A QoS-Aware Decision Model for Web Service Development: Server-side Data Services or Client-side Task Services

Mehran Najafi, Kamran Sartipi and Norman Archer
McMaster University
Hamilton, ON, Canada
{*najafm, sartipi, archer*}*@mcmaster.ca*

Abstract

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

1 Introduction

Service-Oriented Architecture (SOA) [1] is a high-level and technology-independent concept that provides architectural blueprints for enterprise systems. Moreover, web services can provide web-accessible programs and devices that have been widely promoted in cloud environments as platforms for the smart Internet [7]. The smart Internet (also called the Personal Web) is a candidate for the next Internet generation, which is shifting towards a more user-centric and task-oriented network.

SOA based architectures focus on dividing the enterprise application layer, where its components (as services) have a direct relationship with the business functionality of the enterprise. *Data Services*, representing typical web services, process client data completely at the server (provider) side. However, server-side processing of web services requires transferring client data to the service providers, which may cause data privacy violations, security breaches, or network traffic overloads. Moreover, in the real-world business domain, an enterprise organization usually sends an agent or other personnel (e.g., a representative, installer, maintainer, or trainer) to the client site to deliver services locally. To address this issue, we have proposed *Task Services* [13] which are web services with the capability of processing client data at the client side using a generic software agent called the *Service Representative*. Then, a task service performs the required server-side processing and defines a *task* to customize the generic service representative to perform the client-side processing at the client site. Figure 1 shows the extended SOA model that supports task services.

Server-side data services and client-side task services are complementary, and each has advantages and disadvantages that must be considered in their selection. Client-side task service applications include, but are not limited to, the following cases.

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

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

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

There is not always a clear boundary between service types and a web service can be categorized in more than one category. Therefore, service developers have to decide whether a web service should be designed and developed as a data service or as a task service. In this paper, we propose a decision model to guide service developers when deciding about the proper type of web service. Based on the decision model, each service is modeled by a business process and the service developer is given an objective function to compare the data service and task service versions of the web service. The definition of the objective function is based on quality of service (QoS) parameters.

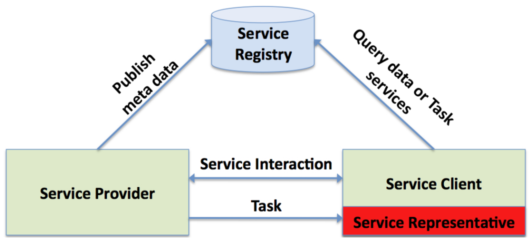


Figure 1: Proposed extended SOA model, where "Service Representative" processes client data based on an assigned "Task" from the service provider.

The organization of this paper is as follows. Data and task services are introduced in Section 2. The proposed decision model is discussed in Section 3. A case study of the decision making process is presented in Section 4. Section 5 discusses work that is related to our approach. Finally, conclusions and future work are discussed in Section 6.

2 Business Services

An enterprise system needs to provide both server-side and client-side web services to model actual services in the business domain. We have proposed to categorize web services into data and task services. Figure 2 shows a comparison between data services and task services.

2.1 Data Service

This represents a typical web service in the current SOA model where the service processes the client's data and resources completely at the server site. Consequently, a data service includes only server-side processing and returns a service response in the form of data that will be consumed directly by the client. In other words, a data service receives *remote service parameters* and returns *remote service responses*, with reference to the service client.

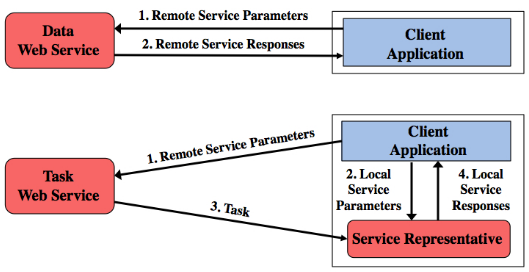


Figure 2: Comparison of Data and Task Services. "Remote" and "Local" are defined from the viewpoint of the service client.

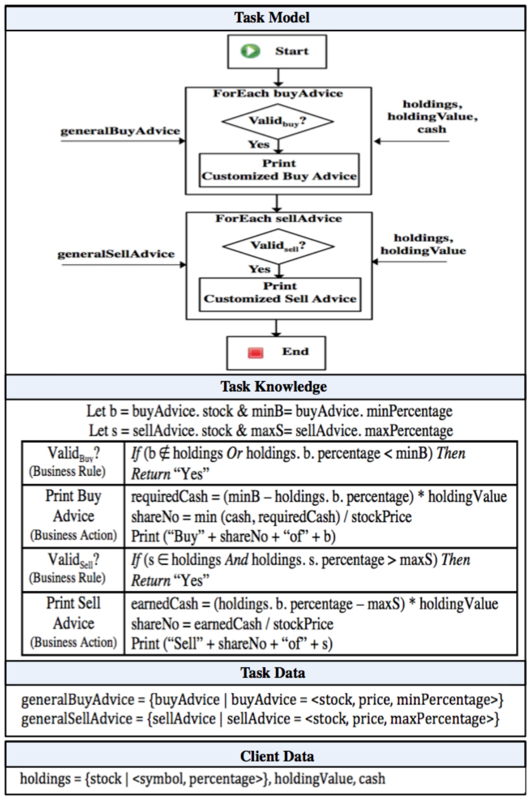
2.2 Task Service

In [13] and [10], we introduced the concept of *Task Services* as web services with the capability of processing the client's data and resources partially or completely at the client site. A task service performs the required server-side processing and then it defines a *task* including the client-side processing to be performed by the service representative at the client site. The service representative is a generic client-side software agent with a built-in process engine that can be employed by different service providers to perform different task services. The service representative requires both task logic and task data to perform client-side processing that can be provided from different sources. We propose to model a task with the following components.

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

- *Task Model* specifies the control flow and data flow among sub-tasks using an abstract Business Process Model (BPM).
- *Task Knowledge* provides the required logic for each sub-task. This includes both descriptive knowledge such as Business Rules (BR) and procedural knowledge such as Business Actions (BA).
- *Task Data* provides the server-side data that are required during the task execution, stored in Business Objects (BO).

