Noname manuscript No.
(will be inserted by the editor)

Modeling Service Representatives in Enterprise Systems using Generic Agents

Mehran Najafi · Kamran Sartipi

Received: date / Accepted: date

Abstract As a common practice in business enterprise systems, a service provider delegates a human agent to a client site to serve the client. On the other hand, in a computerized business application, enterprise organizations adopt Service-Oriented Architecture (SOA), where an enterprise agent is modeled as a software agent that cannot be transmitted efficiently by service messages. In the proposed approach, we extend the traditional architecture of SOA implementations to support generic and lightweight agents that reside at the client site. These agents, that we call "Service Representatives", can be customized and trained based on the provider generated role description and knowledge to perform their assigned tasks. 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. To indicate the variety of roles that can be done by the service representative, we provide three case studies to show how a local and generic agent can be customized by different providers to personalize financial advice, apply medical guidelines, and verify credit card transactions.

Keywords SOA · Resident Agents · Generic Agents · Autonomous Agents · Knowledge Management · Context-aware Services

Mehran Najafi
Department of Computing and Software
McMaster University
Tel.: +1-905-920-4282
E-mail: najafm@mcmaster.ca

Kamran Sartipi
Department of Computing and Software
McMaster University
Tel.: +1-905-525-9140 ext. 26346
E-mail: sartipi@mcmaster.ca

1 Introduction

Enterprise systems [22] 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 be featured by major non-functional qualities such as: simplicity, flexibility, maintainability, reusability, and decoupling technology from functionality.

Service-Oriented Architecture (SOA) [14] 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 services where each service has a direct relationship with a business functionality of the enterprise. In SOA, *enterprise related tasks* are addressed by interactions between service clients and providers through services. A service provider *registers* its services in a service registry. A service client *inquires* the service registry to receive the description of an appropriate service from a provider to satisfy its needs [15]. Further on, the requester and the selected provider(s) may *negotiate* about the service usage terms [18]. After agreement between client and provider, the client *invokes* the service. Also, different services can be either *composed* to serve a client [8], or *customized* based on a client's context [4].

In real-world, a business provider usually sends its agent or personnel (e.g., as representative, installer, and repairer) to its client site to perform the required services. Accordingly, several organizational units in enterprises (e.g., customer service, dealership, training unit, and delivery unit) require to send or employ agents to serve the clients. Therefore, to provide a comprehensive

model of enterprises, SOA needs to model enterprise agents efficiently. Lack of this model may result in limiting the applications and functionality of SOA based systems.

XML-based web services are the dominant platforms for implementing SOA. A web service is defined by the messages it exchanges. That is, a service client sends a request message to a (web) service provider. Then, the web service processes the request message and replies by a response message. An enterprise agent can be modeled as an autonomic agent [12] that is a computer program which cannot be transmitted by communication messages efficiently.

In this paper, we extend the traditional architecture of SOA implementations to enable enterprise systems to employ generic agents as their service representatives. Instead of sending different agents to the service client, we maintain a lightweight and generic agent at the client site that can be *customized* to act different roles and be *trained* to perform different tasks on behalf of the service provider. The customization and training are performed based on the role description and knowledge which are generated by the service provider and can be efficiently transmitted by the messages. The proposed approach significantly enhances the capabilities of the current SOA services. Since the proposed agent is local to the service client, it can customize service responses based on the client's context; this mechanism reduces the security and privacy concerns by eliminating the need to send client's private information to the provider. Finally, the proposed agent represents a service provider at the client site, therefore, we call this agent *Service Representative* (SR).

The organization of this paper is as follows. Related work is discussed in Section 2. Service Representative is introduced in Section 3. The proposed architecture and its details are discussed in Section 4. Section 5 introduces the developed prototype system. Three case studies in business, health care, and insurance domains are explained in Section 6. The applications and challenges of the proposed approach are discussed in Section 7. Finally, conclusions and future work are discussed in Section 8.

2 Related Work

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

1. **Agents as services**. The Intelligence Service System (ISS) [3] 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 [23] and web service platforms have similar components (registry, descriptor, communication protocol and semantic language), AgWebs architecture [16] is proposed to provide interoperability and interaction between them.
2. **Services as agents**. In ASMF [6] 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. In [27], a role-based architecture is adapted to facilitate the service definition and relationship 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. In [17], during a service composition process, software agents engage in conversations with their peers to agree on the web services that participate in this process. Moreover, agents have been proposed as coordinators for web services. For example, [28] introduces 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. In [26], each role or major function of an enterprise system is considered 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 combined with web services can easily communicate with each other. As another example, a distributed market place is modeled by agents [13]. 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.

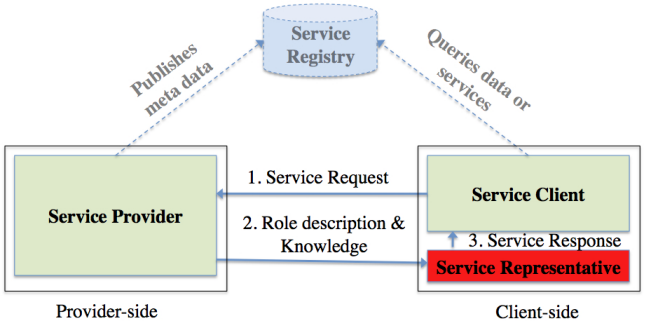

Fig. 1 Proposed extended SOA model. The shaded area represent the non-essential component (service registry) in the SOA model. This model also supports the traditional service invocation where the service provider returns the service response directly to the service client

In the proposed model, we address a new application of collaboration between agents and services. The proposed generic agents are in charge of delivering the functionality of service providers, however, these agents are located at the client site.

Mobile agents can physically travel across a network and perform tasks on 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 [25], Odyssey [24], and Voyager [19] are examples of mobile agent frameworks based on java. Also, mobile agents are suitable to be formally represented using pi-calculus [7]. There are several security and privacy issues to be considered in mobile agent-based computing. 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 considered in these approaches. These issues motivated us to customize generic resident agents as opposed to send mobile agents.

3 Service Representatives

We propose to extend the major components of SOA (service provider, service requester, and service registry) with the service representative, as it is shown in Figure 1. In this section, first we address the limitations of the existing SOA-based technologies to model enterprise services. Then, we introduce the notion of service representative that can be implemented using resident generic agents to facilitate dealing with these limitations.

3.1 Existing Technology Issues

As mentioned earlier, enterprise services are modeled based on message exchanges. A service is typically defined using WSDL technology that represents the request message that the service provider receives and the response message that it generates. This message-based structure imposes limitations on enterprises that aim to use SOA to provide their services. Some of these limitations are listed below.

Functionality limitation. There are several types of services that can not be modeled efficiently by message exchanges, such as:

- *Supervisory service* which is called to control client resources. A set of provider generated messages can not perform this task since it needs an executable platform at the client site that has access to the local resources.
- *Event-triggered service* which is called by a client and the service will wait until a predefined event occurs at the client site. Implementing these services by the message exchange technology requires a permanent connection between the provider and the client.
- *Advertising service* which introduces other services in the enterprise while performing its task. 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 receiving advertisement messages.

Privacy and security issues. Since web services process client requests at the provider site, the client may need to include their personal information in the request messages. This may cause significant privacy and security breaches.

Needs for expertise. As providers pack their responses in the form of messages, the interpretation of these messages is the client's task. So, it is likely that a client lacks enough expertise and knowledge to understand and use the service responses. This situation gets worse as a client has to deal with different providers in different domains.

Service competition. Message-based web service providers are usually passive in dealing with enterprise issues. They register the descriptions of their services into a service registry, then it will be the responsibility of service clients or coordinators to discover and compose those services. Active providers are expected to address the enterprise issues more efficiently. For example, in the case of service discovery, instead of analyzing the service descriptions to find the best service for the client's needs, the candidate services can compete and the client simply chooses the winner.

Stateful services. According to the SOA's requirement, web services should be designed to work in a stateless fashion. However, in some situations message exchange technologies force developers to implement stateful services. For example, efficient service negotiation techniques ask the providers to keep track of a negotiation process initiated by a client.

3.2 Generic Agent as Service Representative

Sending agents as service responses (mobile agents) could facilitate dealing with the above limitations, however the message-based structure of web services does not allow providers to dispatch their agents efficiently. We propose an extension to the existing SOA architecture which utilizes the concept of "*generic agents*" that are resident at the client site and are customizable and trainable for different roles. The proposed architecture requires that the service provider only transfers essential messages to customize and train a generic agent, as opposed to sending the entire agent. Since the agent executes at the client site and has access to the local resources of the client, it can potentially violate the client security and privacy. To prevent this, we limit the power of the agent by restricting the resources that it can access.

