services as advertisements to the service repository. Each published Web service's property is described using a common service ontology. There are two important elements in service descriptions which are elaborated as follows:

- **Service ontology and service class.** A Web service provider needs to specify which service ontology they use to communicate and which service classes they support. For example, a travel service provider may specify that it supports the service ontology `Trip-planning` and the service class `FlightTicketBooking`. Actually, the service ontology specifies the concepts and terminologies that Web services use to communicate and service class describes the capabilities (e.g., operations ) of Web services and how to access Web services.

- **SLAs.** A Service Level Agreement (SLA) is a contract that defines the terms and conditions of service quality that a Web service delivers to service requesters. The major component of an SLA is QoS information. There are a number of criteria that contribute to a Web service's QoS in a SLA. In our system, we consider the *execution price* and *execution duration* of each operation. Some Web service providers publish QoS information in SLAs. Other Web service providers may not publish their QoS information in their service descriptions for confidential reasons. In this case, service providers need to

provide interfaces that only authorized requesters can use to query the QoS information.

Service descriptions are stored in an ontology-based repository. A simple query language is developed that allows service requesters to discover Web services.

### 4.2.2 Composite Services and Communities

A composite Web service is an umbrella structure aggregating multiple elementary and composite Web services, which interact with each other according to a process model. As we discussed in the previous chapter, we choose to specify the process model of a composite service as a statechart [41]. A simplified statechart $W$ specifying a "Travel Planner" composite Web Service is depicted in Figure 4.1. In this composite service, a search for attractions is performed in parallel with a flight and an accommodation booking. After these searching and booking operations are completed, the distance from the hotel to the accommodation is computed, and either a car or a bike rental service is invoked. Note that when two transitions stem from the same state (in this case the state $t_5$), they denote a conditional branching, and the transitions should therefore be labelled with disjoint conditions.

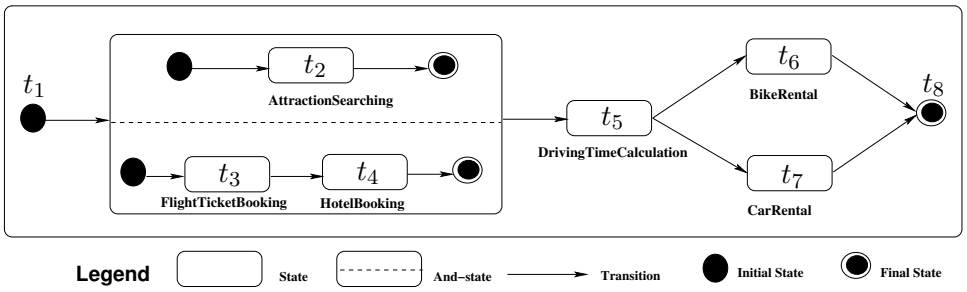



Figure 4.1: Statechart of a Composite Service "Travel Planner"

A basic state (also called task) of a statechart describing a composite service can be labelled with an invocation to either of the following:

- An elementary Web service, i.e., a service which does not transparently rely on other Web services.

- A composite Web service aggregating several other services.

- A Web service community, i.e., a collection of Web services with a common functionality but different non-functional properties (e.g., with different providers, different QoS parameters, reputation, etc.)

The concept of a Web service community addresses the issue of composing a large and changing collection of Web services. Web services in communities share a common service ontology. Service communities provide descriptions of a desired functionality (e.g., flight booking) without referring to any actual service (e.g., Qantas flight booking Web service). The set of members of a community can be fixed when the community is created, or it can be determined through a registration mechanism, thereby allowing service providers to join, quit, and reinstate the community at any time. When a community receives a request to execute an operation, this request is delegated to one of its current members. The choice of the delegation is done at execution time based on the parameters of the request, the characteristics of the members, the history of past executions, and the status of ongoing executions. Sections 4.3, 4.4 and 4.5 deal with the selection of delegations during the execution of a composite service whose states are labelled with invocations to communities.

In the following subsection, we define three concepts, namely *execution path* and *execution plan*. These concepts are used when planning the execution of a composite service.

### 4.2.3 Execution paths and plans

In this section, we define two concepts used in the rest of the paper: *execution path* and *execution plan*. To simplify the discussion, we initially assume that all the Statecharts that we deal with are acyclic. If a statechart contains cycles, a technique for "unfolding" it into an acyclic statechart needs to be applied beforehand. Details of "unfolding" process are discussed in Section 4.5.

**Definition 3** (*Execution path*). An execution path of a statechart is a sequence of states [$t_1$, $t_2$, .. $t_n$], such that $t_1$ is the initial state, $t_n$ is the final state, and for every state $t_i$ ($1 < i < n$), the following holds:

- $t_i$ is a direct successor of one of the states in [$t_1$,...,$t_{i-1}$]
- $t_i$ is not a direct successor of any of the states in [$t_{i+1}$,...,$t_n$]
- There is no state $t_j$ in [$t_1$, ..., $t_{i-1}$] such that $t_j$ and $t_i$ belong to two alternative branches of the statechart.
- If $t_i$ is the initial state of one of the concurrent regions of an AND-state *AST*, then for every other concurrent region *C* in *AST*, one of the initial states of *C* belongs to the set $\{t_1, ..., t_{i-1}, t_{i+1},...,t_n\}$. In other words, when an AND-state is entered, all the concurrent branches of this AND-state are executed.

□

This definition relies on the concept of a *direct successor* of a state. Roughly stated, a basic state $t_b$ in a statechart is a direct successor of another basic state $t_a$ if there is a sequence of adjacent transitions[1] going from $t_a$ to $t_b$ without traversing any other basic

[1]Two transitions are adjacent if the target state of one is the source state of the other.

state. In other words, the first transition in the sequence stems from $t_a$, the last transition leads to $t_b$, and all intermediate transitions stem from and lead to either compound, initial, or final states (but are not incident to a basic state).

Since it is assumed that the underlying statechart is acyclic, it is possible to represent an execution path as a Directed Acyclic Graph (DAG) as follows.

**Definition 4** (*DAG representation of an execution path*). Given an execution path [$t_1$, $t_2$, .. $t_n$] of a statechart ST, the DAG representation of this execution path is a graph obtained as follows:

- The DAG has one node for each task $\{t_1, t_2, .. t_n\}$.
- The DAG contains an edge from task $t_i$ to task $t_j$ iff $t_j$ is a direct successor of $t_i$ in the statechart ST.

□

If a statechart diagram contains conditional branchings, it has multiple execution paths. Each execution path represents a sequence of tasks to complete a composite service execution. Figure 4.2 gives an example of Statechart's execution paths. In this example, since there is one conditional branching after task $t_5$, there are two paths, called $W_{e1}$ and $W_{e2}$ respectively. In the execution path $W_{e1}$, task $t_6$ is executed after task $t_5$, while in the execution path $W_{e2}$, task $t_7$ is executed after task $t_5$.

As stated before, the basic states of a statechart describing a composite service can be labelled with invocations to communities. If this is the case, actual Web services (i.e., members of communities) need to be selected during the execution of the composite service. Hence, it is possible to execute a path in very different ways by allocating different Web services to the states in the path. The concept of execution plan defined below captures the various ways of executing a given execution path.

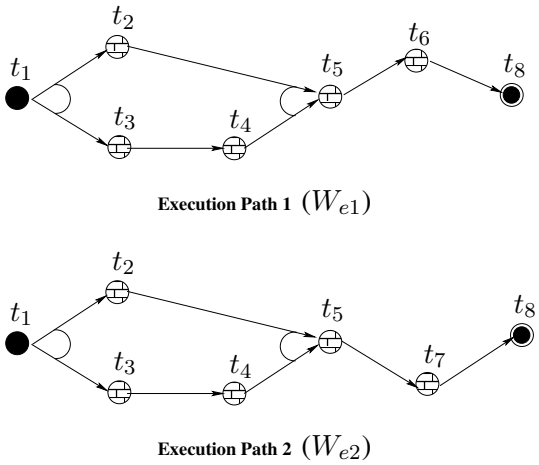



Figure 4.2: DAG Representation of the Execution Paths of the Statechart of Figure 4.1.

**Definition 5** (*Execution plan*). A set of pairs $p = \{< t_1, s_{i1} >, < t_2, s_{i2} >, \ldots, < t_N, s_{iN} >\}$ is an execution plan of an execution path $W_e$ iff:

- $\{t_1, t_2, \ldots t_N\}$ is the set of tasks in $W_e$.
- For each 2-tuple $< t_j, s_{ij} >$ in $p$, service $s_{ij}$ is assigned the execution of task $t_j$.

□

## 4.3 Web Service Quality Model

In a Web environment, multiple Web services may provide similar functionalities with different non-functional property values (e.g., different prices). In the composition model presented in the previous section, such Web services will typically be grouped together in a single community. To differentiate the members of a community during service selection, their non-functional properties need to be considered. For this purpose, we adopt a Web services quality model based on a set of quality criteria (i.e., non-functional properties) that are applicable to all Web services, for example, their

pricing and reliability. Although the adopted quality model has a limited number of criteria (for the sake of illustration), it is extensible: new criteria can be added without fundamentally altering the service selection techniques built on top of the model. In particular, it is possible to extend the quality model to integrate non-functional service characteristics such as those proposed in [74], or to integrate service QoS metrics such as those proposed by [84].

In this section, we first present the quality criteria in the context of elementary services, before turning our attention to composite services. For each criterion, we provide a definition, indicate its granularity (i.e., whether it is defined for an entire service or for individual service operations), and provide rules to compute its value for a given service.

### 4.3.1 Quality Criteria for Elementary Services

We consider five generic quality criteria quality for elementary services: (1) *execution price*, (2) *execution duration*, (3) *reputation*, (4) *reliability*, and (5) *availability*.

- **Execution price**. Given an operation $op$ of a service $s$, the execution price $q_{price}(s, op)$ is the amount of money that a service requester has to pay for executing the operation $op$. Web service providers either directly advertise the execution price of their operations, or they provide means to enquire about it.

- **Execution duration.** Given an operation $op$, of a service $s$, the execution duration $q_{du}(s, op)$ measures the expected delay in seconds between the moment when a request is sent and the moment when the results are received. The execution duration is computed using the expression $q_{du}(s, op) = T_{process}(s, op) + T_{trans}(s, op)$, meaning that the execution duration is the sum of the processing time $T_{process}(s, op)$ and the transmission time $T_{trans}(s, op)$.

Services advertise their processing time or provide methods to enquire about it. The transmission time is estimated based on past executions of the service operations, i.e., $T_{trans}(s, op) = \frac{\sum_{i=1}^{n} T_i(s,op)}{n}$, where $T_i(s, op)$ is a past observation of the transmission time, and $n$ is the number of execution times observed in the past.

- **Reliability.** The reliability $q_{rel}(s)$ of a service $s$ is the probability that a request is correctly responded within the maximum expected time frame (which is published in the Web service description). Reliability is a technical measure related to hardware and/or software configuration of Web services and the network connections between the service requesters and providers. The value of the reliability is computed from historical data about past invocations using the expression $q_{rel}(s) = N_c(s)/K$, where $N_c(s)$ is the number of times that the service $s$ has been successfully delivered within the maximum expected time frame, and $K$ is the total number of invocations.

- **Availability.** The availability $q_{av}(s)$ of a service $s$ is the probability that the service is accessible. The value of the availability of a service $s$ is computed using the following expression $q_{av}(s) = T_a(s)/\theta$, where $T_a$ is the total amount of time (in seconds) in which service $s$ is available during the last $\theta$ seconds ($\theta$ is a constant set by an administrator of the service community). The value of $\theta$ may vary depending on a particular application. For example, in applications where services are more frequently accessed (e.g., stock exchange), a small value of $\theta$ gives a more accurate approximation for the availability of services. If the service is less frequently accessed (e.g., online bookstore), using a larger $\theta$ value is more appropriate. Here, we assume that Web services send notifications to the system about their running states (i.e., available, unavailable).

- **Reputation.** The reputation $q_{rep}(s)$ of a service $s$ is a measure of its trustworthiness. It mainly depends on end user's experiences of using the service $s$. Different end users may have different opinions on the same service. The value of the reputation is defined as the average ranking given to the service by end users, i.e., $q_{rep} = \frac{\sum_{i=1}^{n} R_i}{n}$, where $R_i$ is the end user's ranking on a service's reputation, $n$ is the number of times the service has been graded. Usually, the end users are given a range to rank Web services, for example, in Amazon.com, the range is $[0, 5]$.

The overall quality vector of a service $s$ is defined by the following expression:

$$\begin{aligned} q(s) &= (q_{price}(s), q_{du}(s), q_{av}(s), q_{re}(s), q_{rep}(s)) \\ &= (q_1(s), q_2(s), q_3(s), q_4(s), q_5(s)) \end{aligned} \tag{4.1}$$

It should be noted that the method for computing or measuring the value of the quality criteria is not unique. The global planning model presented Section 4.5 is independent of these computation methods.

### 4.3.2 Quality Criteria for Composite Services

We also use above quality criteria to evaluate the QoS of composite services execution plans.

Assume that $p = \{< t_1, s_{i1} >, < t_2, s_{i2} >, \ldots, < t_N, s_{iN} >\}$ is an execution plan of a composite service CS. Refer to aggregation functions in table 4.1 for the estimation of the QoS of the composite service CS when providing service based on execution plan $p$. Brief explanation of each criterion's aggregation function is given as follows:

Table 4.1: Aggregation Functions for Execution Plan's Quality

| **Criteria** | **Aggregation function** |
|---|---|
| Execution Price | $Q_{price}(p) = \sum_{i=1}^{N} q_{price}(s_i, op_i)$ |
| Execution Duration | $Q_{du}(p) = CPA(q_{du}(s_1, op_1), ..., q_{du}(s_N, op_N))$ |
| Reputation | $Q_{rep}(p) = \frac{1}{N} \sum_{i=1}^{N} q_{rep}(s_i)$ |
| Reliability | $Q_{rel}(p) = \Pi_{i=1}^{N}(e^{q_{rel}(s_i)*z_i})$ |
| Availability | $Q_{av}(p) = \Pi_{i=1}^{N}(e^{q_{av}(s_i)*z_i})$ |

- **Execution price**: Execution price $Q_{price}(p)$ of an execution plan $p$ is a sum of every service $s_i$'s execution price $q_{price}(s_i, op_i)$.

- **Execution duration**: Execution duration $Q_{du}(p)$ of an execution plan $p$ is computed using Critical Path algorithm (CPA)[2]. The critical path algorithm considers execution path $W_e$ and its execution plan $p$ as a project digraph. It identifies the *critical path* of the project digraph. The critical path is a path from the initial state to final state in the project digraph, which has the longest total sum of Web services' execution duration. It should be noted that the sum of Web services' execution duration in the critical path is the execution duration of the execution plan $p$. The task that belongs to the critical path is a *critical task*, while the service that belongs to the critical path is a *critical service*.

[2]The Critical Path algorithm [75] is a graph algorithm which is used in project scheduling application. The description of this algorithm is outside the scope of this paper.

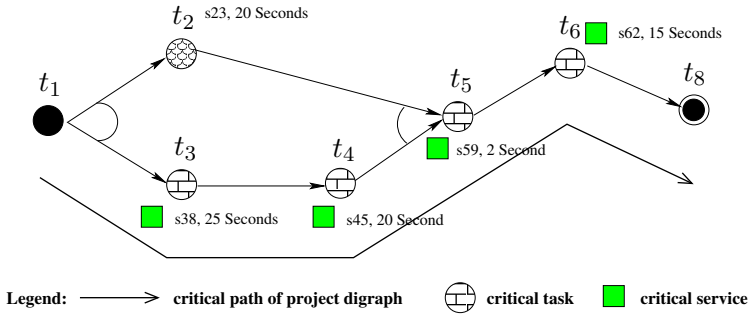



Figure 4.3: Critical Path

Figure 4.3 provides an example of critical path. In this example, the project digraph represents execution path $W_{e1}$ and its execution plan $p$, where $p$=$\{$ $< t_1, s_{13} >, < t_2, s_{28} >, < t_3, s_{35} >, < t_4, s_{49} >, < t_5, s_{52} >\}$. Each task's execution duration is given in the project digraph. There are two *project paths* in this project digraph, where project path 1 is $< t_1, t_4, t_5 >$ and project path 2 is $< t_2, t_3, t_4, t_5 >$. The total execution time of project path 1 (resp. project path 2) is 37 seconds (resp. 62 seconds). Since project path 2's total execution duration is longer than that of project path 1, then the critical path for the project digraph is project path 2. Thus, the execution plan's total execution duration is 62 seconds. Task $t_2$, $t_3$, $t_4$ and $t_5$ are critical tasks. Services $s_{28}$, $s_{35}$, $s_{49}$ and $s_{52}$ are critical services.

- **Reputation**: Reputation $Q_{rep}(p)$ of an execution plan $p$ is the average of each service $s_i$'s reputation $q_{rep}(s_i)$ in the execution plan $p$.

- **Reliability**: Reliability $Q_{rel}(p)$ of an execution plan $p$ is a product of $e^{q_{rel}(s_i)*z_i}$. In the aggregation function, $z_i$ is an integer variable which has value 1 or 0: Value 1 indicates that service $s_i$ is a critical service in the execution plan $p$; Value 0 indicates that service $s_i$ is not a critical service. If $z_i = 0$, i.e., service $s_i$ is not a critical service, then $e^{q_{rel}(s_i)*z_i} = 1$. Thus, the reliability of service $s_i$ will not affect the value of the execution plan's reliability.

- **Availability**: Availability $Q_{av}(p)$ of an execution plan $p$ is a product of $e^{q_{av}(s_i)*z_i}$, where $q_{av}(s_i)$ is service $s_i$'s availability.

Using above aggregation functions, the quality vector of a composite service's execution plan can be computed using the following expression:

$$\begin{aligned} Q(p) &= (Q_{price}(p), Q_{du}(p), Q_{av}(p), Q_{re}(p), Q_{rep}(p)) \\ &= (Q_1(p), Q_2(p), Q_3(p), Q_4(p), Q_5(p)) \end{aligned} \tag{4.2}$$

## 4.4 Service Selection by Local Optimization

In this approach, the execution of a composite service is done by selecting a component Web service to execute each task, without taking into account the other tasks. The procedure of executing a task is: (1) prompting the end user to provide the input parameters' value if it is necessary; (2) locating Web services that can execute the task; (3) selecting a Web service; (4) assigning the task to the selected Web service.