Task components are messages that can be transmitted efficiently over the network. The service representative uses the received task



Task Knowledge

Let b = buyAdvice. stock & minB= buyAdvice. minPercentage
Let s = sellAdvice. stock & maxS= sellAdvice. maxPercentage

Valid$_{Buy}$? (Business Rule)	*If* (b $\notin$ holdings *Or* holdings. b. percentage < minB) *Then* *Return* "Yes"

Print Buy Advice (Business Action)	requiredCash = (minB – holdings. b. percentage) * holdingValue
	shareNo = min (cash, requiredCash) / stockPrice
	Print ("Buy" + shareNo + "of" + b)

Valid$_{Sell}$? (Business Rule)	*If* (s $\in$ holdings *And* holdings. s. percentage > maxS) *Then* *Return* "Yes"

Print Sell Advice (Business Action)	earnedCash = (holdings. b. percentage – maxS) * holdingValue
	shareNo = earnedCash / stockPrice
	Print ("Sell" + shareNo + "of" + s)

Task Data

generalBuyAdvice = {buyAdvice | buyAdvice = <stock, price, minPercentage>}
generalSellAdvice = {sellAdvice | sellAdvice = <stock, price, maxPercentage>}

Client Data

holdings = {stock | <symbol, percentage>}, holdingValue, cash

Figure 3: Financial adviser task message and the corresponding client data.

components and performs the client-side processing of the task service on the *local service parameters* to provide *local service responses* for the service client. The following example illustrates the different applications of data and task services.

- A typical financial adviser data service asks for the client's financial information in order to provide personalized advice.

- A financial adviser task service generates a set of general financial advice, e.g., stock buy and sell advice (server-side processing), according to the client's preferences (remote parameters). Moreover, it defines a task for the service representative (client-side processing) indicating the required procedure (task model) and guidelines (task knowledge) to customize the general financial advice (task data) based on the client's personal information (local parameters). Related task components are shown

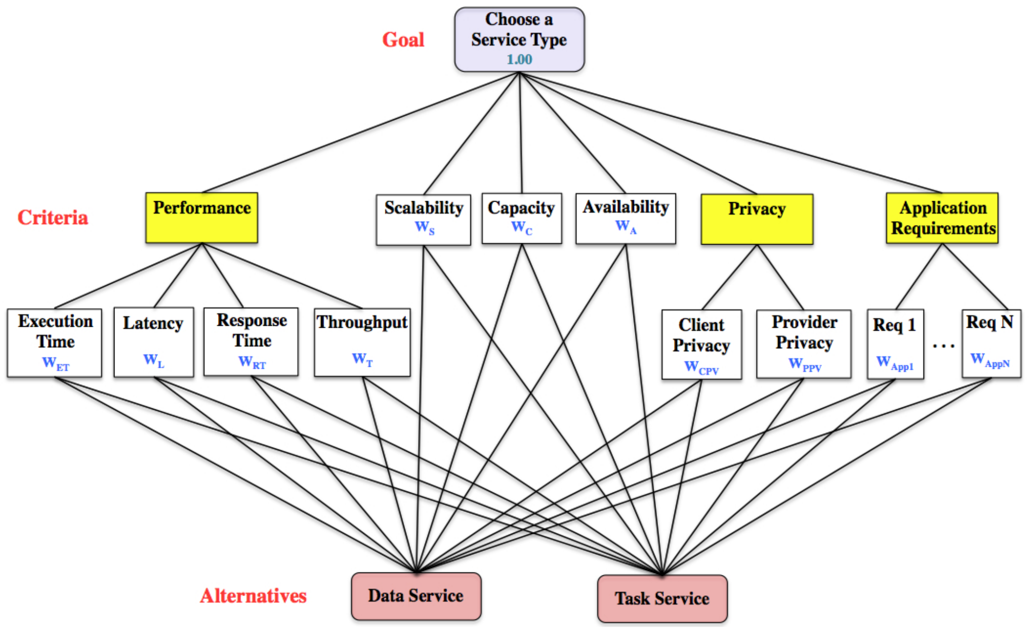


Figure 4: Proposed decision model to help service developers to determine the proper service type (data or task service) for a web service.

in Figure 3. A general stock buy (or sell) advice recommends that the client has a minimum (or maximum) percentage of specific shares in his/her portfolio. According to the task model, for general buy advice which targets stock "b", the service representative first checks to determine whether the client has enough shares of "b" ($Valid_{buy}?$). If this advice is valid, the service representative calculates the number of shares of "b" which the client should purchase and prints the resulting customized buy advice ($Print\ Buy\ Advice$). Similarly, the service representative generates customized sell advice.

3 Decision Model

The Analytic Hierarchy Process (AHP) [21] is a structured technique for dealing with complex decisions. This technique helps decision makers find a decision that best suits their goal and their understanding of the problem. Users of the AHP first decompose their decision problem into a hierarchy of more easily comprehended sub-problems, each of which can be analyzed independently.

Figure 4 represents the proposed decision model (AHP) containing the decision goal (service type), the alternatives for reaching it (data or task service), and the criteria (QoS parameters) for evaluating the alternatives. Prior to applying the decision model, the service developer needs to obtain information about the service clients such as client computing capabilities (e.g., CPU speed) and the client's network bandwidth BW (Figure 5). Since the developed web service will be used by different service clients, the service developer must decide about the characteristics of an average service client.

3.1 Alternatives Services

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

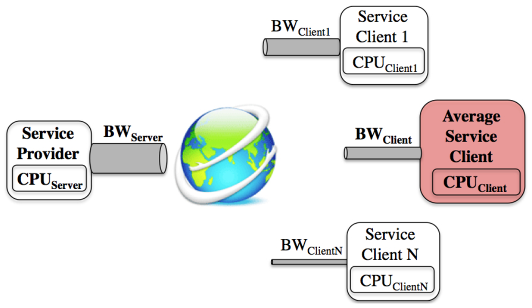


Figure 5: Required network characteristics in the decision making process.

service or a task service. While a data service applies the business components at the server-side, the task service sends the business components to the client-side to be applied by the service representative.

To apply the decision model, the service developer first needs to design and develop equivalent versions of the web service in the form of data services and task services. A data service includes an integrated server-side business process (called a *service* process) and a task service includes a server-side process (called a *task builder* process) and a client-side process (called a *task* process). Figure 6 represents the business processes which are required in the decision making process.

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

$$DWS : X^r_{DWS} \longrightarrow Y^r_{DWS}$$ [1]

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

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

[1]In defining a function (e.g., DWS), we use the sets of input "parameters" (e.g., X_{DWS}) and output "responses" (e.g., Y_{DWS}) to represent the types of the function's input and output.

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

3.2 QoS Parameters

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

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

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

3.3 Decision Criteria

For the proposed decision model, we use *platform-dependent QoS* parameters as decision criteria, which are evaluated differently for data and task services. In this section, we discuss how each decision criteria can be evaluated.

3.3.1 Execution Time

Service Execution Time (ET) is the time spent by a system executing the processes of a web service. The execution time depends on four factors: CPU speed, memory size and access time, and process complexity.