The proposed architecture organizes the contents of the communication messages into three segments "*data*", "*information*", and "*knowledge*" [29], where: i) data is defined as raw fact; ii) information is the result of applying knowledge on data; and iii) knowledge is an understanding of how to process data to produce information, based on evidences, experience and insight that can be represented as guideline, decision-flow or patterns of data.

In this context, the service client asks to receive information from the service provider by sending a request message. The service client may receive the resulting information in a response message directly from the service provider or indirectly through the service representative. Therefore, the proposed service representative can be viewed as an agent that works in a knowledge management environment illustrated in Figure 2. The service representative, that is modeled by a resident generic agent, provides the following facilities for the SOA architecture.

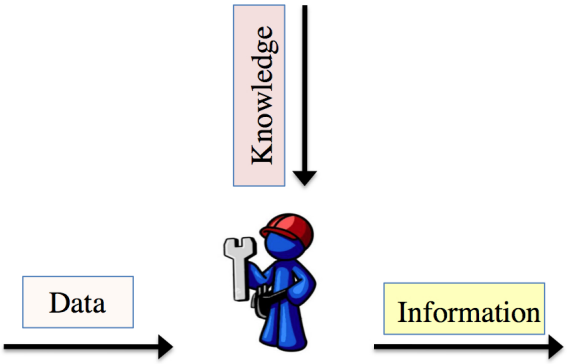

Fig. 2 Relationship among Data, Information, Knowledge and the Service Representative in the proposed model.

More sophisticated functionality. By introducing an executable platform at the client site, service providers can offer innovative services. For example, a service representative can be customized and trained to advertise other provider's services, control local resources or be activated when an event occurs.

More privacy and security. As the service representative has access to local resources at the client site, the client does not need to send its personal data to the provider in order to receive customized services. Moreover, we impose two constraints to preserve the client privacy and security. First, the client determines local resources that the service representative can access to them. Second, the communication between the provider and its agent is one way (from provider to the service representative); which implies that the agent can not return any of the client's resources to the provider(s).

Local and trainable experts. A service provider can train a generic agent to interpret its response messages in the forms that a client can understand. Also, the agent can guide a service client on how to use the service responses.

Active providers. When a generic service representative is customized and trained, it can represent its provider regarding its assigned role in performing the enterprise related tasks. Popular and high-demand tasks can be pre-defined as standard tasks to be dispatched to the generic agents. For example, in the case of service discovery, after customizing generic agents by different providers; they can compete on behalf of their providers to find the best service for a client.

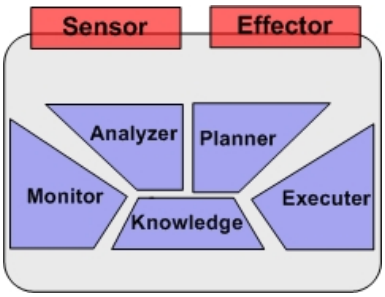

Fig. 3 Structure of an autonomous software agent, called MAPE-K (monitor-analyze-plan-execute over a knowledge base), proposed by IBM [2].

Stateless services. By employing the service representatives, enterprise related tasks can be modeled for stateless services efficiently. In other words, by assigning an agent to each client, providers do not need to keep the state of each request. For example, in a service negotiation scenario, a provider sends essential negotiation skills to the service representative to negotiate with the client about the terms of using the service.

An enterprise agent can be modeled as an autonomic agent, shown in Figure 3, and hence can be defined using a tuple of its components, as below:

- *Sensors*: act as the agent input devices and obtain data from the system.
- *Monitor*: scans the sensed data, generated by the sensors, to extract the relevant data.
- *Analyzer*: analyzes or modifies the monitored data in a way that the agent can use them.
- *Knowledge Base*: contains knowledge sentences that other agent components can use to perform their tasks.
- *Executor*: processes the input data and generates the output as information.
- *Planner*: acts as the brain or controller of the agent that specifies how the executor generates outputs or how and when the knowledge base can be used.
- *Effectors*: act as the agent's output devices.

The *generic service representative* is defined based on its generic components (generic sensors, generic monitor, generic analyzer, generic execute, generic effectors, and generic knowledge base). The agent planner (the only concrete component) is in charge of concertizing the generic components based on the provided role (or role description) and knowledge (or role knowledge) to perform the assigned task. The role description is a list of the tasks, functions, or responsibilities, and role knowledge is the required expertise that the service representative needs to complete the described tasks and responsibilities. The planner transforms a generic service representative into a specific service representative in two phases (customization and training) and then executes the assigned task in the execution phase, as follows.

1. *Customization*. In this phase, the planner sets up the agent configuration (including SR sensors, effectors, and executer) and creates an abstract process in the SR executer based on the role description.
2. *Training*. In this phase, the planner uses the role knowledge to train the customized agent for the assigned role. The role knowledge can be received from the provider or/and extracted from the local knowledge base. Consequently, the abstract process will be completed to perform the specified tasks.
3. *Execution*. In this phase, the customized and trained service representative receives the client's local data (via the sensors and monitors), adapt them (via the analyzer), executes the created process to generate the requested information (by executer), and delivers the information (via the effectors) to the client.

4 Extended SOA Architecture

In this section, we extend the typical architecture of SOA implementation to enable service providers to employ the generic service representative at the client site. The proposed architecture is illustrated in Figure 4 and consists of three main components: service provider, service client, and service representative. The message transmissions among these components are as follows. A service client sends a request message (data) to receive a service response (information) from a service provider. In a simple model of communication (without using the service representative), the provider's information layer receives and processes the request message that contains client data and returns the resulting information back to the client application. This model represents traditional web services. In an agent model of communication (with using the service representative), the client request is processed at both provider and client side as follows.

1. At the provider site, the customization and training layers send a role description and required knowledge to the service representative to process the client data locally to generate the requested information.
2. At the client site, the generic service representative receives the role description and knowledge segments and evolves into a customized service representative. Then, it performs the assigned tasks on the client local data that are available through the communication channel.

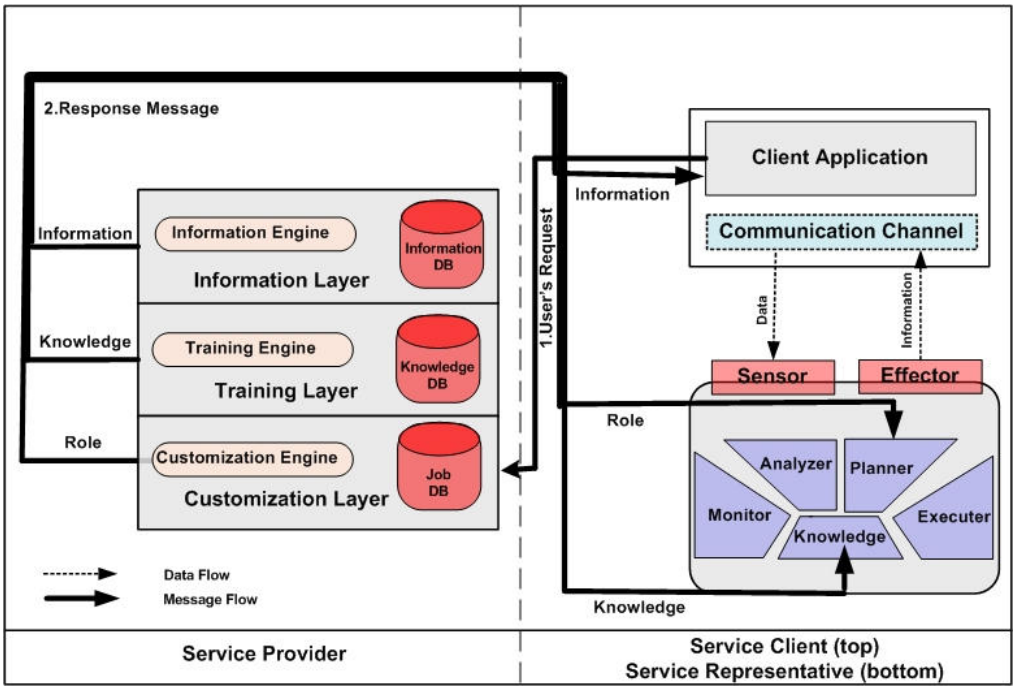

Fig. 4 Proposed architecture. Based on the client's request, the service provider generates a 3-segment response message to customize and train a client-side generic agent as its representative to serve the client.