The selection of a Web service to execute a task is a two-phase process. In the first phase, the system collects candidate Web services' QoS information. In the second phase, the system selects a Web service based on QoS information. After collecting the QoS information, each Web service's quality vector $q(s)$ can be computed.

Once all the candidate Web services' quality vectors are computed, the system can select a Web service to execute the task. Here, we adopt the Multiple Criteria Decision Making (MCDM)[5] approach to select a Web service. Assume that for a task $t_j$ in a composite service, there is a set of candidate Web services $S_j$ ($S_j = \{s_{1j}, s_{2j}, ..., s_{nj}\}$) that can be used to execute the task. Using all the candidate Web services' quality vectors, we can obtain the following matrix $\mathbf{Q}$. Each row in $\mathbf{Q}$ represents a Web service $s_{ij}$, while each column represents one of the quality dimensions.

$$\mathbf{Q} = \begin{pmatrix} q_{1,1} & q_{1,2} & \cdots & q_{1,5} \\ q_{2,1} & q_{2,2} & \cdots & q_{2,5} \\ \vdots & \vdots & \vdots & \vdots \\ q_{n,1} & q_{n,2} & \cdots & q_{n,5} \end{pmatrix} \tag{4.3}$$

A Simple Additive Weighting (SAW)[15] technique is used to select an optimal Web service. Basically, there are two phases in applying SAW:

- Scaling Phase

  Some of the criteria used could be negative, i.e., the higher the value , the

lower the quality . This includes criteria such as execution time and execution price. Other criteria are positive, i.e., the higher the value, the higher the quality. For negative criteria, values are scaled according to Equation 4.4. For positive criteria, values are scaled according to Equation 4.5.

$$v_{i,j} = \begin{cases} \frac{q_j^{max} - q_{i,j}}{q_j^{max} - q_j^{min}} & \text{if } q_j^{max} - q_j^{min} \neq 0 \\ 1 & \text{if } q_j^{max} - q_j^{min} = 0 \end{cases} \tag{4.4}$$

$$v_{i,j} = \begin{cases} \frac{q_{i,j} - q_j^{min}}{q_j^{max} - q_j^{min}} & \text{if } q_j^{max} - q_j^{min} \neq 0 \\ 1 & \text{if } q_j^{max} - q_j^{min} = 0 \end{cases} \tag{4.5}$$

In the above equations, $q_j^{max}$ is maximal value of a quality criteria in matrix $\mathbf{Q}$, i.e., $q_j^{max} = Max(q_{i,j}), 1 \leq i \leq n$. While $q_j^{max}$ is minimal value of a quality criteria in matrix $\mathbf{Q}$, i.e., $q_j^{min} = Min(q_{i,j}), 1 \leq i \leq n$. Applying these two equations on $\mathbf{Q}$, we get matrix $\mathbf{Q}^{'}$ which is shown below:

$$\mathbf{Q}^{'} = \begin{pmatrix} v_{1,1} & v_{1,2} & \dots & v_{1,5} \\ v_{2,1} & v_{2,2} & \dots & v_{2,5} \\ \vdots & \vdots & \vdots & \vdots \\ v_{n,1} & v_{n,2} & \dots & v_{n,5} \end{pmatrix} \tag{4.6}$$

**Example.** Assume that there are eight Web services in $S$. Also assume that their value on service reputation are: $\mathbf{Q}_5$=($q_{1,5}$, $q_{2,5}$, $q_{3,5}$, $q_{4,5}$, $q_{5,5}$, $q_{6,5}$, $q_{7,5}$, $q_{8,5}$) = (7.5,8.4,9,8.3,8.7,9.1,9.4,9.2,9.5). Since reputation is a positive criteria, Equation 4.5 is used for scaling. In this example, $q_5^{max}$ = 9.5, $q_5^{min}$ = 7.5, this resulted in: $\mathbf{Q}_5^{'}$ = (0, 0.55, 0.45, 0.75, 0.4,0.6, 0.8, 0.95,1)

- Weighting Phase

  The following formula is used to compute the overall quality score for each

Web service:

$$Score(s_i) = \sum_{j=1}^{5} (v_{i,j} * W_j) \tag{4.7}$$

where $W_j \in [0, 1]$ and $\sum_{j=1}^{5} W_j = 1$. $W_j$ represents the weight of criterion $j$. End users can give their preference on QoS (i.e., balance the impact of the different criteria) to select a desired Web service by adjusting the value of $W_j$. The system will choose the Web service which has the maximal value of $Score(s_i)$. If there is more than one Web services with the same maximal value of $Score(s_i)$, then a Web service will be selected from them randomly. Formula 4.7 is a local optimization selection policy to select an optimal service for a task.

## 4.5 Service Selection by Global Planning

As mentioned before, in existing approaches, the selection of a component service to execute a task is determined without taking into account the constraints with other tasks of the composite service [8, 23, 38]. More precisely, in our previous section, service selection is done at each service community locally. Although service selection is locally optimized, the global quality constraints may not be satisfied. For example, a global constraint, such as the composite services' execution price being less than 500 dollars, can not be enforced. In this section, we present a global planning based approach for Web services selection. We first present an approach of selecting an optimal execution plan for a composite service, then present a novel linear programming based method for selecting an optimal execution plan.

### Selecting an Optimal Execution Plan

The basic idea of global planning is the same as query optimization in database management systems. Several plans are identified before each execution of a composite service, and the optimal plan is selected. The foregoing discussion makes it clear that a statechart has multiple execution paths and each execution path has its own set of execution plans if the statechart contains conditional branchings. In this subsection, we assume that the statechart does not contain any conditional branchings and has only one execution path. We will discuss the case where a statechart has multiple execution paths in Section 4.5.

We also assume that for each task $t_j$, there is a set of candidate Web services $S_j$ that are available to execute task $t_j$. Associated with each Web service $s_{ij}$ is a quality vector (see equation 4.1). Based on the available Web services, by selecting a Web service for each task in an execution path, the global planner will generate a set of execution plans $P$ :

$$P = \{p_1, p_2, ..., p_n\} \tag{4.8}$$

$n$ is the number of execution plans. After a set of execution plans is generated, the system needs to select an optimal execution plan. When selecting the execution plan, instead of computing the quality vector of a particular Web service, each execution plan's global service quality vector needs to be computed.

The selection of an execution plan also uses MCDM approach. Once the quality vector for each execution plan is derived, by accumulating all the execution plans' quality vectors, we obtain matrix $\mathbb{Q}$, where each row represents an execution plan's quality

vector.

$$\mathbb{Q} = \begin{pmatrix} Q_{1,1} & Q_{1,2} & \dots & Q_{1,5} \\ Q_{2,1} & Q_{2,2} & \dots & Q_{2,5} \\ \vdots & \vdots & \vdots & \vdots \\ Q_{n,1} & Q_{n,2} & \dots & Q_{n,5} \end{pmatrix} \tag{4.9}$$

Again, a SAW technique is used to select an optimal execution plan. The two phases of applying SAW to select an optimal execution plan are:

- Scaling Phase

  As in the previous section, we need equations to scale the value of each quality criterion. For negative criteria, values are scaled according to Equation 4.10. For positive criteria, values are scaled according to Equation 4.11.

$$V_{i,j} = \begin{cases} \frac{Q_j^{max} - Q_{i,j}}{Q_j^{max} - Q_j^{min}} & \text{if } Q_j^{max} - Q_j^{min} \neq 0 \\ 1 & \text{if } Q_j^{max} - Q_j^{min} = 0 \end{cases} \quad j = 1, 2 \tag{4.10}$$

$$V_{i,j} = \begin{cases} \frac{Q_{i,j} - Q_j^{min}}{Q_j^{max} - Q_j^{min}} & \text{if } Q_j^{max} - Q_j^{min} \neq 0 \\ 1 & \text{if } Q_j^{max} - Q_j^{min} = 0 \end{cases} \quad j = 3, 4, 5 \tag{4.11}$$

  In the above equations, $Q_j^{max}$ is maximal value of a quality criterion in matrix $\mathbb{Q}$, i.e., $Q_j^{max} = Max(Q_{i,j}), 1 \leq i \leq n$. $Q_j^{max}$ is the minimal value of a quality criterion in matrix $\mathbb{Q}$, i.e., $Q_j^{min} = Min(Q_{i,j}), 1 \leq i \leq n$.

  In fact, we can compute $Q_j^{max}$ and $Q_j^{min}$ without generating all possible execution plans. For example, in order to compute the maximum execution price (i.e., $Q_{price}^{max}$) of all the execution plans, we select the most expensive Web service for each task and sum up all these execution prices to compute $Q_{price}^{max}$. In order to compute the minimum execution duration (i.e., $Q_{du}^{min}$) of all the

execution plans, we select the Web service that has the shortest execution duration for each task and use CPA to compute $Q_{du}^{min}$. The computation cost of $Q_j^{max}$ and $Q_j^{min}$ is polynomial.

After the scaling phase, we obtain the following matrix $\mathbb{Q}^{'}$:

$$\mathbb{Q}^{'} = \begin{pmatrix} V_{1,1} & V_{1,2} & \dots & V_{1,5} \\ V_{2,1} & V_{2,2} & \dots & V_{2,5} \\ \vdots & \vdots & \vdots & \vdots \\ V_{n,1} & V_{n,2} & \dots & V_{n,5} \end{pmatrix} \tag{4.12}$$

- Weighting Phase

  The following formula is used to compute the overall quality score for each execution plan:

$$Score(p_i) = \sum_{j=1}^{5} (V_{i,j} * W_j) \tag{4.13}$$

  where $W_j \in [0, 1]$ and $\sum_{j=1}^{5} W_j = 1$. $W_j$ represents the weight of each criterion. End users can give their preference on QoS (i.e., balance the impact of the different criteria) to select a desired execution plan by adjusting the value of $W_j$. The global planner will choose the execution path which has the maximal value of $Score(p_i)$ (i.e., $max(Score(p_i))$). If there is more than one execution plan which has the same maximal value of $Score(p_i)$, then an execution plan will be selected from them randomly. Formula 4.13 is a global optimization selection policy for a composite service.

### Handling Multiple Execution Paths

In Section 4.5, we assume that the statechart only has one execution path. In this subsection, we discuss the case where Statecharts have multiple execution paths. Assume

that a statechart has $n$ execution paths. For each execution path, an optimal execution plan can be selected. So, the global planner has $n$ selected execution plans. Since each selected optimal execution plan only covers a subset of the entire statechart, then the global planner needs to aggregate these $n$ execution plans into an overall execution plan that covers all the tasks in the statechart. This overall execution plan will be used to execute the statechart. For example, for the `Travel Planner` statechart W (see Figure 4.1), there are two execution paths $W_{e1}$ and $W_{e2}$. The optimal execution plans $p_1$ and $p_2$ of these two execution paths are selected. From Figure 4.2, it can be seen that both execution paths $W_{e1}$ and $W_{e2}$ are subsets of $W$. Thus neither $p_1$ nor $p_2$ covers all tasks in $W$. Since the global planner conducts planning before the execution, it does not know which execution path will eventually be used for the composite service. Therefore it needs to aggregate $p_1$ and $p_2$ into an overall execution plan which covers all the tasks in $W$.

Assume that statechart $W$ has $k$ tasks (i.e., $t_1$, $t_2$, ..., $t_k$) and $n$ execution paths (i.e., $W_{e1}$, $W_{e2}$,..., $W_{en}$). For each execution path, the global planner selects an optimal execution plan. Consequently, we obtain $n$ optimal execution plans (i.e., $p_1, p_2, ..., p_n$) for these execution paths. The global planner adopts the following approach to aggregate multiple execution plans into an overall execution plan.

1. Given a task $t_i$, if $t_i$ only belongs to one execution path (e.g., $W_{ej}$), then the global planner selects $W_{ej}$'s execution plan $p_j$ to execute the task $t_i$. We denote this as $<t_i, p_j>$. For example, in `Travel Planner` statechart, task $t_6$ (i.e., `BikeRental` ) only belongs to execution path $W_{e2}$. In this case, $W_{e2}$'s execution plan $p_2$ is used to execute $t_6$, i.e., $<t_6, p_2>$.

2. Given a task $t_i$, if $t_i$ belongs to more than one execution paths (e.g., $W_{ej}$, $W_{ej+1}$, ..., $W_{em}$), then there is a set of execution plans (i.e., $p_j$, $p_{j+1}$, ..., $p_m$) that can be used to execute $W_{si}$. In this case, the global planner needs to

select one of the execution plans from $p_j$, $p_{j+1}$, ... , $p_m$. The selection can be done by identifying the *hot path* for task $t_i$. Here, the hot path of a task $t_i$ is defined as the execution path that has been most frequently used to execute the task $t_i$ in past instances of the composite service. For example, in `Travel Planner` statechart, task $t_2$ (i.e., `AttractionSearching`) belongs to both execution path $W_{e1}$ and $W_{e2}$. Assume that the statechart $W$ has been used to execute the composite service 25 times. Also assume that, in 20 times the execution of the composite service follows the execution path $W_{e1}$; while in 5 times, the execution of the composite service follows the execution path $W_{e2}$. This indicates that the execution path $W_{e1}$ has been more frequently used to execute task $t_2$ (i.e., $W_{e1}$ is the hot path for $t_2$). Thus, $W_{e1}$'s execution plan $p_1$ is used to execute $t_2$, i.e., $<t_2, p_1>$.

The system keeps composite service execution traces in an execution history [36]. This allows the global planner to identify the hot path for each task.

### Unfolding Cyclic Statecharts

Hitherto, we have assumed that the composite service Statecharts are acyclic. If a statechart contains cycles, these need to be "unfolded" so that the resulting statechart has a finite number of execution paths. The method used to unfold the cycles of a Statechart is to examine the logs of past executions in order to determine the maximum number of times that each cycle is taken. The states appearing between the beginning and the end of the cycle are then cloned as many times as the cycle (i.e., the transition that represents the cycle) is taken.

This unfolding method works only if the "beginning" and the "end" of each cycle in the statechart can be clearly identified. It does not work, for example, if the transitions causing the cycle are located in two different conditional branches as in Figure 4.4. In

this case, it is not possible to determine which is the first state and which is the last state in the cycle (is it $t_2$ or $t_3$?). An equivalent statechart which can be unfolded using the above method is shown in Figure 4.5. In this equivalent statechart, the transition causing the cycle does not cross the boundaries of any conditional branch.

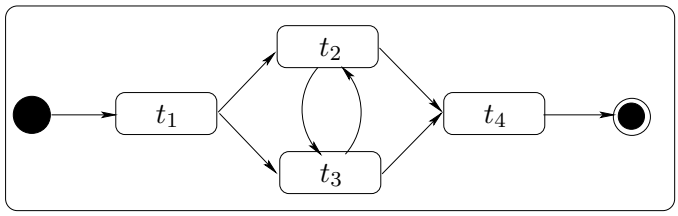



Figure 4.4: "Unfoldable" Statechart

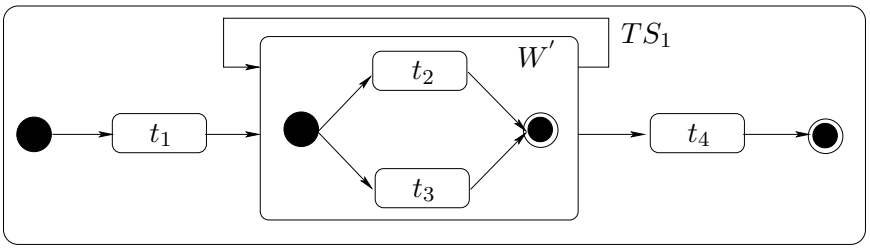



Figure 4.5: Foldable Statechart Equivalent to That in Figure 4.4

It can be proved that any arbitrary statechart can be transformed into an equivalent statechart in which the cycles do not cross the boundaries of conditional branches (as illustrated above). This proof is similar to that of the theorem stating that any program written using "goto", can be transformed into an equivalent program which only uses structured loops (i.e., "while" loops)[3].

Now we use the statechart in Figure 4.5 as an example to present output of the "unfolding" process. Assume that in the logs, the transition $TN_1$ is taken in 1, 2, and 3 times in different instances of the composite service. So, the state $W'$ is cloned three times in the acyclic statechart (see Figure 4.6). From this acycle statechart, we can generate all the possible execution paths. Again, based on execution logs, we can identify a hot path for each task in the statechart.

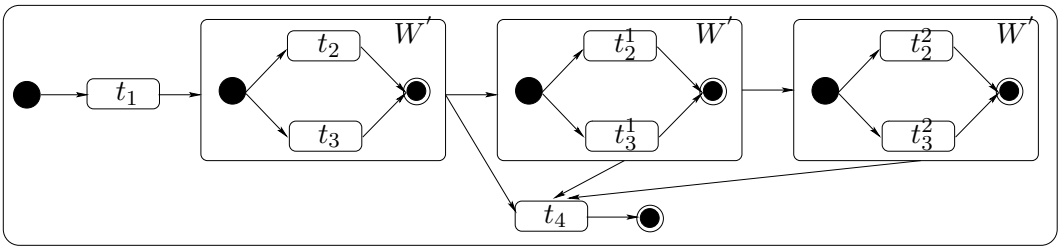



Figure 4.6: Acyclic Statechart Derived from That in Figure 6.

### Linear Programming Solution

The approach of selecting an optimal execution plan in previous section requires the generation of all possible execution plans. Assume that there are $N$ tasks in a statechart and there are $M$ potential Web services for each task. The total number of execution plans is $M^N$. The computation cost of selecting an optimal execution plan is $O(M^N)$. Such an approach is impractical for large scale composite services, where both the number of tasks in the composite services and number of candidate Web services in communities are large. For example, assume that a composite service has one execution path and $10$ tasks, and for each task, there are $10$ candidate Web services. Then the total number of execution plans is $10^{10}$. It is very costly to generate all these $10^{10}$ plans and select an optimal one. In this subsection, we present a method based on linear programming (LP) [51], which can be used to select an optimal execution plan without generating all the possible execution plans.

There are three inputs in LP: *variables*, an *objective function* and *constraints* on the variables, where both the objective function and constraints must be linear. LP attempts to maximize or minimize the value of the objective function by adjusting the values of variables based on the constraints. The output of LP is the maximum (or minimum) value of the objective function and the values of variables. In order to use LP to select an optimal execution plan, we model the selection of an optimal execution plan as an LP problem. Following, we discuss the details about these three inputs of LP according

to selection of an optimal execution plan.

