Extending Service Oriented Architecture Using Generic Service Representatives

Mehran Najafi, B.E., M.S.

A Thesis Submitted to the School of Graduate Studies in Partial Fulfilment of the Requirements for the Degree PhD of Computer Science

McMaster University

April 21, 2012

Abstract

Service-Oriented Architecture (SOA) focuses on dividing the enterprise application layer of an enterprise system into components (as services) that have direct relationships with the business functionality of the enterprise. Web services, which are based on message exchanges, are the most widely adopted SOA technology. Web services provide web-accessible programs and devices that have been widely promoted for cloud computing environments. However, different types of web services are required to model actual services in the business domain. Particularly, enterprises (business providers such as banks, health care, and insurance companies) usually send their agents or other personnel (e.g., representatives, installers, maintainers, and trainers) to client sides to perform required services. An enterprise agent can be modeled as a software agent - a computer program that cannot be transmitted efficiently by communication messages. Lacking an efficient way to model the transmission of enterprise agents in traditional message based technologies restricts the application and usage of service-oriented architectures. The central problem addressed in this thesis is the need to develop an efficient SOA model for enterprise agents that will enable service providers to process client data locally at the client side.

To address the research problem, the thesis proposes to model enterprise agents in SOA with a generic software agent called the Service Representative. This is a generic software agent which stays at the client side and can be customized by different service providers to process client data locally. Moreover, to employ a service representative, the thesis proposes a new type of web services called Task Services. While a traditional web service, called Data Service, processes client data completely at the server side, a task service is a web service with the capability of processing client data and resources partially or completely at the client side, using a Service Representative. Each task service assigns a task with three components to the generic service representative: task model, task knowledge, and task data. The task components are mapped to business components such as business process models, business rules and actions, and business data, where they can be efficiently transmitted by service messages.

The combination of a service representative and task services provides an executable platform for service providers at the client side. Moreover, the client does not need to reveal its data, and hence privacy and security are maintained. Large volume client data is processed locally, causing less network traffic. Finally, real-time and event-triggered web services can be developed, based on the proposed approach.

The main contributions and novelty of this research are: i) a domain independent computational model of enterprise agents in SOA to support a wide variety of client-processing tasks, ii) client-side web services which are compatible with typical server-side web services and comparable to other client-side processing technologies, iii) extensions of the SOA architecture by adding novel

generic components including the service representative, the competition desk, and the service composition certifier, iv) provision of a formal model of client-side and server-side web services based on their construction of business components, v) empirical evaluations of the web service model in a number of different applications, using a prototype system, and vi) the application of the developed model to a number of target domains including the healthcare field. Furthermore, because client-side and server-side web services are complementary, a decision support model is provided that will assist service developers to decide upon the best service type for a web service.

Acknowledgements

First and foremost, I am heartily thankful to my supervisors, Dr.Norm Archer and Dr.Kamran Sartipi, whose encouragement, supervision and support from the preliminary to the concluding level enabled me to develop an understanding of the subject. I would like to express deep gratitude to my PhD committee members: Dr.Tom Maibaum and Dr.Rolf Sebaldt for agreeing to serve in my PhD committee despite their very busy schedules, and for their insightful comments and discussions. Finally, and most importantly, I am indebted to the love and support that I have received from my family members. In special, I am thankful to my fiancee, my parents, and sister who made several sacrifices to ensure that I could realize my dream of pursuing a PhD.

Dedications

I dedicate this dissertation to my beloved family and friends, especially ...

to my mother who taught me loving someone unconditionally and with no expectations;

to my father for instilling the importance of hard work and higher education;

to my fiancee, Aysan, for her love, patience, advice, encouragement, generosity, and goodness of spirit supported me the hurdles of graduate school;

to my little sister, Maryam, for her patience and understanding;

and to grandpa and grandma for remembering me in their prayers every day.

Declaration by Author

This thesis is composed of my original work, and contains no material previously published or written by another person except where due reference has been made in the text. I have clearly stated the contribution by others to jointly authored works that I have included in my thesis. The PhD research problem (demands for client-side web service), the high-level solution (developing client-side web services using generic service representatives), the extended SOA components (service representatives, the competition desk, and the service composition certifier), and the potential applications (the virtual remote nursing system) are all my original ideas/work and contributions during my PhD program at McMaster University. Throughout each step of my research, I received valuable consultations/ feedback/ supervision from my supervisors Dr. Archer and Dr.Sartipi. I also developed a prototype system of the proposed service representative and task services, which are available on my homepage (http://www.cas.mcmaster.ca/ najafm/).

Glossary

analytic hierarchy process is a structured technique that decomposes a decision problem into a hierarchy of independent subproblems.

autonomous agent refers to a category of software agents with capabilities for task selection, prioritization, goal-directed behavior and decision-making without human involvement.

BPEL (Business Process Execution Language) is a standard executable language for specifying interactions between web services. A flow of web services is modeled by this standard.

business data represent data that are passed among different components of a system.

business rule is a structured or non-structured statement that defines or constrains some aspects of a business.

CDSS (Clinical Decision Support System) provides recommendations for both patients and physicians by applying medical guidelines to personal health information, contained in Personal Health Records (PHRs) or Electronic Medical Records (EMRs).

choreography is a service composition model where collaborating web services communicate with each other directly.

cloud computing is the delivery of computing as a service rather than a product whereby applications, platforms and infrastructures are provided by services over a network.

code mobility refers to any technology that sends executable software programs from a server computer to a client system upon request from clients.

contextual information refers to any information that is relevant to an understanding of a specific object or situation.

data service is a typical web service that performs all service processes at the server side.

EMR (Electronic Medical Record) is a computerized medical record of a patient, accessible in physician's office.

ICA (Independent Component Analysis) is a classification technique that linearly decomposes a multidimensional data vector into components that are as statistically independent as possible.

LDA (Linear Discriminant Analysis) is a classification technique that finds a linear combination of features which separates two or more classes of objects or events.

model checker is a tool that receives a model of a system and tests automatically whether this model meets a given specification.

neural network is a mathematical model that is inspired by the functional aspects of biological neural networks. They are used to model complex relationships between inputs and outputs to find patterns in data.

orchestration is a service composition model which requires a central process to coordinate sending and receiving messages among web services.

PCA (Principal Component Analysis) is a classification technique that is useful for the compression of data. This method is based on an orthogonal transformation which converts a set of observations into a set of values of uncorrelated variables called principal components.

PHR (Personal Health Record) is a computerized medical record about a single patient that is controlled by the patient.

process algebra is a widely accepted technique in the specification and verification of parallel and distributed software systems.

rich Internet application is a client-side processing approach where an Internet user installs a plug-in in his/her browser. Then web applications can process a client's data locally by sending their codes (e.g., XAML files in Silverlight) to the plug-in.

rule engine executes one or more business rules in a runtime environment. This software component also allows non-programmers to add or modify business logic in a business process management system.

service composition certifier is a SOA component which verifies that a composition model includes compatible web services. Then it validates that the composite service responses match with client expectations.

service composition policy is a document including client expectations from a web service composition.

smart Internet is expected to be the next Internet generation where Web contents are more customized for users.

SOAP (Simple Object Access Protocol) is an XML based protocol for exchanging structured information in the communications between web services and client applications.

software agent is an executable program that acts for a user on behalf of an agency.

SVM (Support Vector Machine) is a binary classification technique that finds the op-

timal linear decision surface based on the concept of structural risk minimization.

task data is a task message component including server-side business data that are consumed by business rules and actions during the task business process execution.

task knowledge is a task message component that provides required business rules (descriptive knowledge) and business actions (procedural knowledge) to realize the specified task model.

task model is a task message component that specifies a task using an abstract business process model.

task service is a web service with the capability of processing client data partially or completely at the client side using service representatives.

UDDI (Universal Description, Discovery, and Integration) is a web service standard to register and publish web services in the registry in order to be discovered by potential clients.

wavelet transform is similar to the Fourier transform which decomposes a signal into sines and cosines.

WSDL (Web Services Description Language) is a web service standard that describes interface of services.

XML (Extensible Markup Language) is a set of rules for encoding documents in a machine readable form.

Contents

Chapter 1

Introduction

The topic of this thesis is client-side processing using web services. Our focus of attention is how to efficiently integrate web services with software agents to provide more sophisticated web services for enterprise systems.

Enterprise systems (Soja and Paliwoda-Pekosz [2009]) are strategic communication assets for large organizations such as banking, healthcare and insurance companies. An enterprise system is tightly coupled with the internal structure, processes, and business model of an organization. Architectures for enterprise systems must feature non-functional requirements such as: simplicity, flexibility, maintainability, reusability, and decoupling of technology and functionality.

Service-Oriented Architecture (SOA) (Krafzig et al. [2005]) is a high-level and technology-independent concept that provides architectural blueprints for enterprise systems. SOA based architectures focus on dividing the enterprise application layer into components (as services) that have a direct relationship with the business functionality of the enterprise. The service-oriented paradigm replaces the object-oriented paradigm in the development of enterprise systems. This is because applying the object-oriented approach to building distributed systems results in many remote calls with little payload and often very complex interaction patterns. By contrast, service-oriented systems are more data-centric; they produce fewer remote calls with heavier payloads and simpler interaction patterns. SOA can be used in a business-to-business (B2B) environment (corporations offer services to other companies over public networks such as the Internet) as well as a business-to-consumer (B2C) environment. Particularly in a cloud computing environment, web services provide on-demand provision of computational resources.

Typical web services perform all the processes at the server side and lack a mechanism to provide client-side processing. This research proposes an extension to the SOA model which provides client-side web services as well as typical server-side web services. This chapter briefly discusses characteristics of service-oriented architecture in Section 1. The intent and motivation

behind the proposed SOA extension are discussed in Section 2. The aims of this research are addressed in Section 3, ending with the problem statement of the thesis. The contributions of this thesis are briefly discussed in Section 4. Finally, the outline of the remainder of the dissertation is presented in Section 5.

1.1 Characteristics of the Problem Domain

A distributed system consists of multiple computers that communicate through a network in order to achieve a specific goal. While local and integrated applications (e.g, desktop applications) provide faster response times, remote and distributed applications offer more flexibility, easier change management, higher security, and efficient composability.

A distributed Service-Oriented Architecture (SOA) proposes service computing as a solution for enterprise organizations. In this approach, an enterprise offers its business functionality through services. The following characteristics can be seen in any SOA systems. Moreover, these characteristics form part of the requirements that need to be addressed in any extensions proposed for SOA.

1. Message-based services (Krafzig et al. [2005]) (e.g., Web Services) are the dominant platform for implementing SOA in enterprises. In this platform, a service is a program that interacts with users or other programs via message exchanges. Consequently, in order to provide a business functionality, a service provider has to convert the functionality into a message format (e.g., XML). Simplicity and other characteristics of messages facilitate building SOA based enterprise systems in several ways such as:

   - Several message-based standards have been proposed to address interoperability issues among providers and clients;
   - Services can be developed and implemented automatically, based on request and response message schemas;
   - Messages can be delivered easily to more than one recipient (broadcast or multicast);
   - Messages can be stored in and retrieved from storage efficiently (e.g., a message queue helps to prevent the loss of request messages, even if the provider is momentarily unavailable);
   - Messages are mostly short and can be transmitted efficiently.

   Despite these advantageous characteristics, the nature of message exchange technologies has several drawbacks in implementing SOA . The next section describes a number of these problems that have inspired the proposed approach.

2. Each service-oriented architecture has the following components.

   - *Service* provides business functionality that a service client and other services can use. A service consists of an implementation that provides business logic and data, a service contract that specifies the functionality, usage, and constraints of the service, and a service interface that exposes the functionality.
   - *Service provider* is the owner of the business process which provides business functionality through its services.
   - *Service client* seeks a service that offers the required functionality. In the rest of this thesis, the term "provider" will refer to "service provider" and the terms "service requester" and "client" will refer to "service client".
   - *Service registry* stores the service contracts of the individual services of an SOA system.
   - *Service bus* interconnects the service provider, service client, and registry.

3. In SOA, enterprise related tasks are addressed by interactions between service clients and service providers through services. A service provider *registers* its services in a service registry. A service client *queries* the service registry to obtain an appropriate service from a provider to satisfy its needs (Le et al. [2007]). Then the client and the selected provider(s) *negotiate* service usage terms (Hung et al. [2004]). Finally, the client makes an agreement with a provider to use its service. Different services can be either *composed* to serve a client (Dustdar and Schreiner [2005a]), or *customized* based on a client's contexts (Baldauf et al. [2007]). Figure 1.1(left) displays the fundamental concepts and components in SOA.

4. Web services are built on a based of existing web protocols. These protocols define, locate, implement, and make web services interact with each other. The web service protocol stack is displayed in Figure 1.1(right) and is described as follows:

   - XML (Extensible Markup Language) is the messaging standard for the request and response messages.
   - SOAP (Simple Object Access Protocol) is an XML based protocol for exchanging structured information in communications between web services and client applications.
   - WSDL (Web Services Description Language) describes service interfaces.
   - UDDI (Universal Description, Discovery, and Integration) registers and publishes web services in the registry so they can be discovered by potential clients.

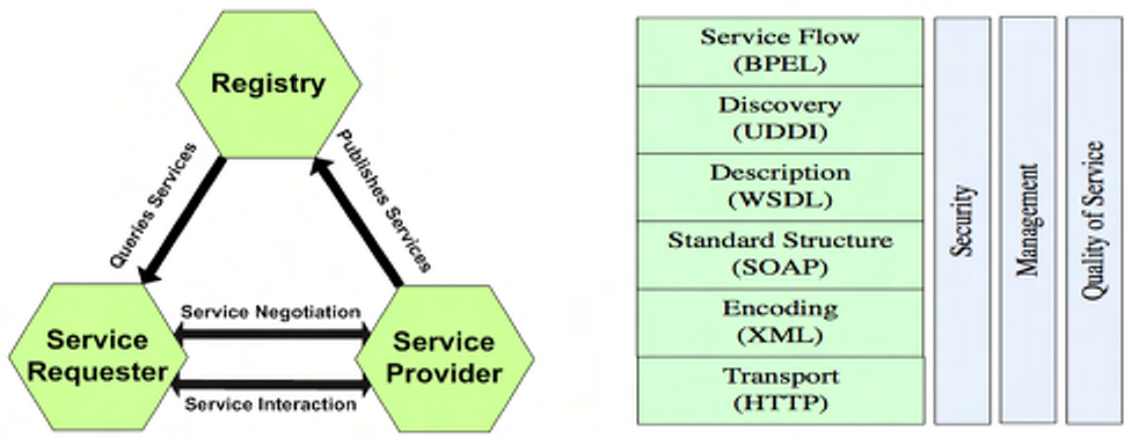


Figure 1.1. SOA components and their relationships [1] (Left) and web service protocol stack [2] (right).

- BPEL (Business Process Execution Language) is a standard executable language for specifying interactions between web services. A flow of web services is modeled by this protocol.

5. SOA vs. client-server architecture. A traditional client/server application runs in a controlled environment such as a LAN/WAN where the client can access the server through a private network. Where the client application needs to be run on machines that would access the data over a public network (the Internet) then the system developer should use web services. This is because the traditional client/server model exposes the server publicly, versus exposing a specific functionality through SOA. Moreover, composite web services demonstrate the integration of different functionalities coming from multiple sources. Finally, web service descriptions are published in service registries which make web services discoverable.

Although a more comprehensive technology platform to implement SOA (instead of a message exchange platform) can be the ideal solution, the wide acceptance and usage of web services motivates improvements the message based SOA, by making it more similar to existing cases in the business domain.

[1] *www.ibm.com/developerworks/websphere/library/techarticles/0409_amanuddin/0409_amanuddin.html*
[2] *www.ibm.com/developerworks/webservices/library/ws − iuse/*

1.2 Research Motivations

Service-oriented architecture is gaining more and more momentum among enterprise business systems by achieving certain strategic benefits, such as reduced automation cost, higher organizational agility, and increased overall flexibility. According to data from *Forrester's Business Data Services* 75% of global 2000 firms had implemented SOA and 62% of medium-sized enterprises were using or planned to use SOA, by the end of 2007 (Heffner and Fulton [2007]).

In general, traditional web services are characterized by the following two principal features: (i) web services are implemented based on message exchanges on the Internet; and (ii) web services perform all their processes at the server side. Therefore, the service client sends all the required service parameters and receives the resulting responses through service messages.

The SOA's mission is to model different aspects of enterprise services by computer services. To achieve this goal, there are two primary concerns:

- Enterprise web services must improve, not restrict, the business functionality provided by primitive enterprise services;
- IT involvement must enhance enterprise features such as privacy and security, billing, and advertisement.

However, the message-based and server-side structure imposes limitations on enterprises that adopt SOA to provide their services. Consequently, the current state of SOA is unable to fully meet these concerns, as follows.

1. **Incomplete enterprise modeling**: SOA, using message exchange technology, is unable to model the entire state of enterprise services. A number of enterprise organizational units can not be modeled directly in SOA, such as customer service units (to support services in different ways), delivery units (to deliver and install services at client sides), and training units (to train clients how to use a service).
2. **Functionality limitations**: there are several types of services that can not be modeled efficiently by message exchanges, such as:
    - *Supervisory services* which are called to control client resources. A set of provider generated messages can not perform this task since an executable platform is needed at the client side that has access to local resources.

- *Event-triggered services* which are called by clients, and the services must wait until a predefined event occurs at the client side. Implementing these services through message exchange technology requires a permanent connection between the provider and the client.
- *Advertising services* which introduce other services to the enterprise while performing their tasks. A message-based service sends a response message to the client, based on the query in the request message, without any opportunity to advertise other services. Even if the provider embeds advertisement messages in the response message, the client cannot extract them since it lacks the required mechanism to predict or detect that advertisement messages will be or are being received.

3. **Client privacy violations**: since web services process client requests at the provider side, the client may need to include personal information in the request messages. This may cause significant privacy and security breaches. Privacy and security issues are becoming more important as the number of services is increasing, and therefore choosing the best services is becoming a major challenge for service clients. A feasible solution for a client would be to trial different services. If trying each service requires revealing personal information, the client is unlikely to attempt trying many choices, and therefore the best service might not be discovered.

4. **Security threats**: a web service can return a mobile agent as the service response to the client side to process client data locally. However, mobile agents are executable codes which can represent malicious behaviour. Moreover, the service client does not have control over the resources (e.g., CPU, memory, data) required by the mobile agent.

5. **Network traffic**: typical web services process client data at the provider side, and the client request message must therefore contain client data. Large volume client data (e.g., local data bases) requires large messages to be sent over the network, increasing network traffic. Moreover, in a dynamic environment, where the client data are changing over time, a traditional service must be called for each change, thus increasing network traffic as well as service cost.

6. **Stateful services**: SOA requires that web services should be designed to work in a stateless fashion. A stateless service does not retain any information about the state of its clients, so its operations can not depend on the history of client interactions. In a stateful service, there are possible sequences of operations that a client can engage with the service. However, in some cases, message exchange technology forces developers to implement stateful services. Stateful services introduce difficulties in SOA such as the following:

- stateful services are not atomic, and composing them poses additional challenges;
- the state of each instance of a stateful service must be checked to verify that the service is in a valid state;
- designing, developing, and using stateful services are more complicated than they are for stateless services.

The SOA community encourages approaches that allow the design of services that respond completely to requests through only one response message. This relieves service providers from keep tracking of the requests.

1.3. Goals of the Research

As discussed in the previous section, some problems in the current state of SOA-based applications originate from the fact that enterprise organizations are not always modeled properly by message based SOA. To simulate the actual business domain, it is necessary to distinguish between two types of services as follows:

- *Server-side Web Services*: service parameters are sent to the provider side to be processed remotely.
- *Client-side Web Services*: service parameters, including confidential, large volume, and/or real-time client data, stay at the client side to be processed locally.

The business domain is well established. Here each business role and unit has its specific tasks and responsibilities. Therefore enterprise modeling approaches should provide counterparts for each of these components. One of the missing parts in current SOA-based models is an enterprise agent that enables a service provider to provide client-side services that process local resources at the client side. Without an efficient model for enterprise agents, all services must process client data at the provider side, resulting in the problems previously described. The problem statement of this research is therefore defined as follows.

> *The goal of this research is to extend the SOA model to provide an efficient model for an enterprise agent that enables service providers to provide client-side web services that process client data locally at the client side. The model thus developed should improve the current state of SOA in terms of maintaining client privacy and security, offering innovative services, and reducing network traffic.*

1.4 Contributions of the Research

The work reported here is an extension to the SOA model which enables client-side processing using web services. The major contributions of this thesis are as follows.

1. An efficient model for enterprise agents, called the *Service Representative*, is described and developed. This is a generic software agent which stays at the client side and can be customized by different service providers to process client data locally. The SOA model is extended by the service representative component to provide a more comprehensive model for enterprise systems. Feasibility of the generic service representative is demonstrated by providing several case studies in different domains.

2. Client-side web services, called *Task Services* are developed in detail. While a traditional web service, called a *Data Service*, processes client data completely at the server side, a task service is a web service with the capability of processing client data and resources partially or completely at the client side using a service representative. This research develops task services based on business components that are compatible with typical web services. Therefore, business components, including business process models, business rules and actions, and business data, are serialized to be sent as service messages. In this research, task services are empirically evaluated and compared against typical data services and other client-side technologies.

3. Composition of data and task services. A BPEL model is proposed to invoke both server-side data services and client-side task services. While typical web services are composed at the server side, the service representative is enhanced, becoming a service orchestrator that composes data services and task services at the client side.

4. An extended SOA architecture, including the service representative, is developed to enable task service invocations. The proposed architecture supports both client-side and server-side web service compositions.

5. A decision making model is developed to guide service developers in their choice of which service types (server-side data services or client-side task services) to use. Data services and task services are complementary and each one has its own applications. This model considers the quality of service parameters when choosing the best service type for a specific application.

6. An innovative application of task services, called the *Virtual Remote Nursing (VRN)* system, is developed for the healthcare domain. This employs the service representative as

a generic full-time nursing agent for patients. Healthcare professionals such as physicians and pharmacists can assign different tasks to the virtual nurse. The virtual nurse provides outputs for the patient (e.g., reminders, and warnings), physicians (e.g., medical reports in customized templates), and third parties (e.g., emergency centres). Each task is defined, based on medical guidelines and workflows which are understandable by physicians. Tasks are predefined and healthcare providers can assign them to the virtual nurse easily.

7. A novel service selection approach is developed which ranks candidate services (both data and task services) based on their performance for a specific client's application. A new SOA component, called the *Competition Desk*, is introduced. This holds a competition among service representatives of the candidate services using a number of test cases. The list of candidate services and test cases is submitted by the service client.

8. A formal model of simple and composite web services. First, a web service is modeled by its business components including: business process, business action, and business data. While a data service applies the business components at the server side, a task service serializes and sends them to the client side to be applied by the service representative. Then, a formal model of simple and composite web services is provided by formalizing the construction of business components using a process algebraic model (Calculus of Communicating Systems).

9. Formal verification and validation of service composition models. For this purpose, a new SOA component, called the *Service Composition Certifier (SCC)* is introduced. This component receives a service composition model as well as a service composition policy from a service client. The service composition policy represents the client's expectations from the service composition. The SCC generates formal models of the entire system that executes the composite service (based on a process algebra) and the service composition policy (based on a temporal logic). Using a built-in model checker, the SCC certifies the service composition model for the composition policy that is submitted.

1.5. Structure of the Thesis

The remainder of the thesis is structured as follows. The generic service representative is introduced in Chapter 2. The client-side task service is proposed and evaluated in Chapter 3. Chapter 4 discusses the composition of data and task services. The virtual remote nursing system as an application of service representatives and task services in the healthcare domain is presented in Chapter 5. Chapter 6 discusses the service selection approach, based on the proposed competition model. A formalized simple composite model of data and task services is developed

in Chapter 7 to introduce the service composition certifier. Chapter 8 guides service developers in choosing the best type of web service. Finally, Chapter 9 presents the conclusions reached and outlines some potential future directions of this research. The dissertation also has two appendixes. One describes the prototype system that was developed, while the second discusses the potential for medical hypertension management, based on the virtual remote nursing system.

Chapter 2

Modeling Service Representatives using Generic Software Agents [1]

As a common practice in the real-world business domain, business functions in an enterprise organization (e.g., customer service, dealerships, training, or delivery) usually send or station an agent or other personnel (e.g., representative, installer, maintainer, and trainer) at the client side to deliver services locally. In contrast, the online services provided by current SOA implementations are limited to services which process client data centrally at the server side. To provide a more comprehensive and parallel model of enterprises, SOA must provide an efficient model for enterprise agents, based on a more distributed approach to customer support. However, the enterprise agent has not been modeled efficiently in SOA, since an enterprise agent is modeled by a mobile agent that can not be transmitted efficiently by service messages. Lack of a more decentralized model may result in limiting the applications and functionality of SOA based systems.

This chapter proposes an efficient model for enterprise agents, that is called the *Service Representative* as it represents service providers at the client side. The service representative is a generic software agent which is customized and trained for different tasks assigned by different service providers. Accordingly, we extend the traditional SOA model to include a generic service representative. In addition to providing innovative applications, such a technique allows for more sophisticated features such as maintaining client privacy and separating the functionality of the service and its delegated agent. For example, since the proposed agent is local to the service client, it can customize service responses based on the client's context. Similarly, this mechanism reduces security and privacy concerns by eliminating the need to send private client information to the provider. To indicate the variety of tasks that can be done by the service representative,

[1] This chapter is mostly based on the following paper: M. Najafi and K. Sartipi, *Extending SOA Architecture using Generic Service Representatives*, Journal of Service Oriented Computing and Applications (SOCA), Springer, 2011 (in press).

three different case studies are presented, where a local and generic agent is customized by service providers to personalize financial advice, apply medical guidelines, and verify credit card transactions respectively.

The organization of this chapter is as follows. Section 1 introduces software agents. Relevant issues in knowledge management are discussed in Section 2. Service representative structure and applications are described in Section 3. Three case studies in business, health care, and insurance domains are presented in Section 4. Section 5 provides an evaluation for the service representative. Related work is discussed in Section 6, and conclusions are discussed in Section 7.

2.1 Software Agents

A software agent (Nwana [1996]) is a piece of software that acts for the user on behalf of an agency. Software agents may have the authority to decide which (if any) action is appropriate. There are several dimensions that can be used to classify existing software agents. This section surveys different agent classes, and outlines a software agent reference model.

2.1.1 Software Agent Categories

There are different types of software agents, including: *intelligent agents* (Wooldridge and Jennings [1995]), capable of learning and reasoning; *autonomous agents* (Huebscher and McCann [2008]), with capabilities for task selection, prioritization, goal-directed behavior and decision-making without human involvement; *distributed agents* (Weiss [1999]), executed in a physically distributed environment; *multi-agent systems* (Weiss [1999]), distributed agents that must communicate to achieve a goal; *mobile agents* (Pham and Karmouch [1998a]), agents that can relocate their execution to different processors; and *customizable agents*, capable of being customized for different tasks. There are common characteristics among these types of software agents such as *social ability*, (agents interact and communicate with other agents), *reactivity*, (agents perceive and respond to make modifications in the environment), and *creativeness*, (agents are able to exhibit goal-directed behavior). An overview of agent classes based on a classification system by Nwana [1996] follows.

1. **Information/Internet agents**: perform the role of managing, manipulating or collecting information from many distributed sources. The information agent needs to know where to look, how to find the information, and how to collect it. An information agent issues various search requests to one or several URL search engines to meet client requests. Some of this search may even be done locally if it has a local cache. The agent collects the information and sends it back to the user.

2. **Autonomous agents**: are differentiated from other types of software agents by their self managing power. Additionally, they enable task selection, prioritization, goal-directed behavior and decision-making without human involvement. Huebscher and McCann [2008] specify the self-managing feature of autonomous agents in the following four categories, where an autonomous agent may have one or more of these features.

   - *Self-Configuration.* An autonomous agent can configure itself according to high-level objectives, based on the needs of the platform and the user. To achieve this goal, the agent specifies what is desired, but not necessarily how to accomplish it.
   - *Self-Optimization.* An autonomous agent can optimize its use of resources.
   - *Self-Healing.* An autonomous agent can detect and diagnose problems.
   - *Self-Protection.* An autonomous agent can protect itself from malicious attacks.

3. **Mobile agents**: can physically travel across a network and perform tasks on different network nodes. Several types of mobile computing that have been introduced have utilized mobile agents, using the following approaches:

   - *Process migration*: a process that is an executable code, migrates from computer to computer and interacts with databases, file systems, information services and other agents. In this approach, the entire process address space is transferred.
   - *Remote evaluation programming*: one computer sends another computer a request in the form of a program. The remote computer executes the program referenced in the request within its own local address space and returns the results to the sending computer.
   - *Mobile objects*: objects can migrate from node-to-node while carrying executable code, data in the form of object-specific properties, and potentially other embedded executable objects.

   Mobile agents can be considered as an improved version of mobile objects, where mobile agents can further reduce network traffic. They provide some autonomy as well as a natural model of asynchronous interaction. A generic mobile agent architecture may consist of six major components: an agent manager; an inter-agent communication manager; a security manager; an application gateway; and a directory manager (Wong et al. [1999b]).

4. **Reactive agents**: represent a special category of agents which do not have internal and symbolic models of their environments; instead, they respond in a stimulus-response manner to the present state of the environment.

5. **Interface agents**: emphasize autonomy and learning in order to perform tasks for their owners. They may also be called personal assistants that collaborate with the user to accomplish certain application tasks. The agent observes and monitors the actions taken by the user, to learn how to assist the user. The user assists the learning process by giving positive and negative feedback to the agent.

6. **Collaborating agents**: cooperate with other agents in order to perform tasks for their owners. Some applications of collaborating agents are: (i) solving problems that are too large for a centralized single agent to do, due to resource limitations; (ii) providing solutions where expertise is distributed; and (iii) improving modularity, speed, reliability, flexibility, and reusability at the knowledge level (hence increasing shareability of resources).

2.1.2 Agent Reference Model

Each software agent consists of a number of components that are specified in a reference model. Therefore, an agent can be defined formally by a tuple of its components. As an example, IBM [2003] suggested a reference model for autonomous agents, which is called the MAPE-K (Monitor, Analyze, Plan, Execute, Knowledge) model and is depicted in Figure 2.1. These components, also found in other agent reference models, are described below.

- *Sensors*: act as agent input devices and obtain data from the system.
- *Monitors*: scan the sensed data generated by the sensors, to extract the relevant data.
- *Analyzers*: analyze or modify the monitored data so the agent can use them.
- *Knowledge bases*: contain knowledge sentences that other agent components can use to perform their tasks.
- *Executors*: process input data to generate outputs.
- *Planners*: act as brains or controllers of agents, specifying how the executors generate outputs or how and when the knowledge base can be used.
- *Effectors*: act as agent output devices.

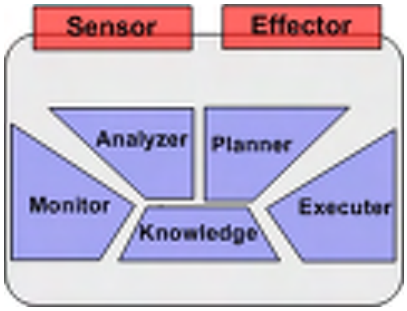


Figure 2.1. Structure of an autonomous software agent, MAPE-K (monitor-analyze-plan-execute over a knowledge base), proposed by IBM [2003].

2.2 Knowledge Transfer

Knowledge Management (Turban et al. [2010]) is a set of techniques and technologies to create, represent, store, transfer, and execute knowledge. To employ a service representative, a service provider sends it the required knowledge (logic) for each assigned task. The service representative receives the knowledge and applies it to client data. Therefore, knowledge representation, transfer, and applications are of interest in this research. Several knowledge representation techniques have been proposed for efficient knowledge transmission over a network, such as:

- **Rule-based model**: consists of rule-based knowledge sentences in the form of *if-then-else* statements, such as the following pattern.

  If Condition (*clientData*) Then *serviceResponse* = Modify (*initialResponse*)

  The above knowledge sentence states that if the defined condition in the client data (*clientData*) is true, the final service response (*serviceResponse*) is obtained by applying the modification function to the initial service response (*initialResponse*).

  Relevant rule-based statements can be grouped into the same category to be evaluated at the same time. Moreover, different rule categories can be ordered for sequential execution. To support the rule-based model, the service representative (SR) requires a built-in rule engine to apply the business rule to the client data. Using the forward chaining strategy, the rule engine matches data against the rules to infer conclusions, which result in actions.

- **Mining model**: represents the result of applying a data mining algorithm to training data, and the resulting model can be used to analyze new data. A mining model is specified by two elements: model *signature* and model *content*. A model signature is in the form of a 3-tuple $< type, inputs, outputs >$ that represents the structure of the model. Moreover, each

mining model has a number of parameters whose values (assigned in the training phase) specialize the model for a specific task. The model parameter values identify the content of each mining model. *Neural networks* (Haykin [2008]) and *decision trees* (Han and Kamber [2000]) are two examples of this model.

  - *Neural network*: includes a network of simple processing elements (called neurons) that can exhibit complex global behaviours, determined by the neuron interconnections and their assigned weights. Learning in a neural network involves adjustments to the neurons and interconnection weights. There are two different styles of training: *incremental* and *batch* methods. In incremental training, the weights and biases of the network are updated each time an input is presented to the network, while in batch training the weights and biases are only updated after all the inputs are presented. To support neural networks, SR requires the following components: i) a neural network builder to build the structure of the model based on the received model signature; ii) a neural network trainer to train the model based on the model content; and iii) a neural network executor to apply the model to the client data and return the result.
  - *Decision tree classifier*: a predictive model that is presented in the form of a tree. Decision tree learning involves constructing a tree by recursively partitioning the training data. In each step, a node is added to the tree to represent a new partitioning. The nodes and their edges represent the content of a decision tree. Similar to the neural network model, the SR can support decision trees if it is equipped with: i) a decision tree builder; and ii) a decision tree executor to work with decision trees.

- **Procedural knowledge**: the knowledge of how to perform a particular task. This is different from other kinds of knowledge, such as declarative knowledge (e.g., rules), in that it can be directly applied to a task. Therefore, it tends to be less general than declarative knowledge. Procedural knowledge can be represented as a sequence of instructions coded in a procedural programming language such as Java or C. To support procedural knowledge, the service representative requires access to the corresponding compiler or interpreter, such as JVM to execute Java codes or a C compiler to support C codes.

2.3 Service Representative

An enterprise agent is modeled by a software agent. Sending agents as service responses (mobile agents) could simulate enterprise agent functionalities in SOA. However the message-based structure of web services does not allow providers to dispatch their agents efficiently. There are

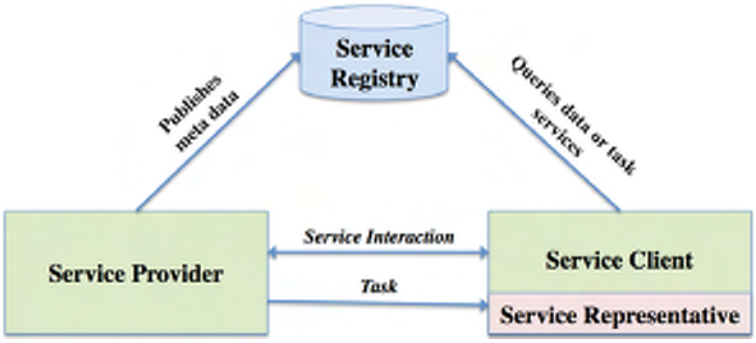


Figure 2.2. Proposed extended SOA model. The proposed generic service representative receives tasks from different service providers to process client data locally.

several security and privacy issues to be considered in mobile agent computing. Viruses and malicious attacks are other possible vulnerabilities of mobile agent systems (Dadhich et al. [2010]). Mobile agent architectures also suffer from low efficiency as they need to send the entire computer program or process to the client side. Moreover, flexibility and interoperability concerns must be addressed in these approaches. These issues motivated the use of resident agents at the client side as opposed to sending them to the client.

This thesis proposes to extend the major components of SOA (service provider, service requester, and service registry) with the service representative, as shown in Figure 2.2. The service representative is modeled by a generic client-side software agent with a built-in process engine. The service representative can be customized and trained by different service providers to perform different client-side processing tasks. This model requires that the service provider only transfers essential messages to customize and train the generic service representative, as opposed to sending the entire agent. The service representative asks for both the task logic and the task data to perform client-side processing. A task is defined to have the following components.

$$Task =< Model, Knowledge, Data >$$

- *Task Model* specifies a task using an abstract Business Process Model (BPM).
- *Task Knowledge* provides the required Business Rules (descriptive knowledge) and Business Actions (procedural knowledge) to realize the specified BPM.
- *Task Data* represents the server-side Business Data (BD) that are consumed by the business rules and actions during business process execution. However, the required client-side business data are provided locally by the service client.

Task components are messages that can be transmitted efficiently over the network. Since the service representative executes at the client side and has access to local resources, it can violate the client's security and privacy. Consequently, the power of the service representative and resources that the service representative has access to should be restricted. Three constraints are imposed to preserve the privacy and security as follows:

1. Clients determine local resources that the service representative has access to. Therefore, the client application must be equipped with a communication channel to communicate with the service representative.

2. Communication between providers and their agents is one way (from providers to the agents); which implies that the agents cannot return any client resources or information to the providers.

3. The service client controls the required computer resources (e.g., CPU time, storage, and memory) for the service representative.

Finally, the generic service representative is transformed into a customized service representative in two phases (customization and training) and then executes the assigned task in the execution phase, as follows.

1. *Customization*: creates an abstract business process based on the task model. The abstract process is assigned to the process engine to be executed. As a result, the service representative is specified for the task.

2. *Training*: realizes the abstract process using task knowledge to generate a complete task instance. The task knowledge can be received from the provider or/and extracted from the local knowledge base. Moreover, the knowledge can be obtained through interactions with the client.

3. *Execution*: passes the task instance with the relevant task data (received from the client's communication channel and task data component) to the business process engine to be executed. Finally, the task results are written to the communication channel to be received by the service client.

2.4 Case Studies

In order to present and evaluate diverse applications of the proposed service representative, three case studies in different domains (banking, health care, and insurance) were designed and

developed. To reduce redundancy and to cover different aspects of the proposed model, similar components have been eliminated in the following case studies.

2.4.1 Case Study 1: Highly Secure Financial Adviser

In order to call a context-aware service, a service client may have to reveal his/her contextual information to the service provider or a context manager, but this may violate the client's information privacy and security. For example, to provide personalized advice, traditional financial advisers ask for personal information from their clients (e.g., client's portfolio or cash information). In this case study, a secure financial adviser is presented in the context of the stock market where a service uses the service representative to personalize financial advice without asking the client to send personal information.

To call the web service, the client sends a request message to the service provider to receive financial advice and then provides his/her financial information (client data) to the service representative through the communication channel. After processing the client request, the financial adviser service generates a set of general financial advice, (e.g., stock buy and sell advice), according to the client's preferences (service parameters). This also defines a task for the service representative (client-side processing) indicating the required procedure (task model) and guidelines (task knowledge) to customize the general financial advice (task data) based on the client's personal information. Related task components are shown in Figure 2.3.

Case Study Specification. The process of generating financial advice could be very complicated and is out of the scope of our discussion. In this case study, the interest is only in the personalization procedure, as follows. This service receives the client's general preferences such as: category of investment (stock, option, or mutual fund); term duration (short term or long term); and risk level (low, medium, or high). However, the client's sensitive information (e.g., financial portfolio or cash balance) is kept local and private. Then the service provider generates a set of general financial advice (stock buy and sell advice) according to the client's preferences.

General financial advice is in the form of either *Buy Advice = < Share Symbol, Min Percentage, Share Price>* or *Sell Advice = <Share Symbol, Max Percentage, Share Price>*. Stock buy (or sell) advice recommends that the client should have a minimum (or maximum) percentage of a specific share in his/her portfolio. The service provider assigns the role of *advice customizer* to the service representative, in order to personalize the general advice based on the local client's financial information, and by performing the following operations.

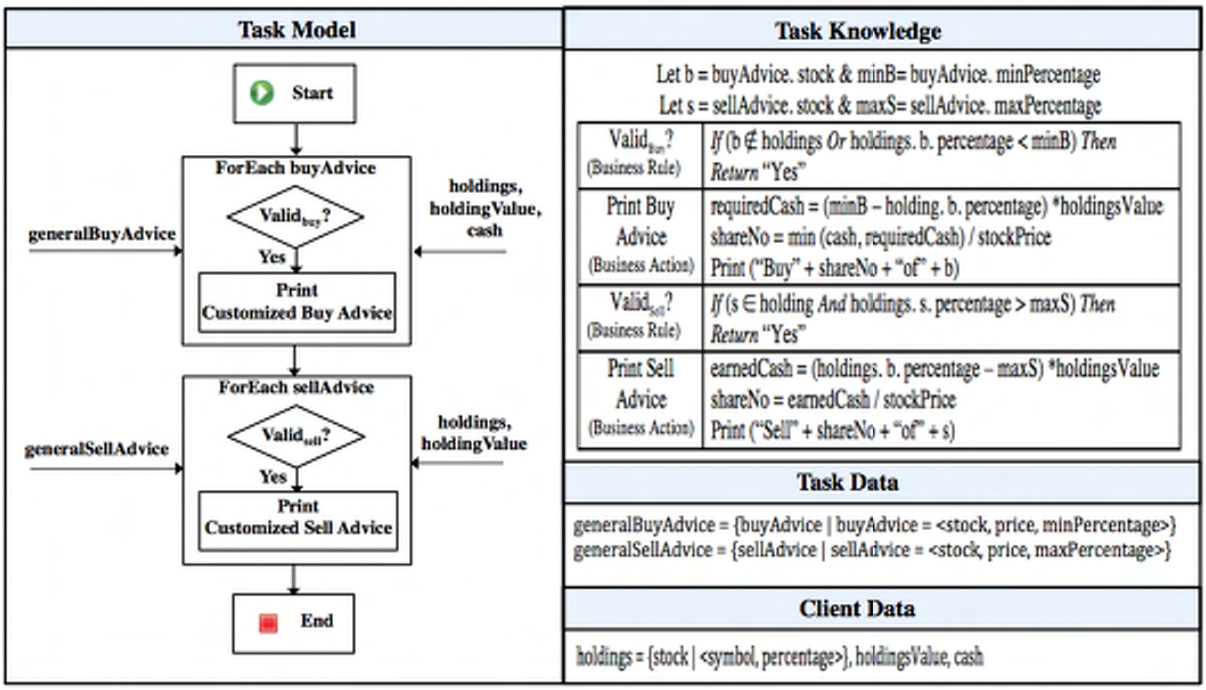


Figure 2.3. Task components of the financial adviser task service and the corresponding client data.

- For each sell advice: if the share symbol is not available in the client's portfolio, ignore the advice. Otherwise, compute the number of shares that the client should sell, based on the client's portfolio and the advice in the *maximum percentage* field.
- For each buy advice: if the client does not have enough cash to buy the corresponding shares, ignore the advice. Otherwise, compute the number of these shares that the client should buy, based on the client's cash and the advice in the *minimum percentage* field.

Service Client. The client application sends a request message to receive financial advice. The request message does not contain the client's sensitive financial information. Moreover, the client application supplies its portfolio and cash information through the communication channel (Figure 2.4), based on the communication channel schema published on the service registry. Finally, the service client receives the final customized advice from the service representative through the communication channel.

Service Provider. The task model specifies a process of applying two categories of rule-based knowledge models to the general advice. The service provider encodes the advice customization rules and actions into the task knowledge segment of the response message. Finally an automated

Portfolio [Share Symbol, Share Number, Holding Percentage] <Read>	Holding Value <Read>	Cash <Read>	Customized Advice <Write>

Figure 2.4. Communication channel schema in the financial adviser case study.

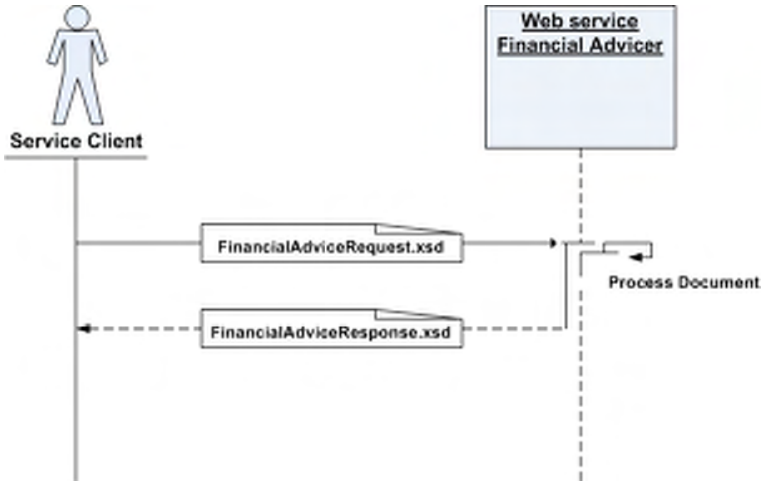


Figure 2.5. Message exchanges between the secure financial adviser web service and the service client.

system or a financial expert generates general financial advice, such as the following. This advice is assembled into the data segment of the response message.

Buy Advice: *<MSFT, 12%, 25.12$>*
Sell Advice: *<AAPL, 5%, 344.00$>*

Service Representative. In the customization phase, the SR uses the received task model to generate an abstract business process with two sub-processes. Moreover, a rule-based knowledge model is assigned to each of the generated sub-processes. In the training phase, the SR loads each rule-based model using the customization knowledge received from the service provider. Finally, in the execution phase, the SR runs the resulting process where in each step of this process, it applies the corresponding rules to customize the general advice. The service representative also stores the customization knowledge into its internal knowledge base to relieve the service provider from sending it each time.

The message exchanges between the service provider and the service client are shown in Figure 2.5. The XML schema of the request and response messages are displayed in Figures 2.6 and 2.7. From these schemas, a WSDL description can be developed for this web service and then a *top-down approach* is used to implement the body of this web service.

```
<xsd:element name="RequestMessageDocument" type="tns:RequestMessage"/>
  <xsd:complexType name="RequestMessage">
   <xsd:sequence>
     <xsd:element name="Category" type="xsd:string"></xsd:element>
     <xsd:element name="Term" type="xsd:string"></xsd:element>
     <xsd:element name="RiskLevel" type="xsd:string"></xsd:element>
    </xsd:sequence>
  </xsd:complexType>
 </xsd:element>
```

Figure 2.6. Financial adviser request message schema (FinancialAdviserRequest.xsd).

```
<xsd:element name="ResponseMessageDocument" type="tns:ResponseMessage"/>
  <xsd:complexType name="ResponseMessage">
   <xsd:sequence>
     <xsd:element name="RoleSegment" type="tns:FinancialAdviser> </xsd:element>
     <xsd:element name="KnowledgeSegment" type="tns:AdviceCustomization> </xsd:element>
     <xsd:element name="InformationSegment" type="tns:GeneralAdvice"> </xsd:element>
    </xsd:sequence>
  </xsd:complexType>

 <xsd:complexType name="FinancialAdviser"> <xsd:sequence>
     <xsd:complexType name="SRStructure"> <xsd:sequence>
         <xsd:complexType name="Sensors"> <xsd:sequence>
             <xsd:element name="1" type="xsd:string" fixed="Portfolio"> </xsd:element>
             <xsd:element name="2" type="xsd:string" fixed="Cash"> </xsd:element>
             <xsd:element name="3" type="xsd:string" fixed="GeneralAdvice"> </xsd:element> </xsd:sequence>
         </xsd:complexType>
         <xsd:complexType name="Effectors"> <xsd:sequence>
             <xsd:element name="1" type="xsd:string" fixed="CustomizedAdvice"> </xsd:element> </xsd:sequence>
        </xsd:complexType>
        <xsd:complexType name="KnowledgeModels"> <xsd:sequence>
             <xsd:element name="CustomizeBuyAdvice" type="xsd:string" fixed="RuleBased"> </xsd:element>
             <xsd:element name="CustomizeSellAdvice" type="xsd:string" fixed="RuleBased"> </xsd:element> </xsd:sequence>
        </xsd:complexType> </xsd:sequence>
 </xsd:complexType>

 <xsd:complexType name="ProcessModel"> <xsd:sequence>
       <xsd:complexType name="Nodes"> <xsd:sequence>
             <xsd:element name="1" type="xsd:string" fixed="Start"> </xsd:element> </xsd:element>
             <xsd:element name="2" type="xsd:string" fixed="CustomizeBuyAdvice"> </xsd:element>
             <xsd:element name="3" type="xsd:string" fixed="CustomizeSellAdvice"> </xsd:element>
             <xsd:element name="4" type="xsd:string" fixed="End"> </xsd:element>  </xsd:sequence>
       </xsd:complexType>
       <xsd:complexType name="Connections"> <xsd:sequence>
             <xsd:element name="Connector1"><xsd:complexType>
                   <xsd:attribute name="From" type="xsd:string" fixed="Node1"/>
                   <xsd:attribute name="To" type="xsd:string" fixed="Node2"/> </xsd:complexType>
             </xsd:element>
             <xsd:element name="Connector2"><xsd:complexType>
                   <xsd:attribute name="From" type="xsd:string" fixed="Node2"/>
                   <xsd:attribute name="To" type="xsd:string" fixed="Node3"/></xsd:complexType>
             </xsd:element>
             <xsd:element name="Connector3"><xsd:complexType>
                   <xsd:attribute name="From" type="xsd:string" fixed="Node3"/>
                   <xsd:attribute name="To" type="xsd:string" fixed="Node4"/> </xsd:complexType>
             </xsd:element>
           </xsd:sequence> </xsd:complexType> </xsd:sequence>
       </xsd:complexType> </xsd:sequence>
  </xsd:complexType>

  <xsd:complexType name="AdviceCustomization"> <xsd:sequence>
     <xsd:complexType name="CustomizeBuyAdvice"> <xsd:sequence>
         <xsd:element name="Rule1" type="xsd:string" fixed="For each b member of BuyAdvice ..."> </xsd:element>
         <xsd:element name="Rule2" type="xsd:string" fixed="For each b member of BuyAdvice ..."> </xsd:element> </xsd:sequence>
     </xsd:complexType>
     <xsd:complexType name="CustomizeSellAdvice"> <xsd:sequence>
         <xsd:element name="Rule1" type="xsd:string" fixed="For each s member of SellAdvice ..."> </xsd:element> </xsd:sequence>
     </xsd:complexType> </xsd:sequence>
  </xsd:complexType>

  <xsd:complexType name="GeneralAdvice"> <xsd:sequence>
     <xsd:element name="BuyAdvice" maxOccurs="unbounded"> <xsd:complexType>
          <xsd:attribute name="ShareSymbol" type="xsd:string"/>
          <xsd:attribute name="ShareNumber" type="xsd:string"/>
          <xsd:attribute name="SharePercentage" type="xsd:integer"/> </xsd:complexType>
      </xsd:element>
      <xsd:element name="SellAdvice" maxOccurs="unbounded"> <xsd:complexType>
          <xsd:attribute name="ShareSymbol" type="xsd:string"/>
          <xsd:attribute name="ShareNumber" type="xsd:string"/>
          <xsd:attribute name="SharePercentage" type="xsd:integer"/> </xsd:complexType> </xsd:element> </xsd:sequence>
 </xsd:complexType>
</xsd:element>
```

Figure 2.7. Financial adviser response message schema (FinancialAdviserResponse.xsd).

2.4.2 Case Study 2: Agent-based Clinical Decision Support System

This section presents a case of a Clinical Decision Support System (CDSS) in the context of vascular diseases. A CDSS provides recommendations for both patients and physicians by applying its medical guidelines to personal health information, contained in Personal Health Records (PHRs) or Electronic Medical Records (EMRs). Sending personal information, as it is required by typical CDSSs, may violate a patient's privacy and security. Moreover, since a medical center calls the same CDSS for different patients, transferring PHR data over the network increases network traffic significantly. Based on the proposed model, a CDSS can employ the SR to apply its medical guidelines to local PHRs to improve security and efficiency.

This work is based on a CDSS called the COMPETE III Vascular Tracker (C3VT) [2]. Using this system, a physician (service client) supplies the patient's PHR information (client data) to the communication channel and sends a request for medical advice. After processing the client request, the CDSS defines a task for the service representative, using a medical workflow (task model) and the corresponding medical guidelines (task knowledge) to check the client's personal health information (client data) and generate messages for both the client and the physician. A sample task message from this service is shown in Figure 2.8.

Case Study Specification. The COMPETE III Vascular Tracker is a decision support system that assists physicians to observe and control patient risk factors within the domains of cardiovascular, diabetes, hypertension, and dyslipidemia diseases. C3VT's database contains a large body of medical guidelines based on a methodology known as evidence-based practice. The clinical algorithms are fine-tuned to cover different cases relevant to most individual patients. The C3VT guidelines are categorized into diabetes, hypertension, dyslipidemia, coronary artery disease, cerebrovascular disease, peripheral vascular disease, and healthy. Each category has a number of corresponding guidelines that can be applied to a patient's PHR in a specific order. As a result, each medical guideline generates recommendation messages for both physicians and patients. Moreover, C3VT defines a schema for the request messages including vascular-related PHR information such as blood pressure, HBA1C results, eye exam, weight, and diet that must be provided by a user interested in using this CDSS. In this case study, we use the service representative to apply the medical guidelines (received from C3VT) to a local PHR at the client side.

Service Client. The client application sends a request message to receive medical advice and recommendations. The request message does not contain the patient's information and only identifies the category of C3VT supported diseases that apply to the specific patient. The client

[2]Compete III Vascular Tracker website http://www.compe testudy.com/

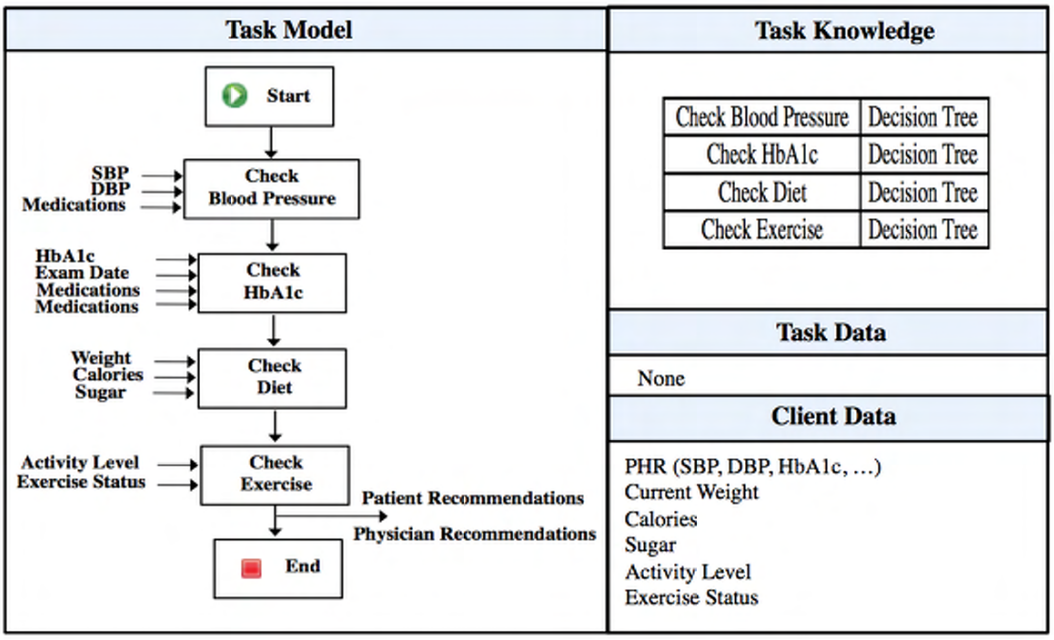


Figure 2.8. Task components of the CDSS agent and the corresponding client data. The corresponding decision tree to the "check HbA1c" step is shown in Figure 2.9. Other task knowledge components are defined similarly.

supplies the patient's information to the communication channel (Figure 2.10), based on the C3VT schema published on the service registry. There are two ports in the communication channel that allow the patient and physician to receive medical recommendations and alerts from the service representative.

Service Provider. The service provider that was developed is a modified version of the C3VT, to offer a high level of privacy for clients. Task components in the case of diabetic patients are displayed in Figure 2.8. The corresponding process defines a sequence of medical guidelines that are applied to the relevant patient's PHR. Moreover, it specifies the required knowledge model in each step of the process, including the type of model (decision tree), model inputs (relevant PHR information), and model outputs (patient and physician recommendations). In other words, it specifies each knowledge model by defining its signature. Task knowledge is provided as medical guidelines, where each guideline is encoded as a decision tree. A corresponding decision tree for a set of C3VT medical guidelines is displayed in Figure 2.9. This guideline gives recommendations to both patient and physician about the result of a blood test (Hb1Ac), when considering three patient PHR fields. The decision tree parameters, including the decision and split nodes information (model content), are serialized into the knowledge segment of the response message.

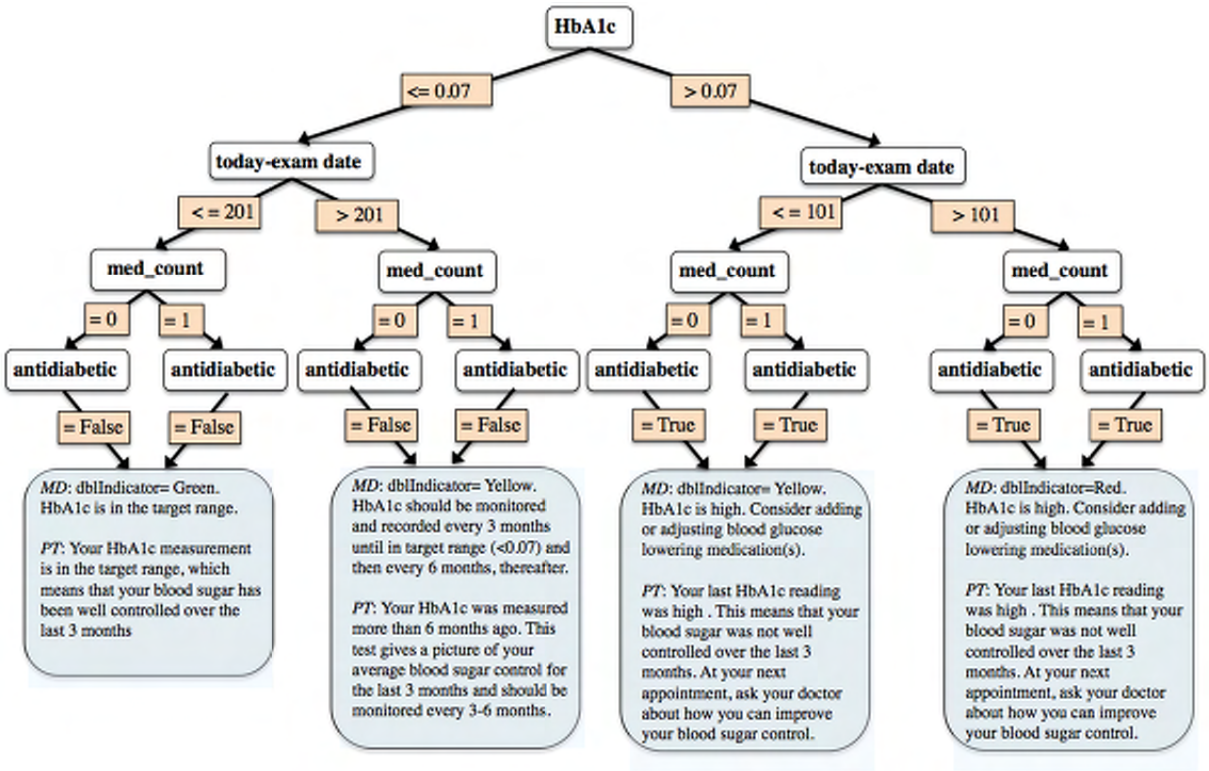


Figure 2.9. Decision tree representing a C3VT medical guideline that corresponds to one step of the process described in Figure 2.8. Decision tree parameters are modified to make sense to readers.