The specification for each component of this architecture is given in the following subsections.

4.1 Service Provider Model

In contrast to the existing SOA models whose service response messages have only one segment (information), the proposed model introduces service response messages with three segments (role, knowledge, and information). Accordingly, the service provider consists of three layers that are designed to work independently, and each layer is responsible to provide one segment of the response message, as follows.

Customization layer. This layer specifies a role for the generic service representative to customize and perform assigned tasks on behalf of the provider. First, a role (e.g., negotiator, customizer, or adaptor) is assigned to the generic agent. The role can be determined explicitly, i.e., the client specifies it in the request message, or implicitly, i.e., the customization engine predicts it based on the previous similar situations. Then, this layer specifies the agent configuration for the assigned role including the type and specification of the required sensors, effectors, analyzer, and executor. Moreover, the role specification includes a process model describing the order in which a series of steps (called tasks) needs to be executed

The required knowledge for each step of the role process can be provided either locally by the SR knowledge base or remotely by the provider's training layer. Since the service provider can employ the service representative for different roles, the roles configuration and description are kept in the role database. Conceptually, this layer can be viewed as a technical support unit in an enterprise organization that informs a technician their responsibilities about a customer or a product. The layering structure of the proposed model implies that the customization layer should be independent from the knowledge layer. It is based on the fact that technicians are assumed to be knowledgeable when they are assigned some tasks. They only receive the overall task description while their knowledge are provided from other sources, such as: education, training, past experiences, or following strict guidelines.

Training layer. This layer generates the knowledge segment of the response message, based on the knowledge model specified in the customization layer. The knowledge is provided using knowledge representation techniques and is stored in the knowledge base.

Information layer. This layer essentially represents the service provider in the traditional model of SOA, where the service provider receives a request message from a client; processes its data; and returns the result of the operation on data (as information) back to the client via the information segment of the response message. This layer provides the compatibility of the proposed SOA model with the existing model.

Fig. 5 Example of a communication channel.

4.2 Service Client Model

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

Client application. This is a traditional client application that generates and sends request messages to service providers. The client application will receive the information segment of the response message. This information can be either consumed directly or passed to the service representative via the communication channel to be modified by the service representative. The service provider publishes the required communication channel schema for each client-side web service in the service registry using WSDL documents. In order to call a client-side web service, the client application needs to put the client data in the communication channel based on this schema.

Communication channel. This channel consists of a number of ports that are connection links to the internal resources of the client application, as well as the means for the client application to receive the result of the requested task through the service representative. A client grants permission to the service representative to read/write a number of its resources through this channel. The ports can be input, output, or input/output (from the client point's of view). Input ports can be read by the SR sensors and output ports can be written by the SR effectors. One instance of a communication channel is shown in Figure 5.

4.3 Service Representative Model

As mentioned earlier, a generic service representative is transformed into a specific service representative after customization and training phases. The agent then modifies the client's internal resources through the communication channel. A service representative is modeled by an autonomous structure as follows.

- The sensors and effectors are connected to ports in the communication channel.
- The knowledge base contains the internal role knowledge that is pre-loaded by the client or received from external resources. The received knowledge from web services can be stored to relieve web services from sending them each time. Moreover, by storing basic knowledge of a specific domain in the SR knowledge base, we can develop domain-specific service representatives where they can perform the domain relevant tasks efficiently.
- The planner has functions to configure the agent and a process engine to follow different steps of the assigned role.
- The monitor and analyzer receive and convert the sensed input data to a format that is understandable for the agent.
- The executor contains one or more knowledge model instances that are specified in the customization phase and trained in the training phase.

In the execution phase, the service representative performs its assigned role as follows: i) the sensors read client data from the communication channel; ii) the relevant data are extracted by the monitor and the analyzer converts them into a proper format for the service representative; iii) these input data are fed to the knowledge models in the executor; iv) the trained knowledge models will be applied to the data to generate the output results; and v) the results are written back to the communication channel by the effectors.

4.4 Types of Supported Services

The proposed architecture is an extension to SOA, hence it must cover the typical SOA services where there is no need for the proposed service representatives. Moreover, the service representative can work in two separate modes, therefore, the proposed architecture can model three types of services, that are described below.

- **Type 1**. The service response only contains "information segment" that is received by the client application, while the role and knowledge segments are empty. Type 1 includes typical web services provided by traditional service providers that do not need to employ agents in order to serve the clients.
- **Type 2**. The service response contains "role" and "knowledge" segments that are received by the service representative, while the information segment is empty. Based on the role description, the service representative applies the received knowledge to the local client data and provides resulting information as the service response for the client.
- **Type 3**. The service response contains "role", "knowledge", and "information" segments. In this case, two scenarios are possible: (a) the service client uses the received information and the service representative performs its assigned task to provide additional information for the client; (b) the service client redirects the information to the service representative in order to be modified or used during the service representative task execution.

Finally, because web services of Type 2 and Type 3 are executed at the client-side, we call them *client-side web services* as opposed to web services of Type 1 which are executed at the server-side.

5 Prototype System

To evaluate the effectiveness and feasibility of the proposed extend SOA model, we developed a prototype system of the proposed architecture including the service representative as well as the extended service client and provider. This prototype, namely *SR version 1.0*, is developed based on J2EE 1.5 technologies and Apache Tomcat 6.0 application server which can be used by the service developers and clients to develop and invoke the client-side web services, respectively.

The developed service representative has a built-in *Drools* [20] process engine (located in the SR planner component) to execute the process included in the role segment of each service response. The *SR v1.0* uses the Drools rule flow as the process model for the client-side web services. Moreover, *SR v1.0* can receive and understand knowledge sentences that are compatible with PMML (Predictive Model Markup Language) V3 [21], as follows.

- **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 on the client data (*clientData*) is true, the final service response (*serviceResponse*) is obtained by applying the modification function to the received 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 to be executed sequentially. SR executer uses the *Drools* rule-engine to apply the rules to the client data using the forward chaining strategy. In this strategy, a 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. 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. *SR Version 1.0* supports two types of mining models: *neural network* [10] and *decision tree* [9].

  - *Neural network*: includes a network of simple processing elements (called neurons) that can exhibit complex global behaviour, determined by the neurons interconnections and their assigned weights. Learning in neural network involves adjustments to the neurons and interconnection weights. There are two different styles of training that are both supported by *SR Version 1.0*. 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 executer has 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 returns the result.

  - *Decision tree classifier*: is 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 executer contains: i) a decision tree builder; and ii) a decision tree executor to work with decision trees.

Finally, *SR version 1.0* is provided as two Java packages: *ServiceDeveloper* and *ServiceRep* which can be imported into any service provider and client applications as follows.

- **ServiceDeveloper** package: a service developer uses this package to develop a client-side web service graphically using the Drools APIs and widgets. Figure 6 (top) represents a snapshot of the *Client-side Web Service Developer* application, developed based on this package.
- **ServiceRep** package: a client application developer uses this package to generate one instance of the service representative and communication channel.