For variables, we use the integer variables $y_{ij}$ to represent the selection of Web services $s_{ij}$. $y_{ij}$'s value can be 1 or 0: Value 1 indicates selecting Web service $s_{ij}$ and 0 indicates otherwise. For the objective function, since we use MADM and SAW to select an execution plan, based on equation 4.10,4.11 and 4.13, we obtain the following objective function:

$$Max\left(\sum_{l=1}^{2}\left(\frac{Q_l^{max}-Q_{i,l}}{Q_l^{max}-Q_l^{min}}*W_l\right)+\sum_{l=3}^{5}\left(\frac{Q_{i,l}-Q_l^{min}}{Q_l^{max}-Q_l^{min}}*W_l\right)\right) \tag{4.14}$$

where $W_l \in [0,1]$ and $\sum_{j=1}^{5} W_j = 1$. $W_l$ is the weight assigned to the quality criterion. Now, the LP has variables and the objective function. In the following subsections, we discuss constraints on variables.

- Constraints on Execution Duration and Execution Price

  In this subsection, we consider constraints on the execution plan's execution duration and execution price. Assume that $A$ is the set of all tasks to be executed in the statechart. For each task $t_j$, there is a set of Web services $S_j$. For each task $t_j$, we only select one Web service, since $y_{ij}$ ($y_{ij} = 0$ or 1) represents the selection of Web services $s_{ij}$, then the constraint on selection of Web services is:

$$\sum_{i \in S_j} y_{ij} = 1,\ \forall j \in A \tag{4.15}$$

  For example, there are 100 potential Web services that can execute task $j$, since only one of them will be selected to execute the task $j$, then we have $\sum_{i=1}^{100} y_{ij} = 1$.

Assume that variable $x_j$ represents the earliest start time of task $t_j$, variable $p_j$ represents the execution duration for task $j$, and variable $p_{ij}$ represents the execution duration for task $t_j$ by service $s_{ij}$. We use the notation $t_j \rightarrow t_k$ to denote that task $t_k$ is task $t_j$'s direct successor task. We have the following constraints:

$$\sum_{i \in S_j} p_{ij}\ y_{ij} = p_j, \forall j \in A \tag{4.16}$$

$$x_k - (p_j + x_j) \geq 0, \forall t_j \rightarrow t_k, j, k \in A \tag{4.17}$$

$$Q_{du} - (x_j + p_j) \geq 0, \forall j \in A \tag{4.18}$$

Constraint 4.16 indicates that the execution duration of a given task $t_j$ must be the execution duration of a Web service in $A$, since only one of the Web services in $A$ will be selected to execute task $t_j$. Constraint 4.17 indicates that if task $t_k$ is task $t_j$'s direct successor task, then the execution of task $t_k$ must start after task $t_j$ has been completed. Constraint 4.18 indicates that the execution of composite services is finished only when all the tasks in composite services are finished.

Assume that $z_{ij}$ is an integer variable that has value $1$ or $0$: $1$ indicates that Web service $s_{ij}$ is a critical service and $0$ indicates otherwise. The relationship between the execution duration of an execution plan and the duration of the critical services of the plan is captured by the following equation.

$$Q_{du} = \sum_{j \in A} \sum_{i \in S_j} p_{ij} z_{ij} \tag{4.19}$$

For execution price, assume that variable $c_{ij}$ represents the execution price of Web service $s_{ij}$, then we have the following constraint on total execution price of composition service:

$$Q_{price} = \sum_{j \in A} \sum_{j \in S_j} c_{ij}\ y_{ij} \tag{4.20}$$

An alternative of constraint 4.20 is given as follows:

$$\sum_{j \in A} \sum_{j \in S_j} c_{ij} y_{ij} \leq B, B > 0 \tag{4.21}$$

where $B$ is the budget constraint that is given by users. This constraint indicates that the whole composite service's execution price can not be greater than $B$. By introducing a budget constraint the above problem needs to be explicitly solved as an integer programming. This problem is a special case of the knapsack problem and hence NP-hard [64]. Notice that constraints on other criteria can be easily incorporated into LP if the aggregation function is a linear function. For example, assume that variable $r_{ij}$ represents the reputation of Web service $s_{ij}$, we can have the following constraint on execution plan's reputation:

$$Q_{rep} = \sum_{j \in A} \sum_{j \in S_j} r_{ij} y_{ij} \tag{4.22}$$

- Constraints on Reliability and Availability

  In this subsection, we consider constraints on criteria where the aggregation function is not a linear function. Among criteria that are used to select Web services, both availability and reliability's aggregation functions are nonlinear (See table 4.1). We can linearize them using a logarithm function as shown below. Assume that variable $a_{ij}$ represents the reliability of Web service $s_{ij}$. Since $z_{ij}$ indicates whether Web service $s_{ij}$ is a critical service or not, the reliability of the execution plan is:

$$Q_{rel} = \Pi_{j \in A} \left( \sum_{j \in S_j} e^{a_{ij} z_{ij}} \right)$$

By applying the logarithm function $ln$, we obtain:

$$ln(Q_{rel}) = \sum_{j \in A} ln \left( \sum_{j \in S_j} e^{a_{ij} z_{ij}} \right)$$

Since $\sum_{j \in A} z_{ij} = 1$ and $z_{ij} = 0$ or $1$, we obtain:

$$ln(Q_{rel}) = \sum_{j \in A} \left( \sum_{j \in S_j} a_{ij} z_{ij} \right)$$

Let $Q'_{rel} = ln(Q_{rel})$, we have the following constraint on execution plan's reliability:

$$Q'_{rel} = \sum_{j \in A} \sum_{j \in S_j} a_{ij} z_{ij} \tag{4.23}$$

Assume that $b_{ij}$ represents the availability of the Web service $s_{ij}$. Similar to reliability, for availability, we have the following constraint:

$$Q'_{av} = \sum_{j \in A} \sum_{i \in S_j} b_{ij} z_{ij} \tag{4.24}$$

where $Q'_{av} = ln(Q_{av})$.

Criteria that can be added into LP are not limited to what we defined in Section 4.3. Other criteria can also be added into LP once the aggregation functions are given.

- Constraints on Uncertainty of Execution Duration

  In the previous sections, we assumed that the execution duration $p_{ij}$ of Web services are deterministic. In fact, Web service $s_{ij}$'s execution duration $p_{ij}$ might be uncertain. For example, given operation $op$, Web service $s$ advertises that the execution duration is 5 seconds. But actual execution duration may be 4.5, 4.6 or 5.2 seconds. Assume that $p_{ij}$ is a normal distribution, where the normal distribution has a probability function given as follows:

$$f(x) = \frac{1}{\sqrt{2\pi}\sigma} exp\Big[ -\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2\Big], -\infty < x < \infty$$

The normal distribution is a 2 parameter distribution with a mean (i.e., $\mu$) and standard deviation (i.e., $\sigma$), where

$$\mu = \frac{1}{n}\sum_{i=1}^{n} x_i \tag{4.25}$$

$$\sigma^2 = \frac{1}{n-1}\sum_{i=1}^{n}(x_i - \mu)^2 \tag{4.26}$$

By applying formula 4.25 and 4.26 on a Web service $s_{ij}$ using its past execution results, we can have $\mu_{ij}$ and $\sigma_{ij}$ for the Web service $s_{ij}$. Since $\sum_{i\in A}\sum_{i\in S_j} p_{ij} z_{ij} = Q_{du}$, the total execution duration $Q_{du}$ must be normal distribution[3] and its deviation $\sigma_{du}$ is given as follows:

$$\sigma_{du}^2 = \sum_{i\in A}\sum_{i\in S_j} \sigma_{ij}^2 z_{ij} \tag{4.27}$$

So, if we consider the total execution time duration's deviation in the LP, we can have the following objective function:

$$Max\left(\sum_{l=0}^{2}\left(\frac{Q_l^{max} - Q_{i,l}}{Q_l^{max} - Q_l^{min}} * W_l\right) + \sum_{l=3}^{5}\left(\frac{Q_{i,l} - Q_l^{min}}{Q_l^{max} - Q_l^{min}} * W_l\right)\right) \tag{4.28}$$

where $Q_0 = \sigma_{du}^2$ and $W_0 \in [0, 1]$, is the weight assigned to the total execution time duration's deviation.

Now, we have all the three inputs for LP. The output of the LP is values of $s_{ij}$ that indicate the selection of Web services. These selected Web services compose an optimal execution plan.

[3]Detail proof can be found in [86]

## 4.6 Evaluation of Two Service Selection Approaches

When executing composite services, an instance is created based on a statechart first. As we discussed in the previous section, during the execution of the composite services, there are two approaches to select the Web services, namely local optimization and global planning approach. In this section, we first outline the metrics that can be used to evaluate the effectiveness of these two service composition approaches and then compare them.

### 4.6.1 Evaluation Metrics

In our framework, the metrics that are used to evaluate the Web services composition approach are *QoS of composite service* and *system cost*.

- **QoS of Composite Service.** Although we seek to reduce the system cost to execute composite services, we must nevertheless ensure that the approach doesn't degrade the user experience. QoS of composite services can be measured in a number of ways; we consider the following dimensions:
  - **Total execution price and execution duration.** In most of the cases, invocation of Web services will incur payment. Total execution price is the total payment in dollars incurred by end users during the execution of a composite service. Another important quality dimension is total execution duration, i.e., how long must end users wait for the whole composite service's execution result. Although there is a trade-off between the price and duration metrics, it is very important that the system is able to find an optimal combination that can satisfy end users' preferences.
  - **Satisfaction** of user's requirements. Although good quality Web service composition requires minimal execution price and duration, the satisfac-

tion of end users' constraints are important aspects too. Basically, there are two kinds of constraints: constraints on a single task and constraints on multiple tasks. The system's ability to accept both kinds of end users' constraints is the key to satisfying user's requirements.

- **System Cost.** When the system receives a request to execute a composite service from an end user, the system use process resources (i.e., *computation cost*) to search the service repository to locate the potential Web services that can be used to execute tasks. The system also needs to spend computation costs to select Web services and plan execution of a composite service on behalf of the end user. Similarly, the system also consumes bandwidth resource (i.e., *bandwidth cost*) when searching Web services for tasks, e.g., the system sends query messages to the service broker to locate Web services and receives response messages from the service broker. When executing composite services, the system also communicates with Web services during task execution. So, the main system cost can be described in terms of computation and bandwidth cost.

In the following subsection, we will apply these metrics to compare our proposed Web service composition approaches.

### 4.6.2 Comparison of the Two Composition Approaches

Now we will use the above metrics to compare the two service selection approaches that are proposed in this paper. In general, we observe a tradeoff between the quality metrics and cost. We discuss the local optimization approach first. The computation cost of the local optimization approach is polynomial. The bandwidth cost is also very limited: for each task, between the service requestor and the service

broker, there are two messages (i.e., `query` message and `result` message); between the service requestor and the selected Web service, there are three messages (i.e., `assign`, `initial` and `completed` message). However, there are two main problems in respect to QoS of the execution result: (1) The local optimization regards the execution of a task as being independent from other tasks when selecting a Web service to execute the task. It cannot consider inter-task constraints, which will lead to sub-optimal execution results in terms of service quality. For example, in the `Travel Planner` statechart, task $t_2$ (i.e., `AttractionSearching`) and task $t_3$ (i.e., `FlightTicketBooking`) are executed concurrently. If we assume execution duration for task $t_3$ is always longer than task $t_2$, optimally, when selecting a Web service to execute task $t_2$, the system should select the Web service that offers the lowest execution price, without consideration of execution duration. But in the local optimization composition approach, the system cannot take advantage of the concurrency between $t_2$ and $t_3$, it may not be able to select the Web service that offers the lowest execution price, since the system considers both execution duration and execution price. (2) When selecting Web services, the local optimization approach can consider constraints on single task. But it cannot consider global constraints, i.e., constraints that cover multiple (or all) tasks in composite services. Although it is always able to select a Web service with minimal execution price, or minimal execution duration for each task. However, it fails when both total execution price and execution duration need to be considered at a global level. For example, it can't enforce a constraint where the composite service's execution price can not exceed 500 dollars and execution duration cannot exceed 3 days.

In the global planning based approach, the planning of composite service execution incurs a substantial computation cost and bandwidth cost. The global planner needs to select an optimal execution plan to execute composite services, it must also monitor the potential Web services that can be used to execute tasks in the composite service.

When changes happen in Web services, the global planner needs to evolve execution plans at runtime. Another problem with the global planning approach is that users are required to provide necessary input for all the tasks in the process schema even if some of the tasks may not be executed. For example, when there are some or-split in the composite service, i.e., there are multiple execution paths.

The advantage of the global planning approach is that it can select the Web services that can satisfy users' global constraints. If end users always specify their constraints on a single task and there is absolutely no requirement for specifying global constraints, then the local optimization Web services composition is ideal because its system cost is low compared to the global planning approach. In our framework, we support both types of service selection approaches to give users the choice to specify the approach that best meets their requirements.

In order to validate the selection approaches introduced in this chapter, both local optimizer and global planner have been implemented as means to select Web services for composite services. We also conduct experiments by executing composite services using these two selection approaches. From the experimental results we conclude that the global planning approach gives a better QoS of composite service execution with little extra system cost. More details about implementing these two components can be found in Chapter 6.

## 4.7 Related Work

Web service composition is a very active area of research and development [6, 24, 31, 32]. In this section, we first briefly examine some Web service standards then look at some service composition prototypes.

Several standards that aim at providing infrastructure to support Web services composition have recently emerged including SOAP [78], WSDL [93], UDDI [81], and

BPEL4WS [12]. SOAP defines an XML messaging protocol for communication among services. WSDL is an XML-based language for describing web service interfaces. UDDI provides the directory and a SOAP-based API to publish and discover services. BPEL4WS provides a process-based language for services composition. Notations for service description and composition have also been proposed in other efforts such as ebXML from the B2B integration community and DAML-S from the semantic Web community. In addition, there are some works focus on QoS of Web services, service level agreement [52, 61]. However, modelling the QoS of composite service is still missing in these standard efforts. It should be noted that the above standards are complementary to our approach. Our approach builds upon the building blocks of these standards (e.g., SOAP, UDDI) to provide a quality-driven and dynamic service composition model.

Not much work has been done on QoS-driven service compositions. Previous work has investigated dynamic service selection based on user requirements. Related projects include CMI [38] and eFlow [23]. CMI's service definition model features the concept of a *placeholder activity* to cater for dynamic composition of services. A placeholder is an abstract activity replaced at runtime with a concrete activity type. A selection policy is specified to indicate the activity that should be executed in place of the placeholder. In eFlow, the definition of a service node contains a *search recipe* represented in a query language. When a service node is invoked, a search recipe is executed in order to select a reference to a specific service. Both CMI and eFlow focus on optimizing service selection at a single task level. In addition, no QoS model is explicitly supported. Our approach focuses on optimizing service selection at a composite service level. Based on a generic QoS model, a novel service selection approach that uses linear programming techniques has been proposed.

Related work on QoS has been done in the area of workflows. In general, most existing projects in this area focus on specifying and enforcing temporal constraints [35, 11].

Other projects such as METEOR [18] and CrossFlow [55] consider more comprehensive QoS models. METEOR [18] considers four quality dimensions, namely time, cost, reliability and fidelity. However, this work does not focus on the dynamic composition of services. It focuses on analyzing, predicting, and monitoring QoS of workflow processes. CrossFlow proposes the use of continuous-time Markov chain to estimate execution time and cost of a workflow instance. It should be noted that efforts in this area are complimentary to our approach.

Other complementary research proposals include [66, 72], which focus on data quality management in cooperative information systems. They investigate techniques to select best available data from different service providers based on a set of data quality dimensions such as accuracy, completeness, and consistency.

## 4.8 Summary

Dynamic selection of component services is an important issue in Web services composition. In this chapter, we present a general and extensible model to evaluate QoS of both elementary and composite services. Based on the QoS model, a global service selection approach that uses linear programming techniques to compute optimal execution plans for composite services is described.

We conducted experiments to compare the proposed technique with the local optimization selection approach. The results show that the global planning approach effectively selects high quality execution plans (i.e., plans which have higher overall QoS). More details about experiments can be found in Chapter 6.

# Chapter 5

# Adaptive Service Composition

The growth of Internet technologies has unleashed a wave of innovations that are marking today's economic truly global. In order to survive dynamic economy environments, it is important for organizations to have the ability to effectively manage business changes. In the previous chapters, we discuss how to generate process schemas for composite services and how to plan the execution of composite services. In this chapter, we switch our attention to the execution of composite services. We will focus on enabling adaptive service composition.

This chapter is organized as follows: Section 5.1 introduces the research issues and outlines the proposed solutions. Section 5.2 presents an approach that handles exceptions occurring in component services. Section 5.3 focuses on handling unexpected exceptions. Section 5.4 gives details on replanning the execution of composite services. Finally, we discuss related work in Section 5.5 and provide a summary of this chapter in Section 5.6.

## 5.1 Introduction

Composite services operate in a highly dynamic environment as new component services may become available at any time, and existing services may be removed, become temporarily unavailable, offer better QoS properties, withdraw advertised QoS properties, etc. Moreover, enterprises are changing constantly: entering into new markets, introducing new products and restructuring themselves through mergers, acquisitions, alliances, and divestitures. Indeed, runtime modification of business processes is necessary to correct errors or deficiencies in process schemas [22, 76], to meet changes in application requirements and technologies, or to incorporate new business policies, etc. Therefore, there is a need for adaptive composition techniques in which composite services will dynamically adjust their operations to respond rapidly to exceptions (e.g., non-availability of a selected component service) and opportunities (e.g., a new component service offering better QoS than existing ones).

Exceptions in workflows can be divided into four categories: *basic failures*, *application failure*, *expected exception*, and *unexpected exception* [34]. The basic failures correspond to failures at the system level (e.g., DBMS, operating system, or network failure); the application failures correspond to failures at application level, i.e., the applications invoked by the WfMS in order to execute a given task; the expected exceptions correspond to predictable deviations from the normal behavior of a process; and the unexpected exceptions correspond to inconsistencies between the business process in the real world and its corresponding workflow schema. According to this classification, in this chapter, we focus on handling application failures[1] and unexpected exceptions for composite services. It should be noted that the expected exceptions can be modelled as service composition rules in our framework, details can be found in Chapter 3.

