

Client-side Service Composition Using Generic Service Representative

Mehran Najafi and Kamran Sartipi
Department of Computing and Software
McMaster University
Hamilton, ON, Canada
{*najafm, sartipi*}@mcmaster.ca

Abstract

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 violation, security breaches, or network traffic overloading. In this context, we introduce the concept of "task service" which is a web service that can process the client data locally at the client-side using a generic software agent that we call "service representative". The proposed task service and service representative allow us to present 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. Therefore the client is not required to reveal its resources to service providers and hence its privacy and security are maintained. Moreover, large client data are processed locally that results in less network traffic. We have developed a prototype system for the proposed extended SOA model. Finally, we will discuss the advantages of the proposed approach over traditional server-side approaches using a case study in healthcare domain.

1 Introduction

In Service Oriented Architecture (SOA) the business functionality of an enterprise is mod-

eled by (web) services. To achieve more sophisticated functionality, web services should be reusable and composable. Business Process Execution Language (BPEL) is an industry-wide standard that models a business process based on collaborating web services. Moreover, the BPEL process is provided as a composite web service that can be called by a service client.

A traditional simple or composite web service processes client request completely at the server-side. This requires the client to send its data as the service parameters to the service provider. On the other hand, there are several cases in business domain that the client data should be processed locally at the client-side such as:

- Client data is confidential and revealing it to a service provider violates the client's privacy and security.
- Client data is large or changing over time, thus transferring it to a service provider increases the required bandwidth.
- Real-time and time-critical services, where transferring client data to a service provider increases the response time.

Since, traditional web services lack the ability of client-side processing, they fail to provide an efficient and secure way to model these services. As a solution, a service provider can customize a software agent and send it as its service response to process client data locally. However, this solution increases the net-

Copyright © 2010 Mehran Najafi and Dr. Kamran Sartipi. Permission to copy is hereby granted provided the original copyright notice is reproduced in copies made.

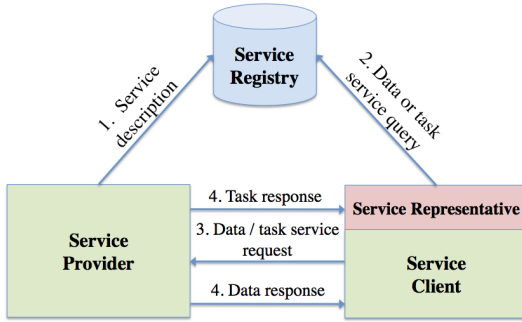


Figure 1: Proposed extended SOA reference model, where "Service Representative" processes client data based on an assigned "Task" from the service provider.

work traffic and introduces security and privacy problems. As an alternative, we propose a generic software agent that is located at the client side and can be customized and trained with the service messages to perform a task (i.e., processing client data locally). We define this generic agent as *Service Representative*, and the type of service that requires client-side processing as *Task Services*. Figure 1 shows the extended SOA reference model to support task services. Furthermore, we extend the notion of task service and service representative to perform composite tasks at the client side. Consequently, client data are processed at the client side and web services are composed via the service representative. As a result, the proposed model improves the current state of SOA in terms of maintaining the client privacy and security, reducing the network traffic as well as improving the web service interoperability.

The organization of this paper is as follows. Task services are introduced in Section 2. Client-side service composition is described in Section 3. The proposed architecture and its main components are discussed in Section 4. The developed prototype system is described in Section 5. A case study of composite task services is presented in Section 6. Section 7 discusses related work to our approach. Finally, conclusions and future work are discussed in Section 8.

2 Task Service

Data service represents a typical web service that receives client data, processes it entirely at the server-side, and returns the results as the service response to the client. The service response will be consumed directly by the client.

Task service represents a new type of web service that processes client data partially or completely at the client-side. According to the client request, a task service first performs required server-side processing and then it defines a task for the generic service representative to perform client-side processing. The task will be executed at the client-side and the client will use the task result as the service response. A task service returns a task message to the client-side with the following components:

$$Task = \langle Model, Knowledge, Data \rangle$$

- *Task Model* specifies a task using an abstract Business Process Model (BPM).
- *Task Knowledge* provides the required Business Rules (BR) and Business Actions (BA) to realize the specified BPM.
- *Task Data* represents Business Objects (BO) that are consumed by the business rules and actions during the business process. Task data consists of two parts: server-side (provided by the service provider) and client-side (provided locally by the service client).

Service representative is a generic and client-side software agent that can be employed by a service provider to perform different task services. For this purpose, the service representative uses the received task components to customize and train itself in order to perform client-side processing of the web service on the client data (i.e., client-side task data). Task components are messages that can be transmitted efficiently over the network. Consequently, the proposed "generic service representative and its assigned task" relieve a service provider from sending the entire "specialized agent" each time. Moreover, the client determines both the local data that the service