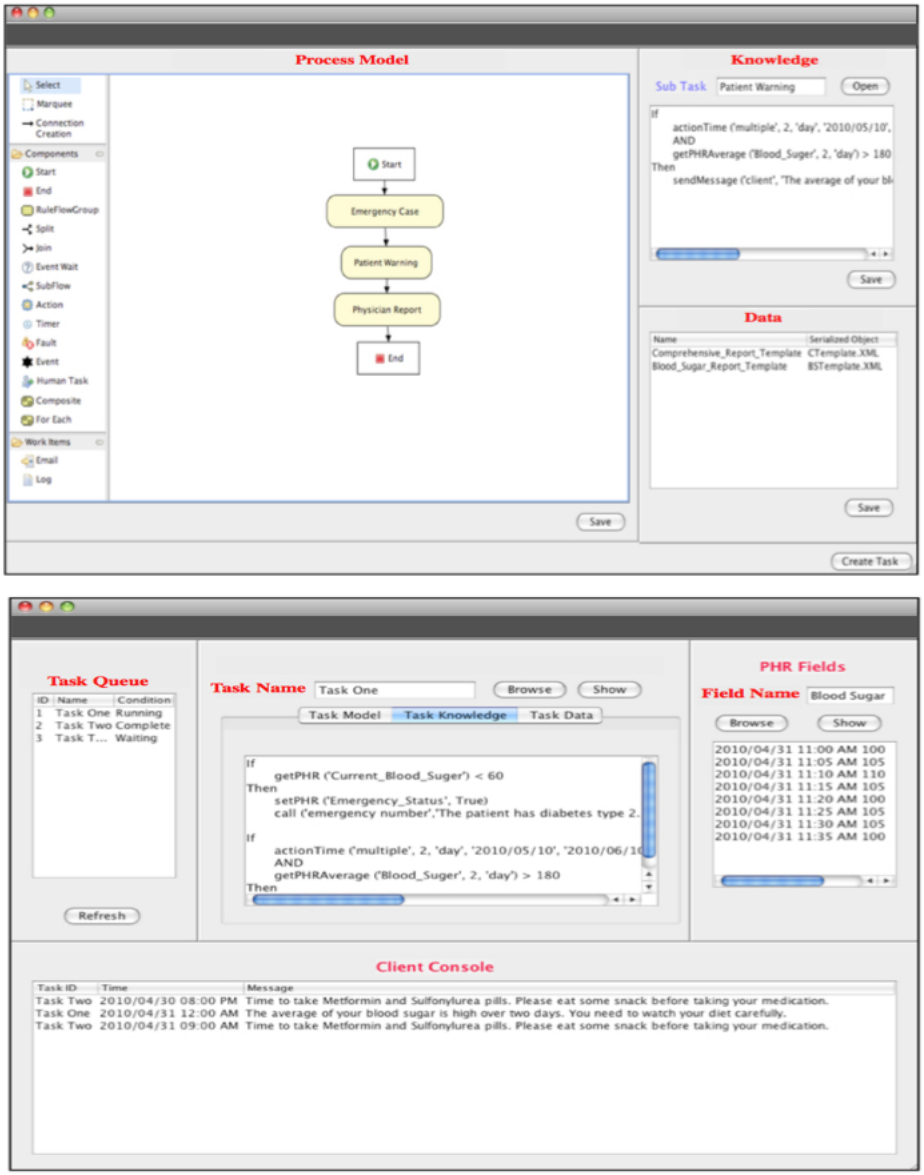

Fig. 6 (Top): snapshot of the "*Client-side Web Service Developer*", used by a service developer to develop a client-side web service. (Bottom): snapshot of the "*Service Representative Manager*", used by a service client to monitor a client-side web service invocation (e.g., a decision support service in this case).

Using the provided APIs, the client application can supply the client data to the communication channel, according to the channel schema that is obtained from the service registry. The communication channel schema is also passed to the service representative instance to configure itself. After configuration, the client invokes the client-side web service from the provider and blocks itself to receive the service response from the service representative and through the communication channel. Figure 6 (bottom) shows a snapshot of the *Service Representative Manager* which uses this package to monitor different phases of a client-side web service invocation.

6 Case Studies

In order to present and evaluate the diverse applications of the proposed service representative, we designed and developed three case studies in different domains: banking, health care, and insurance. Since web services of Type 1 in the proposed SOA model refer to typical web services, we only focus on web services of Type 2 and Type 3, discussed in subsection 4.4. To reduce the redundancy and cover different aspects of the proposed model, similar parts are eliminated in the following case studies.

6.1 Case Study 1: Highly-Secure Financial Adviser

In order to call a context-aware service, a service client shall reveal her contextual information to the service provider or a context manager, while this may violate her 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 the first case study, we present a secure financial adviser in the context of stock market where a service uses the service representative to personalize financial advice without asking the client to send her personal information. To call this web service, the client sends a request to the service provider to receive financial advice and then provides her financial information (client data) to the service representative through the communication channel. After processing the client request, the service provider responds a message with the following components.

- *Role segment*: financial advice customizer.
- *Knowledge segment*: guidelines to personalize general advise based on the client's portfolio.
- *Information segment*: general financial advice.

Case Study Specification. The process of generating financial advice could be very complicated and is out of scope of our discussion. In this case study, we are interested only in the personalization procedure, as follows. 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). However, the client keeps her sensitive information local and private, such as client's financial information (portfolio, and cash). Then, the service provider generates a set of general financial advice (stock buy and sell advice) according to the client preferences.

Each 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>*. A stock buy (or sell) advice recommends the client to have minimum (or maximum) percentage of a specific share in their portfolio. The service provider assigns the role of *advice customizer* to the service representative to personalize the general advice based on the local client's financial information and by performing the following operations.

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

Service Client. The client application sends a request message to receive financial advice. The request message does not contain sensitive financial information of the client. Moreover, the client application supplies its portfolio and cash information into the communication channel (Figure 7), based on the communication channel schema published on the service registry. Finally, the service client receives the final customized advice from the service representative and through the communication channel.

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

Fig. 7 Communication channel schema in the financial adviser case study.

Service Provider.The three layers of the service provider are specified as follows.

1. *Customization layer* specifies the role of financial advice customizer for the service representative (shown in Figure 8). For this purpose, it first assigns a process of applying two categories of rule-based knowledge models to the general advice. Moreover, client's portfolio, holding, and cash information are assigned to the SR sensors and an effector is considered to return the customized advice to the client.
2. *Training layer* encodes the advice personalization knowledge sentences (represented in Figure 9) into the knowledge segment of the response message.
3. *Information layer* is fed by either an automated system or a financial expert who generates financial advice in the specified format in Case Study Specification section, such as the following advice.

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

   This advice is assembled into the information segment of the response message.

Service Representative. The SR planner uses the role segment of the service response to customize the generic service representative to be a financial adviser by performing the following tasks.

1. Generates an abstract process with two sub-processes based on their descriptions in the role segment. This process is placed in the SR executer.

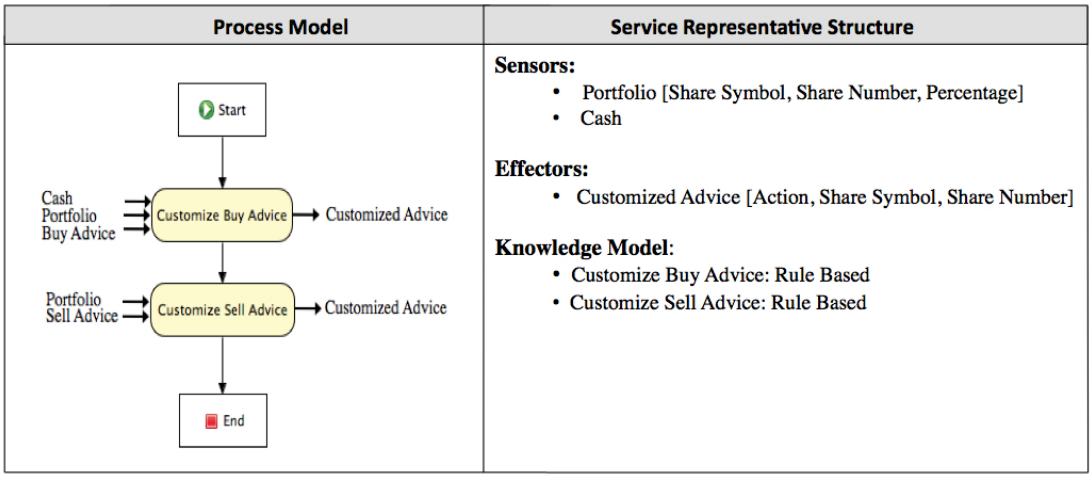

Fig. 8 Role description of the service representative in the financial adviser case study.

Customize Buy Advice	Rule 1	For each b ε buyAdvice
		If (not exist p ε client. portfolio ∧
		p.shareSymbol = b.shareSymbol)
		Then
		requiredCash = b.minPercentage * client. holdingValue;
		shareNumber = min (requiredCash, client.cash) / b.sharePrice;
		customizedAdvice += (" Buy", shareSymbol, shareNumber);

	Rule 2	For each b ε buyAdvice
		If (exist p ε client. portfolio ∧
		p.shareSymbol = b.shareSymbol ∧
		p. percentage < b.minPercentage)
		Then
		requiredCash = (b.minPercentage – p.percentage) * client. holdingValue;
		shareNumber = min (requiredCash, client.cash) / b.sharePrice;
		customizedAdvice += (" Buy", shareSymbol, shareNumber);

Customize Sell Advice	Rule 1	For each s ε sellAdvice
		If (exist p ε client. portfolio ∧
		p.shareSymbol = s.shareSymbol ∧
		p. percentage > s.maxPercentage)
		Then
		shareNo = (p.percentage - s.maxPercentage) * client. holdingValue / s.sharePrice;
		customizedAdvice += (" Sell", shareSymbol, shareNumber);

Fig. 9 Advice personalization knowledge.