[1] In composite services, the application failures can be considered as component service execution failures, violation of QoS constraints, etc.

In the rest of this chapter, we present the design of an adaptive service composition framework. Our solution can handle exceptions occurring in component services in peer-to-peer fashion, as well as unexpected exceptions by dynamic modification on process schemas of composite service. The salient features of our framework are:

- *Adaptive service composition*. We propose an adaptive service composition approach in which composite services continuously monitor the behavior of their components and adapt themselves to appropriately react to run-time exceptions on component services (e.g., component service failures, violation of QoS constraints). The adaptive behavior of services is centered around the concepts of *service coordinators* and *control tuple spaces*.

- *Handling unexpected exceptions*. We propose an approach that handles unexpected exceptions by modifying process schemas of composite services. During the execution of a composite service, the service composition manager continuously checks the consistencies between business rules and the process schema of a composite service and migrate the composite service to a new process schema if any modifications on the current process schema are required.

## 5.2 Handling Component Exceptions for Composite Services

We take the view that in order to support adaptive execution of composite services over the Internet, services should be *self-managing*: they should be capable of adapting themselves to appropriately react to run-time exceptions. Expected exception events are generated in response to changes in service execution states. Examples of such events are: component service failure, violation of QoS constraints, and emergence

of new services with better QoS. In our approach, exception handling is facilitated by means of *service coordinators*. A coordinator is essentially an extensible object attached to a service. It is responsible for:

- *Orchestrating service executions based on an execution plan*, i.e., receiving service requests (i.e., SOAP request messages), creating and invoking service instances.
- *Tracing service executions*, i.e., recording when service instances are created, when service instances are completed, etc.
- *Monitoring and controlling service executions*, i.e., detecting, notifying, and handling exceptions.

In this section, we focus on exception handling in composite services, where the exceptions occur in component services. The information required by a coordinator to handle exceptions is represented in the form of a *control tuple space*. The notion of control tuple space builds upon the traditional *tuple space model* [16] and extends it to support exception handling in composite services. The tuple space model is recognized as an appropriate model for managing interactions among loosely coupled entities [63]. In our approach, exception handling is facilitated via control tuples creation, notification, and extraction. The control tuples of a coordinator are : (i) generated based on an exception handling policy which is specified by the service provider, or (ii) provided by other coordinators that have partnerships with the service (e.g., a coordinator of a composite service may add tuples into the spaces associated with its components).

### 5.2.1 An Overview of Control Tuples

A control tuple is a rule of the form `E[C]|A` such that:

- `E` is an *execution exception* or *QoS exception* event. Examples of execution exception events are: (i) `failure(s,t)`, meaning that `s` is unable to execute the task `t`, (ii) `delay(s,t)`, meaning that the execution of the task `t` by the service `s` will take longer than the estimated time, and (iii) `advance(s,t)`, meaning that the execution of the task `t` by the Service `s` completed earlier than the estimated time. Examples of QoS exception events are (i) `QoSImprovment(s,p)` (respectively, `QoSDegradation(s,p)`), meaning that the service `s` has advertised a better (respectively, lower) value for the QoS property `p` (e.g., lower execution price, higher execution price), (ii) `unavailable(s,t)`, meaning that service `s` that can be used to execute task `t` become unavailable.

- `C` is a conjunction of conditions on execution states including event parameter values and service information (e.g., inputs and outputs of tasks).

- `A` is an exception handling action. Table 5.1 summarizes a list of exception handling actions supported in our approach (action signatures are omitted for clarity reasons). Some of the actions (e.g., timeout(), retry()) are based on *service combinators* introduced in [17].

Assume that: (i) `CS` is a composite service, (ii) `p1`, `p2` are the top two execution plans of `cs`, (ii) `p1` is the best execution plan and (iii) $t_1$, $t_2$ are tasks of `CS`. The following are examples of control tuples in tuple space of `CS`: {`Failure(p1.`$t_1$`.service,` $t_1$`)[true]|Replan(CS),` `[true]|Alternative(p1.`$t_2$`.service, p2.`$t_2$`.service)`}. The first tuple indicates that a re-planning is required whenever the corresponding service of $t_1$ (i.e., the service selected to execute $t_1$) fails to execute $t_1$. The second tuple indicates that if the service of p1, which is selected to execute $t_2$ fails, the corresponding service of `p2` should be invoked.

| Actions | Brief Explanation |
|---|---|
| `Replan(BP)` | Generates a new optimal execution plan of a business process `BP` based on the already completed service invocations and the available services that can be used to execute the remaining tasks of the business process. |
| `Timeout(s,t)` | Allows a time limit `t` to be placed on the invocation of a service `s`. If the service execution has not completed within that time, it is considered as failure. |
| `Retry(s)` | Allows to re-invoke a service `s` after a failure. |
| `Forward(`$s_1$`,`$s_2$`)` | Allows a service $s_1$ to forward an invocation message to another service $s_2$. |
| `Alternative(`$s_1$`,`$s_2$`)` | Allows to invoke a second service $s_2$ in case the invocation of the primary service $s_1$ fails. |
| `Multiple(`$s_1$`, `$s_2$`, ..., `$s_n$`)` | Allows to invoke multiple services (i.e., $s_1$, $s_2$, ..., $s_n$) at the same time and returns the results of the service that completes execution first. |

Table 5.1: Exception Handling Actions

### 5.2.2 Multi-level Exception Handling Policies

Each coordinator possesses a set of exception handling policies (i.e., ECA rules) that are used to generate control tuples for exception handling. In our framework, we classify these policies into three levels according to their usages:

1. Composite-level. Exception handling policies in this level are used by coordinators of composite services to generate control tuples for handling execution exceptions in component services and QoS exceptions in candidate component services. An example of exception handling policy in this level is: `delay(s,t)[t ∈ CS]|Replan(CS)`, meaning that it will require the coordinator of the composite service `CS` to replan the execution if a component service `s` that are used to execute a task `t` in `CS` delays its execution.

2. Community-level. Exception handling policies in this level are used by coordinators of communities to generate control tuples for reporting QoS exceptions in candidate component services. An example of exception handling policy in this level is: `unavailable(s,t)[t ∈ CS ]|Notify(CS)`,

meaning that it will require the coordinator of the community ( where the service `s` is a member of the community) to notify the composite service `CS` if a service $s$ that can be used to execute a task `t` in `CS` becomes unavailable.

3. Component-level. Exception handling policies in this level are used by coordinators of component services to generate control tuples for handling the execution exceptions in component services and reporting QoS exceptions in candidate services. An example of exception handling policy in this level is: `failure(s,t)[t ∈ CS]|Notify(CS)`, meaning that it will require the coordinator of the component service `s` to notify the composite service `CS` if the service `s` fails to execute a task `t` in the composite service `CS`.

### 5.2.3 Control Tuples Generation

Tuple generation policies are set by service providers via multi-level exception handling rules. Service coordinators generate control tuples and distribute them to relevant spaces based generation policies. In this section, we present the details.

Generation of tuples is either *explicit* or *implicit*. In an explicit mode, the service provider sets the collection of tuples that specify exception handling policies at service-definition time. For example, when a composite service is created, the tuples that specify how to handle situations when one or more components fails or is unavailable are created and stored in the tuple space of the composite service coordinator. In an implicit mode, tuples are automatically generated by a coordinator and injected into the space of another coordinator to adapt it to the specific needs of service interactions[2]. Implicit tuple generation is of three types: *composite-community*, *community-member*, and *composite-component* and *component-component*.

[2]It should be noted that although access control, i.e., which coordinators can manipulate which tuples in which spaces, is an important issue, it is outside the scope of this thesis because it is a challenging issue by itself.

- In a *composite-community* mode, the coordinator of the composite service sets the collection of tuples to be added to tuple spaces of the communities which are referenced in the statechart of the composite service, at service-definition time. For instance, a coordinator of a composite service may add a tuple (i.e., `QoSImprovment(s,`$q_{price}$`)[`$q_{price}$`< p.t.price]|Notify(CS)`) to the tuple space of a community in order to be notified about the availability of new members of the community offering lower execution price than current price in execution plan `p`.

- In a *community-member* mode, the community coordinator sets the collection of tuples to be added to the tuple space of a member at member-registration time. This allows, for instance, a community to instruct its members to notify changes regarding their QoS properties. An example tuple can be: `QoSImprovment(s,`$q_{duration}$`)[] |Notify(C)`.

- In a *composite-component* mode, the coordinator of a composite service sets the collection of tuples to be added to tuple spaces of services which are part of a selected execution plan at service-execution time. This allows for instance composite services to delegate exception handling tasks to component services. In practice, it means that a coordinator of a composite service `C` may add a tuple to a coordinator of a component service `s1` and instruct it to forward service invocation to another component service `s2` in case `s1` is overloaded.

## 5.3 Handling Unexpected Exceptions

In the previous section, we presented adaptive features of our service composition approach, which mainly focus on handling exceptions that occur in component services

and assume that the process schema is static during the execution of a composite service. However, business processes in real work are changing over the time. More precisely, in our framework, during the execution of composite services, new rules may be added into the repository, and existing rules may be updated or removed from the repository. The process schemas that are used to create composite services may not conform to the updated business rule repository. Therefore, it is necessary to check whether the current executing composite services need to modify their process schemas in order to adopt the changes in business rule repository. For example, the forward-chain rule (see Table 5.2) is added into the rule repository after the execution of a composite service is started. Assume that the composite service contains the task of `SafetyTesting` and the task has not been completed when the rule is added into the repository. Therefore, when execution of the task of `SafetyTesting` is completed, it is necessary to use the updated business rule repository to regenerate the process schema. Also assume that in the task of `SafetyTesting`, the runtime value of `fatigue-Index` is 28. Such an execution result will trigger the forward-chain rule `fcr2`. Then there is a need to add the task of `verifyTesting` into the process schema.

**FORWARD-CHAIN RULE** `fcr2`
**EVENT** `TaskEvent::executing_task(SafetyTesting)`
**CONDITION** `(SafetyTesting::fatigue-Index < 30)`
**ACTION** `add_task(verifyTesting)`

Table 5.2: A New Forward-chain Rule

In our framework, unexpected exceptions are handled by re-generating new process schemas and migrating composite services to the new schemas. Detail of the procedure is as follows:

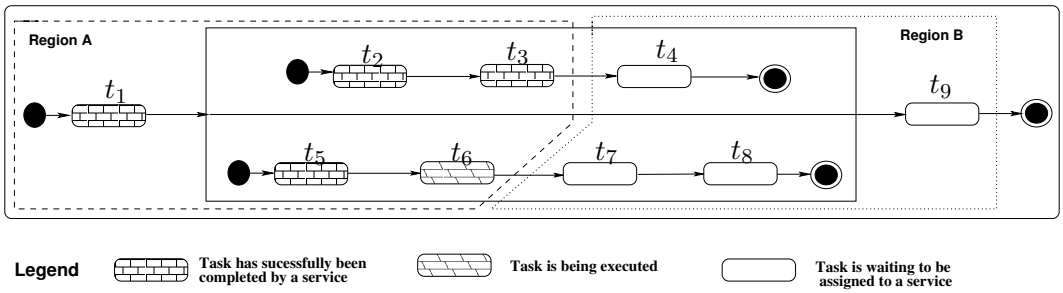



Figure 5.1: Partition a Composite Service into Regions for Regenerating Process Schema

1. Detect Events. In order to check the consistencies between process schemas and business processes in real world, two kinds of events are interested by the coordinator of composite services: task completion events and business rule change events. The task completion event contains runtime values and can be used to generate new process schemas. It should be noted that the task completion events are reported by the coordinator of component services. Another type of event is related to changes in business rule repositories. The business rule repository notifies the coordinator of composite services whenever a change occurs (e.g., insert, delete or update business rules) in the repository.

2. Generating New Process Schema. When the coordinator of composite service is notified of changes in the business repository, it invokes the process schema generator to re-generate process schemas. In our framework, in order to generate new process schemas, we partition the tasks of a composite service into two regions: one region contains the tasks that have been completed or are currently being executed by the component services, another region contains the tasks that are not yet assigned to any component services. An example can be found in Figure 5.1, where region A contains tasks that have been completed or being executed by component services, region B contains tasks that are not yet assigned to any component services. It should be noted that we only check consistencies between the updated business rule repository and the

region of a composite service where the tasks are not yet assigned to any component services for execution (e.g., region B in Figure 5.1). The tasks that have been completed or are currently being executed by component services are considered as the initial state to generate new process schemas. So, the three inputs for the process schema generation become:

(a) Initial state and user's context (i.e., user profile). Here the initial state should be the current execution state of a composite service (e.g., execution result of tasks that had been completed by component services),

(b) The description of user's business objectives, and

(c) The updated service composition rules

The details about process schema generation can be found in Chapter 3. If the new generated process schema is different from the existing process schema, then composite service migration is necessary.

3. Migration Composite Services. Business process migration is a very challenge problem itself, especially when the changes affect the tasks that are currently executing or have been completed by component services. More detail discussion can be found in [19]. In this work, we only consider the case that changes only affect the tasks that have not been assigned to any component service for execution. We focus on improving the QoS of composite services. Since in our framework, a process schema is generated for a composite service (instance), the changes in the process schema will only effect the composite service that it associates with. The changes will not propagate to any other composite services. Assume that a new process schema is generated, the coordinator of the composite service will first suspend the service instance, then generate an execution state based on the new process schema. After that, the coordinator will invoke the global planner to generate a new optimal execution

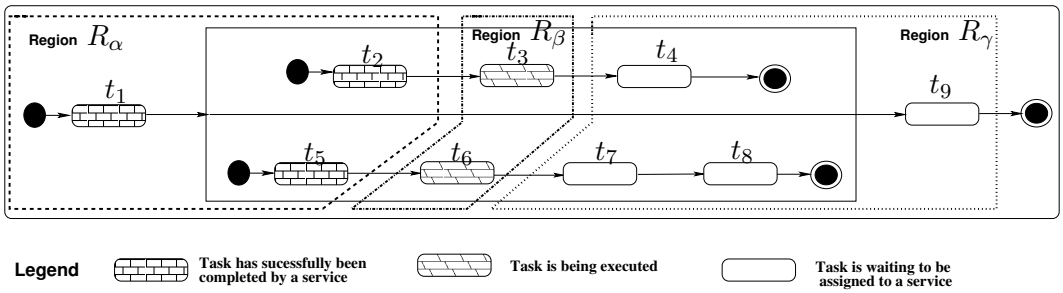



Figure 5.2: Partition a Composite Service into Regions for Replanning

plan for the composite service based on the new execution state, new process schema and current available candidate component services. The coordinator will resume the execution based on the new execution plan.

## 5.4 Replanning the Execution of Composite Services

In our framework, when handling exceptions that occur in component services or unexpected exceptions, replanning the execution of composite services may be used to guarantee that QoS of execution results for composite services is optimal. In this section, we present details on how to conduct the replanning when applying linear programming techniques to plan execution of composite services.

Assume that the composite service that needs to be replanned contains a set of task $T$, where $T = \{t_1, t_2, ..., t_n\}$. Based on the execution statues of tasks, $T$ can be partitioned into three regions: the first region (denoted as $R_\alpha$) contains tasks that have been completed by component services, the second region (denoted as $R_\beta$) contains tasks that are currently being executed, the third region (denoted as $R_\gamma$) contains tasks that are not yet assigned to any component services. An example of partitioning a composite service can be found in Figure 5.2. As we discussed in Chapter 4, there are three inputs in Linear Programming (LP): *variables*, an *objective function* and *constraints* on the variables. When we replan the execution of composite service in runtime, the *vari-*

*ables* and *objective function* remain the same as what we used in pre-execution time. In addition to the constraints we have in pre-execution time, we add some constraints that represent the current execution status. More precisely, each task's execution results in region $R_\alpha$ and each task's assignment results in region $R_\beta$ are used to generate constraints for the LP. For example, assuming that the task $t_1$ is completed by the service $s_{14}$, the actual task execution duration is 20 seconds and execution cost is 10 dollars, then the following constraints can be generated:

$$y_{14} = 1, \tag{5.1}$$

$$a_{14} = 1, b_{14} = 1, \tag{5.2}$$

$$p_{14} = 20, c_{14} = 10 \tag{5.3}$$

Constraint 5.1 indicates that the service $s_{14}$ is selected to execute the task. Constraint 5.2 indicates the service $s_{14}$ that had completed the task and availability and reliability are set to 1. Constraint 5.3 indicates that LP use the actual execution duration and execution price of the task $t_1$ to select the optimal execution plan. For another example, assume that the task $t_3$ is currently being executed by the service $s_{39}$, then a constraint $y_{39} = 1$ can be generated.

With the above constraints, the output of LP will give an optimal execution plan for tasks in region $R_\gamma$.

## 5.5 Related Work

In this section, we review some related work on adaptive workflow, workflow exception handling and service composition frameworks and prototypes.

Adaptive workflow [53, 54] focuses on defining less prescriptive workflow models, which aims for having more flexibility to execute workflow instances. It tries to prevent

the occurrence of exceptions, instead of handling exceptions. CrossFlow [28] provide a solution call FCC (flexible change control) which focuses on making the workflow structure flexible. It identifies three kinds of flexible elements namely alternative activities, non-vital activities, and optional execution orders. When executing workflows, it maps flexible elements to a static workflow schema, which focus on optimizing the QoS of workflow, without considering the changes (e.g., exceptions) occurred at runtime. In our framework, we consider adapting process schema to satisfy the business constraints first and then optimize the QoS of composite services by re-planning. MOBILE is a modular workflow management system mainly addressing late modelling adaptation. It consists of several modules for orthogonal workflow perspectives, such as the so called behavior perspective (i.e., that which is executed in the control flow) or the information perspective (i.e., what data is consumed and produced) [42, 50]. In MOBILE, workflow definitions can be left incomplete in build time, for example, the suitable activity, activity order or subworkflow can only be determined at execution time. However, the user has to decide how a workflow shall be completed concerning aspects left open at workflow definition time. ADEPT [76] concentrated on issues regarding dynamic structure changes of workflow instances at runtime. Based on a conceptual, graph-based workflow model, a complete and minimal set of change operations is introduced that supports users in modifying the structure (i.e., schema) of running workflow instances while preserving their correctness and consistency. However, ADEPT does not provide algorithms that automatically decide under which circumstances which structural adaptations should be applied to a workflow instance. In our framework, we focus on automatically generating new process schemas and migrating composite services to new schemas.