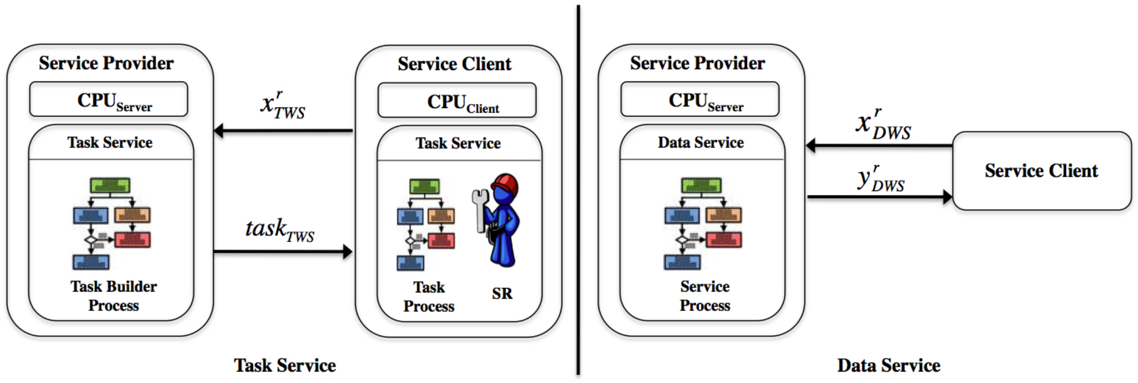


Figure 6: Data and task service alternatives for a web service.

We define $ET^{process}_{platform}(x)$ as the time spent by the *platform* to execute the *process* which takes input parameters x. The service developer can obtain execution time values using software profiling techniques available as Application Programming Interfaces (APIs).

The following equation represents the execution time evaluation function. While the data service process is performed at the server-side, the task service process is divided into two processes: *task builder* and *task* which are performed at the server-side and client-side, respectively.

$$C_{ET} = \begin{cases} \textbf{Data Service}: \\ ET^{service}_{server}(x^r_{DWS}) \\ \\ \textbf{Task Service}: \\ ET^{task\ builder}_{server}(x^r_{TWS}) + ET^{task}_{client}(x^l_{TWS}) \end{cases}$$

3.3.2 Latency

Service Latency (L) is defined as the time span from the time a service client issues a request for service to the time it receives a response message. Both data and task services receive remote service parameters x^r within the request messages. However, the resulting response messages include remote service responses y^r and task messages for data and task services, respectively.

Network tools such as *ping* tests can be used to measure latency. Moreover, the service latency can be estimated using network characteristics. The following evaluation function, which is a simplified version of the equation given by B.Liu [2], provides an estimate of the service latency. $Size(x)$ returns the size of data x and BW_{Client} represents the network bandwidth of the service client expressed in kilobytes/second.

$$C_L = \begin{cases} \textbf{Data Service}: \\ \frac{Size(x^r_{DWS})+Size(y^r_{DWS})}{BW_{Client}} \\ \\ \textbf{Task Service}: \\ \frac{Size(x^r_{TWS})+Size(task_{TWS})}{BW_{Client}} \end{cases}$$

Data services require transferring all client data volumes from the client to the server. On the other hand, task services process client data locally, implying that the service message size is independent of the volume of client data. Therefore, the task request message is short while the response message containing the task definition could be large.

3.3.3 Response Time

Service Response Time (RT) is the time between a service request sent and the corresponding service response received, which is defined as follows.

$$C_{RT} = C_{ET} + C_L$$

3.3.4 Throughput

Throughput is the number of web service requests served in a given time interval (e.g., 1 sec). Throughput depends on the service execution time at the provider site [4]. In the case of task services, service clients have their own service representatives, which can process client data in parallel, and therefore the throughput is improved significantly. The following equation, which was inspired by the model given by Ofuji [24], estimates the service throughput for data and task services.

A data service has one second to complete service requests. However, service representatives asks for ET^{task}_{Client} to perform client-side process of a task service. During this time, the server is free to serve other services (i.e., the server and service representatives are working in parallel). Consequently, all service requests processed in $[0, 1\ sec - ET^{task}_{Client}]$ by the server, will be processed by the service representatives before 1 second time period is expired.

$$C_T = \begin{cases} \textbf{Data Service}: \\ \frac{1(sec)}{ET^{service}_{server}(x^r_{DWS})} \\ \\ \textbf{Task Service}: \\ \frac{1(sec) - ET^{task}_{client}(x^l_{TWS})}{ET^{task\ builder}_{server}(x^r_{TWS})} \end{cases}$$

3.3.5 Scalability

Scalability represents the capability of increasing the computing capacity of the server to process more client requests per second. The computing capacity can be increased by adding more computing units and/or assigning more CPU shares to the web service processes. In this case, C_L represents the latency associated with each individual service client, whereas the computation power of the server is shared among different service clients. The following equation represents the gain in response time comes from increasing the server computing capacity by α_S factor ($\alpha_S \geq 1$).

$$C_S = \begin{cases} \textbf{Data Service}: \\ \frac{C_{RT}}{\frac{ET^{service}_{server}(x^r_{DWS})}{\alpha_S} + C_L} \\ \\ \textbf{Task Service}: \\ \frac{C_{RT}}{\frac{ET^{task\ builder}_{server}(x^r_{TWS})}{\alpha_S} + ET^{task}_{client}(x^l_{TWS}) + C_L} \end{cases}$$

3.3.6 Capacity

Capacity is the limiting number of simultaneous service requests which can be served by the provider with guaranteed performance (e.g., service response time). The capacity for the service response time depends on two factors: the server's network bandwidth, and the server-side execution time of the web service [23]. C_{RT} represents the service response time where there is only one service request. In case of multiple service requests, the acceptable service response time is defined by α_{RT}. Servers can simultaneously transfer messages and process service requests. Therefore, $\frac{\alpha_{RT}}{C_{RT}}$ represents a lower limit for the service capacity (in the case of a data service). The maximum limit is the number of service messages which can be transferred through the server channel in α_{RT}. Therefore, the service capacity is estimated by the average of the minimum and maximum values. For a task service, the service capacity can be estimated similarly by considering the service process distribution between server and client platforms.

$$C_C = \begin{cases} \textbf{Data Service}: \\ Avg\left(\frac{\alpha_{RT}}{C_{RT}}\ ,\ \frac{\alpha_{RT} \times BW_{server}}{Size(x^r_{DWS}) + Size(y^r_{DWS})}\right) \\ \\ \textbf{Task Service}: \\ Avg\left(\frac{\alpha_{RT} - ET^{task}_{client}(x^l_{TWS})}{C_L + ET^{task\ builder}_{server}(x^r_{TWS})}\ ,\right. \\ \left. \qquad \frac{\alpha_{RT} \times BW_{server}}{Size(x^r_{TWS}) + Size(task_{TWS})}\right) \end{cases}$$

3.3.7 Availability

The service availability is the probability that the service is up and operating. A web service can be temporarily shut down for several reasons such as service maintenance. However, a task service can take advantage of its client-side service representative during the service shut down period. For this purpose, a task definition needs to include an activation time to start operating when the server is not available. The service availability for a data service is simply defined by the following equation [20].

$$C_A = \frac{totaltime - idletime}{totaltime}$$

On the other hand, the service availability for a task service is not straightforward. During an idle time, the task service can not be invoked by service clients. But service clients who have invoked the service during the service activation period have access to the service through the service representative. However, this service availability is limited to task services where the service representative does not require any task updates in order to provide proper service responses. Moreover, in this case, an estimation of service client distribution over the activation and idle time is required.