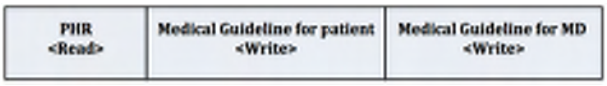


Figure 2.10. Communication channel schema in the agent-based CDSS.

Service Representative. In the customization phase, the SR instantiates an abstract process that is based on the task model. To complete the customization phase, the SR assigns one decision-tree builder to each subprocess. Also, the received knowledge is stored in the SR knowledge base so it can be reused in future service calls. In the training phase, each decision-tree is reconstructed based on the knowledge received, to represent an executable medical guideline at the client side. Finally, the SR applies each decision tree to the patient's PHR and the outcome (recommendations) are written to the communication channel.

2.4.3 Case Study: Customizable Credit Card Fraud Detector

This section presents a case of fraud detection in the context of credit card transaction systems. Legal or fraud patterns in credit card transactions can be identified by either symbolic or numerical models. A symbolic approach uses known fraud patterns while a numerical model uses a neural network to classify the transactions. In general, a sophisticated fraud detector system requires a large number of training instances from different locations of the covered region which may have different patterns of fraud. In such cases, a fraud detector that is customized, based on local data, seems to be more accurate for small and medium size organizations such as small banks or insurance companies.

This case study describes a fraud detector web service that takes local data into account to verify credit card transactions. Based on the proposed model, a service provider can use the service representative to build a customized fraud detection model at the client side, as follows: A service client gives permission to the service representative to read the local transaction information via the communication channel and sends a request message to receive fraud detector service. After processing the client request, the service defines a task for the service representative indicating a fraud detection procedure (task model) and the corresponding symbolic fraud detection model (task knowledge) as well as guidelines to build a local numerical fraud detection model (task knowledge) to verify local credit card transactions (client data). In this case study, both *rule-based* and *mining-model* knowledge are used to train the service representative. Related task components are shown in Figure 2.11.

Case Study Specification. Each transaction is represented as a tuple x of features ($x =< x_1, ..., x_n >$). Features can be symbolic (e.g., type, address) or numerical (e.g., time, money). Consequently, symbolic and numerical fraud detectors operate on symbolic and numerical features, respectively. Two metrics are usually used to evaluate a fraud detector system: *precision* - indicating the number of fraud transactions found relative to the total number of transactions tested; and *confidence* - indicating the accuracy of the method. The symbolic model offers high precision and the numerical model yields higher confidence. A sequential combination of these models is proposed by Brause et al. [1999] to provide both high precision and confidence. Instead of applying a general fraud detector model to a target transaction at the provider side, the proposed approach uses the service representative agent to customize and apply a fraud model to local transactions at the client side.

Service Client. The client application connects a read port of the communication channel to its database containing the log of collected local transactions (training data). The target transac-

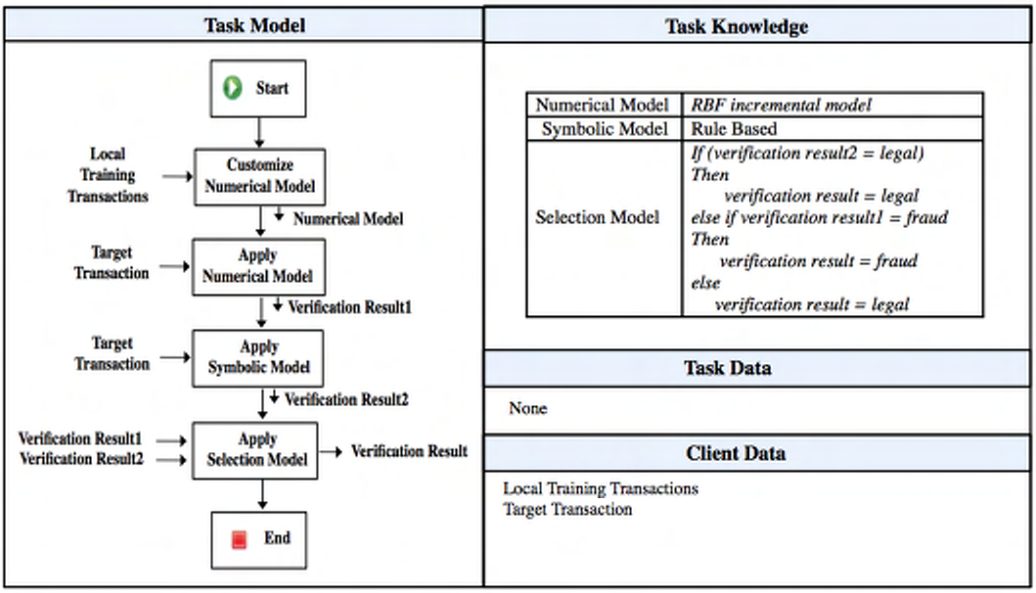


Figure 2.11. Task components for the service representative in the customizable fraud detector. Figure 2.13 represents two symbolic fraud patterns which can be easily transformed into a rule-based model.

tions to be checked for fraud (testing data) are also provided to a read port of the communication channel. There is also a port where the service representative writes the results of local transactions verification for the client. The structure of the communication channel is shown in Figure 2.12. In this case, the client application sends a request message to the service provider which does not include local transaction information.

Service Provider. The service provider is a modified version of the sequential fraud detector presented in Brause et al. [1999], as follows:

1. The service defines a task for the service representative to customize a numerical model using client data, and then applies this model to target credit card transaction data. The SR is asked to reconstruct a symbolic model from the model parameters received and then apply this model to local transactions. The final verification is obtained by a selection model. Figure 2.11 illustrates the task components where the business process includes one mining and two rule-based knowledge models as follows.

    - *Model I* is an incremental Radial Basis Function (RBF) model (Orr [1996]) that represents a numerical fraud detector. This model is generated and customized at the provider and client side, respectively.

Previous Transactions <Read>	Target Transactions <Read>	Verification Results <Write>

Figure 2.12. Communication channel schema in the customizable credit card fraud detector case study.

Rule	TRN_TYP	CURR_CD	POS_ENT_CD	FAL_SCOR	CDR_TYP	ICA_CD	AID_CD	SIC_CD	ACT_CD	MSG_TYP	MER_ID	CTY_1	POST_CD_1	CNTY_CD_1	CR_LMT	ATV_IND	ACCT_STAT	CTY_2	POST_CD_2	ADDR_STAT	EMIT_NBR	INST_NBR	ISS_REAS	GEN_CD
1	EA	840	*	*	EM	2768	8403184	*	0	1100	*	*	*	*	*	I	*	*	*	0	*	*	N	*
2	EA	840	*	995	EM	*	*	*	0	1100	*	*	0	*	*	I	F8	*	*	0	*	*	*	*

Figure 2.13. Credit card fraud patterns reported in (Brause et al. [1999]). Each column represents one symbolic transaction feature.

- *Model II* is a rule-based model to represent a symbolic fraud detection model.
- *Model III* is a rule-based model that acts as an arbitrator between the other two models.

2. The service generates or extracts the model content for each specified model in the customization layer, as follows.

   - *Model I*: Initiated on the basis of the provider training transactions. In this case study, each training instance is a tuple $(x_1, x_2, ..., x_8, y)$ where x_i represents the amount of money that a credit card holder spent in the i_{th} week and y represents the legal or fraud result for this instance. After training the RBF, its parameters are encoded by PMML and are placed in the knowledge segment of the response message.
   - *Model II*: Includes a number of if-then-else rules that represent the relations between the symbolic features and fraud. These rules can be based on the generalization techniques described by Brause et al. [1999] and shown in Figure 2.13. In this technique, fraud transactions are compared with each other to find similar pairs. Each pair is then merged into a generalized rule by replacing a non-identical feature by a don't-care symbol " $*$ ".
   - *Model III*: Describes a sequential combination of Models I and II that improves the performance metrics of the fraud detector. These rules were listed in Figure 2.11, where the fraud decisions by the symbolic model are checked additionally by the numerical model to increase confidence and decrease the number of false alarms.

Service Representative. After generating the abstract process in the customization phase, the SR trains the three specified knowledge models as follows. The numerical model (Model I) is initially built from model parameters received and then completed by the client training transactions received through the communication channel. The symbolic model (Model II) and the selection model (Model III) are loaded with the rules that are received. In the execution phase, the service representative applies the customized numerical model and the symbolic model to the local transactions. The final adjusted result is obtained by applying the third model, which is written to the communication channel to be used by the client.

2.5 Evaluation

To compare web services which utilize a service representative with typical server-side web services, we developed an equivalent typical web service for each of the described case studies, as follows.

1. A financial adviser web service that takes a client's portfolio and cash information and returns personalized advice to the client.

2. A clinical decision support service that takes a patient's PHR information, applies the C3VT guidelines, and returns recommendations for both the patient and the physician.

3. A customizable credit card fraud detector service that takes transaction records stored in the client database as well as the target transaction and returns the verification result to the client.

For evaluation metrics, QoS parameters were used to compare proposed client-side and traditional server-side web services. The QoS parameters for web services refer to the quality aspect of a web service. These parameters are used as constraints when a service client searches for the best service. Service Level Agreements (SLA) are also defined, based on the QoS parameters. These may include performance, availability, scalability, accuracy, accessibility, security, privacy, throughput, and network-related QoS requirements.

Typical web services are differentiated from proposed services by the platform where the client data are processed. While the former integrates all the processing at the server platform, the latter distributes processing between the server and client platforms. This directly affects the performance parameters (e.g., response time), network-related QoS metrics (e.g., message size), and client privacy. There are also QoS parameters which depend on the performance and network parameters such as throughput, scalability, and capacity. However, other QoS metrics that are independent of client-side or server-side processing of client data include accessibility, security,

accuracy, and availability. Three service parameters were used as evaluation criteria: *service message size*, *service response time*, and *client privacy*, because they are representative of QoS comparison for client-side and server-side web services.

Client privacy is defined as the client ability to keep sensitive and confidential data local and private. The proposed web services process confidential client data locally using the service representatives, while the traditional web services process confidential client data at the provider side. The comparison results are illustrated in Table 2.1.

Table 2.1. Client privacy comparison results.

Privacy	**Proposed Web Service**	**Traditional Web Service**
Case Study 1	$\surd$	$\times$ (revealing financial information)
Case Study 2	$\surd$	$\times$ (revealing PHR information)
Case Study 3	$\surd$	$\times$ (revealing credit card information)

Message Size (MS) is the total size of service request and response messages defined for a web service "s" as follows.

$$MS(s) = Size_{Request}(s) + Size_{Response}(s)$$

Traditional approaches require transferring complete client data from service clients to service providers. On the other hand, the proposed web services process client data locally, implying that MS is independent of the size of the client data. Table 2.2 illustrates the Message Size comparison of the traditional and proposed web services for the described case studies.

Table 2.2. Message Size comparison results.

Message Size	**Proposed Web Service (KByte)**	**Traditional Web Service (KByte)**
Case Study 1	8	11
Case Study 2	12	7
Case Study 3	6	835

Based on Table 2.2, the traditional and proposed web services represent compatible Message Sizes for the first and second case studies. However, the proposed approach outperforms the

traditional approach in the third case study where the client has to send the entire local database to the service provider in order to receive customized verification results.

The proposed approach improves the *Total Message Size (TMS)* significantly, which represents the total size of service messages where the same service is called multiple times by a service client. In traditional web services, the TMS is calculated by multiplying Message Size by the number of service calls. In the proposed web services, the received knowledge is stored in the SR knowledge base, resulting in a reduction in the size of the response messages. Figure 2.14(top) compares the Total Message Size of the proposed and traditional web services for each case study.

Response Time (RT) is divided into two factors: Network time (N) and Processing time (P) and is defined for a web service "s" as follows.

$$RT(s) = N(s) + P(s)$$

Network time is the amount of time required to transfer request and response messages. This depends on both network bandwidth and message size. Processing time is the amount of time it takes a web service to perform its designated task. Since service providers use more powerful CPUs, traditional approaches have less process time. On the other hand, the proposed web services require smaller messages, resulting in less network time.

For this case study, the processing time, $P(s)$, was obtained for the three case studies using a 2.4 GHZ Intel dual-core CPU. The service provider was assumed to have a CPU that is twice as fast as the service client. Finally, a 128 KByte/Sec link connects the service client to the service provider.

Table 2.3 shows the Response Time comparison between the proposed and traditional web services. These results show the proposed approaches perform better than the traditional approaches as the amount of client data increases.

Table 2.3. Response Time comparison results.

Response Time	Proposed Web Service (msec)	Traditional Web Service (msec)
Case Study 1	131	112
Case Study 2	125	75
Case Study 3	207	6612

The Total Response Time (TRT) of the proposed and traditional web services was also compared in the context of these case studies. This comparison, illustrated in Figure 2.14 (bottom), confirms that client-side processing of client data improves the TRT.

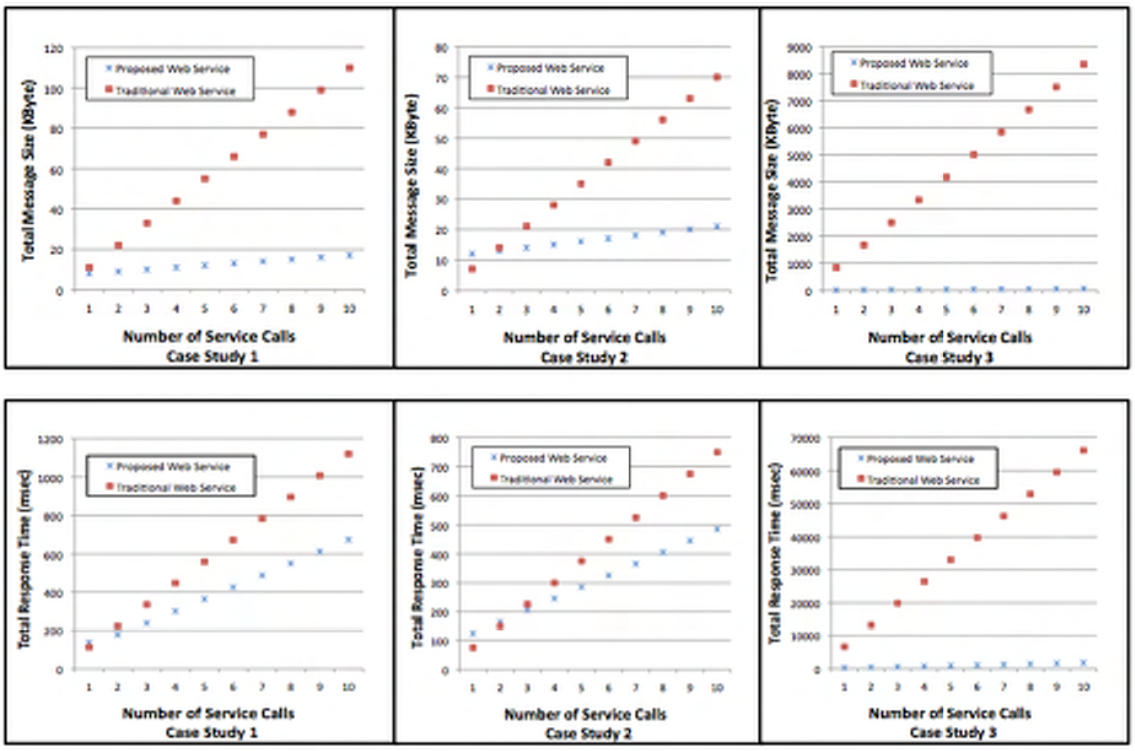


Figure 2.14. (Top) Total Message Size and (bottom) Total Response Time comparisons of the traditional and the proposed web services where they implement the described case studies.

2.6 Related Work

Web services have had growing success and broad acceptance by developers and users of enterprise systems. However, there are still a number of impediments that limit the wide applications of web services in industry for which agent-based techniques seem to be proper solutions for enabling dynamic collaboration among e-Business systems. Therefore, there is a growing demand for using agent-based approaches that can evolve the current architecture of SOA in several aspects, as follows:

1. **Agents as services**. The Intelligence Service System (ISS) (Alessandrini et al. [2008]) is introduced as a framework for integrating expert systems into service oriented landscapes. In this framework, a computerized expert system (intelligent agent) acts as a service, which receives requests (including query and training data) from business applications. By using the training data, the expert system is trained and returns its response to the query. Since agent platforms (Sycara et al. [2004]) and web service platforms have similar components (registry, descriptor, communication protocol and semantic language), AgWebs architecture (Lee et al. [2007]) is proposed to provide interoperability and interaction between them.

2. **Services as agents**. In ASMF (Cheng et al. [2008]), a network of web services is modeled by a number of autonomic agents (each service is wrapped into an agent). Furthermore, these agents interact with each other to form service relationships. In addition to service agents, service brokers are designed as autonomic elements. Xu et al. [2008] adapted a role-based architecture to facilitate service definitions and relationships among SOA components.

3. **SOA related tasks by employing agents**. Agents have been used to facilitate SOA related tasks such as service composition and service negotiation. For example, during a service composition process, software agents can engage in conversations with their peers to agree on the web services that participate in this process (Maamar et al. [2005]). Moreover, agents have been proposed as coordinators for web services. For example, Yamato et al. [2008] introduced a service processing agent that searches, selects, and invokes service components for a service composition, dynamically and according to the user's context.

4. **Agent-based enterprise modeling**. Integration of agents and web services has been proposed to model the business aspects of enterprise systems. Xiang [2007] considers each role or major function of an enterprise system as an agent (e.g., supplier agent, producer agent, cooperative agent, information service agent, and customer service agent). Then each agent is wrapped into a web service. The agents, when combined with web services, can easily communicate with each other. As another example, a distributed market place is modeled by agents (Khalili et al. [2008]). In this approach, service providers and clients are considered as sellers and buyers, where an agent models each buyer or seller. These agents can negotiate with each other until they reach an agreement.

In the proposed model, the thesis addresses a new application of collaboration between agents and services. The proposed generic agents are in charge of delivering the functionality of service providers, but these agents are located at the client side.

Mobile agents can physically travel across a network and perform tasks at different nodes. Agent mobility requires facilities that convert an agent into a form suitable for network transmission (e.g., messages) and, on the receiving end, allow the remote system to reconstruct the agent. Java's object serialization accomplishes this conversion and reconstruction. Concordia (Wong et al. [1997]), Odyssey (Wong et al. [1999a]), and Voyager (Pham and Karmouch [1998b]) are examples of mobile agent frameworks based on Java. Also, mobile agents are suitable for formal representation using pi-calculus (Sangiorgi and Walker [2001]). There are several security and privacy issues to be considered in mobile agent-based computing, which motivated this research in customizing generic resident agents as opposed to transmitting mobile agents. Tasks differ from mobile agents in three aspects: (i) a mobile agent is a complete process while a task represents an

incomplete process which is completed with local resources at the client side; (ii) a mobile agent is serialized into byte codes to be transmitted over networks, while task components are messages by their definitions; and (iii) task processes are compatible with traditional web services as they are developed based on web service protocols and standards.

2.7 Conclusions

In this chapter, a novel model for SOA based systems was presented that enables enterprise organizations to delegate their agents to operate on the client platform. In the proposed model, a generic client-side service representative applies the knowledge that is received from the service provider to the client data and delivers the requested information to the client. To support this model, an architecture is proposed that introduces an executable platform at the client side for service providers to enhance the privacy and security aspects of web services. The next chapter will introduce a novel type of web services, called *Task Services*, which return task messages that employ the service representative to process client data locally. Task services are compatible and composable with typical web services. The proposed service representative can be employed by collaborating service providers to perform a composite task at the client side. This motivates an extension of the model to introduce the concept of client-side service composition.

Chapter 3

Task Services: Client-side Web Services Using Generic Service Representatives [1]

A business web service is a traditional web service with additional requirements such as *security* and *privacy*, *response time*, *required bandwidth*, *modularity*, and *composability*. Different types of web services are required to model actual services in the business domain. As we discussed in Chapter 1, the enterprise agent has not been previously modeled efficiently in SOA. Consequently, services provided by current SOAs are limited to services which process client data completely at the server side. To provide a comprehensive model for enterprises, SOA needs to model web services with the capability of client-side processing of local data.

The generic service representative was proposed in Chapter 2 to simulate an enterprise agent operating at the client side. The proposed service representative is a generic software agent that is located at the client side and can be customized along with service messages to perform a task (i.e., client-side processing). In this chapter, we use the service representative to propose a new type of web services, called *Task Services*, which return task messages that customize the generic service representative to perform required client-side processing. While a traditional web service, called a *Data Service*, processes client data completely at the server side, a task service is a web service with the capability of processing client data and resources partially or completely at the client side, using the Service Representative.

The proposed task services are implemented using message exchange technology as well as a protocol stack similar to that used in traditional data services. Consequently, task services are compatible with data services and can be composed with each other through a BPEL (Business Process Execution Language) process. The proposed model improves the current state of SOA in

[1] This chapter is mostly based on the following paper: M. Najafi, K. Sartipi, and N.Archer, *Task Services: Client-side Web Services Using Generic Service Representatives*, IEEE Transaction on Service Computing, 2011 (accepted with revisions).

terms of maintaining client privacy and security, and reducing network traffic, as well as offering a client-side service composition solution. In this chapter, the thesis compares the proposed client-side task services with traditional server-side data services as well as other client-side processing technologies (scripting and Rich Internet Application) through a case study.

The organization of this chapter is as follows. Task services are introduced in Section 1. The task service protocol stack is described in Section 2 and the proposed architecture is discussed in Section 3. We evaluate task services using a case study in Section 4. The discussion (Section 5) explores the applications and challenges posed by the task services approach. Section 6 discusses work related to our approach and finally, conclusions are discussed in Section 7.

3.1 Business Web Services

A web service is a computer program which is accessible through the web using message exchange technology. In other words, a web service can be modeled by a function which receives some parameters from a client (through a request message), processes them, and returns the resulting responses to the client (through a response message). To reflect real business services, the required service parameters are distributed between the client and provider sides. Moreover, the resulting service responses are delivered by the provider or generated at the client side. Therefore, we propose to classify the service parameters and responses into two types: *local* and *remote*, as follows.

- *Local Service Parameter*: processed locally at the client side.
- *Remote Service Parameter*: processed remotely at the provider side.
- *Local Service Response*: generated locally at the client side.
- *Remote Service Response*: generated remotely at the provider side.

In this section, web services are categorized into server-side data services and client-side task services.

3.1.1 Server-side Data Service

Server-side data services represent typical web services in the current SOA model where the service processes the client data using resources completely at the server side. Consequently, a data service includes only server-side processing and returns a service response in the form of data that will be consumed directly by the client. In other words, a data web service (DWS) is modeled by a function that receives the required remote service parameters (RSP_{DWS}) from the

service client and returns the resulting remote service responses (RSR_{DWS}) to the service client, as follows.

$$DWS : RSP_{DWS} \longrightarrow RSR_{DWS}$$ [2]

3.1.2 Client-side Task Service

This thesis introduces the concept of *Task Services* as web services with the capability of processing the client data and resources partially or completely at the client side. Based on the received service parameters, a task service defines a *task*, including client-side processing, to be performed by the Service Representative at the client side. In this context, the service representative simulates an enterprise agent which operates at the client-side on behalf of the enterprise.

As we discussed in Chapter 2, the service representative is modeled by a generic and client-side software agent with a built-in process engine that can be customized and employed by different service providers to perform different task services. Moreover, we propose to define a task with three components: *Task Model* specifies a client-side processing task using an abstract business process model; *Task Knowledge* provides the required logic to perform the task including business rules and actions; and *Task Data* provides the server-side business data that are consumed by the business rules and actions during the business process execution. Finally, the service representative has an internal knowledge base to provide part of the required task knowledge locally, and the required client-side business data are provided locally by the service client.

The service representative uses the received task components to perform client-side processing of the task service on *local service parameters* in order to provide *local service responses* for the service client. Consequently, a task web service (TWS) is modeled by a function where it receives the required remote service parameters (RSP_{TWS}) from the service client and then it returns the resulting task message ($Task_{TWS}$) to the service representative (SR) to be customized (SR_{TWS}) so it can process local service parameters (LSP_{TWS}) and generate local service responses (LSR_{TWS}).

$$TWS : RSP_{TWS} \longrightarrow Task_{TWS}$$
$$SR_{TWS} : Task_{TWS} \times LSP_{TWS} \longrightarrow LSR_{TWS}$$

Figure 3.1 shows the related comparison between data services and task services.

[2] In defining a function (e.g., DWS), we use the sets of "service parameters" (e.g., RSP_{DWS}) and "service responses" (e.g., RSR_{DWS}) to represent the types of function input and output.

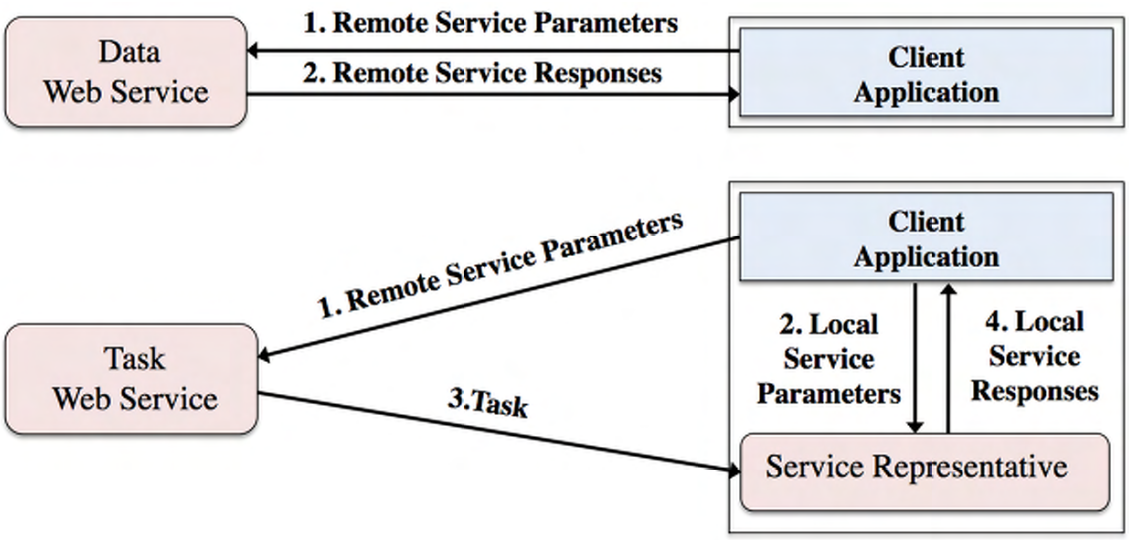


Figure 3.1. Comparison of Data and Task Services. Remote and Local are defined from the viewpoint of the service client.

3.2 Task Service Protocol Stack

A web service *protocol stack* includes networking protocols that are used to define, locate, implement, and compose web services. The proposed task services are invoked by message exchange technology and use the same protocol stack as data services. Consequently, they are compatible with traditional data services.

- *Transport Protocol*: used in communications between task service clients and task service providers. Standard protocols such as HTTP, SMTP, and FTP can be used as the transport protocol for task services.

- *Messaging Protocol*: used to define the request and response message templates. Similar to data services, task services use SOAP as their messaging protocol. In both cases, the SOAP request message includes the required remote service parameters. However, while the data service returns a SOAP response message including the resulting remote service responses, the task service returns a SOAP response message including the resulting task components.

- *Description Protocol*: used to describe web services including the service location and the operations (or methods) the service exposes. Similar to data services, task services use WSDL as their description protocol. Figure 3.2 compares the data and task services in terms of their SOAP and WSDL protocols.

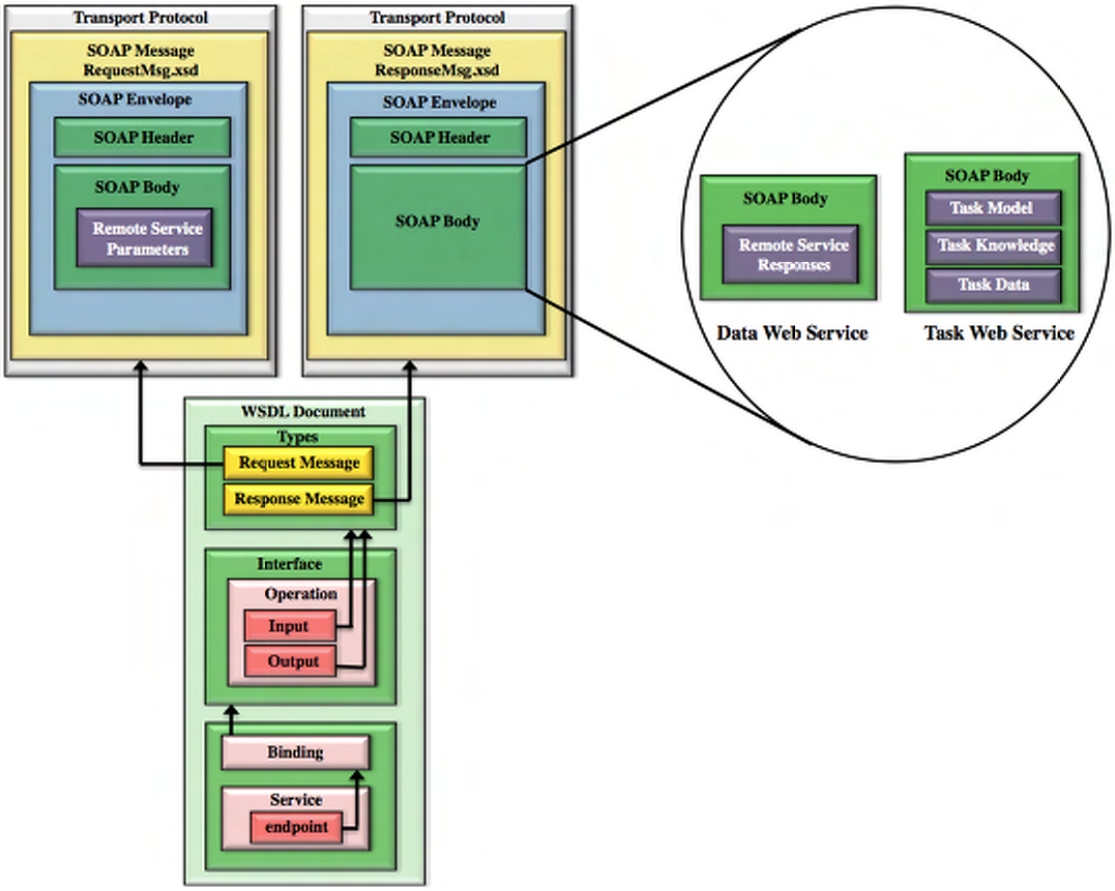


Figure 3.2. SOAP messages and WSDL documents for data web services and task web services with the differences highlighted.

- *Discovery Protocol*: a set of service descriptions (e.g., service locations, and WSDL documents), which is published by service providers to a common registry for use by service clients. As in the case of data services, UDDI is used for task service discovery. UDDI includes three components: white pages (information about the business supplying the service), yellow pages (service categories) and green pages (technical information describing how to access the web service). White and yellow pages for task services are similar to those used for data services. However, in addition to the typical technical information about services (e.g., address of the service, service parameters, and service interfaces), the green pages for a task service include the service representative interface. This includes the required local service parameters as input and the resulting local service responses as output. The extra information is organized by *tModel* (Curbera et al. [2002]) components. A tModel is a data structure representing a service type in the UDDI. Any abstract concept can be registered within UDDI as a tModel.

The web service protocol stack also includes a range of recently defined protocols such as BPEL, WS-Security (Atkinson and et al [2002]), and Rest (Richardson and Ruby [2007]) which can be used directly by task services. The BPEL standard, which describes task service composition, is introduced in the next Chapter.

3.3 Architecture

Figure 3.3 represents the proposed SOA architecture, extended to support both data and task services. This architecture enables the composition of both data and task services. A composite web service is defined by a process which is executed by a service orchestrator and includes a number of data and task service invocations in a defined order. The proposed architecture includes four main components, described below.

3.3.1 Service Client

The service client consists of a client application and a communication channel as follows.

Client Application. This is a traditional client application that sends a data or task service request (including remote service parameters) to a simple service provider or a service orchestrator. In the case of a simple data service, the client application receives the remote service responses directly from the service provider. However, in the case of a simple task service, the client application puts the required local service parameters into the communication channel and waits to receive the local service responses from the service representative through the communication channel. Finally, in the case of a composite service, the client application submits its remote service parameters to the service orchestrator and then receives the remote and local service responses from the service orchestrator and the service representative, respectively.

Communication Channel. This component has a port for every local service parameter as well as for every local service response. In other words, each port is a connection link which enables the client application and the service representative to communicate with each other. The communication ports can be input, output, or bi-directional (from the client's point of view). Moreover, each port is assigned a scope as public or private. The contents of a public port can be sent to a service provider as service parameters while the contents of a private port can only be used locally by the service representative. The communication channel schema for a specific task service is specified by the service description published in the service registry.

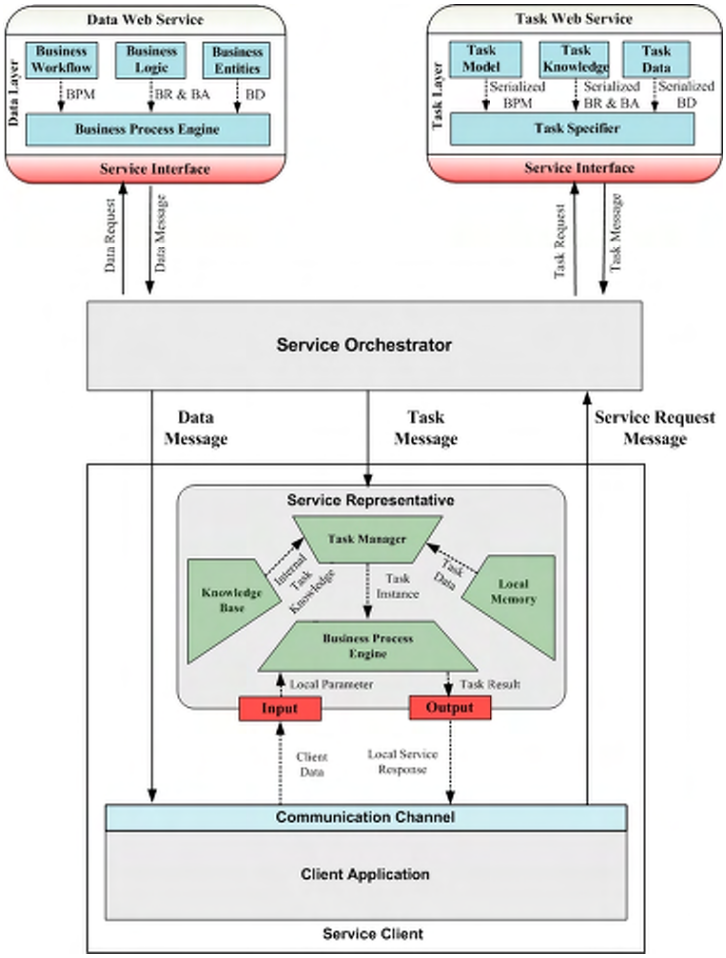


Figure 3.3. Extended SOA architecture. Based on the client's request message, the service provider generates a 3-segment response message to customize and train a client-side generic agent as its representative to serve the client.

3.3.2 Service Provider

A web service executes one or more business processes of its enterprise (i.e., the service provider), where each business process applies business rules and performs business actions on internal (server-side) and external (client-side) business objects in a defined order. Therefore, an enterprise can be modeled by a collection of business components which are business processes, rules, actions, and objects. On the other hand, a business process can have server-side and/or client-side processing. Accordingly, the enterprise provides two types of services and the enterprise business components are divided between these two services. While a data service applies the business components at the server-side, a task service sends the business components to the client-side to be applied by the service representative.

Data Service. The service interface supports communication contracts (message-based communication, formats, protocols, security, exceptions, etc) for the services. Moreover, server-side business processes, rules and actions, and data are stored in the business workflow, logic, and entity components, respectively. The business process engine executes the corresponding business process with each service. This service responds to the client requests with a single-segment response message (a data message).

Task Service. While the task service interface is similar to the data service interface, the client-side business processes, rules and actions, and data are stored in the task model, task knowledge, and task data components, respectively. Since business components in this layer are sent to the client side, they must be serializable. The task specifier provides the required model, knowledge, and data for each task request to be sent by a three-segment response message (the task message) to the client.

3.3.3 Service Representative

The proposed service representative is modeled by a software agent whose components are introduced below.

- *Input*: inputs client data (local service parameters) through the communication channel.
- *Knowledge Base*: receives task knowledge (extracted from a task message). It includes internal and domain-based business rules and actions, to relieve the service provider from sending them each time. For this purpose, versioning techniques can be used.
- *Business Process Engine*: executes a task instance by applying business rules and performing business actions.
- *Local Memory*: receives and stores task data (extracted from a task message) which are invoked by the business process engine during task execution.
- *Task Manager*: supports the entire life cycle of a task instance, from creation to termination, divided into two phases as follows.

    1. *Task Instantiation*: creates an abstract business process based on the task model and then realizes the abstract process using internal and external task knowledge to generate a task instance. A task is instantiated during the service representative customization and training phases.

2. *Task Execution*: passes the task instance with the relevant business data (received from the task data segment) to the business process engine to be executed. Finally, the task manager sends the task results to the output component. A task is executed during the service representative execution phase.

- *Output*: outputs task responses (local service responses) to the client through the communication channel.

3.3.4 Service Orchestrator

The service orchestrator has a built-in BPEL engine that executes the BPEL process representing a composite service by invoking the partner data and task services through dedicated service stubs, as follows.

- *Data Service Invocation*: the service orchestrator sends the remote service parameters and receives the remote service responses which will be forwarded to the client application.

- *Task Service Invocation*: the service orchestrator sends the remote service parameters and receives the resulting task message which will be forwarded to the service representative to generate the local service responses for the client application.

3.4 Evaluation

In this section, the proposed client-side task services was evaluated with both traditional server-side data services and other client-side processing technologies, applied to a case of face detection.

3.4.1 Test Case Specification

A face detection service is a primary need in many fields, including face recognition, video surveillance, and human motion detection. A face detector takes an image and determines its regions that contain face(s). Several face detection approaches have been proposed in the literature (Jafri and Arabnia [2009]), including approaches based on face spaces, neural networks, and template matching. The following two-step template matching algorithm was adapted to be used in this case study.

1. Find skin regions in this image, using explicit boundary rules on color values (Kakumanu et al. [2007]). This step generates a binary image where all *skin* and *non-skin* pixels are assigned *Black* and *White*, respectively.

Figure 3.4. Face detection algorithm applied to a sample image. (Left) original image, (middle) skin-detected image, and (right) face-detected image.

Table 3.1. Skin and Face Detector Web Services

Data Services
$\mathbf{SkinDetector_{DWS}}(original\ image) = binary\ image$ $\mathbf{FaceDetector_{DWS}}(binary\ image) = face\ rectangles$
Task Services
$\mathbf{SkinDetector_{TWS}}() = task_{SD}$ $SR(task_{SD}, original\ image) = binary\ image$ $\mathbf{FaceDetector_{TWS}}() = task_{FD}$ $SR(task_{FD}, binary\ image) = face\ rectangles$

2. Extract face regions from skin regions using face templates, as follows: (i) enhance the binary image to eliminate noise; (ii) segment the enhanced image into connected regions; (iii) select the potential face regions based on their size and width-to-height ratio; (iv) create a feature vector for each selected region by detecting its edges; (v) compare feature vectors with the given face templates to find face regions; and (vi) show face regions in rectangles.

Figure 3.4 represents an example that applies these two steps to a sample image. The test set, which was used in our experiments, includes RGB images taken from a stream video. The size of each image is between 750 KB - 1.1 MB. The client is connected to the server via a 1 Mbyte/Sec connection link and has a (2.66 GHZ) 2 Core CPU. The server is considered to be twice as fast as the client system in processing client data.

3.4.2 Task vs Data Web Services

A face detector data service must transfer images and videos from the service client to the server, which is not efficient. As opposed to this server-side approach, a face detector task service, which customizes the client-side service representative to detect faces, offers higher performance. Moreover, one of the main discriminating features of web services is its composability. Therefore, the proposed face detection algorithm was divided into two services. For each service, a corresponding data and task service was developed, as represented in Table 3.1. $SkinDetector_{TWS}$

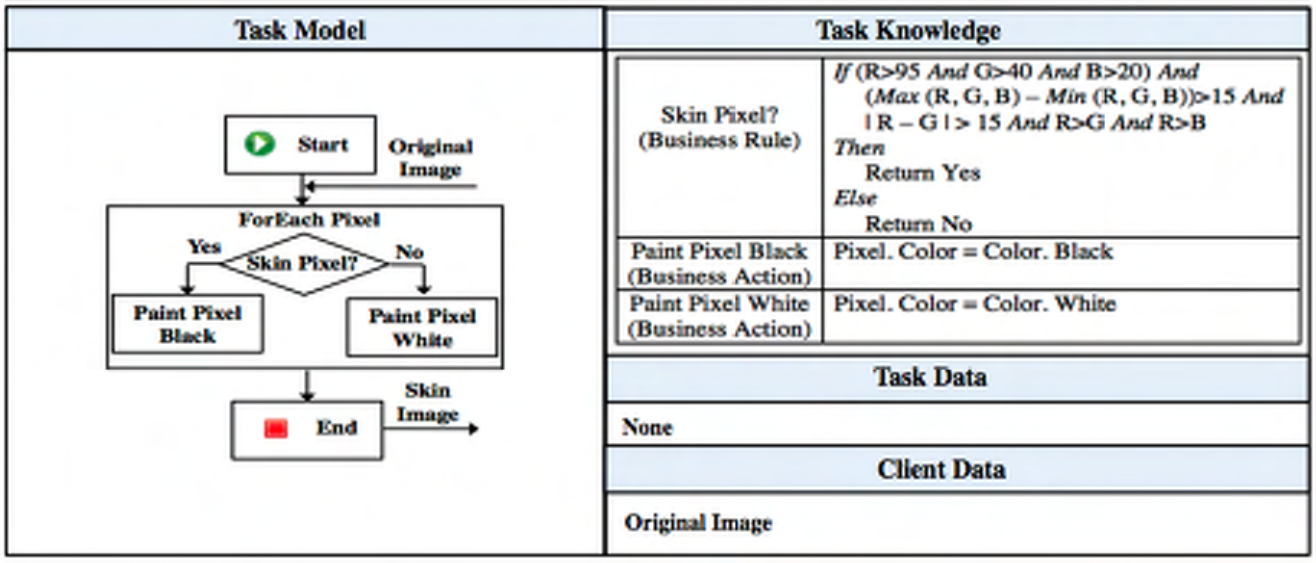


Figure 3.5. $\mathbf{SkinDetector_{TWS}}$ **task components.**

and $FaceDetector_{TWS}$ task components are represented in Figures 3.5 and 3.6, respectively. In this experiment, three different compositions of data and task services were considered. In addition to a composite data service ($SkinDetector_{DWS} + FaceDetector_{DWS}$) and a composite task service ($SkinDetector_{TWS} + FaceDetector_{TWS}$) , a hybrid approach ($SkinDetector_{TWS} + FaceDetector_{DWS}$) was evaluated where the client data is processed partially at the server and client sides. The evaluation metrics are as follows.

1) Message Size (MS) is the total size of service request and response messages. The traditional data services require the transfer of complete client data (i.e., image frames) from service clients to service providers. The service provider returns the resulting image back to the service client. Consequently, the message size will be equal to the size of the image multiplied by two ($2 \times 750\ KB$). On the other hand, the proposed web services process client data locally, implying that MS is independent of the size of the client data. In this case, $SkinDetector_{TWS}$ returns a task (4.5 KByes), including the task model (1 KBytes) and task knowledge (3.5 KBytes). Similarly, $FaceDetector_{TWS}$ returns a task (28.5 KByes) including task model (2 KBytes), task knowledge (10.5 KBytes), and task data (16 KBytes). The task knowledge is stored in the SR internal knowledge base. As a result, the service provider does not need to send it every time and the message size is reduced in future service calls.

2) Response Time (RT) is divided into two factors: Network time (N) and Processing time (P), as follows: $RT(s) = N(s) + P(s)$. Since service providers use more powerful CPUs, data

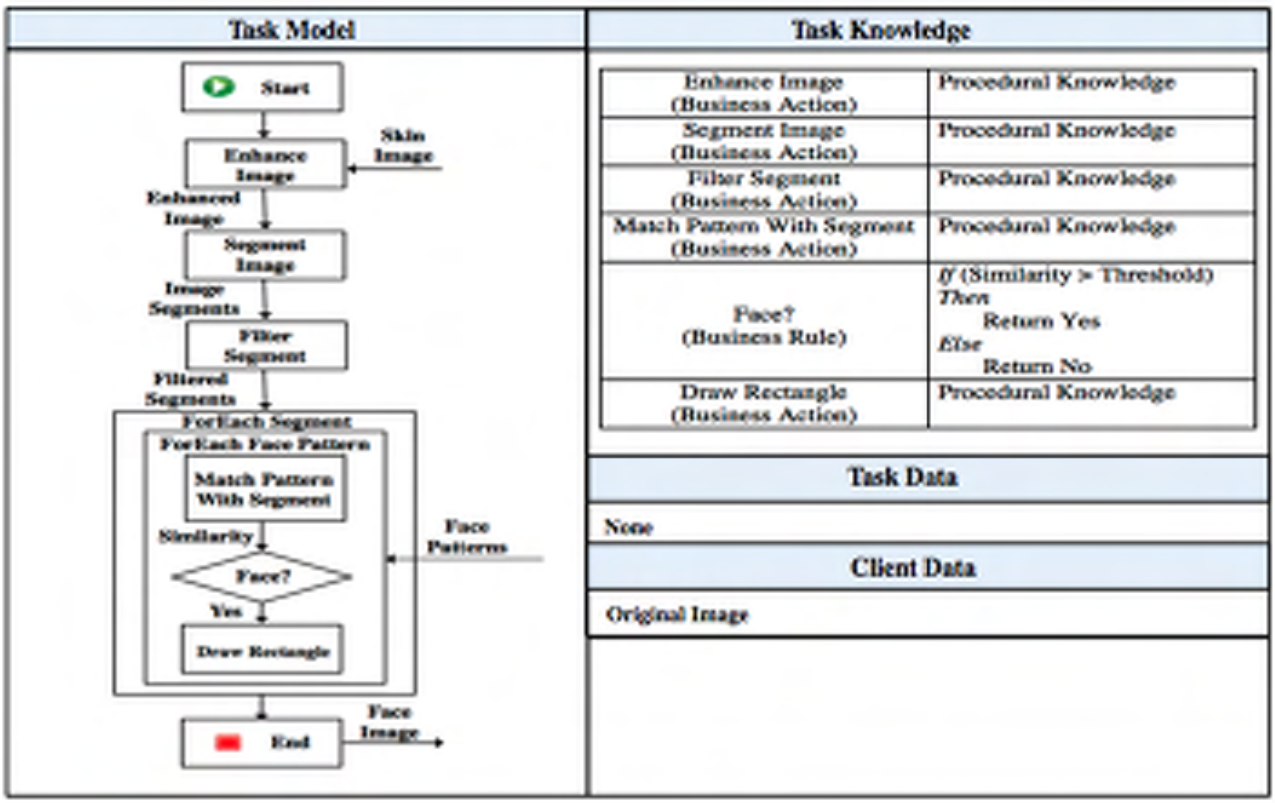


Figure 3.6. $\mathrm{FaceDetector}_{\mathrm{TWS}}$ **task components. To reduce figure size, the procedural knowledge codes are not included.**

services have less processing time. On the other hand, the proposed task services require smaller messages, resulting in less network time. Figure 3.7(left) presents a comparison of task, data, and hybrid services in terms of Response Time and different numbers of image frames. Task services outperform data services significantly as the amount of client data increases. Therefore, dividing service processes between client and server (hybrid services) can improve service response time significantly. In particular, a task service can be called to process client data initially and then the processed data sent to a data service to complete the data processing.

3) Average Response Time (ART) extends the RT metric when multiple clients invoke a service simultaneously. The ART for service *"s"* where it has *"n"* simultaneous clients is defined as follows.

$$ART(s,n) = \frac{1}{n} * \sum_{i=1}^{n} RT(s_i)$$

To compute ART for data services, we consider a non-preemptive queue for the server processor, resulting in the following equation.

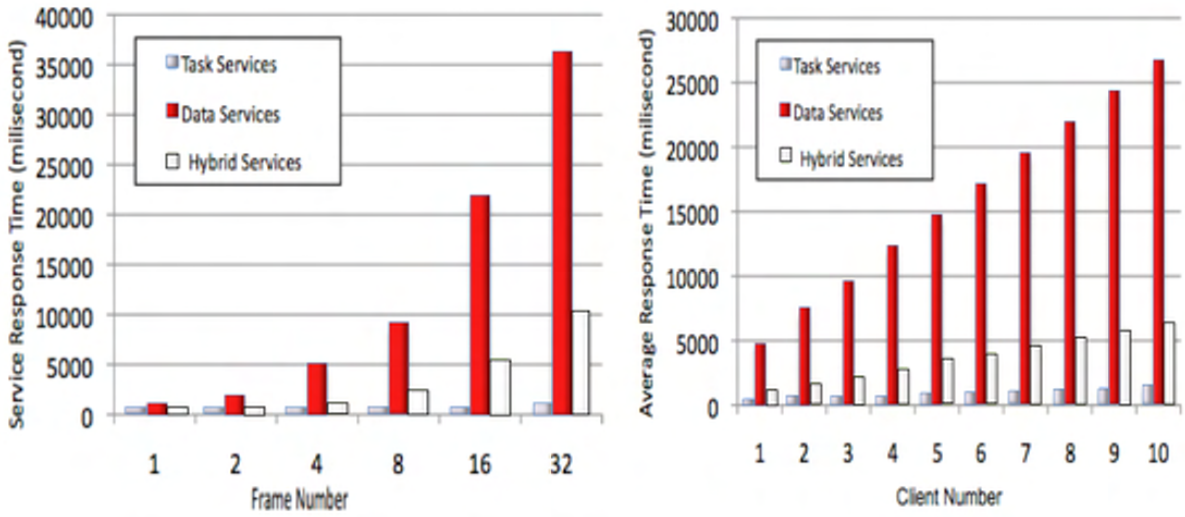


Figure 3.7. Response Time (left) and Average Response Time (right) comparisons of data and task services in the face detection case study.

$$P_{DataService}(s, n) = n * P_{DataService}(s)$$

$P(s, i)$ represents the processing time of service "s" when "i" clients call the service simultaneously. Figure 3.7(right) compares task, data, and hybrid services in terms of Average Response Time. In this evaluation, each service client calls the services to process four image frames. Since service representatives process images locally and in parallel, task services outperform data services significantly.

4) Client Privacy is defined as the client's ability to keep sensitive and confidential data local and private. In order to use data services, the service client has to send its images to the server, which violates client privacy. Task services and the hybrid approach process client images locally, improving client privacy.

3.4.3 Task Services vs Client-side Processing Technologies

There are several technologies which enable client-side processing, such as the following approaches.

- *Rich Internet application* (e.g., Microsoft Silverlight, Java Applet, Adobe Flash): an Internet user installs a plug-in in his/her browser. Then web applications can process a client's data locally by sending their codes (e.g., XAML files in Silverlight) to the plug-in.

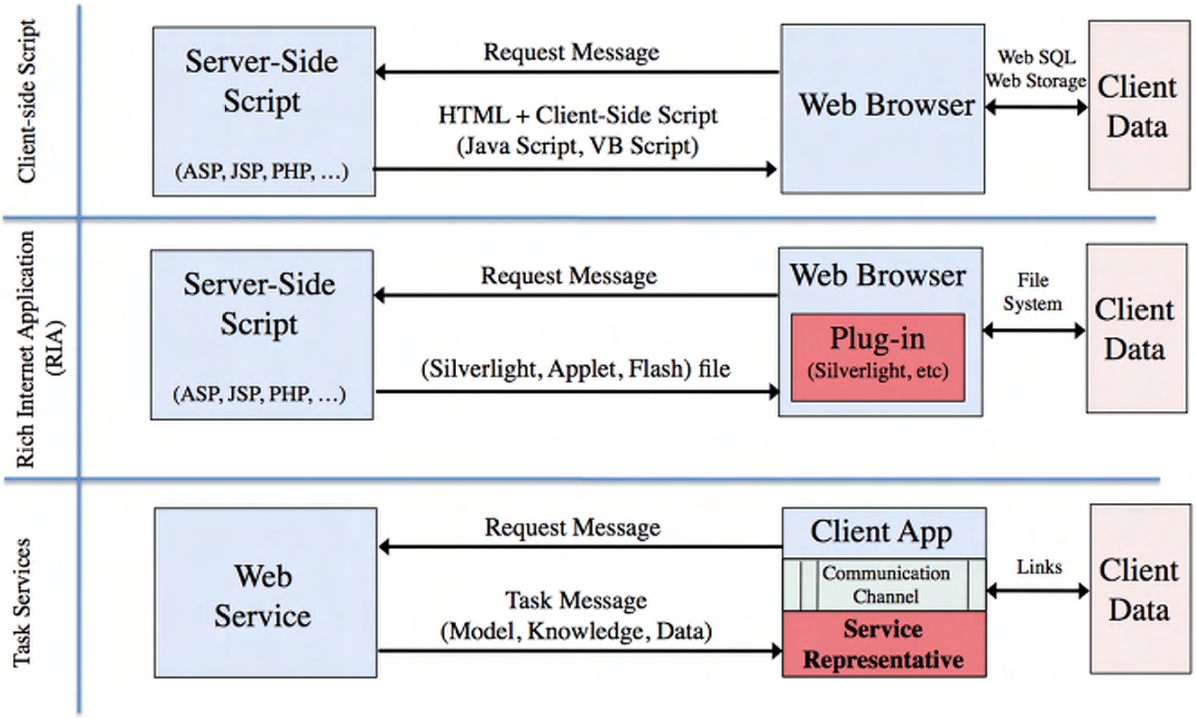


Figure 3.8. Client-side Processing Technologies.

- *Client-side Scripting* (e.g., VB Script, and Java Script): an Internet user enables a browser to run script codes. Then the web browser runs the scripts inside the HTML files to locally process client data.

The proposed task service is an alternative for these technologies in data intensive applications. Figure 3.8 compares these three approaches. Since web browsers are not intended to perform heavy computations on client data, they are not good platforms for client-side data intensive applications. Consequently, client-side scripting is only used for minor client-side processing (e.g., customizing web pages). In the following, the proposed task services was compared with one of the Rich Internet Application (i.e., Silverlight) technologies. The Silverlight plug-in loads the XAML files containing skin and face detector objects to process image frames sequentially. Similarly, the service representative receives $task_{SD}$ and $task_{FD}$ once and processes the image frames sequentially.

1) Message Size: this is comparable in both task service (33 KBytes) and Silverlight cases (21 KBytes). The Silverlight approach requires relatively smaller messages since it can take advantage of rich classes (i.e., knowledge) stored in the Silverlight plug-in. On the other hand, the service provider needs to customize the generic service representative by providing its required knowledge. However, the volume of knowledge is insignificant in comparison with the volume of client data.

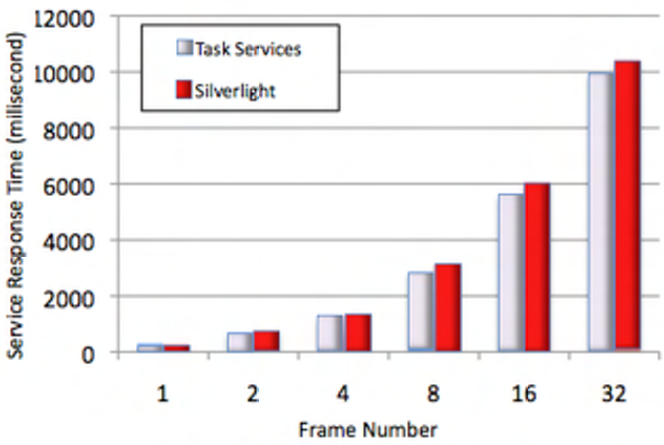


Figure 3.9. Service Response Time comparison between client-side processing approaches in the face detection case study.

2) Response Time: this is comparable in both approaches, since the client's data are processed locally using the same algorithms. Figure 3.9 compares task services with the Silverlight approach in terms of their response times. Since service representatives are customized for each task they can improve the response time (in this case up to 10%). However, the Silverlight plug-in can support different cases and different issues (especially graphics and animations) which could increase the response time. Finally, client-side processing in both approaches provides similar performance with respect to Average Response Time (ART).

3) Client Privacy: this is preserved by both approaches. The Silverlight plug-in and service representative both have access to client data through the browser and the communication channel, respectively. Consequently, the web service client and the Silverlight client do not need to send their images to the server.

4) Client Security: this is threatened by the ability of an external system to inflict malicious behaviour on the client's system. The service representative simulates an enterprise agent. A client decides about resources which it makes available and accessible to the enterprise agent. Similarly, a task service client provides the requested data in terms of local service parameters. Moreover, the service representative plug-in enables the service client to control the resources (e.g., CPU, Memory) consumed by the service representative. There is no such obligation for the Silverlight plug-in which has access to client data and events. There are a number of security and privacy regulations for Silverlight applications (e.g., sandboxes in modern web browsers), but they are out of the client's control.

5) Composability: one of the principles that is part of the service-orientated design paradigm. A task service can be composed with other task services or traditional data services using the BPEL standard. However, Silverlight processes, which are represented by XAML files, are not intended to be composed with other XAML components.

6) Discoverability: one of the principles that is part of the service-orientation design paradigm. Similar to data services, task services can be specified by service descriptions using different standards such as UDDI. This allows service descriptors stored in service registries to be discovered by other services or service clients dynamically or statically. However, Silverlight and other rich Internet applications lack a discoverability feature.

7) Reusability: refers to the quality of a software component that is sufficiently general to be used in different applications. Tasks provided by task services are composed of reusable components. Task knowledge and task data can be efficiently used by different task services. Moreover, a task model can be considered as a sub-task model of a composite task service. However, Silverlight XAML files are not reusable resources in different applications. Similarly, task services address modularity more efficiently than Silverlight.

In client-side data intensive applications, the major component of the service response time is the network time which is very small and can be neglectable using client-side technologies. As it is represented in Figure 3.8, all rich Internet applications use client-side plug-ins and server-side scripts to locally process client data. As a result, by using similar processing logic, other rich Internet applications would represent comparable response time to Silverlight. However, similarly to the Silverlight technology, Java Applet and Flash technologies are neither discoverable nor composable.

3.5 Discussion

The proposed task services are differentiated from traditional data services, as they can process client data locally using local and generic service representatives. In this section, a few important issues are listed, such as the applications and challenges of task services.

3.5.1 Applications

The proposed web services are not intended to replace traditional web services. However, web services can be developed efficiently using service representatives in the following cases.

1. The two types of web services (data and task services) offer a tradeoff between client-side complexity and the importance of local processing of client data. Consequently, task service

applications include client data that should remain at the client side since they cannot be transmitted to the server for reasons of *confidentiality, real time response requirements, or being too large to transmit efficiently.*

2. If a *context-aware web service* can be modeled as a pair (general service response, customization knowledge), it is eligible to be delivered by the service representative. This improves the privacy and security aspects of these services. Specifically, in a dynamic environment, where the client's context (e.g., location) is changing frequently, traditional data services must be called for each change, which increases network traffic as well as service cost. In contrast, the service representative applies customization knowledge to the general service response and generates dynamic service responses for each change in context. Najafi and Sartipi [2010] presented a case of this type of application.

3. *Dynamic web services*, where the provider's knowledge is changing over time, can be developed efficiently by task services. The service representative enables services to separate enterprise knowledge from service implementation and facilitates change management.

4. *Mobile applications*, where the service representative is developed as a mobile App, can be installed on mobile devices. The developed service representative prototype (Appendix A) requires a relatively small memory (a few MBytes) and offers reasonable computational speed. Consequently, it can be used in mobile devices. Thanks to mobile devices and their application platforms, even smart phones can become servers and provide web services. Therefore, a mobile device can provide the service representative as a local service to be used by task services. As a result, the contextual information of the mobile user is processed locally and its privacy improved. Moreover, the service representative can provide services offline for mobile devices.