Exception handling in workflows is widely discussed in literatures [19, 20, 40, 62]. In [20], ECA rules are used to handle expected exceptions. A rich *exception-specification language* is proposed. It also provides patterns to facilitate the designing of execution

handling rules. In [62], justified ECA rules are used to handle the expected exceptions. Furthermore, a case-based reasoning (CBR) mechanism with integrated human involvement is used to improve the exception handling capabilities. This involves collecting cases to capture experiences in handling exceptions, retrieving similar prior exception handling cases, and reusing the exception handling experiences captured in those cases in new situations. However, most of these works use centralized exception handling techniques which are not appropriate in the context of composite Web services. Given the highly dynamic and distributed nature of Web services, novel techniques involving peer-to-peer exception handling will become increasingly attractive.

Currently, exception handling in current service composition frameworks and prototypes focus on exceptions occurred in component services and expected exception. In ebXML [33], a set of business protocol exceptions is predefined, which can be considered as expected exception. BPEL4WS [12] adopts approaches of active workflows to handle excepted exceptions. It specifies exceptional conditions and their consequences, including recovery sequences as part of process schemas. Both eFlow [22] and CMI [38] proposed a centralized exception handling mechanism to handle expected exception. In addition, eFlow also handles unexpected exceptions by dynamic modification on composite services. However, modification either conducted by users or target process schemas needs to be predefined. We propose an adaptive service composition framework that supports peer-to-peer exception handling. In order to handle unexpected exception, new process schemas are generated automatically. In addition, we replan the execution to provide optimal QoS of execution results for composite services.

## 5.6 Summary

In this chapter, we presented a framework that supports adaptive service composition, which includes:

- An adaptive service composition approach in which composite services continuously monitor the behavior of their components and adapt themselves to appropriately react to run-time exceptions on component services.
- An approach that handles unexpected exceptions by runtime modification on composite services.

In order to validate the framework introduced in this chapter, the service composition manager and service coordinators have been implemented to enable adaptive service composition. More details on the implementation can be found in Chapter 6.

# Chapter 6

# Prototype

In this chapter, we present the current implementation of the $DY_{flow}$ prototype to illustrate the key concepts and ideas in our approach. We also conduct some experiments to verify our solutions.

This chapter is organized as follows: Section 6.1 gives an overview of the prototype. Section 6.2 briefly describes the implementation of service broker. Section 6.3 overviews the components of service composition manager. Section 6.4 presents the result of experiments. Finally, we provide a summary of the chapter in Section 6.5.

## 6.1 System Architecture

$DY_{flow}$ is a service-oriented architecture (see Figure 6.1) that aims to integrate a large number of distributed, autonomous services. The prototype composed of a *service composition manager*, a *business rule repository*, a *user profile repository*, *a pool of services*, and a *service broker*. All components communicate through SOAP messages (XML documents).

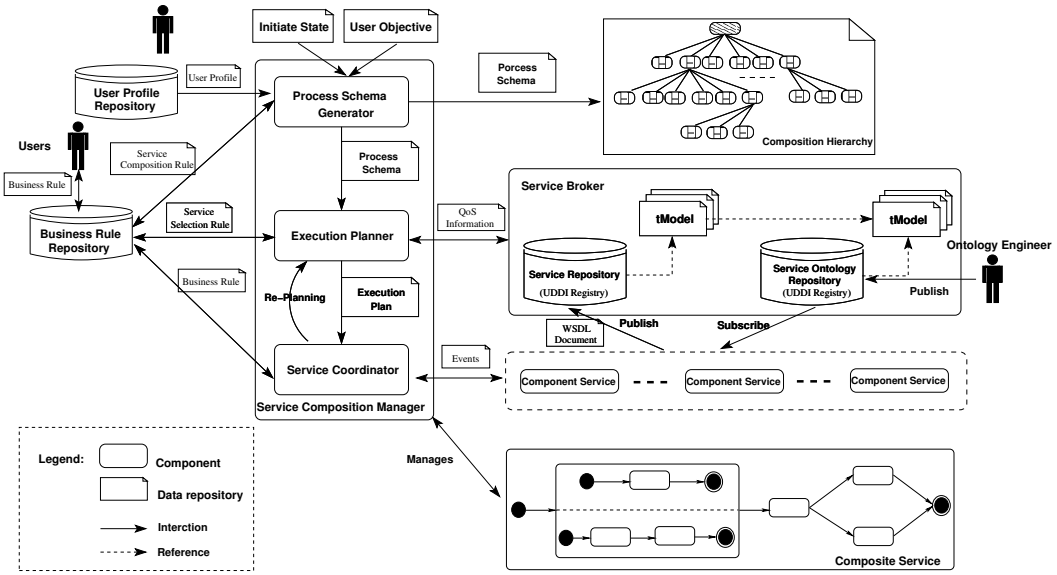



Figure 6.1: Architecture of the $DY_{flow}$ Prototype

The service broker has two repositories: *service repository* and *service ontology repository*. In order to participate to business processes, services need to subscribe to the service ontology and register with the service broker. The ontology engineers are responsible for publishing service ontologies on the repository.

The service composition manager provides a GUI that allows users (e.g., business process designers) to define business rules, specify initiate states and user objectives to generate process schemas to build the composition hierarchy on-the-fly for composite services. Prior to executing composite services, the service composition manager needs to plan execution. It contacts the service broker to retrieve the QoS information of candidate services. It will generate an optimal execution plan for the composite service based on current available services and their QoS properties. Then the service composition manager will orchestrate the service execution on the execution plan. During the execution, the service composition manager will trace service execution and handling exceptions.

In the following sections, we present the implementation of the service broker and the service composition manager. We present the details on how to use the UDDI

registry to implement service ontology repository and Web service repository. We will also describe the main components that are used for the implementation of the service composition manager.

## 6.2 Implementing the Service Broker

There are two meta-data repositories in the $DY_{flow}$ system, namely the service ontology repository and Web service repository. We adopt the UDDI registry to implement both meta-data repositories (see Figure 6.2). The UDDI specification provides a platform independent way of describing services and discovering businesses. The UDDI data structures provide a framework for the description of basic business and service information, and provide an architecture for an extensible mechanism (i.e., tModel) to provide detailed service information using any description language.

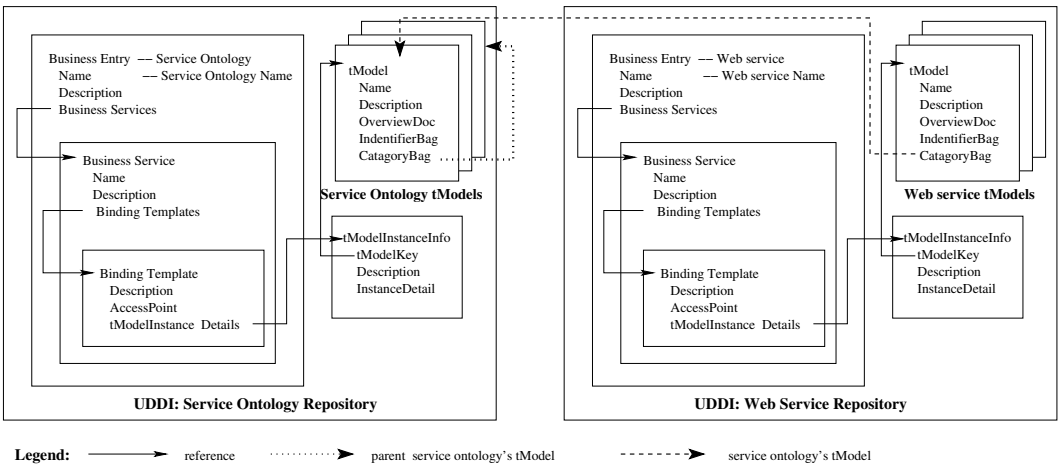



Figure 6.2: Service Ontology Repository and Web Service Repository

We define an XML schema for service ontologies. Each service ontology is represented as an XML document. Table 6.1 shows an example of a `Trip-planning` service ontology for the domain `tourism`. A separate tModel of type `serviceOntologySpec` is created for each service ontology. The information that makes up a `serviceOntologySpec` tModel is quite simple. There

```
< ontology-service NAME=”Trip-planning-services” VERSION=”1.0” >
   < domain > tourism < /domain >
   < domainSynonym > leisure < /domainSynonym >
   < domainSynonym > trip < /domainSynonym >
   < domainSynonym > journey < /domainSynonym >
   < domainSynonym > travel < /domainSynonym >
   < superDomain > ROOT < /superDomain >
   < variable NAME=BonusPoint TYPE=integer>
   < variable NAME=DrivingTime TYPE=real>
   < variable NAME=Discount TYPE=real>
   <serviceclass NAME = ”FlightTicketBooking”
      SUPERCLASS-OF=”domestic-ticket-booking-service, intl-ticket-booking-service” >
      <serviceDescription > This is a service for booking the flight ticket </serviceDescription >
      <attribute NAME = ”serviceProvider” TYPE= ”string” > </attribute>
      <attribute NAME = ”url” TYPE=”string”> </attribute>
      <operation NAME = ”FindTicket”>
         < inputData NAME= ”DepartingAirport” TYPE=”String” </inputData>
         < inputData NAME= ”ArrivalAirport” TYPE=”String” </inputData>
         < inputData NAME= ”FromDate” TYPE=”Date” </inputData>
         < inputData NAME= ”ToDate” TYPE=”Date” </inputData>
         < inputData NAME= ”NumberOfPassenger” TYPE=”integer” </inputData>
         < outputData NAME= ”Flight-schedule” TYPE=”XMLDoc” </outputData>
         < outputData NAME= ”Price” TYPE=”float” </outputData>
         < outputData NAME= ”availability” TYPE=”boolean” </outputData>
      </operation>
      <operation NAME = ”Book-ticket”>
         < inputData NAME= ”Flight-schedule” TYPE=”XMLDoc” </inputData>
         < inputData NAME= ”Credit-card” TYPE=”Credit-card” </inputData>
         < inputData NAME= ”Traveler” TYPE=”Traveler” </inputData>
         < outputData NAME= ”Ticket-receipt” TYPE=”XMLDoc” </outputData>
         < outputData NAME= ”Confirmation-no” TYPE=”String” </outputData>
      </operation>
   </serviceclass>
   <serviceclass NAME = ”AccomodationBooking”>
      <attribute NAME = ”Hotel” TYPE=”Accomodation”> </attribute>
      <attribute NAME = ”Url” TYPE=”string”> </attribute>
      <operation NAME = ”HotelBooking”>
         ...
   </serviceclass>
   <serviceclass NAME = ”Car-rental-services”>
      <operation NAME = ”CarRental”>
         ...
   </serviceclass>
</ontology-service>
```

Table 6.1: Simplified Service Ontology for Trip Planning

is a tModel key, a name (i.e., service ontology's name), optional description, and a URL that points to the location of the service ontology description document. In the child service ontology, the `cateloryBag` should contain the parent service ontology's tModel key and the `keyValue` is `parentServiceOntologySpec`. In the table 6.2, an example of service ontology's tModel in UDDI registry is given.

```
<tModel tModelKey="uuid: 84fe307a-fe3e-4fff-a9fb-79140b265177">
   <name>Trip-planning-services</name>
   <description lang="en">XML specifications of service ontology</description>
   <overviewDoc>
      <description lang="en">Service Ontology</description>
      <overviewURL>http://dyflow.cse.unsw.edu.au/ontology/trip-planning-services.xml</overviewURL>
   </overviewDoc>
   <categoryBag>
      <keyedReference tModelKey="uuid:C1ACF26D-9672-4404-9D70-39B756E62AB4"
         keyName="uddi: A specification" keyValue="specification"/>
      <keyedReference tModelKey="uuid:C1ACF26D-9672-4404-9D70-39B756E62AB4"
         keyName="uddi: An XML specification" keyValue="xmlSpec"/>
      <keyedReference tModelKey="uuid:7341164E-FC52-4813-9810-22270DB32E0E"
         keyName="uddi: An XML specification for Service Ontology" keyValue="serviceOntologySpec"/>
      <keyedReference tModelKey="uuid:1FC3CA8C-1742-4EF5-B8F0-E93B5D84FDDB"
         keyName="uddi: An XML specification for Parent Service Ontology" keyValue="parentServiceOntologySpec"/>
   </categoryBag>
</tModel>
```

Table 6.2: tModel for a Service Ontology

In the Web service repository, we adopt WSDL (Web Service Description Language) [93] to specify services. An example of a WSDL document can be found in Table 6.3. It should be noted that the Web service's tModel contains the key of a service ontology's tModel in `categoryBag`, the `keyValue` is `serviceOntologySpec`. In Table 6.4, an example of Web service's tModel in UDDI registry is given.

Based on UDDI API, the service broker provides two kinds of interfaces for both repositories: the publish interface and the search interface. For the service ontology repository, the publish interface allows an ontology engineer to create a new service ontology. It also provides methods to modify the service ontology such as add a new service class, delete an existing service class, etc. The search interface allows service providers and end users to search and browse the existing service ontologies. The

```
<?xml version="1.0" encoding="utf-8"?>
<wsdl:definitions name="flightBooking"
targetNamespace="http://dyflow.cse.unsw.ede.au/services/flightBooking.wsdl"
xmlns="http://dyflow.cse.unsw.ede.au/services/flightBooking.wsdl"
xmlns="http://dyflow.cse.unsw.ede.au/ontology/Trip-planning-services.xml"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
   <wsdl:types>
      <xsd:schema targetNamespace="http://dyflow.cse.unsw.ede.au/services/flightBooking.wsdl">
      <xsd:element name="Flight-schedule" type="xsd:string"/>
      ... ...
      <xsd:element name="NumberofPassenger" type="xsd:integer"/>
      <xsd:complexType name="FlightTicketBooking_outParametersType">
         <xsd:sequence>
            <xsd:element name="Flight-schedule" type="xsd:XMLDoc"/>
            <xsd:element name="Price" type="xsd:long"/>
            <xsd:element name="Availability" type="xsd:string"/>
         </xsd:sequence>
      </xsd:complexType>
      <xsd:element name="FlightTicketBooking_outParameters" type="FlightTicketBooking_outParametersType"/>
      <xsd:complexType name="FlightTicketBooking_inParametersType">
         <xsd:sequence>
            <xsd:element name="DepartingAirport" type="xsd:string"/>
            ... ...
            <xsd:element name="NumberofPassenger" type="xsd:integer"/>
         </xsd:sequence>
      </xsd:complexType>
      <xsd:element name="FlightTicketBooking_inParameters" type="FlightTicketBooking_inParametersType"/>
         </xsd:schema>
   </wsdl:types>
   <wsdl:message name="FlightTicketBookingSoapOut">
      <wsdl:part element="FlightTicketBooking_outParameters" name="Parameters"/>
   </wsdl:message>
   <wsdl:message name="FlightTicketBookingSoapIn">
      <wsdl:part element="FlightTicketBooking_inParameters" name="Parameters"/>
   </wsdl:message>
   <wsdl:portType name="flightBookingPortType">
      <wsdl:operation name="FlightTicketBooking">
         <wsdl:input message="FlightTicketBookingSoapIn" name="FlightTicketBookingInput"/>
         <wsdl:output message="FlightTicketBookingSoapOut" name="FlightTicketBookingOutput"/>
      </wsdl:operation>
   </wsdl:portType>
   <wsdl:binding name="flightBookingSOAP" type="flightBookingPortType">
      <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
      <wsdl:operation name="FlightTicketBooking">
         <soap:operation soapAction="http://dyflow.cse.unsw.ede.au/services/FlightTicketBooking" style="document"/>
         <wsdl:input name="FlightTicketBookingInput">
            <soap:body use="literal"/>
      </wsdl:input>
      <wsdl:output name="FlightTicketBookingOutput">
            <soap:body use="literal"/>
         </wsdl:output>
      </wsdl:operation>
      </wsdl:binding>
   </wsdl:definitions>
```

Table 6.3: Simplified WSDL Document for a Web Service

```
<tModel tModelKey="uuid: 9760f81e-badd-492b-99ee-77ea408f6645">
   <name>FlightTicketBooking</name>
   <description lang="en">XML specifications of a Web service </description>
   <overviewDoc>
      <description lang="en">Web service description</description>
      <overviewURL>http://dyflow.cse.unsw.edu.au/services/flightBooking.wsdl</overviewURL>
   </overviewDoc>
   <categoryBag>
      <keyedReference tModelKey="uuid:C1ACF26D-9672-4404-9D70-39B756E62AB4"
         keyName="uddi: A specification" keyValue="specification"/>
      <keyedReference tModelKey="uuid:C1ACF26D-9672-4404-9D70-39B756E62AB4"
         keyName="uddi: An XML specification" keyValue="xmlSpec"/>
      <keyedReference tModelKey="uuid:C1ACF26D-9672-4404-9D70-39B756E62AB4"
         keyName="uddi: types" keyValue="wsdlSpec"/>
      <keyedReference tModelKey="uuid:84FE307A-FE3E-4FFF-A9FB-79140B265177"
         keyName="uddi: An XML specification for Service Ontology" keyValue="serviceOntologySpec"/>
   </categoryBag>
</tModel>
```

Table 6.4: tModel for a Web Service

search can be based on a service ontology's domain name, synonyms, service class, etc. For the Web service repository, the publish interface allows service providers to publish or advertise their service descriptions. While the search interface allows the user to discover services by service class name, operation name, input and output data.

## 6.3 Implementation of Service Composition Manger

The service composition manager consists of three modules, namely the *process schema generator*, the *execution planner* and the *service coordinator*. All the components of the service composition manager have been implemented in Java.

The process schema generator receives end users' initiate state (i.e., initiate task) and business objectives (i.e., the target task) as input. It locates the user's profile in the repository and consults business rules to dynamically generate a set of independent process schemas (Statecharts) in XML documents. There are two generation modes: interactive and automatic. In the interactive mode, a process schema (either top level or task level process schema) is generated step by step, allowing end users to refine their business objectives and constraints during the generation of process schemas. In

the automatic mode, all the possible process schemas based on the user's business objectives are generated. In this mode, users supply a value indicating how many inference steps the generator is to use to generate the process schemas. This avoids the problem of the process schema generator spending excessive computation time generating complex and unusable process schemas. The generator may also cache some generation results for frequently used business objectives. These cached process schemas are reused if the business rules or conditions have not been modified.