The following equation represents the task service availability under different conditions, where $C\#_T$ represents the number of service clients during the evaluation period (e.g., a day). Similarly, $C\#_I$ represents the number of service clients during the service idle time.

$$C_A = \begin{cases} \textit{If task updates required} \\ \frac{total\ time - idle\ time}{total\ time} \\ \textit{Else} \\ \frac{total\ time - idle\ time}{total\ time} + \frac{idle\ time}{total\ time} \times \frac{C\#_T - C\#_I}{C\#_T} \end{cases}$$

3.3.8 Client Privacy

We define client privacy as the ability to keep the client's personal and sensitive information private and local. The service client may need to send its personal information to the service provider as service parameters. However, task services divide the service parameters into two categories: local and remote, where local service parameters which include client sensitive information are processed locally by the service representative.

To compare client privacy provided by data and task services, the service developer first needs to identify the client sensitive information among the local and remote service parameters, represented by the set S_{client}. A degree of sensitivity is assigned to each member of this set, represented by a positive number $Sensitivity_x$ for each $x \in S_{client}$. The following function gives a measure of how client privacy is violated by different type of services, where an increase in value represents more client privacy violations.

$$S^{WS}_{client} = \{x \mid x \in X^r_{WS} \ \wedge \ x \ is \ sensitive\}$$
$$C_{CPV} = \sum_{x \in S^{WS}_{client}} Sensitivity_x$$

3.3.9 Provider Privacy

Similarly to client privacy, data and task services represent different behaviours regarding service provider privacy. A data service performs all the processes at the server side. However, required task knowledge and task data for the service representative can include enterprise assets and resources. Revealing them to the client-side may violate enterprise privacy. To prevent this privacy vulnerability, a service provider can use one of the following techniques.

- Enterprise knowledge can be divided into locally applied (at the provider side) by the service or externally applied (at the client side) by service representatives. Therefore, the critical knowledge (e.g., market analysis) remains at the service provider, while non-critical knowledge (e.g., advice customization guidelines) can be sent to the service representative.
- The service client receives service responses from the service representative. The client does not have access to the knowledge transferred between the service provider and its representative. Encryption techniques can be used for data transmissions between a service provider and representative to improve enterprise privacy.

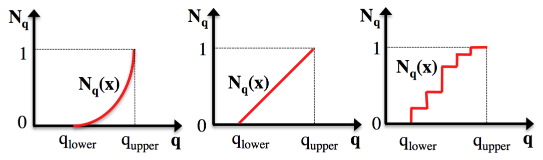


Figure 7: Exponential (left), linear (middle), and step (right) normalization functions for QoS parameters.

The following function C_{PPV} evaluates the provider's privacy violation where $S_{provider}$ includes sensitive task knowledge and task data. The encryption factor α_{Enc} is in [0,1], representing the efficiency of the encryption technique used.

$$S^{TWS}_{provider} = \{x \mid x \in Task_{TWS} \ \wedge \ x \ is \ sensitive\}$$

$$C_{PPV} = \begin{cases} \textbf{Data Service} \\ 0 \\ \\ \textbf{Task Service} \\ \sum_{x \in S^{TWS}_{provider}} Sensitivity_x \times (1 - \alpha_{Enc}) \end{cases}$$

3.3.10 Application Requirements

A web service can be invoked by different client applications where each application has specific requirements. While previous QoS metrics describe general features of a web service, this metric is based on client-side service applications. Although it is not possible for the service developers to consider all different types of client applications that would invoke this service, the service developer can categorize main applications.

For example, in a dynamic environment, where client data are changing frequently, a web service must be invoked whenever the service parameters are changed. In this case, a task service is more desirable, because the service representative can take care of changes in the local service parameters. Consequently, task service invocations will be decreased dramatically, improving performance metrics. For this application, the corresponding comparison measure is C_{SC} which represents the number of service calls in a given time. The following equation is the evaluation function for this metric, where β^r_{WS} represents the frequency of updates in the values of X^r_{WS} in a given time (1 minute).

$$C_{SC} = \beta^r_{WS}$$

3.4 Objective Function

QoS parameter values are not comparable and they must be transformed, or normalized, before the comparison takes place. A normalization function is defined for each QoS parameter in order to map values from its domain in its co-domain [0,1], preserving the original input data distribution. Platform-dependent QoS parameters (decision factor) are categorized into two main groups: desirable factors (including $C_T, C_S, C_C,\ and\ C_A$) and undesirable factors (including $C_{ET}, C_L, C_{RT}, C_{CPV}$, and C_{PPV}). A desirable (undesirable) decision factor is maximized (minimized) by the objective function. Therefore, a normalization function for a desirable (undesirable) decision factor must assign greater numbers to greater (smaller) QoS values.

A service developer can define a variety of customized normalization functions. For example, given QoS values for a data service q_{dws} and a task service q_{tws}, a simple linear normalization function can be defined as follow.

$$N_q(x) = \begin{cases} q\ is\ desirable \\ \frac{x}{max(q_{dws}, q_{tws})} \\ \\ q\ is\ undesirable \\ 1 - \frac{x}{max(q_{dws}, q_{tws})} \end{cases}$$

Alternatively, normalization functions which utilize upper and lower bounds can be defined. Figure 7 represents three examples of such functions. In addition to normalization functions, the service developer needs to define a numerical weight w_q for each QoS parameter q in the decision model.

The following equation represents the objective function derived from the decision model, represented in Figure 4. This function is a weighted sum of the QoS parameters where μ is the set of decision factors, $\mu = \{ET, L, RT, T, S, C, A, CPV, PPV, App\}$, and $\sum_{x \epsilon \mu} w_x = 1$.

$$\mathbf{C_{total}} = \sum_{x \epsilon \mu} w_x \times N_x(C_x)$$

Using the objective function, the service developer obtains C_{total} for both data service and task service alternatives and chooses the one with the higher value as the better candidate for providing the web service.

Finally, the objective function is likely to generate different results for different service input parameters. Therefore, in order to make an accurate decision, the service developer needs to categorize the service inputs based on their size and complexity and then calculate the objective function for each category. The final decision should be made by considering different categories.

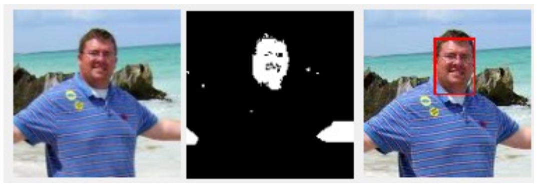


Figure 9: Face detection algorithm applied to a sample image. (Left) original image, (middle) skin-detected image, and (right) face-detected image.