3.5.2 Challenges

Although the distributed service processing offered by the service representative improves SOA performance in several cases, it imposes a number of challenges for both the service client and the service developer, as follows.

1. *Service Provider Privacy.* Required knowledge for the service representative can include enterprise assets and resources. Revealing them may violate enterprise privacy. To prevent this security vulnerability, a service provider can use one of the following techniques.

    - Enterprise knowledge can be divided so it can be applied locally (at the provider side) by the service or externally (at the client side) by service representatives. Therefore, the

critical knowledge (e.g, market analysis) remains at the service provider, while non-critical knowledge (e.g., advice customization guidelines) can be sent to the service representative.

- The service client receives the service response from the service representative. The client does not have access to the knowledge transferred between the service provider and its representative. Consequently, encryption techniques can be used for data transmissions between a service provider and representative to improve enterprise privacy.

2. *Testing.* Task services use the service representative to execute a process on behalf of the service provider at the client side. Therefore, testing and error handling procedures are more challenging for service developers since they do not have direct access to the client's resources and execution platform. An effective evaluation technique requires test cases for different client platforms and contexts. Moreover, the interaction between the client application and the service representative needs to be evaluated using proper test cases. Finally, techniques have been proposed for evaluating Rich Internet Applications (e.g., Java Applets or Microsoft Silverlight) which can be useful in these cases (Raffelt et al. [2008]).

3. *Service Client Adaptation.* To call task services, a service client must have installed a generic service representative with the corresponding communication channel. After the generic service representative is installed, the service client can call different task services that employ service representatives to provide their final service responses as follows. The service client puts the local service parameters into the communication channel and calls a task service by sending its remote parameters. When the client receives the service response, it will forward the message to the service representative and wait until the service representative completes its job. The client can then read the communication channel to get the final service response.

3.6 Related Work

In this section, we briefly present some of the research literature related to task services. In distributed computing, code on demand (Brooks [2004]), also called code mobility, refers to any technology that sends executable software programs from a server computer to a client system upon request from the client. As we showed in the evaluation section, task services deliver performance that is compatible with code on demand technologies (*Silverlight* in our case). However, task services have advantages over code on demand approaches due to their composability and scalability. Finally, task messages are not executable, which improves client security.

Mobile agents (Braun and Rossak [2004]) can physically travel across a network and perform tasks on different nodes. Therefore, instead of customizing the local and generic service representative, a service provider can send a customized mobile agent to the client to perform client-side processing of its web services. There are several security and privacy issues to be considered in mobile agent computing (Dadhich et al. [2010]). Viruses and malicious attacks are other possible vulnerabilities of mobile agent systems. Mobile agent architectures also suffer from low efficiency as they need to send the entire computer program or process. Moreover, flexibility and interoperability concerns must be addressed in these approaches.

Each web service can be modeled by a business process (Lu and Sadiq [2007]) where a data service provides a server-side business process. By task services, we propose modeling the client-side processing of a service by a business process called a task. A task is defined by a service provider and then it is completed, instantiated, and executed by the service representative.

Software agent technology has been applied in modeling and executeing business processes to provide flexibility and expressiveness in business process automation. The idea of the agent-based business process management system (BPMS) is to split business processes into parts and assign the control over such parts to individual software agents (Miller et al. [2004]). The integration of agents and web services is proposed to model the business aspects of enterprise systems, where each role or major function of an enterprise system is considered as an agent (Xiang [2007] and Khalili et al. [2008]).

The distributed nature of business process executions has been addressed recently in several articles. For example, Li et al. [2010] propose a distributed agent-based framework for business process execution in SOA. The combination of task service and service representative provide a distributed framework to locally process client data. In this framework, the task components are provided from different sources such as: i) task model, provided by the service provider; ii) task knowledge, provided externally by the service provider (or knowledge services) or internally by the SR knowledge base; and iii) task data, provided by both the service provider and the service client.

3.7 Conclusion

This Chapter presented an extension to the SOA infrastructure model to support client-side task services in addition to the traditional server-side data services. In the proposed model, a generic and client-side service representative processes client data based on a task message received from the service provider. Moreover, the task message includes three components (task model, task knowledge, task data) which can be provided from different sources. These web services are complementary and have their own applications and features represented in Table 3.2.

The SOA model extension proposed in this thesis to support client-side web services has not been previously addressed in the current state of SOA. While the extended SOA proposed here adheres to SOA concepts and principles, a number of web service artifacts and requirements need to be extended and revised. In other words, to simulate employing a service agent at the client side, which is a common enterprise task, the following SOA components have been extended.

- *Generic Service Representative Plug-in.* A service client is required to install a plug-in to enable an external system/agent/code to process data and events at the client-side. For example, an Internet user installs Flash Player, Silverlight, and JVM plug-ins to have Flash, XAML, and Java applets running on his/her system. Similarly, to invoke task services, service clients require one-time installation of the generic service representative plug-in on their systems. Because this plug-in is generic and independent of any particular web services and client environments, it does not deviate from SOA principles (i.e., service loose-coupling).

- *Extended WSDL Document.* To invoke a real business service, a service requester knows whether the service is provided remotely or whether the service provider needs to send an agent to deliver the service locally. Moreover, the service requester inquires whether data and resources are to be processed locally by the agent and to be sent to the provider. Similarly, the WSDL definition of a web service in the proposed SOA model needs to be extended to include: i) the type of a web service (server-side data service or client-side task service); and ii) local and remote service parameters and responses.

- *Standard Knowledge Representation for the Generic Service Representative.* The proposed generic service representative requires task knowledge to perform a task. Knowledge statements are of different types: descriptive knowledge, procedural knowledge, and data mining models. The service representative has knowledge engines to execute (or learn from) knowledge statements. For example, in the case study presented, if-then-else statements and Java codes are used to represent descriptive knowledge and procedural knowledge. Accordingly, the SR business process engine has a business rule engine to execute business rules and a Java Virtual Machine to run Java codes.

  In order to have a generic service representative, there must be a standard representation for each type of knowledge. Several standards have been proposed for knowledge transfer and representation. This thesis focuses on the feasibility and the potential advantages of having client-side web services. Hence, developing a universal and standard knowledge representation for the service representative is outside the scope of this study. When there is a standard form of knowledge for the service representative, task knowledge will be independent of any programming language which adheres to SOA principles. For example,

Table 3.2. Comparison of different types of web services.

Type	Applications	Pros	Cons
Data Services	Server-side processing of non-sensitive client data	- Integrated service logic - Easy to assess service reliability - Maintains enterprise privacy	- Violates client privacy - Increases network traffic - Increases service response time
Task Services	Client-side processing of confidential, large volume, or real time client data	- Maintains client privacy - Reduces network traffic - Improves service response time	- Distributed service logic - Difficult to assess service reliability

Predictive Model Markup Language (PMML) represents data mining models and if-then-else statements efficiently. As a result, the generic service representative can support these two types of knowledge using a PMML engine. However, standardizing the procedural knowledge is more challenging.

Chapter 4

Data and Task Service Composition [1]

One of the strengths of web services is their capacity to be composed into higher-level composite services. Several approaches have been proposed to address service compositions. These approaches can be classified in different ways, including orchestration and choreography (Peltz [2003]). While orchestration requires a central process to coordinate sending and receiving messages among web services, collaborating web services communicate with each other directly in choreography approaches. In these cases, static composition takes place during design-time while dynamic composition selects collaborating services at run-time. The Business Process Execution Language (BPEL) (Juric [2006]) is an industry-wide standard that models a business process based on collaborating web services. The BPEL process is provided as a composite web service that can be called by a service client.

Traditionally, composition of web services is performed at the server side. This requires transferring client data among collaborating web services, which may cause data privacy violations, security breaches, or network traffic overloading. The task service and service representative proposed in this thesis allows a new concept called "client-side service composition", where collaborating web services employ the service representative to provide a composite task service at the client side. In this case, the client is not required to reveal its resources to service providers and hence its privacy and security are maintained. Moreover, large client data files are processed locally, resulting in less network traffic.

In this chapter, BPEL is adapted to invoke task services. Therefore, a BPEL process is used to compose both data and task services. Moreover, the service representative is enhanced to act as a service orchestrator to address the client-side composition of web services. Finally, the client-side and server-side service compositions are compared using a case study in the healthcare domain.

[1] This chapter is mostly based on the following paper: M. Najafi, K. Sartipi, and N.Archer, *Client-side Service Composition Using Generic Service Representative*, Conference IBM CASCON 2010, pages 238-252, Toronto, Canada.

The organization of this chapter is as follows. composite web services in the proposed extended SOA model are introduced in Section 1. Section 2 discusses the required service composition model. Client-side and server-side service compositions are discussed in Section 3. A case study of data and task service composition is presented in Section 4. Using this case study, we discuss the advantages of the client-side service composition over traditional server-side approaches in Section 5. Finally, conclusions are discussed in Section 6.

4.1 Composite Web Service

In the proposed extension to the SOA model, a composite web service is defined as a composition of data and task services over a designated flow structure that is performed by a service orchestrator component. The service orchestrator coordinates the execution of the web services involved (both data and task services) where these web services are not aware that they are taking part in a higher-level business process. Business Process Execution Language (BPEL) is an industry-wide standard that models a business process based on collaborating web services. The BPEL process is wrapped into a web service to represent a composite service that is called by service clients.

To invoke a composite service, the service client sends the required remote service parameters to the service orchestrator and sends the local service parameters to the service representative. The service orchestrator uses the remote service parameters to invoke partner services. The service orchestrator sends the response of each data service (remote service response) to the service client, while the response of each task service (task message) is sent to the service representative to generate the local service responses.

A composite web service (CWS) is modeled by a function that receives the required remote service parameters (RSP) from the service client and returns the resulting remote service responses (RSR) and task messages ($Task$) to the client application and the service representative, respectively. The following function represents the composite web service, where n and m represent the number of collaborating data and task services, respectively.

$$CWS : RSP \longrightarrow \prod_{i=1}^{n} RSR_i \times \prod_{j=1}^{m} Task_j$$

4.2 Service Composition Model

A service composition model specifies the exact order in which participating data and task services should be invoked, either sequentially or in parallel. Conditional behaviours can also be expressed. In this work, BPEL is used to model a composite service that allows complex

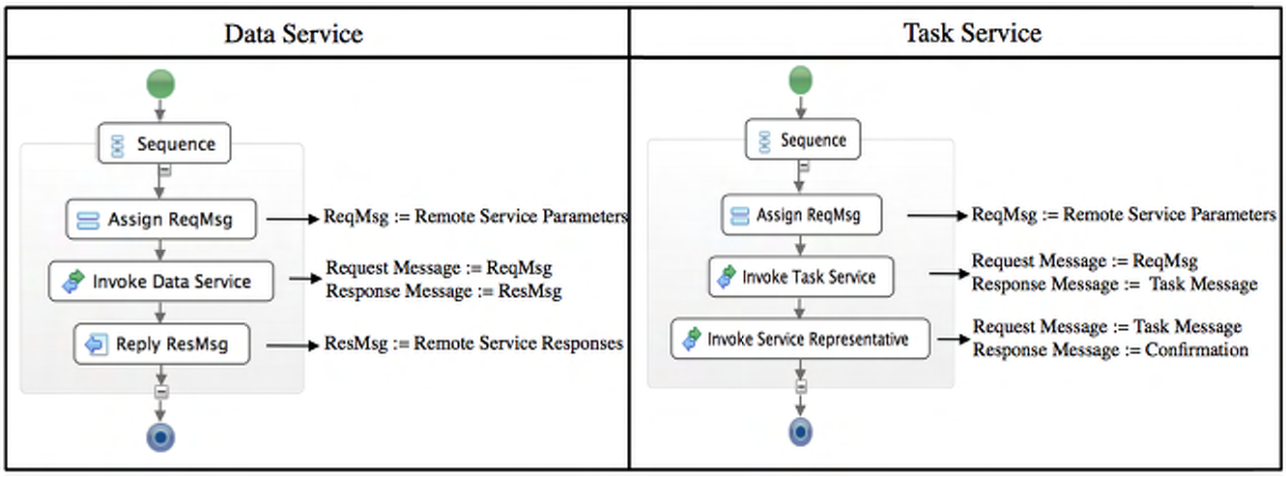


Figure 4.1. BPEL activities to call data and task services.

business processes to be defined in an algorithmic manner, including loops, variables, and fault handlers. The BPEL "invoke" activity invokes a web service by sending a service request message and receiving a service response message. The required steps to invoke data and task services are shown in Figure 4.1. To invoke a task service, the service representative is provided as a local service at the client side, which can be used by different task services to process the client data locally.

4.3 Server-side vs Client-side Service Composition

Traditionally, composition of web services is performed at the server side. This requires transferring client data among collaborating web services, which may cause data privacy violations, security breaches, or network traffic overloading. Figure 4.2 compares client-side and server-side service composition in terms of the internal (local) and external data transmissions. The client-side requires fewer external data transmissions, which improves the client's privacy as well as reducing network traffic. However, it could increase client-side complexity.

The service orchestrator has a business process engine to execute BPEL processes. Based on the proposed extended SOA model, the service representative has a business process engine to execute task processes. Therefore, the service representative can be extended to execute BPEL processes to address client-side service compositions. For this purpose, the service client calls a composite service, which returns a BPEL model to the service representative. Here, the service representative plays two roles: the service orchestrator to run the BPEL process, and the service representative for each of the collaborating task services.

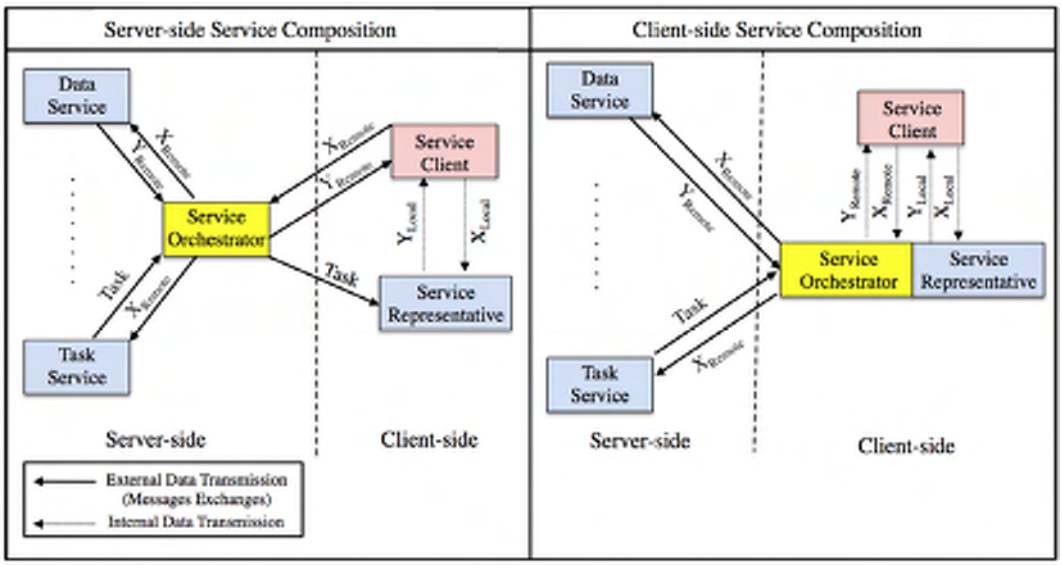


Figure 4.2. Comparison of server-side and client-side services composition in terms of internal and external data transmissions.

Finally, the client-side service composition approach is not meant to replace traditional server-side service composition. However, when collaborating web services require processing confidential, large, or dynamic client resources, composing them at the client side can improve SOA performance.

4.4 Case Study

There are two types of Clinical Decision Support System (CDSS): guideline-based and model-based (Greenes [2006]). A guideline-based approach takes patient information and matches it with the patterns obtained from medical experiments and observations. A model-based approach initially builds a decision model according to known data (training data) and then applies this model to unknown data (test data). Since each approach has both benefits and limitations, a combined approach could lead to correct or better actions for patient management decisions.

CDSSs cannot be developed efficiently and securely using traditional web services for the following reasons. Guideline-based approaches require transferring patient health information which is highly sensitive since disclosure of this information could identify patients. On the other hand, model-based approaches do not consider local data (patient health information) when building their decision models. Consequently, the model obtained may not be well-matched with client data. However, the local data files are often too large to be transferred efficiently to the server side for processing.

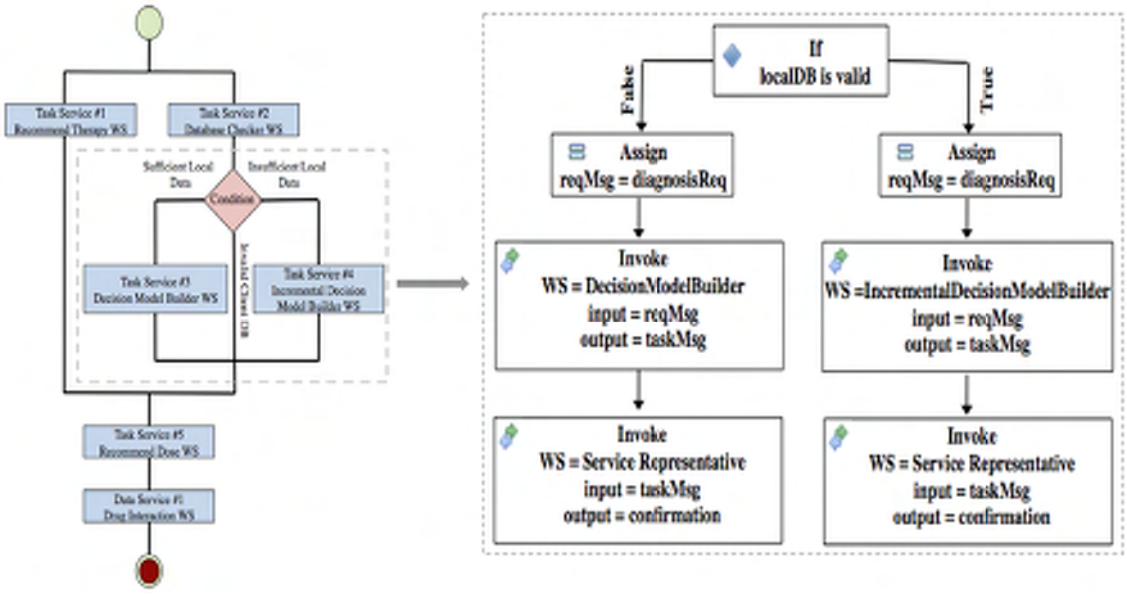


Figure 4.3. composite task model (left) and the corresponding BPEL model (right) of secure and context-aware CDSS.

Using the proposed methodology and implemented prototype (Appendix A), a secure and context-aware CDSS was modeled and developed through a composite task service. While the client passes patient health information (as client data) to the communication channel, the service representative generates both model-based and guideline-based recommendations without being affected by the issues mentioned. The composite task is shown in Figure 4.3 and discussed in the following subsections.

4.4.1 Service Client

The client puts its data and resources into the communication channel (Figure 4.4 (left)) with the following ports.

1. *Patient health record* (private, read only): contains the health information of the target patient.

2. *Visit information* (public, read only): contains the information that a physician gathers by visiting the target patient. The medication and dose fields are completed by the physician after consulting with the service representative.

3. *Patient database* (private, read only): contains the patient's health records within the health-care organization.

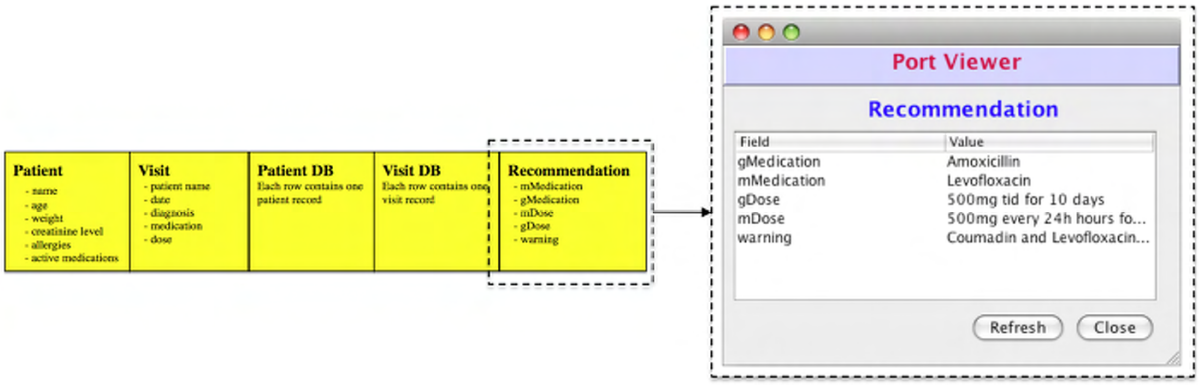


Figure 4.4. Communication channel schema for the secure and context-aware CDSS (left), and the Recommendation port after the service representative completes the composite task (right).

4. *Visit database* (private, read only): contains patient visit information within the healthcare organization.

5. *Recommendation* (private, read/write): receives i) recommended medication: guideline-based (gMedication) and model-based (mMedication), ii) proper dosage: guideline-based (gDose) and model-based (mDose), and iii) drug to drug interaction warnings from the service representative.

4.4.2 Collaborating Services

The composite task model includes five task services and one data service, as follows:

1. *Recommend Therapy* (task service): receives a diagnosis report and returns the corresponding medication guidelines.

2. *Database Checker* (task service): receives a database schema and returns a task to verify whether the client database matches with this schema. This task also reports some statistical information about the local database.

3. *Decision Model Builder* (task service): receives a diagnosis report and returns a task to build and apply a decision model based on the client-side patient and visit databases.

4. *Incremental Decision Model Builder* (task service): receives a diagnosis report and returns a task to rebuild, complete, and apply an incremental decision model based on the client-side patient and visit databases (incremental models can be adjusted and modified based on the new training data).

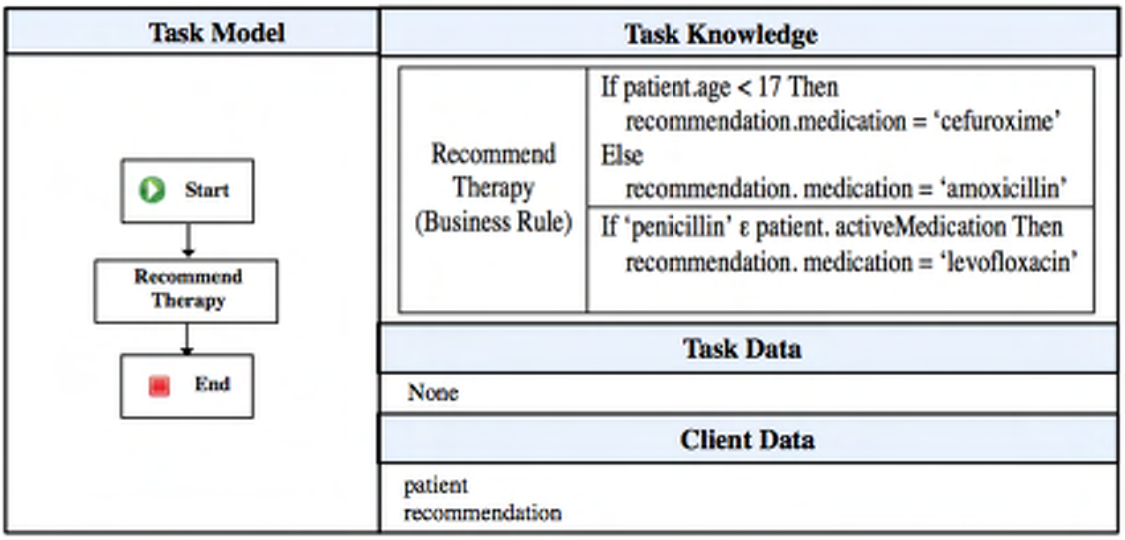


Figure 4.5. Task generated by Recommend Therapy web service where it receives *"acute sinusitis"* **as its parameter.**

5. *Recommend Dose* (task service): receives a recommended medication and returns the corresponding dose guidelines.

6. *Drug Interaction* (data service): receives a target medication and a list of active medications and returns warnings if there is one or more drug-to-drug interaction.

4.4.3 Service Representative

The service representative performs the composite task as follows. The service representative executes the task received from the *recommend therapy* service to calculate gMedication (Figure 4.5). Then, it verifies local databases using the *database checker* task service (Figure 4.6). If the patient database and the visit database have enough records, the service representative executes the task received from the *decision model builder* to build a decision model completely from local data (Figure 4.7). Otherwise, the service representative receives an incomplete model from the *incremental decision model builder* and completes it, based on local databases (Figure 4.8). Then the service representative applies the decision model to the patient's health record to obtain *mMedication*. The internal knowledge base of the service representative contains the required functions to build and apply decision trees (both incremental and non-incremental). Next, the service representative executes the task received from the *recommend dose* for *gMedication* and *mMedication* to calculate *gDose* and *mDose* (Figure 4.9). Finally, the *drug interaction* data service

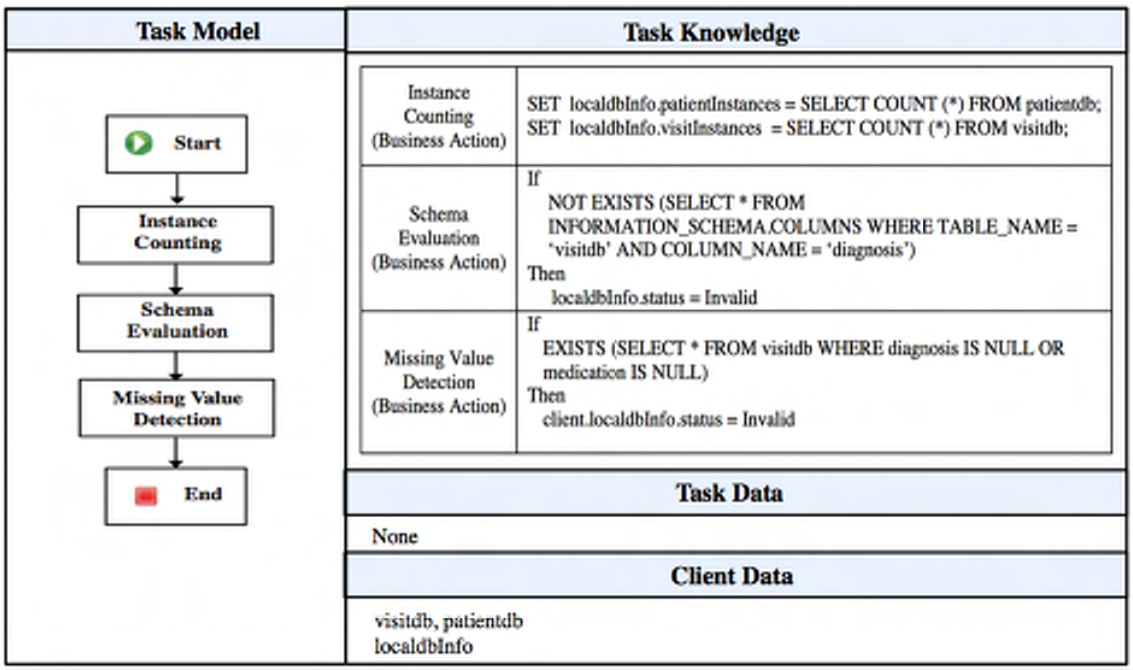


Figure 4.6. Task generated by Database Checker web service where it receives $visitdb\ schema$ **as its parameter.**

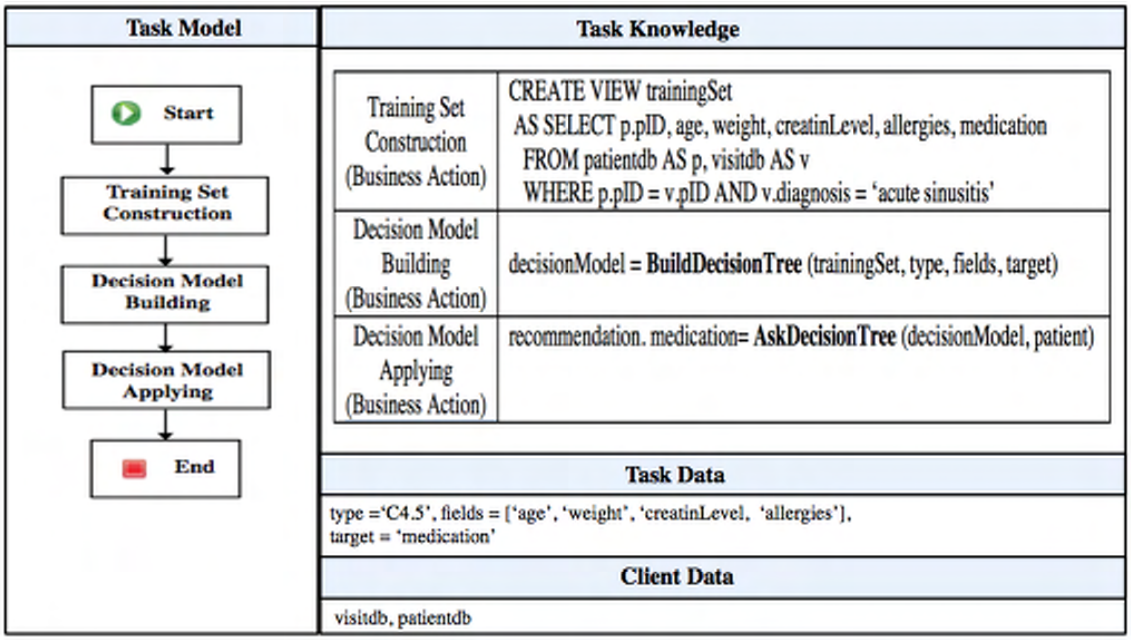


Figure 4.7. Task generated by Decision Model Builder web service where it receives $"acute\ sinusitis"$ **as its parameter. The internal task knowledge are highlighted and** $C4.5$ **is the type of decision tree.**

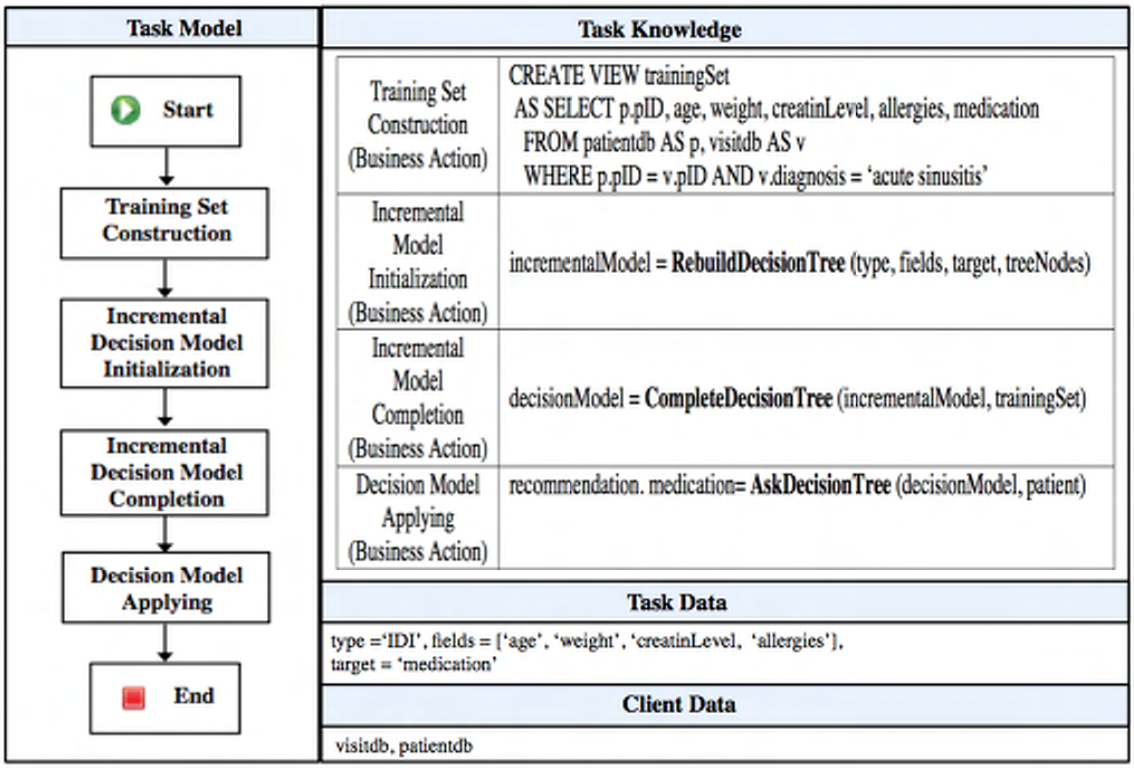


Figure 4.8. Task generated by Incremental Decision Model Builder web service where it receives *"acute sinusitis"* **as its parameter. The internal task knowledge are highlighted and** IDI **is the type of incremental decision tree.**

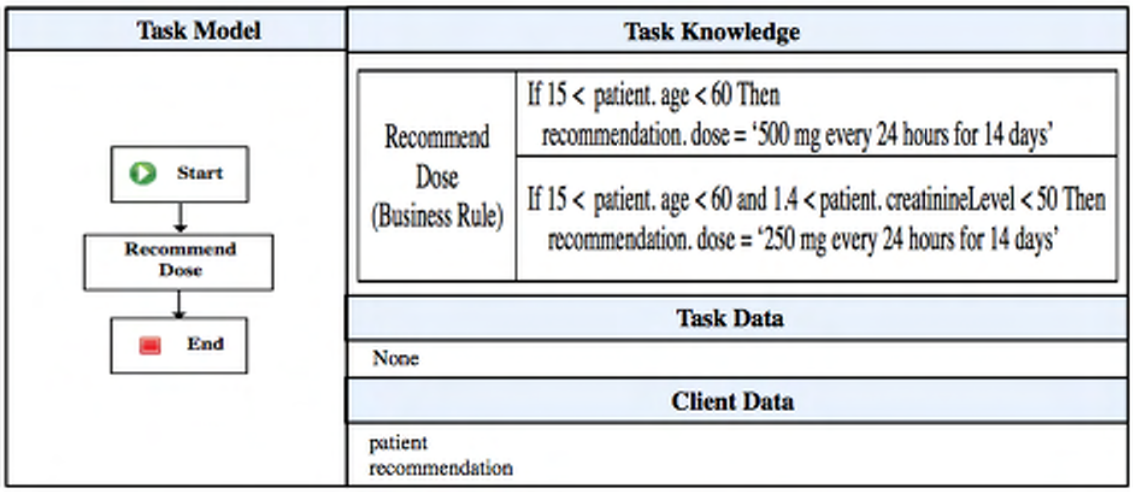


Figure 4.9. Task generated by Recommend Dose web service where it receives *"amoxicillin"* **as its parameter.**

is called for (*gMedicine, activeMedication*) as well as (*mMedicine, activeMedication*) to return the necessary drug interaction warnings. Tasks results are stored in the recommendation port of the communication channel that is shown in Figure 4.4(right).

4.5 Evaluation

In this section, the case study discussed in Section 4 is used to evaluate the performance of the model. For this purpose, three alternative composite services were developed, as follows.

1. *Data Services Composition*: This represents a typical composite service where it takes complete client data (i.e., patient and visit databases and records); processes it completely at the server side; and returns the final recommendations to the client (Figure 4.10).

2. *Data and Task Services Composition (Client-side)*: This represents the composite web service presented in Figure 4.3, where the collaborating services are composed at the client side by the service representative.

3. *Data and Task Services Composition (Server-side)*: This represents the composite web service presented in Figure 4.3, where the collaborating services are composed at the server side by the service orchestrator.

composite services which use task services (options 2 and 3) perform better compared to the traditional approach (option 1) according to the following metrics.

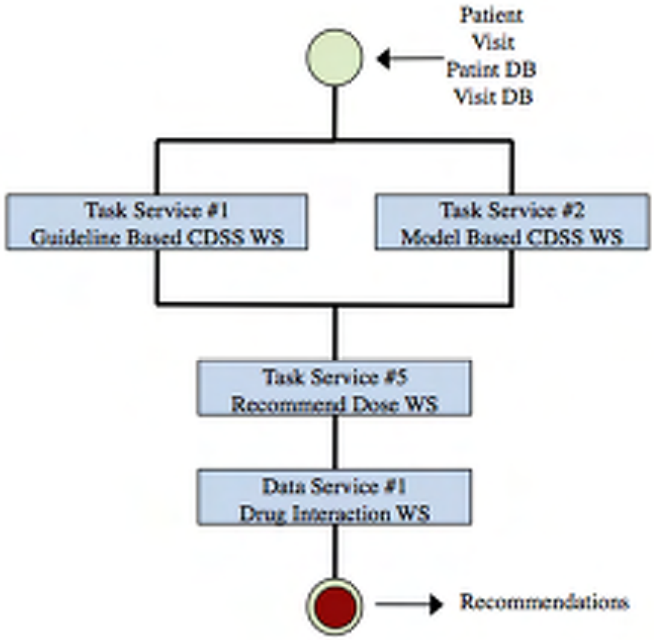


Figure 4.10. Equivalent composite data service for the case study presented in Section 4.

1. **Client Privacy**. Using task services, the patient and visit information are kept local, and therefore the client's privacy is maintained. However, the client has to reveal its information to the composite data service (option 1) and hence client privacy is violated.

2. **Service Message Size (SMS)** is the total size of service request and response messages. The composite data service approach requires transferring complete client data from client to the server. Although it returns small response messages containing final decisions, the request message is almost the same length as the client data. On the other hand, the composite services using task services process client data locally, implying that SMS is independent of the length of the client data. Therefore, the request message is short while the response messages containing the task definitions are large.

   The client-side composition of data and task services outperforms the server-side approach since the latter requires additionally sending the remote service parameters from the service client to the service orchestrator (Client–> Orchestrator –> Services). Figure 4.11 illustrates a comparison of these approaches in terms of SMS where they model the case study presented above. A logarithmic scale is used to show the lower values more clearly.

3. **Service Response Time (SRT)** is divided into two factors: Network time (N) and Processing time (P) defined as follows: $SRT(s) = N(s) + P(s)$. For this case study, we obtained the processing time $P(s)$ for different sizes of client data files using a 2.4 GHZ Intel dual-core

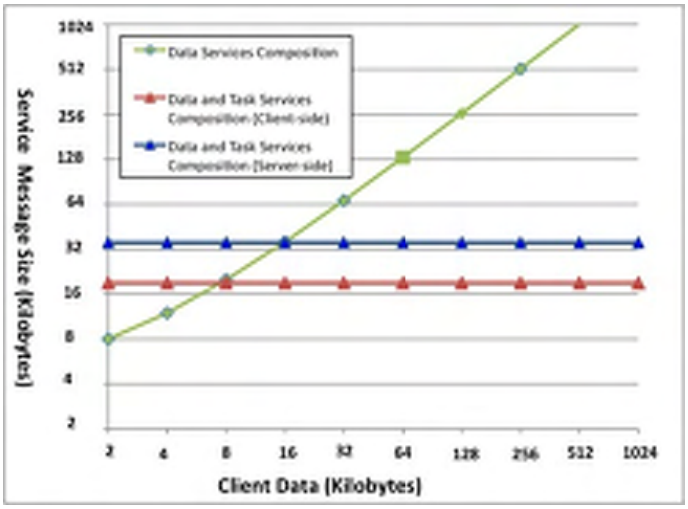


Figure 4.11. Service Message Size (SMS) comparison.

CPU. We also assumed $P_{Serverside}(s) = \frac{1}{2}P_{Clientside}(s)$ and that a high-speed bandwidth (1 Mbyte/Sec) connects the client to the servers. Figure 4.12 shows the result of a comparison between three alternative service composition approaches in terms of SRT. This result confirms that the proposed approach overcomes the traditional approaches when the client data file is large.

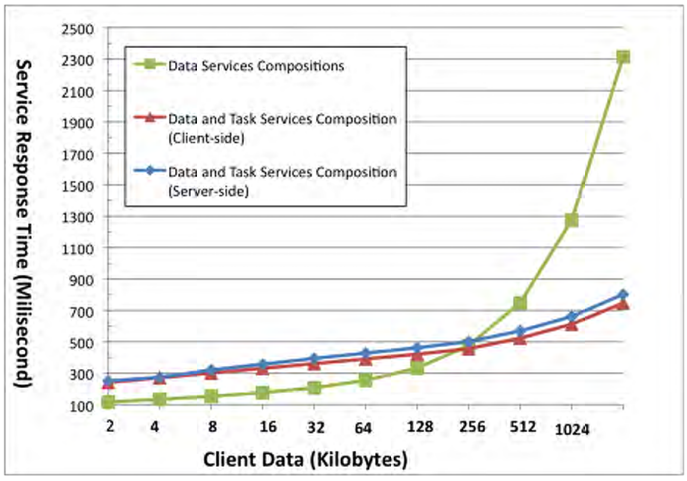


Figure 4.12. Service Response Time comparison.

4. **Average Response Time (ART)** extends the SRT metric when multi clients invoke a service simultaneously. A comparison of these approaches in terms of ART is shown in Figure 4.13. Using task services, service clients have their own service representative, which can process client data in parallel, and therefore the ART is improved significantly.

Figure 4.13. Average Response Time comparison.

Finally, based on these results, the server-side and client-side composition of data and task services show comparable performances for this case study. The reason is that the service request messages are relatively small as they only include the diagnosis report and local database schema. However, in other cases these two approaches may perform differently, especially when the client has to send more client data to be processed by the collaborating task services.

4.6 Conclusions

In service-oriented architecture (SOA), the business functionality of an enterprise is modeled by (web) services. To achieve more sophisticated functionality, web services should be reusable and composable. The proposed task services are composable with data services by BPEL models. This requires enabling the generic service representative as a local service to be invoked by the BPEL process. Finally, while all traditional approaches compose web services at the server side, the service representative can be extended to act as a service orchestrator at the client side in order to introduce the client-side composition of services. This improves security, privacy, and efficiency features in certain applications.

Chapter 5

Case Study: Virtual Remote Nursing System [1]

New demands for better health services has resulted in enhancements to electronic health (eHealth) systems concepts such as patient-centric, ease of use, smarter interactions, and appropriateness of decisions. While eHealth services aim to provide continuous medical and health services for consumers and healthcare professionals, limited access to experts such as nursing professionals is still a challenge. This chapter proposes an application of service representatives and task services in the healthcare domain. A new framework, namely Virtual Remote Nursing (VRN), provides a virtual nurse agent that can be installed on the client's personal computer or smart phone to help control the client's health condition continuously. In this approach, medical practitioners can assign different tasks to a virtual nurse using a generic task definition mechanism, where a task is defined as a combination of medical workflow, operational guidelines, and associated data. The VRN acts as a personalized and full-time nurse for its client patient by performing practitioner tasks, based on the client's health information. Such patient information can be obtained from a personal health record system such as *Microsoft HealthVault*. Finally, a sample scenario demonstrates how diabetic patients might take advantage of this system.

5.1 Introduction

Healthcare is a major factor in governmental expenditures in most developed countries[2], and healthcare systems are consequently under continuing pressure to lower costs and improve out-

[1]This chapter is mostly based on the following paper: M. Najafi, S.Aghtar, K. Sartipi, and N.Archer, *Virtual Remote Nursing System*, IEEE Consumer Communications and Networking Conference (CCNC), pages 13-18, Las Vegas, Nevada, USA, 2011.

[2]See http://www.creditloan.com/blog/2010/03/01/healthcare-costs-around-the-world/

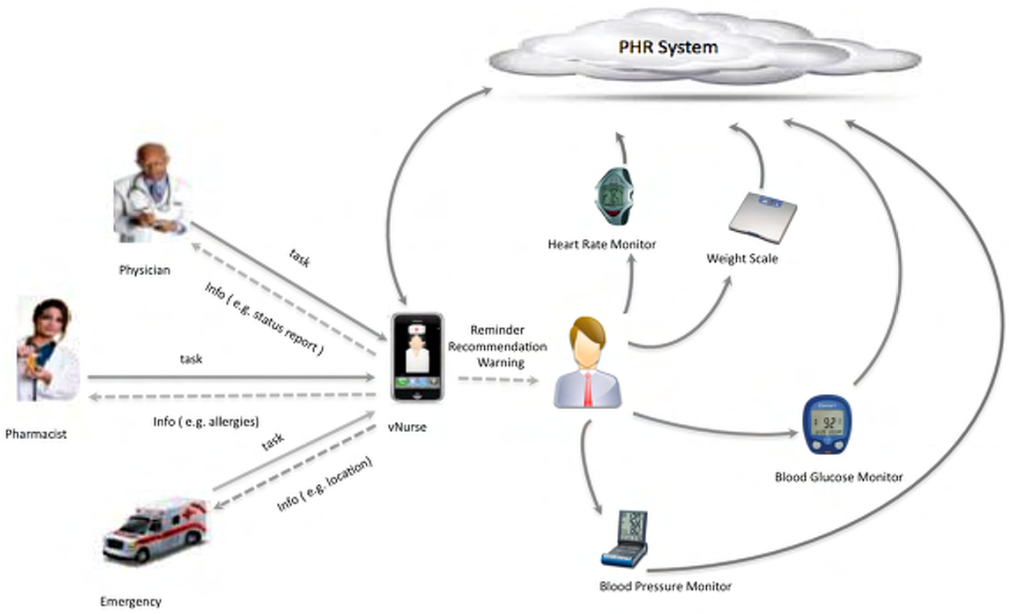


Figure 5.1. A high level view of the Virtual Remote Nurse.

comes. eHealth technologies have been developed to address healthcare problems through Electronic Medical Record (EMR) systems, Personal Health Record (PHR) systems, Clinical Decision Support Systems (CDSS), and telemedicine applications.

The Smart Internet, also known as the user-centric Internet, encourages individuals to use PHR systems to track their health situations. PHR systems such as *Microsoft HealthVault* are becoming popular, with an increasing number of people in the United States interested in having PHR accounts[3].

Similarly, monitoring patients via mobile health solutions is becoming more widely used (Yonglin et al. [2010]). Current medical devices that are integrated with PHR systems can enable updates on the patient's health status continuously and automatically. Therefore, a continuing stream of medical information that must be interpreted by healthcare professionals is unavoidable. The current shortage of healthcare workers, such as physicians and nurses, in most countries could limit the potential use of PHR applications (Kuehn [2007] and Naicker et al. [2009]).

A software agent is a piece of software that acts on behalf of an agency to serve a user. Software agents have been applied in different domains, to model roles and practices such as financial advisers in banking, fraud detection in insurance, and trip planners in entertainment. However, agent computing has not yet been widely adopted in the healthcare domain. Generally, software agents tend to operate autonomously, while in the healthcare domain decision making is a critical task that must be performed under health professional control.

[3]http://www.chcf.org/publications/2010/04/consumers-and-health-information-technology-a-national-survey

This thesis proposes an approach which simulates the functionality of nurses remotely, in the form of a Virtual Remote Nursing (VRN) system. In this approach a generic software agent operating as a virtual Nurse (vNurse), is installed on the client's personal computer or smart phone. A medical practitioner who is responsible for the client's health can define different tasks for this generic agent. In other words, the medical practitioner tells the virtual nurse *what to do* and *how to do it* for each assigned task, and the virtual nurse will then perform the task at the appropriate time. Consequently, the medical practitioner is the decision maker and the virtual nurse is the agent that performs the task requested by the medical practitioner, based on having full-time access to the client's up-to-date health information that is stored in the PHR system. A physician can ask the vNurse to report on the patient's health status, or to give appropriate warnings, recommendations, and reminders to the patient in defined situations. Figure 5.1 represents a high-level view of this system.

The VRN could reduce healthcare costs by lowering the number of patient homecare nursing visits and avoiding unnecessary hospitalization. As a result, patient wait time for medical services should decrease. At the same time, the virtual nurse provides information about the patient's health condition promptly and continuously to medical practitioners in a manner that could reduce response times. Moreover, since medical practitioners would have access to the patient's health information through the virtual nurse, patient conditions would be diagnosed faster and their treatment could start sooner. The prototype virtual nurse system is currently operating with short messages which are suitable for smart phone devices. This allows patients to have continuous access to their nurses and medical practitioners if they install the virtual nurse on their smart phones.

The remainder of this chapter is organized as follows. Section 2 describes the virtual remote nursing system. Section 3 discusses a sample scenario using virtual remote nursing in diabetes management. Section 4 references Appendix B where several potential tasks for hypertension management are described. Section 5 elaborates on related work, and finally, Section 6 concludes the chapter.

5.2 Virtual Remote Nursing

The proposed Virtual Remote Nursing (VRN) system introduces the virtual Nurse (vNurse) as a generic software agent installed on the client's machine, which would be either a mobile device or a personal computer. A medical practitioner uses a healthcare provider system to define tasks for vNurse based on a generic task definition approach. Each task can be considered as a function that is applied on the client's PHR, depending on the context, to provide medical information for the practitioner or client. vNurse has access to the client's PHR information (through the client's

PHR system) as well as the client's context, such as current location and time (through local applications), and performs assigned tasks relevant to the context. In this section, we propose a generic task definition approach, as well as the required architecture for the VRN system.

5.2.1 Generic Task Definition

Each medical guideline represents a pattern in the healthcare domain that corresponds to a decision or an action. An actual nurse is asked to look for these patterns and perform the corresponding actions. This means that every task assigned to the nurse can be modeled by a sequence of guideline steps, called a medical workflow. We define a task for vNurse with the following components.

$$Task =< Schedule, Model, Knowledge, Data >$$

- *Task Schedule* determines the execution time for the task.
- *Task Model* specifies different steps of a task through a medical workflow.
- *Task Knowledge* provides the corresponding medical guidelines for each step of the medical workflow.
- *Task Data* is the server-side data that is consumed by the medical guidelines provided during task execution. The client-side data is provided by different sources such as the PHR system, and certain local applications.

As a simple example, a physician assigns a task to vNurse to report the general health status of a patient every week (i.e., task schedule). To perform this task, the patient's vNurse needs to follow a medical workflow consisting of two steps: first, check the patient's blood pressure and second, check the patient's body temperature (i.e., task model). Each step corresponds to a set of medical guidelines that define the patient's status for each range of blood pressure or body temperature, which can be low, normal, or high (i.e., task knowledge). Finally, the information obtained should be reported with a specific medical format (i.e., task data).

5.2.2 Architecture

To allow a medical practitioner to define a task to be performed by the vNurse, an architecture is provided in Figure 5.2 which consists of the following components.

Personal Health Record System (PHR). This system (e.g., *Microsoft HealthVault* or *Google Health*) monitors and collects current health information from the patient. A growing number of health devices such as heartbeat rate monitors, glucometers, and blood pressure monitors can

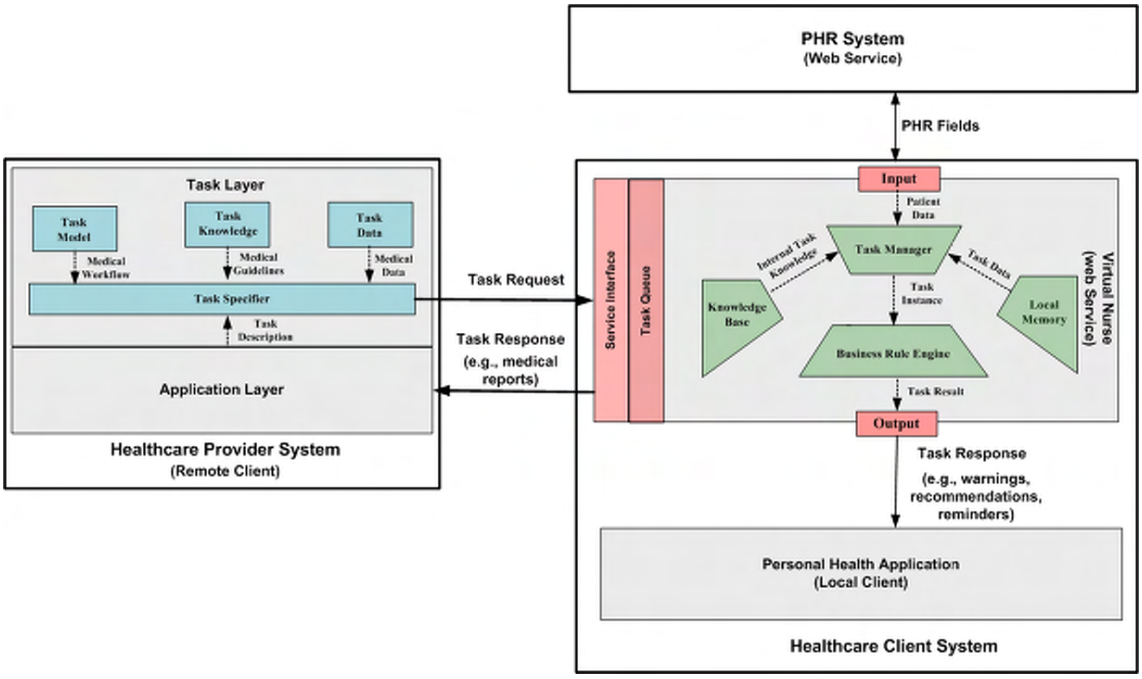


Figure 5.2. Proposed architecture. Each medical practitioner uses the healthcare provider system to define a task request message to be executed by the virtual nurse (vNurse).

upload their collected data directly into a PHR system. The vNurse uses PHR Application Programming Interfaces (APIs) to obtain its client health information.

Healthcare Provider System. The application layer represents a typical Electronic Medical Record (EMR) system that enables a medical practitioner to record and access patient medical information and profiles. Based on this information, the practitioner can describe a task for the virtual nurse that controls the patient's status during treatments. The task description is sent to the task layer, where the task specifier retrieves the required task components from the corresponding modules to formally define the described task. Moreover, the application layer can receive and display task results from vNurses, such as patient health reports.

Personal Health Application. This component represents a typical personal health application that is enhanced to receive task results generated by vNurse. Task results are given to this application in the form of messages, including medical recommendations, warnings, reminders, etc. This application can simply display the messages, or use other Human Computer Interaction (HCI) approaches to inform the client.

Virtual Nurse. The proposed virtual nurse is modeled as a software agent, wrapped into a web service to be called by different healthcare provider systems. Thanks to mobile devices and their

application platforms, even smart phones can become servers that provide web services. vNurse has a service interface that takes a task request and sends the task response to the healthcare provider (remote client) or healthcare client (local client). In order to apply medical guidelines, vNurse considers each medical guideline as a business rule, including condition and action parts. Similarly, vNurse considers medical workflows as business process models and task data as business objects. The agent components are discussed below.

- *Input* receives client health information from the PHR system.
- *Output* sends task results (e.g., recommendations, warnings, reminders, etc) to the personal health application to be delivered to the client.
- *Task Queue* stores tasks that are waiting to be selected for execution. Initially, tasks are stored in the task queue based on First-In First-Out (FIFO) strategy and their specific priority.
- *Local Memory* receives and stores task data which are invoked by the business rule engine during task execution.
- *Knowledge Base* stores received task knowledge to relieve the service caller resending them each time.
- *Business Rule Engine* executes a task instance, by applying business rules and performing business actions on the business objects.
- *Task Manager* controls the entire life cycle of a task instance, divided into three phases as follows.

    1. *Selection phase*: selects a task for execution from the task queue based on its order and schedule.
    2. *Instantiation phase*: i) creates an abstract business process based on the received task components; and ii) instantiates a task instance from the abstract business process using relevant PHR fields and internal functions stored in the knowledge base.
    3. *Execution phase*: passes the task instance to the business rule engine to be executed. The task response is sent to the client (via output) or healthcare providers (via interface).

5.3 Case Study: Diabetes Management

This section demonstrates applications of the Virtual Remote Nursing system through a case study that describes the involvement of three healthcare providers in the use of *VRN v1.0* prototype in assigning their tasks to the vNurse.

5.3.1 Scenario

Sherry is an elderly woman with type 2 diabetes. She has a vNurse installed on her smart phone. She also uses a blood glucose sensor that collects her blood sugar level at regular intervals and sends the data to her account in a PHR system. During her last visit the physician prescribes Metformin and Glimepiride and also advises her to go on a diet. The physician wants to ensure that Sherry stays on her diet until her next regular monthly appointment. Meanwhile, the physician is interested in knowing how well the prescribed medications work during this period. However, if Sherry's blood sugar level stays high even for a week, she will need to visit her doctor sooner than her scheduled appointment. As with other diabetic patients taking medications, Sherry is at the risk of hypoglycemia (i.e., low blood sugar level) that could result in losing consciousness. Consequently, the doctor wants to make sure she receives emergency medical services at the earliest time, in the case of extreme hypoglycemia.

Sherry visits a pharmacy to obtain her prescribed medications. The pharmacist wants to make sure Sherry does not forget the right dosage, time, and frequency of her medication, as prescribed by her doctor. Moreover, since the prescribed medications could cause side effects, he wants to have Sherry's doctor informed if the medications cause any complications. Finally, in case of an emergency situation, the emergency center should have access to the latest information about Sherry's health condition.

5.3.2 Virtual Nurse Assigned Tasks

Sherry's physician, pharmacist, and the emergency center define the following tasks for the Sherry's vNurse to manage their concerns about her.

Task One. The physician assigns a task to Sherry's vNurse to control her blood sugar level. This task is scheduled to be executed every time Sherry's blood sugar level is updated in the PHR system. This task, as shown in Figure 5.3, defines the following responsibilities for vNurse.

- In the case of hypoglycemia, vNurse contacts the emergency center.
- Every two days, vNurse checks the average of her blood sugar level. If it is high, Sherry will receive a warning to watch her diet more closely.

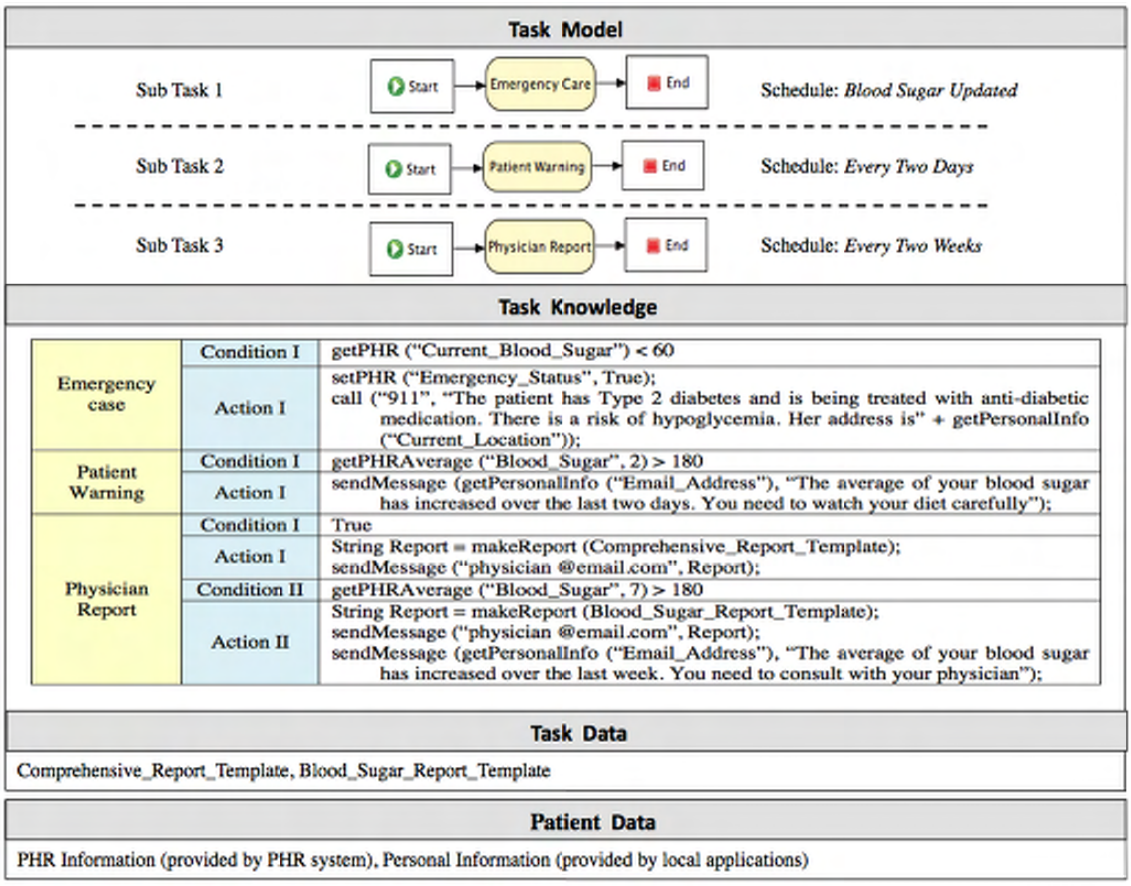


Figure 5.3. Task One. Sherry's physician assigns this task to vNurse to control her blood sugar level continuously.