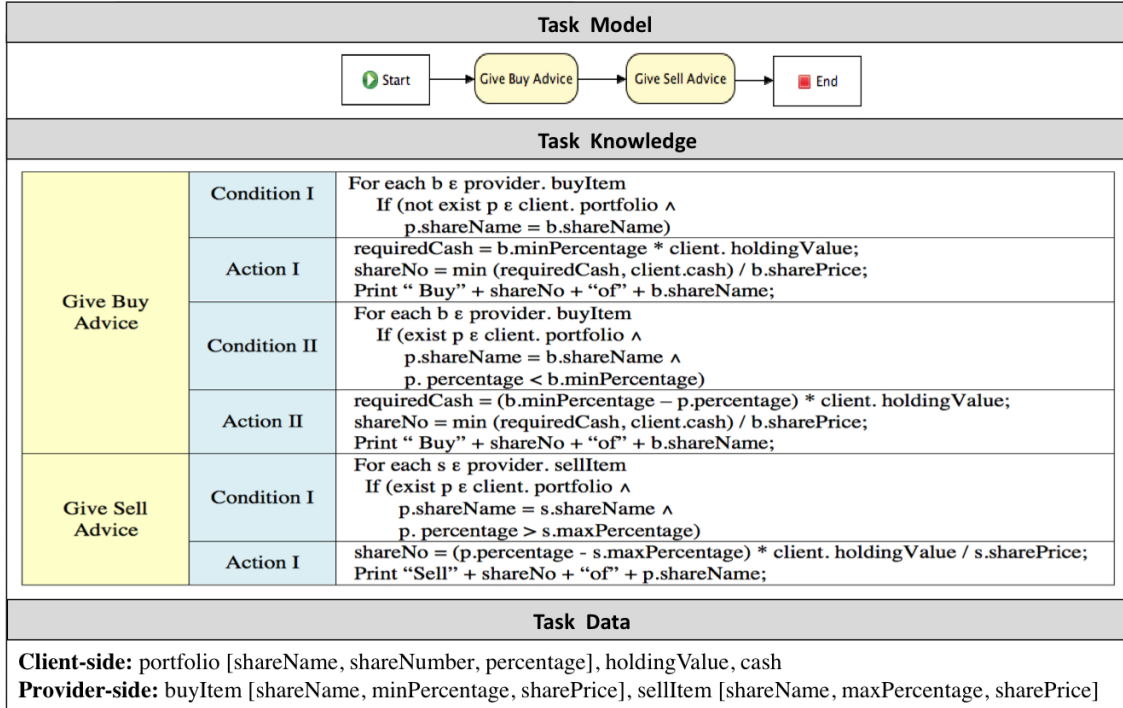


Figure 2: Task service to personalize general financial advice based on the client personal information at the client site. Conditions are expressed using First Order Logic (FOL) and actions are defined using Java statements. In each step of the task model, the corresponding rules and actions are applied.

representative can access the computer resources (e.g., CPU time, storage, and memory) that the service representative requires. Therefore processing client data using service representative improves the security, privacy, and efficiency features of both traditional web services (data services) and mobile agent approaches.

Figure 2 illustrates different components of a task service that provides personalized financial advice without asking the client to send its personal information. This service receives 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). The client keeps the sensitive information local and private, such as client's financial information (portfolio, saving, debt, and salary). This task service has both server-side and client-side processing as follows.

- *Server-side processing:* the service provider generates a set of general financial advice (stock buy and sell advice) according to the client's preferences. A stock buy (or sell) advice recommends the client to have minimum (or maximum) percentage of a specific share in their portfolio. Moreover, it specifies a task for the service representative to personalize the generated general advice based on the local client's financial information.
- *Client-side processing:* a service representative receives the task, which allows it to customize and train itself in order to perform the task. According to the task model, service representative applies *Give Buy Advice* and *Give Sell Advice* business rules on both client-side and server-side task data and returns the generated personalized advice as the service response to the client.

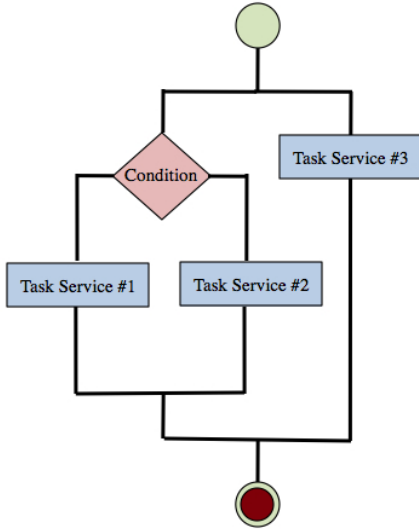


Figure 3: Example of a composite task service.

3 Composite Task Service

In the context of our approach, a composite task service (as a client-side service composition approach) is defined as a combination of data and task services over a designated flow structure that is performed by a service representative at the client side. The service representative plays the role of a service orchestrator that coordinates the execution of the involved web services (both data and task services). The incorporating web services are not aware that they are taking part in a higher-level business process. An example of a composite task service is shown in Figure 3.

For each composite task service, the corresponding "composite task model" and "task data schema" are published as a part of the service description in a service registry. A client obtains the service description from the service registry. Then, the client forwards the composite task model to a service representative, and uses the task data schema to provide the required client-side task data for the service representative. The service representative calls each involved task service according to the model and uses the received task message to customize and train itself for executing that task. Each executed task modifies the client data, which collectively represent the composite service response.

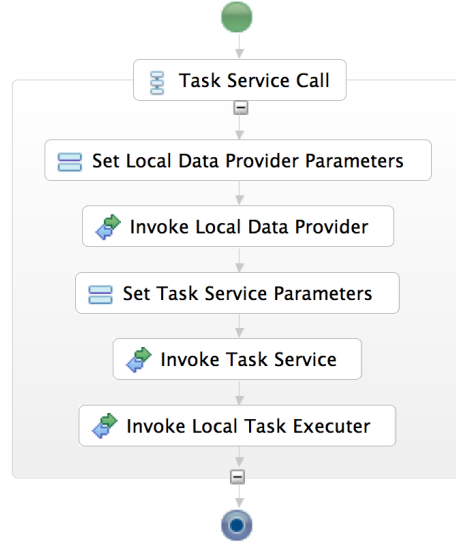


Figure 4: Steps for invoking a task service in a BPEL model.

3.1 Composite Task Model

A composite task model specifies the exact order in which participating task services should be invoked, either sequentially or in parallel. Moreover, conditional behaviours can be expressed.

We use BPEL to model a composite task service that allows us to define complex business processes in an algorithmic manner including loops, variables, and fault handlers. We consider two local services for the service representative: *local data provider* to read client data from a communication channel and *local task executor* to execute a received task. The required steps to call and execute a task service are shown in Figure 4.