4 Case Study

We use a case study of a face detection web service to demonstrate the different phases of the proposed decision making process.

4.1 Case Study Specification

A face detection service is a primary need in many fields, including face recognition, video surveillance, and human motion detection. A face detector takes an image and determines its regions that contain face(s). Several face detection approaches have been proposed in the literature, including approaches based on face spaces, neural networks, and template matching [17].

We adapted a two-step template matching algorithm for use in this case study. This algorithm includes two steps, where each step is developed by a process as follows.

1. *Skin Detector*: finds skin regions using explicit boundary rules on color values [14]. This step generates a binary image where all *skin* and *non-skin* pixels are assigned *Black* and *White*, respectively.
2. *Face Detector*: extracts face regions from skin regions using face templates, as follows: (i) enhances the binary image to eliminate noise; (ii) segments the enhanced image into connected regions; (iii) selects the potential face regions based on their size and width-to-height ratio; (iv) creates a feature vector for each selected region by detecting its edges; (v) compares feature vectors with the given face templates to find face regions; and (vi) shows face regions in rectangles.

Figure 9 represents an example that applies these two steps to a sample image.

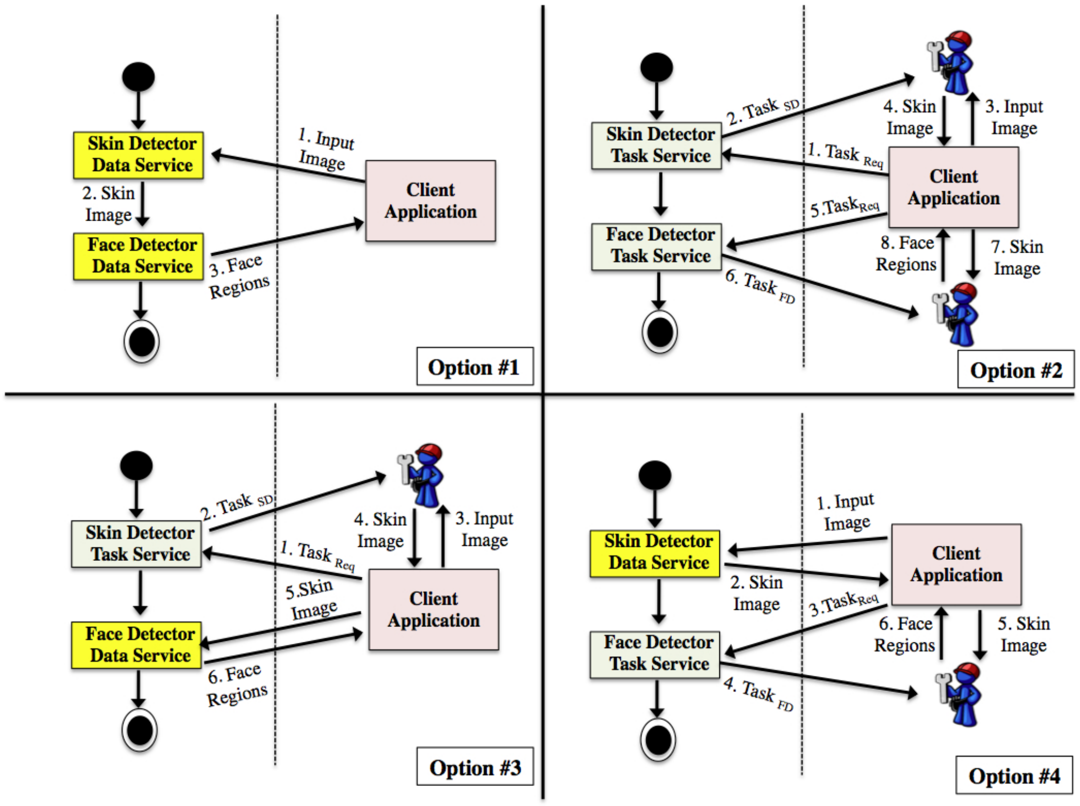


Figure 8: Alternative services include a composite data service (*option 1*), a composite task service (*option 2*), and two hybrid services (*options 3* and *4*).

4.2 Alternative Services

As discussed in Section 2, each business process can be provided as either a data service or a task service. Given two business processes, the alternatives include four services, represented in Figure 8. The alternative services include a composite data service (*option 1*), a composite task service (*option 2*), and two hybrid services (*options 3* and *4*).

While the composite data service was implemented by AJAX-RPC APIs, we implemented the composite task service and hybrid services by a developed prototype named *Extended SOA Version 1.1* (ESOA v1.1) [13]. This prototype is based on J2EE 1.5 technologies and an Apache Tomcat 6.0 application server. Moreover, it uses *Drools* [15] rule flow engine as the service representative process engine. Finally, the *ESOA v1.1* is divided into two Java packages: *TaskService* package used by service providers to develop task services, and *ServiceRepresentative* package used by service clients to install the generic service representative and invoke different task services.

4.3 Decision Model

In this section, we obtain the value of QoS parameters for each alternative service and then we apply the objective function to obtain the best service type. To perform the evaluation, we used a server platform with a (2.66 GHZ) 2 Core Intel CPU connected to the network via a 1 Mbyte/Sec link. On the other side, we consider an average service client with a (1.8 GHZ) single Core Intel CPU connected to the network via a 128 KByte/Sec link.

Our test suite consisted of 10 randomly selected images where all images were in color with various visual qualities, details and different sizes (80 Kbytes - 2 Mbytes). This test set includes images with and without facial shapes.

1. Execution Time: depends on the size and complexity of the input image as well as the processing speed of the computing unit. Figure 10 represents C_{ET} for our test samples. Based on the applied algorithm, the skin detector execution time depends on the size of the input image. However, the face detector execution time depends on the number of detected skin regions. Consequently, the execution time is not always incremental. The composite data service (option 1) performs all the processes at the server-side and provides the fastest results. For example, given a 360 Kbyte input image, the execution time for the skin detector process is 33 milliseconds at the server-side and 49 milliseconds at the client-side. Similarly, the execution time for the face detector process which takes the same image is 84 and 164 milliseconds at the server-side and client-side, respectively. Finally, we calculated the average execution times to be used in the other QoS evaluations, represented by $\overline{C_{ET}}$.