- Every week, vNurse checks the average of her blood sugar level. If it is high, vNurse sends a report about Sherry's blood sugar status over the last week to the physician. Moreover, Sherry will be advised to make an appointment to visit her physician.
- Every two weeks, vNurse sends a comprehensive report about Sherry's health condition to her physician.

Task Two. The pharmacist assigns a task to Sherry's vNurse to handle medication-related issues. This task, as shown in Figure 5.4, is scheduled to be executed two times a day, as follows.

- At medication times, vNurse reminds Sherry to take her medications.
- Through interaction with the patient, the vNurse informs Sherry's physician about any side-effects caused by taking medications.

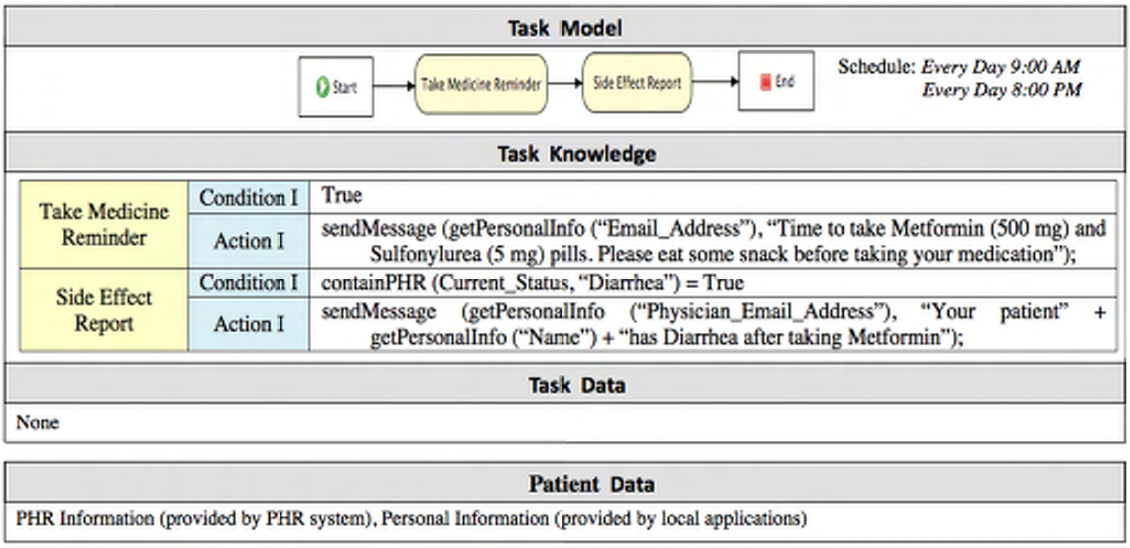


Figure 5.4. Task Two. Sherrys pharmacist assigns this task to vNurse to remind Sherry to take her medications, and to report any side effects to her physician.

Task Three. In case of an emergency, Sherry's vNurse contacts the emergency center. At this time, the emergency center assigns the following task, as shown in Figure 5.5, with a high-priority (i.e., immediate execution) to the vNurse.

- If the emergency situation is confirmed, the vNurse calls Sherry's emergency contact to let her/him know about Sherry's condition.
- vNurse sends the required information about Sherry to the emergency center in order to provide better and faster medical services for Sherry when she arrives.
- If Sherry's health condition changes, her vNurse informs the emergency center immediately.

In order to perform these tasks, the virtual nurse must call some functions that are stored in the internal knowledge base of vNurse. These are presented in Figure 5.6.

5.4 Case Study: Hypertension Management

Appendix B describes several tasks that could be defined for the virtual nurse to support hypertension management.

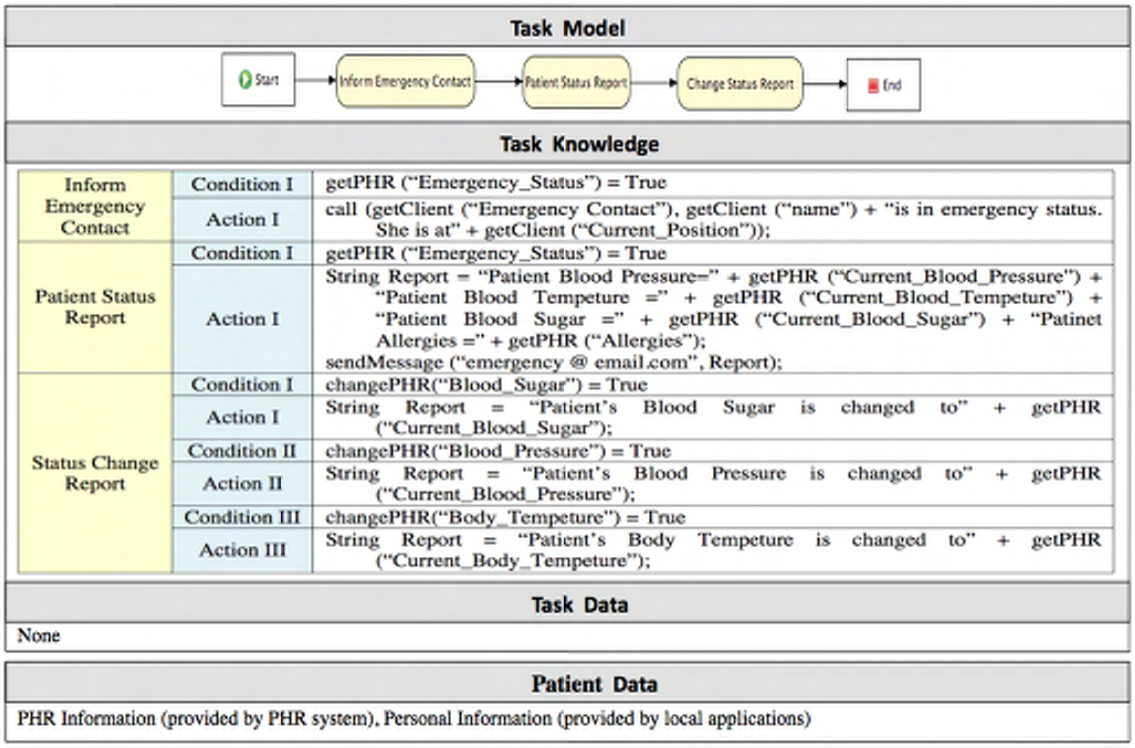


Figure 5.5. Task Three. Emergency center assigns this task to vNurse to receive latest information about Sherry's health condition in the case of hypoglycemia.

5.5 Related Work

The user-centric Internet introduces Medicine 2.0 (Eysenbach [2008]) as medical services that are based on client health information maintained by on-line PHR systems. Moreover, Web 2.0 technologies, the semantic web, and virtual reality tools facilitate social networking and collaboration between patients and physicians. Also, the integration of Medicine 2.0 with health monitoring systems (Korhonen [2003]), telemedicine (Wootton et al. [2006]), and sensor networks (Akyildiz et al. [2002]) provide more beneficial services. For example, an architecture for monitoring body motion and heart activities through sensor networks is proposed by Otto et al. [2006] where the monitoring data can be sent to the patient's PHR system. Similarly, Garsden et al. [2004] introduce a health monitoring system including intelligent reporting and alerts. In this chapter, a healthcare monitoring system was proposed that is controllable by medical practitioners.

A number of IT researchers are pursuing the simulation of certain health practitioner activities (Lenz and Reichert [2007]). For instance, Clinical Decision Support Systems (CDSS) (Greenes [2006]) and Electronic Medical Record Systems (EMRs) (Agrawal et al. [2007]) have been introduced. A CDSS could assist physicians in better diagnosis of diseases like heart failure (Leslie et al.

Method Summary	
String	getPHR (String *field*) Returns the value of the given PHR *field*.
double	getAveragePHR (String *field*, int *days*) Calculates the average value of the given PHR *field* during the given *days*.
void	setPHR (String *field*, String *value*) Sets the value of the given PHR *field* to the given *value*.
boolean	updatedPHR (String *field, int seconds*) Returns true if the value of the given *field* is changed during the given *seconds*.
String	getPersonalInfo (String *field*) Returns the personal information associated with this *field*.
void	call (String *phoneNumber*, String *message*) Makes a call to the given *phone number* and tells the given *message*. It uses a text2Speach function to convert the message into a voice message.
void	sendMessage (String *emailAddress*, String *message*) Emails the given *message* to the given *email address*.
String	makeReport (Report *reportTemplate*) Makes a report based on the given *template* which specifies required PHR fields and a report structure.

Figure 5.6. Virtual Nurse APIs stored in the internal knowledge base. The left column represents the method output and the right column represents the method name and method input parameters.

[2006]) or for administration tasks like elective surgery scheduling (Everett [2002]). An EMR could be used as an administrator for creating and maintaining patient data electronically. Although nursing plays an important role in healthcare domain, its roles have not been modeled efficiently. Only a few approaches have been proposed to model nursing activities: Goorman and Berg [2000] used patient record modifications; and Kumar et al. [2004] proposed a concept-oriented approach. The virtual remote nursing approach proposed in this chapter uses a task-oriented approach to provide an efficient model for nursing.

In the proposed architecture, a nurse is modeled as a generic software agent. Software agents have been integrated into the healthcare domain by Moreno and Nealon [2004] and Haigh et al. [2002]. Traditional approaches can model a nurse as a mobile agent that can physically travel across a network and perform tasks on different nodes. This method causes several security and privacy vulnerabilities, especially in healthcare. This unsafe mechanism has been resolved in VRN by sending task and knowledge to vNurse, a resident generic agent. Mobile agent architectures based on classic methods (e.g. Mole Baumann et al. [2009]) also suffer from low efficiency, as they need to transmit the entire computer program and process to the local computer. In the proposed

architecture, this problem is solved by sending short task messages. Finally, the proposed architecture extends the work by Kazemzadeh and Sartipi [2005]. This research introduces a framework for data and knowledge interoperability where it enables knowledge (medical guidelines) to be transferred in association with data (patient EMR or PHR records).

5.6 Conclusions

The healthcare domain may well continue to embrace the smart Internet and mobile technology in the near future. The smart Internet can assist individuals in accessing data and functionalities in PHR systems; mobile technology provides inexpensive devices that can efficiently collect medical or contextual information from their clients. The proposed virtual nurse approach takes advantage of these technologies to enable healthcare professionals, as the main decision makers, to control their patients' treatments and conditions. In this approach, a healthcare professional can define a specific task for a patient's virtual nurse using medical workflows, guidelines, and data. The virtual nurse then performs the specified tasks as needed and returns the task results to the healthcare professional or the patient.

The VRN system described in this chapter introduces a simple virtual nurse where the nurse agent performs assigned tasks by itself. An interactive VRN system can also be developed where an assigned task is accomplished through collaboration between the patient and the virtual nurse. Finally, a healthcare provider would also be able to assign certain simple decision making tasks to the virtual nurse.

Chapter 6

Web Service Competition: A New Approach to Service Selection [1]

As the number of web services that offer similar functionality increases, more sophisticated techniques for service discovery and selection will be needed. Traditional approaches compare web services, based on their descriptions as published in service registries. These descriptions include Quality of Service (QoS) parameters and price/performance ratios, as well as adaptability. This information is generated by the service developer and may not be fully trusted by the client. Moreover, alternative services perform differently in different client contexts that cannot necessarily be described accurately by service descriptors. In this chapter, a novel service selection approach is proposed that compares alternative services based on their performance in a specific client context. For this purpose, the SOA infrastructure model was extended through a component named *the competition desk*, which holds a competition among alternative services available to the service client. As a result, clients can choose the service that works best for their needs. Finally, the advantages of the proposed approach will be discussed, using two case studies.

6.1 Introduction

Emerging technologies such as cloud computing (Reese [2009]) propose applications as services which can then be distributed across the network and reused in other applications. Consequently, the number of web services that offer similar functionality continually increases, and discovering relevant services (service discovery) and choosing the best one to meet the client's needs (service selection) is becoming more challenging. At the same time, the Smart Internet (Ng et al. [2010])

[1] This chapter is mostly based on the following paper: M. Najafi, K. Sartipi, and N.Archer, *Web Service Competition: A New Approach to Service Selection*, International Journal of Web Intelligence and Agent Systems (WIAS), 2011 (Submitted).

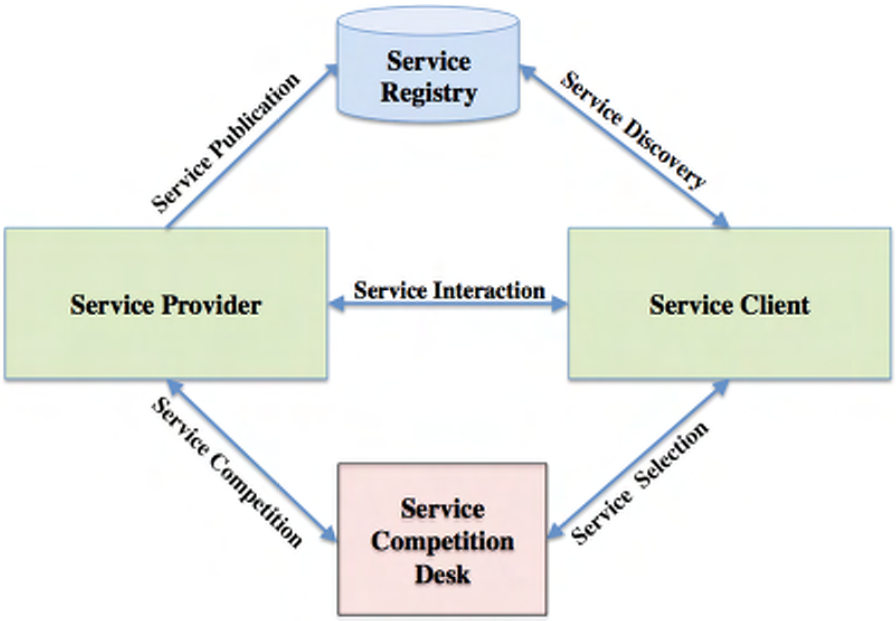


Figure 6.1. Proposed extended SOA infrastructure model.

has been introduced recently as one of the candidates for the next Internet generation, as the Internet moves towards a more user-centric network. The Smart Internet will allow clients to select services that work best, based on client contexts and needs. To further these goals, the underlying conceptual model and infrastructure of SOA must be extended and modified to meet the new requirements.

Based on the traditional SOA model, service providers publish descriptions of their services in a service registry that is used by service clients to discover and select services. This model suffers from the following limitations.

- Service descriptions that are provided by service providers may not be trustable or accurate enough.
- Service descriptions are usually expressed globally while service features such as performance and accuracy are different for different clients, depending on their needs and contexts.
- Less well-known services are not given an opportunity to show their features.
- Service features vary with different measures and are obtained under different situations. Therefore they cannot be simply and fairly compared, based only on their descriptions.

Inspired by the business domain, where in many cases service providers compete to win clients, this thesis proposes a service selection approach based on service competition (*WS-Competition*).

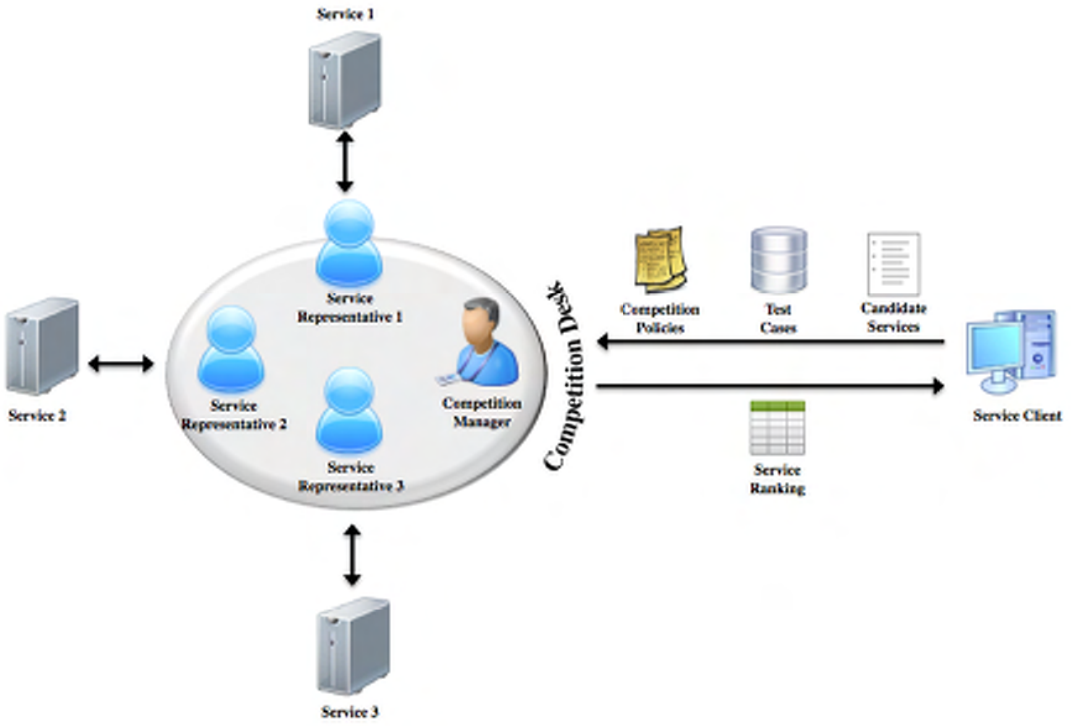


Figure 6.2. High level view of the proposed web service competition.

While traditional passive service selection relies on service description comparisons, the proposed active service competition compares services based on their performance for a specific client application.

This proposal extends the traditional SOA infrastructure model by adding a new component that is called the *Competition Desk* (Figure 6.1). This enables a service client to submit a list of candidate services (competitors) as well as a set of test cases and competition policies. The competition desk asks each of the candidate services to employ a representative and then it holds a competition among the service representatives and returns the competition results to the client (Figure 6.2). As a result, the client can choose the service that works best for its needs. The competition desk supports both data and task services. While a data service processes the client's data at the server side, a generic and client-side software agent (the service representative), is employed by a task service to process client data at the client side and deliver the service response locally to the client. The proposed performance-based approach increases the required time for service selection, compared to existing description-based approaches. However, in most cases, there is a long term relationship between service clients and service providers. As a result, a service client would prefer to choose a better service even if the selection phase requires more effort.

The organization of this chapter is as follows. Service selection based on the proposed competition model is described in Section 2. Two case studies based on a prototype system are presented in Section 3. The discussion (Section 4) explores the applications and challenges posed by the web service competition approach. Section 5 discusses work related to this approach. Finally, conclusions are provided in Section 6.

6.2 Web Service Competition

In order to perform service selection based on the proposed service competition model, an architecture is required. The proposed architecture (Figure 6.3) includes three main components as follows.

6.2.1 Service Client

A service client consists of a traditional client application that sends a data or task service request to a service provider. In order to receive task services, a service client is equipped with a generic service representative. Consequently the client application receives the service response directly from the service provider (data services) or indirectly from the service representative (task services).

To select a service that best matches with the client's need, a service client first searches for candidate services from the service registry. When these have been found, the service client submits the following information to the competition desk:

1. **Candidate Services** which are described by WSDL (Web Service Description Language) documents obtained from the service registry.

2. **Test Cases** where each test case is represented by a couple (*Service Parameters*, *Expected Response*). For a data service, *Service Parameters* include the remote service parameters and *Expected Response* is the expected remote service response. However, a task service requires both remote and local service parameters as *Service Parameters* as well as the corresponding local service response (*Expected Response*).

3. **Competition Policies** define the relevant competition factors such as accuracy, cost, response time and other performance metrics. Each of these factors is associated with an evaluation function to guide the competition organizer to obtain the required parameter values. Finally, a ranking policy is included to rank services based on their results from the competition. The ranking policy determines the weight of each factor in the service ranking.

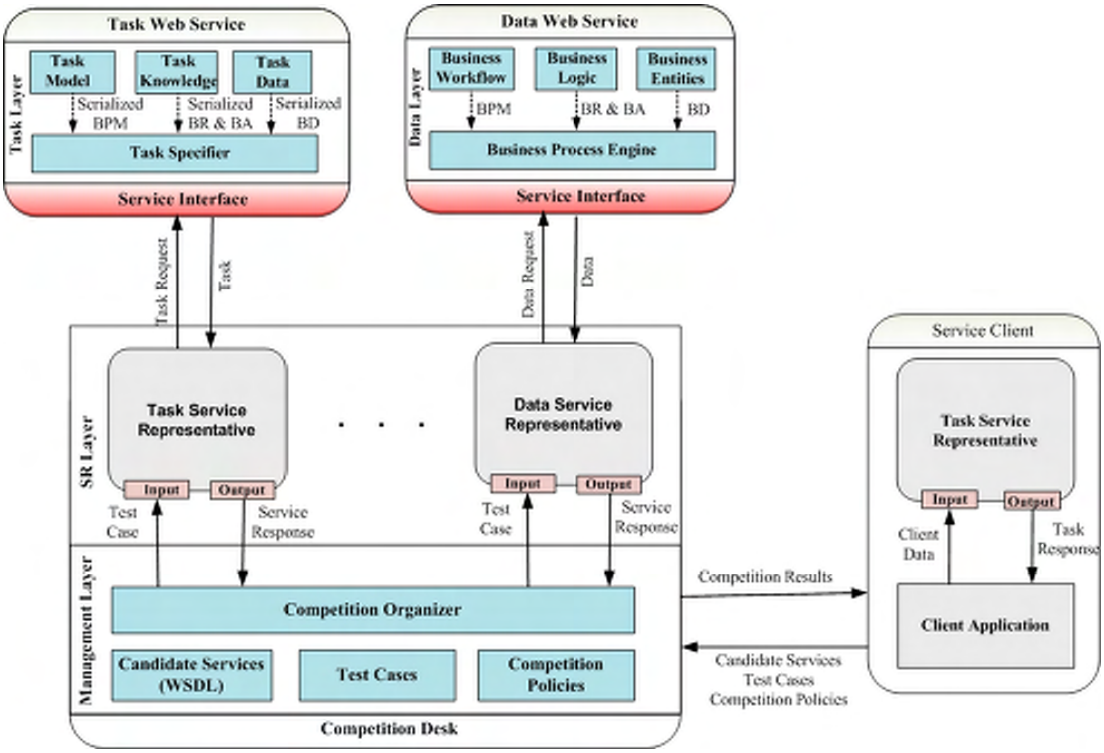


Figure 6.3. Architecture for the proposed web service competition. The competition desk holds a competition among the service representatives of the candidate services using test cases and policies submitted by the client.

6.2.2 Services Provider

Corresponding to the type of service, each service provider has either a data or task layer where the former performs the server-side processing of a web service and the latter defines a task for the service representative to perform client-side processing. A hybrid service that includes both client-side and server-side processing will require both data and task layers. The enterprise business components are divided between these two layers. While the data layer applies the business components, the task layer sends the business components to the client-side to be applied by the service representative.

Service Interface. This component supports the communication contracts (message-based communication, formats, protocols, security, exceptions, etc) for the services.

Data Layer. Server-side business processes, rules and actions, and data are stored in the business workflow, logic, and entity components, respectively. The business process engine executes the corresponding business process with each service. This layer responds to the client with a single-segment response message (a data message).

Task Layer. Client-side business processes, rules and actions, and data are stored in the task model, knowledge, and data components, respectively. Since business components in this layer are sent to the client side, they must be serializable. The task specifier provides the required model, knowledge, and data for each task request to be sent by a three-segment response message (the task message) to the client.

6.2.3 Competition Desk

The competition desk manages the web service competition and has two layers: a service representative layer and a management layer, as follows.

Service Representative (SR) Layer. This layer includes the service representatives of the candidate services. A service representative is a generic software agent that represents a service at the competition desk (or client side) where it has different components for task and data services, as follows.

- *Task Service Representative*: This agent, shown in Figure 6.4, performs a task with the following components.

  - *Service Stub* : invokes a task service by sending its remote service parameters to receive a task message.
  - *Input*: inputs local service parameters of each test case from the competition organizer.
  - *Knowledge Base*: stores basic and internal business rules and actions to relieve the service provider from sending them each time. Using this internal knowledge base, a service representative can be customized for each domain.
  - *Business Process Engine*: executes a task instance by applying business rules and performing business actions on business data.
  - *Task Manager*: provides the following functionalities to support the entire life cycle of a task instance (i.e., from creation to termination).

    1. *Task Invocation*: calls a task service through the service stub to receive a task message.
    2. *Task Instantiation*: creates an abstract business process based on the task model and then realizes the abstract process using internal and external task knowledge to generate a task instance.

3. *Task Execution*: passes the task instance with the relevant business data (as received from the input component and task data segment) to the business process engine to be executed. Finally, the task manager sends the task results to the output.

– *Local Memory*: stores the process variables of the task process during task execution.

– *Output*: outputs task service response for each test case to the competition organizer.

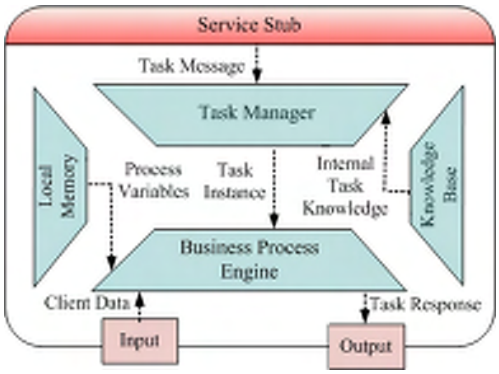


Figure 6.4. Task service representative components.

- *Data Service Representative*. This agent, shown in Figure 6.5, is assigned to a data service and invokes it for each test case using the following components.

  – *Service Stub* : invokes the data service for each test case to receive the corresponding service response.

  – *Input*: inputs service parameters of each test case from the competition organizer.

  – *Output*: outputs data service response to the competition organizer.

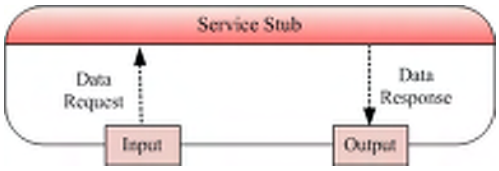


Figure 6.5. Data service representative components.

Management Layer. The competition organizer sets up a service competition based on the information received from the client, executed in the following four phases.

1. *SR Instantiation Phase*: creates one generic Service Representative (SR) for each candidate service. Then, it configures each SR's service stub using the WSDL description (which specifies a service address and port) of the corresponding service. When this phase is completed, there is an assigned SR for each candidate service that can communicate with the service.

2. *SR Initialization Phase*: calls the task manager of each task service representative to invoke its task from the assigned service and then to instantiate the task. There is no initialization phase for data service representatives.

3. *Competition Phase*: passes the test cases one by one to each SR and waits until they respond. During the competition, it collects information regarding the specified competition factors (such as response times). When the competition organizer receives all the service responses, it computes the value of each competition factor for each candidate service, based on the evaluation functions specified by the client.

4. *Ranking Phase*: applies the ranking policy to the results obtained, in order to rank the candidate services.

This layer also contains internal components storing competition policies, test cases, and candidate services in different categories that can be used by clients when they cannot or do not intend to provide this information.

6.3 Case Studies

To evaluate the effectiveness and feasibility of service selection based on the proposed web service competition approach, a prototype system of the proposed architecture was developed including the competition desk, data and task service providers, data and task service representatives, and service client. This prototype, *WS-Competition version 1.0*, is based on *J2EE* 1.5 technologies and the *Apache Tomcat* 6.0 application server. Task service representatives use *Drools* version 5.0 as their business process engine to execute task instances. The competition desk has an internal *MySQL* database to store test cases. Business process models are converted to XML format; business rules are expressed by production rules and encoded by *PMML* standard version 3 (Raspl [2004]); business actions are defined by Java statements or functions; and business data are defined by Java beans and serialized to form task messages. Two web service competition scenarios were run among data and task services and reported in this section. There are two issues that are addressed:

Figure 6.6. Two test cases for the data service competition: (top) a test case in experiment one with the expected result = "Yes" ; (bottom) a test case in experiment three with the expected result = "No". The former represents the client's interest in face recognition services which recognize facial orientation, but in the latter, the client searches for a face recognition service which works under variable illuminations.

1. Web services have been used widely as remote procedures inside web applications. Mobile applications also invoke web services to communicate with their own web servers. While most of web services are developed to be used by the web and mobile applications of the same enterprise, there are not many web services available with similar functionalities for the public. However, as cloud computing is getting more attention, the number of public and similar web services will increase in the future. As a result, this research focused on developing our own web services which deliver the same functionality but by applying different approaches.

2. Existing approaches for service discovery compare web services based on their descriptions, including service interface and service functionality. They take a client's query and then try to match keywords in the query with service descriptions using a semantic model (e.g., OWL-S Martin et al. [2005]). Consequently, they are very sensitive to client query and service descriptors. Services with the same functionality have similar descriptions and there-

fore description-based approaches would fail to compare them efficiently. In other words, description-based web service discovery is useful to extract relevant services for client request, but the proposed performance-based approach aims to select the best service from services with the similar functionalities. As a result, this approach was evaluated in a general case where each of the candidate services has a chance to be selected by a description-based service selection approach. For this purpose, the research shows the winner in each competition as the best service among candidate services, without considering descriptions.

6.3.1 Data Service Competition

A client searches for a face recognition web service that verifies whether facial images belong to the same person. Although there are several algorithms for face recognition, there is no evidence to show that any one of them is consistently the best under all circumstances (e.g., different lighting and orientation conditions) (Jafri and Arabnia [2009]).

Candidate Services. Each service takes a number of facial images belonging to the same person as well as an unknown facial image (target face). Then it verifies whether the target face represents that person. This result is associated with a confidence level that ranges between 0 and 1, where 0 represents zero confidence that a match has been made, and 1 represents full confidence in the match.

$$WS\ (face\ 1, \cdots, target\ face) = (Yes\ /\ No,\ confidence)$$

In this case study, the candidate services include five face recognition services respectively based on *Principal Component Analysis* (PCA), *Independent Component Analysis* (ICA), *Linear Discriminant Analysis* (LDA), *Support Vector Machine* (SVM), and *Incremental Neural Network* (NN) techniques (these techniques are discussed by Jafri and Arabnia [2009]). In addition to a recognition technique, a face recognition service requires a data set that includes a training set of facial images. To verify the target face, each service adds the known faces to its training set; then it builds or completes its recognition model; and finally the model is applied to the target face.

Test Cases. Each test case is represented in the form of *(face 1, . . . , target face, expected response)* where the expected response is either "Yes" or "No". The service client chooses test cases that have conditions similar to the actual cases that he/she wants to have recognized.

Competition Policies. The client specifies the competition factors as *Accuracy*, *Confidence*, and *Response Time* with the following evaluation functions, where S and F represent the set of successful and failure test cases, respectively, for a service and $\mid A \mid$ represents the cardinality of set A.

$$Accuracy = \frac{\mid S \mid}{\mid Test\ Cases \mid}$$
$$Confidence = \frac{\sum_{i \epsilon S} confidence_i - \sum_{i \epsilon F} confidence_i}{\mid Test\ Cases \mid}$$
$$Time = Average(Response\ Time)$$

Finally, the client assigns weights of 0.5, 0.3, and 0.2 to *Accuracy*, *Confidence*, and *Time* in the ranking policy.

Experimental Results. Three experiments were run with the prototype system to discover the best candidate services in each specific condition. *ORL* [2] and *Yale* [3] face data sets were used in these experiments. *ORL* includes 400 facial images of 10 persons while *Yale* has 165 facial images of 15 persons. Facial images were divided into three sets: (1) training data, (2) competition data, and (3) evaluation data, in the proportions of 60%, 5%, and 35%, respectively. The training data were used by the candidate services to build their recognition models; competition data were the test cases; and evaluation data were used for evaluating the competition results.

- Experiment 1 : training and competition data belong to the same data set (*ORL*); the client submits five test cases where each test case contains two known facial images.
- Experiment 2 : training and competition data belong to the same data set (*Yale*); the client submits five test cases where each test case contains five known facial images.
- Experiment 3 : training data belong to the *ORL* data set while the competition data belong to the *Yale* data set; the client submits five test cases where each test case contains three facial images.

In each experiment, the client submitted to the competition desk the WSDL document for candidate services, the corresponding test cases, and the defined competition policies. Figure 6.6 shows two instances of the corresponding test cases. Table 6.1 represents the competition results and the winner in each experiment (Rank =1), based on a 2.4 GHZ dual-core CPU. A high-speed bandwidth (1 Mbyte/Sec) link was used between the servers and the competition desk.

To verify whether the service competition guides the client to select the best service, the performance of each web service was calculated in each experiment. The evaluation data (50 facial images in each experiment) were selected from the same data set as the competition data. Table 6.2 represents the results of this evaluation where the competition desk recommended the best service (Rank=1) in all three cases. The total ranking obtained from the competition had a 73 percent accuracy in comparing the rankings on the evaluation data.

[2] http://www.cl.cam.ac.uk/research/dtg/attarchive/facedatabase.html
[3] http://vision.ucsd.edu/ leekc/ExtYaleDatabase/ExtYaleB.html

Table 6.1. Experimental results for the data service competition using five different face recognition web services.

No	Competition Factor	PCA WS	ICA WS	LDA WS	SVM WS	NN WS
1	*Accuracy*	60%	80%	40%	80%	40%
	Confidence	38%	66%	16%	61%	12%
	Time (msec)	1690	2404	1012	6214	2605
	Score	***0.53***	***0.67***	***0.44***	***0.61***	***0.31***
	Rank	***3***	***1***	***4***	***2***	***5***
2	*Accuracy*	80%	100%	100%	100%	60%
	Confidence	49%	69%	87%	71%	39%
	Time (msec)	2237	2873	1148	8519	3012
	Score	***0.64***	***0.78***	***0.96***	***0.73***	***0.48***
	Rank	***4***	***2***	***1***	***3***	***5***
3	*Accuracy*	60%	80%	60%	80%	60%
	Confidence	43%	68%	41%	48%	29%
	Time (msec)	2023	2603	1129	7147	2078
	Score	***0.54***	***0.69***	***0.63***	***0.57***	***0.48***
	Rank	***4***	***1***	***2***	***3***	***5***

6.3.2 Task Service Competition

A skin detector service (Kakumanu et al. [2007]) is a primary need in many fields including face detection, semantic filtering of web contents, video surveillance, and human motion detection. A skin detector data service requires transferring images and videos from clients to the server, which is not efficient. In comparison, a skin detector task service which uses a client-side service representative to evaluate the skin regions offers higher performance.

Candidate Services. Several methods for discriminating between skin and non-skin regions have been proposed. They can be categorized into pixel based and block based approaches. The former provides faster results while the latter provides more accuracy. A client should choose a skin detector service based on its performance in the specific client application.

In this case study, the client wants to evaluate the performance of three skin detector web services: pixel-based; block-based using color features; and block-based using texture features (these techniques are discussed in Sajedi et al. [2007]). To show that a task service outperforms a data service for skin detection, both data and task services for each of the candidate services were developed. Then their service response times were compared for a sample image (450 Kbytes). The comparison results are shown in Table 6.3. Processing time and network time are the factors

Table 6.2. Evaluation results for the data service competition.

No	Competition Factor	PCA WS	ICA WS	LDA WS	SVM WS	NN WS
1	*Accuracy*	57%	72%	42%	68%	33%
	Confidence	31%	39%	19%	38%	12%
	Time (msec)	1720	2411	1123	6094	2889
	Score	***0.51***	***0.57***	***0.46***	***0.49***	***0.27***
	Rank	***2***	***1***	***4***	***3***	***5***
2	*Accuracy*	76%	89%	92%	88%	64%
	Confidence	39%	61%	79%	64%	37%
	Time (msec)	2325	3103	1059	8618	2097
	Score	***0.59***	***0.69***	***0.89***	***0.65***	***0.53***
	Rank	***4***	***2***	***1***	***3***	***5***
3	*Accuracy*	52%	77%	57%	69%	51%
	Confidence	26%	59%	31%	47%	26%
	Time (msec)	2051	2526	1145	6985	2106
	Score	***0.44***	***0.65***	***0.57***	***0.51***	***0.45***
	Rank	***5***	***1***	***2***	***3***	***4***

Table 6.3. Service time comparison between data and task skin detector web services.

Service Time (msec)	Pixel WS	Block-Color WS	Block-Texture WS
Task Service	582	1241	1351
Data Service	1094	1519	1594

that influence total response time. To boost data service performance, a dedicated server (it has no other tasks or clients) was used with a processor that was two times faster than the client processor (2.4 GHZ dual-core CPU). The service client and servers were connected by a 1 Mbyte/Sec link. While a task service receives a request message (1 Kbyte) and sends a task message (up to 3 Kbytes) to a client-side service representative, a data service receives an original image (450 Kbytes) and replies by sending the modified image (120 Kbytes). Moreover, while a data service must be called for each video frame, a task service which needs to be called once assigns a task to a service representative to process the frames locally. The skin detector task services work as follows.

1. *Pixel-based skin detector*: assigns a task to the generic service representative to apply explicit rules for the color values (i.e., Red (R), Green (G), and Blue (B)) of each pixel. Different components of this task service are shown in Figure 6.7.

2. *Block-based skin detector using color features*: sends the following task definition as well as skin patterns (the server-side task data) to the generic service representative to customize it for this service. i) Subdivides the image into equal sized blocks; ii) extracts three color features (color mean, color variance, color skewness) from each block; iii) compares each block features to all the skin patterns received from the service and chooses the blocks whose Mean Square Error (MSE) is less than a defined threshold as skin blocks; and iv) paints skin blocks black and non-skin blocks white. Figure 6.8 shows the task components for this web service.

3. *Block-based skin detector using texture features*: assigns a task to the generic service representative similarly to the previous service except that it considers texture features calculated by wavelet transforms for each block.

As mentioned before, a service representative is generic and can represent any services; however, the SR internal knowledge base enables the service representative to store the required business rules and actions for a specific domain. In this case study, each task service asks the competition desk to assign it a service representative that has basic image processing functions stored in its internal knowledge base. Otherwise, the service providers are asked to send the required knowledge to the generic SRs.

Test Cases. The client provides test cases in the form of $(image, expected\ result)$ where the expected result is an image with the same number of pixels, but skin and non-skin pixels are assigned black and white colors, respectively.

Competition Policies. The client specifies the competition factors as *Accuracy* and Response *Time* with the following evaluation functions where FP and TN represent the False Positives (number of non-skin pixels considered as skin pixels) and True Negatives (number of not recognized skin-pixels), respectively.

$$Accuracy = Average(\frac{\mid Pixels \mid - 2*TN - FP}{\mid Pixels \mid})$$
$$Time = Average(\ Response\ Time\ )$$

In this case study, the client is searching for a web service that captures most of the skin pixels, so the accuracy evaluation function assigns more weight to TN. Different ranking policies were used for each experiment (discussed next).

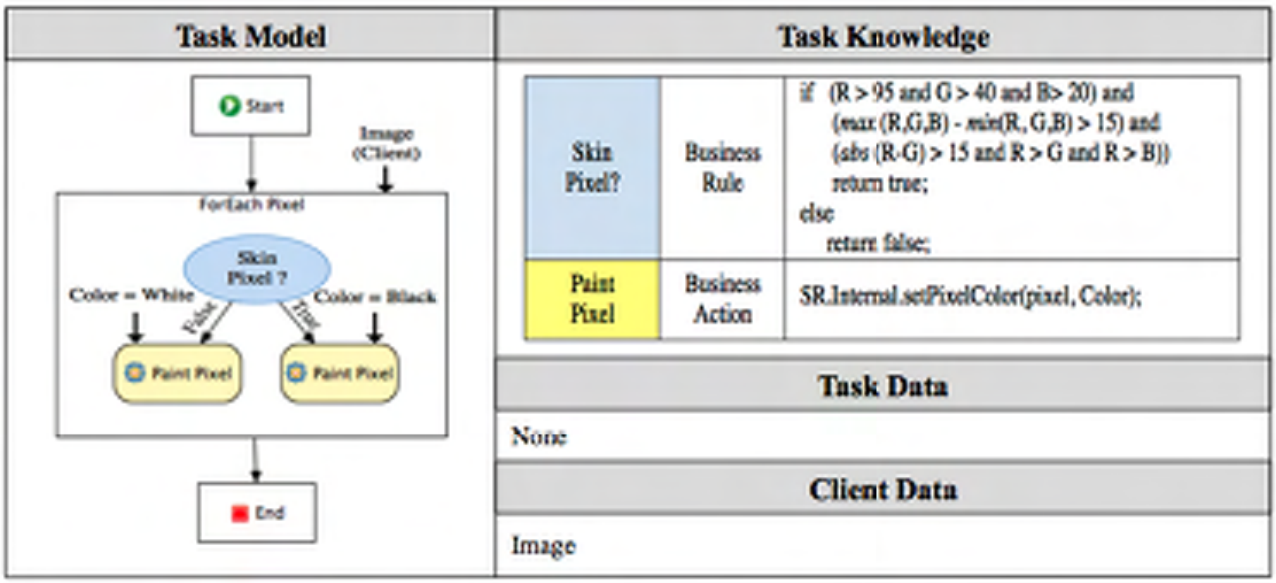


Figure 6.7. Task service components of the Pixel WS where there is no server-side task data. In the task model, a business rule is assigned the '?' symbol to differentiate from a business action. *SR.Internal.x* **refers to the business actions stored in the SR internal knowledge base.**

Experimental Results. 60 random images were collected, where all images were in color with various visual qualities, details and different light conditions. These images were equally divided into competition data and evaluation data. The following two experiments were run with the prototype system to determine the best services in each client application.

- Experiment 1: the client uses this web service for face detection; therefore the test cases contain facial images. Since the client prefers accuracy rather than speed, 0.9 and 0.1 were assigned as the weights for *Accuracy* and *Time*, respectively, in the ranking policy.

- Experiment 2: the client uses this web service for web content filtering; therefore test cases contain body and non-body images. Since this web service will be used as a part of a live video streaming system, it needs to be fast. Therefore the client assigns 0.8 and 0.2 as the weights for *Accuracy* and *Time*, respectively, in the ranking policy.

In each experiment, the client submitted five test cases (two instances are shown in figure 6.9). Table 6.4 represents the competition results and the winner service in each experiment. Similarly to the first case study, the performance of each web service was calculated from the corresponding evaluation data (which includes 30 images) in each experiment. Table 6.5 presents the results of this evaluation, confirming the service competition results.

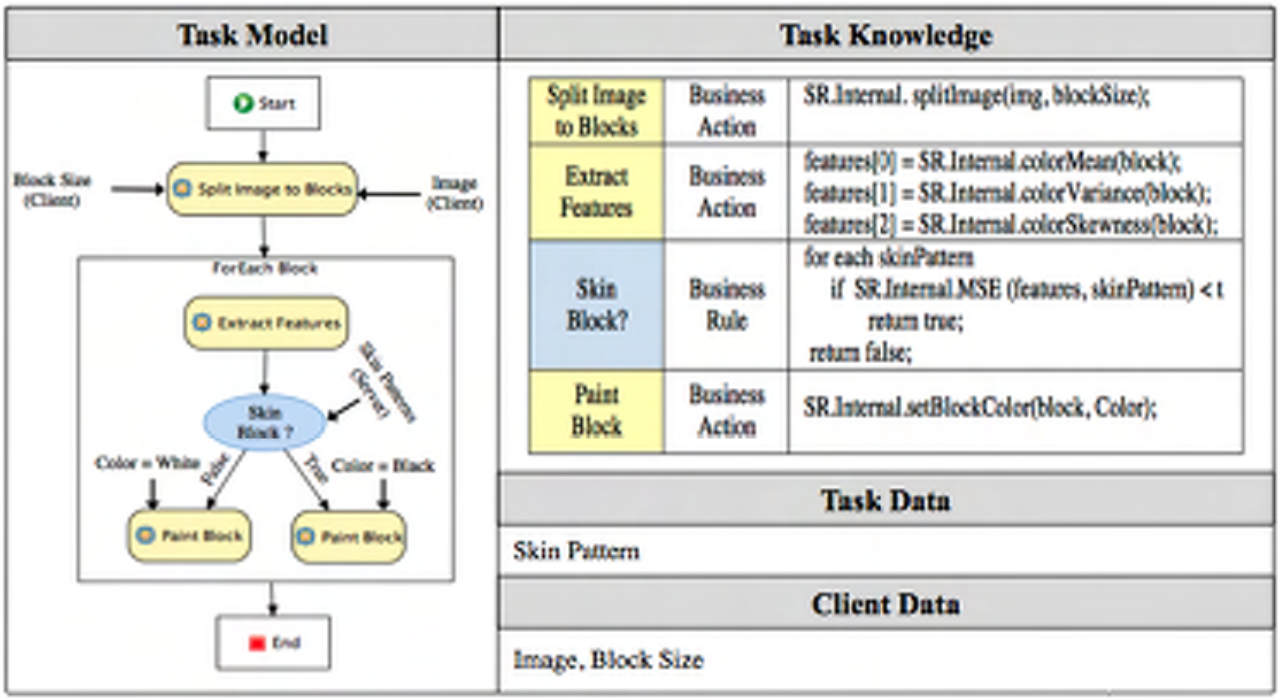


Figure 6.8. Task service components of the Block-Color WS where the service provider sends skin patterns as server-side task data.

6.4 Discussion

The proposed web service competition is differentiated from other web service selection methods since it ranks services based on their performance in a specific client's context. This section lists a few important issues such as the applications and challenges of the proposed WS-Competition approach.

1. The proposed web service competition is not intended to replace traditional web service selection approaches. However, this approach provides a more accurate selection mechanism, compared to traditional approaches, in the following cases: i) The service descriptions are not accurate or comprehensive enough when the services are to be ranked based on their performance. ii) The service client is to choose the best service among multiple candidate services where they represent different performance levels based on the client's context or specific application. iii) The service client specifies the service ranking criteria to test and compare available services.

2. The proposed service competition can not be offered free of charge for pay-by-use services because of costs associated with tests that involve the competition desk. However, some services

Table 6.4. Experimental results for the task service competition using three skin detection services.

No	Competition Factor	Pixel WS	Block-Color WS	Block-Texture WS
1	*Accuracy*	91%	85%	89%
	Time (msec)	829	2071	2979
	Score	***0.92***	***0.79***	***0.82***
	Rank	***1***	***3***	***2***
2	*Accuracy*	67%	83%	86%
	Time (msec)	837	2004	3709
	Score	***0.73***	***0.76***	***0.74***
	Rank	***3***	***1***	***2***

Table 6.5. Evaluation results for the task service competition.

No	Competition Factor	Pixel WS	Block-Color WS	Block-Texture WS
1	*Accuracy*	88%	81%	83%
	Time (msec)	823	2012	3020
	Score	***0.89***	***0.77***	***0.77***
	Rank	***1***	***2***	***2***
2	*Accuracy*	64%	85%	81%
	Time (msec)	817	2051	3631
	Score	***0.71***	***0.76***	***0.69***
	Rank	***2***	***1***	***3***

provide free trials or test versions of their web services which could be used by the competition desk. Otherwise, the competition by itself may have costs which must be paid by the service client. To support these cases, a billing model for the competition desk is required.

3. The competition desk can be provided as a web service search engine where the service client sends a query including the category of services as well as the search criteria (competition factors). Moreover, the service client either submits its own test cases or asks the search engine to use its predefined test cases for the specified service category, in order to hold the web service competition. Finally, the web service engine returns a ranked list of services to be selected by the service client.

4. The introduction of an actor in the SOA model and the associated tests may increase the required time for service selection. However, in most cases, service selection includes a long-term agreement between the service client and the service provider. Consequently, the service client would prefer service selection approaches which offer more accurate and customized results rather

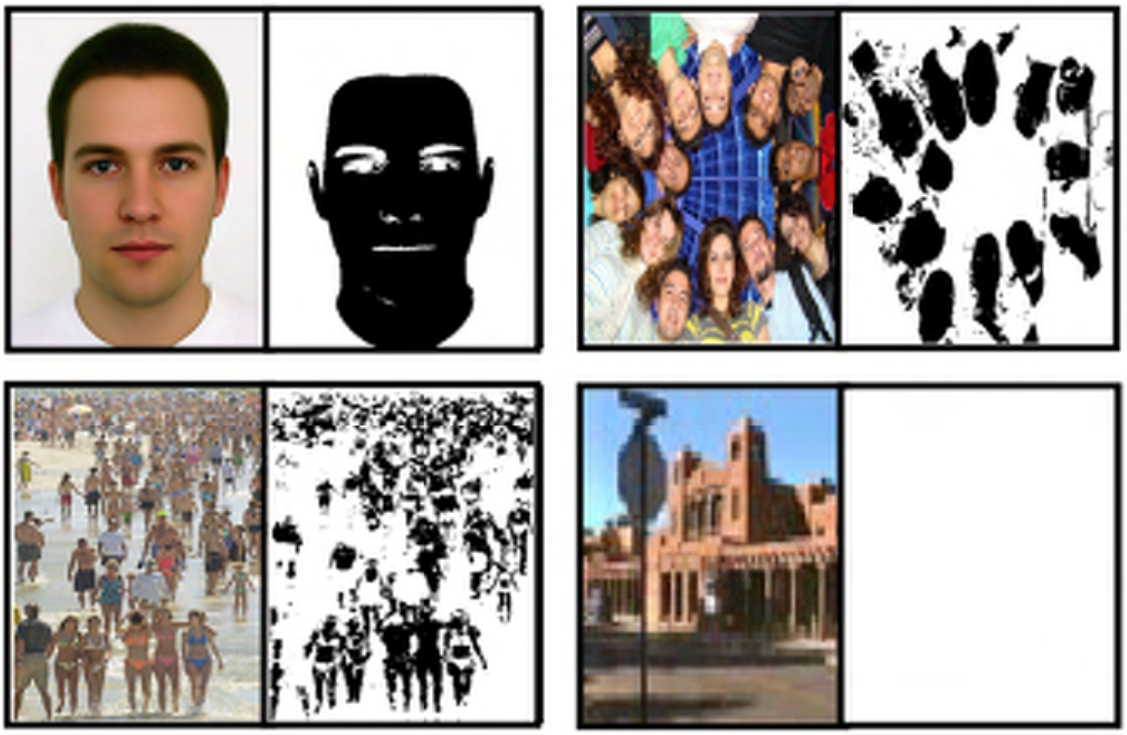


Figure 6.9. Four test cases *(image, expected result)* **for the task service competition: (top) represents two test cases for experiment one where the client needs this service for a face detection application; (bottom) represents two test cases for experiment two where the client will use the service for web content filtering.**

than those that generate their results faster. However, this may limit the applications of web service competition for dynamic service discovery, which requires run-time re-binding to new and un-known services whenever the actual QoS deviates from initial estimates, or when a service is not available.

6.5 Related Work

Current web service discovery is based on *UDDI* (Universal Description, Discovery, and Integration), a standard for centralized repositories. Several protocols (such as *Jini*, *UPnP*, and *Salutation*) (Hagemann et al. [2007]), middle-wares (Gu et al. [2005]), and frameworks (Meshkova et al. [2008]) address service discovery, all based on the service descriptions published in a *UDDI* service registry. Semantic web technology has been applied to create and analyze web service descriptors. For example, *WS-Inspection* (Ballinger et al. [2001]) is a web service specification for discovery documents that describes services at different levels and from various perspectives.

Similarly, Kuster et al. [2008] proposes semantic and ontology-based service descriptors to be used in service discovery. Formal methods are another way of describing services. For example, Benatallah et al. [2005] uses description logic and the best covering problem to propose a formal matchmaking algorithm, and Shen and Su [2005] propose a behavior model to describe the operational level of web services using automata and logic formalisms. Service descriptions are promising in theory, but unfortunately most of the descriptions available are low quality. Therefore, a major challenge that web service technology faces is the discovery and selection of services, based on their capabilities and performance. That challenge has been addressed in this work.

The integration of software agents and web services has been proposed to model the business aspects of enterprise systems, where each role or major function of an enterprise system is considered as an agent. Moreover, autonomous computing and software agents have had increasing application in service discovery. For example, Kokash [2005] proposes agents that rank services by analyzing the client's behaviour and satisfaction about services. The Service Location Protocol (SLP) (Guttman [1999]) has also been used in service discovery including user, service, and directory agents. In comparison to that approach, this research has assigned a new role (i.e., service representative at the competition desk) to software agents to model enterprise agents in the business domain. As an alternative to the suggested generic service representatives, one could consider sending customized mobile agents from the candidate services to the competition desk. However, there are several security and privacy issues to be considered in mobile agent computing (Hadzic et al. [2009]). Mobile agent architectures also suffer from low efficiency, as they need to transmit the entire computer program or process through the network.

As a part of the proposed web service competition, the competition desk implicitly tests the candidate web services based on the test cases submitted by a service client. Several approaches have been proposed for web service testing, as surveyed in Canfora and Di Penta [2009]. For example, a scenario-based testing method was proposed by Tarhini et al. [2006] which requires an extension to WSDL service descriptions. Tsai et al. [2002] presented a testing framework which supports both test execution and test scenario management using a test master and a test engine, respectively. While the test master generates test cases from the WSDL description, the test engine interacts with the web service to provide tracing information for the test master. Moreover, there are several approaches which propose to extend the UDDI registries with testing features (Kourtesis et al. [2008] and Bertolino and Polini [2005]). Based on these approaches, the service provider releases the test suites together with the service description to the service registries. As a result, the service registry moves toward an active role as a service tester. The web service competition proposed in the current research is differentiated from other service testing approaches by at least two factors: (i) to the best of our knowledge, the web service competition

is the very first attempt to test web services based on the client's context; and (ii) the testing of client-side web services (e.g., task services) has not been covered in previous work.

6.6 Conclusions

Service-based software development addresses the integration of applications via the web. For efficient service discovery, a service client needs to know both the functionality of each candidate service and its capability to perform well for the client's specific application. Because service descriptors can not support efficiently both the functional and non-functional aspects of a service, a service client is forced to either examine many potential services or simply make a random choice. In this chapter, a mechanism is proposed to help a service client to select web services based on both their performance and the client's needs. For this purpose, the SOA infrastructure model was extended through a component named the service competition desk, that holds a competition among alternative services according to information provided by the service client. Moreover, the proposed service selection approach supports both traditional server-side data services and client-side task services.

Chapter 7

Service Composition Certifier [1]

In this chapter, a new component, *Service Composition Certifier (SCC)*, is proposed as an addition to the SOA model, to certify client-defined service composition models including server-side data services and client-side task services. The certification process includes two phases: first, the SCC verifies the composition model to check whether the collaborating services are compatible. Second, the SCC validates the composition model based on a policy submitted by the client to check whether this model meets the client's expectations. For this purpose, the web service is modeled by its business components including: business process, business action, and business data. While a data service applies the business components at the server side, a task service serializes and sends them to the client side to be applied by the service representative. Then, a formal model of simple and composite web services is provided by formalizing their construction of business components using a process algebraic model (Calculus of Communicating Systems). On the other hand, a service client interacts with a web service by sending, sharing, and receiving service input and output parameters which are business data. Consequently, the service composition policy which describes the client's expectations from a composite service can be defined in terms of different relationships among the relevant business data. In this chapter, a formal model of the service composition policy is proposed using a temporal logic (Modal μ-Calculus). Using the formal models of the composite service and the client's policy, the proposed SCC uses its model checker to certify the service composition model for the given policy. A case study describes how the proposed SCC works.

[1] This chapter is mostly based on the following paper: M. Najafi, K. Sartipi, and N.Archer, *Formal Verification and Validation of composite Web Services Using Service Composition Certifier*, Journal of Computer Networks. Elsevier 2011 (Submitted).

7.1 Introduction

In the next Internet generation (the Personal Web), the user is moved to the center of gravity for web integration. Consequently, the user is envisioned to be able to compose web services statically or dynamically at either the client side or the server side. The Business Process Execution Language (BPEL) standardizes the composition of web services to provide more sophisticated business services. BPEL processes are currently executed by a centralized server side orchestration engine. For integrated business process execution, issues such as scalability, modularity, privacy, and security can be difficult to manage in such situations. Moreover, server side processing of web services requires transferring client data to the service providers, which may cause data privacy violations, security breaches, or network traffic overloading. Therefore,this research has proposed task services which are web services with the capability of processing the client data at the client side using a generic software agent called the *Service Representative*.

In a distributed process execution environment, client side task services can be composed with traditional server side services (data services) through a BPEL process. Such collaborative and distributed business process execution can increase the complexity of service composition in comparison with traditional (server side) and integrated approaches. As one of the main requirements for this user-centric and dynamic environment, the client-defined composite web services must be verified (to check whether the partner services are compatible) and validated (to check whether the composite service satisfies the client's expectations) automatically.

This chapter proposes an extension to the SOA model (Figure 7.1) by adding a new component called the *Service Composition Certifier (SCC)* to formally verify and validate composite services including both data services and task services. For this purpose, the dynamic behaviour of services is formally modeled with business components including business processes, business actions, and business data. Then, the service provider publishes this model as part of the service description in the service registries. A service client can create a customized composite service by composing data and task services in a BPEL model. In addition, the service client specifies its expectations from the composite service (*Service Composition Policy*) in terms of relevant business components. Then the service client submits both the BPEL model and the service composition policy to the SCC. Using the service models in the service registries, the SCC creates a formal model of the entire system. Finally, it formalizes the client-defined service composition policy so it can be checked automatically by its internal model checker.

The organization of this chapter is as follows. The next two sections cover the basis of the mathematical formalism behind the service composition certifier. Formal models for business components are proposed in Section 3. Simple and composite web services are formalized in Section 4. The service composition policy is defined and formalized in Section 5, and the service

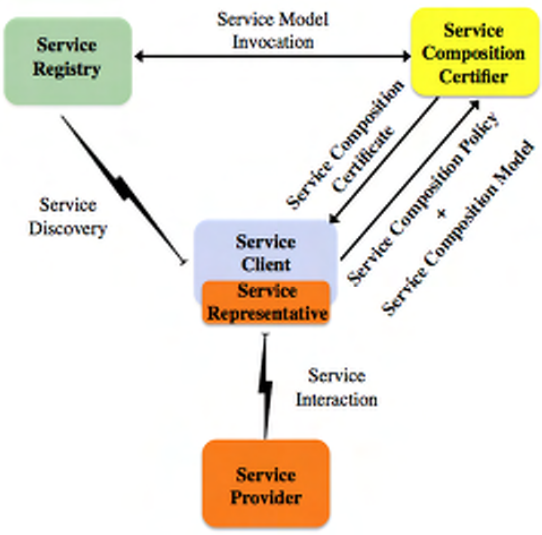


Figure 7.1. High level view of the proposed Service Composition Certifier (SCC) in the SOA model.

composition certifier is proposed in Section 6. A case study in the business domain that utilizes the proposed approach is described in Section 7. Section 8 discusses previous work related to the proposed approach, and conclusions are presented in Section 9.

7.2 Calculus of Communicating Systems

The nature and features of simple and composite web services suggest the use of process algebra to formalize them. Among the different available notations, the Calculus of Communicating Systems (CCS) (Milner [1982]) was chosen, because it is expressive enough for formalizing business components. In addition, CCS is widely supported by automated tools required by the certifier component, while other such notations lack complete tool support.

In CCS, a system is modeled as a collection of processes that communicate with each other by sending and receiving messages through a number of channels. CCS does not model time, and process communication is asynchronous. Where $[[P]]$ is a notation to represent the algebraic model of process P, CCS components are defined as follows.