3.2 Client-side vs Server-side Service Composition

The proposed client-side service composition approach is not meant to replace the 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 the following SOA features.

- *Privacy*: the confidential and sensitive client data are kept local and processed at the client side.
- *Security*: the service representative functionality is under client control and it can not perform malicious behaviour.
- *Interoperability*: task services can be composed efficiently using well-known forms of task model, knowledge, and data.
- *Efficiency*: large client data are processed locally that reduces the required bandwidth.
- *Response Time*: realtime and variable data are processed offline that reduces the response time.

4 Architecture

In order to develop task services that are executed and composed by the service representative an architecture is required. In this section, we extend the typical architecture of SOA implementations to enable service providers to employ the generic service representative at the client side. The proposed architecture (Figure 5) includes three main components as follows.

4.1 Service Client

Each service client (or "client" for short) consists of a client application and a communication channel as follows.

Client Application. It is a traditional client application that sends a data or task service request to a simple or composite service provider. Then, the client application puts the required client data for a task service into a communication channel based on the schema received from the service registry. In a case of a data service, the client receives the service response directly from the server (traditional method). However, in a case of a task service or a composite service, the client application receives the service response indirectly from the communication channel.

Communication Channel. It consists of a number of ports that are connection links to the client data, as well as the means for the client application to receive the task service result through the service representative. The client grants permission to the service representative to read/write a number of its data through these ports where ports can be input, output, or bi-directional (from the client point's of view). Moreover, each port is assigned a scope as public or private. The content of a public port can be sent to a service provider as service parameters while the content of a private port can only be used locally by the service representative. In addition to typical descriptions for data services, the service registry must include the required communication channel schema for each task service.

4.2 Service Provider

An enterprise system provides a number of services for its clients where each service executes one or more business processes of the enterprise. 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 proposed service provider has two layers: the processing layer performs the server-side processing of a web service while the task layer defines a task for the service representative to perform the client-side processing of the web service at the client side. Therefore, the enterprise business components are divided between these two layers. While the processing 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 supports the communication contracts (message-based communication, formats, protocols, security, exceptions, and so on) for the service.

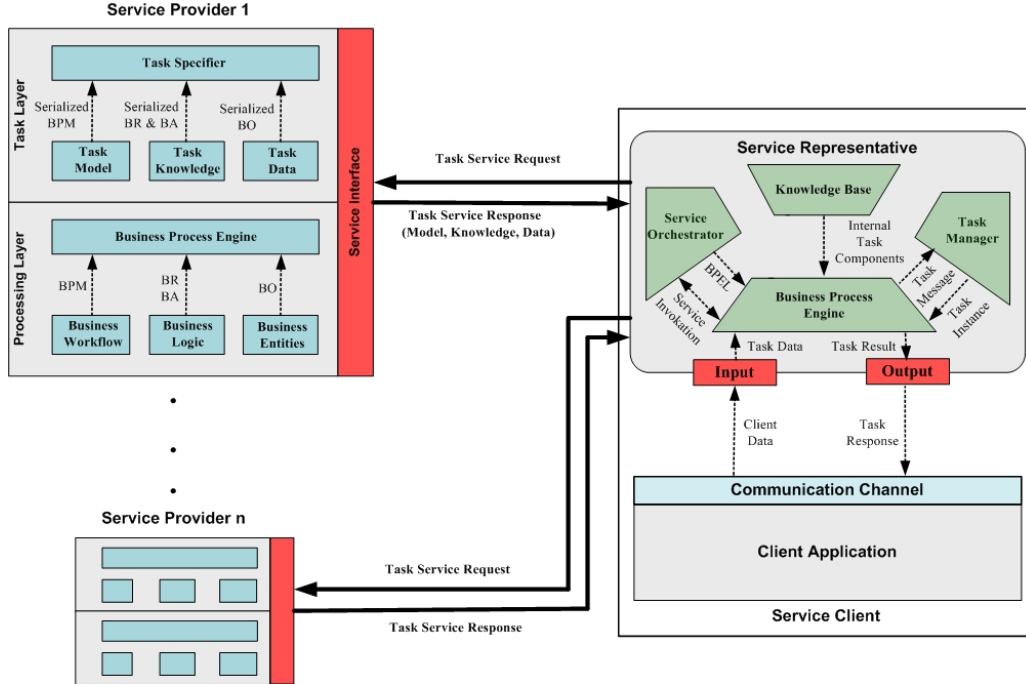


Figure 5: Extended SOA architecture to perform a client-side service composition. Each service partner generates a three-segment task message to customize the generic service representative to perform a sub task of the composite task.

Processing Layer. Server-side business processes, rules and actions, and objects are stored in the business workflow, logic, and entities components, respectively. The business process engine executes the corresponding business process with each service. If the service only requires server-side processing, this layer responds to the client by a single-segment response message (i.e., called data message). Moreover, the business process engine can modify the business components in the task layer, if the requested task depends on server-side processing.

Task Layer. Client-side business processes, rules and actions, and objects 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. Task specifier provides the required model, knowledge, and data for each task request to be sent by a three-segment response message (i.e., called task message) to the client.

4.3 Service Representative

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

- *Input:* inputs client data through the communication channel.
- *Output:* outputs task response to the client through the communication channel.
- *Knowledge Base:* stores internal and domain-based business rules and actions to relieve the service provider from sending them each time.
- *Business Process Engine:* executes a task instance by applying business rules and performing business actions. Moreover, it executes BPEL processes representing composite task models.
- *Task Manager:* supports the entire life cycle of a task instance (i.e., from creation to termination) that is divided into three phases as follows.