The execution planner is the module that plans the execution of a composite service using the global planning based approach. The planner is implemented as a linear programming solver based on IBM's Optimization Solutions and Library (OSL) [48]. The advantage of using linear programming is that it can select an optimal execution plan of a composite service without enumerating all the possible execution plans.

The service coordinator orchestrates the execution of composite services. When it receives an execution plan, it creates an instance of composite services. It manages the service instance lifecycle and enables state control, service collaboration and monitoring by executing business rules. It assigns tasks to component services based on the execution plan. It orchestrates the coordination among the component services by executing control and data flows. To be adaptive, whenever exceptions occur in component services, it cooperates with component coordinators to handle exceptions based on exception handling policies. When it receives event notification messages that are initiated by component services, it invokes runtime inference algorithms to check whether there is a need to modify the process schemas. When the service coordinator is notified that there is a change has occurred in the business rule repository, it will invoke the process schema generator to re-generate process schemas for composite services. If the new generated schema is different from the existing one, then the service coordinator will migrate the service instance into to the new process schema.

The service composition manager GUI (see Figure 6.3) provides a single point-of-

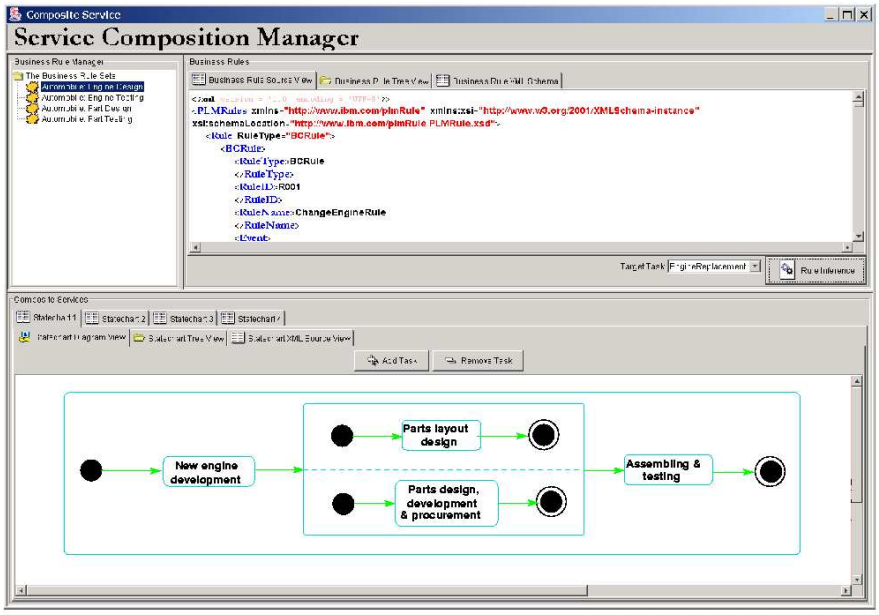



Figure 6.3: GUI of Service Composition Manager

access to the $DY_{flow}$ system. Using the *Business Rule Manager*, users can edit, modify, and delete business rules. Using the *Exception Handling Policy Editor*, users can edit, modify and delete exception handling policies. The GUI also provides a *Profile Manager* that allows users to manage their profile. The lower panel shows the tool for displaying the diagram of a statechart graphically.

### 6.3.1 An Application

To illustrate the viability of our approach, we have implemented an automobile R&D application. We used about 100 business rules in this application. The application incrementally generates composite services to manage the `replacing engine` R&D product process (see Section 3.2). The detailed scenario is as follows:-

- Step 1: Creating top level composite service schema

  In this step, the chief engineer provides a description of his/her business objective (i.e., `replacing engine`) as input to the process schema generator. The process schema generator locates the user's profile and appropriate

business rules to generate an XML document that represents a statechart of service. The graphical presentation of the statechart is shown in Figure 6.3. After creating the top level composite service, the chief engineer will initiate the R&D product process. The task of `new engine development` will be assigned to an *engine designer*.

- Step 2: Creating task level composite service schema

  Assuming that an engine designer is assigned to execute the first task `new engine development` in the top level composite service. The process schema generator needs to generate a composite service for the engine designer to execute this task, since there is a set of service composition rule associates with it. Having the business objective and the engine designer's profile as initial context, the process schema generator can create a task level composite service schema as shown in Figure 6.4. It should be noted that, for a task in a composite service, either an elementary service is used to execute it, or the process schema generator creates a statechart to execute it. For example, for the task of `Cost Evaluation`, since there is no service composition rule for it, an elementary service is used to execute it. However, for the task of `Outsourcing Engine`, since there is a set of service composition rules, the process schema generator create a composite service to execute it.

The above scenario shows that the system only creates the necessary composite service schemas for the R&D product process. It does not enumerate all the possible tasks, control flows, and data flows. Instead of using a single large one-level schema to represent the whole R&D product process, we use composition hierarchy that consists of multiple nested composite services to represent the R&D product process. This modular approach allows distinct processes to be encapsulated in a composite service. This representation is more scalable and makes it easy to implement runtime modification

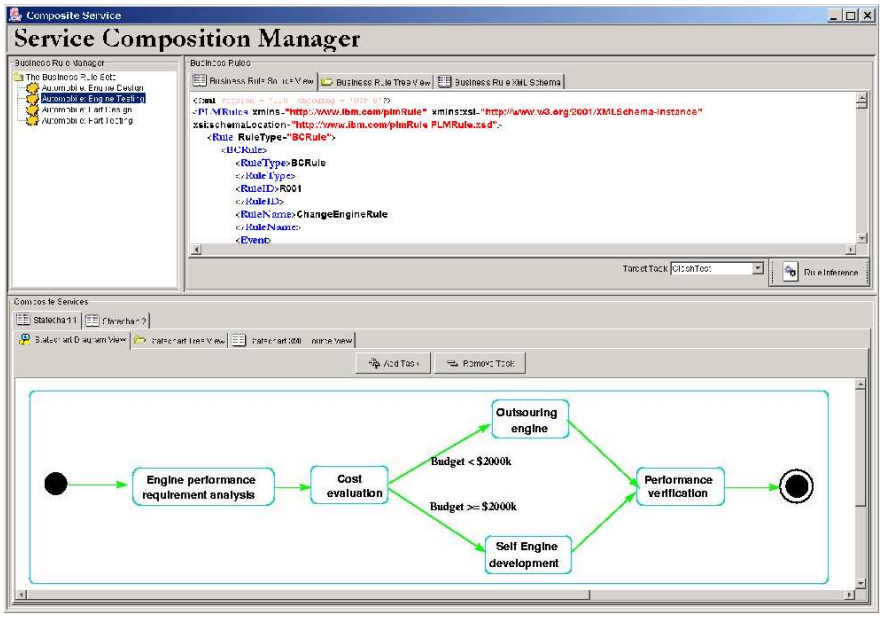



Figure 6.4: Task Level Composite Service for New Engine Development

on composite services.

## 6.4 Experimentation

We conducted experiments using the implemented prototype system. In order to evaluate the performance of both service selection approaches, we developed a travel planning application, which is evolved from [100]. It should be noted that in our system, the performance of service selection approaches is independent of application domains. In this application, a collection of services are created based on the service ontology given in Table 6.1. It should be noted that some QoS information of services are retrieved via generic operations ( e.g., execution duration (resp. execution cost) is retrieved via getExecutionDuration() (resp. getExecutionDuration() ); some QoS information of services are calculated by the service composition manager (e.g., service reputation, reliability and availability) using the formulas that are presented in Section 4.3.1. Services are deployed on a cluster of PCs. All PCs have the same configuration of Pentium

III 933MHz with 512M RAM. Each PC runs Windows 2000, Java 2 Edition V1.3.0, and Oracle XML Developer Kit (Oracle XDK, for XML parsing). They are connected to a LAN through 100Mbits/sec Ethernet cards.

We simulate both static and dynamic environments. In a static environment, there is no change in any component services' QoS properties during executing a composite service and all the component services are able to execute the tasks successfully conforming to their QoS properties. In a dynamic environment, when executing composite services, QoS of component services may undergo changes: existing component services may become unavailable, new component services with better QoS properties may become available, component services may not be able to complete the execution of tasks, etc.

We conduct experiments in each environment. In the experiment, we created several composite services with different numbers of basic states. The composite services were created by randomly adding states to the composite service shown in Figure 4.1. The number of states ranges over the values 10, 20, 30, 40, 50, 60, 70, and 80.

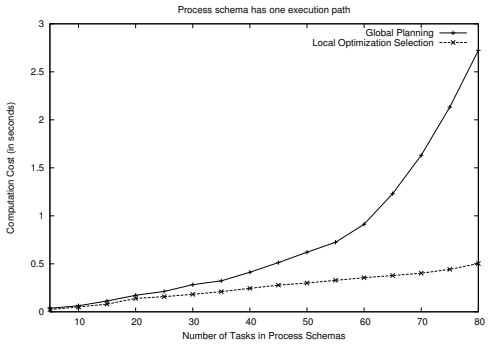
In the experiments, we vary the configuration parameters such as the number of tasks in process schemas, the number of candidate component services per task, the number of execution paths in process schemas to execute composite services using both service composition approaches. We compare both approaches by measuring: (1) computation cost (in seconds) of selecting component services; (2) bandwidth cost (in KBytes, i.e., total network bandwidth between the execution planner and the service broker as well as between the execution planner and services) for selecting and executing component services. For each testing case, we execute composite services 10 times and compute the average computation or bandwidth cost. The aim of this experiment is to investigate the system costs of executing composite services using the LP-based global planning and local optimization approaches.

### 6.4.1 Experiments in Static Environments

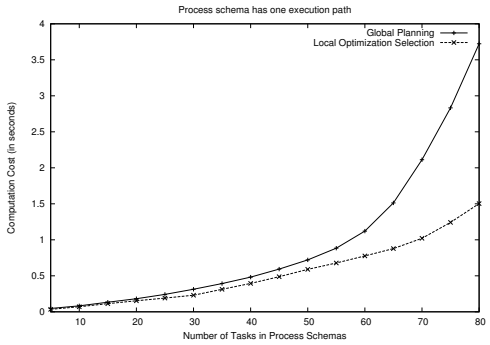
Under static environments, using a global planning approach, once a process execution plan is created, the execution planner does not need to re-plan the process execution. So, the global planner is invoked only once for a composite service. For the local optimization approach, in a composite service, if we assume the number of tasks that are executed is $N$, then the service selector in the execution planner needs to be invoked $N$ times to select component services. In the following subsection, we present the detail experiment results on system cost respectively.

#### Experiment Results on System Cost in Static Environments

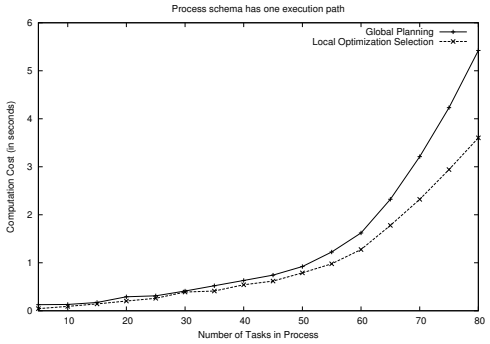
In this subsection, we present the experiment results on computation cost and bandwidth cost respectively in static environments.



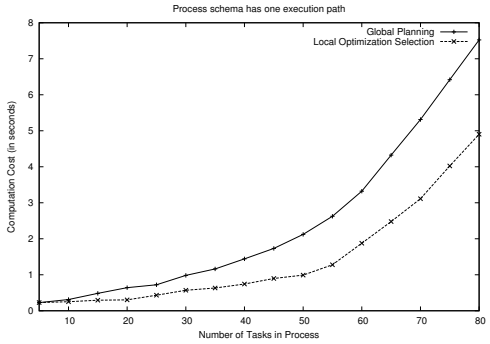

(a) Each Task has 5 candidate component services




(b) Each Task has 10 candidate component services




(c) Each Task has 20 candidate component services



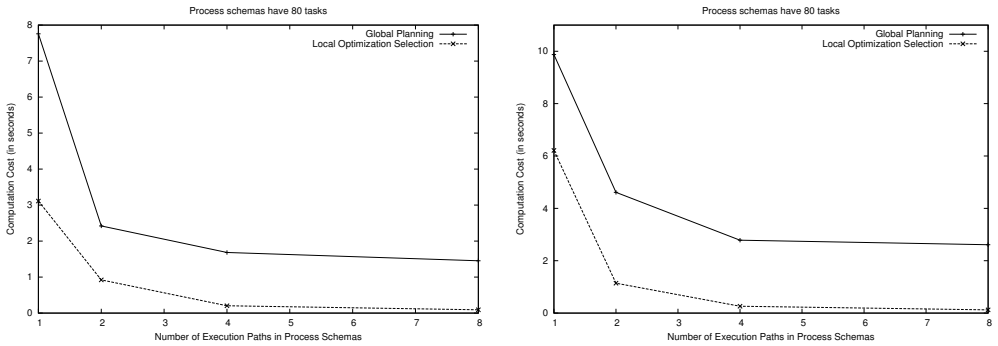

(d) Each Task has 40 candidate component services

Figure 6.5: Experimental Results (computation cost) in a Static Environment, Varying the Number of Tasks in Process Schemas and the Number of Candidate Component Services for Each Task.

1. Experiment Results on Computation Cost in a Static Environment

   Figure 6.5 presents computation cost (in seconds) of selecting services for process schemas that have only one execution path, where we vary the number of tasks in process schemas and the number of candidate services for each task. As expected, in both approaches, the computation cost increases when the number of tasks increases and the number of candidate services increases. Also as expected, the computation cost of global planning is higher than that of local optimization selection. When there are 80 tasks in a process schema and 40 candidate Web services for each task, the computation cost of the global

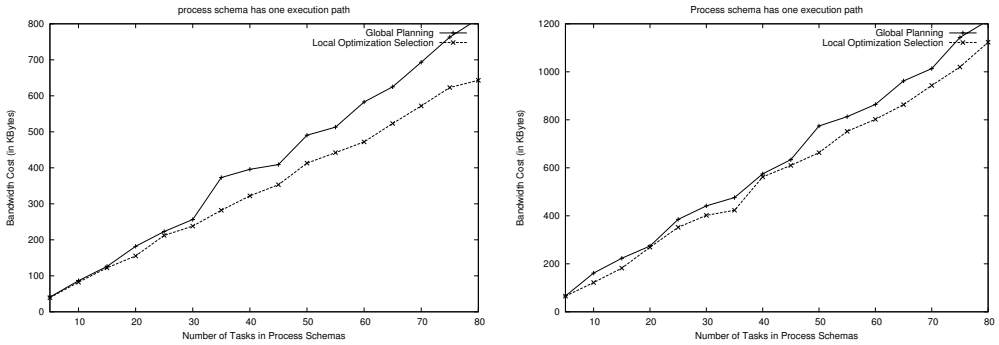
planning (1.6 seconds) is almost 1.5 times higher than the local optimization approach (0.7 seconds). However, the global planning approach is very efficient, it only spends about 8 seconds when selecting services for a process schema that has 80 tasks and each task has 40 candidate services.




(a) Each task has 20 candidate services (b) Each task has 40 candidate services

Figure 6.6: Experimental Results (computation cost) in a Static Environment, Varying Number of Execution Paths in Process Schemas and the Number of Candidate Services for Each Task.

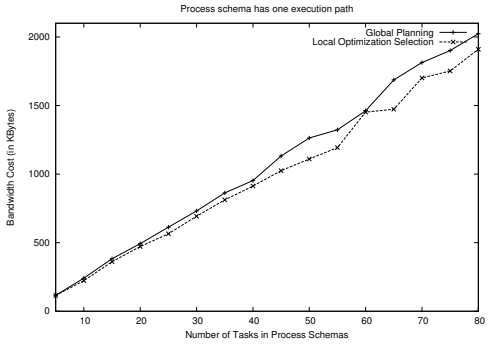
Figure 6.6 represents computation cost (in seconds) of selecting services for process schemas that have 80 tasks, where we vary the number of execution paths in process schemas and the number of candidate services for each task. As expected, in both approaches, the computation cost decreases when the number of execution paths increases, since given a fixed number of tasks in process schemas, when the number of execution paths increases, there are less tasks need to be executed in composite services.



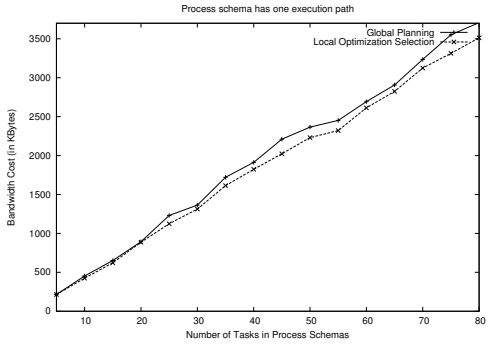

(a) Each Task has 5 candidate services




(b) Each Task has 10 candidate services




(c) Each Task has 20 candidate services



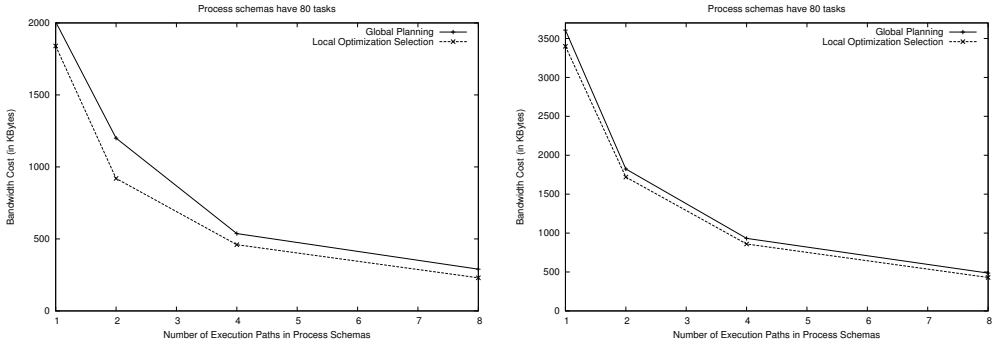

(d) Each Task has 40 candidate services

Figure 6.7: Experimental Results (bandwidth cost) in a Static Environment, Varying the Number of Tasks in Process Schema and the Number of Candidate Services for Each Task.