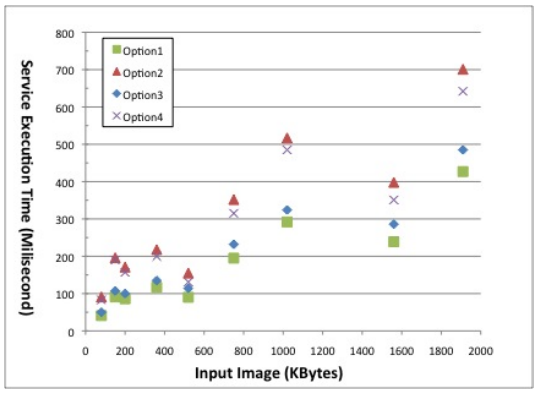


Figure 10: Service Execution Time for the candidate services. $\overline{C_{ET}^{\#1}} = 202$, $\overline{C_{ET}^{\#2}} = 357$, $\overline{C_{ET}^{\#3}} = 234$, and $\overline{C_{ET}^{\#4}} = 326$.

2.Latency: depends on both the size of transferred messages and the client network bandwidth. Figure 11 represents C_L for the test set where the composite task service (option 2) outperforms other options since the service messages only include the task definitions, not original or modified images. The third option, which requires transferring the (black-white) skin image and the resulting face regions, shows an acceptable latency. For example, given the 360 Kbyte input image, the skin image and face regions are 23 Kbytes, and 3 Kbytes, respectively. Finally, the size of task messages is independent of the size of input images. $Task_{Skin\ Detector}$ is 2Kbytes and $Task_{Face\ Detector}$ is 52 Kbytes, including face patterns.

3.Response Time: depends on both the execution time and latency. However, it is needed as it is usually one of the key QoS metrics asked by service clients. Moreover, based on Figures 10 and 11, it turns out that good execution time performance

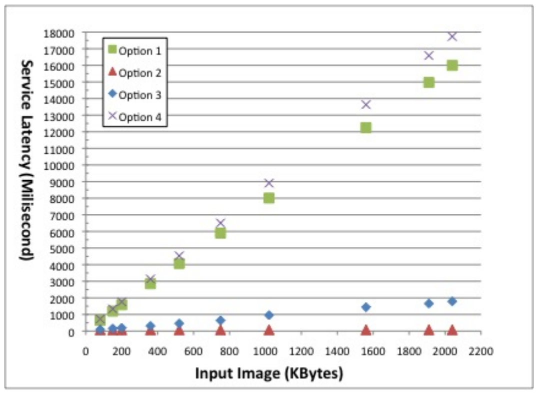


Figure 11: Service Latency for the candidate services. $\overline{C_{L}^{\#1}} = 6747$, $\overline{C_{L}^{\#2}} = 78$, $\overline{C_{L}^{\#3}} = 769$, and $\overline{C_{L}^{\#4}} = 7485$.

does not necessarily mean good performance in response time, and vice versa. Figure 12 compares C_{RT} among candidate services using the test set. Based on these results, the second and third services perform better than the first and fourth services since large client data volume affects service latency more than execution time.

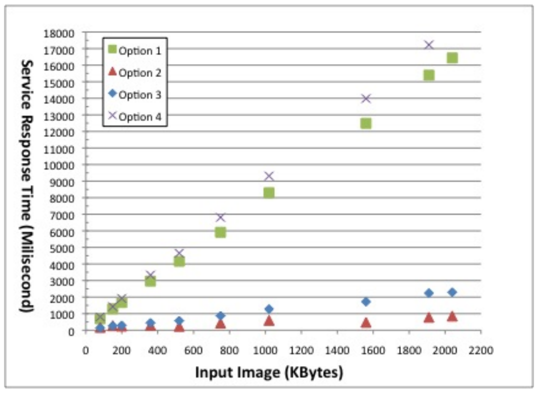


Figure 12: Service Response Time for the candidate services. $\overline{C_{RT}^{\#1}} = 6931$, $\overline{C_{RT}^{\#2}} = 435$, $\overline{C_{RT}^{\#3}} = 1013$, and $\overline{C_{RT}^{\#4}} = 7791$.

4.Throughput: is improved by performing processes in parallel and at the client-side. The composite task service (option 2) performs all the processes at the client-side and therefore it provides the best throughput. Using the average service execution time, the throughput for the candidate services are as follows: $C_{RT}^{\#1} = 5$, $C_{RT}^{\#2} = 36$, $C_{RT}^{\#3} = 7$, and $C_{RT}^{\#4} = 12$.

5.Scalability: is improved where the server performs some parts of the service processing and the execution time is comparable with the latency. In this case, we evaluated the gain obtained by increasing the server processing capability to process the client's requests twice as fast ($\alpha_S = 2$). The results are as follows: $C_S^{\#1} = 1.5\%$, $C_S^{\#2} = 2\%$, $C_S^{\#3} = 9\%$, and $C_S^{\#4} = 0.1\%$.

6.Capacity: is increased by task services as they require less network bandwidth and server computing share. Assuming that the minimum acceptable response time for this service (α_{RT}) is 8 seconds. The resulting capacities are as follows: $C_C^{\#1} = 1.15$, $C_C^{\#2} = 82$, $C_C^{\#3} = 8.6$, and $C_C^{\#4} = 1.1$.

7.Availability: is similar for all the services as we assume that the service does not need maintenance and is functioning 24/7. Therefore, $C_A^{\#1} = 1$, $C_A^{\#2} = 1$, $C_A^{\#3} = 1$, and $C_A^{\#4} = 1$.

8. Client Privacy: is violated by data services as the service client needs to transfer sensitive information (input images) as service parameters.

$$S_{client} = \{input\ image\}$$

We apply a sensitivity measure which assigns a number in [0,1] to sensitive data where higher value represents more sensitivity, $Sensitivity_{input\ image} = 0.5$. Then, the corresponding privacy violation costs for the candidate services are as follows: $C_{CPV}^{\#1} = 0.5$, $C_{CPV}^{\#2} = 0$, $C_{CPV}^{\#3} = 0$, and $C_{CPV}^{\#4} = 0.5$.

9. Provider Privacy: is violated by task services since the service provider needs to transfer sensitive information, which includes facial patterns as well as skin detection (SD) and face detection (FD) algorithms, required by the service representative to perform the tasks.

$$S_{provider} = \{patterns, SD, FD\}$$

Using the same sensitivity scale as client privacy, we assigned $Sensitivity_{pattern} = 0.8$, $Sensitivity_{SD} = 0.3$, and $Sensitivity_{FD} = 0.5$. Moreover, we assume $\alpha_{Encryption} = 0.6$. Then, the corresponding privacy violation costs for the candidate services are as follows: $C_{PPV}^{\#1} = 0$, $C_{PPV}^{\#2} = 0.64$, $C_{PPV}^{\#3} = 0.12$, and $C_{PPV}^{\#4} = 0.52$.