1. A CCS atomic action α is defined by

$$\alpha ::= \ a?(x) \mid a!(x) \mid \tau$$

   Where a is the name of a channel, x is data (or message). $a?(x)$ represents an atomic input action where the message x is received through the channel a. Conversely, $a!(y)$ represents an atomic output action where the message y is sent over the channel a, τ represents an internal (or silent) action.

2. Given a set of atomic actions, a single or composite CCS process P is defined by the following grammar:

$$P ::= 0 \mid \alpha \cdot P \mid P + P \mid P||P$$

0 denotes an empty process or deadlock. Each process can be prefixed by an atomic action α, or composed (either using the choice + operator or the parallel || operator) with other processes. For example, $a!(x).b?(y).P$ represents a process that sends the message x through the channel a, waits to receive message y via the channel b, and then behaves like process P.

3. Parallel processes are directly convertible into the algebraic model.

$$Par ::= \ parallel\ P1\ P2\ \cdots\ Pn$$

$$[[Par]] \ = \ P1 \ || \ P2 \ || \ \cdots \ || \ Pn$$

However, a sequential composition of processes is not directly translatable. In fact, a sequence has to be defined using parallel composition as follows. (a_1) [[P]] (a_2) is defined in such a way that process P waits to receive a signal (i.e., a message) on channel a_1; then after proceeding with all actions in P, a signal on channel a_2 is sent.

$$Seq ::= \ sequence\ P1\ P2\ \cdots\ Pn$$

$$[[Seq]] = sq_1!()||(sq_1)[[P1]](sq_2)|| \cdots || \ sq_n?() \cdot [[Pn]]$$

Consequently, the parallel processes wait to receive their own signal to proceed with their operation, and send a signal to the next parallel process to proceed in sequence. Moreover, this consideration allows only one process to run at each time instance.

4. A conditional construct is where one out of several sets of activities is executed based on the evaluation of conditions, and expressed using the choice operator in CCS. The resulting process $[[Cond]]$ will be nondeterministically processed by one of the branches. Since data are not modeled in CCS, the translation to CCS abstracts away some details like conditions.

$$Cond ::= \ \ If\ \ condition\ Then\ P1\ Else\ P2$$

$$[[Cond]] = \tau \cdot [[P1]] + [[P2]]$$

5. To model a loop construct in CCS, a signal is sent on the channel $loop()$ as long as another iteration of process P is to be performed.

$$Whl \ ::= \ while\ condition\ P$$

$$[[Whl]] = \tau \cdot 0 + [[P]](loop)||loop?() \cdot [[Whl]])$$

7.3 Modal μ-Calculus

Modal μ-Calculus (Bradfield and Stirling [2007]) is an extension of propositional logic that extends the standards of formal logic to include the elements of modality (possibility and necessity) and fixed point [2] combinators, allowing the definition of recursive propositional formula. Modal μ-Calculus is widely used to describe properties of labeled transition systems and verifying these properties.

CCS processes generate labeled transition systems $(P, \{\xrightarrow{a} | a \epsilon \Lambda\})$ where P is a non-empty set of processes; Λ is an action set; and $\xrightarrow{a}$ is the transition relation for each a belongs to Λ. As an example, Figure 7.2 represents a transition system for three different vending machines.

Letting K range over subsets of Λ, a propositional formula (or simply a formula) A in modal μ-Calculus is defined using the following syntax.

$$A \ ::= \ T \mid \neg A \mid A_1 \wedge A_2 \mid [K]A \mid Z \mid \nu Z.A$$

A formula A is either the constant true T; or a negated formula $\neg A$; or a conjunction of two formulas $A_1 \wedge A_2$. Disjunction is defined as $\vee$ or False F via negation and De Morgan's law. A modalized formula $[K]A$ means: *A holds after every (performance of any) action in K*. Similarly, $<K>A$ is defined as the dual of $[K]A$, meaning that *A holds after (a performance of) an action in K* . Finally, $[-K]$ and $[-]$ are abbreviations for $[\Lambda - K]$ and $[\Lambda]$, respectively.

$$< K > A = \neg[K]\neg A$$

A temporal property is described by a modal μ-Calculus formula and $E \models A$ can be used to express process E satisfies the property A. An alternative semantics inductively defines for each formula A where $\| A \|$ means the subset of processes in P having the property A.

$$\begin{aligned}
&\| \, true \, \| = P \\
&\| \neg A \| = P - \| A \| \\
&\| A \wedge B \| = \| A \| \cap \| B \| \\
&\| [K]A \| = \overline{[K]} \, \| A \|
\end{aligned}$$

Here $\overline{[K]}$ is the following process transformer where $P' \subseteq P$.

$$\overline{[K]}P' = \{E \, \epsilon \, P \mid \forall E' \, \epsilon \, P \cdot \forall a \, \epsilon \, K \cdot if E \xrightarrow{a} E' \, then \, E' \, \epsilon \, P'\}$$

The dual transformer $\overline{< K >}$ is defined as follows:

$$\overline{< K >}P' = \{E \, \epsilon \, P \mid \exists E' \, \epsilon \, P \cdot \exists a \, \epsilon \, K \cdot E \xrightarrow{a} E'\}$$

[2] A fixed point of a function f is a value x such that $f(x) = x$.

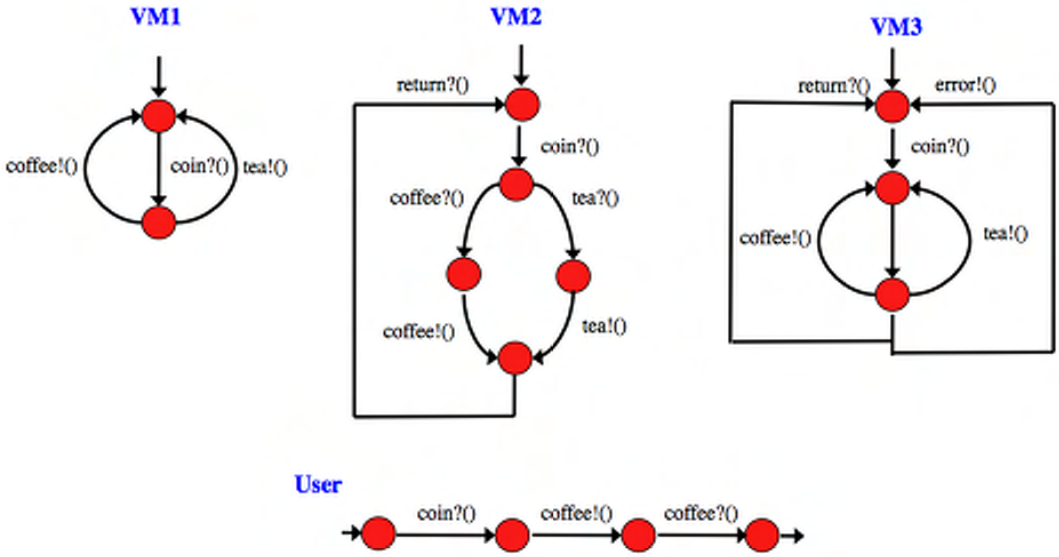


Figure 7.2. A transition system composed of four CCS processes including three vending machines and a user process.

A propositional variable denoted by Z and the fixed point operator νZ are used to define recursive properties. A recursive modal equation can be viewed as expressing (various) properties of transition systems. Each of these properties is determined by a solution to the recursive equation. In other words, a property expressed by a set is the common feature among all the processes in the set.

Consider the modal equation $Z =< a > Z$. A solution is a set of processes P' which obeys the following equality.

$$P' = \overline{< a >} P'$$
$$P' = \{E \epsilon P \mid \exists E' \epsilon P' \cdot E \xrightarrow{a} E'\}$$

An example is an uncluttered clock $Cl = tick \cdot Cl$ where each subset of processes generated by Cl solves the modal equation $Z =< tick > Z$, ordered by $\theta \subseteq \{Cl\}$ offering the smallest and largest solution. Where the least case expresses the same property as $False$, the largest case expresses a capacity to endlessly perform the $tick$ action.

$$\theta = \{F \epsilon P \mid \exists E \epsilon \theta \cdot F \xrightarrow{tick} E\}$$
$$\{Cl\} = \{F \epsilon \{Cl\} \mid \exists E \epsilon \{Cl\} \cdot F \xrightarrow{tick} E\}$$

The *Tarski* theorem (Tarski [1995]) guarantees that a recursive modal equation always has a least and a largest (which may coincide) solution, as follows.

Let P be any set and assume that $f : 2^P \longrightarrow 2^P$ is monotonic. Then f

i. has a least fixed point given by $\bigcap\{\epsilon \subseteq P \mid f(\epsilon) \subseteq \epsilon\}$

ii. has a greatest fixed point given by $\bigcup\{\epsilon \subseteq P \mid \epsilon \subseteq f(\epsilon)\}$

The process transformer $\overline{< K >}$ is monotonic. Therefore, $Z =< K > Z$ always has both a least and greatest solution. However, not every modal equation has solutions. For example, $Z = \neg Z$ can never be solved on any transition system. We can guarantee monotonicity on every transition system by restricting the form of an equation $Z = A$. This restriction is that every free occurrence of Z in A lies within the scope of an even number of negations.

The formula $\nu Z.A$ denotes the largest solution of the fixed point equation $Z = A$ where the operator $\nu Z.$ binds free occurrences of Z in A. An important derived operator is μZ that expresses the property given by the least solution to $Z = A$. The following examples are instances of properties defined by the fixed point operators.

- $\mu Z. < K > Z$ and $\nu Z. < K > Z$ express the property given by the least and largest solutions to $Z =< K > Z$, where the least case is of little important as it expresses $False$. The largest case expresses a capacity to endlessly perform K actions.

- $\mu Z = A \vee \ < K > Z$ expresses a capacity to continuously perform actions drawn from K until A holds. The maximal solution includes an extra possibility of performing K actions endlessly. In particular, when K is the complete set of actions, this formula expresses weak eventuality of A, that A may eventually hold.

- $\nu Z = A \wedge \ < K > Z$ expresses a capacity to endlessly perform K actions with A holding throughout. The least solution expresses $False$.

$[[K]]$ represents modality when internal action τ is not observable where $K \subseteq (\Lambda - \{\tau\}) \cup \{\epsilon\}$ and $\xrightarrow{\epsilon}$ means zero or more τ moves. The temporal logic employing the $[[K]]$ modalities is definable within the logic based on the $[K]$ modalities. However, it is useful to have the defined operators $[[K]]$.

The binders $\vee Z.$ and $\mu Z.$ are duals of each other, as follows. Here $\neg A[Z := \neg Z]$ is the result of replacing all free occurrences of Z in A with $\neg Z$.

$$\mu Z \cdot A = \neg \vee Z.\neg A[Z := \neg Z]$$
$$\nu Z \cdot A = \neg \mu Z.\neg A[Z := \neg Z]$$

These equations can be used to express important liveness properties such as *a must eventually happen*, implying that a occurs in every maximal sequence of actions. This property is given by $Z = Live \wedge [[-\{a, \epsilon\}]]Z$. Moreover, several properties can be defined using the temporal modal

Table 7.1. Basic properties defined with modal μ-calculus propositions.

Properties	Modal μ-calculus
Capacity to perform an action in K	$< K > T$
Inability to perform any action in K	$[K]F$
Inability to perform any action (deadlock)	$[-]F$
Capacity to perform action a and inability to perform other actions (necessity)	$< - > T \wedge [-a]F$
A process can not silently terminate (Live)	$[[\epsilon]] << -\epsilon >> T$
Capacity to endlessly perform K actions	$\nu Z. < K > Z$
Capacity to continuously perform actions drawn from K until A holds	$\mu Z.A \vee < K > Z$
Property A may eventually hold (Weak eventuality)	$\mu Z.A \vee < - > Z$
Action a must eventually happen (liveness)	$\mu Z.Live \wedge [[-\{a, \epsilon\}]]Z$
Absence of deadlock	$\nu Z.Live \wedge [[-]]Z$
Property A holds through a maximal performance of K actions even the process is finite	$\nu Z.A \wedge (< K > Z \vee [K]F)$
Capacity to endlessly perform K actions with A holding throughout	$\nu Z.A \wedge (< K > Z)$
Property A holds throughout every ongoing performance of K actions. (safety or property invariance)	$\nu Z.A \wedge [K]Z$

propositions such as *liveness* (i.e., something good does eventually happen), *safety* (i.e., nothing bad ever happens), and *cyclic* (i.e., a sequence of actions must perform cyclically) properties. Table 7.1 shows a number of basic properties defined in Modal μ-Calculus. More complex properties can be defined based on these properties. The proofs of these formulas are out of scope for this discussion and can be found in Stirling [2001]. However, further elaboration is not required to understand this work. The service composition certifier formalizes the client-defied composition polices by *mapping* them into the basic temporal properties.

7.4 Formalizing Business Components

The dynamic behaviour of a system refers to the interactions between the system and its environment. In this work, the dynamic behaviour of web services is modeled, based on their business components. In this section, a process algebraic model is proposed for business components including *business data*, *business action*, and *business process*.

7.4.1 Business Data

Business data represent data that are passed among different components of a system (e.g., account information, transactions, confirmations, and cancellations). In this context, "system" refers to the extended SOA architecture represented in Figure 3.3 (Chapter 3).

Business data x is modeled by a CCS message where there is a channel between every pair of its sender and receiver processes ($sender/receiver/x()$). Consequently, there are two basic actions: receive x (*sender/receiver/x?()*) and send x (*sender/receiver/x!()*) for each channel. The values transmitted through channels correspond to the actual contents of the messages. Empty values are used since we are not interested in the actual contents of the messages being exchanged.

The send and receive action from process $P1$ (as sender) to process $P2$ (as receiver) and for a composite business data $\vec{Y} = \{y_1, \cdots, y_n\}$, are defined as follows.

$$P1/P2/\vec{Y}!() = P1/P2/y_1!() \cdot \cdots \cdot P1/P2/y_n!()$$
$$P1/P2/\vec{Y}?() = P1/P2/y_1?() \mid\mid \cdots \mid\mid P1/P2/y_n?()$$

$P1$ sends y_i sequentially and in a custom order. This reduces the number of concurrent sub-processes. On the other hand, $P2$ receives y_i in parallel because the order is not important and the input data can be provided from different sources and at different times.

7.4.2 Business Action

A business action F is a function which returns one or more output business data $\epsilon\ Output_F = \{Y_1, \cdots, Y_m\}$ for each input business data $\epsilon\ Input_F = X_1 \times \cdots \times X_n$. For example, a business action representing a money transfer function takes details of two accounts (x_1 and x_2) and a transaction (x_3), as the input business data. Then it returns either the transaction confirmation (y_1) or cancellation (y_2), as the output business data. The following function represents business action F.

$$F : Input_F \longrightarrow 2^{Output_F}$$
$$F(x_1, \cdots, x_n) = (\vec{Y_1}\ or\ \cdots\ or\ \vec{Y_l}),\ \vec{Y_i}\ \epsilon\ 2^{Output_F}$$

The business action F is modeled by a CCS process where it receives required input messages from an orchestration process Θ and then it conditionally returns one or more output messages. Finally, we consider two signals, *start* and *end*, for each business action which will be used by the orchestration process. The following CCS process provides a formal model for the business action F.

$$[[F]] = F/\Theta/start!() \cdot (\Theta/F/x_1?() \mid\mid \cdots \mid\mid \Theta/F/x_n?()) \cdot \tau \cdot (F/\Theta/\vec{Y_1}!() + \cdots + F/\Theta/\vec{Y_m}!()) \cdot F/\Theta/end!()$$

Table 7.2. Mapping the primitive activities of a business process model in CCS.

$\Theta : Orchestrator$ $C : Process\ Client$ $F : Business\ Action$ $\eta : Local\ Memory$ $x, y : Business\ Data$ $X :< x_1, \cdots, x_n > \ \epsilon\ Input_F$ $Y : \{\vec{Y_1}, \cdots, \vec{Y_m}\}$ $\vec{Y_i}\ \epsilon\ 2^{Output_F}$	**Invoke** $(\Theta, F, \eta, X, Y) = F \mid\mid (F/\Theta/start?() \cdot$ $\Theta/\eta/x_1 Req!() \cdot \eta/\Theta/x_1?() \cdot \Theta/F/x_1!() \cdot \ \cdots \ \cdot \Theta/\eta/x_n Req!() \cdot \eta/\Theta/x_n?() \cdot$ $\Theta/F/x_n!() \cdot (F/\Theta/\vec{Y_1}?() \cdot \Theta/\eta/\vec{Y_1}!() + \ \cdots \ + F/\Theta/\vec{Y_l}?() \cdot \Theta/\eta/\vec{Y_m}!()))$ **Receive** $(\Theta, C, \eta, x) = C/\Theta/x?() \cdot \Theta/\eta/x!()$ **Reply** $(\Theta, C, \eta, x) = \Theta/\eta/xReq!() \cdot \eta/\Theta/x?() \cdot \Theta/C/x!()$ **Assign** $(\Theta, \eta, x, y) = \Theta/\eta/xReq!() \cdot \eta/\Theta/x?() \cdot \Theta/\eta/y!()$

7.4.3 Business Process

A business process P is a stepwise procedure made up of a series of activities, each of which includes applying a number of business actions to their corresponding business data. A business process is specified by three components: *process model*, *process knowledge*, and *process data*, where the process model defines the order of applying the process knowledge to the process data. Moreover, a business process is executed by a business process engine where it needs a local memory to store/retrieve business data. Consequently, this work formalizes a business process P by a collection of CCS components as follows:

$$[[P]] = [[(P_{Model}, P_{Knowledge}, P_{Data})]]_P = (\Theta, \Lambda, \Delta, \eta)$$

- Θ: is the CCS process (representing the orchestration process) corresponding to the business process model which includes primitive and structured activities of business data exchanges, as follows.
  - *Primitive activities*: include business action invocations, returning business data to the process client, receiving business data from the process client, and business data assignments.
  - *Structured activities*: include parallel, sequential, and/or conditional compositions of primitive activities.

Figure 7.3 represents business data transmissions as well as primitive and structured activities in a business process. Table 7.2 provides CCS processes of the primitive activities. The *Invoke* activity calls the local memory to retrieve the required input business data. Then it calls the business action and forwards the data received from memory to the business action. Finally, it receives the resulting output from the business action and passes it to the memory

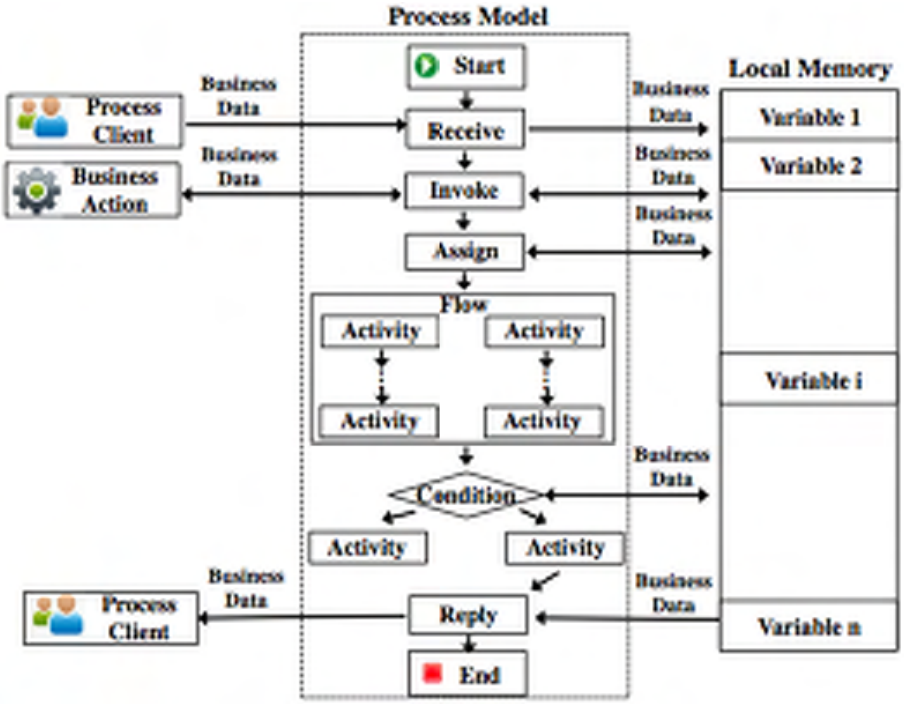


Figure 7.3. Business data transmissions in a business process. The local memory stores and then provides access to business data for different process activities.

to be stored. Similarly, the *Receive* and *Reply* activities use the local memory to store and retrieve, respectively, the target business data. Since the actual contents of the business data being exchanged are not needed, the *Assign (x,y)* activity just reads business data x (to check whether it has been written) and writes business data y in the local memory. The structured activities are simply mapped to their corresponding structures in CCS. In the following section, two samples of the process model are presented: the task model and the BPEL model. While the former invokes local business actions (provided by the service representative knowledge base), the latter invokes external business actions (provided by data and task services).

- Λ: denotes the set of CCS processes corresponding to the process knowledge invoked in the business process model such as local and external business actions.

- Δ: is the set of CCS messages corresponding to the business data used by the business process. Moreover, $\Delta = \Delta_{Input} \cup \Delta_{Output} \cup \Delta_{Internal}$, where Δ_{Input} represents the process input parameters provided by the process client; Δ_{Output} represents the process response data which will be returned to the process client; and $\Delta_{Internal}$ includes the business data which are generated and consumed internally by the business process.

If $\Lambda = \{F_1, \cdots, F_n\}$ represents the set of involved business actions, Δ can be obtained by the following equation.

$$\Delta = (\cup_{i=1}^{n} Input_{Fi}) \cup (\cup_{i=1}^{n} Output_{Fi})$$

Consequently, $\Delta_{Internal}$ can be identified explicitly based on Δ, Δ_{Input}, and Δ_{Output}.

- η: is a CCS process which simulates a local memory for the orchestration process to store/retrieve process variables. There is a process variable for each business data $x \epsilon \Delta$, and consequently η consists of a subprocess for each process variable. After storing each business datum, the orchestration process Θ can retrieve it by sending a request message $xReq$ to η.

$$\eta = \Theta/\eta/x_1?() \cdot X_1 \mid\mid \cdots \mid\mid \Theta/\eta/x_n?() \cdot X_n$$
$$X_i = \Theta/\eta/x_i Req?() \cdot \eta/\Theta/x_i!() \cdot X_i$$

7.5 Formalizing Business Services

This work models web services based on their business components for the following reasons.

- Each web service executes one or more business processes of its enterprise. While a data service applies its business components completely at the server side, a task service sends some part of its business components to the client side to be applied locally by the service representative.
- Business components can be easily understood by both service clients and service developers. There are several business process management systems which enable enterprises to model their services into business components and provide them through web services (e.g., *IBM WebSphere*).
- Service clients can describe their expectations of business services in terms of business components.

This section proposes a process algebraic model for each component of the extended SOA architecture presented in Chapter 3. The dynamic behaviour of a composite web service CWS is formalized as being represented by a BPEL model that includes both data services (DWS) and task services (TWS). In order to call the CWS, a service client sends required remote service parameters to the service orchestrator. The service orchestrator returns the generated remote service responses directly to the service client. However, the resulting task messages are sent to the service representative to process the local service parameters and generate local service responses for the service client.

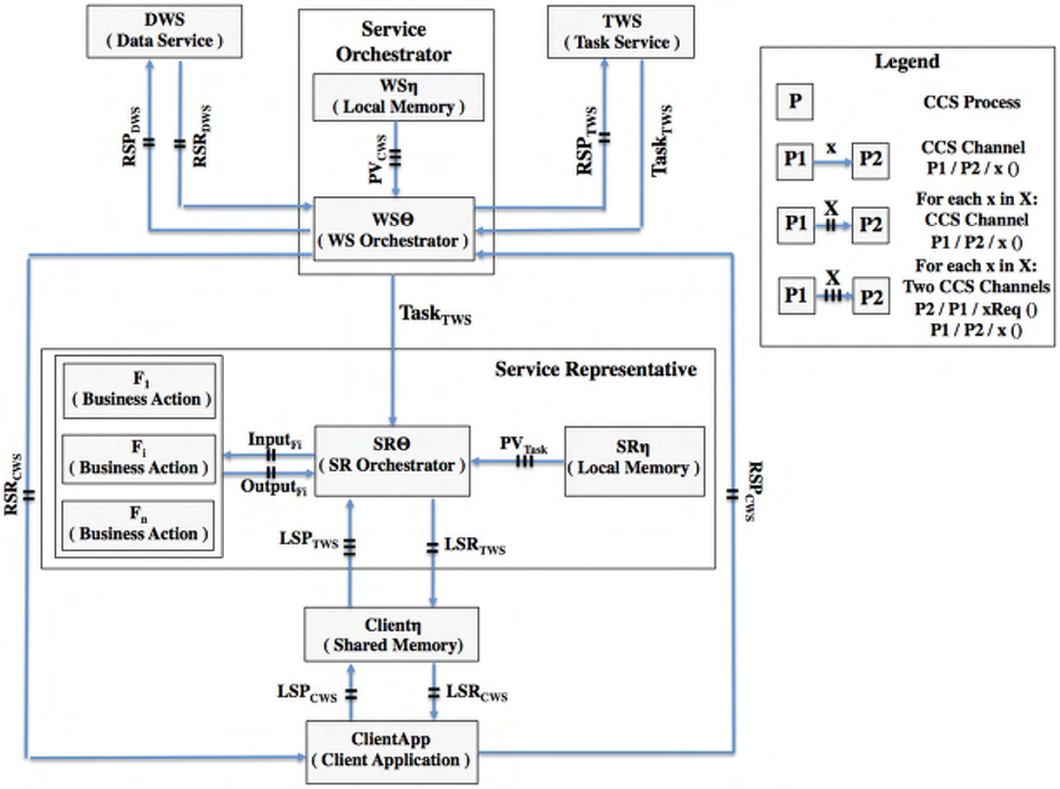


Figure 7.4. Corresponding CCS data, CCS channels, and CCS processes in the proposed extended SOA architecture shown in Figure 3.3.

7.5.1 CCS Messages and Processes

Each SOA component in the proposed architecture is modeled by one or more CCS processes, which will be defined later in this section. Business data are transferred over the SOA components which are categorized into the following types: Local Service Parameter (LSP); Remote Service Parameter (RSP); Local Service Response (LSR); Remote Service Response (RSR); Task Data (TD); Task Message ($Task$); Internal Process Data ($Internal$); and Process Variable (PV). Additionally, SOA components use Signals (S) to notify other SOA components of some system events. As described in Section 7.4, business data are modeled by CCS messages and there is a CCS channel between each pair of its sender and receiver processes. Figure 7.4 represents the corresponding CCS messages, CCS channels, and CCS processes in the proposed extended SOA architecture described in Chapter 3.

7.5.2 Service Client

The client application is modeled by a CCS process *ClientApp* which sends the remote service parameters $RSP_{CWS} =< x_1^r, \cdots, x_{n1}^r >$ and the local service parameters $LSP_{CWS} =< x_1^l, \cdots, x_{n2}^l >$ of the composite web service CWS to the service orchestrator and the local shared memory, respectively. Then, it waits to receive the remote service responses $RSR_{CWS} = \{y_1^r, \cdots, y_{n3}^r\}$ and the local service responses $LSR_{CWS} = \{y_1^l, \cdots, y_{n4}^l\}$ from the service orchestrator and the shared memory, respectively.

This model represents a typical client application where the client receives service responses after sending all the service parameters. However, the service client can define a process model to send and receive service parameters and responses, respectively, in a customized order. This client process model is formalized similarly to the BPEL model or task model discussed later in this section.

In addition to the client application, the service client includes a shared memory to communicate with the service representative. There is a corresponding CCS process for the shared memory, *Client*η, which includes one port for every local service parameter x_i^l as well as every local service response y_i^l. Each port can be accessed in three different ways: *initialization*, *read* , and *write*. Table 7.6 represents the process algebraic model of the *ClientApp* as well as the *Client*η processes. It should be noted here that different task services can communicate with each other through the shared memory. Consequently, while a local service parameter is written by the client application and read by the service representative, a local service response is written by the service representative and can be read by both the client application and the service representative.

7.5.3 Data Service

The dynamic behaviour of a data web service DWS is modeled by a function that receives remote service parameters RSP_{DWS} from the service orchestrator $WS\Theta$ and then it returns remote service responses RSR_{DWS}.

$$DWS : RSP_{DWS} \longrightarrow RSR_{DWS}$$

In other words, a data service is a business action which can be invoked remotely using message exchanges. Table 7.6 represents the process algebraic model of the data service where it notifies the service orchestrator using two signals: *start* and *end*.

7.5.4 Task Service

Similarly to a data service, the dynamic behaviour of a task web service TWS is modeled by a function where it receives remote service parameters RSP_{TWS} from the service orchestrator and

then it returns the generated task message $task_{TWS}$ to the service representative (SR). Then the service representative processes the local service parameters LSP_{TWS} and the received task data TD_{TWS} generates the local service responses LSR_{TWS} for the client.

$$TWS : RSP_{TWS} \longrightarrow Task_{TWS}$$
$$SR : Task_{TWS} \times LSP_{TWS} \times TD_{TWS} \longrightarrow LSR_{TWS}$$

Each task message *(Task Model, Task Knowledge, Task Data)* represents an incomplete business process *(Process Model, Process Knowledge, Process Data)* which is completed by the service representative using the local resources, as follows.

- $Process\ Model = Task\ Model$
- $Process\ Knowledge = Task\ Knowledge \cup SR\ Internal\ Knowledge$
- $Process\ Data = Task\ Data \cup LSP_{TWS} \cup LSR_{TWS} \cup Internal_{TWS}$

The resulting business process is formalized into CCS components $(\Theta, \Lambda, \Delta, \eta)$ as follows.

- Θ: the CCS process corresponding to the task model including parallel, sequential, and/or conditional compositions of internal business actions invocations. Any business process modeling technique which is understandable by the service representative and supports the primitive and structured activities discussed in Section 3 can be used as the task model. Table 7.3 presents CCS processes corresponding to the primitive activities of a task model where they are a customized version of CCS processes introduced in Table 7.2. The structured activities are simply mapped to the CCS constructs introduced in Section 7.2.

Table 7.3. Mapping the primitive activities of task models in CCS.

Invoke	$\mathbf{Invoke}\ (\mathbf{SR\Theta}, \mathbf{F}, \mathbf{SR}\eta, \mathbf{Input_F}, \mathbf{Output_F})$ $F : Business\ Action$
Receive	$\mathbf{Receive}\ (\mathbf{SR\Theta}, \mathbf{Client}\eta, \mathbf{SR}\eta, \mathbf{x})$ $x : Local\ Service\ Parameter$
Reply	$\mathbf{Reply}\ (\mathbf{SR\Theta}, \mathbf{Client}\eta, \mathbf{SR}\eta, \mathbf{x})$ $x : Local\ Service\ Response$
Assign	$\mathbf{Assign}(\mathbf{SR\Theta}, \mathbf{SR}\eta, \mathbf{x}, \mathbf{y})$ $x, y : Task\ Varaibles$

- Λ: the set of CCS processes corresponding to the business actions in the task knowledge segment of a task message and the SR internal knowledge base.
- Δ: the set of CCS messages corresponding to the business data used by the invoked business actions, where $\Delta_{Input} = TD_{TWS} \cup LSP_{TWS}$, and $\Delta_{Output} = LSR_{TWS}$. As discussed in Section 7.4, $\Delta_{Internal}$ can be identified explicitly.

Table 7.4. Mapping of BPEL Primitive Activities in CCS.

BPEL Code	CCS Model
< invoke partnerLink="pLink" portType = "Port" operation = "Operation" inputVariable = "X" outputVariable = "Y" />	*Data Service:* $\mathbf{Invoke}\ (\mathbf{WS\Theta}, \mathbf{DWS}, \mathbf{WS}\eta, \mathbf{RSP_{DWS}}, \mathbf{RSR_{DWS}})$ $DWS = "pLink/Port/Operation"$ $RSP_{DWS} = X =< x_1^r, \cdots, x_n^r >$ $RSR_{DWS} = Y = \{y_1^r, \cdots, y_m^r\}$ *Task Service:* $\mathbf{Invoke}\ (\mathbf{WS\Theta}, \mathbf{TWS}, \mathbf{WS}\eta, \mathbf{RSP_{TWS}}, \mathbf{RSR_{TWS}})$ $TWS = "pLink/Port/Operation"$ $RSP_{TWS} = X =< x_1^r, \cdots, x_n^r >$ $RSR_{TWS} = Y = task_{TWS}$
< receive partnerLink="pLink" portType = "Port" operation = "Operation" variable = "x" />	$\mathbf{Receive}\ (\mathbf{WS\Theta}, \mathbf{Partner}, \mathbf{WS}\eta, \mathbf{x})$ $Partner = "pLink/Port/Operation"$
< reply partnerLink="pLink" portType = "Port" operation = "Operation" variable = "x" />	$\mathbf{Reply}\ (\mathbf{WS\Theta}, \mathbf{Partner}, \mathbf{WS}\eta, \mathbf{x})$ $Partner = "pLink/Port/Operation"$
<assign name ="A"> <copy><from variable ="x" /> <to variable="y" /> </copy> </assign>	$\mathbf{Assign}(\mathbf{WS\Theta}, \mathbf{WS}\eta, \mathbf{x}, \mathbf{y})$
<throw faultName="faultname" faultVariable="x" / >	$\mathbf{WS\Theta}/\mathbf{WS\Theta}/\mathbf{x}!()$
<wait for = "T"/>	τ

- η: the CCS process corresponding to the SR local memory to store and retrieve task process variables. Task data are provided by the service provider to be used by the service representative. Therefore, their corresponding ports in the local memory do not need the initialization step.

Table 7.6 represents the process algebraic model of the task service and the service representative.

7.5.5 Service Orchestrator

The service orchestrator runs the composite web service CWS which is represented by a BPEL model. The BPEL model specifies the order of partner services invocations. As illustrated in Figure 7.4, the service orchestrator receives remote service parameters RSP_{CWS} sent by the service client. After invoking the partner services, it returns: (i) remote service responses RSR_{CWS} generated by the partner data services DWS_{CWS} to the service client, and ii) task messages

Table 7.5. Mapping of BPEL Structure Activities in CCS.

BPEL Code	CCS Model
<flow> <activity name="A_1"> ... </activity> ... <activity name="A_n"> ...</activity> </flow>	$[[\mathbf{A_1}]] \mid\mid \cdots \mid\mid [[\mathbf{A_n}]]$
<sequence> <activity name="A_1"> ... </activity> ... <activity name="A_n"> ...</activity> </sequence>	$\mathbf{sq_1}!() \mid\mid (\mathbf{sq_1})[[\mathbf{A_1}]](\mathbf{sq_2}) \mid\mid \cdots \mid\mid \mathbf{sq_n}?() \cdot [[\mathbf{A_n}]]$
<while condition > <activity name="A">... </activity> </while>	$[[\mathbf{while}]] = \tau \cdot \mathbf{0} + ([[\mathbf{A}]](\mathbf{loop}) \mid\mid \mathbf{loop}?() \cdot [[\mathbf{while}]]))$
<switch> <case condition> <activity$_1$/ > </case> ... <case condition> <activity$_n - 1$/ > </case> <otherwise> <activity$_n$ > </otherwise> </switch>	$[[\mathbf{activity_1}]] + \cdots + [[\mathbf{activity_{n-1}}]] + [[\mathbf{activity_n}]]$
<pick> <onMessage partnerLink="pLink" portType= "Port" operation="Operation" variable="x"> <activity name="A">...</activity> </onMessage> <onAlarm for="y"> <activity name="B"> ...</activity> </onAlarm> </pick>	$\mathbf{Partner/WS\Theta/x}?() \cdot [[\mathbf{A}]] + \mathbf{Partner/WS\Theta/y}?() \cdot [[\mathbf{B}]]$ Partner = pLink/Port/Operation

$Task_{CWS}$ generated by the partner task services TWS_{CWS} to the service representative.

The composite web service is modeled by a business process, as follows.

- *Process Model* = *BPEL Model*
- *Process Knowledge* = $DWS_{CWS} \cup TWS_{CWS}$
- *Process Data* = $RSP_{CWS} \cup RSR_{CWS} \cup Task_{CWS} \cup Internal_{CWS}$

The BPEL business process differs from the task business process as the former invokes external services while the latter invokes internal business actions as their required process knowledge. CCS components corresponding to the BPEL business process are defined as follows.

- Θ: the CCS process corresponding to the BPEL model, obtained as follows.

  - *Mapping of Primitive Activities.* Basic BPEL activities are transformed into their CCS counterparts according to the transformations shown in Table 7.4. Faults are considered as events and therefore they are modeled by channels named after their corresponding fault name. To raise a fault, we perform an output to the corresponding channel. The translation to CCS abstracts away some details irrelevant to the orchestration itself, like timing and conditions.

  - *Mapping of Structured Activities.* The translation scheme of the CCS formula for structured activities is given in Table 7.5. In these patterns, activity can be any type of a primitive or structured BPEL activity. The *flow*, *sequence*, *while*, *switch*, and *pick* constructs can be easily translated by considering the parallel, sequence, loop, and conditional CCS constructs.

- Λ: the set of CCS processes corresponding to the partner services, where each partner service is identified by *partnerLink/Port/Operation*.

- Δ: the set of CCS data corresponding to the business data used by the BPEL process where $\Delta_{Input} = RSP_{CWS}$, and $\Delta_{Output} = RSR_{CWS} \cup Task_{CWS}$.

- η: the CCS process corresponding to the service orchestrator local memory which stores and retrieves the BPEL variables.

7.6 Service Composition Policy

The Personal Web aims to enable service end-users to compose services, statically or dynamically. As a result, different clients may have different expectations from a service composition. In addition to interacting with the service provider, the service client needs to interact with the service representative in order to invoke a task service. Consequently, the dynamic behaviour of services becomes more complex and needs to be verified.

While service descriptions (e.g., UDDI) include the functional and non-functional properties of web services, the *service composition policy (SCP)* is introduced to represent a client's expectations from a web service composition. In other words, each collaborating service in a service composition is assumed to behave exactly based on its description, and the service client needs to check what is obtained from the composition of these services, as represented in a BPEL model.

In Section 7.5, a formal model of the SOA components was provided based on the business components, including business processes, actions, and data. While the enterprise business processes and actions are hidden from the service client, the service client interacts with business services,

Table 7.6. Process algebraic model of the extended SOA components where the client application invokes a composite web service (CWS), including a data service (DWS) and a task service (TWS). In this table, $RSP_{CWS} =< x_1^r, \cdots, x_{n1}^r >$**,** $LSP_{CWS} =< x_1^l, \cdots, x_{n2}^l >$**,** $RSR_{CWS} = \{y_1^r, \cdots, y_{n3}^r\}$**, and** $LSR_{CWS} = \{y_1^l, \cdots, y_{n4}^l\}$**. Finally,** $\vec{Y_i^r} \epsilon 2^{RSR_{CWS}}$**.**

Client Application	$ClientApp = ClientApp/WS\Theta/x_1^r!() \cdot \cdots \cdot ClientApp/WS\Theta/x_{n1}^r!() \cdot$ $ClientApp/Client\eta/x_1^l!() \cdot \cdots \cdot ClientApp/Client\eta/x_{n2}^l!() \cdot$ $(WS\Theta/ClientApp/y_1^r?() \mid\mid \cdots \mid\mid WS\Theta/ClientApp/y_{n3}^r?() \mid\mid$ $ClientApp/Client\eta/y_1^l Req!() \cdot Client\eta/ClientApp/y_1^l?() \mid\mid \cdots \mid\mid$ $ClientApp/Client\eta/y_{n4}^l Req!() \cdot Client\eta/ClientApp/y_{n4}^l?()) \cdot$ $ClientApp/WS\Theta/ack!()$
Shared Memory	$Client\eta = X_1^l \mid\mid \cdots \mid\mid X_{n2}^l \mid\mid Y_1^l \mid\mid \cdots \mid\mid Y_{n4}^l$ $X_i^l = ClientApp/Client\eta/x_i^l?() \cdot (ReadX_i^l \mid\mid WriteX_i^l)$ $ReadX_i^l = SR\Theta/Client\eta/x_i^l Req?() \cdot Client\eta/SR\Theta/x_i^l!() \cdot ReadX_i^l$ $WriteX_i^l = ClientApp/Client\eta/x_i^l?() \cdot WriteX_i^l$ $Y_i^l = SR\Theta/Client\eta/y_i^l?() \cdot (ReadY_i^l \mid\mid WriteY_i^l)$ $ReadY_i^l = (ClientApp/Client\eta/y_i^l Req?() \cdot Client\eta/ClientApp/y_i^l!() +$ $SR\Theta/Client\eta/y_i^l Req?() \cdot Client\eta/SR\Theta/y_i^l!()) \cdot ReadY_i^l$ $WriteY_i^l = SR\Theta/Client\eta/y_i^l?() \cdot WriteY_i^l$
Data Service	$DWS = DWS/WS\Theta/start!() \cdot (WS\Theta/DWS/x_1^r?() \mid\mid \cdots \mid\mid WS\Theta/DWS/x_{n1}^r?()) \cdot$ $\tau \cdot (DWS/WS\Theta/\vec{Y_1^r}!() + \cdots + DWS/WS\Theta/\vec{Y_m^r}!()) \cdot DWS/WS\Theta/end!()$
Task Service	$TWS = TWS/WS\Theta/start!() \cdot (WS\Theta/TWS/x_1^r?() \mid\mid \cdots \mid\mid WS\Theta/TWS/x_{n1}^r?()) \cdot$ $\tau \cdot TWS/WS\Theta/task_{TWS}!())$
Service Representative	$SR = WS\Theta/SR\Theta/task_{TWS1}?() \cdot Task_{TWS1} \mid\mid \cdots \mid\mid WS\Theta/SR\Theta/task_{TWSn}?() \cdot$ $Task_{TWSn}$ $Task_{TWSi} = [[(Task\ Model_{TWSi},$ $Task\ Knowledge_{TWSi} \cup SR\ Knowledge,$ $Task\ Data_{TWSi} \cup LSP_{TWSi} \cup LSR_{TWSi} \cup Internal_{TWSi})]]_P$
Service Orchestrator	$WS\Theta_{CWS} = [[(BPEL_{CWS},$ $DWS_{CWS} \cup TWS_{CWS},$ $RSP_{CWS} \cup RSR_{CWS} \cup Task_{CWS} \cup Internal_{CWS})]]_P$

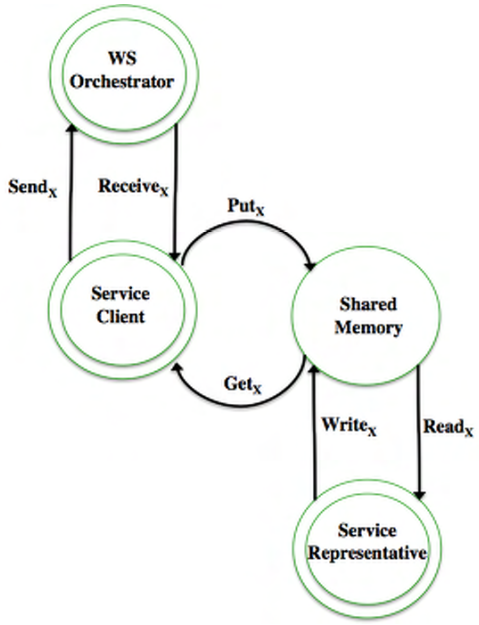


Figure 7.5. Business data life cycle from the client's point of view. Business data can originate from the client (local service parameter and remote service parameter), the WS orchestrator (remote service response), or the service representative (local service response).

using business data. The service client provides some business data as service parameters and then it asks for some business data as the service responses. Consequently, a client-originated service composition policy must be defined based on the business data involved in the service composition. As a result, a non-technical service client can define a SCP with less effort.

Processes of the extended SOA model generate a labelled transition system. Therefore, modal μ-calculus propositions can be defined to check dynamic behaviours of the entire system. In this work, the SCP is defined as a list of *policy conditions* where each policy condition defines a relationship among the involved business data. A policy condition is identified by $Name\ (param_1, \cdots, param_n)$ where each parameter is a business data event selected from the business data life cycle shown in Figure 7.5. Then, each policy condition is formalized by a parameterized modal $\mu - calculus$, where the corresponding CCS actions to the business data events are shown in Table 7.7. Consequently, while the service client defines its expectations from the service composition based on business data relationships, the proposed service composition certifier formalizes and verifies it using modal μ-calculus.

Several useful policy conditions can be defined and included in the service composition policy. In this section, a few instances of the policy conditions are defined, where their corresponding μ-calculus propositions are provided in Table 7.8.

Table 7.7. CCS actions corresponding to the business data events in Figure 7.5.

Event	CCS Action
Put_x	$ClientApp/Client\eta/x!()$
Get_x	$Client\eta/ClientApp/x?()$
$Send_x$	$ClientApp/WS\Theta/x!()$
$Receive_x$	$WS\Theta/ClientApp/x?()$
$Write_x$	$SR\Theta/Client\eta/x!()$
$Read_x$	$Client\eta/SR\Theta/x?()$

1. **Required** (x): is satisfied if the specified business event x is observed during the composite service execution. Similarly, the negated version of this condition policy represents an unwanted business data event (e.g., error messages). However, there is no need for checking a particular property in the case of optional business data events. The corresponding μ-calculus property to this condition uses the important liveness property that the business data event x must be eventually observed.

2. **Order** $(x_1, \cdots, x_n)$: is satisfied if the input parameters (events) are observed in the specified order by the parameter positions, during the composite service execution. This policy represents a complete order condition where all the input parameters must be observed in each sequence. The partial order condition can be defined similarly.

3. **Dependent** (x, y): is satisfied if there is one instance of the business data event x for every instance of the business data event y during the composite service execution. The crucial safety property of the corresponding $\mu - calculus$ property is that no two business data events x occur together in any {x,y} paths.

4. **Transaction Rollback** $(x_1, \cdots, x_n, y_1, \cdots, y_n)$: is satisfied if the events identified by x_i are observed completely in the defined order by their positions or if there are one or more missing events, the corresponding complementary events identified by y_i are observed to cancel their effects. For example, if the input events include *a, b, c, d* and the input co-events include *a', b', c', d'* , the acceptable execution traces are: (i) *a,b,c,d*; (ii) *a,b,c,c',b',a'*; (iii) *a,b,b',a'*; and (iv) *a,a'*.

The above policy conditions are verified throughout composite service execution. However, the service client can define an interval for each policy condition to bound it between two service invocations. Then, the service composition certifier uses the signals generated by the collaborating services to apply the defined interval. The CCS actions corresponding to the data and task service signals are: $DWS/WS\Theta/started!()$ and $TWS/WS\Theta/started!()$, respectively.

Table 7.8. Some instances of policy conditions.

Required (x)
$\mu Z.Live \wedge [[-\{x\ ,\ \epsilon\}]]Z$
Order $(x_1, \cdots, x_n)$
$\nu Z.\ [x_1]\ P_1 \wedge [-]Z$ $P_i = \mu Z. < - > P_{i+1} \vee [-x_i]\ Z\ \ if\ i \neq n$ $P_n = \mu Z. < - > True \vee [-x_n]\ Z$
Dependent (x, y)
$\nu Z.\ [x]P \wedge [-]Z$ $P = \mu Y.\ [y]\ True\ \vee ([x]\ False\ \wedge\ [-]\ Z)$
Transaction Rollback $(x_1, \cdots, x_n, y_1, \cdots, y_n)$
$Order(x_1, \cdots, x_n) \vee P_1$ $P_i = Order(x_1, \cdots, x_{n-i}, y_{n-i}, \cdots, y_1) \vee P_{i+1}\ \ if\ i \neq n$ $P_n = False$

Table 7.9 represents two bounded policy conditions. An interval $[I_1, I_2]$ is used to define a boundary. Moreover, ∞ represents no boundary where $(\infty, y]$ expresses that the condition must be checked as soon as the composite service is called. Similarly, $[I_1, \infty)$ expresses that the verification ends when the composite service terminates.

7.7 Service Composition Certifier

The service composition certifier certifies a service composition model in two steps. First, it verifies that the composition model includes compatible web services and then it validates that the composite service responses match with the client's expectations. Different components of the SCC, displayed in Figure 7.6, are described in this section. Moreover, in order to certify composite services, each service provider is required to publish the service process model (discussed in Section 7.5) into an enhanced UDDI registry (Figure 7.7).

Table 7.9. Two bounded policy conditions.

Bounded Dependency (x, y) $Interval = [DWS, \infty)$
$Order(DWS/WS\Theta/started!(), x, y) \wedge P$ $P = \nu Z.[x][x]False \wedge [x, y]Z$
Bounded Order $(x_1, \cdots, x_n)$ $Interval = [DWS, TWS]$
$Order(DWS/WS\Theta/started!(), x_1, \cdots, x_n,$ $TWS/WS\Theta/started!())$

Model Checker. Given a formal model of a system and a property, the model checker tests automatically whether this model meets the property. Several model checkers have been developed to help verify concurrent asynchronous systems. We use the *Edinburgh Concurrency Workbench* (Moller and Stevens [1997]), which is based on a tableau decision procedure to verify CCS processes properties defined in modal μ-calculus.

System Modeller. This component receives a BPEL model of a composite service from the service client. Then it invokes the process model of each of the partner services from the service registry. Using the proposed approach in Section 7.5, the System Modeller provides a formal model of the entire system including the CCS processes corresponding to the service client, the client shared memory, the service representative, the service orchestrator, and the partner services. Finally, the generated model is given to the model checker.

Compatibility Checker. This component includes pre-defined properties to verify whether collaborating services in a composite model are compatible. These are general properties and are applied to all submitted composition models, such as:

- *Deadlock Free*: satisfied if there is no deadlock in the system during composite service execution. It also implies that the service client receives all the expected remote and local service responses from the composite service. Otherwise, the service client waits to receive them causing deadlock.
- *Convergence*: represents the inability to perform silent actions forever by partner services or service representatives.

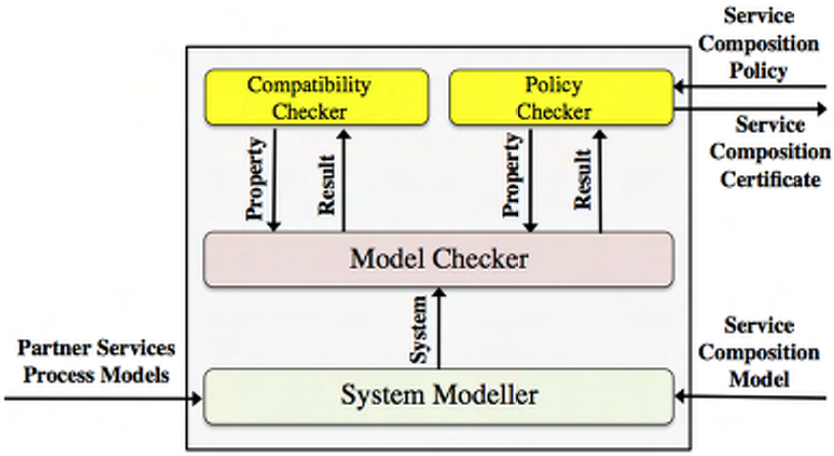


Figure 7.6. Service Composition Certifier architecture.

- *Privacy*: violated if the service representative sends any of the local service parameters to the partner services.

The compatibility checker includes the corresponding μ-calculus propositions with each of these conditions (Table 7.10). Then this component calls the model checker to verify these properties in the given system.

Table 7.10. Three general temporal properties used by the compatibility checker.

Deadlock Free
$\mu\ Z.(Live \wedge [[-\{ClientApp/WS\Theta/ack!()\ ,\ \epsilon\}]]Z)$
Convergence
$\neg(\nu Z.[\tau]Z)$
Privacy
For each x in LSP_{CWS} $\neg\ (\mu\ Z.Live \wedge [[-\{Send_x\ ,\ \epsilon\}]]Z)$

Policy Checker. This component checks whether the composite service meets the client's expectation, submitted by the client's service composition policy. First, it formalizes the client's policy as described in Section 7.6 and then it calls the model checker to verify each policy condition. If the compatibility checker and the policy checker approve compatibility and satisfaction, respectively, of the service composition model, the SCC generates a certificate for the composition model according to the given composition policy.

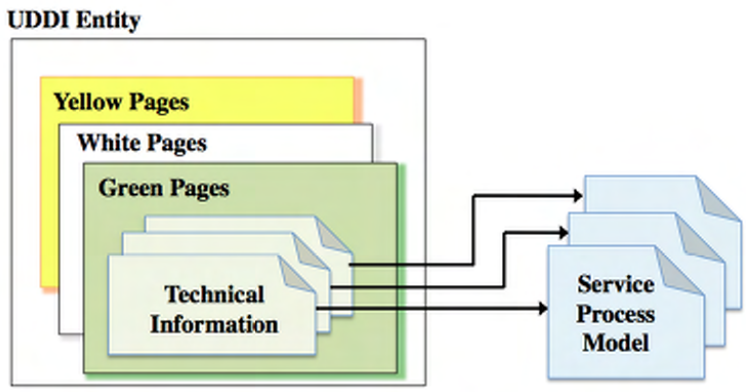


Figure 7.7. Enhanced UDDI registry including service process models.

7.8 Case Study

To describe how the proposed SCC certifies a composition model of collaborating services, this section presents a case study in the financial domain. In this case study, the service client plans to compose web services in order to receive personalized stock advice. Moreover, the composite service can enable the service client to place stock buy and stock sell orders. Finally, the client's portfolio and cash information must be updated by the composite service, based on the client's financial activities.

To provide customized advice, a financial adviser data service asks for personal information from its clients (e.g., client's portfolio or cash information). However, the service client looks for a secure service composition to allow keeping personal information local. Therefore, the service client calls task services whenever personal information needs to be processed.

7.8.1 Partner Services

The service client composes the following data and task services in a BPEL model.

1. *Stock Adviser WS* (task service): receives the client's general preferences and returns some general financial advice that will be customized for the client by the service representative using the client's holding and cash information. As a result, the service representative generates customized advice (*sell advice* and *buy advice*) that are written to the communication channel. This service was discussed in Chapter 2. The task components were displayed in Figure 2.3. This service is described by the following function where the corresponding business data are: *preferences (RSP), generalBuyAdvice (TD), generalSellAdvice (TD), holdings (LSP), cash (LSP), buyAdvice (LSR), sellAdvice (LSR)*, and $task_{StockAdviser}$ *(Task)*.

$$\text{StockAdviser } (preferences) = task_{StockAdviser}$$
$$\text{SR } (task_{StockAdviser}, cash, holdings, generalBuyAdvice, generalSellAdvice) = (buyAdvice, sellAdvice)$$

2. *Portfolio Manager WS* (task service): returns a task message to employ the service representative to update the client's holding and cash information every time the client performs a buy or sell stock action. The task model of the portfolio manager web service is displayed in Table 7.12. This service is described by the following function where the corresponding business data are: *holdings (LSP), cash (LSP), deposit (LSP), withdraw (LSP), buyOrderConfirmation (LSP), sellOrderConfirmation (LSP), uHolding (LSR), uCash (LSR)*, and $task_{PortfolioManager}$ *(Task).*

$$\text{PortfolioManager } () = task_{PortfolioManager}$$
$$\text{SR } (task_{PortfolioManager}, cash, holdings, deposit, withdraw\ buyOrderConfirmation, sellOrderConfirmation) = (uHolding, uCash)$$

3. *Buy Stock WS* (data service): represents a trading system that receives *buy orders* and returns *buy order confirmations* or *cancellations.* This service requires the service client to submit the required finances (*withdrawals*) with the buy order. In the case of a buy order cancellation, the service returns the submitted money (*deposit*) to the service client. The following function represents this service where the corresponding business data are: *buyOrder (RSP), withdraw (RSP), deposit (RSR), buyOrderConfirmation (RSR)*, and *buyOrderCancellation (RSR).*

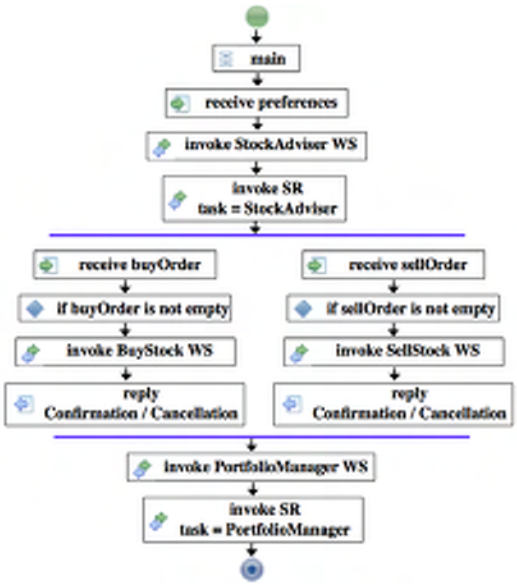
$$\text{BuyStock } (buyOrder, withdraw) = (buyOrderConfirmation \quad or \quad (buyOrderCancellation, deposit))$$

4. *Sell Stock WS* (data service): represents a trading system that receives *sell orders* and return *sell order confirmations* or *cancellations.* Moreover, the sell stock web service returns the money obtained from selling the stock to the client (*deposit*). The following function represents this service where the corresponding business data are: *sellOrder (RSP), sellOrderConfirmation (RSR), sellOrderCancellation (RSR)*, and *deposit (RSR).*

$$\text{SellStock } (sellOrder) = ((sellOrderConfirmation, deposit) \quad or \quad sellOrderCancellation)$$

Table 7.11. BPEL model representing the client-defined composite service in the case study and the corresponding CCS model.

BPEL MODEL



CCS MODEL

$$\mathbf{WS\Theta} = sq1!() \mid\mid (sq1)Seq1(sq2) \mid\mid (sq2)Seq2(sq3) \mid\mid sq3?() \cdot Seq3$$

$$\begin{aligned}\mathbf{Seq1} = {} & ClientApp/WS\Theta/preferences?() \cdot WS\Theta/WS\eta/preferences!() \cdot (StockAdviserWS \mid\mid \\ & StockAdviserWS/WS\Theta/start?() \cdot WS\Theta/WS\eta/preferencesReq!() \cdot WS\eta/WS\Theta/preferences?() \cdot \\ & WS\Theta/StockAdviserWS/preferences!() \cdot StockAdviserWS/WS\Theta/task_{StockAdviser}?() \cdot \\ & WS\Theta/SR\Theta/task_{StockAdviser}!() \cdot SR\Theta/WS\Theta/stockAdviser_{completed}?())\end{aligned}$$

$$\mathbf{Seq2} = Flow1 \mid\mid Flow2$$

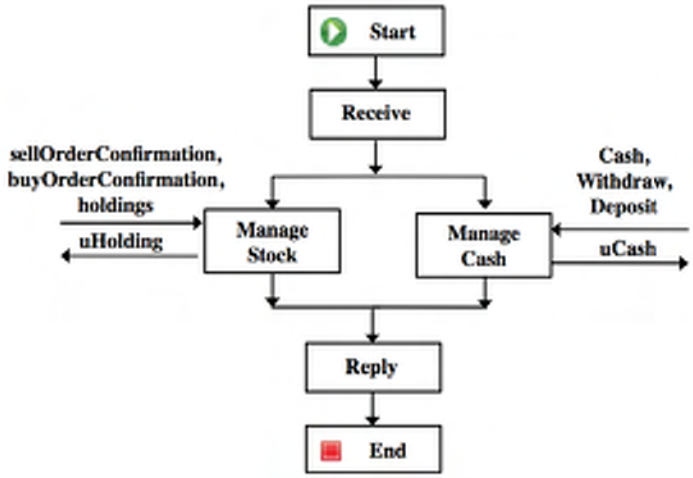
$$\begin{aligned}\mathbf{Flow1} = {} & ClientApp/WS\Theta/buyOrder?() \cdot ClientApp/WS\Theta/withdraw?() \cdot WS\Theta/WS\eta/buyOrder!() \cdot \\ & WS\Theta/WS\eta/withdraw!() \cdot (0 + BuyStockWS \mid\mid (BuyStockWS/WS\Theta/start?() \cdot \\ & WS\Theta/WS\eta/buyOderReq!() \cdot WS\eta/WS\Theta/buyOrder?() \cdot WS\Theta/BuyStockWS/buyOder!() \cdot \\ & WS\Theta/WS\eta/withdrawReq!() \cdot WS\eta/WS\Theta/withdraw?() \cdot WS\Theta/BuyStockWS/withdraw!() \cdot \\ & (BuyStockWS/WS\Theta/buyConfirmation?() \cdot WS\Theta/WS\eta/buyConfirmation!() + \\ & BuyStockWS/WS\Theta/buyCancellation?() \cdot WS\Theta/WS\eta/buyCancellation!() \cdot \\ & BuyStockWS/WS\Theta/deposit?() \cdot WS\Theta/WS\eta/deposit!()) \cdot (WS\Theta/WS\eta/buyConfirmationReq?() \cdot \\ & WS\eta/WS\Theta/buyConfirmation!() \cdot WS\Theta/ClientApp/buyConfirmation!() + \\ & WS\Theta/WS\eta/buyCancellationReq?() \cdot WS\eta/WS\Theta/buyCancellation!() \cdot \\ & WS\Theta/ClientApp/buyCancellation!() \cdot WS\Theta/WS\eta/depositReq?() \cdot \\ & WS\eta/WS\Theta/deposit!() \cdot WS\Theta/ClientApp/deposit!()))\end{aligned}$$

$$\mathbf{Flow2} = \cdots$$

$$\begin{aligned}\mathbf{Seq3} = {} & PortfolioManagerWS \mid\mid (PortfolioManagerWS/WS\Theta/start?() \cdot \\ & PortfolioManagerWS/WS\Theta/task_{PortfolioManager}?() \cdot \\ & WS\Theta/SR\Theta/task_{PortfolioManager}!() \cdot SR\Theta/WS\Theta/portfolioManager_{completed}?())\end{aligned}$$

Table 7.12. Portfolio Manager task service: task model and its corresponding CCS model.