2. Assigns a rule-based knowledge model to each of the generated sub-processes.
3. Connects the SR sensors to the Read ports of the communication channel to provide client personal data as inputs for the knowledge models.
4. Connects the SR effectors to the Write port of the communication channel to return the customized advice.

In the training phase, the planner loads each rule-based model by the received customization knowledge from the service provider. Finally, in the execution phase, the SR executer runs the generated process where in each step of this process, it applies the corresponding rules to customize the general advice.

This web service can be more sophisticated if the SR analyzer is involved to convert the client data into a proper format for the knowledge models. For example, if the client uses a different currency than the general advice, the agent analyzer can exchange their currency before applying the models. Finally, the service representative stores the customization knowledge into its internal knowledge base to relieve the service provider from sending them each time.

The message exchanges between the service provider and the service client is shown in Figure 10. The XML schema of the request and response messages are displayed in Figures 11, and 12. By using these schemas, we developed WSDL description of this web service and then we used a *top-down approach* to implement the body of this web service.

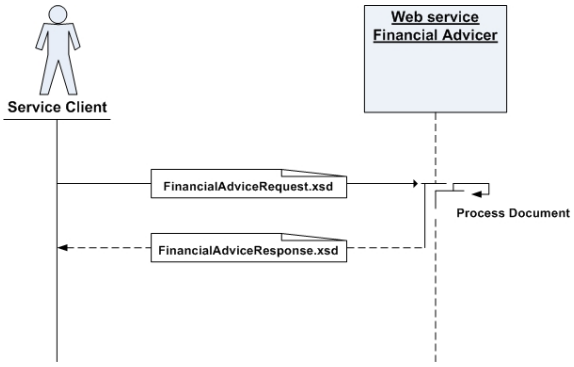

Fig. 10 Message exchanges between the secure financial adviser web service and the service client.

```
<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>
```

Fig. 11 Financial adviser request message schema (FinancialAdviserRequest.xsd).

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

In this section, we present 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 the personal health information, known as Personal Health Record (PHR) or Electronic Medical Record (EMR). A typical CDSS requires that the patients send their information that may violate their privacy and security. Moreover, as a medical center calls the same CDSS for different patients, transferring PHRs over the network increases the network traffic significantly. Based on the proposed model, a CDSS can employ the service representative to apply its medical guidelines to the local PHRs to improve the security and efficiently.

We modified a CDSS that is called Vascular Tracker (VT) [1] to work based on the proposed model, as follows. A physician (service client) supplies the patient's PHR information (client data) into the communication channel and sends a request message to receive medical advice. The CDSS (service provider) response message contains three segments as follows:

- *Role*: clinical decision support agent.
- *Knowledge*: medical guidelines.
- *Information*: none.

This service is categorized as Type 2 of the proposed services and uses mining models as its knowledge models. Different parts of this system are described below.

PHR <Read>	Medical Guideline for patient <Write>	Medical Guideline for MD <Write>

Fig. 13 Communication channel schema in the agent-based CDSS.

Case Study Specification. COMPETE III Vascular Tracker (C3VT) [1] is a decision support system that assists physicians to observe and ideally control patient's different risk factors within the domains of cardiovascular, diabetes, hypertension, and dyslipidemia diseases. C3VT's database contains a large body of medical guidelines collected using a methodology known as evidence-based practice. The clinical algorithms are so fine-tuned that cover different cases of most individual patients and is confidently used by a large group of physicians. The VT 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 the result, each medical guideline generates recommendation messages for both physicians and patients. Moreover, VT 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 caller to use this CDSS. In this case study, we use the service representative to apply the medical guidelines (received them from VT) to a local PHR at the client site.

```
<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>
```

Fig. 12 Financial adviser response message schema (FinancialAdviserResponse.xsd).

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 VT supported diseases that apply for the patient. The client supplies the patient's information into the communication channel (Figure 13), based on the VT schema published on the service registry. There are also two ports of the communication channel that allow the patient and physician to receive the medical recommendations and alerts from the service representative.

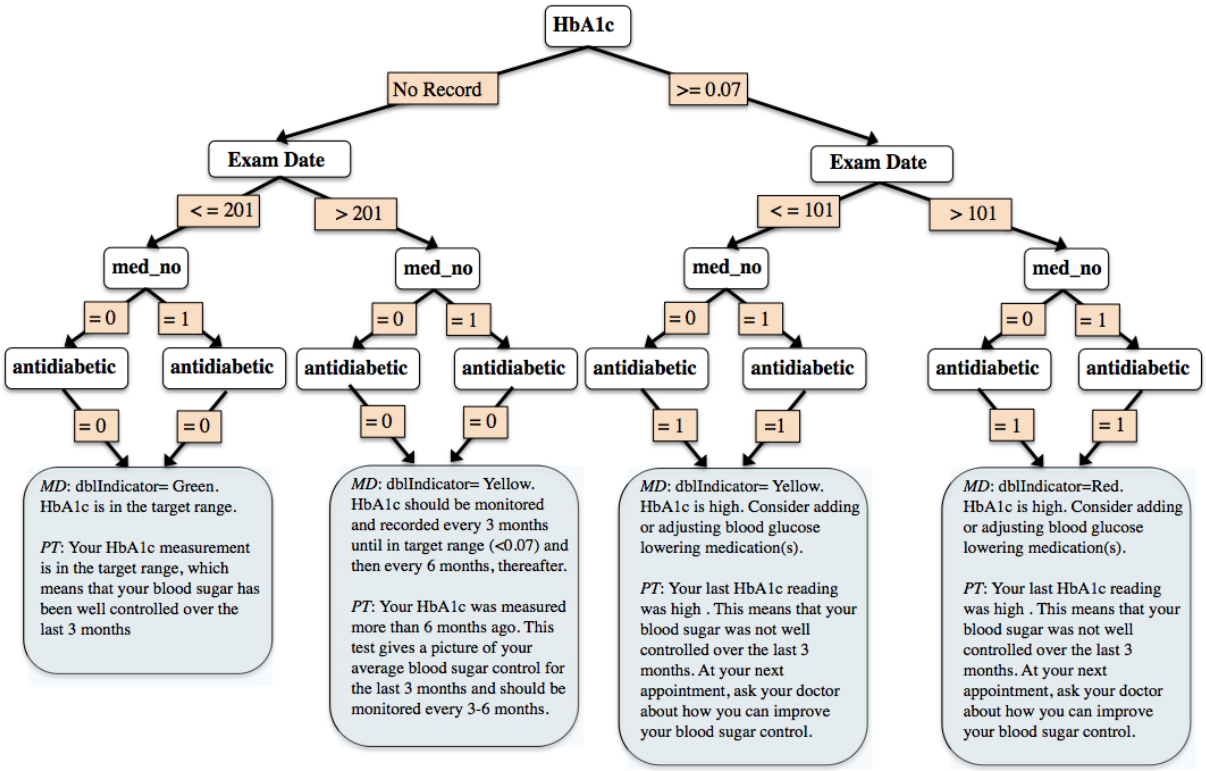

Fig. 14 Decision tree representing a VT medical guideline that corresponds to one step of the process described in Figure 15.

Service Provider. The developed service provider is a modified version of the VT CDSS where its three-layer architecture enables VT to offer high privacy for their clients. Since this service is of Type 2, the data is provided to the SR by the client application and therefore the information layer is not required. The customization and training layers of the service provider are specified as follows.

1. *Customization layer* specifies the role of CDSS agent for the service representative based on the received VT diseases category from the client. A SR role description to provide recommendations in the case of diabetic patients is displayed in Figure 15. The corresponding process defines a sequence of medical guidelines that should be applied to the relevant patient's PHR. Moreover, this layer defines the SR configuration to perform this role as follows. It assigns a sensor for the relevant patient's PHR and two effectors for the patient and physician recommendations. Finally, this layer specifies the required knowledge model in each step of the process that includes the type of the 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.
2. *Training layer* provides the specified medical guidelines in the customization layer where each guideline is encoded as a decision tree. A corresponding decision tree to a set of VT medical guidelines is displayed in Figures 14. This guideline gives recommendations to both patient and physician about the result of a blood test (Hb1Ac) with considering three patient PHR fields. The decision tree parameters including the decision and split nodes information (i.e., model content) are serialized into the knowledge segment of the response message.