10. Application Requirements: are evaluated for three different applications as follows.

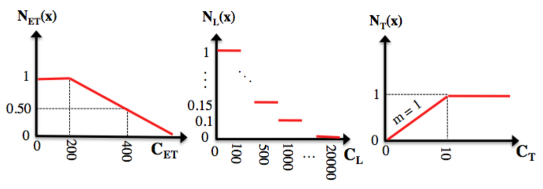
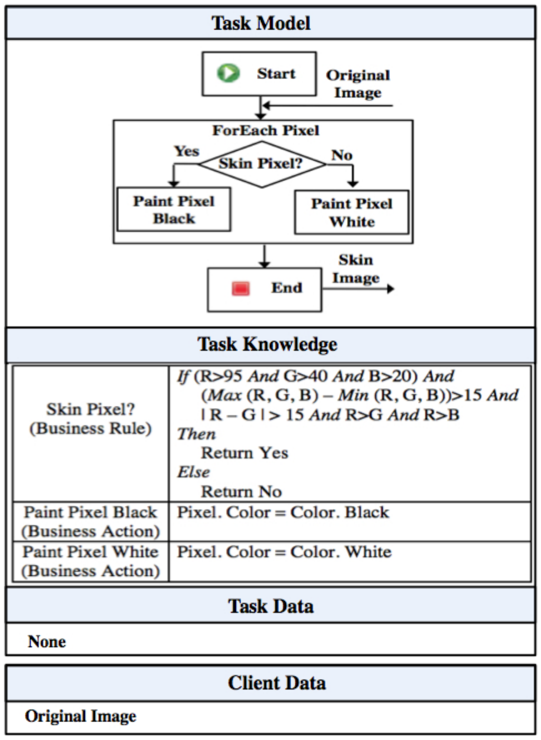


Figure 13: Three normalization functions used in the face detection case study.

1. *Still-Image Face Recognition.* This application requires only one web service invocation. $C_{SC1}^{\#1} = 1$, $C_{SC1}^{\#2} = 1$, $C_{SC1}^{\#3} = 1$, and $C_{SC1}^{\#4} = 1$.
2. *Video Surveillance.* This application requires invoking the web service every five seconds. The composite task service (option 2) processes all the images at the client-side. Consequently, it requires only one web service invocation. $C_{SC2}^{\#1} = 12$, $C_{SC2}^{\#2} = 1$, $C_{SC2}^{\#3} = 12$, and $C_{SC2}^{\#4} = 12$.
3. *Human Motion Detection*: This application requires invoking the web service every second. Similarly to the video surveillance application, the composite task service is the only option which does not require any updates. $C_{SC3}^{\#1} = 60$, $C_{SC3}^{\#2} = 1$, $C_{SC3}^{\#3} = 60$, and $C_{SC3}^{\#4} = 60$.

The obtained QoS parameters became comparable after normalization. Figure 13 shows three instances of the defined normalization functions. Other normalization functions were defined similarly. Table 1 represents the normalized QoS values and the assigned weight to each of them in the process of decision making. The weights should be assigned based on the provider's policies as well as potential clients' expectations. This can be done through an analysis of the business domain which is out of scope of this paper. In this case, we consider equal weights for privacy and performance metrics (each one receives 40%). Application specific QoS requirements are assigned 10%, where they are also evaluated by the performance metrics. The remaining 10% is assigned to the other criteria.

Using the Analytic Hierarchy Process (AHP), we obtained the total score for each alternative service by applying the objective function defined in Section 3. Based on the total scores, the preferred alternative is the third option. This composite service first uses a task service (*skin detector*) to employ the generic service representative to initially process the client's data at the client side. Figure 14 shows the task message generated by this



Task Knowledge

Skin Pixel? (Business Rule)	*If* (R>95 *And* G>40 *And* B>20) *And* (*Max* (R, G, B) − *Min* (R, G, B))>15 *And* | R − G | > 15 *And* R>G *And* R>B *Then* Return Yes *Else* Return No
Paint Pixel Black (Business Action)	Pixel. Color = Color. Black
Paint Pixel White (Business Action)	Pixel. Color = Color. White

Task Data

None

Client Data

Original Image

Figure 14: **SkinDetector$_{\mathbf{TWS}}$** task components and the corresponding client data.

service. In the next step, it uses a data service (*face detector*) to perform the more complicated and more sensitive face detection process at the server side. By analyzing the results of this experiments, we conclude that the selected composite service outperforms other services by reducing network traffic, reducing execution time, and maintaining client and provider privacy.

5 Related Work

In this section, we briefly present some research work from the related literature. In distributed computing, code on demand [16] (also called code mobility) refers to any technology that sends executable software programs from a server computer to a client system upon request from the client. Java applets and Microsoft Silverlight are two instances of code mobility. Task service is a client-side processing approach and also is an alternative to code mobility. Task services have advantages over code on demand approaches due to their composability and scalability. Moreover, task messages are not executable, which improves client security.

Code mobility has been compared with the traditional client/server approaches. For example, a performance analysis of client/server versus agent based communication was performed in [3] where the entire agent moves from device to device rather than having to spend processing time creating a static agent for each device. A hierarchical framework of benchmarks was proposed in [9] for performance analysis of mobile agent systems. In [18], Java Remote Invocation (RMI) and .Net Remoting Architecture were compared based on an experimental performance analysis. Similarly, [19] investigated three Java-based approaches to distributed computing (Java RMI, Java applet-servlet communication and Mobile Agents) using performance parameters such as code size, latency, response time, partial failure, concurrency, and ease of development. To the best of our knowledge, this paper is one of a few attempts to compare traditional (server-side) web services with client-side processing approaches (task services).

Web service performance testing is an emerging field of software engineering which must be carried out at both the server-side and the client side. However, choosing the relevant performance criteria is a critical task. At the client-side, web service performance depends on the amount of data transmitted over the network. At the server-side, selection of programming language and platform, and implementation complexity are the primary contributors to web service performance [6]. Data and task services provide the same business logic and are implemented using identical technologies. Consequently, we need to compare them, based on their QoS parameters which are relatively easy to obtain. Open-source performance analysis tools such as Apache JMeter, Firebug and YSlow can be used to obtain QoS parameter values efficiently. Moreover, QoS monitoring approaches (e.g., [8]) can be used.

QoS parameters have been proposed widely in service selection to compare alternative services. For example, an interactive web service choice-making process is proposed in [12], which takes QoS as a key factor when choosing from functionally equivalent services. Similarly, [5] introduces the web service relevancy function used for measuring the relevancy ranking of a particular Web service based on client's preferences, and QoS metrics. By considering the service selection problem as an optimization problem, [22] maximizes an application-specific utility function under end-to-end QoS constraints.

Table 1: Normalized QoS parameters values for candidate face detection web services. The third option receives the highest score and is chosen as the best candidate to model the web service.