TASK MODEL



CCS MODEL

$$\mathbf{PortfolioManagerWS} = PortfolioManagerWS/WS\Theta/start!() \cdot \tau \cdot PortfolioManagerWS/WS\Theta/task_{PortfolioManager!()}$$

$$\mathbf{Task_{PortfolioManager}} = sq_1!() \mid\mid (sq_1)\ Seq1\ (sq2) \mid\mid (sq_2)\ Seq2\ (sq_3) \mid\mid sq_3?() \cdot Seq3$$

$$\mathbf{Seq1} = SR\Theta/Client\eta/cashReq!() \cdot Client\eta/SR\Theta/cash?() \cdot SR\Theta/SR\eta/cash!() \cdot \cdots \cdot SR\Theta/Client\eta/holdingsReq!() \cdot Client\eta/SR\Theta/holdings?() \cdot SR\Theta/SR\eta/holdings!()$$

$$\mathbf{Seq2} = Invoke_{ManageCash} \mid\mid Invoke_{ManageStock}$$

$$\begin{aligned}\mathbf{Invoke_{ManageCash}} = {} & ManageCash \mid\mid (MangeCash/SR\Theta/start?() \cdot \\ & SR\Theta/SR\eta/withdrawReq!() \cdot SR\eta/SR\Theta/withdraw?() \cdot \\ & SR\Theta/SR\eta/depositReq!() \cdot SR\eta/SR\Theta/deposit?() \cdot \\ & SR\Theta/SR\eta/cashReq!() \cdot SR\eta/SR\Theta/cash?() \cdot \\ & SR\Theta/ManageCash/withdraw!() \cdot SR\Theta/ManageCash/deposit!() \cdot \\ & SR\Theta/ManageCash/cash!() \cdot ManageCash/SR\Theta/uCash?() \cdot \\ & SR\Theta/SR\eta/uCash!() \cdot MangeCash/SR\Theta/end?())\end{aligned}$$

$$\mathbf{Invoke_{ManageStock}} = \cdots$$

$$\begin{aligned}\mathbf{ManageCash} = {} & ManageCash/SR\Theta/start!() \cdot (SR\Theta/ManageCash/deposit?() \mid\mid \\ & SR\Theta/ManageCash/withdraw?() \mid\mid SR\Theta/ManageCash/cash?()) \cdot \tau \cdot \\ & ManageCash/SR\Theta/uCash!() \cdot ManageCash/SR\Theta/end!()\end{aligned}$$

$$\mathbf{ManageStock} = \cdots$$

$$\mathbf{Seq3} = SR\Theta/SR\eta/uCashReq!() \cdot SR\eta/SR\Theta/uCash?() \cdot SR\Theta/Client\eta/uCash!() \cdot SR\Theta/WS\Theta/portfolioManager_{completed}!()$$

7.8.2 Service Composition Model

The service client designs a BPEL model (represented in Table 7.11) to compose the partner data and task services. Based on this model, the service client sends its *preferences* to the composite service. Then the service client puts the *holdings* and *cash* into the communication channel.

The composite service first invokes the *StockAdviserWS* by sending its *preferences* and then forwards the resulting task message to the service representative. The service representative uses this task to customize the received *generalBuyAdvice* and *generalSellAdvice* (task data) based on the client information. Then the client considers the customized *buyAdvice* and *sellAdvice* to generate *buyOrder* and *sellOrder* which are sent to the *BuyStock* and *SellStock* web services. These services process the client orders and return the *buy/sellOrderConfirmation* or *Cancellation* to the client. Finally, the composite service invokes the *PortfolioManager* task service which returns a task message for the service representative to modify the client holding and cash information, based on the confirmed transactions.

7.8.3 Service Composition Policy

Table 7.13 represents the client's expectations from the composite service, as follows.

1. Providing *sell stock advice* and *buy stock advice* are mandatory, while providing *currency exchange advice* is optional.

2. The service client expects to complete the entire stock selling process prior to starting the stock buying process. This is required because the stock adviser web service takes into account the client's total cash for generating buy stock advice.

3. The client's *cash* must be updated for every *deposit* and *withdrawal* transaction.

4. The client's *holdings* must be updated for every received *buyOrderConfirmation* or *sellOrder-Confirmation*.

5. A rollback transaction is required for a buy stock order, where the client's money must be deposited back to the client's account if the buy order is cancelled for any reason.

7.8.4 Service Composition Certifier

The service client submits both the BPEL model and the service composition policy to the SCC. The SCC invokes the corresponding CCS-model for each partner service from the service registry. The CCS model of the portfolio manager task service is displayed in Table 7.12.

Table 7.13. Client-defined Service Composition Policy.

Policy Condition #1
Required ($Get_{sellAdvice}$) AND Required ($Get_{buyAdvice}$) AND Optional ($Get_{exAdvice}$)
Policy Condition #2
Order ($Receive_{sellOrderConfirmation}$, $Get_{buyAdvice}$) AND Order ($Receive_{sellOrderCancelation}$, $Get_{buyAdvice}$)
Policy Condition #3
Dependent ($Receive_{deposit}$, $Write_{uCash}$) AND Dependent ($Receive_{withdraw}$, $Write_{uCash}$)
Policy Condition #4
Dependent ($Receive_{buyOrderConfirmation}$, $Write_{uHolding}$) AND Dependent ($Receive_{sellOrderConfirmation}$, $Write_{uHolding}$)
Policy Condition #5
Transaction ($Send_{buyOrder}$, $Send_{withdraw}$, $Receive_{buyOrderConfirmation}$, 0, $Receive_{deposit}$, $Receive_{buyOrderCancelation}$)

The SCC model checker provides a formal model of the entire system by providing the CCS processes of the client SOA components and the BPEL model (Table 7.11). The SCC compatibility checker calls the model checker with its predefined properties. In the first phase, the *Edinburgh Concurrency Workbench* verified the three conditions defined in Table 7.10. Consequently, the collaborating services are compatible with the composition model. In the second phase, the SCC policy checker translates the submitted service composition policy into the modal $\mu - calculus$ propositions (Table 7.14) and then it calls the model checker to verify these properties. The *Edinburgh Concurrency Workbench* verified all the policy conditions except the second policy condition. The reason is that the *Stock Adviser* web service generates stock buy and sell advice at the same time. Consequently, the SCC can not certify the submitted composition model for this specific client's policy. However, the service client can change the service composition model in order to satisfy the second policy condition. This includes changing the partner services and their order of occurrence, as indicated in the service composition model presented in Figure 7.8.

7.9 Related Work

There is an ongoing debate on the best foundation for Web services: Petri nets or process algebra (Bordeaux and SalaN [2005] and Aalst [2003]). In this section, we review significant prior work in the field of web service modeling and certification.

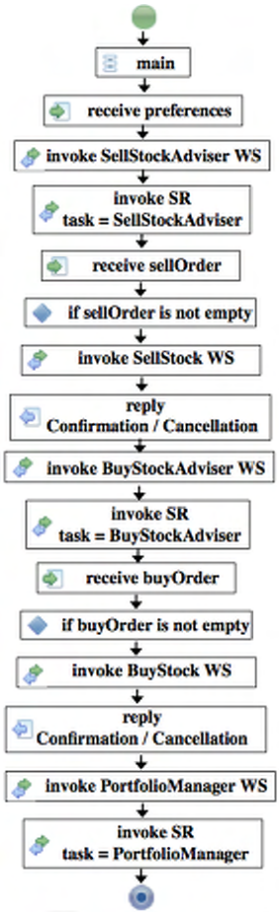


Figure 7.8. Alternative composition of the partner services in the Case Study certified by the SCC according to the client's policy, as defined in Table 7.13 .

7.9.1 Web Service Modeling by Petri Nets

Petri nets (Murata [1989]) is a formal tool for study and modeling of systems and processes. This formal modeling technique focuses on interactions and dynamic relationships among systems or processes elements. Petri Nets consist of three basic elements: places, transitions, and tokens. Places and transitions represent states and state changes, respectively. Tokens reside in places and are used to define the execution of a Petri net. Finally, a distribution of tokens over the places of a net is called a marking. A Petri net is formally defined by a four-tuple $C = (P, T, I, O)$, where:

- P is a finite set of places $P = \{p_1, \cdots, p_n\}$.
- T is a finite set of transitions $T = \{t_1, \cdots, t_m\}$. The set of places and the set of transitions are disjoint.

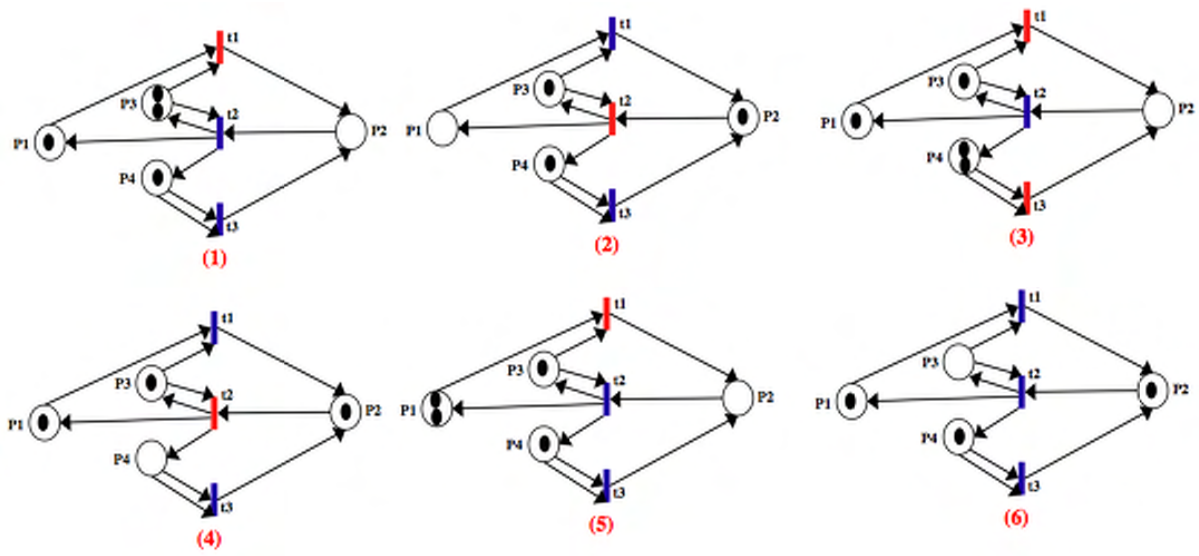


Figure 7.9. A Petri net execution. Circles are places and bars are transitions (based on an example discussed by Zhovtobryukh [2007]).

- I is the input function, a mapping from transitions to subsets of places.
- O is the output function, a mapping from transitions to subsets of places.

$$I, O : T \longrightarrow 2^P$$

A Petri net is executed by firing transitions. A transition must be enabled in order to be fired. A transition is enabled whenever each of its input places contain as many tokens as the number of arcs connecting this place to the transition. On firing the transition, tokens are removed from all its input places and deposited into all its output places. As a result, transition firings change Petri net states. Petri net execution persists until a deadlock state is reached, i.e., no transitions can be any more fired. An example of a Petri net is given in Figure 7.9.

The Petri net formalism has been used to model simple and composite web services, as follows.

1. Web services are modeled as processes exhibiting concurrency and non-determinism. A state of a web service is a unique set of attribute values and a web service operation is an action which switches the service from one state to another. A service always has an initial state and a final state at which it terminates. Service execution is a sequence of service operations which leads from the services initial state to its final state.

2. A simple web service is modeled by a Petri net where places model states of the service and transitions model service operations.

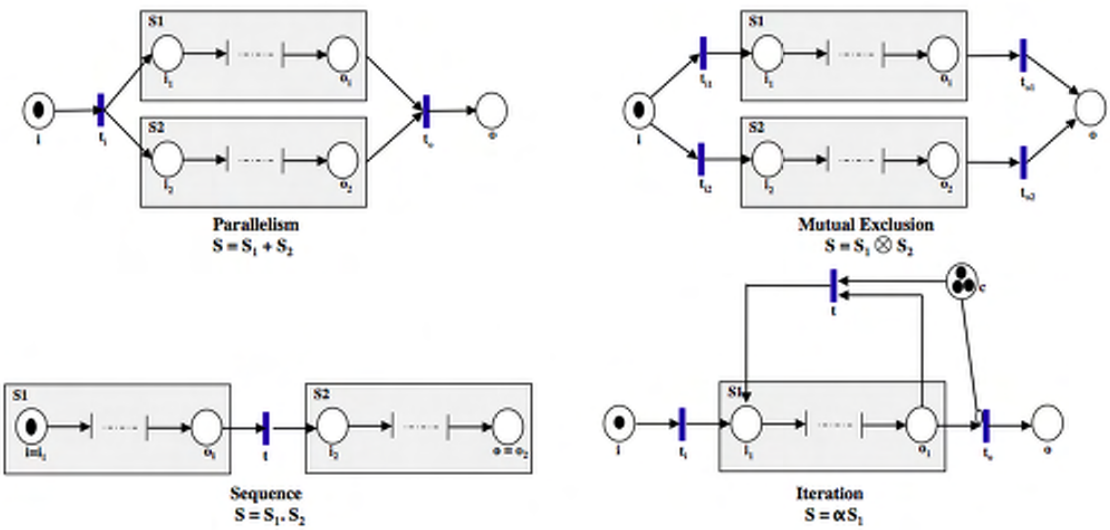


Figure 7.10. Service Composition Operators modeled in Petri Nets (Zhovtobryukh [2007]).

3. A composite service is a service which comprises other services based on a composition model. The composition model can be uniquely and unambiguously represented by an expression written in service composition algebra. Service composition algebra includes basic composition operators including: sequence, mutual exclusion, parallelism, and iteration. Figure 7.10 represents basic service composition operators.

Analysis of several important properties of Petri net models helps to discover flaws of service design or verify its correctness. Reachability tree (Peterson [1977]) is the major analysis tool for problems of safeness, conservation, and liveness. For example, Narayanan and McIlraith [2002] proposed a first order semantic model for DAML-based service descriptions and then encodes them in a Petri net formalism. The resulting Petri net provides decision procedures for web service simulation, verification, and composition. Moreover, it addresses the verification of safety and deadlock properties. However, validation needs to be done through interactive simulation where a service client feeds a test suite to the system to see whether it generates the expected output.

7.9.2 Web Service Modeling by Process Algebra

Process algebra has been proposed widely for modeling and analyzing web services for certain reasons. For example, algebraic structures are simple and at the same time efficient for modeling the dynamic behaviour of web services. CCS and π-calculus have been used to model simple and composite web services in several studies.

Service Operation Type	Operation Example	π-calculus Process Expression
one-way	< operation name = "a" > < input message = "m"/ > < / operation >	$a < m >$
request-response	< operation name = "a" > < input message = "m"/ > < output message = "n"/ > < / operation >	$a < m > \bar{a} < n >$
solicit-response	< operation name = "a" > < output message = "m"/ > < input message = "n"/ > < / operation >	$\bar{a} < m > a < n >$
notification	< operation name = "a" > < output message = "m"/ > < / operation >	$\bar{a} < m >$

Figure 7.11. Service operations modeled in π-calculus (Sangiorgi and Walker [2003]).

1. This chapter discussed how to formalize a web service using CCS. Similarly, Koshkina and van Breugel [2004] provided a CCS model of simplified BPEL (called BPE) by providing its syntax and semantics. A CCS formalization of BPEL business processes that adds protocol information to the specifications of interacting Web Services was described in Mara et al. [2006]. CCS has been also used widely for model checking, pre-order checking, and equivalence checking of web services (Liu et al. [2005]). Finally, there are several automated tools for analysis of systems modeled by CCS such as the Concurrency Workbench of the New Century and Edinburgh Concurrency Workbench.

2. π-calculus (Deng et al. [2006]) is an extension to CCS, suitable for modeling dynamic and mobile distributed systems. π-calculus permits channel name to transmit on channels (using basic sending and receiving actions) and to dynamically create or destroy channels.

   - $\alpha ::= \ x < y > \ | \ \overline{x} < y > \ | \ \tau$ representing atomic actions (input, output, and silence) where x and y are channel names.
   - $P ::= 0 \mid \alpha \cdot P \mid P + P \mid P || P \mid (\nu x)P \mid !P$ representing processes where (!) is used for replication and (νx) restricts name x to process P.

   The behavior of web services includes two aspects: (i) how the service interacts with the system through sending and receiving messages (external view); (ii) how the service generates and consumes messages which refers to the state transitions (internal view).

   Using the π-calculus to model the behavior of a service (Su et al. [2008]), a π-calculus process represents a web service. The service operations are channels used to communicate

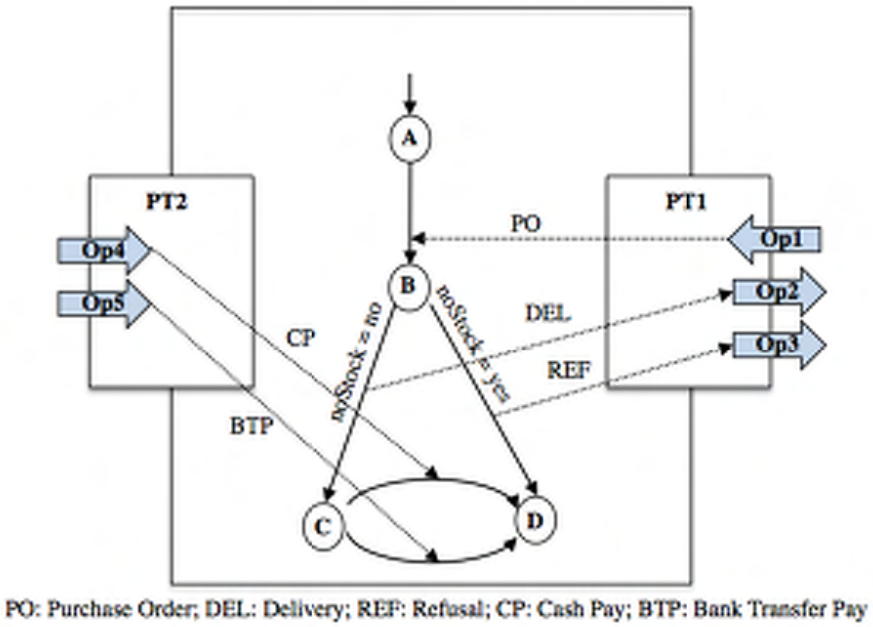


Figure 7.12. A vendor service discussed in Sangiorgi and Walker [2003].

with other processes. There are four types of operation: one-way, request-response, solicit-response and notification. Figure 7.11 shows these operations and the corresponding π-calculus expressions. For example, Figure 7.12 illustrates a vendor service interacts with customer services. This service can be formalized as follows:

$$P_v = Op1 < PO > \cdot(\overline{Op2} < DEL > \cdot(Op4 < CP > + Op5 < BTP >) + \overline{Op3} < REF >)$$

After modeling services as π-calculus processes, the interaction between services can be modeled as a combination of processes.

In addition to CCS and π-calculus, other process calculi have been used to formalize dynamic behaviour of web services. For example, Rouached et al. [2006] proposed an event driven approach to verify web services that are coordinated by a composition process expressed in BPEL by introducing Event Calculus. Transactions and their corresponding compensations in web services were proposed to model in a modified version of π-calculus, *web*π (Mazzara and Govoni [2005]).

7.9.3 Service Certification

Web service testing (verification and validation) is essential for managing service-based enterprise systems, and is receiving increasing attention in new technologies such as Web 2.0 and the Personal Web. Certification of services has been indicated in several literature studies. Damiani

et al. [2009] introduced the concept of a third party certifier that checks and certifies SOA systems based on test cases. A customized service certifier, called an evaluation body, was proposed by Anisetti et al. [2010] to check the conformance of the selected services with security requirements expressed in terms of security properties. Similarly, the problem of assessing and certifying the correct functioning of SOA and Web services has been studied by Damiani and Mana [2009]. They introduced a framework together with certificates based on test cases. Testing web services using fault coverage has been proposed to check the conformance of Web services to their WSDL specification (Dong and YU [2006]). A formal test-case generation technique was proposed by Bai et al. [2005] to automatically generate test cases starting from the services WSDL specification. Consequently, the proposed approaches mostly use test cases to certify web services. However, in this thesis, a formal approach is proposed to certify service composition models based on business component and client-defined service composition policies.

In model-based testing of web services, a model is used to generate a set of test cases for validating the web service. In Bertolino et al. [2006], a service provider publishes the behavioural descriptions of its web service (modeled in a UML 2.0 Protocol State Machine diagram) into a service registry. Based on this model, the registry generates a set of test cases, which run on the web service. Similarly, Wang et al. [2007] derives an equivalent Petri-net model of the OWL description of a service composition model. Test cases, which are based on this model, are used for model checking. Model-based approaches based on Finite State Machines (Sinha and Paradkar [2006] and Keum et al. [2006]) and graph transformation (Heckel and Mariani [2005]) have also been proposed for validating composite web services.

7.9.4 Scalability of Model Checking Tools

Model checking is usually linear in the size of the state space, but the size of the state space is usually exponential in the system program. Model checking can be defined as a tree construction, where nodes correspond to sets of program states and program statements label edges. The goal is to generate all states that the program can reach, and then check if error states were included. Several studies have been done to improve the scalability of model checkers. The existing solutions can be categorized as follows.

1. **Divide-and-Conquer Approaches**: utilize distributed computation environments to overcome memory limitations of a single workstation, as their primary objective. Some of these approaches also deliver increase of performance. For example, Stern and Dill [2001] checked first local properties of individual components, then combined these to prove correctness of the whole system. Lopes and Rybalchenko [2011] proposed a centralized approach with a single master node and a set of slaves nodes. The master node keeps the reachability tree

and a queue of nodes whose successors need to be computed by the slaves. Finally, Holzmann et al. [2008] exploited the availability of multiple CPUs that run several instances of a model checker in parallel with different state space exploration strategies. Each instance runs reachability computation sequentially.

2. **Algorithmic Approaches**: include methods to tackle the effects of state explosion such as state space compression and caching. They also provide different representations for a state space or avoid exploring parts of the state space (e.g., Lamborn and Hansen [2006]).

3. **Bounded Model Checking**: verifies only counterexample executions of a state space which are shorter than some fixed length (e.g., Clarke et al. [2001]).

This thesis provides a formal model to describe the dynamic behavior of web services. The state space of a data service can be considered by a single node tree. However, a task service requires two separate trees representing the task builder (single node) and the task process where each node corresponds to a business action invocation. Similarly, a composite service is modeled by a tree whose nodes correspond to partner services invocation. Consequently, divide-and-conquer approaches could be used where the master keeps track of the composition model and the slaves verify state spaces generated by partner services. However, this task is out of scope for this thesis but it can be a topic for future research.

7.10 Conclusions

Dynamic substitutability of web service composition refers to the ability to use candidate services. However, service clients may have different expectations from the composition of web services. Therefore, SOA requires an automated mechanism for verifying and validating service composition models. This chapter presented a formal model for web services in a distributed business process execution environment, where business components of web services can be provided from different sources. Based on this model, a service composition certifier was introduced to verify the compatibility and validate satisfaction with partner services in a service composition model.

Table 7.14. Modal μ-Calculus propositions corresponding to the policy conditions in Table 7.8 (left) and their verification results (right).

Policy Condition #1	
$(\mu\ Z.Live \wedge [[-\{Get_{sellAdvice}\ ,\ \epsilon\}]]Z)$ $\wedge$ $(\mu\ Z.Live \wedge [[-\{Get_{buyAdvice}\ ,\ \epsilon\}]]Z)$	$\surd$
Policy Condition #2	
$(\nu\ Z.\ [Receive_{sellOrderConfirmation}]\ P1 \wedge [-]Z)$ $\wedge$ $(\nu\ Z.\ [Receive_{sellOrderCancellation}]\ P2 \wedge [-]Z)$ $P1 = \mu Z. < - > True \vee [-Get_{buyAdvice}]Z$ $P2 = \mu Z. < - > True \vee [-Get_{buyAdvice}]Z$	$\times$
Policy Condition #3	
$(\nu\ Z.\ [Receive_{deposit}]\ P1 \wedge [-]Z) \wedge P2$ $P1 = \mu Z. < - > True \vee [-Write_{uCash}]Z$ $P2 = \nu Z.[Receive_{deposit}][Receive_{deposit}]False \wedge [Receive_{deposit}, Write_{uCash}]Z$ $\wedge$ $(\nu\ Z.\ [Receive_{withdraw}]\ P3 \wedge [-]Z) \wedge P4$ $P3 = \mu Z. < - > True \vee [-Write_{uCash}]Z$ $P4 = \nu Z.[Receive_{withdraw}][Receive_{withdraw}]False \wedge [Receive_{withdraw}, Write_{uCash}]Z$	$\surd$
Policy Condition #4	
$(\nu\ Z.\ [Receive_{buyOrderConfirmation}]\ P1 \wedge [-]Z) \wedge P2$ $P1 = \mu Z. < - > True \vee [-Write_{uHolding}]Z$ $P2 = \nu Z.[Receive_{buyOrderConfirmation}][Receive_{buyOrderConfirmation}]False \wedge$ $[Receive_{buyOrderConfirmation}, Write_{uHolding}]Z$ $\wedge$ $(\nu\ Z.\ [Receive_{sellOrderConfirmation}]\ P3 \wedge [-]Z) \wedge P4$ $P3 = \mu Z. < - > True \vee [-Write_{uHolding}]Z$ $P4 = \nu Z.[Receive_{sellOrderConfirmation}][Receive_{sellOrderConfirmation}]False \wedge$ $[Receive_{sellOrderConfirmation}, Write_{uHolding}]Z$	$\surd$
Policy Condition #5	
$Order(Send_{buyOrder}, Send_{withdraw}, Receive_{buyOrderConfirmation}) \vee$ $Order(Send_{buyOrder}, Send_{withdraw}, Receive_{buyOrderCancellation}, Receive_{deposit}, 0) \vee$ $Order(Send_{buyOrder}, Send_{withdraw}, Receive_{deposit}, 0)$	$\surd$

Chapter 8

A QoS-Aware Decision Model for Web Service Development [1]

An enterprise system needs to provide different types of web services to model actual services in the corresponding business domain. This work has proposed to categorize web services into data and task services. While a data service processes client data at the server side, a task service employs a service representative, as a generic client side software agent, to process the client data locally at the client side. Task services maintain client privacy by locally processing client sensitive data and reducing the required network bandwidth. However, they limit the computational power of web services to the client platform. This chapter proposes a decision model, which uses the analytic hierarchy process method to help service developers decide on the best type of business service for a specific functionality. The decision model includes evaluation functions to determine the relevant quality of service (QoS) parameters. A case study is used to demonstrate the process of decision making concerning alternative services.

8.1 Introduction

Data Services, representing typical web services, process client data completely at the server (provider) side. However, server-side processing of web services requires transferring client data to the service providers, which may cause data privacy violations, security breaches, or network traffic overloads. In the real-world business domain, an enterprise organization usually sends an agent or other personnel (e.g., a representative, installer, maintainer, or trainer) to the client side to deliver services locally. To address this issue, *Task Services* have been proposed, which are web

[1] This chapter is mostly based on the following paper: M. Najafi, K. Sartipi, and N.Archer, *A QoS-Aware Decision Model for Web Service Development: Server-side Data Services or Client-side Task Services*, IBM CASCON 2011 (Accepted).

services with the capability of processing client data at the client side using a generic software agent called the *Service Representative*. Then a task service performs the required server-side processing and defines a *task* to customize the generic service representative that performs client-side processing. Server-side data services and client-side task services are complementary, and each has advantages and disadvantages that must be considered in their selection. Client-side task service applications include, but are not limited to, the following situations.

- *Context-aware services*: improving client privacy by processing confidential client data (e.g., personal health data and financial information) at the client side.
- *Real time and event-triggered services*: improving service response time by locally processing client data.
- *Client data intensive services*: reducing network traffic by processing large client data volumes (e.g., live video streaming) at the client side.
- *Client-side service composition*: task services can be composed at the client side. Server-side composition imposes extra client data transmission loads between the service client and an external service orchestrator which could increase response time, network traffic, and security vulnerability.

Alternatively, server-side data service represents a better option in the following applications.

- *Server data intensive services*: data services have direct access to the server's databases which is essential in data-intensive applications.
- *Compute intensive services*: service representatives provide limited computational power which may result in increasing the service response time.
- *Intelligent services*: client-side processing of a task service may require some enterprise assets (e.g., sensitive data or knowledge). Revealing them to the client side may violate enterprise privacy.

There is not always a clear boundary between service types and a web service can be classified into more than one category. Therefore, service developers have to decide whether a web service should be designed and developed as a data service or as a task service. This chapter proposes a decision model to guide service developers when deciding on the proper type of web service. Based on the decision model, each service is modeled by a business process and the service developer is given an objective function to compare the data service and task service versions of the web

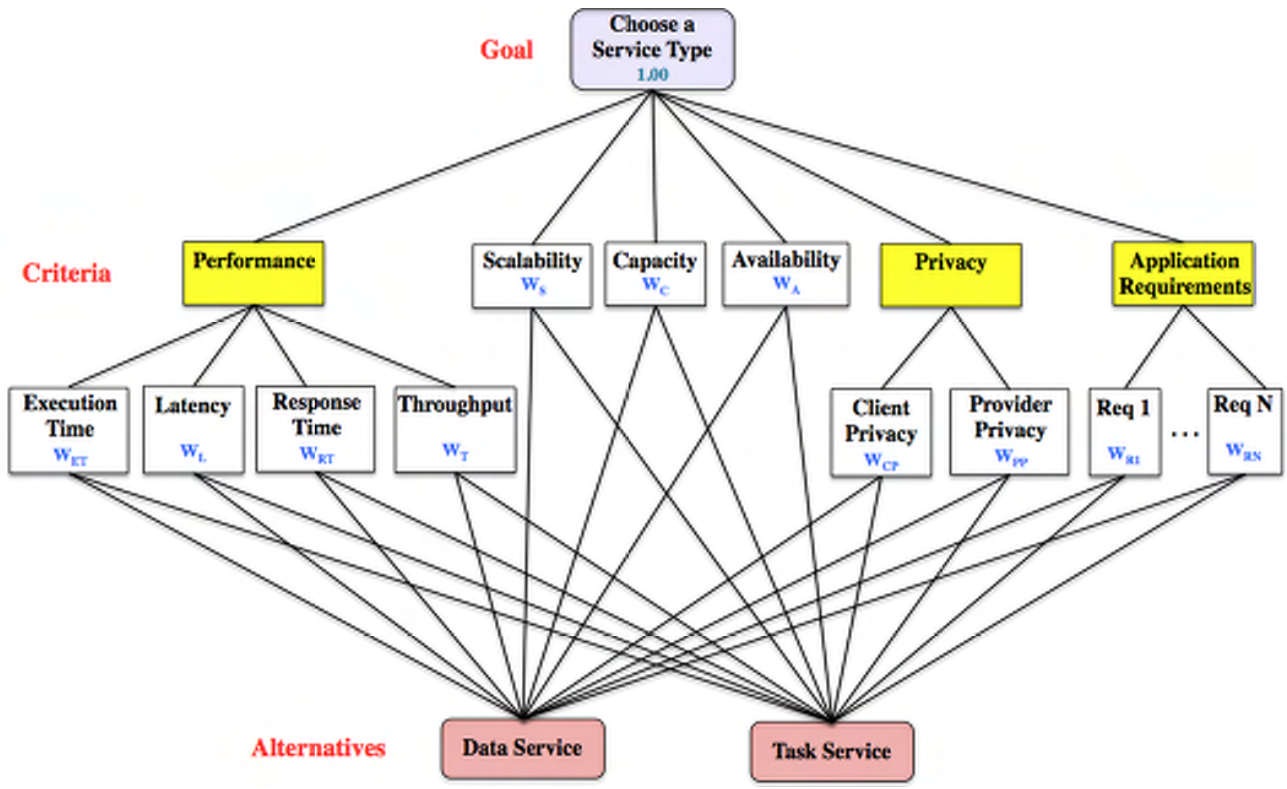


Figure 8.1. Proposed decision model to help service developers to determine the proper service type (data or task service) for a web service.

service. The definition of the objective function is based on quality of service (QoS) parameters. Finally the foreseen applications of this work for web service development are where there is a long term relationship between service providers and clients that justifies the cost associated with the decision making process.

The organization of this chapter is as follows. The proposed decision model is discussed in Section 2. A case study of the decision making process is presented in Section 3. Section 4 discusses work that is related to this approach. Finally, conclusions are discussed in Section 5.

8.2 Decision Model

The analytic hierarchy process (AHP) (Saaty [2008]) is a structured technique for dealing with complex decisions. This technique helps decision makers find a decision that best suits their goal and their understanding of the problem. Users of the AHP first decompose their decision problem into a hierarchy of more easily comprehended sub-problems, each of which can be analyzed independently.

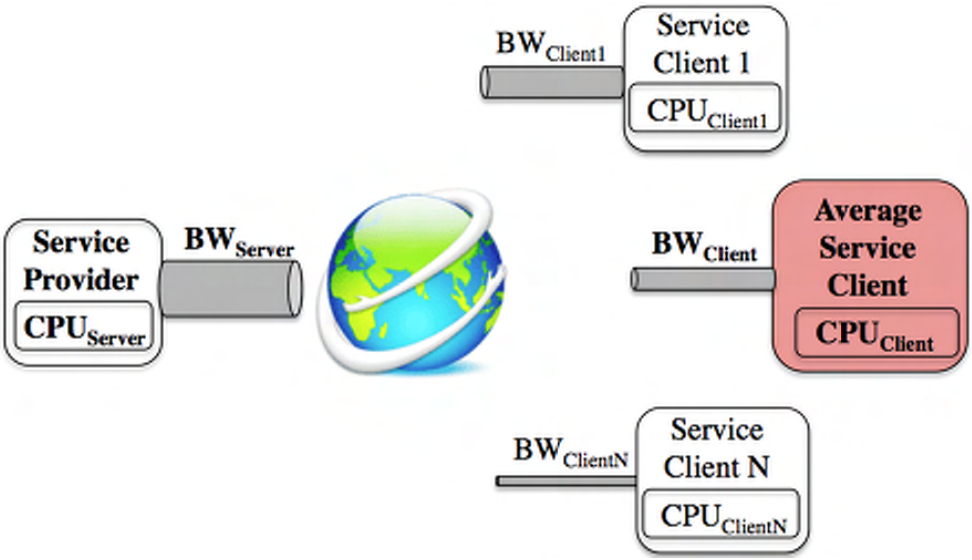


Figure 8.2. Required network characteristics in the decision making process.

Figure 8.1 represents the proposed decision model (AHP), containing the decision goal (service type), the alternatives for reaching it (data or task service), and the criteria (QoS parameters) for evaluating the alternatives. Prior to applying the decision model, the service developer needs to obtain information about the service clients such as client computing capabilities (e.g., CPU speed) and the client's network bandwidth BW (Figure 8.2). Since the web service developed will be used by different service clients, the service developer must decide on the characteristics of an average service client.

8.2.1 Alternative Services

A web service executes one or more business processes for its enterprise (the service provider), where each business process applies business rules and performs business actions on internal (server-side) and external (client-side) business objects in a defined order. Therefore, a web service can be modeled by a collection of business components, including business processes, rules, actions, and objects. Moreover, a web service can be modeled as either a data service or a task service. While a data service applies the business components at the server-side, the task service sends the business components to the client-side to be applied by the service representative.

To apply the decision model, the service developer first needs to design and develop equivalent versions of the web service in the form of data services and task services. A data service includes an integrated server-side business process (called a *service* process) and a task service includes a server-side process (called a *task builder* process) and a client-side process (called a *task* process). Figure 8.3 represents the business processes which are required in the decision making process.

A data web service DWS is modeled by a function that receives remote service parameters x^r_{DWS} from the service client and returns remote service responses y^r_{DWS}.

$$DWS : X^r_{DWS} \longrightarrow Y^r_{DWS}$$ [2]

Similarly to a data service, a task web service TWS is modeled by a function through which it receives remote service parameters x^r_{TWS} from the service client and returns the generated task message $task_{TWS}$ to the service representative SR. Then the service representative processes the local service parameters x^l_{TWS} to generate the local service responses y^l_{TWS} for the service client.

$$TWS : X^r_{TWS} \longrightarrow Task_{TWS}$$
$$SR : Task_{TWS} \times X^l_{TWS} \longrightarrow Y^l_{TWS}$$

Consequently, in order to design business service processes, the service developer needs to identify the local and remote service parameters and responses for each alternative service.

8.2.2 QoS Parameters

The QoS requirements for web services refer to the quality aspect of web services. Data and task services process client data at the server and client platforms, respectively, which affect some of the QoS parameters. Consequently, we divide QoS parameters into two categories as follows.

- *Platform independent QoS*: includes reliability, robustness, exception handling, and interoperability. These parameters depend on the service logic and are offered at the same quality level by a data or task version of a web service. This also includes service security because similar authentication and authorization approaches can be used in data and task services.

- *Platform dependent QoS*: includes performance (execution time, latency, response time, and throughput), scalability, capacity, availability, and privacy (for both client and provider). A web service can be used in different client applications where each application requires specific QoS requirements. Consequently, this category also includes application-specific requirements.

8.2.3 Decision Criteria

For the proposed decision model, *platform-dependent QoS* parameters are used as decision criteria. These are evaluated differently for data and task services. This section, describes how each decision criterion can be evaluated.

[2] In defining a function (e.g., DWS), we use the sets of input "parameters" (e.g., X_{DWS}) and output "responses" (e.g., Y_{DWS}) to represent the types of the function's input and output.

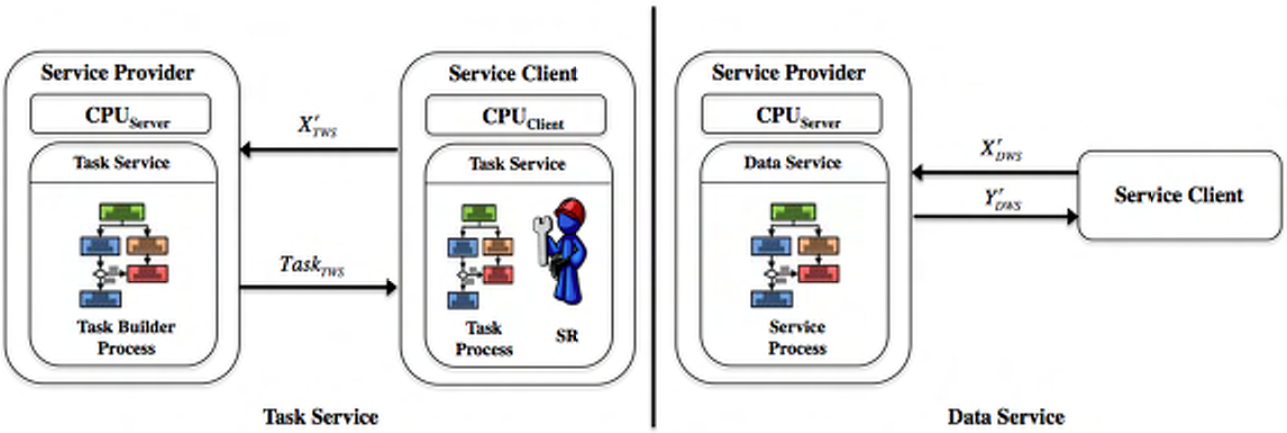


Figure 8.3. Data and task service alternatives for a web service.

1. **Service Execution Time** (ET): the time spent by a system executing the web service process. The execution time depends on four factors: CPU speed, memory size and access time, and process complexity. $ET^{process}_{platform}(x)$ is defined as the time spent by the *platform* to execute the *process* which takes input parameters x. The service developer can obtain execution time values using software profiling techniques available as Application Programming Interfaces (APIs).

   The following equation represents the execution time evaluation function. While the data service process is performed at the server side, the task service process is divided into two processes: *task builder* and *task*, which are performed at the server side and client side, respectively.

$$C_{ET} = \begin{cases} \textbf{Data Service}: \\ ET^{service}_{server}(x^r_{DWS}) \\ \\ \textbf{Task Service}: \\ ET^{task\ builder}_{server}(x^r_{TWS}) + ET^{task}_{client}(x^l_{TWS}) \end{cases}$$

2. **Service Latency** (L): is defined as the time span from the time a service client issues a request for service to the time it receives a response message. Both data and task services receive remote service parameters x^r within the request messages. However, the resulting response messages include remote service responses y^r and task messages for data and task services, respectively.

   Network tools such as *ping* tests can be used to measure latency. The service latency can also be estimated using network characteristics. The following evaluation function, which is a simplified version of the equation given by Liu et al. [2003], provides an estimate of

the service latency. $Size(x)$ returns the size of data x and BW_{Client} represents the network bandwidth of the service client expressed in kilobytes/second.

$$C_L = \begin{cases} \textbf{Data Service}: \\ \frac{Size(x^r_{DWS})+Size(y^r_{DWS})}{BW_{Client}} \\ \\ \textbf{Task Service}: \\ \frac{Size(x^r_{TWS})+Size(task_{TWS})}{BW_{Client}} \end{cases}$$

Data services require transferring all client data volumes from the client to the server. On the other hand, task services process client data locally, implying that the service message size is independent of the volume of client data. Therefore, the task request message is short while the response message containing the task definition could be long.

3. **Service Response Time** (RT): the time between a service request sent and the corresponding service response received, which is defined as follows.

$$C_{RT} = C_{ET} + C_L$$

4. **Throughput**: this is the number of web service requests served in a given time interval (usually measured in number per second). Throughput depends on the service execution time at the provider side (Menasce [2002]). In the case of task services, service clients have their own service representatives, which can process client data in parallel, and therefore the throughput is improved significantly. The following equation, which was inspired by the model given by Ofuji et al. [2002], estimates the service throughput for data and task services.

$$C_T = \begin{cases} \textbf{Data Service}: \\ \frac{1(sec)}{ET^{service}_{server}(x^r_{DWS})} \\ \\ \textbf{Task Service}: \\ \frac{1(sec)-ET^{task}_{client}(x^l_{TWS})}{ET^{task\ builder}_{server}(x^r_{TWS})} \end{cases}$$

Suppose a data service has one second to complete service requests. However, service representatives ask for ET^{task}_{Client} to perform client-side process of a task service. During this time, the server is free to serve other services (the server and service representatives are working in parallel). Consequently, all service requests processed in $[0, 1\ sec - ET^{task}_{Client}]$ by the server will be processed by the service representatives before the 1 second time period expires.

5. **Scalability**: this represents the capability of increasing the computing capacity of the server to process more client requests per second. The computing capacity can be increased by adding more computing units and/or assigning more CPU shares to the web service processes. In this case, C_L represents the latency associated with each individual service client, whereas the computational power of the server is shared among different service clients. The following equation represents the gain in response time that comes from increasing the server computing capacity by a factor α_S ($\alpha_S \geq 1$).

$$C_S = \begin{cases} \textbf{Data Service}: \\ \frac{C_{RT}}{\frac{ET^{service}_{server}(x^r_{DWS})}{\alpha_S} + C_L} \\ \\ \textbf{Task Service}: \\ \frac{C_{RT}}{\frac{ET^{task\ builder}_{server}(x^r_{TWS})}{\alpha_S} + ET^{task}_{client}(x^l_{TWS}) + C_L} \end{cases}$$

6. **Capacity**: the limiting number of simultaneous service requests which can be served by the provider with guaranteed performance (e.g., service response time). The capacity for the provider depends on two factors: the server's network bandwidth, and the server-side execution time for the web service (Chen and Mohapatra [2002]). C_{RT} represents the service response time when there is only one service request. In case of multiple service requests, the acceptable service response time is defined by α_{RT}. Servers can simultaneously transfer messages and process service requests. Therefore, $\frac{\alpha_{RT}}{C_{RT}}$ represents a lower limit for the service capacity (in the case of a data service). The maximum limit is the number of service messages which can be transferred through the server channel in α_{RT}. The service capacity is estimated by the average of the minimum and maximum values. For a task service, the service capacity can be estimated similarly by considering the service process distribution between server and client platforms.

$$C_C = \begin{cases} \textbf{Data Service}: \\ Avg(\frac{\alpha_{RT}}{C_{RT}}\ ,\ \frac{\alpha_{RT} \times BW_{server}}{Size(x^r_{DWS}) + Size(y^r_{DWS})}) \\ \\ \textbf{Task Service}: \\ Avg(\frac{\alpha_{RT} - ET^{task}_{client}(x^l_{TWS})}{C_L + ET^{task\ builder}_{server}(x^r_{TWS})}\ ,\ \frac{\alpha_{RT} \times BW_{server}}{Size(x^r_{TWS}) + Size(task_{TWS})}) \end{cases}$$

7. **Service Availability**: this is the probability that the service is up and operating. A web service can be temporarily shut down for several reasons such as service maintenance.

However, a task service can take advantage of its client-side service representative during the service shut down period. For this purpose, a task definition needs to include an activation time to start operating when the server is not available. The service availability for a data service is simply defined by the following equation (Ran [2003]).

$$C_A = \frac{totaltime - idletime}{totaltime}$$

On the other hand, the service availability for a task service is not straightforward. During an idle period, the task service can not be invoked by service clients. But service clients that have invoked the service during the service activation period have access to the service through the service representative. However, this service availability is limited to task services where the service representative does not require any task updates in order to provide proper service responses. Moreover, in this case, an estimation of service client distribution over the activation and idle time is required. The following equation represents task service availability under different conditions, where $C\#_T$ represents the number of service clients during the evaluation period (e.g., a day). Similarly, $C\#_I$ represents the number of service clients during the service idle time.

$$C_A = \begin{cases} \textit{If task updates required} \\ \frac{total\ time - idle\ time}{total\ time} \\ \textit{Else} \\ \frac{total\ time - idle\ time}{total\ time} + \frac{idle\ time}{total\ time} \times \frac{C\#_T - C\#_I}{C\#_T} \end{cases}$$

8. **Client Privacy**: the client's ability to keep its personal and sensitive information private and local. The service client may need to send its personal information to the service provider as service parameters. However, task services divide the service parameters into two categories: local and remote, where local service parameters, including client sensitive information, are processed locally by the service representative.

   To compare client privacy provided by data and task services, the service developer first needs to identify any client sensitive information among the local and remote service parameters, represented by the set S_{client}. A degree of sensitivity is assigned to each member of this set, represented by a positive number $Sensitivity_x$ for each $x \epsilon S_{client}$. The following function gives a measure of how client privacy is violated by different types of services, where an increase in value represents more client privacy violations.

$$S^{WS}_{client} = \{x \mid x \epsilon X^r_{WS} \ \wedge \ x \textit{ is sensitive}\}$$

$$C_{CPV} = \sum_{x \epsilon S^{WS}_{client}} Sensitivity_x$$

9. **Provider Privacy**: this represents the provider's ability to keep its personal and sensitive information private and local. Similarly to client privacy, data and task services represent different behaviours regarding service provider privacy. A data service performs all the processes at the server side. However, required task knowledge and task data for the service representative can include enterprise assets and resources. Revealing them to the client side may violate enterprise privacy. To prevent this privacy vulnerability, a service provider can use one of the following techniques.

   - Enterprise knowledge can be divided into locally applied (at the provider side) by the service or externally applied (at the client side) by service representatives. Therefore, the critical knowledge remains at the service provider, while non-critical knowledge can be sent to the service representative.
   - The service client receives service responses from the service representative. The client does not have access to the knowledge transferred between the service provider and its representative. Encryption techniques can be used for data transmissions between a service provider and representative to improve enterprise privacy.

   The following function C_{PPV} evaluates the provider's privacy violation where $S_{provider}$ includes sensitive task knowledge and task data. The encryption factor α_{Enc} is in the range [0,1], representing the efficiency of the encryption technique used.

$$S^{TWS}_{provider} = \{x \mid x \in Task_{TWS} \ \wedge \ x \ is \ sensitive\}$$

$$C_{PPV} = \begin{cases} \textbf{Data Service} \\ 0 \\ \\ \textbf{Task Service} \\ \sum_{x \in S^{TWS}_{provider}} Sensitivity_x \times (1 - \alpha_{Enc}) \end{cases}$$

10. **Application Requirements**: A web service can be invoked by different client applications where each application has specific requirements. While previous QoS metrics describe only the general features of a web service, this metric is based on client-side service applications. Although it is not possible for service developers to consider all different types of client applications that would invoke this service, the service developer can categorize all main applications.

For example, in a dynamic environment, where client data are changing frequently, a web service must be invoked whenever the service parameters are changed. In this case, a task service is more desirable, because the service representative can take care of changes in the local service parameters. Consequently, the number of task service invocations will be decreased dramatically, thus improving performance metrics. For this application, the corresponding comparison measure is C_{SC} which represents the number of service calls in a given time. The following equation is the evaluation function for this metric, where β^r_{WS} represents the frequency of updates in the values of X^r_{WS} in a given amount of time (e.g., 1 minute).

$$C_{SC} = \beta^r_{WS}$$

8.2.4 Objective Function

QoS parameter values are not comparable and they must be transformed or normalized before the comparison takes place. A normalization function is defined for each QoS parameter in order to map values from its domain to its co-domain [0,1], preserving the original input data distribution. Platform-dependent QoS parameters (decision factors) are categorized into two main groups: desirable factors (including $C_T, C_S, C_C,\ and\ C_A$) and undesirable factors (including $C_{ET}, C_L, C_{RT}, C_{CPV}$, and C_{PPV}). A desirable (undesirable) decision factor is maximized (minimized) by the objective function. Therefore, a normalization function for a desirable (undesirable) decision factor must assign greater numbers to greater (smaller) QoS values.

A service developer can define a variety of customized normalization functions. For example, given QoS values for a data service q_{dws} and a task service q_{tws}, a simple linear normalization function can be defined as follow.

$$N_q(x) = \begin{cases} q\ is\ desirable \\ \frac{x}{max(q_{dws}, q_{tws})} \\ \\ q\ is\ undesirable \\ 1 - \frac{x}{max(q_{dws}, q_{tws})} \end{cases}$$

Alternatively, normalization functions which utilize upper and lower bounds can be defined. Figure 8.4 represents three examples of such functions. In addition to normalization functions, the service developer needs to define a numerical weight w_q for each QoS parameter q in the decision model.

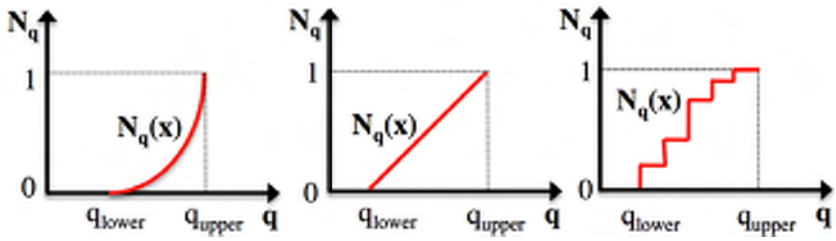


Figure 8.4. Exponential (left), linear (middle), and step (right) normalization functions for QoS parameters.

The following equation represents the objective function derived from the decision model, represented in Figure 8.1. This function is a weighted sum of the QoS parameters where Φ is the set of decision factors, $\Phi = \{ET, L, RT, T, S, C, A, CPV, PPV, App\}$, and $\sum_{x \epsilon \Phi} w_x = 1$. It should be noted here that the weighting is subjective so it is important to decide on the weights in advance of the calculated results to avoid introducing bias into the decision.

$$\mathbf{C_{total}} = \sum_{x \epsilon \Phi} w_x \times N_x(C_x)$$

Using the objective function, the service developer obtains C_{total} for both data service and task service alternatives and chooses the one with the higher value as the better candidate for providing the web service.

Finally, the objective function is likely to generate different results for different service input parameters. Therefore, in order to make an accurate decision, the service developer needs to categorize the service inputs, based on their size and complexity, and then calculate the objective function for each category. The final decision should be made by considering these different categories.

8.3 Case Study

A case study of a face detection web service is useful in demonstrating the different phases of the proposed decision making process.

8.3.1 Case Study Specification

A face detection service is a primary need in many fields, including face recognition, video surveillance, and human motion detection. A face detector reviews an image to determines which of its regions contain face(s). Several face detection approaches have been proposed in the literature, including approaches based on face spaces, neural networks, and template matching (Jafri and Arabnia [2009]). The following adopts a two-step template matching algorithm for use in

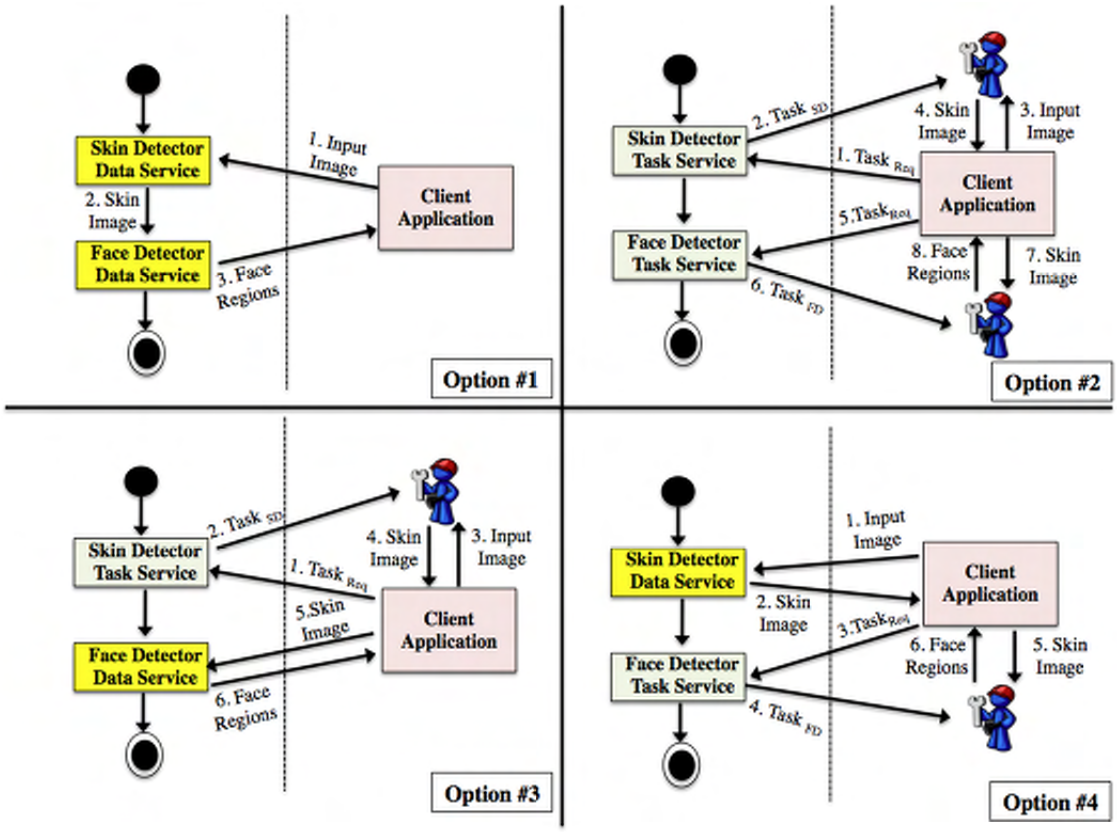


Figure 8.5. Alternative face detection services include a composite data service (*option 1*), a composite task service (*option 2*), and two hybrid services (*options 3* and *4*).

this case study. This algorithm includes two steps, where each step is developed by a process as follows.

1. *Skin Detector*: finds skin regions using explicit boundary rules on color values (Kakumanu et al. [2007]). This step generates a binary image where all *skin* and *non-skin* pixels are assigned *Black* and *White*, respectively.

2. *Face Detector*: extracts face regions from skin regions using face templates, as follows: (i) enhances the binary image to eliminate noise; (ii) segments the enhanced image into connected regions; (iii) selects the potential face regions based on their size and width-to-height ratio; (iv) creates a feature vector for each selected region by detecting its edges; (v) compares feature vectors with the given face templates to find face regions; and (vi) shows face regions in rectangles.

8.3.2 Alternative Services

As discussed in Section 8.2, each business process can be provided as either a data service or a task service. Given two business processes, the alternatives include four services, represented in Figure 8.5. The alternative services include a composite data service (*option 1*), a composite task service (*option 2*), and two hybrid services (*options 3* and *4*).

The composite data service was implemented with AJAX-RPC APIs, and the composite task service and hybrid services were implemented through a developed prototype named *Service Representative Version 1.1* (SR v1.1). This prototype is based on J2EE 1.5 technologies and an Apache Tomcat 6.0 application server. Moreover, it uses the *Drools* rule flow engine as the service representative process engine. Finally, *SR v1.1* is divided into two Java packages: a *TaskService* package used by service providers to develop task services, and a *ServiceRepresentative* package used by service clients to install the generic service representative and invoke different task services. See Appendix A for more details.

8.3.3 Decision Model

In this section, the values of QoS parameters for each alternative face detection service in Figure 8.5 are obtained, and then the objective function is applied to obtain the best service type. To perform the evaluation, a server platform with a (2.66 GHZ) 2 Core Intel CPU was used, connected to the network via a 1 Mbyte/Sec link. For the client side, an average service client was used, with a (1.8 GHZ) single Core Intel CPU connected to the network via a 128 KByte/Sec link.

The test suite consisted of 10 randomly selected images where all images were in color with various visual qualities, details and different sizes (80 Kbytes - 2 Mbytes). This test set includes images with and without facial shapes.

1. **Execution Time**: depends on the size and complexity of the input image as well as the processing speed of the computing unit. Figure 8.6 represents C_{ET} for the test samples. Based on the applied algorithm, skin detector execution time depends on the size of the input image. However, the face detector execution time depends on the number of detected skin regions. Consequently, the execution time is not always incremental. The composite data service (option 1) performs all the processes at the server side and provides the fastest results. For example, given a 360 Kbyte input image, the execution time for the skin detector process is 33 milliseconds at the server side and 49 milliseconds at the client side. Similarly, the execution time for the face detector process, which takes the same image, is 84 and 164 milliseconds at the server side and client side, respectively. The calculated average execution times, represented by $\overline{C_{ET}}$, were used in the other QoS evaluations.