Service Representative. The planner customizes the generic SR by connecting the sensors and effectors to the corresponding ports of the communication channel and instantiating a process with the corresponding sub-processes in the SR executer. To complete the customization phase, the planner assigns one decision-tree builder object to each sub process. Also, the received knowledge is stored in the SR knowledge base that can be reused in the next service calls. In the training phase, each decision-tree is reconstructed based on the received knowledge to represent an executable medical guideline at the client site. Finally, the executer applies each decision tree to the patient's PHR and the outcome (recommendations) is written to the communication channel.

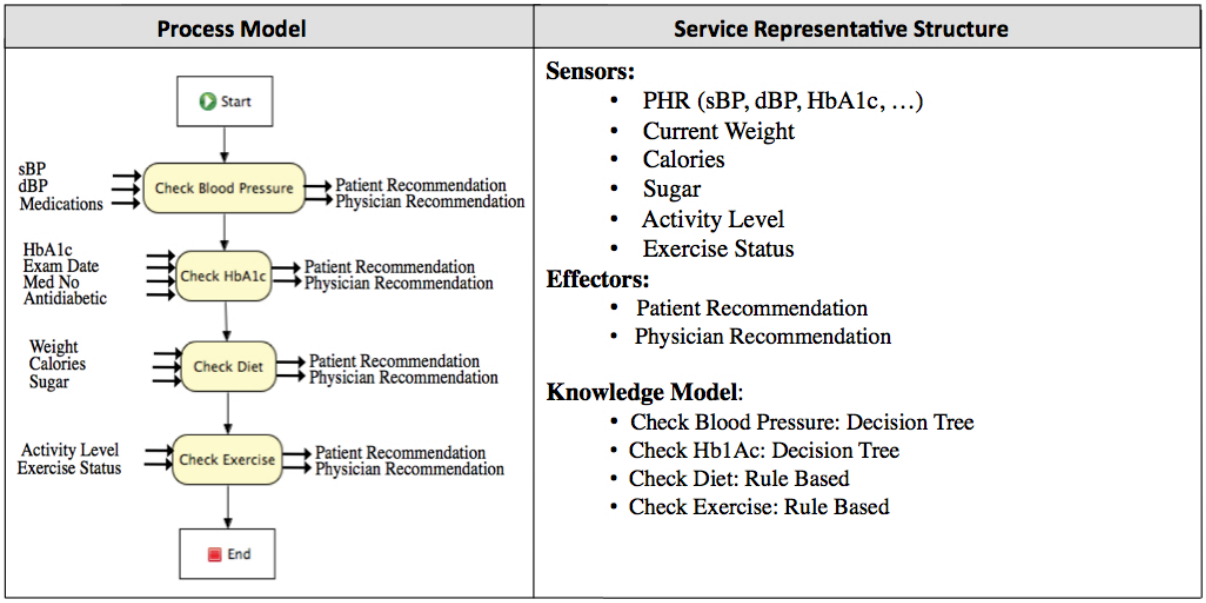

Fig. 15 Role description for the service representative to perform as a CDSS agent.

6.3 Case Study : Customizable Credit Card Fraud Detector

In this section, we present a case of fraud detection in the context of credit card transaction systems. The legal or fraud patterns in the 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 proper and accurate for small and medium size organizations such as a bank or an insurance company. In this case study, we are interested in 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 site, 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 a fraud detector service. The fraud detector service responds with a message containing three segments as follows:

- *Role*: credit card fraud detector.
- *Knowledge*: symbolic fraud detection model and guidelines to build a local numerical fraud detection model.
- *Information*: none.

In this case study, we use both the *rule-based* and *mining-model* knowledge to train the service representative. Different parts of this system are described below.

Case Study Specification. Each transaction is represented as a tuple x of features ($x =< x_1, ..., x_n >$). Where, the features can be symbolic (e.g., type, address) or numerical (e.g., time, money). Consequently, the symbolic and numerical fraud detectors operate on symbolic and numerical features, respectively. Two metrics are usually used to evaluate a fraud detector system as follows: *precision* indicating the number of found fraud transactions relative to the total tested transactions; and *confidence* indicating the accuracy of the method. While, the symbolic model offers high precision, the numerical model yields higher confidence. A sequential combination of these models is reported in [5] to provide both high precision and confidence. Instead of applying a general fraud detector model to a target transaction at the provider site, the proposed approach uses the service representative agent to customize and apply a fraud model to the local transactions at the client site.

Service Client. The client application connects a read port of the communication channel to its database containing the log of the collected local transactions (training data). The target transactions to be checked for fraud (testing data) are also supplied into a read port of the communication channel. There is also a port that the service representative writes the results of the local transactions verification for the client. The structure of the communication channel is shown in Figure

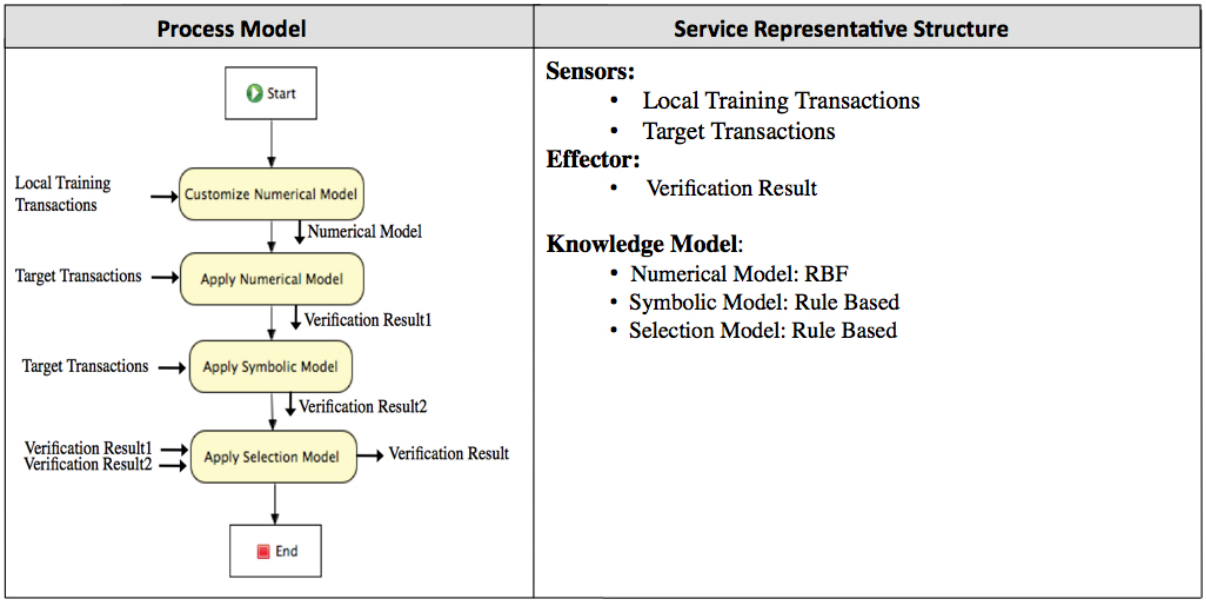

Fig. 16 Role description for the service representative in the customizable fraud detector case study.

Apply Selection Model	Rule 1	If (verification result2 = Legeal) Then verification result = Legal
	Rule 2	If (verification result1 = Fraud and verification result2 = Fraud) Then verification result = Fraud
	Rule 3	If (verification result1 = Legal and verification result2 = Fraud) Then verification result = Legal

Fig. 17 Required knowledge for Model III, in the customizable fraud detector, to combine the verification results obtained by a symbolic and a numerical approach.

18. Therefore, the client application only sends a request message to the service provider which does not include local transaction information.

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

Fig. 18 Communication channel schema in the customizable credit card fraud detector case study.

Service Provider. The service provider is a modified version of the sequential fraud detector presented in [5]. Similar to the second case study, this web service is categorized as Type 2 and its required data is supplied solely by the client. The customization and training layers of this service provider are specified as follows.

1. *Customization layer* defines a role for the service representative to customize a numerical model using client data and then apply this model. Moreover, the SR is asked to reconstruct a symbolic model from the received model parameters from the service provider and then apply this model to the local transactions. The final verification will be obtained by a selection model. Figure 16 illustrates the role description where the assigned process includes one mining and two rule-based knowledge models as follows.
   - *Model I* is an incremental Radial Basis Function (RBF) model to represent a numerical fraud detector. This model is generated and customized at the provider and client site, respectively.
   - *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. *Training layer* generates or extracts the model content for each specified model in the customization layer, as follows.