1. *Customization phase*: sets up the agent configuration (including inputs and outputs) based on the service description published on the service registry and creates an abstract process based on the model segment of a received task message.
 2. *Training phase*: generates a task instance from the abstract process using internal and external task knowledge.
 3. *Execution phase*: passes the task instance with the relevant task data (i.e., received from the input component and task data segment) to the business process engine to be executed. Finally, service representative passes the task results to its outputs to be passed to the communication channel.
- *Service Orchestrator*: supports the client-side service composition as follows.
 1. Creates a communication port (i.e., a service stub) for each collaborating service as well as the local services based on the composite task model.
 2. Sends the BPEL model to the business process engine to be executed.
 3. Business process engine invokes service partners through the service stubs; the received task messages will be given to the task manager to perform the client-side processing.

5 Prototype System

To evaluate the effectiveness and feasibility of our model, we developed a prototype system of the proposed architecture including service representative, two-layer service provider, and service client. This prototype, namely *Enterprise Representative version 1.1* (EntRep v1.1), is implemented based on J2EE 1.5 technologies and Apache Tomcat 6.0 application server. Service representative includes Apache ODE 1.2 to run BPEL process and Drools 5 to apply business rules and execute business actions. Communication channel is implemented by an

array of pointers to client data stored in a MySQL database.

EntRep v1.1 supports production rules as the form of task knowledge where each business rule is defined as follows:

```

when
    < conditions >
then
    < actions >
  
```

conditions are expressed by First Order Logic and *actions* are defined by Java statements or functions. Business process models are converted to XML format, business rules are encoded by PMML standard version 3, and business objects are serialized to form the task message. EntRep v1.1 is provided by two Java packages, namely the *TaskService* and the *ServiceRep*, that can be imported into any service provider and client applications as follows.

- **TaskService** package: provides graphical APIs and widgets for a service developer to develop a task service by defining the task components.
- **ServiceRep** package: provides APIs for a service client to call a task service by following two steps.
 1. Install a service representative and a communication channel that work for every task service. The installed service representative including a business process engine requires less than 3MB of hard disk.
 2. Pass the client data into the communication channel according to a schema obtained from the service registry. This package provides get and set methods to access the communication channel.

We also developed Service Representative Manager v1.1, using ServiceRep APIs, to control different phases of a composite task execution. The client can view the content of each port within the communication channel during the task execution. Figure 6 illustrates a snapshot of the Service Representative Manager executing a sample composite task service.

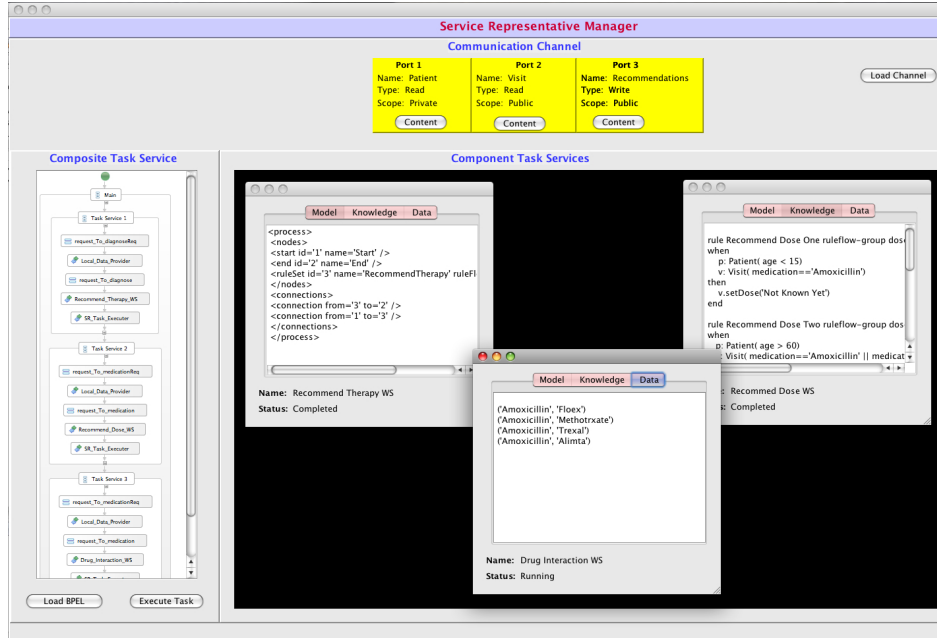


Figure 6: Snapshot of the prototype service representative manager that is running a composite task service.

6 Case Study

There are two types of Clinical Decision Support System (CDSS): guideline-based and model-based [8]. 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 on unknown data (test data). Since each approach has both benefits and limitations, a combined approach could lead to a more accurate diagnosis.

CDSSs cannot be developed efficiently and securely using traditional web services due to the following reasons. Guideline-based approaches require transferring patient’s health information while they are highly sensitive and disclosure of this information would identify patients. On the other hand, model-based approaches do not consider local data to build their decision models. Consequently, the obtained model may not be matched with the client data. However, the local data are often too large to be transferred efficiently to the services.

Using the proposed methodology and implemented tool (EntRep version 1.1), we modeled and developed a secure and context-aware CDSS by 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 the mentioned issues. The composite task is shown in Figure 7 and discussed in the following subsections.

6.1 Service Client

The client puts its data and resources into the communication channel (Figure 8) 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 filled by the physician after consulting with the service representative.