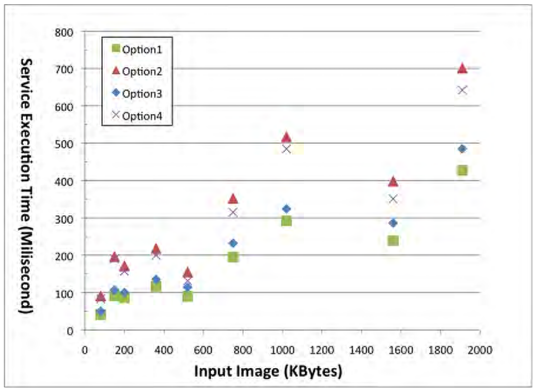


Figure 8.6. Service Execution Time for the candidate services. $\overline{C_{ET}^{\#1}} = 202$, $\overline{C_{ET}^{\#2}} = 357$, $\overline{C_{ET}^{\#3}} = 234$, **and** $\overline{C_{ET}^{\#4}} = 326$.

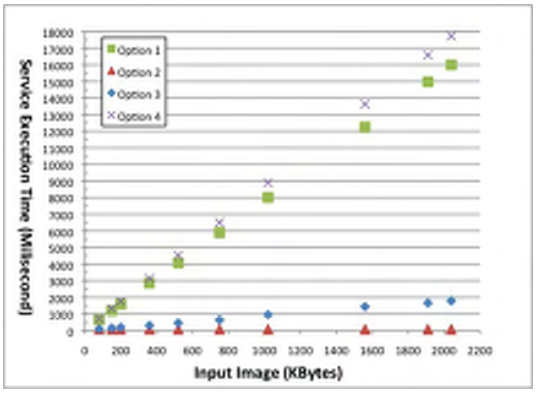


Figure 8.7. Service Latency for the candidate services. $\overline{C_{L}^{\#1}} = 6747$, $\overline{C_{L}^{\#2}} = 78$, $\overline{C_{L}^{\#3}} = 769$, **and** $\overline{C_{L}^{\#4}} = 7485$.

2. **Latency**: depends on both the size of transferred messages and the client network bandwidth. Figure 8.7 represents C_L for the test set where the composite task service (option 2) outperforms other options since the service messages only include the task definitions, not original or modified images. The third option, which requires transferring the (black-white) skin image and the resulting face regions, shows an acceptable latency. For example, given the 360 Kbyte input image, the skin image and face regions are 23 Kbytes, and 3 Kbytes, respectively. Finally, the size of task messages is independent of the size of input images. $Task_{Skin\ Detector}$ is 2Kbytes and $Task_{Face\ Detector}$ is 52 Kbytes, including face patterns.

3. **Response Time**: depends on both the execution time and latency. It is needed as it is usually one of the key QoS metrics required by service clients. Moreover, based on Figures 8.6 and 8.7, it turns out that good execution time performance does not necessarily mean

good response time performance, and vice versa. Figure 8.8 compares C_{RT} among candidate services using the test set. Based on these results, the second and third services perform better than the first and fourth services since large client data volume affects service latency more than execution time.

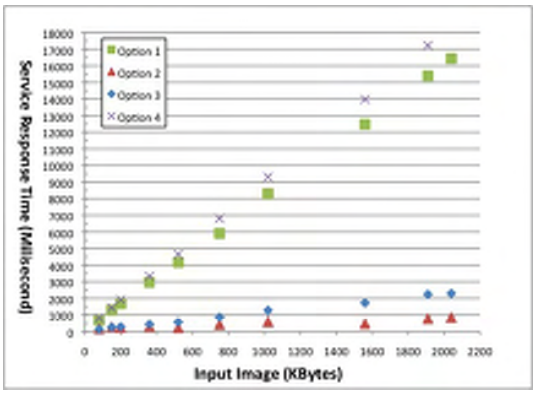


Figure 8.8. Service Response Time for the candidate services. $\overline{C_{RT}^{\#1}} = 6931$, $\overline{C_{RT}^{\#2}} = 435$, $\overline{C_{RT}^{\#3}} = 1013$, **and** $\overline{C_{RT}^{\#4}} = 7791$.

4. **Throughput**: improved by performing processes in parallel and at the client side. The composite task service (option 2) performs all the processes at the client side and therefore it provides the best throughput. Using the average service execution time, the throughputs for the candidate services are as follows: $C_{RT}^{\#1} = 5$, $C_{RT}^{\#2} = 36$, $C_{RT}^{\#3} = 7$, and $C_{RT}^{\#4} = 12$.

5. **Scalability**: improved where the server performs some parts of the service processing and the execution time is comparable with latency. In this case, the gain obtained by increasing the server processing capability to process the client's requests was found to be twice as fast ($\alpha_S = 2$). The results are as follows: $C_S^{\#1} = 1.5\%$, $C_S^{\#2} = 2\%$, $C_S^{\#3} = 9\%$, and $C_S^{\#4} = 0.1\%$.

6. **Capacity**: increased by task services as they require less network bandwidth and server computing share. Assuming that the minimum acceptable response time for this service (α_{RT}) is 8 seconds, the resulting capacities are as follows: $C_C^{\#1} = 1.15$, $C_C^{\#2} = 82$, $C_C^{\#3} = 8.6$, and $C_C^{\#4} = 1.1$.

7. **Availability**: is similar for all the services as we assume that the service does not need maintenance and is functioning 24/7. Therefore, $C_A^{\#1} = 1$, $C_A^{\#2} = 1$, $C_A^{\#3} = 1$, and $C_A^{\#4} = 1$.

8. **Client Privacy**: violated by data services since the service client needs to transfer sensitive information (input images) as service parameters.

$$S_{client} = \{input\ image\}$$

A sensitivity measure assigns a number in [0,1] to sensitive data where a higher value represents more sensitivity, $Sensitivity_{input\ image} = 0.5$. The corresponding privacy violation costs for the candidate services are as follows: $C_{CPV}^{\#1} = 0.5$, $C_{CPV}^{\#2} = 0$, $C_{CPV}^{\#3} = 0$, and $C_{CPV}^{\#4} = 0.5$.

9. **Provider Privacy**: violated by task services since the service provider needs to transfer sensitive information, which includes facial patterns as well as skin detection (SD) and face detection (FD) algorithms, required by the service representative to perform the tasks.

$$S_{provider} = \{patterns, SD, FD\}$$

Using the same sensitivity scale as client privacy, $Sensitivity_{pattern} = 0.8$, $Sensitivity_{SD} = 0.3$, and $Sensitivity_{FD} = 0.5$. $\alpha_{Encryption} = 0.6$ was assumed. The corresponding privacy violation costs for the candidate services are as follows: $C_{PPV}^{\#1} = 0$, $C_{PPV}^{\#2} = 0.64$, $C_{PPV}^{\#3} = 0.12$, and $C_{PPV}^{\#4} = 0.52$.

10. **Application Requirements**: evaluated for three different applications as follows.

    (a) *Still-Image Face Recognition.* This application requires only one web service invocation. $C_{SC1}^{\#1} = 1$, $C_{SC1}^{\#2} = 1$, $C_{SC1}^{\#3} = 1$, and $C_{SC1}^{\#4} = 1$.

    (b) *Video Surveillance.* This application requires invoking the web service every five seconds. The composite task service (option 2) processes all the images at the client side. Consequently, it requires only one web service invocation. $C_{SC2}^{\#1} = 12$, $C_{SC2}^{\#2} = 1$, $C_{SC2}^{\#3} = 12$, and $C_{SC2}^{\#4} = 12$.

    (c) *Human Motion Detection*: This application requires invoking the web service once every second. Similarly to the video surveillance application, the composite task service is the only option which does not require any updates. $C_{SC3}^{\#1} = 60$, $C_{SC3}^{\#2} = 1$, $C_{SC3}^{\#3} = 60$, and $C_{SC3}^{\#4} = 60$.

The QoS parameters that were obtained became comparable after normalization. Figure 8.9 shows three instances of the defined normalization functions. Other normalization functions were defined similarly. Table 8.1 represents the normalized QoS values and the weight assigned to each of them in the process of decision making. Assigned weights should be based on the provider's policies as well as potential client expectations. This can be done through an analysis of the business domain, which is outside the scope of this paper. In this case, we consider equal weights for privacy and performance metrics (each one receives 40%). Application specific QoS requirements are assigned a value of 10%, where they are also evaluated by the performance metrics. The remaining 10% is assigned to the other criteria.

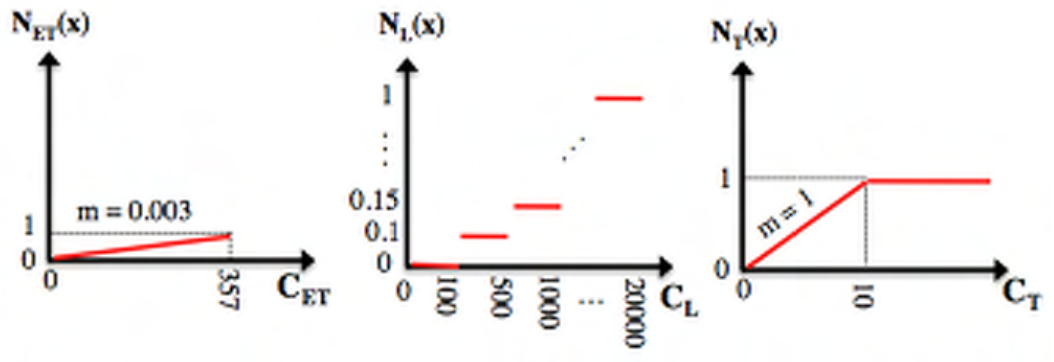


Figure 8.9. Three normalization functions used in the face detection case study.

Table 8.1. Normalized QoS parameters values for candidate face detection web services. The third option receives the highest score and is chosen as the best candidate to model the web service.

QoS metric (weight)	C_{ET} (0.1)	C_L (0.1)	C_{RT} (0.1)	C_T (0.1)	C_S (0.03)	C_C (0.05)	C_A (0.02)	C_{CPV} (0.20)	C_{PPV} (0.20)	C_{SC1} (0.03)	C_{SC2} (0.05)	C_{SC3} (0.02)	**Score** (1.00)
Option 1	1	0.3	0.12	0.5	0.13	0.12	1	0.5	1	1	0.4	0	0.57
Option 2	0.48	1	1	1	0.3	1	1	1	0.5	1	1	1	0.82
Option 3	0.93	0.9	0.83	0.7	1	0.9	1	1	0.8	1	0.4	0	0.85
Option 4	0.65	0.2	0.08	1	0.03	0.1	1	0.5	0.6	1	0.4	0	0.48

Using the Analytic Hierarchy Process (AHP), the total score for each alternative service was obtained by applying the objective function defined in Section 8.2. Based on the total scores, the preferred alternative is the third option. This composite service first uses a task service (*skin detector*) to employ the generic service representative to initially process the client's data at the client side. In the next step, it uses a data service (*face detector*) to perform the more complicated and more sensitive face detection process at the server side. By analyzing the results of this experiments, we conclude that the selected composite service outperforms other services by reducing network traffic and execution time, while at the same time maintaining client and provider privacy.

8.4 Related Work

In this section, some research work found in the related literature is briefly presented. In distributed computing, code on demand, also called code mobility, (Brooks [2004]) refers to any technology that sends executable software programs from a server computer to a client system upon request from the client. Java applets and Microsoft Silverlight are two instances of code mobility. Task service is a client-side processing approach that is an alternative to code mobility.

Task services have advantages over code on demand approaches due to their composability and scalability. Moreover, task messages are not executable, which improves client security.

Code mobility has been compared with traditional client/server approaches. For example, a performance analysis of client/server versus agent based communication was performed by Barnes et al. [2009] where the entire agent moves from device to device rather than having to spend processing time creating a static agent for each device. A hierarchical framework of benchmarks was proposed by Dikaiakos and Samaras [2001] for performance analysis of mobile agent systems. Schwarzkopf et al. [2008] compared Java Remote Invocation (RMI) and .Net Remoting Architecture based on an experimental performance analysis. Similarly, Mangalwede and Rao [2009] investigated three Java-based approaches to distributed computing (Java RMI, Java applet-servlet communication and Mobile Agents) using performance parameters such as code size, latency, response time, partial failure, concurrency, and ease of development. To the best of our knowledge, this work is one of a few attempts to compare traditional (server-side) web services with client-side processing approaches (task services).

Web service performance testing is an emerging field of software engineering which must be carried out at both the server side and the client side. However, choosing the relevant performance criteria is a critical task. At the client side, web service performance depends on the amount of data transmitted over the network. At the server side, selection of programming language and platform, and implementation complexity are the primary contributors to web service performance (Cane [2003]). Data and task services provide the same business logic and are implemented using identical technologies. Consequently, these should be compared, based on their QoS parameters which are relatively easy to obtain. Open-source performance analysis tools such as Apache JMeter, Firebug and YSlow can be used to obtain QoS parameter values efficiently. QoS monitoring approaches (e.g., Zeng et al. [2007]) can also be used.

QoS parameters have been proposed widely in service selection to compare alternative services. For example, an interactive web service choice-making process is proposed by Yu-jie et al. [2005], which takes QoS as a key factor when choosing from functionally equivalent services. Similarly, Al-Masri and Mahmoud [2007] introduce a web service relevancy function used for measuring the relevancy ranking of a particular Web service based on client preferences, and QoS metrics. By considering the service selection problem as an optimization problem, Yu et al. [2007] maximizes an application-specific utility function under end-to-end QoS constraints.

8.5 Conclusions

This chapter proposed a decision model to help service developers make decisions about the proper type (server-side data service or client-side task service) to adopt for web services. The decision criteria are platform dependent QoS which are evaluated differently for data and task services. Moreover, based on the case study provided, a composition of data and task services could exhibit good QoS scores if the task service initially processes the client's data and the pre-processed data are transferred to the server for more complex processing.

Chapter 9

Discussion and Future Work

Conclusions and the directions for future research are presented in this final chapter.

9.1 Discussion

This thesis has presented a solution for the problem of client-side processing using web services. Lacking an efficient way to model client-side web services in traditional SOA models restricts the application and usage of SOA in developing enterprise systems (Chapter 1). Service clients may need to include personal information in the service request messages. This may cause significant privacy and security breaches. Moreover, large volume client data requires large messages to be sent over the network, increasing network traffic.

The proposed solution is based on generic service representatives (Chapter 2) and task services (Chapter 3). The generic service representative provides an executable platform for service providers at the client side which can be customized by the service providers through task messages. While a traditional data service processes client data completely at the server side, a task service is a web service with the capability of processing client data and resources partially or completely at the client side, using a Service Representative. Each task service includes a task definition with three components: task model, task knowledge, and task data. The task components are mapped to business components such as business process models, business rules and actions, and business data, where they can be efficiently transmitted by service messages.

This thesis compared the proposed client-side task services with traditional server-side data services using several case studies. Our experiments showed that a task service can improve QoS parameters in some applications where client data is confidential, large, or dynamic. Since the proposed client-side task services work through message exchanges, they are compatible with typical server-side web services. Therefore, the same protocol stack is used to describe, discover, invoke, and compose task services. The thesis evaluated task services with other client-side pro-

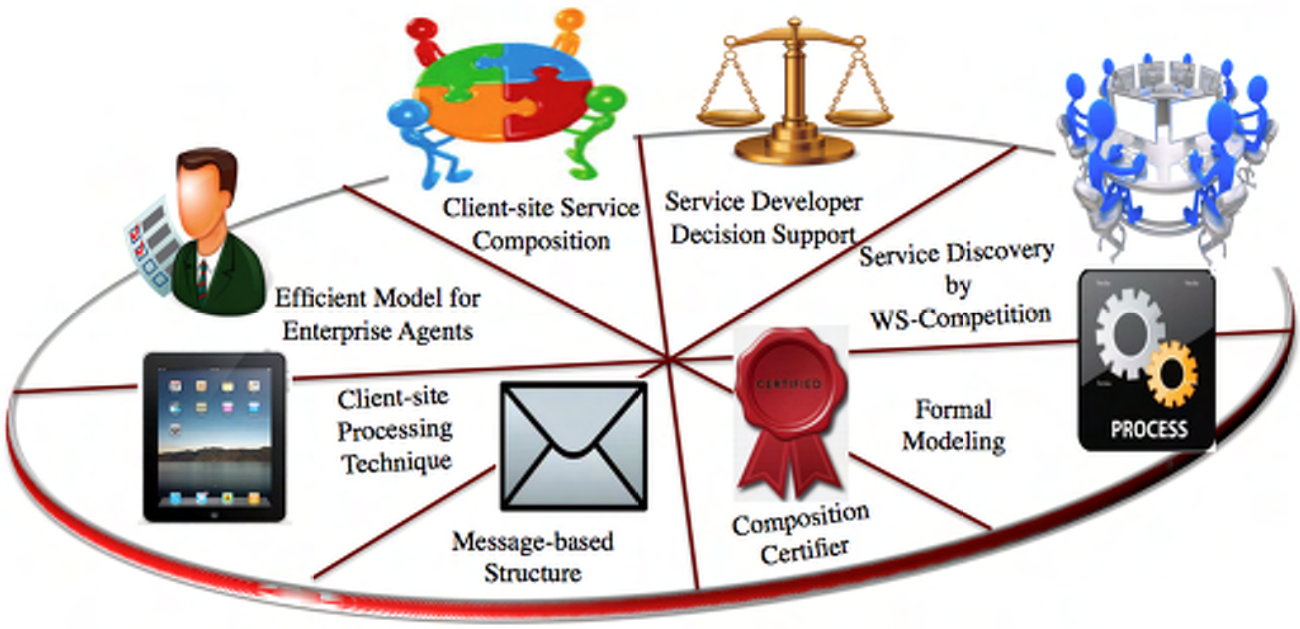


Figure 9.1. Benefits of task services in the proposed extended SOA.

cessing technologies including scripting and rich Internet applications. Although both represent compatible performance metrics, task services can be better options due to their composability and discoverability.

Task services are composable by typical data services using BPEL processes (Chapter 4). Moreover, the service representative is equipped with a service orchestrator to call different task services in a defined order to address a client-side service composition approach. Consequently, the client is not required to transfer confidential or large data to the service provider. Thus the security, privacy, and efficiency features of the enterprise systems tend to be greatly improved.

Different applications of the service representative were investigated in the healthcare domain (Chapter 5 and Appendix B). This domain was chosen for demonstration purposes, as the client data are sensitive and should normally be processed locally. The virtual nurse is a customized version of the generic service representative which can perform tasks on the client's PHR and contextual data plus applicable rules from medical guidelines or personalized treatment plans. The thesis provided several task instances in the case of diabetes and hypertension managements.

The integration of software agents and web services enable more sophisticated approaches to be proposed for some SOA related tasks. For example, candidate services customize generic service representatives to compete on their behalf (Chapter 6). This results in better service selection where the selected service provides the best service performance for the specific client's needs.

Table 9.1. Comparing different types of web services.

Type	Applications	Pros	Cons
Data Services	Server-side processing of non-sensitive client data	- Integrated service logic - Easy to assess service reliability - Maintains enterprise privacy	- Violates client privacy - Increases network traffic - Increases service response time
Task Services	Client-side processing of confidential, large volume, or real time client data	- Maintains client privacy - Reduces network traffic - Improves service response time	- Distributed service logic - Difficult to assess service reliability

Each task service includes business components which are efficiently transmitted over messages and also modeled in formal methods. This feature was used to introduce a new SOA component named the Service Composition Certifier (Chapter 7). Using a process algebraic model of the entire system, the service composition certifier verifies whether a composite service is compatible. Then it validates the service composition model by formalizing the service client's policy using a temporal logic. As a result, in a user-centric and dynamic environment, the client-defined composite web services can be verified and validated automatically.

A decision model is provided for service developers to make decisions about the best service type (either a data service or a task service) to adopt for web services. These web services are complementary and have their own applications and features (Table 9.1). Based on the experiments in this thesis, a composition of data and task services can exhibit good QoS scores if the task service initially processes the client's data and the pre-processed data are transferred to the server for more complex processing (Chapter 8). Figure 9.1 highlights some of the key features about task services presented in this thesis.

9.2 Future Work

The contributions of this work will be realized by advances in two rapidly developing technologies: cloud computing and mobile devices. In a cloud environment, computer and network resources are available through web services. However, available web services are limited, as they can not process client data locally. Similarly, mobile devices such as smart phones and tablets provide sophisticated computational powers for their clients which can not be used by typical web services. Consequently, a proposed service representative which is installed on a mobile device and can be used by different task services to perform client-side processing improves web service features. The following items represent potential future extensions to the research presented in this thesis, that could be of benefit to future developments in cloud computing and mobile device applications.

1. *Employing the generic service representative to take SOA roles.* For example, a service provider can negotiate with the service client by defining a task for the service representative. Moreover, the negotiation policies could be encoded and sent to the service representative as task knowledge. In this case, the service representative could take on a negotiator role which is closer to real cases in the business domain. In an agent based environment, service clients could have their own representatives to negotiate with the service representative (see Chapter 2).

2. *Enabling service representatives to receive their required knowledge via distributed knowledge services.* Enterprise knowledge is a valuable asset for enterprises. However, enterprises can offer this knowledge via knowledge services to be applied remotely by service representatives or other services. Knowledge management provides techniques to represent, store, transfer, and apply different types of knowledge. For example, descriptive and procedural knowledge can be managed by business rules and actions, respectively. A knowledge service could receive a knowledge request from a service client, service representative, or service provider. Then it could use a knowledge representation technique (e.g., PMML for descriptive knowledge or Dynamic-Link Library for procedural knowledge) to encode and send the requested knowledge. It would be the knowledge receiver's responsibility to use the knowledge to process the data and generate information (see Chapter 3).

3. *Extending the Virtual Remote Nursing (VRN) application to a Smart VRN.* In a smart VRN, the vNurse is asked to model the patient's behavior using data mining techniques. This behavior could then be used as part of the task knowledge. For example, vNurse could be asked to model situations where the patient experiences high blood pressure, allowing it to recommend that the patient take recommended BP medications before these situations arise. Consequently, a healthcare provider would be able to assign some simple decision making tasks to the virtual nurse (see Chapter 5).

4. *Extending the Virtual Remote Nursing (VRN) to Interactive VRN.* In the interactive VRN, a task assigned to the vNurse would be completed through the collaboration of the patient and the vNurse. For this purpose, a more sophisticated business process modeling is required which enables user interaction during business process execution (see Chapter 5).

5. *Proposing a web service engine based on the web service competition paradigm.* By addressing potential interoperability issues and developing the required protocols, the ultimate goal would be to extend the WS-Competition by introducing a web service engine that finds and ranks services based on their performance for the specific client's needs. Competition policies would be standardized and extended to support both functional and non-functional

service features. For example, decision support services which generate recommendations for service clients are good candidates to be ranked by this kind of search engine. This is because the service responses (recommendations) are comparable, and it is hard to find a decision support service which generates the best recommendations for all the client's contexts (see Chapter 6).

6. *Extending the proposed service composition certifier by adding more business properties.* Since the main target of this research is the Personal Web where a non technical service client must be able to compose web services, this work would require further research in the business domain to investigate how effectively service clients can define service composition policies. Moreover, precise syntax and semantics are required for service composition policies and a protocol is needed that effectively supports service composition certification (see Chapter 7).

7. *Discovering the best combination of data and task services representing a composite business process.* The proposed decision model in Chapter 8 evaluates each business process as either a data service or a task service. However, solving an optimization problem can divide the business process into a number of data and task services where their composition provides the best QoS parameters. Moreover, the service client could be involved in the decision making process as follows. The list of decision criteria (QoS parameters) are presented to the service clients for weight adjustments. Then the system re-calculates the decision model and produces a ranked list of different service options. This would allow the user to have more control over the results, based on different priorities (see Chapter 8).

8. *Designing a customized version of the service representative for mobile devices.* Thanks to mobile devices and their application platforms, even smart phones can become servers and provide web services. Moreover, task services require short messages to process client resources. This works well with mobile devices where a mobile App representing the service representative can be installed to perform task services and process the client's contextual information (see Appendix A).

9.3 Relevant Publications and Submissions by the Author

- **Refereed Journal Articles**

  1. M.Najafi and K.Sartipi. *Extending SOA Architecture using Generic Service Representatives.* **Journal of Service Oriented Computing and Applications**, 2010, (Published).
  2. M. Najafi, K.Sartipi, and N.Archer, *Task Services: Client-side Web Services Using Generic Service Representatives*, **IEEE Transactions on Service Computing**, 2011 (Accepted with revisions).
  3. M. Najafi, K.Sartipi, and N.Archer, *Formal Verification and Validation of composite Web Services Using Service Composition Certifier*, **Journal of Computer Networks**, Elsevier 2011 (Accepted with revisions).
  4. M. Najafi, K.Sartipi, and N.Archer, *Web Service Competition: A New Approach to Service Selection*, **International Journal of Web Intelligence and Agent Systems**, 2011 (Submitted).

- **Refereed Book Chapter**

  1. K. Sartipi, M. Najafi and R. Kazemzadeh, *Data and Mined-Knowledge Interoperability in eHealth Systems*, Invited Chapter of Book: **Data Mining in Medical and Biological Research**, 2008.

- **Referred Conference Proceedings**

  1. M. Najafi and K.Sartipi, and N.Archer, A QoS-Aware Decision Model for Web Service Development: Server-side Data Services or Client-side Task Services. In **IBM CASCON 2011**,Toronto, Canada (Accepted).
  2. M.Najafi, K.Sartipi, and N.Archer, *Data, Information, Knowledge Web Services Using Service Representatives*, In The **2011 World Congress in Computer Science, Computer Engineering, and Applied Computing**, Jul 17-12, pages 6-13, 2011, Las Vegas, Nevada, USA.
  3. M.Najafi, S.Aghtar, K.Sartipi, and N.Archer. *Virtual Remote Nursing System*, In **IEEE Consumer Communications and Networking Conference (CCNC)**, 2011, Jan 9-12, pages 13-17, Las Vegas, Nevada, USA.

4. M. Najafi and K.Sartipi, *Client-side Service Composition Using Generic Service Representative*, In **IBM CASCON 2010**, Nov 1-4, pages 238 252, Toronto, Canada, 2010.
5. M.Najafi and K.Sartipi, *A Framework for Context-Aware Services Using Service Customizer*, **IEEE International Conference On Advanced Communication Technology (ICACT 2010)**, Volume 2, pages 1339-1344, Phoenix Park, Korea, February 7-10, 2010.
6. M. Najafi and K. Sartipi. A modular Event-based Architecture for Workflow Systems. **IASTED International Conference on Software Engineering and Applications (SEA 2008)**, pages 70-76, Orlando, Florida, USA, Nov 2008.

- **Workshop**

1. M.Najafi and K.Sartipi, *Service Representative to Enhance SOA Features*, **SITCON: The CAS / NSERC Strategic Workshop in Smart Internet Technologies**, Markham, Ontario, Canada, Nov 2, 2009.

Bibliography

V Aalst. Pi calculus versus petri nets: Let us eat "humble pie" rather than further inflate the "pi hype", 2003.

R Agrawal, G Tyrone, C Johnson, and J Kiernan. Enabling the 21st century healthcare information technology revolution. *Commun. ACM*, 50(2):34–42, 2007.

F Akyildiz, W Su, Y Sankarasubramaniam, and E Cayirci. A survey on sensor networkd. *IEEE Communication Magazine*, 40(8):102–114, 2002.

E Al-Masri and Q Mahmoud. Qos-based discovery and ranking of web services. In *Computer Communications and Networks, 2007. ICCCN 2007. Proceedings of 16th International Conference on*, pages 529 –534, aug. 2007.

M Alessandrini, W Lippe, and W Nuesser. Intelligent Service System: An Agent-Based Approach for integrating Artifical Intelligence Components in SOA Landscapes. In *International Conference on Web Intelligence and Intelligent Agent Technology*, pages 496–499, Sydney, Australia, 2008.

M Anisetti, A Ardagna, and E Damiani. Toward certification of services. In *International Workshop on Business System Management and Engineering (BSME 2010)*, Malaga, Spain, June 2010.

B Atkinson and et al. Web services security (ws-security). Technical report, IBM and Verisign, 2002.

X Bai, W Dong, W Tsai, and Y Chen. Wsdl-based automatic test case generation for web services testing. In *Proceedings of the IEEE International Workshop*, pages 215–220, Washington, DC, USA, 2005. IEEE Computer Society.

M Baldauf, S Dustdar, and F Rosenberg. A survey on context-aware systems. *International Journal of Ad Hoc and Ubiquitous Computing*, 2(4):263–277, 2007.

K Ballinger, P Brittenham, A Malhotra, W Nagy, and S Pharies. *Web Services Inspection Language (WS-Inspection) 1.0.* IBM, 2001.

D Barnes, S Sankaranarayanan, and S Ganesan. Performance analysis of client/server versus agent based communication in wireless sensor networks for health applications. In *Electro/Information Technology, 2009. eit '09. IEEE International Conference on*, pages 271 –276, june 2009.

J Baumann, F Hohl, K Rothermel, M Schwehm, and M Strasser. Mole 3.0: a middleware for java-based mobile software agents. In *The International Conference on Distributed Systems Platforms and Open Distributed Processing*, pages 355–370, London, UK, 2009. Springer-Verlag.

B Benatallah, M Hacid, A Leger, C Rey, and F Toumani. On automating web services discovery. *The VLDB Journal*, 14(1):84–96, 2005. ISSN 1066-8888.

A Bertolino and A Polini. The audition framework for testing web services interoperability. In *Software Engineering and Advanced Applications, 2005. 31st EUROMICRO Conference on*, pages 134 – 142, 2005.

A Bertolino, L Frantzen, A Polini, and J Tretmans. Audition of web services for testing conformance to open specified protocols. In *Architecting Systems with Trustworthy Components*, volume 3938 of *Lecture Notes in Computer Science*, pages 1–25. Springer Berlin / Heidelberg, 2006.

L Bordeaux and SalaN. Using process algebra for web services: Early results and perspectives. In Ming-Chien Shan, Umeshwar Dayal, and Meichun Hsu, editors, *Technologies for E-Services*, volume 3324 of *Lecture Notes in Computer Science*, pages 54–68. Springer Berlin / Heidelberg, 2005.

J Bradfield and C Stirling. Modal mu-calculi. In *Handbook of Modal Logic*, pages 721–756. Elsevier, 2007.

P Braun and W Rossak. *Mobile Agents: Basic Concepts, Mobility Models, and the Tracy Toolkit.* Morgan Kaufmann Publishers Inc., San Francisco, USA, 2004.

R Brause, T Langsdorf, and M Hepp. Neural data mining for credit card fraud detection. In *ICTAI '99: Proceedings of the 11th IEEE International Conference on Tools with Artificial Intelligence*, page 103, Washington, DC, USA, 1999. IEEE Computer Society.

R Brooks. Mobile code paradigms and security issues. *Internet Computing, IEEE*, 8(3):54 – 59, May 2004.

J Cane. Performance measurements of web applications. In *SoutheastCon, 2003. Proceedings. IEEE*, pages 87 – 93, april 2003.

G Canfora and M Di Penta. Service-oriented architectures testing: A survey. In Andrea De Lucia and Filomena Ferrucci, editors, *Software Engineering*, volume 5413 of *Lecture Notes in Computer Science*, pages 78–105. Springer Berlin / Heidelberg, 2009.

H Celis, E Den Hond, and J Staessen. Self-measurement of blood pressure at home in the management of hypertension. *Clinical Medicine and Research*, 3(1):19–26, 2005.

X Chen and P Mohapatra. Performance evaluation of service differentiating internet servers. *IEEE Transactions on Computers*, 51:1368–1375, 2002.

Y Cheng, A Leon-Garcia, and l Foster. Toward an Autonomic Service Management Framework: A Holistic Vision of SOA, AON, and Autonomic Computing. *IEEE Communications Magazine*, 46(5):138–146, 2008.

E Clarke, A Biere, R Raimi, and Y Zhu. Bounded model checking using satisfiability solving. *Formal Methods in System Design*, 19:7–34, 2001.

F Curbera, F Duftler, R Khalaf, W Nagy, N Mukhi, and S Weerawarana. Unraveling the web services web: an introduction to soap, wsdl, and uddi. *Internet Computing, IEEE*, 6(2):86 –93, mar/apr 2002.

P Dadhich, K Dutta, and M Govil. Security issues in mobile agents. *International Journal of Computer Applications*, 11(4):1–7, December 2010.

E Damiani and A Mana. Toward ws-certificate. In *Proceedings of the 2009 ACM workshop on Secure web services*, SWS '09, pages 1–2, New York, NY, USA, 2009. ACM.

E Damiani, N Ioini, A Sillitti, and G Succi. Ws-certificate. In *International Congress on Services*, pages 637–644. IEEE, 2009.

S Deng, Z Wu, M Zhou, Y Li, and J Wu. Modeling service compatibility with pi-calculus for choreography. In *Conceptual Modeling - ER 2006*, volume 4215 of *Lecture Notes in Computer Science*, pages 26–39. Springer Berlin / Heidelberg, 2006.

M Dikaiakos and G Samaras. A performance analysis framework for mobile agent systems. In Tom Wagner and Omer Rana, editors, *Infrastructure for Agents, Multi-Agent Systems, and Scalable Multi-Agent Systems*, volume 1887 of *Lecture Notes in Computer Science*, pages 180–187. Springer Berlin / Heidelberg, 2001.

W Dong and H YU. Web service testing method based on fault-coverage. In *Proceedings of the 10th IEEE on International Enterprise Distributed Object Computing Conference Workshops*, pages 43–, Washington, DC, USA, 2006. IEEE Computer Society.

S Dustdar and W Schreiner. A survay on web service composition. *International Journal of Web Grid Services*, 1(1):1–30, 2005a.

S Dustdar and W Schreiner. A survey on web services composition. *International Journal of Web and Grid Services*, 1(11):1–30, 2005b.

J Everett. A decision support simulation model for the managment of an elective surgery waiting system. *Health Care Manage Science*, 5(2):89–95, 2002.

G Eysenbach. Medicine 2.0: social networking, collaboration, participation, apomediation, and openness. *Journal of Medical Internet Research*, 10(3), 2008.

H Garsden, J Basilakis, B Celler, K Huynh, and N Lovell. A home health monitoring system including intelligent reporting and alerts. pages 3151, 3155, San Francisco, CA, USA, 2004.

E Goorman and M Berg. Modelling nursing activities: Electronic patient records and their discontents. *Nursing Inquiry*, 7(1):3–9, 2000.

R Greenes. *Clinical Decision Support: The Road Ahead*. Academic Press, Inc., Orlando, USA, 2006.

T Gu, H Pung, and D Zhang. A service-oriented middleware for building context-aware services. *Journal of Network and Computer Applications*, 28(1):1 – 18, 2005.

E Guttman. Service location protocol: Automatic discovery of ip network services. *IEEE Internet Computing*, 3(4):71–80, 1999.

M Hadzic, P Wongthongtham, T Dillo, E Chang, M Hadzic, P Wongthongtham, T Dillon, and E Chang. Introduction to multi-agent systems. In *Ontology-Based Multi-Agent Systems*, volume 219 of *Studies in Computational Intelligence*, pages 15–35. Springer Berlin / Heidelberg, 2009.

S Hagemann, C Letz, and G Vossen. Web service discovery - reality check 2.0. *Next Generation Web Services Practices, International Conference on*, 0:113–118, 2007.

K Haigh, J Phelps, and C Geib. An open agent architecture for assisting elder independence. In *The Proceedings of the first international joint conference on Autonomous agents and multiagent systems*, pages 578–586, New York, NY, USA, 2002. ACM.

J Han and M Kamber. *Data mining: concepts and techniques.* Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2000.

S Haykin. *Neural Networks and Learning Machines.* Prentice Hall, 2008.

R Heckel and L Mariani. Automatic conformance testing of web services. In *Fundamental Approaches to Software Engineering*, volume 3442 of *Lecture Notes in Computer Science*, pages 34–48. Springer Berlin / Heidelberg, 2005.

R Heffner and L Fulton. Topic overview: Service oriented architecture. Technical report, Cambridge,UK, 2007.

G Holzmann, R Joshi, and R Groce. Petri nets for services modeling and composition. Technical report, Spin, 2008.

C Huebscher and J McCann. A survey of autonomic computing: degrees, models, and applications. *ACM Comput. Surv.*, 40(3):1–28, 2008.

P Hung, H Li, and J Jeng. WS-Negotiation: an overview of research issues. In *The 17th Annual Hawaii International Conference on System Sciences*, pages 10–18, Hawaii, USA, 2004.

IBM. An architectural blueprint for autonomic computing, technical report. Technical report, I.B.M. Corporation, 2003. http://www-306.ibm.com/autonomic/pdfs/ACwpFinal.pdf.

R Jafri and H Arabnia. A survey of face recognition techniques. *Journal of Information Processing Systems*, 5(2):41–68, 2009.

M Juric. *Business Process Execution Language for Web Services BPEL and BPEL4WS 2nd Edition.* Packt Publishing, 2006.

P Kakumanu, S Makrogiannis, and N Bourbakis. A survey of skin-color modeling and detection methods. *Pattern Recogn.*, 40(3):1106–1122, 2007. ISSN 0031-3203.

R Kazemzadeh and K Sartipi. Interoperability of data and knowledge in distributed health care systems. In *The IEEE International Workshop on Software Technology and Engineering Practice*, pages 230–240, Washington, USA, 2005. IEEE Computer Society.

C Keum, S Kang, I Ko, J Baik, and Y Choi. Generating test cases for web services using extended finite state machine. In *Testing of Communicating Systems*, volume 3964 of *Lecture Notes in Computer Science*, pages 103–117. Springer Berlin / Heidelberg, 2006.

A Khalili, A Badrabadi, and F Khoshalhan. A Framework for Distributed Market Place Based on Intelligent Software Agents and Semantic Web Services. In *IEEE International Congress on Services Part II*, pages 141–148, Hawaii,USA, 2008.

N Kokash. Web service discovery with implicit qos filtering. In *In Proceedings of the IBM PhD student symposium, in conjunction with ICSOC 2005*, pages 61–28666, Netherlands, 2005.

I Korhonen. Health monitoring in the home of the future. *IEEE Engineering in Medicine and Biology*, 22(3):66–73, 2003.

M Koshkina and F van Breugel. Modelling and verifying web service orchestration by means of the concurrency workbench. *SIGSOFT Softw. Eng. Notes*, 29:1–10, September 2004.

D Kourtesis, E Ramollari, D Dranidis, and I Paraskakis. Discovery and selection of certified web services through registry-based testing and verification. In *Pervasive Collaborative Networks*, volume 283 of *IFIP International Federation for Information Processing*, pages 473–482. Springer Boston, 2008.

D Krafzig, K Banke, and D Slama. *Enterprise SOA: Service Oriented Architecture Best Practices*. Prentice-Hall, 2005.

B. Kuehn. Global shortage of health workers, brain drain stress developing countries. *The Journal of the American Medical Association*, 298:1853–1855, 2007.

A Kumar, B Smith, D Pisanelli, A Gangemi, and M Stefanelli. An ontological framework for the implementation of clinical guidelines in health care organizations. *Studies in Health Technology and Informatics*, (102):95–107, 2004.

U Kuster, H Lausen, and B Konig-Ries. Evaluation of semantic service discovery—a survey and directions for future research. *Emerging Web Services Technology, Volume II*, pages 41–58, 2008.

P Lamborn and E Hansen. Memory-efficient graph search in plannig and model checking. In *The Doctoral Consortium of the 16th International Conference on Automated Plannig and Scheduling*, pages 20–34, Cumberia,UK, 2006.

D Le, S Angela Eck, and T Cao. A Survey of Web Service Discovery Systems. *International Journal of Information Technology and Web Engineering*, 2(2):65–80, 2007.

S Lee, K Choi, H Shin, and D Shin. Ag Webs: Web Services based on Intelligent Agent Platform. In *The 9th International Conference on Advanced Communication Technology*, pages 353–356, Phonex, Korea, 2007.

R Lenz and M Reichert. It support for healthcare processes - premises, challenges, perspectives. *Data and Knowledge Engineering*, 61(1):39 – 58, 2007.

S Leslie, M Hartswood, C Meurig, S McKee, R Slack, R Procter, and M Denvir. Clinical decision support software for management of chronic heart failure: Development and evaluation. *Computers in Biology and Medicine*, 36(5):495 – 506, 2006.

G Li, V Muthusamy, and H Jacobsen. A distributed service-oriented architecture for business process execution. *ACM Transation on Web*, 4(1):1–33, 2010.

B Liu, P Ray, and S Jha. Mapping distributed application sla to network qos parameters. In *Proc. of 10th International Conference on Telecommunications, IEEE ICT 2003*, volume 8, pages 1230 – 1235, 2003.

F Liu, L Zhang, Y Shi, L Lin, and B Shi. Formal analysis of compatibility of web services via ccs. In *Proceedings of the International Conference on Next Generation Web Services Practices*, pages 143–150, Washington, DC, USA, 2005. IEEE Computer Society. ISBN 0-7695-2452-4.

N Lopes and A Rybalchenko. Distributed and predictable software model checking. In *Verification, Model Checking, and Abstract Interpretation*, volume 6538 of *Lecture Notes in Computer Science*, pages 340–355. Springer Berlin / Heidelberg, 2011.

R Lu and S Sadiq. *Lecture Notes in Computer Science*, chapter A Survey of Comparative Business Process Modeling Approaches, pages 82–94. Springer Berlin/ Heidelberg, 2007.

Z Maamar, S Mostefaoui, and H Yahyaoui. Toward an agent-based and context-oriented approach for Web services composition. *IEEE Transactions on Knowledge and Data Engineering*, 17(5): 686–697, 2005.

S Mangalwede and D Rao. Performance analysis of java-based approaches to distributed computing. *International Journal of Recent Trends in Engineering*, 1:556–559, 2009.

C Mara, C Carlos, C Javier, and V Antonio. Formalizing wsbpel business processes using process algebra. *Electronic Notes in Theoretical Computer Science*, 154(1):159 – 173, 2006.

D Martin, M Paolucci, S McIlraith, M Burstein, D McDermott, D McGuinness, B Parsia, T Payne, M Sabou, M Solanki, N Srinivasan, and K Sycara. Bringing semantics to web services: The owl-s approach. In *Semantic Web Services and Web Process Composition*, volume 3387 of *Lecture Notes in Computer Science*, pages 26–42. Springer Berlin / Heidelberg, 2005.

M Mazzara and S Govoni. A case study of web services orchestration. In *Coordination Models and Languages*, volume 3454 of *Lecture Notes in Computer Science*, pages 1–16. Springer Berlin / Heidelberg, 2005.

D Menasce. Qos issues in web services. *Internet Computing, IEEE*, 6(6):72 – 75, 2002.

E Meshkova, J Riihijrvi, M Petrova, and P Mhnen. A survey on resource discovery mechanisms, peer-to-peer and service discovery frameworks. *Computer Networks*, 52(11):2097 – 2128, 2008.

R Miller, U Greiner, and E Rahm. Aw: a workflow system supporting rule-based workflow adaptation. *Data and Knowledge Engineering*, 51(2):223–256, 2004.

R Milner. *A Calculus of Communicating Systems*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1982.

F Moller and P Stevens. *The Edinburgh Concurrency Workbench user manual Version 7.1*. http://www.dcs.ed.ac.uk/home/cwb, 1997.

A Moreno and J Nealon. *Applications of Software Agent Technology in the Health Care Domain*. Birkhauser (Architectural), 2004.

T Murata. Petri nets: Properties, analysis and applications. *Proceedings of the IEEE*, 77(4):541 –580, apr 1989.

S. Naicker, J. Plange-Rhule, R. Tutt, and J Eastwood. Shortage of healthcare workers in developing countries-africa. *Ethnicity and Disease*, 19:160–164, 2009.

M Najafi and K Sartipi. A Framework for Context-Aware Services Using Service Customizer. In *The IEEE International Conference On Advanced Communication Technology*, volume 2, pages 1339–1344, Phoenix Park, Korea, 2010.

N Narayanan and S McIlraith. Simulation, verification and automated composition of web services. In *Proceedings of the 11th international conference on World Wide Web*, WWW '02, pages 77–88, New York, NY, USA, 2002. ACM.

J Ng, M Chignell, J Cordy, and Y Yesha. Overview of the smart internet. In *The Smart Internet*, volume 6400 of *Lecture Notes in Computer Science*, pages 49–56. Springer Berlin / Heidelberg, 2010.

H Nwana. Software Agents:An Overview. *Knowledge Engineering Review*, 11(3):205–244, 1996.

Y Ofuji, A Morimoto, S Abeta, and M Sawahashi. Comparison of packet scheduling algorithms focusing on user throughput in high speed downlink packet access. In *Personal, Indoor and Mobile Radio Communications, 2002. The 13th IEEE International Symposium on*, volume 3, pages 1462 – 1466, 2002.

M. Orr. Introduction to radial basis function networks. Technical report, Institute for Adaptive and Neural Computation, Edinburgh University, 1996.

C Otto, A Milenkovis, C Sanders, and E Jovanov. System architecture of a warieless body area sensor network for ubiquitous health monitoring. *ACM Trans. Program. Lang. Syst.Journal of Mobile Multimedia*, 1(4):307–326, January 2006.

C Peltz. Web services orchestration and choreography. *Computer*, 36(10):46–52, 2003.

J. Peterson. Petri nets. *ACM Computer Survey*, 9:223–252, September 1977.

V Pham and A Karmouch. Mobile software agents: an overview. *Communications Magazine, IEEE*, 36(7):26–37, 1998a.

V Pham and A Karmouch. Mobile software agents: An overview. *IEEE Communications Magazine*, 36:26–37, 1998b.

H Raffelt, T Margaria, B Steffen, and M Merten. Hybrid test of web applications with webtest. In *Proceedings of the 2008 workshop on Testing, analysis, and verification of web services and applications*, pages 1–7, New York, NY, USA, 2008. ACM.

S Ran. A model for web services discovery with qos. *SIGecom Exch.*, 4:1–10, 2003.

S Raspl. PMML Version 3.0 - Overview and Status. In *The ACM Workshop on Data Mining Standards, Services and Platforms*, pages 18–22, Philadelphia, USA, 2004.

G Reese. *Cloud Application Architectures: Building Applications and Infrastructure in the Cloud*. O'Reilly Media, Inc., 2009.

L Richardson and S Ruby. *RESTFul Web Services*. OReilly, 2007.

M Rouached, O Perrin, and C Godart. Towards formal verification of web service composition. In *Business Process Management*, volume 4102 of *Lecture Notes in Computer Science*, pages 257–273. Springer Berlin / Heidelberg, 2006.

T Saaty. Decision making with the analytic hierarchy process. *International Journal of Services Sciences*, 1:83–98, 2008.

H Sajedi, M Najafi, and S Kasaei. A boosted skin detection method based on pixel and block information. pages 146 –151, sep. 2007.

D Sangiorgi and D Walker. *PI-Calculus: A Theory of Mobile Processes*. Cambridge University Press, New York, NY, USA, 2001.

D Sangiorgi and D Walker. *The Pi-Calculus: A Theory of Mobile Processes*. Cambridge University, 2003.

R Schwarzkopf, M Mathes, S Heinzl, B Freisleben, and H Dohmann. Java rmi versus .net remoting architectural comparison and performance evaluation. *International Conference on Networking*, 0:398–407, 2008.

Z Shen and J Su. Web service discovery based on behavior signatures. In *SCC '05: Proceedings of the 2005 IEEE International Conference on Services Computing*, pages 279–286, Washington, DC, USA, 2005. IEEE Computer Society.

A Sinha and A Paradkar. Model-based functional conformance testing of web services operating on persistent data. In *Proceedings of the 2006 workshop on Testing, analysis, and verification of web services and applications*, TAV-WEB '06, pages 17–22, New York, NY, USA, 2006. ACM.

P Soja and G Paliwoda-Pekosz. What are real problems in enterprise system. *Industrial Management and Data Systems*, 109(5):610–627, 2009.

U Stern and D Dill. Parallelization the mur verifier. *Formal Methods in System Design*, 18: 117–129, March 2001.

C Stirling. *Modal and temporal properties of processes*. Springer-Verlag New York, Inc., New York, NY, USA, 2001.

S Su, S Yu, and H Guo. Dynamic substitutability analysis of web service composition via extended pi-calculus. In *Embedded and Ubiquitous Computing, 2008. EUC '08. IEEE/IFIP International Conference on*, volume 1, pages 447 –452, 2008.

K Sycara, M Paolucci, J Soundry, and N Srinivasan. Dynamic discovery and coordination of agent-based semantic Web service. *IEEE Internet Computing*, 8(3):66–73, 2004.

A Tarhini, H Fouchal, and N Mansour. A simple approach for testing web service based applications. In *Innovative Internet Community Systems*, volume 3908 of *Lecture Notes in Computer Science*, pages 134–146. Springer Berlin / Heidelberg, 2006.

A Tarski. A lattice-theoretical fixpoint theorem and its applications. *Pacific J.Math*, (5):285–309, 1995.

W Tsai, R Paul, W Song, and Z Cao. Coyote: An xml-based framework for web services testing. *High-Assurance Systems Engineering, IEEE International Symposium on*, 0:173, 2002.

E Turban, R Sharda, and D Delen. *Decision Support and Business Intelligence Systems*. Prentice Hall Press, Upper Saddle River, NJ, USA, 9th edition, 2010.

Y Wang, X Bai, J Li, and R Huang. Ontology-based test case generation for testing web services. *International Symposium on Autonomous Decentralized Systems*, 0:43–50, 2007.

G Weiss. *Multiagent Systems: A modern approach to distributed artificial intelligence*. MIT Press, 1999.

D Wong, N Paciorek, T Walsh, J DiCelie, M Young, and B Peet. Concordia: An infrastructure for collaborating mobile agents. In *MA '97: Proceedings of the First International Workshop on Mobile Agents*, pages 86–97, London, UK, 1997. Springer-Verlag.

D Wong, N Paciorek, and D Moore. Java-based mobile agents. *Commun. ACM*, 42(3):92–ff., 1999a.

D Wong, N Paciorek, and D Moore. Java-based mobile agents. *Commun. ACM*, 42(3):92–104., 1999b.

M Wooldridge and N Jennings. Intelligent agents: Theory and practice. *Knowledge Engineering Review*, 10(2):115–152, 1995.

R Wootton, J Craig, and V Patterson. *Introduction to telemedicine*. Royal Society of Medicine, 2006.

L Xiang. A Multi-Agent-Based Service-Oriented Architecture for Inter-Enterprise Cooperation System. In *Second International Conference on Digital Telecommunications*, pages 22–32, Silicon Vally, USA, 2007.

B Xu, X Yang, Y Shen, L Shanping, and A Ma. A role-based SOA architecture for community support systems. In *International Symposium on Collaborative Technologies and Systems*, pages 408–415, Irvine,USA, 2008.

Y Yamato, H Ohnishi, and H Sunaga. Study of Service Processing Agent for Context-Aware Service Coordination. In *IEEE International Conference on Service Computing*, pages 275–282, Hawaii,USA, 2008.

R. Yonglin, R. Pazzi, and A. Boukerche. Monitoring patients via a secure and mobile healthcare system. *Wireless Communications, IEEE*, 17(1):59 –65, february 2010.

T Yu, Y Zhang, and K Lin. Efficient algorithms for web services selection with end-to-end qos constraints. *ACM Trans. Web*, 1, May 2007.

M Yu-jie, C Jian, S Shen-sheng, and J Zand. Interactive web service choice-making based on extended qos model. In *Proceedings of the The Fifth International Conference on Computer and Information Technology*, CIT '05, pages 1130–1134, Washington, DC, USA, 2005. IEEE Computer Society. ISBN 0-7695-2432-X.

L Zeng, H Lei, and H Chang. Monitoring the qos for web services. In Bernd Krmer, Kwei-Jay Lin, and Priya Narasimhan, editors, *Service-Oriented Computing*, volume 4749 of *Lecture Notes in Computer Science*, pages 132–144. Springer Berlin / Heidelberg, 2007.

D Zhovtobryukh. Petri nets for services modeling and composition. Technical report, University of Jyvaskyla, 2007.

Appendix A

A Prototype System

To evaluate the effectiveness and feasibility of the extended SOA model, a prototype system of the proposed architecture was developed including service representative, data and task service providers, and service client. This prototype, *SR Version 1.1* (SR v1.1), was implemented using J2EE 1.5 technologies and an Apache Tomcat 6.0 application server. The completed prototype was divided into two Java packages, as follows.

- **TaskService Package**. This package includes APIs which can be imported and used by service developers to develop task services. The service developer designs and implements a task service by defining its task components, as follows: (i) task model is defined by a Drools ruleflow; (ii) task knowledge is defined by Drools rules and actions where a rule is defined in either statement or tabular format and an action is defined by either a separate Java procedure or as part of a rule definition; and (iii) task data, which are Java beans. Then the task model is converted to XML format; the rules are encoded by *PMML* standard version 3 (Raspl [2004]), and the Java functions and beans are serialized to form the task message to be sent to the service representative.

- **ServiceRepresentative Package**. This package includes APIs which can be imported and called by the service client to generate the service representative and communication channel. The service representative, included in *SR v1.1*, has a built-in Drools rule engine and can receive and apply knowledge sentences that are compatible with PMML V3 Raspl [2004]. Moreover, the communication channel is implemented by an array of pointers to client data stored in a MySQL database. To invoke a task service, the service client uses these APIs to create one instance of the service representative and its communication channel. Next, the client application supplies the client data (local service parameters) to the communication channel, based on the service description published in the service registry. Finally, the

client invokes the task service from the service provider and blocks itself to receive the local service responses from the service representative through the communication channel. The following code represents how a task service can be invoked at the client side, using the Java APIs provided.

```
ServiceRepresentative SR = new ServiceRepresentative();
CommunicationChannel CC = new CommunicationChannel();
TaskMessage taskMsg = new TaskMessage();
CC.Add (localServiceParameters);
taskMsg = ServiceStub.TaskService(remoteServiceParameters);
SR.ExecuteTask(taskMsg);
CC.Get (localServiceResponses);
```

The service representative business rule engine (Drools) will be introduced in the following section. The design and implementation details of the implemented prototype are also described.

A.1 Introduction to Drools

Drools is a java-based open source rule engine that works with a forward chaining strategy. The Drools inference engine matches fact (i.e., data) against production rules to infer conclusions, which result in actions. The production rule is a two-part structure using First Order Logic for reasoning over knowledge representations.

```
when
    < conditions >
then
    < actions >;
```

The rules can be categorized into different groups to be evaluated at the same time. Rules are stored in the *production memory* and the facts that the *inference engine* matches against are kept in *working memory*. A system may result in several rules being true for the same fact; these rules are said to be in conflict. The *agenda* manages the execution order of these conflicting rules using a pre-defined strategy. The Drools architecture is displayed in Figure A.1.

In addition to the inference engine, drools has a workflow or process engine that allows integration of processes and rules. A process describes the order in which a series of steps need to be executed, using a flow chart that is called a rule flow. For example, Figure A.2 shows a process where Task1 and Task2 need to be executed in parallel. After completion of both, Task3 will be executed.

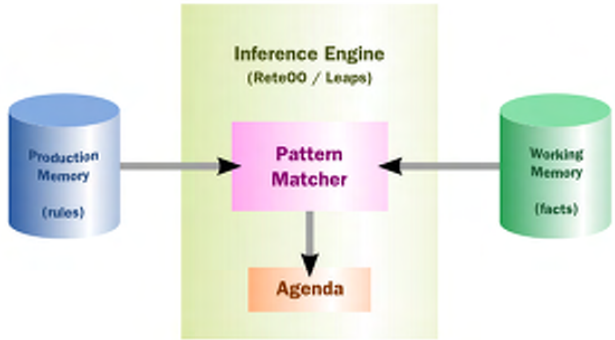


Figure A.1. High-level view of the drools rule engine [1].

Figure A.2. A rule flow example in drools.

SR version 1.1 uses drool's rule flow, rules, and actions as its business process model, business rules and actions, accordingly. Drools provides Java APIs to design rule flows graphically (e.g., Figure A.3) and integrates them in the task layer of a task service provider. A business rule can be defined in either statement or tabular format (e.g., Figure A.4) that will provide tools to modify the business rules and actions in the task knowledge component of the proposed task service provider.

While the service representative uses drools APIs to run a business process and apply the business rules, the service provider uses the corresponding APIs to define business processes, rules, and actions. The following sections describe how the developed prototype uses the Drools for these two purposes.

A.2 Architecture

The prototype's package diagram, shown in Figure A.5, indicates that UML packages represent the system architecture of the prototype. This architecture shows communication between a service provider and a server client entity. There are two types of packages: developer (white) and system (grey). Below are the functionalities of the system packages.

[1] $http://docs.jboss.org/drools/release/5.2.0.CR1/drools-expert-docs/html/ch01.html$

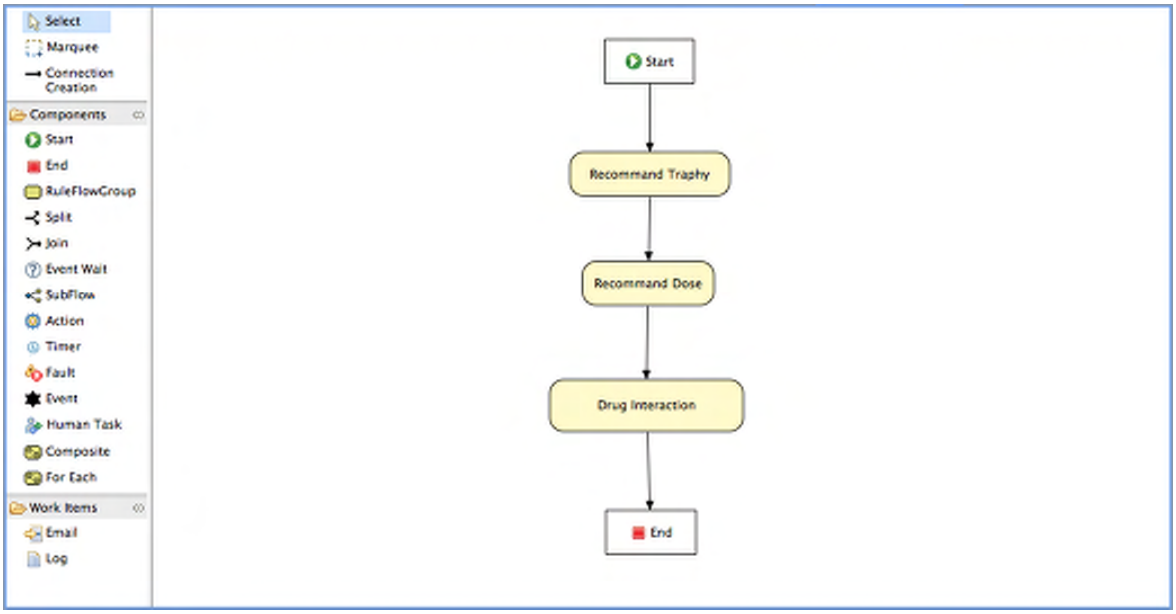


Figure A.3. Drools Eclipse plug in to design a rule flow. This process shows a medical guideline consisting of three steps.

- *Stub* contains classes that enable communication between a service endpoint and a service client. The stub class is responsible for converting a request from a JAX-RPC service client to a SOAP message and sending it across to the service endpoint using the specified protocol. It also converts the response from the service endpoint, which it receives in the form of a SOAP message, to the format required by the client.

- *JAX-RPC* (Java APIs for XML-based Remote Procedure Call) helps with web service interoperability and accessibility by providing Java APIs that Java applications use to develop and access web services.

- *Drools* defines the required Java APIs to define and work with business rules, actions, and rule flows in the Drools environment.