QoS metric (weight)	C_{ET} (0.1)	C_L (0.1)	C_{RT} (0.1)	C_T (0.1)	C_S (0.03)	C_C (0.05)	C_A (0.02)	C_{CPV} (0.20)	C_{PPV} (0.20)	C_{SC1} (0.03)	C_{SC2} (0.05)	C_{SC3} (0.02)	**Score** (1.00)
Option 1	1	0.3	0.12	0.5	0.13	0.12	1	0.5	1	1	0.4	0	0.57
Option 2	0.48	1	1	1	0.3	1	1	1	0.5	1	1	1	0.82
Option 3	0.93	0.9	0.83	0.7	1	0.9	1	1	0.8	1	0.4	0	0.85
Option 4	0.65	0.2	0.08	1	0.03	0.1	1	0.5	0.6	1	0.4	0	0.48

Finally, the generic service representative and task services were proposed in [10] and [13]. The former introduced a generic software agent (a simplified version of service representative) that resides at the client side and customizes web service responses based on the client context. The latter proposed an architecture for the extended SOA model including the generic service representatives and task service providers which supports the novel concept of client-side service composition. A real-world application of task services in the healthcare domain was proposed in [11], where the service representative plays the role of a virtual nurse performing assigned tasks by different physicians.

6 Conclusions and Future Work

In this paper, we proposed a decision model to help service developers make decisions about the proper type (server-side data service or client-side task service) to adopt for web services. The decision criteria are platform dependent QoS which are evaluated differently for data and task services. Moreover, based on the case study provided, a composition of data and task services can exhibit good QoS scores if the task service initially processes the client's data and the pre-processed data are transferred to the server for more complex processing.

The proposed model evaluates each business process individually. As future work, we plan to solve an optimization problem to find the best combination of data and task services representing a composite business process. Finally, we plan to involve service clients in the decision making process as follows. The list of decision criteria (QoS parameters) will be presented to the service clients for the weight adjustments. Then the system will recalculate the decision model and produces a ranked list of different service options. Therefore, this user will have more control on the results based on different priorities.

References

[1] B.Benatallah and H.Motahari Nezhad. Service oriented architecture: Overview and directions. In *Advances in Software Engineering*, volume 5316 of *Lecture Notes in Computer Science*, pages 116–130. Springer Berlin / Heidelberg, 2008.

[2] B.Liu, P.Ray, and S.Jha. Mapping distributed application sla to network qos parameters. In *Proc. of 10th International Conference on Telecommunications, IEEE ICT 2003*, volume 8, pages 1230 – 1235, 2003.

[3] D.Barnes, S.Sankaranarayanan, and S.Ganesan. Performance analysis of client/server versus agent based communication in wireless sensor networks for health applications. In *Electro/Information Technology, 2009. eit '09. IEEE International Conference on*, pages 271 –276, june 2009.

[4] D.Menasce. Qos issues in web services. *Internet Computing, IEEE*, 6(6):72 – 75, 2002.

[5] E.Al-Masri and Q.Mahmoud. Qos-based discovery and ranking of web services. In *Computer Communications and Networks, 2007. ICCCN 2007. Proceedings of 16th International Conference on*, pages 529 –534, aug. 2007.

[6] J.Cane. Performance measurements of web applications. In *SoutheastCon, 2003. Proceedings. IEEE*, pages 87 – 93, april 2003.

[7] J.Ng, M.Chignell, J.Cordy, and Y.Yesha. Overview of the smart internet. In *The Smart Internet*, volume 6400 of *Lecture Notes in Computer Science*, pages 49–56. Springer Berlin / Heidelberg, 2010.

[8] L.Zeng, H.Lei, and H.Chang. Monitoring the qos for web services. In Bernd Krmer, Kwei-Jay Lin, and Priya Narasimhan, editors, *Service-Oriented Computing*, volume 4749 of *Lecture Notes in Computer Science*, pages 132–144. Springer Berlin / Heidelberg, 2007.

[9] M.Dikaiakos and G.Samaras. A performance analysis framework for mobile agent systems. In Tom Wagner and Omer Rana, editors, *Infrastructure for Agents, Multi-Agent Systems, and Scalable Multi-Agent Systems*, volume 1887 of *Lecture Notes in Computer Science*, pages 180–187. Springer Berlin / Heidelberg, 2001.

[10] 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.

[11] M.Najafi, S.Aghtar, K.Sartipi, and N.Archer. Virtual Remote Nursing System. In *The 1st IEEE Conference on Consumer eHealth Platforms*, pages 101–105, USA, 2011.

[12] M.Yu-jie, C.Jian, and Zhang Z.Shen-sheng, Zand Jian-hong. Interactive web service choice-making based on extended qos model. In *Proceedings of the The Fifth International Conference on Computer and Information Technology*, CIT '05, pages 1130–1134, Washington, DC, USA, 2005. IEEE Computer Society.

[13] M. Najafi and K.Sartipi. Client-side Service Composition Using Generic Service Representatives. In *CASCON 2010: Proceedings of the 2010 conference of the Center for Advanced Studies on Collaborative research*, pages 238–252, Toronto, Canada, 2010.

[14] P.Kakumanu, S.Makrogiannis, and N.Bourbakis. A survey of skin-color modeling and detection methods. *Pattern Recogn.*, 40(3):1106–1122, 2007.

[15] M. Proctor, M. Neale, P.Lin, and M.Frandsen. Drools documentation. Technical report, JBoss.org, 2008.

[16] R.Brooks. Mobile code paradigms and security issues. *Internet Computing, IEEE*, 8(3):54 – 59, 2004.

[17] R.Jafri and H.Arabnia. A survey of face recognition techniques. *Journal of Information Processing Systems*, 5(2):41–68, 2009.

[18] R.Schwarzkopf, M.Mathes, S.Heinzl, B.Freisleben, and H.Dohmann. Java rmi versus .net remoting architectural comparison and performance evaluation. *International Conference on Networking*, 0:398–407, 2008.

[19] S.Mangalwede and D.Rao. Performance analysis of java-based approaches to distributed computing. *International Journal of Recent Trends in Engineering*, 1:556–559, 2009.

[20] S.Ran. A model for web services discovery with qos. *SIGecom Exch.*, 4:1–10, 2003.

[21] T.Saaty. Decision making with the analytic hierarchy process. *International Journal of Services Sciences*, 1:83–98, 2008.

[22] T.Yu, Y.Zhang, and K.Lin. Efficient algorithms for web services selection with end-to-end qos constraints. *ACM Trans. Web*, 1, May 2007.

[23] X.Chen and P.Mohapatra. Performance evaluation of service differentiating internet servers. *IEEE Transactions on Computers*, 51:1368–1375, 2002.

[24] Y.Ofuji, A.Morimoto, S.Abeta, and M.Sawahashi. Comparison of packet scheduling algorithms focusing on user throughput in high speed downlink packet access. In *Personal, Indoor and Mobile Radio Communications, 2002. The 13th IEEE International Symposium on*, volume 3, pages 1462 – 1466, 2002.