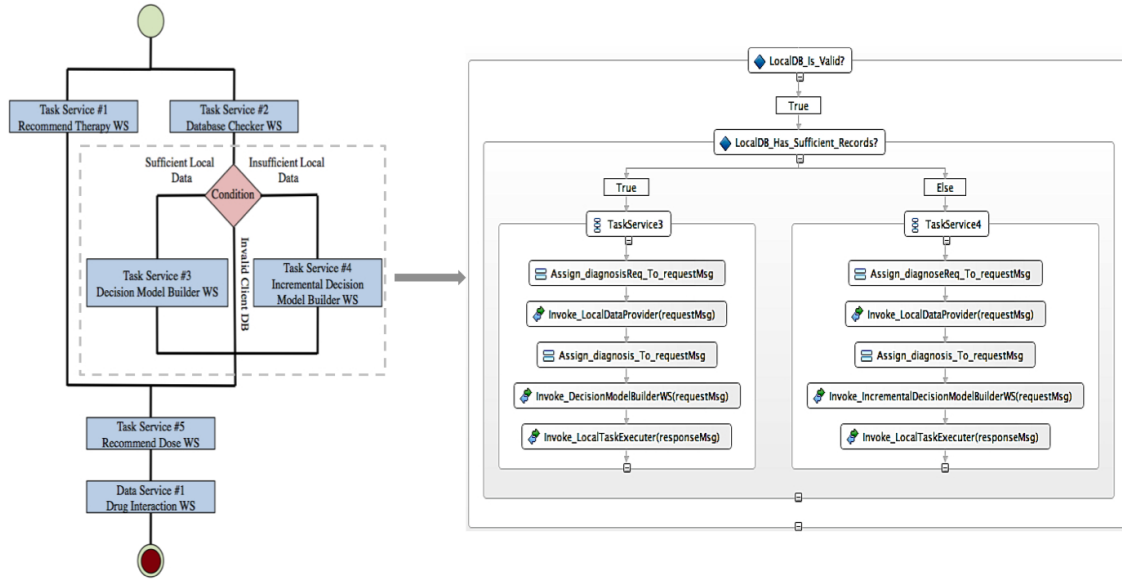


Figure 7: Composite task model (left) and the corresponding BPEL model (right) of the secure and context-aware CDSS.

3. *Patient database* (private, read only): contains the patients' health records within the healthcare organization.
4. *Visit database* (private, read only): contains doctor visits' information within the healthcare organization.
5. *Recommendation* (private, read / write): receives i) recommended medication: guideline-based (gMedicataion) 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.

6.2 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. Moreover, this task 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).
5. *Recommend Dose* (task service): receives a 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.

Patient	Visit	Patient DB	Visit DB	Recommendation
- name - age - weight - creatinine level - allergies - active medications	- patient name - date - diagnosis - medication - dose	Each row contains one patient record	Each row contains one visit record	- mMedication - gMedication - mDose - gDose - warning

Figure 8: Communication channel schema for the secure and context-aware CDSS

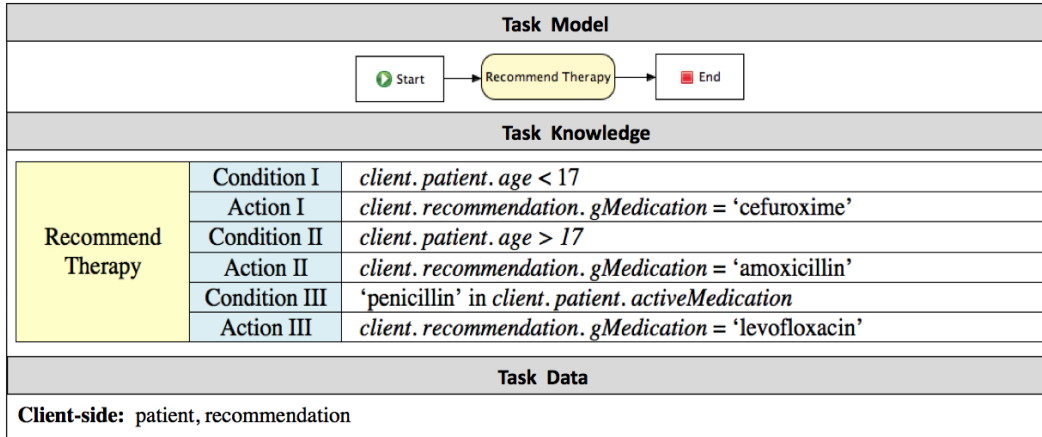


Figure 9: Task generated by Recommend Therapy web service where it receives "acute sinusitis" as its parameter.

6.3 Service Representative

The service representative performs the composite task as follows. The service representative executes the task received from *recommend therapy* to calculate gMedication (Figure 9). Then, it verifies local databases using *database checker* task service (Figure 10). If the patient database and the visit database have enough records, service representative executes the task received from *decision model builder* to build a decision model completely from local data (Figure 13). Otherwise, the service representative receives an incomplete model from *incremental decision model builder* and completes it based on local databases (Figure 14). Then, the service representative applies the decision model on 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 *recommend dose* for gMedication and mMedication to calculate gDose

and mDose (Figure 12). Finally, *drug interaction* data service 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 11.

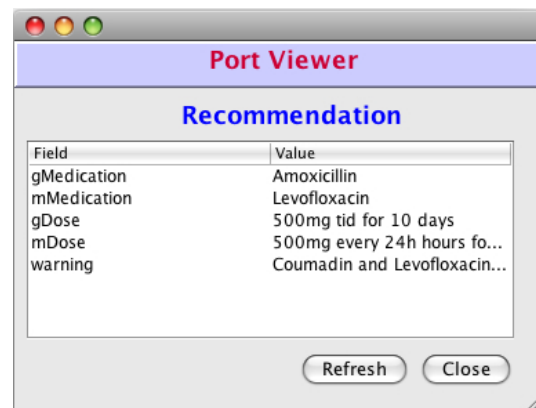


Figure 11: Recommendation port after the service representative completes the composite task.