2. Experiment Results on Bandwidth Cost in Static Environments

   Figure 6.7 presents the bandwidth cost (in KBytes) of selecting and executing Web services for composite services that have only one execution path, where we vary the number of tasks in process schemas and number of candidate services for each task. As expected, in both service selection approaches, the linear increase in the number of tasks increases and the number of candidate services leads almost a linear increase in bandwidth cost. Also as expected, the bandwidth cost of global planning is a little bit higher than that of the local optimization selection approach. When executing composite services with 80 tasks, and there are 40 candidate services for each task, the bandwidth cost of

the global planning (3710 KBytes) requires about an extra 200 KBytes more than the local optimization approach (3503 KBytes) in a static environment.




(a) Each task has 20 candidate services (b) Each task has 40 candidate services

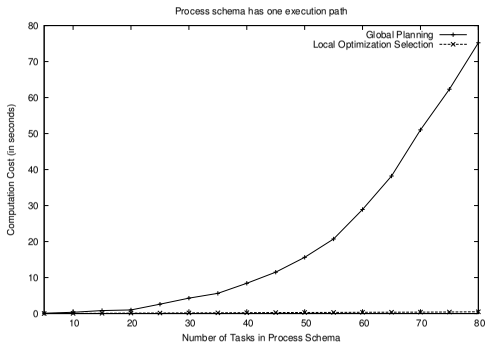
Figure 6.8: Experimental Results (bandwidth cost) in a Static Environment, Varying the Number of Execution Paths in Process Schemas and the Number of Tasks in Process Schemas.

Figure 6.8 presents the bandwidth cost (in KBytes) of selecting and executing Web services for composite services that have 80 tasks, where we vary the number of execution paths in process schemas, the number of candidate services that can be used to execute each task. As expected, in both service selection approaches, the bandwidth cost decreases when the number of execution paths increases.

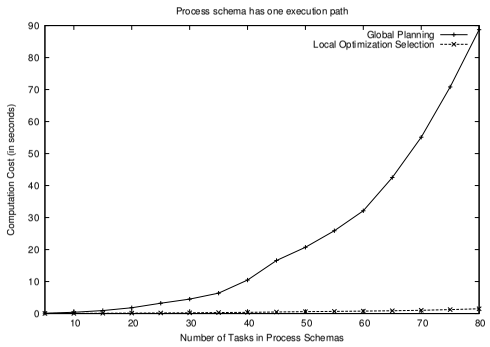
### 6.4.2 Experiments in Dynamic Environments

We also conduct experiments in dynamic environments. In such environments, using a global planning approach, execution of a composite service needs to be replanned whenever a task in the composite service is completed. Assuming the number of tasks in a composite service is $N$, in order to optimize the service selection, the global planner in execution planner needs to be invoked $N$ times during the execution of a composite service. At the same time, the execution planner needs to monitor the

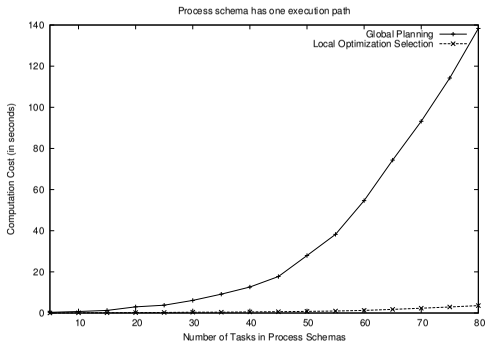
changes in services, which requires extra bandwidth cost. For the local optimization approach, services are selected and executed in the same way as in static environments, thus there is no extra system cost involved. In the following, we present the detail experiment results on system cost.



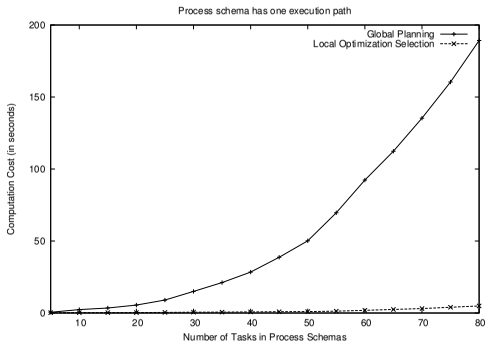

(a) Each Task has 5 candidate services




(b) Each Task has 10 candidate services




(c) Each Task has 20 candidate services




(d) Each Task has 40 candidate services

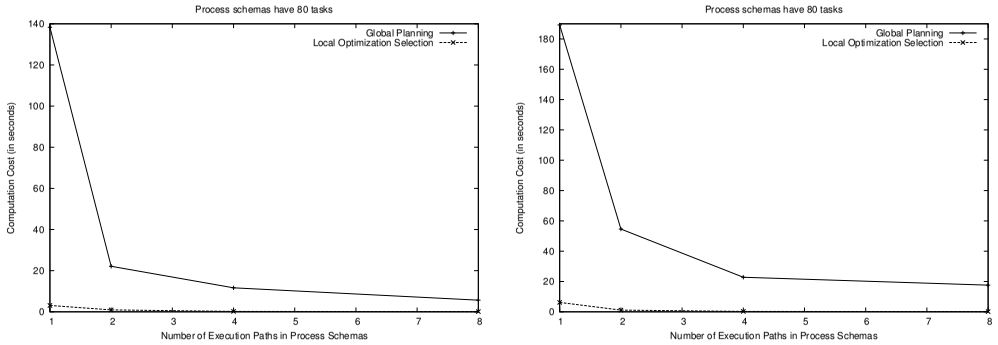
Figure 6.9: Experimental Results (computation cost) in a Dynamic Environment, Varying the Number of Tasks in Process Schema and the Number of Candidate Services for Each Task.

### Experiment Results on System Cost in Dynamic Environments

In this subsection, we present the experiment results on computation cost and bandwidth cost respectively in dynamic environments.

1. Experiment Results on Computation Cost in Dynamic Environments

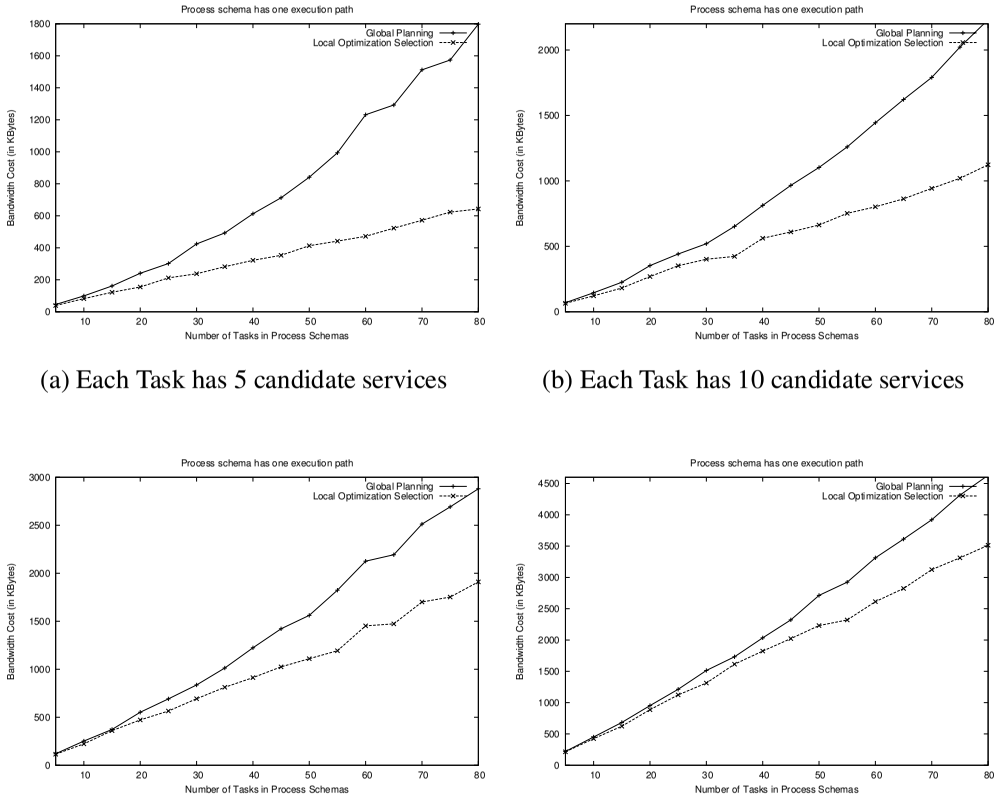
Figure 6.9 presents computation cost (in seconds) of selecting services for process schemas that have only one execution path, where we vary the number of tasks in process schemas and the number of candidate services for each task. As expected, in both service selection approaches, the computation cost increases when the number of tasks increases and the number of candidate services increases. The computation cost of the global planning approach is much higher than that of the local optimization selection. When there are 80 tasks in a process schema and 40 candidate services for each task, the computation cost of the global planning (190 seconds) is almost 40 times higher than the local optimization approach (4.8 seconds). However, the global planning approach is still very efficient, it only spends about 190 seconds when selecting services for a process schema that has 80 tasks and each task has 40 candidate services.




(a) Each task has 20 candidate services (b) Each task has 40 candidate services

Figure 6.10: Experimental Results (computation cost) in Dynamic Environment, Varying the Number of Execution Paths in Process Schemas and the Number of Candidate Services for Each Task.

Figure 6.10 represents the computation cost (in seconds) of selecting services for process schemas that have 80 tasks, where we vary the number of execution paths in process schemas and the number of candidate services for each task. As expected, in both approaches, the computation cost decreases when the number of execution paths increases in process schemas.




(a) Each Task has 5 candidate services

(b) Each Task has 10 candidate services

(c) Each Task has 20 candidate services

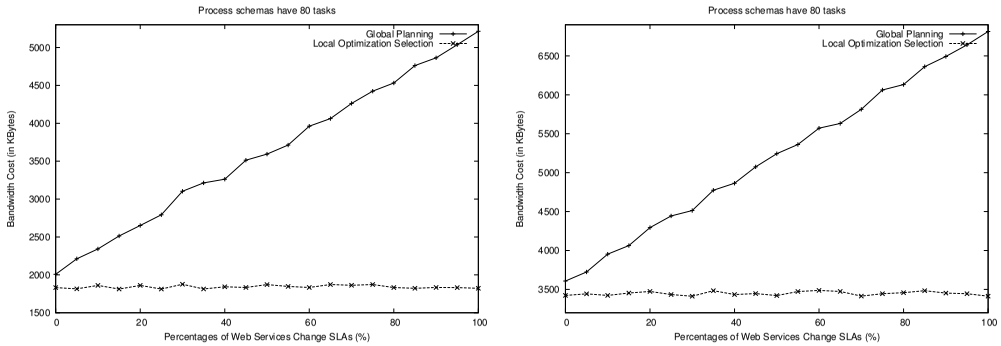
(d) Each Task has 40 candidate services

Figure 6.11: Experimental Results (bandwidth cost) in a Dynamic Environment, Varying the Number of Tasks in Process Schemas and the Number of Candidate Services for Each Task.

2. Experiment Results on Bandwidth Cost in Dynamic Environments

   Figure 6.11 presents the bandwidth cost (in KBytes) of selecting and executing Web services for composite services that have only one execution path, where we assume that 30% of services are changing their SLAs. In the experiment, we also vary the number of tasks in process schemas and the number of candidate services for each task. As expected, in both service selection approaches, the linear increase in the number of tasks increases and the number of candidate services leads almost a linear increase in the bandwidth cost. Also as expected, the bandwidth cost of the global planning approach is higher than that of the local optimization selection approach. When there are 80 tasks in

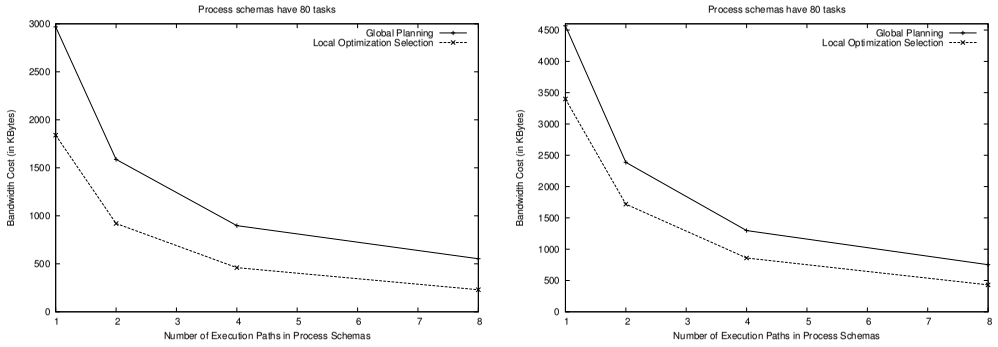
a process schema and 40 candidate services for each task, the bandwidth cost of the global planning (4602 KBytes) requires about extra 1000 KBytes more than the local optimization approach (3608 KBytes) in a dynamic environment.




(a) Each task has 20 candidate services (b) Each task has 40 candidate services

Figure 6.12: Experimental Results (bandwidth cost) in a Static Environment, Varying the Percentage Services Change SLAs and the Number of Candidate Services for Each Task.

Figure 6.12 presents the bandwidth cost (in KBytes) of selecting and executing Web services for composite services that have 80 tasks, where we vary the percentage of services that change SLAs and the number of candidate services for each task. As expected, in the global planning approach, the linear increase in percentage of services that change SLAs leads to almost a linear increase in the bandwidth cost increase. While in the local optimization approach, the bandwidth cost remains almost the same when more services change SLAs.




(a) Each task has 20 candidate services (b) Each task has 40 candidate services

Figure 6.13: Experimental Results (bandwidth cost) in a Static Environment, Varying the Number of Execution Paths in Process Schemas and the Number of Candidate Services for Each Task.

Figure 6.13 presents the bandwidth cost (in KBytes) of selecting and executing Web services for composite services that have 80 tasks, where we vary the number of execution paths in process schemas and the number of candidate services for each task. As expected, in both service selection approaches, the bandwidth cost decreases when the number of execution paths increases.

## 6.5 Summary

In this chapter, we present the implementation of $DY_{flow}$ and some experiments on it. In order to validate the feasibility and benefits of the proposed approaches, The $DY_{flow}$ platform has been used to develop an application for the automobile industry. The application illustrates that the system can efficiently generate process schemas for composite services. In the experiments, relatively large number of services has been integrated using this platform. The results have been encouraging: planning computation cost for integrating 80 component services to execute composite service is about 8 seconds in static environments and about 190 seconds in dynamic environments.

# Chapter 7

# Concluding Remarks

In the previous chapters, the $DY_{flow}$ framework for dynamic service composition has been presented, starting from generating process schemas, then optimal execution planning and finally adaptive execution. In this chapter, we summarize the contributions and identify directions for possible future work.

## 7.1 Contributions

With the proliferation of the Internet and the wide acceptance of e-commerce, an increasing number of distributed and heterogeneous Web services are being offered. These Web services are commonly composed and coordinated to execute business processes, using process-based approaches. Current process modelling provides adequate support for static business processes. However, it has severe limitations on composing Web services for e-business. There are two major requirements for Web service composition. First, frequently changing business conditions imply that business processes need to be composed and refined on the fly. Second, the volatile and dynamic nature of Web environments also leads to the dynamic availability of Web service offerings.

In this work, we propose a dynamic Web service composition framework. The major contributions of this work are:

1. Dynamic creation of process schemas via runtime business rules inference. Instead of statically defining workflows to accommodate an explosive number of possibilities, we advocate a rule-directed approach to dynamically generate and execute composite services. As a result, end users can focus on the business goals to be achieved and the business policies that should be followed, without worrying about detailed description of control and data flow constraints.

2. Quality-driven and dynamic service selection. Instead of fixed association of tasks to Web services at build time, we have devised a quality-driven service selection mechanism that makes use of service quality information, inter-task constraints in business processes as well as preferences set by the requester. In this approach, quality constraints and preferences are assigned to composite services rather than to individual tasks within a composite service. Service selection is then formulated as an *optimization problem* and a *linear programming method* is used to compute optimal service execution plans for composite services.

3. Adaptive Service Composition. We propose an adaptive service composition approach to handle both component and unexpected exceptions. In our approach, composite services continuously monitor the behavior of their components and adapt themselves to appropriately react to run-time exceptions. At the same time, composite services also continuously check the consistencies between their process schemas and the business processes in real world, and modify process schemas if changes occur in business processes.

## 7.2 Directions for Future Work

$DY_{flow}$ proposes an initial solution addressing dynamic service compositions. Such an area is currently heavily investigated by both industries and research institutions, as it is a key solution to enable business process management. In the following, several directions for possible future work are identified.

- Mobile Services and Wireless Environment. In this thesis, we focus on services that are statically hosted in one site. Current, there is a trend to enable mobile services so that service can move from one host to another. It will be a big challenge to optimally select composite mobile services. Another issue is that services may be operated in a wireless environment. It is possible that services are disconnected from the network at runtime. This posts a challenge on how to provide a robust composition solution.

- QoS Negotiation. In this thesis, QoS is non-negotiable. However, in real world, negotiation is very common when people conduct business activity. It is necessary to enable services to conduct QoS negotiation. Another issue is how composite services should negotiate with component services in order to achieve optimal execution result on QoS.

- Knowledge Management. Composite services are operated in dynamic environments. For each execution instance, composite services can log the runtime data, such as QoS of each component services, the context information of exception handling, etc. These data contain useful knowledge that can be used for future execution of composite services. However, the challenge is how to log the runtime data and how to mine the knowledge from logs.

# Bibliography


[1] M. Aiello, M. Papzoglou, J. Yang, M. Carman, M. P. an d L. Serafini, and P. Traverso. A Request Language for Web-services Based on Planning and Constraint Satisfaction. In *VLDB workshp on Technologies for E-Services (TES)*, LNCS, pages 76–86. Springer, 2002.

[2] A. Ankolekar, M. Burstein, J. R. Hobbs, O. Lassila, D. McDermott, D. Martin, S. A. McIlraith, S. Narayanan, M. Paolucci, T. Payne, and K. Sycara. DAML-S: Web Service Description for the Semantic Web. In *Proc. 1st Int'l Semantic Web Conf. (ISWC 02)*, 2002.