- *Model I*: is initiated based on the provider training transactions. In this case study, each training instance is a tuple of $(x_1, x_2, ..., x_8, y)$ 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 put in the knowledge segment of the response message.

- *Model II*: is a number of if-then-else rules that represent the relations between the symbolic features and fraud. These rules can be obtained based on the generalization techniques described in [5] and is shown in Figure 19. In this technique, fraud transactions are compared with each other to find the 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 are listed in (Figure 17), where the decisions for fraud by the symbolic model are checked additionally by the numerical model to increase confidence and decrease the number of false alarms.

Service Representative. After setting the SR configuration in the customization phase, the SR planner trains the three specified knowledge model as follows. The numerical model (Model I) is initially built from the received model parameters and then it is completed by the client training transactions received from the SR sensors. Moreover, the symbolic model (Model II) and the selection model (Model III) are loaded with the received rules. In the execution phase, the service representative applies the customized numerical model and the symbolic model to the local transactions and finally the adjusted result that is obtained by applying the third model is written back into the communication channel to be used by the client.

6.4 Evaluation

To compare the proposed client-side web services with the traditional server-side web services, we developed an equivalent traditional web service for each of the described case studies, as follows.

1. A financial adviser web service that takes client's portfolio and cash information and returns personalized advice to the client.
2. A clinical decision support service that takes patient's PHR information; applies the vascular tracker guidelines; and returns recommendations for both the patient and the physician.
3. A customizable credit card fraud detector service that takes the transaction records stored in the client database as well as the target transaction and returns the verification result to the client.

As our evaluation metrics, we used the QoS parameters 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.

The traditional web services (Type 1) are differentiated from the proposed services (Type 2 and Type 3) by the platform where the client data are processed. While the former integrates all the processing at the server platform, the latter distributes the 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 the 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 are independent of client-side or server-side processing of client data such as accessibility, security, accuracy, and availability. Then, we considered three service parameters as our evaluation criteria: *service message size*, *service response time*, and *client privacy*, because they are representative for QoS comparison of client-side and server-side web services.

Client privacy is defined as the client ability to keep her 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 site. The comparison results are illustrated in Table 1.

Table 1 Client privacy comparison results.

Privacy	**Proposed Web Service**	**Traditional Web Service**
Case Study 1	√	× (revealing financial information)
Case Study 2	√	× (revealing PHR information)
Case Study 3	√	× (revealing credit card information)

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	*	*	*	*

Fig. 19 Credit card fraud patterns reported in [5]. Each column represents one symbolic transaction feature.

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

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

The 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 that implies the MS is independent of the size of the client data. Table 2 illustrates the MS comparison of the traditional and proposed web services for the described case studies.

Table 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, the traditional and proposed web services represent compatible Message Size 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 her 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 the traditional web services, the TMS is equal to multiplication of Message Size by the number of service calls. While in the proposed web services the received knowledge is stored in the SR knowledge base that results in reducing the size of the response messages. Figure 20(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 Process 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 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, service providers use more powerful CPUs, traditional approaches have less process time. On the other hand, the proposed web services require smaller messages results in less network time.

For this case study, we obtained the process time, $P(s)$, for the three case studies using a 2.4 GHZ dual-core CPU. Moreover, we assumed the service provider has a CPU that is twice faster than the service client. Finally, there is a 128 KByte/Sec link connects the service client to the service provider.

Table 3 shows the Response Time comparison between the proposed and the traditional web services. These results show the proposed approach overcomes the traditional approaches when the client data grows.

Table 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

Similar to the Total Message Size metric, we compared the Total Response Time (TRT) of the proposed and traditional web services in the context of these case studies. This comparison, that is illustrated in Figure 20 (bottom), confirms that client-side processing of client data improves the TRT.

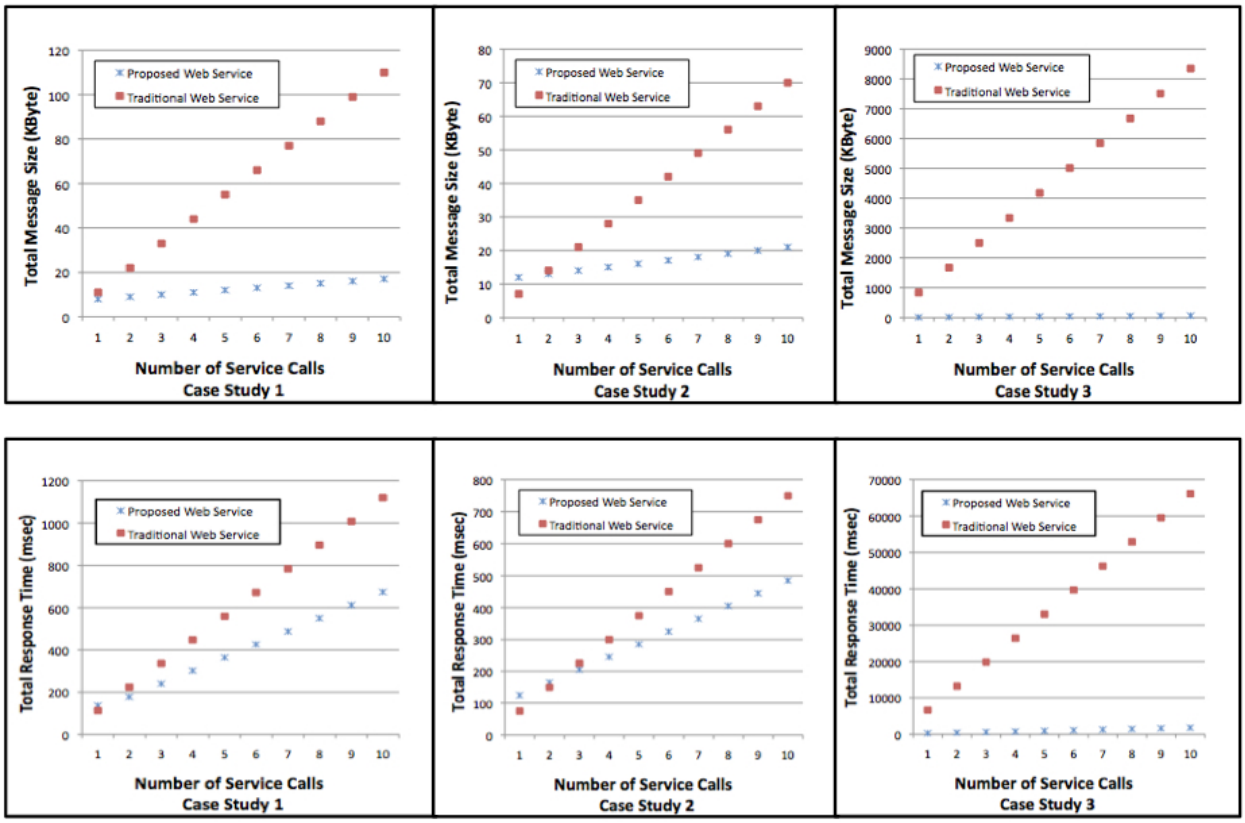

Fig. 20 (Top) Total Message Size and (bottom) Total Response Time comparisons of the traditional and the proposed web services where they implement the described case studies.

7 Discussion

The proposed client-side web services are differentiated from the traditional server-side web services as they can process client data locally using local and generic agents. In this section, we list a few important issues such as the applications and challenges of the proposed web services.

7.1 Applications

The proposed web services do not intend to replace the traditional web services. However, web services can be developed efficiently and securely using service representatives in the following cases.

1. *Context-aware services*, where the services operate according to the available contextual information from the environment. If a context-aware web service can be modeled with a pair of (general service response, customization knowledge), it is eligible to be delivered by the service representative. Therefore, the privacy and security aspects of these services will be improved (Case Study 1).
2. *Sensitive and confidential client data*, where these data should be processed locally (Case Study 2).
3. *Large volume client data*, where sending these data to a service provider requires large messages (Case Study 3).
4. *Dynamic environments*, where the client's context is changing over time. If the context (e.g., location) is changing frequently, a traditional service must be called for each change which increases the network traffic as well as the service cost. In contrast, the SR utilizes the role knowledge to generate dynamic service response for each change of the context.
5. *Dynamic services*, where the provider's knowledge is changing over time. The SR enables providers to separate the required knowledge from the service implementation and facilitates change management.