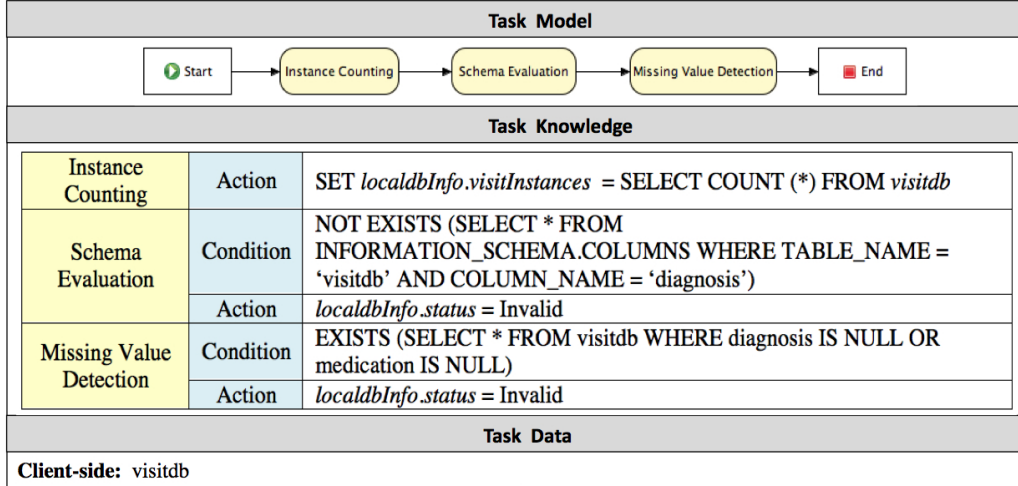


Figure 10: Task generated by Database Checker web service where it receives *visitdb* schema as its parameter.

6.4 Evaluation

An equivalent traditional (server-side) composite web service for the presented context-aware CDSS takes complete client’s data (i.e., patient and visit databases and records) and returns the final recommendations. Using the proposed (client-side) composite web service, the patient and visit information are kept local, and therefore the client’s privacy is maintained. Moreover, the proposed approach shows a better performance compared to the traditional approach according to the following metrics.

Service Message Size (SMS) is the total size of service request and response messages defined as follows.

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

The server-side approach requires transferring complete client data from client to the server. Although it returns small response messages containing final decision, the request message has almost the same length as the client data. Moreover, the collaborating web services are either integrated into a central provider or distributed over the network. The latter requires additional sending the client data from the central service provider to the partner service providers.

On the other hand, the proposed (client-side) approach processes client data locally that implies SMS is independent of the length of the client data. Therefore, the request message is short while the response messages are large containing the task definitions. Figure 15 illustrates a comparison of these approaches in terms of SMS where they model the presented case study. A logarithmic scale is used to show the lower values more clearly.

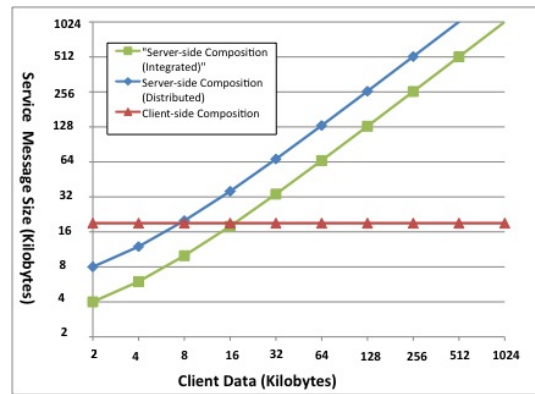


Figure 15: Service Message Size comparison.

Service Response Time (SRT) is divided into two factors: Network time (N) and Process time (P) defined as follows.

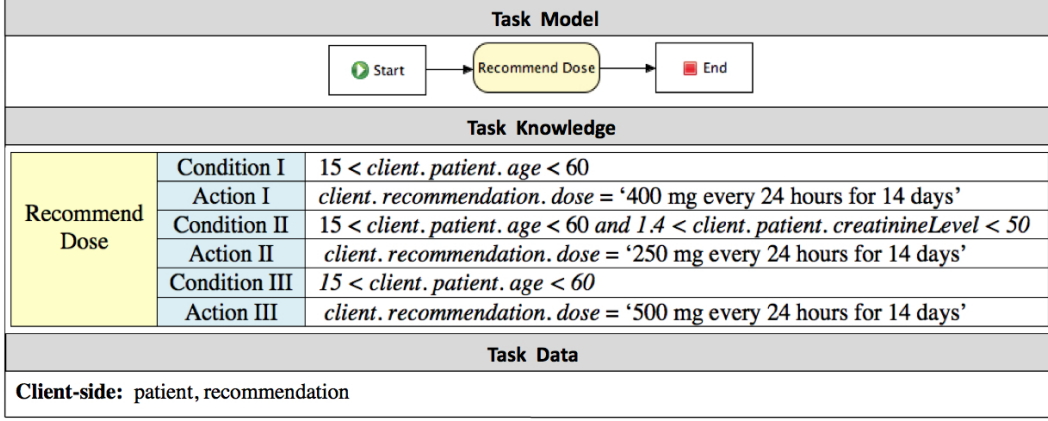


Figure 12: Task generated by Recommend Dose web service where it receives "amoxicillin" as its parameter.

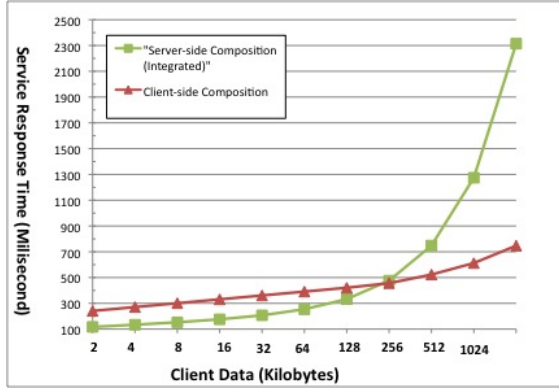


Figure 16: Service Response Time comparison.