The author designed and implemented the developer packages, based on the specifications defined in Chapter 3 of the thesis. The Service Representative package includes the generic service representative class and its communication channel class that can be imported and instantiated in any service client application. To develop a web service, a service provider needs to modify the Service package and a service client provider adjusts the UI and Client Application packages.

```
rule "Recommend Dose One"
    ruleflow-group "dose"
    when
       p : Patient( age < 15 )
       v : Visit( medication == "Amoxicillin" )

    then
       v.setDose("Not Known Yet");
 end
rule "Recommend Dose Two"
    ruleflow-group "dose"
    when
       p : Patient( age > 60 )
       v : Visit( medication == "Amoxicillin" || medication == "Cefuroxime" || medication == "Levofloxacin" )

    then
       v.setDose("Not Known Yet");
 end
rule "Recommend Dose Three"
    ruleflow-group "dose"
    when
       p : Patient( age > 15 , age < 60 )
       v : Visit( medication == "Amoxicillin" || medication == "Cefuroxime" || medication == "Levofloxacin" )

    then
       v.setDose("500mg every 24h hours for 14 days");
 end
rule "Recommend Dose Four"
    ruleflow-group "dose"
    when
       p : Patient( age > 15 , age < 60 , creatinineLevel > "1.4" , creatinineClearance < 50 )
       v : Visit( medication == "Amoxicillin" || medication == "Cefuroxime" || medication == "Levofloxacin" )

    then
       v.setDose("250mg every 24h hours for 14 days");
 end
```

Rules void recommendDose(Patient patient, Visit visit)					
C1	C2	C3	C4	C5	A1
medication equals(visit.medication)	patient.age > minAge	patient.age <= maxAge	patient.creatinineLevel > clMin	creatinineClearance(patient) < ccrMax	visit.dose = dose
String medication	int minAge	int maxAge	double clMin	double ccrMax	String dose
If medication is	If patient is older than	If patient is younger than	If patient's Creatinine Level is more than	If patient's Creatinine Clearance is less than	Recommended Dose
Amoxicillin		15			Not Known Yet
Amoxicillin	60				Not Known Yet
Amoxicillin	15	60			500mg every 24h hours for 14 days
Amoxicillin	15	60	1.4	50	250mg every 24h hours for 14 days

Figure A.4. Statement (top) and tabular (bottom) format for defining business rules in drools. The defined rules are used in the recommendDose step of the specified medical guideline task in Figure A.3.

A.3 Design Model

In this section, the classes in the *SR version 1.1* packages are specified by class diagrams.

A.3.1 Service Client Design Model

The UML class diagram shown in Figure A.6 represents the classes involved at the client side. ClientApp, CommunicationChannel, and ServiceRepresentative classes implement the proposed client application, communication channel, and service representative specified in Chapter 3. Other classes are described below.

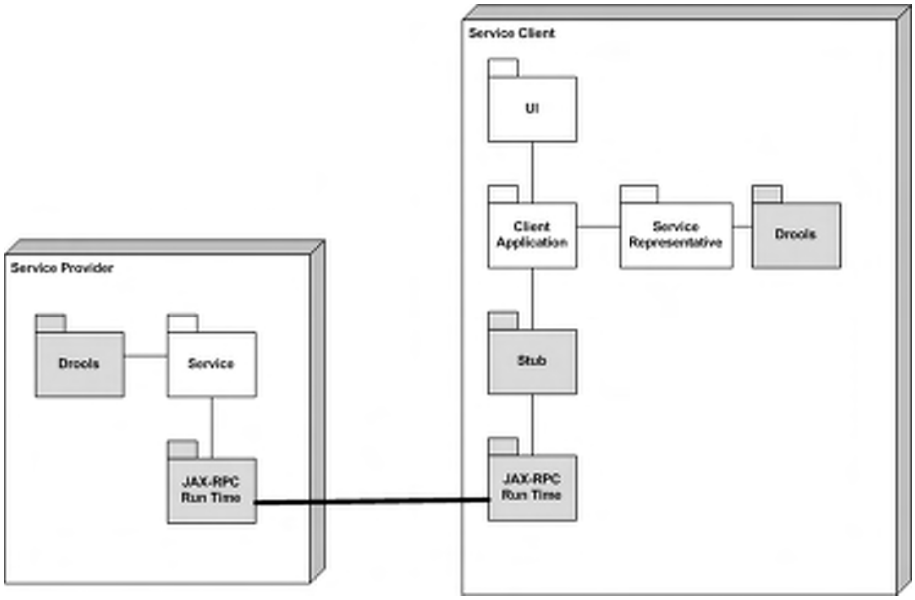


Figure A.5. Package diagrams for the prototype

- *ClientBusinessObject* is in charge of creating a Java bean from a business object that is stored as a record in a rational database. Moreover, it generates a pointer to this object to be put in the communication channel. In this prototype, each business object (either client or server) is modeled by a Java bean that is a serializable Java object and allows access to properties using getter and setter methods.
- *TaskManager* has methods to create and execute a task, while other task provisioning functionality is to be added to the future versions of SR.
- *BusinessRuleEngine* has methods to work with a business rule engine (i.e., Drools in SR version 1.1) as follows:
  1. create a session and load the internal knowledge base with the business rules and actions;
  2. insert the business objects into the session;
  3. create a business process based on the given rule flow;
  4. fire the business rules and their corresponding actions.

A.3.2 Service Provider Design Model

The UML class diagram shown in Figure A.7 represents classes used at the server side. Task Model, Task Knowledge, and Task Data implement the specified task components of a task message which have been specified in Chapter 3 of the thesis. Other classes are described as follows.

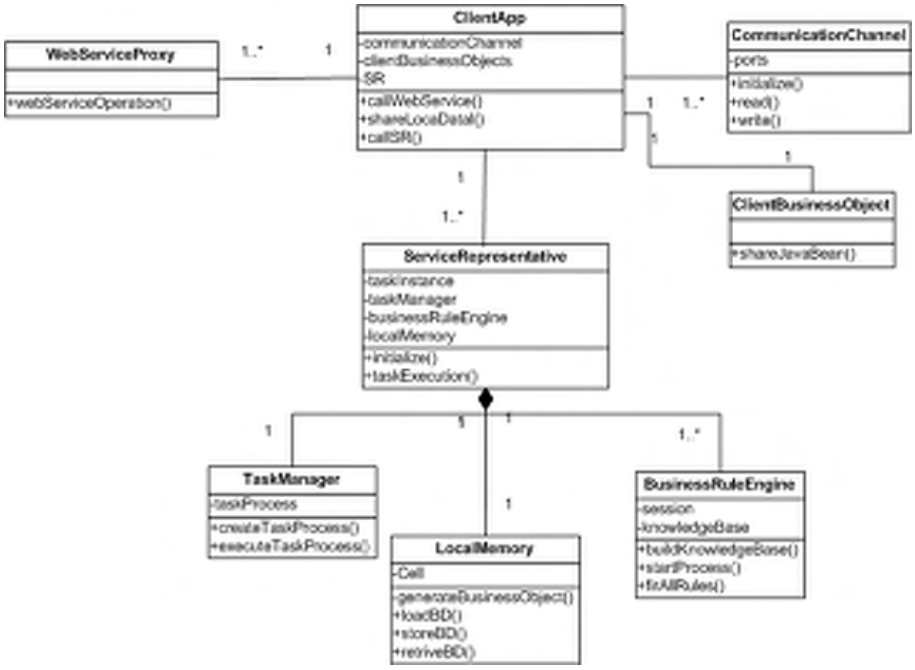


Figure A.6. Class diagrams showing the classes in the client

- *Drools* provides graphical tools to design business process models, business rules and actions. This class has methods that serialize business process models, rules and actions so they can be sent through messages (called marshalling).
- *ServerBusinessObject* is in charge of creating Java beans from the business objects stored in a rational database. Moreover, it serializes the Java beans generated to be sent to the client side.

A.4 Interactions

In this section,the interactions involved between classes in each entity are described using sequence diagrams. It should be mentioned here that minor utility functions are dropped in these diagrams for the purpose of clarity.

A.4.1 Sequence Diagram for the Service Client

The UML sequence diagram represented in Figure A.8 and Table A.1 shows the interaction between the classes involved at the client side.

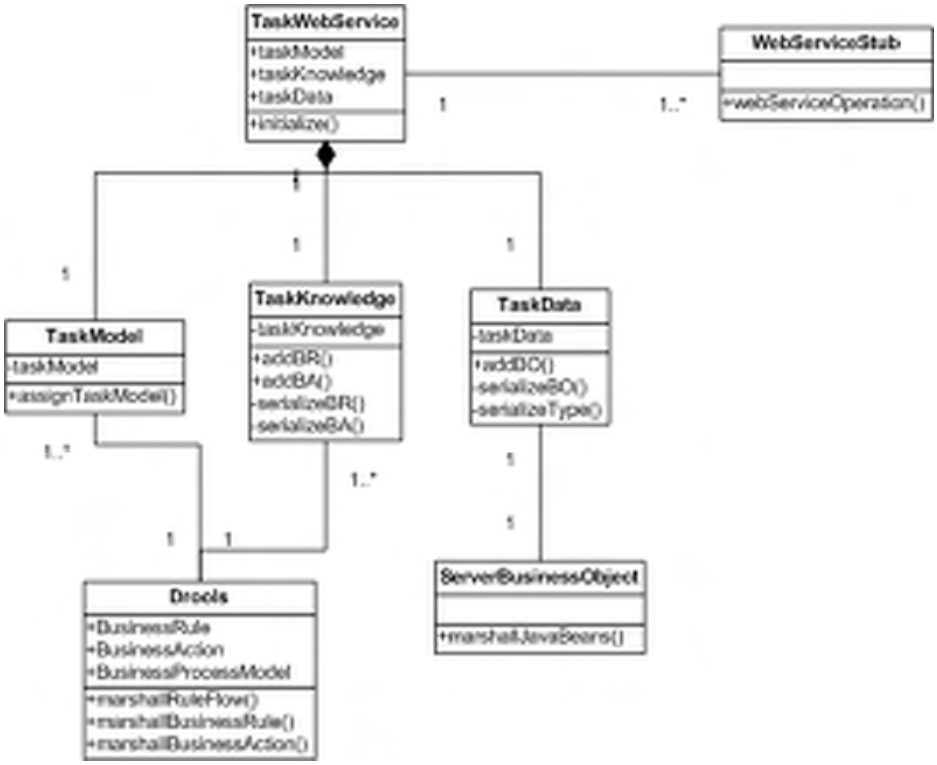


Figure A.7. Class diagrams showing the classes in the service provider

A.4.2 Sequence Diagram for the Service Provider

The UML sequence diagram represented in Figure A.9 and Table A.2 shows the interaction between the classes involved at the server side.

A.5 Conclusion

A customized business process engine for the proposed service representative may result in a more efficient software agent (in terms of execution time and size of the Java package), but Drools has been in use for some time, it is therefore more trustable than a new process engine that could be specifically designed for the service representative. Moreover, the developed SR is modular, which means that Drools can be replaced simply by another business rules engine. The case studies that are specified in this thesis use SR version 1.0.

Using these Java packages, several tools can be developed to analyze different aspects of task services. For example, Figure A.10 (bottom) illustrates a snapshot of the *Service Representative Manager v1.0* executing a sample composite task service. This tool monitors different phases of composite task execution where the service client can view the content of each port within the communication channel during task execution. Similarly, Figure A.10 (top) represents a

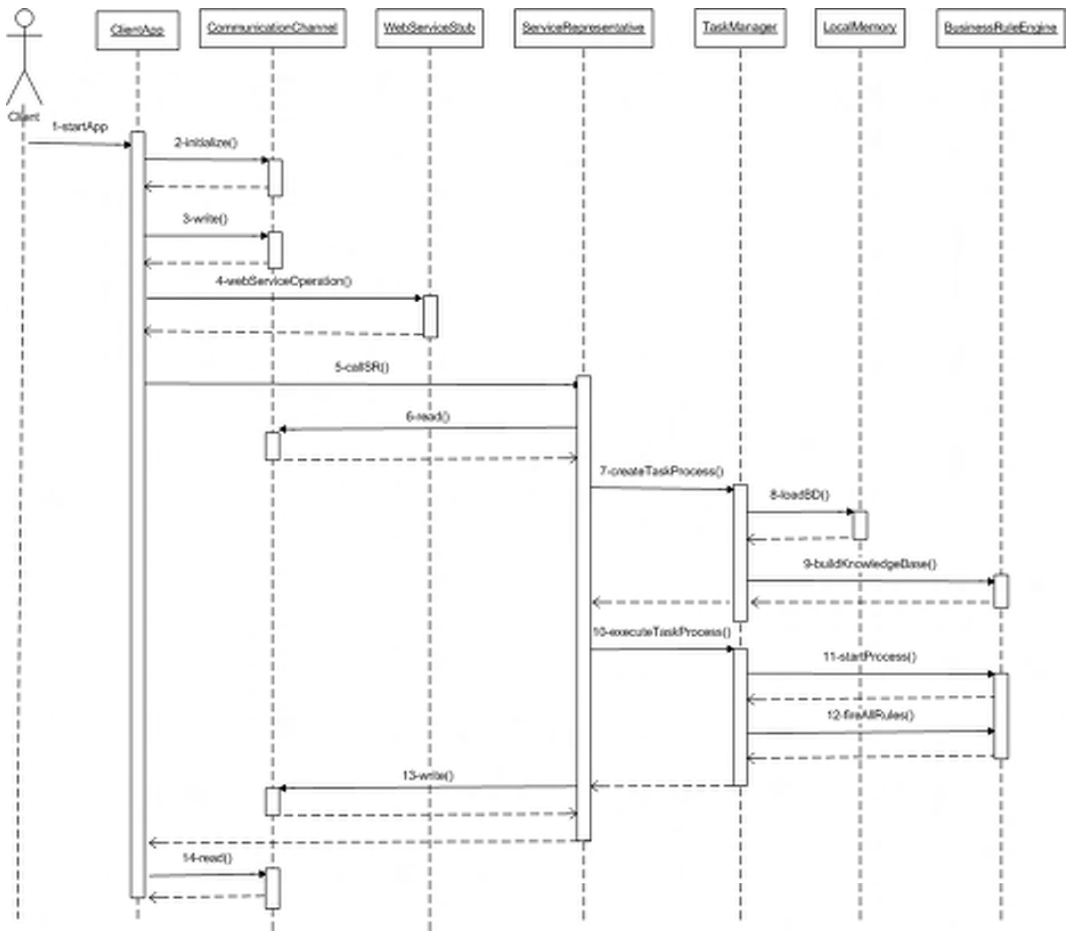


Figure A.8. Sequence diagram showing the interactions between classes in the service client

snapshot of the *Task Service Developer v1.0* which enables developing a task service graphically using Drools APIs and widgets. Finally, the *SR Version 1.1* Java packages are accessible via http://www.cas.mcmaster.ca/ najafm/codes.

Table A.1. Message description for the service client sequence diagram

Message	Description
1	The user starts an application on the client machine by running the client application.
2 - 3	The client application calls ClientBusinessObjects to create Java beans business objects. Then, the client application creates one instance of the communication channel and initializes it by putting the generated java beans into its ports based on the web service communication channel schema published on the service registry.
4	The client application calls the web service through the proxy object. The service proxy receives the service response and forwards it to the client application.
5	The client application generates a new instance of the service representative. Moreover, the service representative is initialized by sending the communication channel schema and the service response message.
6	The service representative reads the communication channel to receive the client side business objects.
7-9	The service representative splits the service response message into the task components. Business process, rules, and actions are written into the files that can be used by the rule engine. Moreover, the service representative unmarshalles the server-side business objects that are received by the data segment.
10-12	The task manager executes the task by building the internal knowledge base containing business rules and actions, inserting business objects, starting the process, and firing the business rules.
13	The service representative writes the task result into the communication channel.
14	The service client inputs the task results as the local service parameters from the communication channel.

Table A.2. Message description for the service provider sequence diagram

Message	Description
1	The stub sends a request message to the service provider to call a task service. The service initializes the response message with three empty components: task model, knowledge, and data.
2	The web service calls the task model component to specify the task by assigning a business process model. This component reads a rule flow file and marshalls it into the task model segment.
3-6	The web service calls the task knowledge component to provide the required business rules and actions for the specified task. This component reads the corresponding business rules and actions from the files and marshalls them into the knowledge segment.
7-9	The web service calls the task data component to provide the required server side business data for the specified task. This component calls the ServerBusinessObject to marshalls the Java beans into the task segment. The response message is back to the Stub object to be sent to the client application.

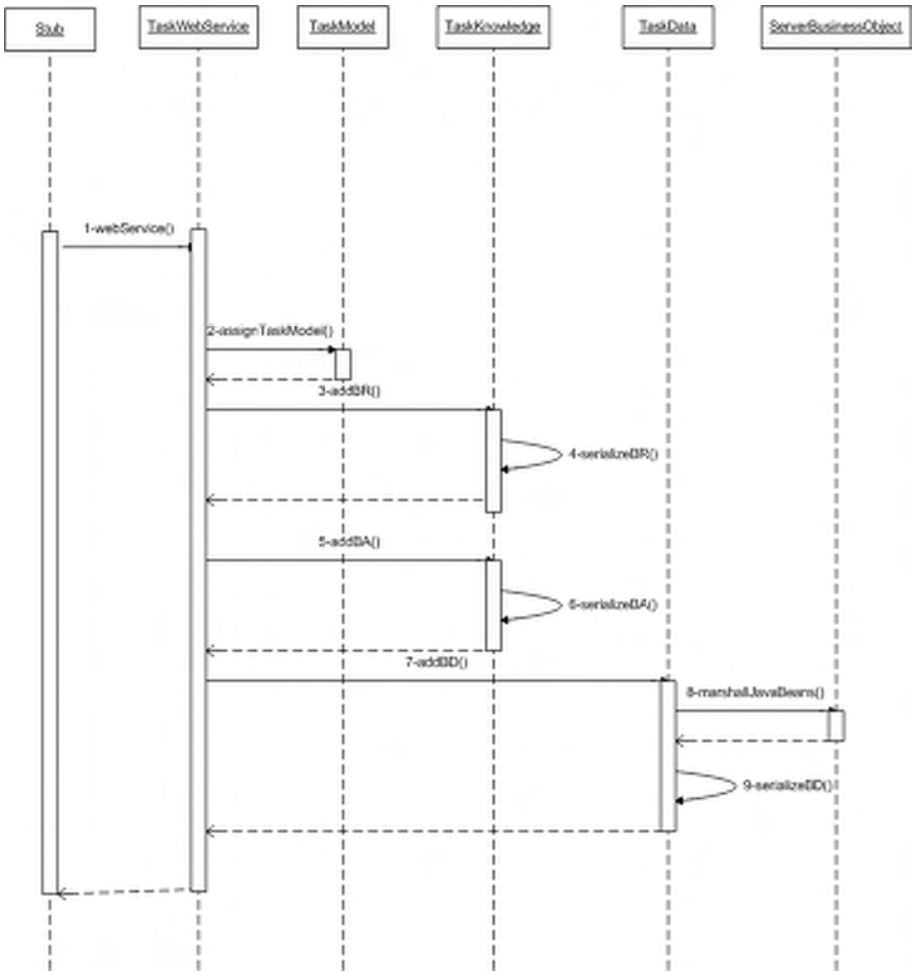


Figure A.9. Sequence diagram showing the interactions between classes at the server side

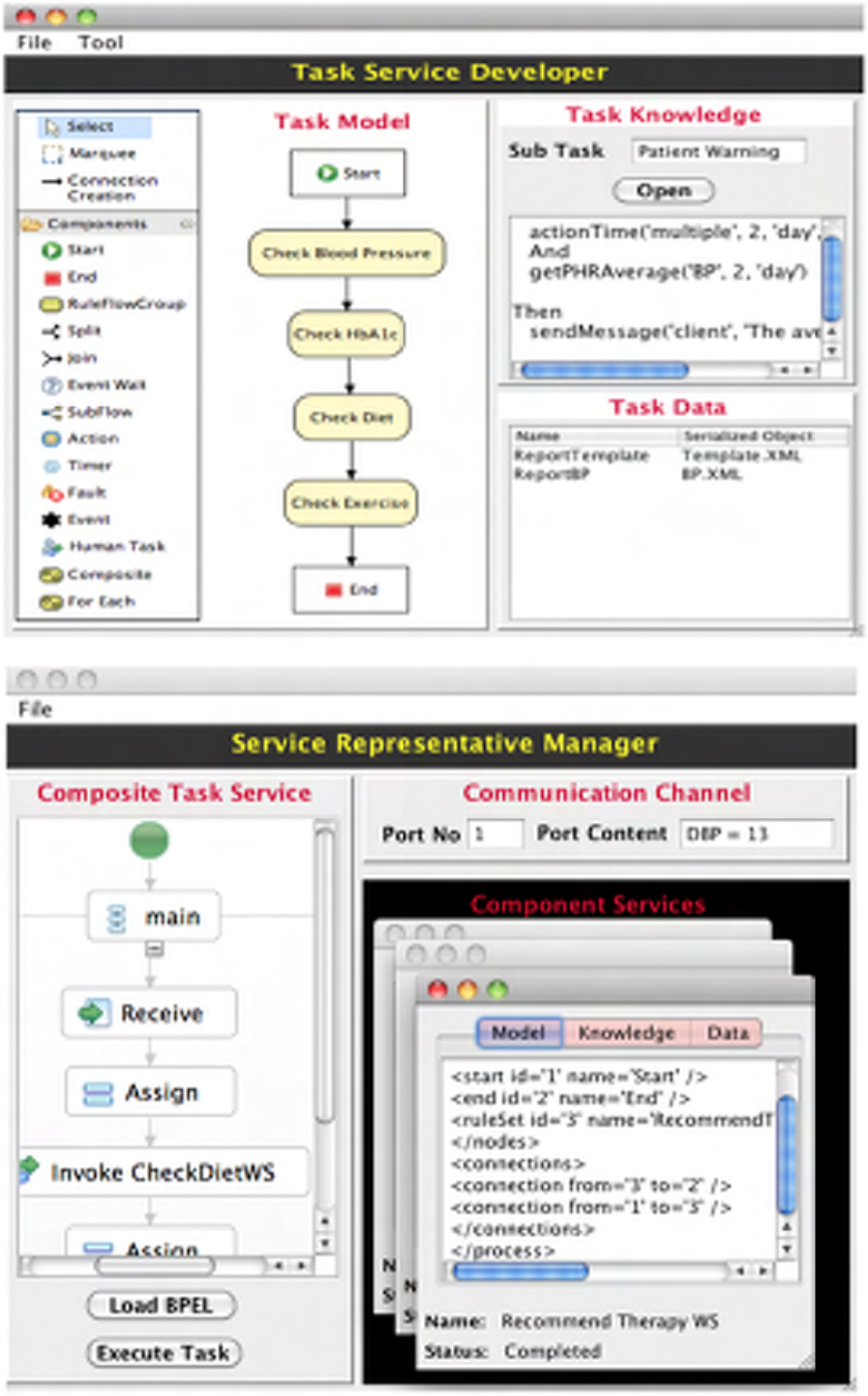


Figure A.10. Snapshots of the *Task Service Developer v1.0* **(top) and the** *Service Representative Manager v1.0* **(bottom).**

Appendix B

Hypertension Management Using Virtual Remote Nursing

Hypertension or high blood pressure is a cardiac chronic medical condition in which the blood pressure is elevated. Hypertension is one of the risk factors for stroke, heart failure, and is a leading cause of chronic kidney failure. Hypertension management includes three main phases: diagnosis, evaluation, and long-term treatment, including periodic re-evaluations and alterations in treatment as needed over time.

A physician takes several blood pressure readings during separate appointments before diagnosing a patient with high blood pressure. This is because blood pressure normally varies throughout the day, even during visits to the doctor, a condition called white-coat hypertension. A doctor may ask patients to record their blood pressure at home and at work to provide additional information. Dietary and lifestyle changes improve blood pressure controls and decrease the risk of associated health complications. However, a patient with hypertension needs to take high blood pressure medications. It is the doctor's task to decide on the best treatment to be given to a patient. However, treatments need to be evaluated for safely and efficacy on a case by case basis. Consequently, patients with hypertension need to be examined by their physicians regularly, which increases their time spent waiting for service as well as healthcare costs.

The Virtual Remote Nursing (VRN) system proposed in Chapter 5 of the thesis is a customized version of the service representative which is installed on a patient's mobile device as a mobile App. This App has access to the patient's healthcare and contextual information through online PHR systems and device sensors, respectively. The virtual nurse is assigned different tasks from healthcare professionals and provides reports/recommendations/warnings for both the patient and healthcare professionals. In this chapter, we design several VRN tasks that correspond to several tasks in hypertension management. These tasks are provided for physicians so they can assign

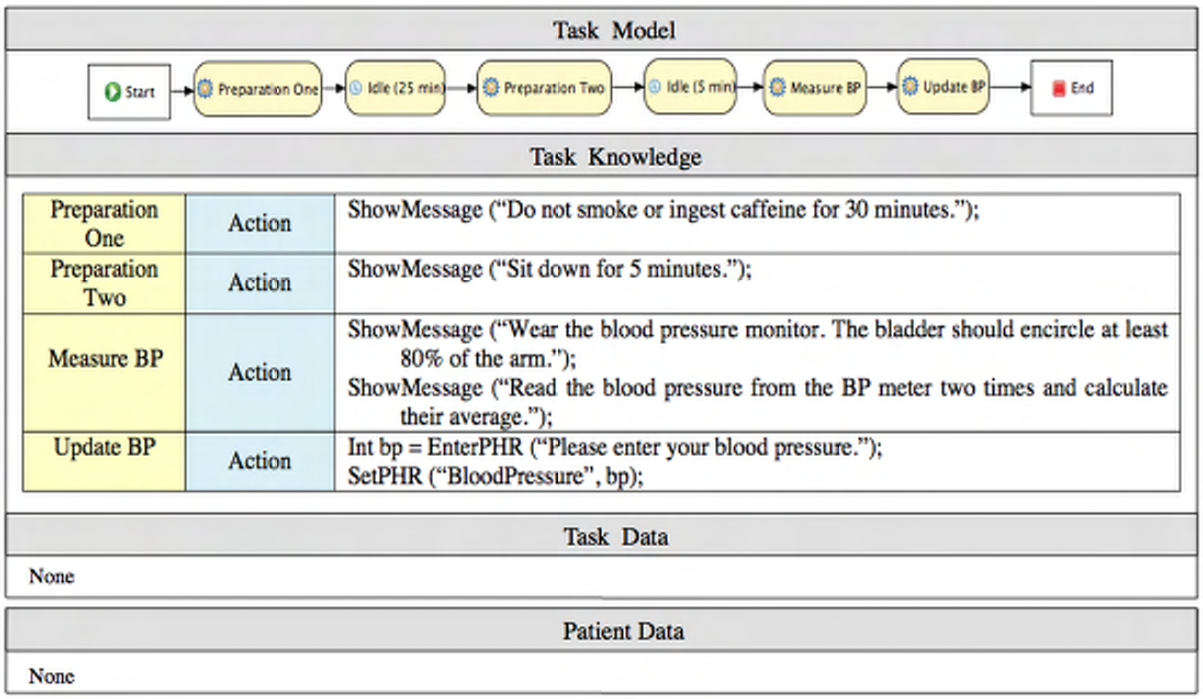


Figure B.1. *Check BP Task*: **vNurse guides a patient to measure blood pressure at home.**

them to the virtual nurse easily to support hypertension diagnosis, treatment, and evaluation. This eliminates unnecessary patient visits and it supports the physician's decision making process. Refereed medical guidelines can be extracted from different sources such as Celis et al. [2005] and various online sources[1]. It must be noted here that, during this study, we aim to show the feasibility and potential applications of the proposed VRN system. The correctness and efficiency of the medical guidelines provided, and the VRN system application for this specific application are out of scope of this thesis and need to be evaluated separately.

B.1 Blood Pressure Monitoring

Self-measurement of blood pressure at home makes it possible to obtain multiple readings and avoid the white-coat syndrome. Furthermore, this may also reduce the number of required visits for the diagnosis and treatment of hypertension. However, several steps ensure that the measured blood pressure truly represents the patient's blood pressure:

1. Patients should not smoke or ingest caffeine for 30 minutes prior to blood pressure measurement.

[1]For example, http://www.healthcommunities.com/high-blood-pressure/

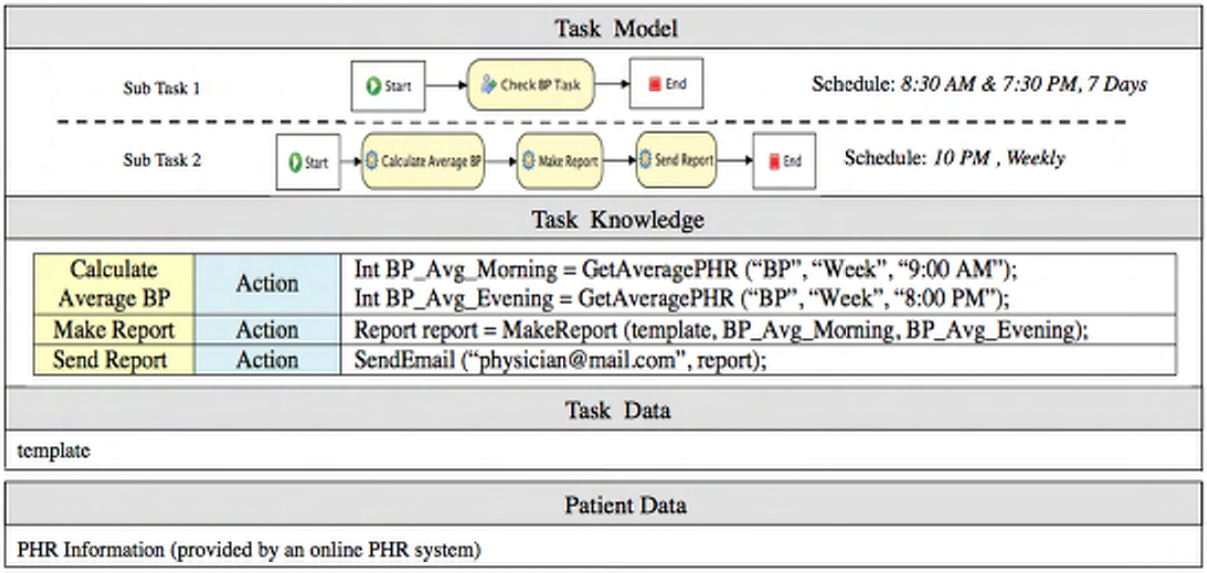


Figure B.2. *Diagnosis BP Task*: **vNurse provides a report to the physician, including the patient's average blood pressure over a week.**

2. Patients should sit, with their arms supported at heart level, for at least 5 minutes before blood pressure is measured.

3. The bladder of the blood pressure cuff should encircle at least 80% of the arm.

4. Two or more readings should be taken at least 2 minutes apart.

The task corresponding to these guidelines is displayed in Figure B.1. Using this task, the vNurse guides the patient in correctly measuring his/her blood pressure at the scheduled times. Each time, the vNurse updates the online PHR system with the current BP value. This task can also be used as a subtask for other tasks.

For the diagnosis of hypertension, multiple readings must be taken at various times throughout the waking hours of the patient. Experts have not yet reached a general consensus about a standard protocol (how many measurements and on how many days) to follow to measure their patient's BP at home. Therefore, a physician needs to choose which approach to use. Figure B.2 represents a task that includes checking blood pressure two times a day for seven days. Similarly, a physician can define a different approach by changing the schedule of the first sub-task to six times a day for only two days. Based on the blood pressure information captured, the virtual nurse generates a report according to the template sent by the physician. Then, the virtual nurse e-mails the report to the physician.

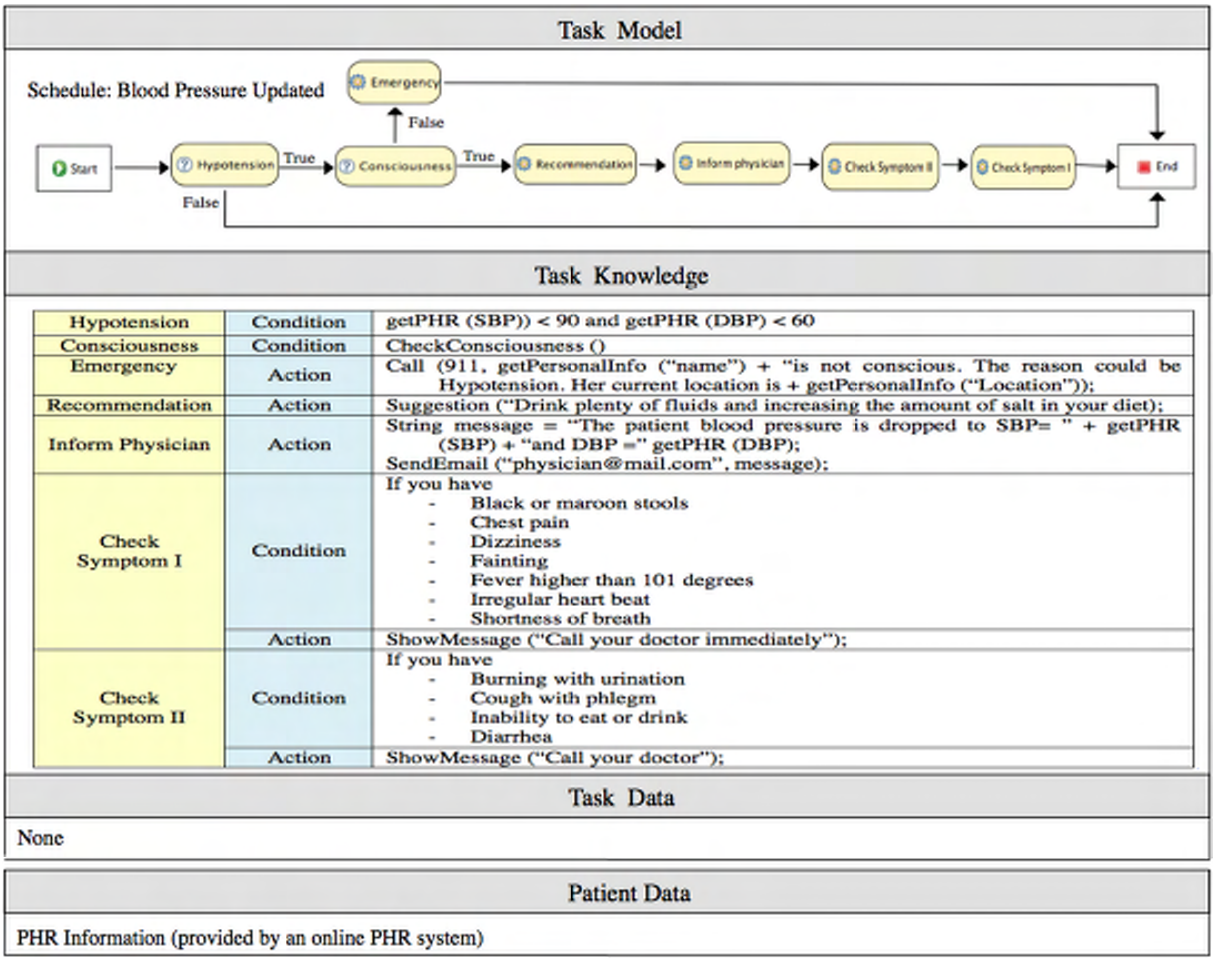


Figure B.3. *Hypotension Support Task*: **The vNurse calls the emergency center if the patient is unconscious. Otherwise it provides recommendations to the patient.**

B.2 Hypotension

High blood pressure medications may result in hypotension (low blood pressure) and temporary unconsciousness. Consequently, the doctor should make sure the patient receives emergency medical services promptly in the case of extreme hypotension. For example, the patient should drink fluids and salty foods to increase blood pressure level. There are some important symptoms that must be reported to the physician (with different priorities). The corresponding VRN task and medical guidelines are shown in Figure B.3, where the virtual nurse interacts with the patient several times.

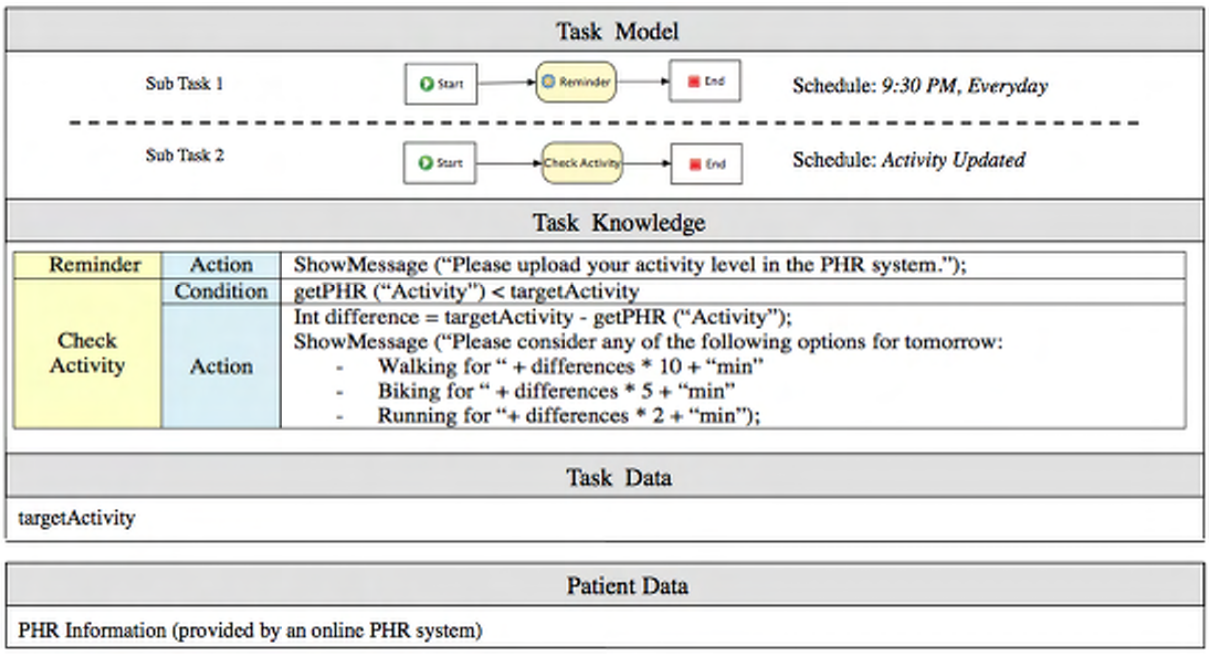


Figure B.4. *Physical Activity Monitoring Task*: **The vNurse keeps track of the patient's activity and provides advice to the patient.**

B.3 Physical Activity Monitoring

In addition to a healthy diet, physical activity is advised for a patient with hypertension. However, physicians do not have enough time to monitor the activity levels of their patients. Using the VRN system, a physician can assign a task to the virtual nurse to monitor and control the patient's activity level. This task sets activity targets and plans for the patient. The patient is advised to use a wearable device that can capture his/her activity level (e.g., fitbit) by counting the steps taken during some specified period of time. If the amount of exercise does not meet the target threshold value, the vNurse suggests extra activities for the following day. This task is shown in Figure B.4 where the activity level is measured in the range 0 to 100.

B.4 Diet and Weight Control

A healthy diet plays a very important role in hypertension. This could also affect the amount of medication which must be taken by the patient. A physician wants to make sure the patient is on a customized diet which forbids or restricts certain types of foods. The total calories consumed during a day should not exceed a defined limit. Using the VRN system, the physician defines a diet task for the virtual nurse. At the end of each day, the vNurse asks the patient about the total

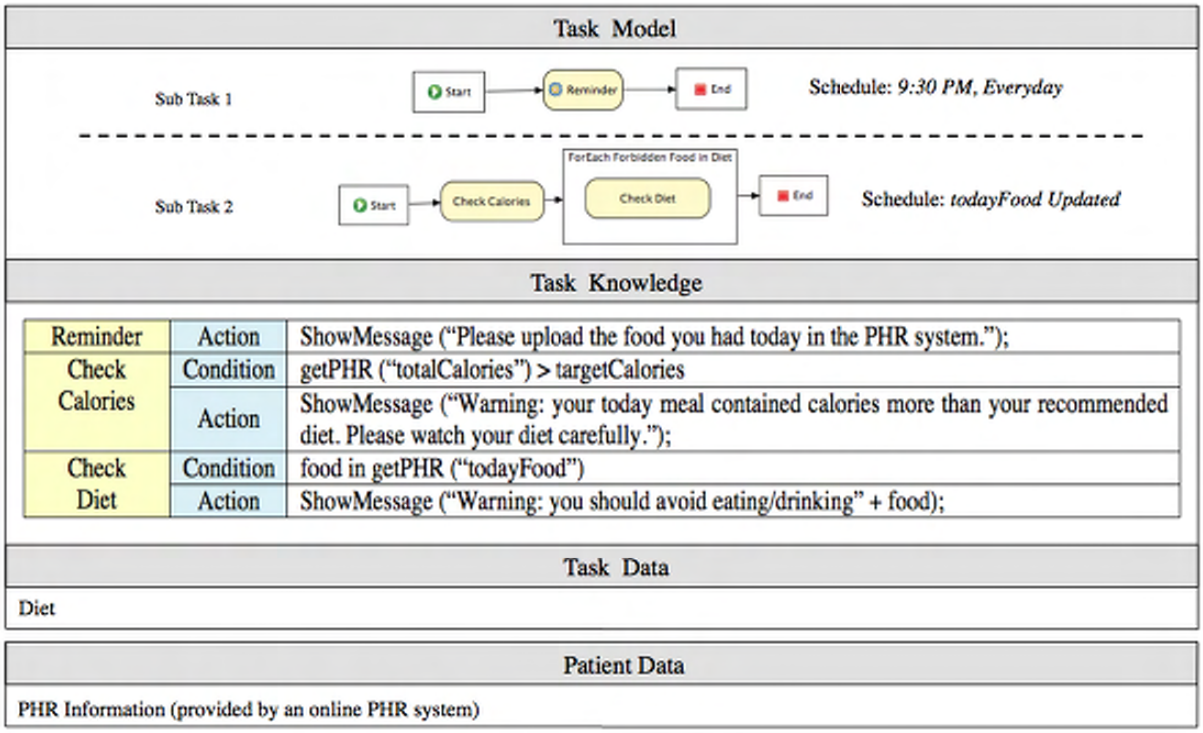


Figure B.5. *Diet Control Task*: **The vNurse provides required warnings if the patient does not watch his/her diet carefully.**

calories as well as forbidden and restricted foods. It also gives the patient appropriate warnings if the diet is not being watched carefully. This task is shown in Figure B.5.

Similarly, the physician can define an acceptable range for the patient's weight and use the vNurse to make sure the patient watches her diet carefully. If her weight goes up, the vNurse uses the assigned task (shown in B.6) to give her proper diet and advice. For this purpose, the virtual nurse reminds the patient to report her weight once a day at a specific time. The doctor may also ask the patient to lose some weight based on a specific schedule. Therefore, the vNurse will check the patient's weight against the target weight of the current week.

B.5 Treatment Evaluation

Treatment evaluation helps to improve quality of care and save money and resources. Hypertension treatment may include prescribing one or more medications; starting a regular exercise program; and improving diet and eating habits. Outcome evaluations aim to assess treatment effectiveness by seeking to find out, among other things:

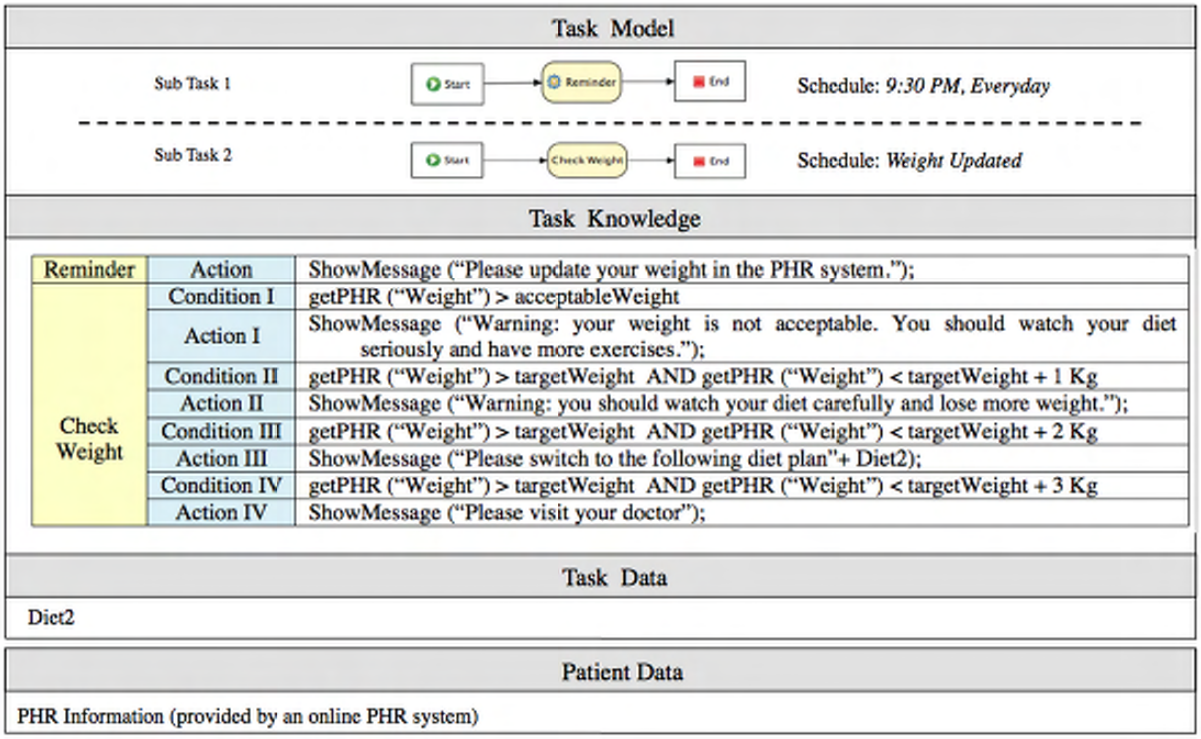


Figure B.6. *Weight Control Task*: **The virtual nurse monitors the patient's weight and provides required warnings/messages to control weight.**

- How a prescribed drug (e.g., Calcium Channel Blockers) works. For this purpose, multiple BP readings should be taken at various times throughout the waking hours of the patient. Using the VRN system, the physician assigns a task to the vNurse to first remind the patient to take the medication (e.g., at 11:00 AM) and then keep track of patient blood pressure during the day for a few days.

- Whether the patient should start taking a second medication. To evaluate how effective it is to take two medications, the physician assigns a task to the virtual nurse to ask the patient to take the second medication in 3 days. The virtual nurse also monitors the patient's blood pressure before and after taking the second medication. Finally it generates and e-mails a report for the physician.

- When is the best time to take a medication? The physician asks the patient to take the medication at different times each day and then assigns a task to the virtual nurse to monitor the patient's blood pressure several time during the day for a few days. Finally, the physician will receive a report from the virtual nurse to help decide about the best time for taking the medication.

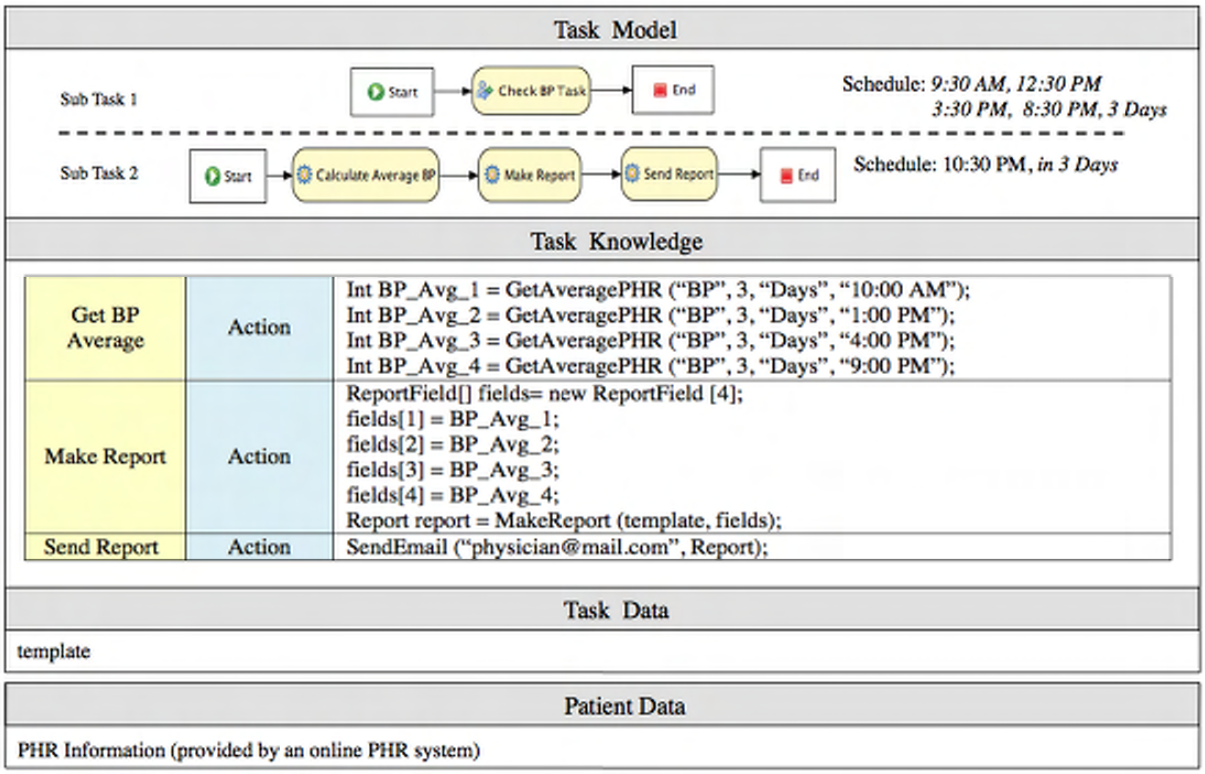


Figure B.7. *Medication Evaluation Task*: **The virtual nurse monitors the patient's blood pressure while the patient is taking a medication, and sends a report to the physician.**

The task message corresponding to the medication evaluation task (first item) is shown in Figure B.7. Task messages for the second and third items can be defined similarly by modifying the task knowledge and sub task schedules.

B.6 Heartbeat Monitoring

A regular heartbeat is necessary to stay healthy and active. However, a person may experience different heartbeat behaviours at different locations and situations. Moreover, irregular, high, and low hearth beat rates could be the result of taking certain medications. A patient with irregular heartbeat rhythm may be recommended to take medications. It could be useful for patients if they can predict irregular heartbeat patterns and prevent them by taking medications in advance. For this purpose, the physician assigns a task (shown in Figure B.8) to the virtual nurse to monitor the patient's heartbeat and notify him/her if this exceeds a threshold. The patient may then be informed about the situation which caused the irregular heartbeat. The patient could take this information into account in deciding about the best time of take medications. Patient heartbeat

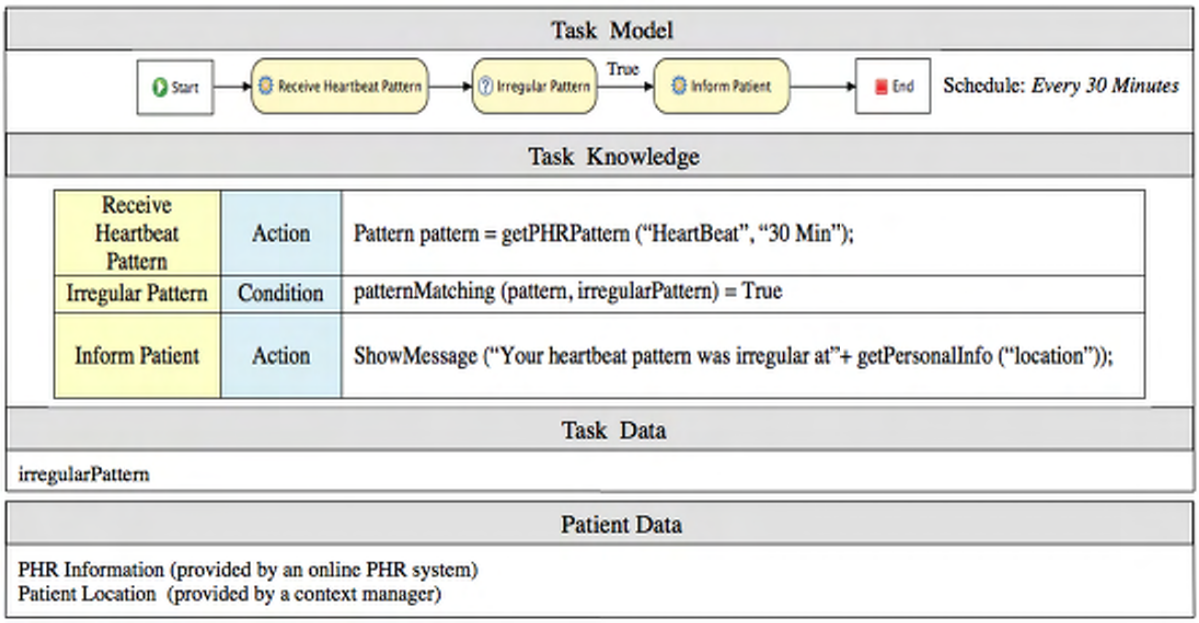


Figure B.8. *Heartbeat Monitoring Task*: **The virtual nurse monitors the patient's heartbeat and informs him/her if there is any irregular heartbeat pattern.**

is monitored using a wearable device. This device captures the heartbeat continuously and then updates it through a wireless transmission to the patient's PHR system. Finally, there is a GPS system installed on the client's smart phone which tracks the patient's location and updates it via a context manager.

B.7 First-aid Tasks

In case of an emergency situation, an emergency center needs to access the latest information about the patient's health condition. After receiving a call from a virtual nurse, the emergency center assigns a task to the vNurse based on its policies in case of emergencies. Figure B.9 shows different components of this task. In this case, the virtual nurse first makes a call to the patient's emergency contact and informs her about the patient's situation. Then, if the patient is in cardiac arrest, the virtual nurse asks the people around the patient to begin CPR. Finally, it sends the patient's PHR information to the emergency center and keeps the center updated about the patient's conditions.

Task Model

Schedule: *Immediately*

Task Knowledge

Emergency Contact	Action	Call (getPersonalInfo ("Emergency_Contact"), getPersonalInfo ("Name") + "is in emergency status. She is at" + getPersonalInfo ("Current_Location"));
CPR	Action	Alert (" If the patient is not breathing or has no pulse, begin CPR");
Patient Status Report	Action I	String report = "Medication History:" + getPHR ("Active Medication") +getPHR ("Antihypertensive Medication") + getPHR ("Illicit Drugs") + "Hypertension History" + getPHR ("Prior Hypertension") + getPHR ("Hypertension Level") + getPHR ("BP Control") + "End-organ Dysfunction") + getPHR (" Cushing Disease") + getPHR ("Thyroid Disease") + getPHR ("Renal Disease"); SendEmail ("emergency @ email.com", report);
Status Change Report	Condition I	UpdatedPHR ("Consciousness Level") = True
	Action I	String report = "Patient's consciousness level is changed to" + currentPHR ("Consciousness Level"); SendEmail ("emergency @ email.com", report);
	Condition II	UpdatedPHR ("BP") = True
	Action II	String report = "Patient's Blood Pressure is changed to" + currentPHR ("BP"); SendEmail ("emergency @ email.com", report);
	Condition III	UpdatedPHR ("Body Temperature") = True
	Action III	String report = "Patient's Body Temperature is changed to" + currentPHR ("BT"); SendEmail ("emergency @ email.com", report);

Task Data

None

Patient Data

PHR Information (provided by an online PHR system)

Figure B.9. *First-aid Task*: **The virtual nurse calls the emergency contact and informs the emergency center about the patient's health condition.**

B.8 Pharmacy Tasks

Pharmacy tasks allow pharmacists to make independent technical therapy decisions. These tasks include prescribed medication timing and dosage schedules. Pharmacists should inform the patient about potential side effects when filling prescriptions. Pharmacists can assign the following tasks to control medication usage virtually and remotely.

- Sending reminders to the patient to take him/her medication on time. The reminder should be based on best practices in order to increase positive medication outcomes. Figure B.10 represents one example of these tasks.
- Each medication may have some side effects. After three days of applying the medication, the vNurse asks the patient if she has experienced any of the medication side effects. In the case

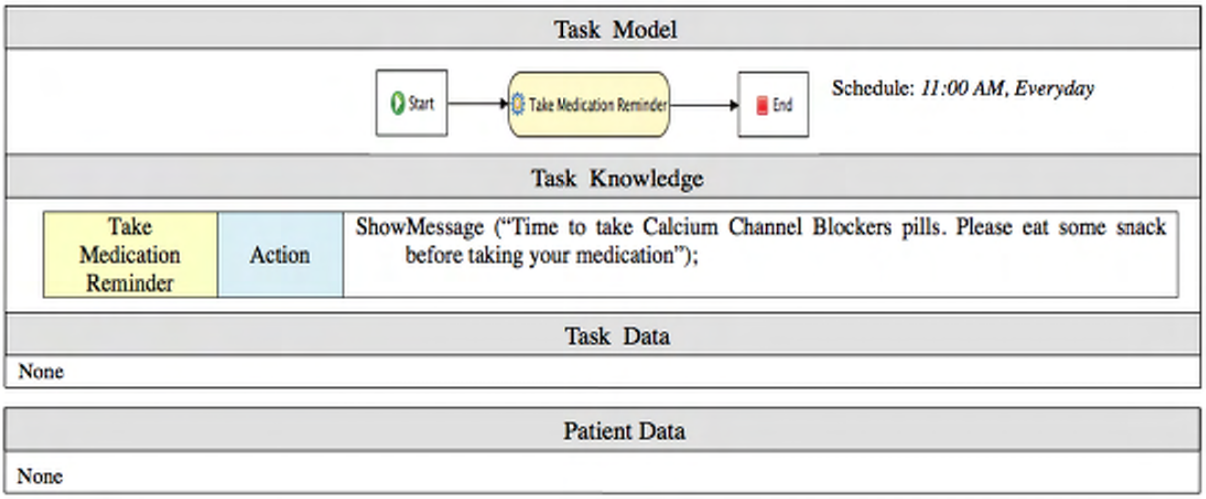


Figure B.10. *Take Medication Reminder Task*: **The virtual nurse sends a reminder to the patient to take her medication.**

that a medication side effect occurs, the virtual nurse provides the required recommendations for the patient. This could be a change in the medication dosage or asking the patient to contact her physician. The corresponding components to this task are shown in Figure B.11.

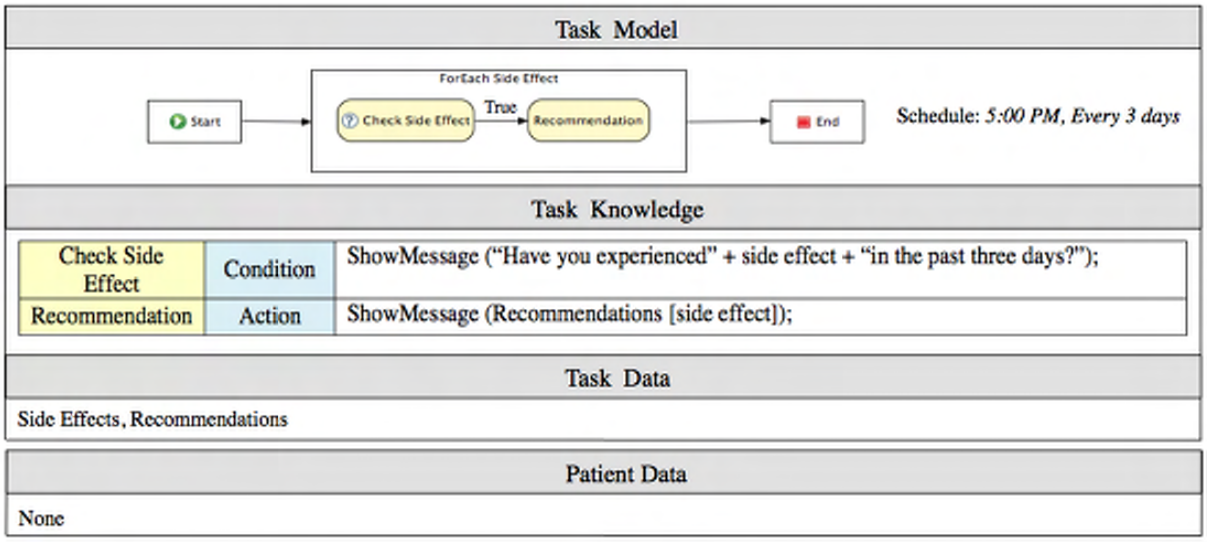


Figure B.11. *Monitoring Medication Side Effects Task*: **The virtual nurse provides required recommendations if the patient experiences any medication side effects**