The proposed web services are described by WSDL documents, which can be stored in XML repositories. A WSDL description for a traditional web service includes what the web service does, how it is accessed, where it is located, and name and type of the service parameters. The traditional WSDL documents are sufficient enough to describe the proposed services where the service parameters are divided into two parts: *local* that are used

by the service representative and *remote* that are sent to the service provider to be processed remotely. As a result, the web services which employ service representatives to process client data can be discovered both statically (at design time) or dynamically (at run time) using existing WSDL-based approaches.

Finally, the proposed client-side web services can also be composed with the traditional server-side web services using BPEL models. To invoke a client-side web service, a BPEL process first calls the web service and sends its remote parameters. When it receives the service response message which includes the role description and the required knowledge and information, it will forward it to the client side to be executed by the service representative.

7.2 Challenges

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

- **Service Provider Privacy**: Required knowledge for the service representative can be enterprise assets and resources that revealing them may violate the enterprise privacy. To prevent this security vulnerability, a service provider can use one of the following techniques.
  1. Enterprise knowledge can be divided to be applied locally (at the provider side) by the service or externally (at the client side) by the service representatives. Therefore, the critical knowledge (e.g, market analysis in Case Study 1) remains at the service provider, while the non-critical knowledge (e.g., advice customization guidelines in Case Study 1) will be sent to the service representative.
  2. The service client only receives the service response from the service representative. Therefore, the client does not have access to the transferred knowledge between the service provider and representative. Consequently, encryption techniques can be used for data transmissions between a service provider and representative to improve the enterprise privacy.

- **Service Client Adaptation**: To call the proposed web services, the service client is required to install a generic service representative with the corresponding communication channel, which increases the client side complexity. However, the current version of the service representative (*SR version 1.0*) needs a few megabytes (about 3 Mbytes) hard disk space and offers reasonable computing speed (discussed in Evaluation section). Moreover, after the generic service representative installed, the service client can invoke different client-side web services.

- **Testing**: The service representative executes a process on behalf of the service provider at the client site. 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 is required which includes test cases for different client's platform and context. Moreover, the interaction between the client application and the service representative should be evaluated using proper test cases. Finally, evaluation techniques for the Rich Internet Applications (e.g., Java Applets or Microsoft Silverlight) [11] can be useful for testing the client-side web services.

- **Interoperability**: Interoperability of service client and provider is another challenge in using the client-side web services. Traditional web services force service clients to send service parameters that are understandable for the service provider. However in the proposed approach, the service representative uses its analyzer component to convert the client data into an understandable data format for the service representative which may increase the interoperability issues. To support different types of client data, the service provider is required to send the corresponding conversion functions (as the knowledge) to the service representative to be applied by the SR analyzer.

8 Conclusions and Future Work

In this paper, we presented a novel model for SOA based systems that enables enterprise organizations to delegate their agents to operate on the client platform. According to the proposed model, a generic and 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, we also proposed an architecture that introduces an executable platform at the client site for service providers which enhances the privacy and security aspects of web services. In addition to existing web services, the proposed approach models two novel types of services.

Different types of web services are complementary and a business service can be implemented based on one or more of these types. As a future work, we will define cost functions for each of the involved factors (e.g., testing, response time, interoperability, and complexity) to guide service developers about the proper type for each business service. Moreover, we intend to assign the service representative more SOA relevant roles such as *negotiator* to develop new approaches for SOA related tasks. The proposed service representative can be employed by collaborating service providers to perform a composite role at the client-side. This motivates us to extend our model to introduce the concept of client-side service composition. Finally, we are also working on a web service extension that supports service representatives directly.

Acknowledgements The authors would like to thank COMPETE group for the use of their Vascular Tracker (VT) materials for our experiments.

References

1. Compete III Vascular Tracker website. http://www.competestudy.com/
2. An architectural blueprint for autonomic computing. Tech. rep., IBM Corporation (2004). 4th eddition
3. Alessandrini, M., Lippe, W., Nuesser, W.: 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, pp. 496–499. Sydney, Australia (2008)
4. Baldauf, M., Dustdar, S., Rosenberg, F.: A survey on context-aware systems. International Journal of Ad Hoc and Ubiquitous Computing **2**(4), 263–277 (2007)
5. Brause, R., Langsdorf, T., Hepp, M.: Neural data mining for credit card fraud detection. In: ICTAI '99: Proceedings of the 11th IEEE International Conference on Tools with Artificial Intelligence, p. 103. IEEE Computer Society, Washington, DC, USA (1999)
6. Cheng, Y., Leon-Garcia, A., I.Foster: Toward an Autonomic Service Management Framework: A Holistic Vision of SOA, AON, and Autonomic Computing. IEEE Communications Magazine **46**(5), 138–146 (2008)
7. D.Sangiorgi, D.Walker: PI-Calculus: A Theory of Mobile Processes. Cambridge University Press, New York, NY, USA (2001)
8. Dustdar, S., Schreiner, W.: A survay on web service composition. International Journal of Web Grid Services **1**(1), 1–30 (2005)
9. Han, J., Kamber, M.: Data mining: concepts and techniques. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (2000)
10. Haykin, S.: Neural Networks and Learning Machines. Prentice Hall (2008)
11. H.Raffelt, T.Margaria, B.Steffen, 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, pp. 1–7. ACM, New York, NY, USA (2008)
12. Huebscher, M.C., McCann, J.A.: A survey of autonomic computing—degrees, models, and applications. ACM Comput. Surv. **40**(3), 1–28 (2008)
13. Khalili, A., Badrabadi, A., Khoshalhan, F.: A Framework for Distributed Market Place Based on Intelligent Software Agents and Semantic Web Services. In: IEEE International Congress on Services Part II, pp. 141–148. Hawaii,USA (2008)
14. Krafzig, D., K.Banke, D.Slama: Enterprise SOA: Service Oriented Architecture Best Practices. Prentice-Hall (2005)
15. Le, D., Eck, S.A., Cao, T.: A Survey of Web Service Discovery Systems. International Journal of Information Technology and Web Engineering **2**(2), 65–80 (2007)
16. Lee, S., Choi, K., H.Shin, D.Shin: Ag Webs: Web Services based on Intelligent Agent Platform. In: The 9th International Conference on Advanced Communication Technology, pp. 353–356. Phonex, Korea (2007)
17. Maamar, Z., Mostefaoui, S., 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)
18. P.C.K., H., H.Li, Jeng, J.: WS-Negotiation: an overview of research issues. In: The 17th Annual Hawaii International Conference on System Sciences, pp. 10–18. Hawaii, USA (2004)
19. Pham, V., Karmouch, A.: Mobile software agents: An overview. IEEE Communications Magazine **36**, 26–37 (1998)
20. Proctor, M., Neale, M., P.Lin, M.Frandsen: Drools documentation. Tech. rep., JBoss.org (2008)
21. Raspl, S.: PMML Version 3.0 - Overview and Status. In: Proceedings of the ACM Workshop on Data Mining Standards, Services and Platforms, pp. 18–22. Philadelphia, USA (2004)
22. Soja, P., Paliwoda-Pekosz, G.: What are real problems in enterprise system. Industrial Managment and Data Systems **109**(5), 610–627 (2009)
23. Sycara, K., Paolucci, M., Soundry, J., Srinivasan, N.: Dynamic discovery and coordination of agent-based semantic Web service. IEEE Internet Computing **8**(3), 66–73 (2004)
24. Wong, D., Paciorek, N., Moore, D.: Java-based mobile agents. Commun. ACM **42**(3), 92–ff. (1999)
25. Wong, D., Paciorek, N., T.Walsh, J.DiCelie, M.Young, B.Peet: Concordia: An infrastructure for collaborating mobile agents. In: MA '97: Proceedings of the First International Workshop on Mobile Agents, pp. 86–97. Springer-Verlag, London, UK (1997)
26. Xiang, L.: A Multi-Agent-Based Service-Oriented Architecture for Inter-Enterprise Cooperation System. In: Second International Conference on Digital Telecommunications, pp. 22–32. Silicon Vally, USA (2007)
27. Xu, B., Yang, X., Shen, Y., L.Shanping, Ma, A.: A role-based SOA architecture for community support systems. In: International Symposium on Collaborative Technologies and Systems, pp. 408–415. Irvine,USA (2008)
28. Y.Yamato, H.Ohnishi, Sunaga, H.: Study of Service Processing Agent for Context-Aware Service Coordination. In: IEEE International Conference on Service Computing, pp. 275–282. Hawaii,USA (2008)
29. Zins, C.: Knowledge map of information science . Journal of the American Society for Information Science and Technology **58**(4), 526–535 (2007)