[3] E. Ashcroft and Z. Manna. The translation of goto programs into while programs. In *Proceedings of IFIP Congress 71*.

[4] D. Baker, D. Georgakopoulos, H. Schuster, A. R. Cassandra, and A. Cichocki. Providing Customized Process and Situation Awareness in the Collaboration Management Infrastructure. In *Conference on Cooperative Information Systems*, pages 79–91, 1999.

[5] V. Belton and T. Stewart. *Multiple Criteria Decision Analysis: An Integrated Approach.* Kluwer Academic Publishers, 2002.

[6] B. Benatallah and F. Casati, editors. *Distributed and Parallel Database, Special issue on Web Services.* Kluwer Academic Publishers, 2002.

[7] B. Benatallah, M. Dumas, and Z. Maamar. Definition and Execution of Composite Web Services: The SELF-SERV Project. *Bulletin of the IEEE Technical Committee on Data Engineering*, 25(4):47–52, 2002.

[8] B. Benatallah, M. Dumas, Q. Z. Sheng, and A. Ngu. Declarative Composition and Peer-to-Peer Provisioning of Dynamic Web Services. In *Proc. of ICDE'02, IEEE Computer Society*, pages 297–308, San Jose, 2002.

[9] B. Benatallah, M.Dumas, and Q. Z. Sheng. The SELF-SERV Environment for Web Services Composition. *IEEE Internet Computing*, Jan/Feb issue 2003.

[10] E. Best and J. Desel. Partial Order Behaviour and Structure of Petri Nets. *Formal Aspects of Computing*, 2:123–138, 1990.

[11] C. Bettini, X. Wang, and S. Jajodia. Temporal Reasoning in Workflow Systems. *Distributed and Parallel Databases*, 11(3):269–306, 2002.

[12] Business Process Execution Language for Web Services, Version 1.0, 2000. http://www-106.ibm.com/developerworks/library/ws-bpel/.

[13] The Business Process Management Initiative, 2003. http://www.bpmi.org/initiative.esp.

[14] The Business Process Management Language, 2003. http://www.bpmi.org/bpml.esp.

[15] H. C.-L and K. Yoon. *Multiple Criteria Decision Making*. Lecture Notes in Economics and Mathematical Systems, Springer-Verlag, 1981.

[16] G. Cabri, L. Leonardi, and F. Zambonelli. Engineering Mobile Agent Applications via Context-dependent Coordination. *IEEE Transactions on Software Engineering*, 28(11):1040–1056, 2002.

[17] L. Cardelli and R. Davies. Service Combinators for Web Computing. *Software Engineering*, 25(3):309–316, 1999.

[18] J. Cardoso. Quality of service and semantic composition of workflows. *Ph.D Thesis, University of Georgia*, 2002.

[19] F. Casati. Models, semantics and formal methods for the design of workflows and their exceptions. *Ph.D Thesis, Politecnico di Milano*, 1998.

[20] F. Casati, S. Ceri, S. Paraboschi, and G. Pozzi. Specification and Implementation of Exceptions in Workflow Management Systems. *TODS*, 24(3):405–451, 1999.

[21] F. Casati, S. Ilnicki, L. Jin, and M.-C. Shan. An Open, Flexible, and Configurable System for E-Service Composition. Technical Report HPL-2000-41, HP Laboratoris Palo Alto, 2000.

[22] F. Casati, S. Ilnicki, L.-J. Jin, V. Krishnamoorthy, and M.-C. Shan. Adaptive and Dynamic Service Composition in eFlow. Technical Report HPL-2000-39, HP Laboratoris Palo Alto, 2000.

[23] F. Casati, S. Ilnicki, L.-J. Jin, V. Krishnamoorthy, and M.-C. Shan. eFlow: a Platform for Developing and Managing Composite e-Services. Technical Report HPL-2000-36, HP Laboratoris Palo Alto, 2000.

[24] F. Casati, M.-C. Shan, and D. Georgakopoulos, editors. *VLDB Journal, Special issue on E-Services*. Springer-Verlag, 2001.

[25] S. Ceri, P. W. P. J. Grefen, and G. Sanchez. WIDE: A Distributed Architecture for Workflow Management. In *Seventh International Workshop on Research Issues in Data Engineering*.

[26] A. Cichocki, A. Helal, M. Rusinkiewicz, and D. Woelk. *Workflow and Process Automation: Concepts and Technology*. Kluwer academic publishers, 1998.

[27] OMG CORBA/IIOP specifications, 2003. http://www.omg.org/technology/documents/corba_spec_catalog.htm.

[28] CrossFlow project web page, 2000. http://www.crossflow.org.

[29] F. Curbera, M. Duftler, R. Khalaf, W. Nagy, N. Mukhi, and S. Weerawarana. Unraveling the Web Services: an Introduction to SOAP, WSDL, and UDDI. *IEEE Internet Computing*, March/April Issue 2002.

[30] DAML Services, 2002. http://www.daml.org/services/.

[31] A. Dogac, editor. *ACM SIGMOD Record 27(4), Special issue on Electronic Commerce*. ACM, December 1998.

[32] A. Dogac, editor. *ACM SIGMOD Record 31(1), Special Secttion on Data Management Issues in E-Commerce*. ACM, March 2002.

[33] ebXML (Electronic Business using eXtensible Markup Language), 2002. http://www.ebxml.org/.

[34] J. Eder and W. Liebhart. The Workflow Activity Model WAMO. In *Proceeding of th 3th Conference on Cooperative Information Systems*, pages 87–98, 1995.

[35] J. Eder, E. Panagos, and M. Rabinovich. Time Constraints in Workflow Systems. *Lecture Notes in Computer Science*, 1626, 1999.

[36] M.-C. Fauvet, M. Dumas, and B. Benatallah. Collecting and Querying Distributed Traces of Composite Service Executions. In *Proceeding of the 10th International Conference on Cooperative Information Systems (CoopIS)*, Irvine CA, USA, 2002.

[37] D. Georgakopoulos, M. F. Hornick, and A. P. Sheth. An Overview of Workflow Management: From Process Modeling to Workflow Automation. *Distributed and Parallel Databases*, 3(2):119–153, 1995.

[38] D. Georgakopoulos, H. Schuster, A. Cichocki, and D. Baker. Managing Process and Service Fusion In Virtual Enterprises. *Information System, Special Issue on Information System Support for Electronic Commerce*, 24(6):429–456, 1999.

[39] A. Geppert and D. Tombros. Event-based Distributed Workflow Execution with EVE. Technical report, 1998.

[40] C. Hagen and G. Alonso. Flexible Exception Handling in the OPERA Process Support System. In *International Conference on Distributed Computing Systems*, pages 526–533, 1998.

[41] D. Harel and A. Naamad. The STATEMATE Semantics of Statecharts. *ACM Transactions on Software Engineering and Methodology*, 5(4):293–333, 1996.

[42] P. Heinl, S. Horn, S. Jablonski, J. Neeb, K. Stein, and M. Teschke. A Comprehensive Approach to Flexibility in Workflow Management Systems. In *Proceedings of the International Joint Conference on Work Activities Coordination and Collaboration*, pages 79–88. ACM Press, 1999.

[43] S. Helal, S. Su, J. Meng, R. Krithivasan, and A. Jagatheesan. The Internet Enterprise. In *Proceedings of the Second IEEE/IPSJ Symposium on Applications and the Internet (SAINT'02)*, Jan/Feb 2002, Nara, Japan.

[44] S. Helal and M. Wang. Service-Centric Brokering In Dynamic E-Business Agent Communities. In *Proceedings of the fifth International Symposium on Autonomous Decentralized Systems (ISADS) With an Emphasis on Electronic Commerce*, March 26-28, 2001, Dallas, Texas.

[45] J. Hopkins. Component Primer. *Communications of the ACM*, 43(10):27–30, Oct. 2000.

[46] R. Hull, B. Kumar, G. Zhou, F. Llirbat, G. Dong, and J. Su. Optimization Techniques for Data-intensive Decision Flows. In *Proceeding of 16th International Conference on Data Engineering*, 2000.

[47] IBM MQseries Workflow, 2002. http://www-3.ibm.com/software/ts/mqseries/workflow/.

[48] IBM Optimization Solutions and Library, 2002. http://www-3.ibm.com/software/data/bi/osl/index.html.

[49] The Internet Enterprise: Composable e-Services, 2003. http://www.harris.cise.ufl.edu/projects/e-services.htm.

[50] S. Jablonski, K. Stein, and M. Teschke. Experiences in Workflow Management for Scientific Computing. In *Eighth International Workshop on Database and Expert Systems Applications, DEXA '97*. IEEE Computer Society Press, 199.

[51] H. Karloff. *Linear Programming*. Birkhauser, 1991.

[52] A. Keller and H. Ludwig. The WSLA Framework: Specifying and Monitoring Service Level Agreements for Web Services. Technical Report RC22456, IBM research, New York, 2002.

[53] M. Klein, editor. *CSCW-98 Workshop Towards Adaptive Workflow Systems*. 1998.

[54] M. Klein, C. Dellarocas, and A. Bernstein. Introduction to the Special Issue on Adaptive Workflow Systems. *Computer Supported Cooperative Work*, 9(3/4):265–267, 2000.

[55] J. Klingemann, J. Wsch, and K. Aberer. Deriving Service Models in Cross-Organizational Workflows. In *Nineth International Workshop on Research Issues in Data Engineering: Virtual Enterprise, RIDE-VE'99*, Sydney, Australia, March 1999.

[56] C. A. Knoblock, S. Minton, J. L. Ambite, N. n Ashish, I. Muslea, A. Philpot, and S. Tejada. The Ariadne Approach to Web-Based Information Integration. *International Journal of Cooperative Information Systems*, 10(1-2):145–169, 2001.

[57] P. Koksal, I. Cingil, and A. Dogac. A Component-Based Workflow System with Dynamic Modifications. In *Next Generation Information Technologies and Systems*, pages 238–255, 1999.

[58] R. Krithivasan and S. Helal. BizBuilder - An e-Services Framework Targeted for Internet Workflow. In *Proceedings of the third Workshop on Technologies for E-Services, Springer Lecture Notes in Computer Science series, VOL. 2193. In conjunction with VLDB 2001*, Sept 2001, Rome, Italy.

[59] M. Kksalan and S. Zionts, editors. *Multiple Criteria Decision Making in the New Millennium*. Springer-Verlag, 2001.

[60] F. Leymann, D. Roller, and A. Reuter. *Production Workflow: Concepts and Techniques*. Prentice Hall, 1999.

[61] H. Ludwig, A. Keller, A. Dan, and R. P. King. A Service Level Agreement Language for Dynamic Electronic Services. In *The 4th IEEE International Workshop on Advanced Issues of E Commerce and Web Based Information Systems (WECWIS 2002),Los Alamitos, CA, IEEE Computer Society,*, pages 25–32, 2002.

[62] Z. Luo, A. P. Sheth, K. Kochut, and J. A. Miller. Exception Handling in Workflow Systems. *Applied Intelligence*, 13(2):125–147, 2000.

[63] M. Mamei, F. Zambonelli, and L. Leonardi. Tuples On The Air: a Middleware for Context-Aware Computing in Dynamic Networks. Technical Report DISMI-UNIMO-2002-23, University of Modena and Reggio Emilia, 2002.

[64] S. Martello and P. Toth. *Knapsack Problems : Algorithms and Computer Implementations*. John Wiley and Sons, 2001.

[65] D. Mcdermott. Estimated-Regression Planning for Interactions with Web Services. In *6th Int. Conf. on AI planning and scheduling*, 2002.

[66] M. Mecella, M. Scannapieco, A. Virgillito, R. Baldoni, T. Catarci, and C. Batini. Managing Data Quality in Cooperative Information Systems. In *Proc. of the 10th International Conference on Cooperative Information Systems (CoopIS)*, Irvine CA, USA, 2002.

[67] B. Medjahed, B. Benatallah, A. Bouguettaya, A. H. H. Ngu, and A. Elmagarmid. Business-to-Business Interactions: Issues and Enabling Technologies. *The VLDB Journal*, to appear.

[68] J. Meng, S. Y. Su, H. Lam, and A. Helal. Achieving Dynamic Inter-organizational Workflow Management by Integrating Business Processes, Events, and Rules. In *Proceedings of the Thirty-Fifth Hawaii International Conference on System Sciences (HICSS-35)*, 2002.

[69] Workflow and Semantic Web Processes: Quality of Service, Discovery and Composition, 2003. http://lsdis.cs.uga.edu/proj/meteor/SWP.htm.

[70] C. Mohan. Dynamic E-business: Trends in Web Services. In *VLDB workshp on Technologies for E-Services (TES)*, LNCS. Springer, 2002.

[71] T. Murata. Petri Nets: Properties, Analysis and Applications. *Proceedings of the IEEE*, 77(4):541–580, 1989.

[72] F. Naumann, U. Leser, and J. C. Freytag. Quality-driven Integration of Heterogenous Information Systems. In *Proceedings of the International Conference on Very Large Databases (VLDB)*, pages 447–458, Edinburgh, UK, 1999.

[73] The Object Management Group, 2003. http://www.omg.org/.

[74] J. O'Sullivan, D. Edmond, and A. ter Hofstede. What's in a Service. *Distributed and Parallel Databases*, 12(2–3):117–133, September 2002.

[75] M. Pinedof. *Scheduling: Theory, Algorithms, and Systems (2nd Edition)*. Prentice Hall, 2001.

[76] M. Reichert and P. Dadam. $ADEPT_{flex}$ -Supporting Dynamic Changes of Workflows Without Losing Control. *Journal of Intelligent Information Systems*, 10(2):93–129, 1998.

[77] Q. Z. Sheng, B. Benatallah, M. Dumas, and E. Mak. SELF-SERV: A Platform for Rapid Composition of Web Services in a Peer-to-Peer Environment. In *Proc. of the 28th VLDB Conference*, Hong Kong, China, August 2002.

[78] Simple Object Access Protocol (SOAP) . http://www.w3.org/TR/SOAP.

[79] S. Y. Su, C. Huang, J. Hammer, Y. Huang, H. Li, L. Wang, Y. Liu, C. Pluempitiwiriyawej, M. Lee, and H. Lam. An Internet-based Negotiation Server for E-Commerce. *The VLDB Journal*, Vol. 10, No. 1, Aug. 2001, pp. 72-90.

[80] C. Szyperski. *Component Software Beyond Object-Oriented Programming*. Addison-Wesley and ACM Press, ISBN 0-201-17888-5,1998.

[81] Universal Description, Discovery and Integration of Business for the Web, 2000. http://www.uddi.org.

[82] W. van der Aalst. Woflan: A Petri-net-based Workflow Analyzer. *Systems Analysis - Modelling - Simulation*, 35(3):345–357, 1999.

[83] W. van der Aalst, A. ter Hofstede, B. Kiepuszewski, and A. Barros. Workflow Patterns. Technical Report QUT Technical report. FIT-TR-2002-02 (To appear in Distributed and Parallel Databases.), Queensland University of Technology, Brisbane,, 2002.

[84] A. van Moorsel. Metrics for the Internet Age: Quality of Experience and Quality of Business. Technical Report HPL-2001-179, HP Labs, August 2001. Also published in 5th Performability Workshop, September 2001, Erlangen, Germany.

[85] The World Wide Web Consortium (W3C), 2003. http://www.w3.org.

[86] D. D. Wackerly, W. Mendenhall, and R. L. Scheaffer. *Mathematical Statistics with Application*. Duxbury Press, 1996.

[87] D. S. Weld. Recent Advances in AI Planning. *AI Magazine*, 20(2):93–123, 1999.

[88] The Workflow Management Coalition, 2003. http://www.wfmc.org/.

[89] Workflow Management Coalition Terminology & Glossary, 2003. http://www.wfmc.org/standards/docs/TC-1011_term_glossary_v3.pdf.

[90] G. Winskel. Petri Nets, Algebras, Morphisms and Compositionality. *Information and Computation*, 72(3):197–238, 1987.

[91] Web Service Choreography Interface (WSCI) 1.0 Specification, 2002. http://wwws.sun.com/software/xml/developers/wsci/.

[92] Web Services Conversation Language (WSCL) 1.0, 2002. http://www.w3.org/TR/wscl10/.

[93] Web Services Description Language (WSDL). http://www.w3.org/wsdl.

[94] Web Services Flow Language (WSFL) Version 1.0, 2002. http://www-3.ibm.com/software/solutions/webservices/pdf/WSFL.pdf.

[95] XLANG: Web Services for Business Process Design, 2002. http://www.gotdotnet.com/team/xml_wsspecs/xlang-c/default.htm.

[96] Workflow Process Definition Interface – XML Process Definition Language (XPDL), 2003. http://www.wfmc.org/standards/docs/TC-1025_10_xpdl_102502.pdf.

[97] 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 (WWW), Budapest, Hungary*. ACM Press, May 2003.

[98] L. Zeng, B. Benatallah, H. Lei, A. Ngu, D. Flaxer, and H. Chang. Flexible Composition of Enterprise Web Services. *Electronic Markets - The International Journal of Electronic Commerce and Business Media*, 2003, to appear.

[99] L. Zeng, B. Benatallah, and A. H. H. Ngu. On Demand Business-to-Business Integration. In *Proceedings of Ninth International Conference on Cooperative Information Systems*, Trento, Italy, 2001.

[100] L. Zeng, B. Benatallah, A. H. H. Ngu, and P. Nguyen. AgFlow: Agent-based Cross-Enterprise Workflow Management System(demonstration paper). In *Proceedings of 27th International Conference on Very Large Data Bases*, Roma, Italy, 2001.

[101] L. Zeng, B. Benatallah, F. A. Rabhi, and J. Kalagnanam. Towards a Scalable and Adaptive Infrastructure for Web Services Composition (submitted for publication). Technical report, School of Computer Science and Enginnering, University of New South Wales, 2003.

[102] L. Zeng, D. Flaxer, H. Chang, and J.-J. Jeng. $PLM_{flow}$– Dynamic Business Process Composition and Execution by Rule Inference. In *VLDB workshp on Technologies for E-Services (TES)*, LNCS. Springer, 2002.

[103] L. Zeng, D. Flaxer, H. Chang, and J.-J. Jeng. Method and Apparatus for Product Livecycle Management in a Distributed Enviroment Enabled by Dynamic Business Process Composition and Execution using Rule Inference, 2003. US patent pending.