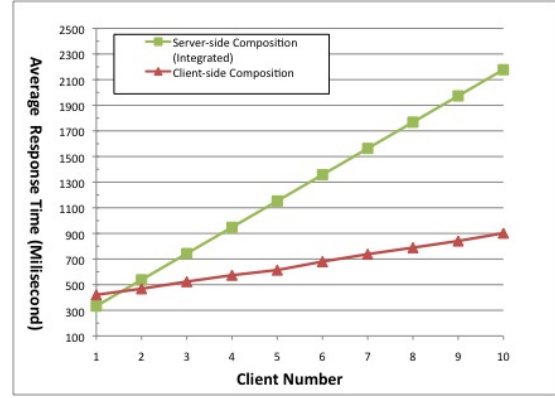


Figure 17: Average Response Time comparison.

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

Network time is the amount of time required to transfer request and response messages that depends on both network bandwidth and message size. Process time is the amount of time it takes a web service to perform its designated task. Since, servers use more powerful CPUs, server-side approaches have less process time. On the other hand, the proposed client-side approach requires smaller messages results in less network time.

For this case study, we obtained the process time ($P(s)$) for different size of client data using a 2.4 GHZ dual-core CPU. Moreover, we assumed $P_{ServerSide}(s) = \frac{1}{2}P_{ClientSide}(s)$ and a high-speed bandwidth (1 Mbyte/Sec)

connects the client to the servers. Figure 16 shows the result of comparison between the client-side and an integrated server-side service composition in terms of SRT. This result shows the proposed approach overcomes the traditional approaches when the client data grows.

Average Response Time (ART) extends the SRT metric when multi clients invoke a service simultaneously. ART for service s where it has n simultaneous clients is defined as follows.

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

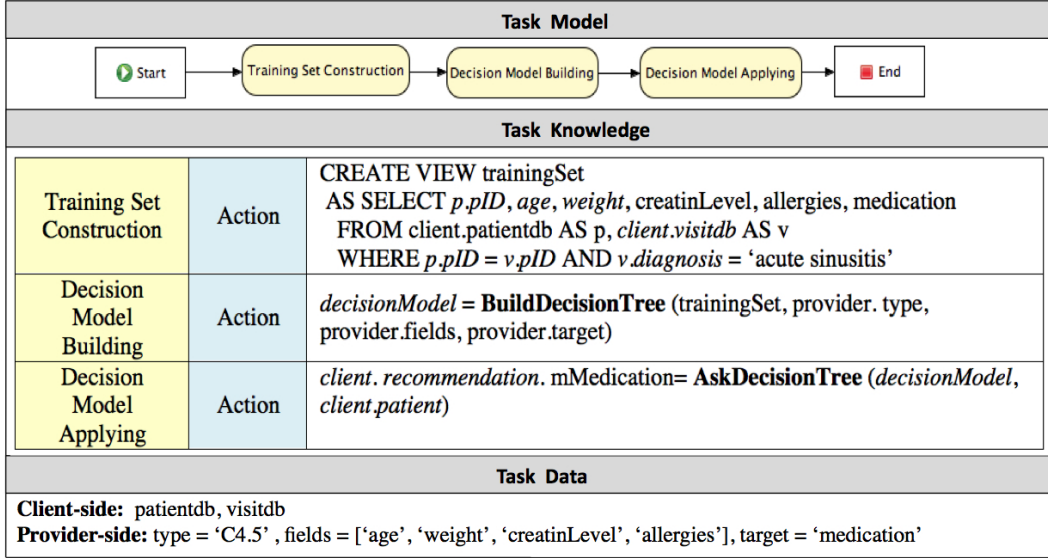


Figure 13: Task generated by Decision Model Builder web service where it receives "acute sinusitis" as its parameter. The internal task knowledge are highlighted and *C4.5* is the type of decision tree.

To compute ART for the traditional approach, we consider a non-preemptive queue for the server processor that results in the following equation.

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

$P(s, i)$ represents the process time of service s when i clients call the service simultaneously. A comparison of these approaches is shown in Figure 17. In the proposed approach, service clients have their own service representative, which can process client data in parallel, and therefore the ART is improved significantly.

7 Related Work

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. Orchestration and choreography describe two aspects of organizing web services in a composite web service [3]. While orchestration requires a central process to coordinate sending and receiving messages among web services, collaborating web services com-

municate with each other directly in choreography. Static composition takes place during design-time while dynamic composition selects collaborating services at run-time [10]. While all traditional approaches compose web services at server side, the proposed approach intends to compose services at client side to improve security, privacy, and efficiency features in some applications.

Contextual information has been considered for selecting and binding services in a service composition [5]. These approaches work based on semantic description of web service and client context [14]. Therefore, more service metadata and analysis are required. In our approach, a service representative processes client context based on a context-free task service to provide a context-aware service response. As a result, context-free services facilitate service composition.

A software agent is a piece of software that acts on behalf of an agency to serve a user [7]. Software agents have been integrated into web services to facilitate SOA related tasks such as service composition [13]. For example, [4] proposes a distributed agent-based framework for business process execution in SOA. Moreover, each role or major function of an enterprise sys-

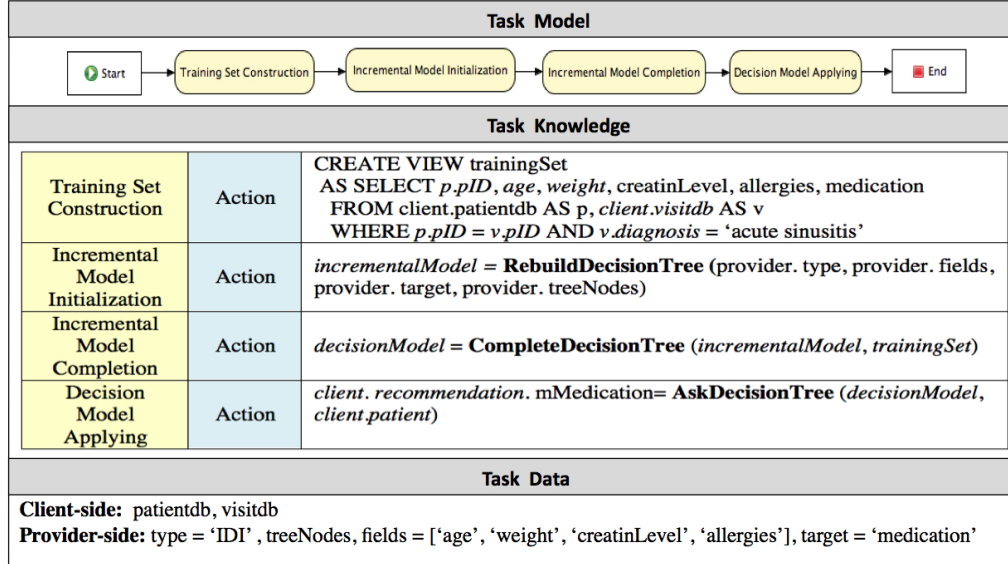


Figure 14: Task generated by Incremental Decision Model Builder web service where it receives "acute sinusitis" as its parameter. The internal task knowledge are highlighted and *IDI* is the type of incremental decision tree.

tem has been modeled by software agents [12]. In this paper, we model an enterprise agent in business domain by the proposed service representative.

Mobile agents [2] can physically travel across a network and perform tasks on different nodes. There are several security and privacy issues to be considered in mobile agent computing. Mobile agent architectures (e.g., Concordia [11] and Mole [1]) also suffer from low efficiency as they need to send the entire computer program or process. In contrast, we propose to employ generic client-side agents and customize them with service messages.

Finally, the proposed architecture extends our previous work in [9] and [6]. The former introduces a framework for data and knowledge interoperability in the health field where it enables knowledge (i.e., medical guidelines) to be transferred in association with data (i.e., patient EMR information) and the latter introduces a generic software agent that resides at the client side and customizes web service responses based on the client context.

8 Conclusions

This paper proposes a new computational model for SOA client-server computing when service provider processes the server part and defines a task for the generic service representative to perform the client part. We equipped the service representative 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 and thus, the security, privacy, and efficiency features of the enterprise systems will be greatly improved.

By defining cost functions for business components, we will investigate how to divide business components between client and server. Moreover, we plan to extend the SOA reference model with other generic components such as composition certifier to support client-side service composition. Finally, the proposed approach requires short messages to process client resources that works well with mobile devices, and therefore, our next work will revolve on developing a light version of the service representative to be installed on mobile devices and perform task services.

About the Authors

Mehran Najafi is a PhD student in computer science program at McMaster university. His research interests include service-oriented architecture, software agents, data mining, and knowledge management. He received his Master degree in Artificial Intelligent from the Sharif University in Iran. He was awarded OGS scholarship in 2009.

Dr. Kamran Sartipi received his B.Sc and M.Sc in Electrical Engineering and M.Math and Ph.D in Computer Science. He is a licensed Professional Engineer of Ontario. In 2003 he joined Department of Computing and Software at McMaster University. Currently he is with the DeGroote School of Business at McMaster University. He has been the codeveloper and co-director of eHealth program at McMaster University.

References

- [1] 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.
- [2] P. Braun and W. Rossak. *Mobile Agents: Basic Concepts, Mobility Models, and the Tracy Toolkit*. Morgan Kaufmann Publishers Inc., San Francisco, USA, 2004.
- [3] C.Peltz. Web services orchestration and choreography. *Computer*, 36(10):46–52, 2003.
- [4] G.Li, V.Muthusamy, and H.Jacobsen. A distributed service-oriented architecture for business process execution. *ACM Transaction on Web*, 4(1):1–33, 2010.
- [5] L.Bastida, F.Nieto, and R.Tola. Context-aware service composition: a methodology and a case study. In *The International Workshop on Systems Development in SOA Environments*, pages 19–24, New York, USA, 2008.
- [6] 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.
- [7] H. Nwana. Software agents:an overview. *Knowledge Engineering Review*, 11(3):205–244, 1996.
- [8] R.Greenes. *Clinical Decision Support: The Road Ahead*. Academic Press, Inc., Orlando, USA, 2006.
- [9] 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.
- [10] S.Dustdar and W.Schreiner. A survey on web services composition. *International Journal of Web and Grid Services*, 1:1–30, 2005.
- [11] D. Wong, N. Paciorek, T.Walsh, J.DiCeglie, M.Young, and B.Peet. Concordia: An infrastructure for collaborating mobile agents. In *The International Workshop on Mobile Agents*, pages 86–97, London, UK, 1997.
- [12] L. Xiang. A multi-agent-based service-oriented architecture for inter-enterprise cooperation system. In *The International Conference on Digital Telecommunications*, pages 22–32, Silicon Vally, USA, 2007.
- [13] Y.Yamato, H.Ohnishi, and H. Sunaga. Study of service processing agent for context-aware service coordination. In *The IEEE Conference on Service Computing*, pages 275–282, Hawaii,USA, 2008.
- [14] Y.Yamato and H.Sunaga. Context-aware service composition and component change-over using semantic web techniques. In *The IEEE International Conference on Web Services*, pages 687–694, Los Alamitos, USA, 2007.