**RJ10344 (A0504-002) April 1, 2005**
**Computer Science**

# IBM Research Report

## Multiagent System for Dynamic Web Services Selection

**E. Michael Maximilien**
IBM Research Division
Almaden Research Center
650 Harry Road
San Jose, CA 95120-6099

**Munindar P. Singh**
North Carolina State University
Department of Computer Science
Raleigh, NC 27695

**Research Division**
**Almaden - Austin - Beijing - Haifa - India - T. J. Watson - Tokyo - Zurich**

# Multiagent System for Dynamic Web Services Selection

E. Michael Maximilien
IBM Almaden Research Center
650 Harry Road
San Jose, CA 95120
maxim@us.ibm.com

Munindar P. Singh
North Carolina State University
Department of Computer Science
Raleigh, NC 27695
singh@ncsu.edu

## Abstract

*Service Oriented Architectures (SOAs) promise to enable the creation of business applications from independently developed and deployed* services*—roughly, software components that encapsulate and provide business functionality through standardized interfaces. A key advantage of SOAs is that they enable services to be dynamically selected and integrated at runtime, thus enabling system flexibility and adaptiveness—autonomic attributes that are key for modern business needs. However, current techniques provide no support for actually making rational selections, which are key to accomplishing autonomic behavior.*

*We develop a multiagent framework based on an ontology for QoS and a new model of trust. The ontology provides a basis for providers to advertise their offerings, for consumers to express their preferences, and for ratings of services to be gathered and shared. The ratings are essential, because they give an* empirical *basis for the selection of services. The ratings are quality-specific and are obtained via automatic monitoring or, if appropriate, user input.*

*The agents thus form an ecosystem in which they help each other. We show how this approach matches the well-known* self-* *traits of autonomic computing, and introduce the notion of* ***self-adjusting trust****. We empirically evaluate the resulting system via simulation. Our results show that the agents are able to dynamically adjust their trust assignments and thus continually select the best available services for their consumers' needs.*

## 1. Introduction

Service-oriented architectures (SOAs) offer an essential ingredient of autonomic computing (AC). This is especially so when AC is understood generally as producing continual automatic adjustment to all aspects of a system, including its software configuration. To our way of thinking, the power of AC lies not just in managing data centers but more broadly in computing, including for business services. SOAs enable the flexible binding of services—i.e., well-encapsulated modules of functionality—to construct applications and systems that best meet user needs.

The SOA vision is that, first, providers will offer several (potentially competing) services and, second, prospective users of services will dynamically choose the best offerings for their own purposes. For example, you might choose the best hotel booking service or the best bookseller, where you alone decide what is best for you. Likewise, you might choose the best component services (such as logging, backing up, and so on) to construct and deploy an application that meets your needs. Viewed over multiple episodes, such dynamic selection is a key prerequisite for autonomic computing (AC), because it would enable an application or system to continually evolve to maintain or improve its performance along the axes of interest.

However, current approaches only partially address the SOA vision. They enable services to be described and listed in public registries (analogous to telephone directories). But they provide no means of selecting among multiple services that appear to perform the same function. In other words, you are forced to make an ad hoc decision about which of the many hotel booking services or booksellers to use. Because tens of thousands of specialized and not widely known services are involved, a practical approach cannot merely pre-select a few famous companies such as Amazon.com, but must apply at a much larger scale. Further, when we consider not only business services but also component services, the choices can become quite subtle because the same provider may offer multiple alternatives.

The thesis of this paper is that service selection can be rationally carried out only on an *empirical* basis—that is, how a given service has behaved, not only how it was advertised. Given the large number of services, users must share information about their experiences—in effect, multiplying the benefit of their empirical evaluations by sharing them. Traditional, proprietary reputation systems (such as those maintained by eBay) and proprietary recommender

systems (such as those maintained by Amazon.com) are not suitable for services. In particular, they do not allow a customizable schema in terms of the qualities of interest to different users, interject themselves into each transaction, and own the data that is gathered.

What is needed is a means to allow service consumers to share quality opinions, which presuppose an agreed upon set of QoS definitions. Using these quality opinions, service consumers can derive the reputation of service implementations on these qualities. By knowing its quality needs for an application, a service consumer can derive a *trust* value for each available service implementation. Thus, selecting the best service implementation simply corresponds to selecting the most trusted implementation. We can automate the service selection task with software agents acting on behalf of service consumers.

### 1.1. Contribution

An important characteristic of automatic selection by using trust in open environments, such as the Web, is that trust should be *self-adjusting*. That is, service implementations that behave incorrectly should (in essence) be purged from the system by virtue of not being selected. Poor service implementations should accumulate a low reputation. Conversely, when a once awry service implementation starts to behave correctly, we would like the agents to increasingly consider it for selection. This dynamic and self-adjusting consideration of trust for selection matches the goals of autonomic computing [19].

**Self-adjusting trust.** The autonomic characteristic of a multiagent system whereby the levels of trust between the interacting parties are dynamically established and adjusted to reflect recent interactions.

We develop a multiagent framework that uses an ontology for QoS to support self-adjusting trust. The ontology provides a basis for providers to advertise their offerings, for consumers to express their preferences, and for ratings of services to be gathered and shared. The ratings yield an *empirical* basis for the trust placed in different implementations. Moreover, the agents thus form an ecosystem in which they help each other identify the best implementations. Poorly performing implementations can thus be avoided. The converse challenge is to introduce new services or revive services that behaved poorly but are now functioning well again. To this end, this paper introduces what we term *explorer agents*. The explorer agents provide a means to monitor different service implementations, especially those that are new or currently out of favor. Thus, they provide a basis for consumers to select implementations that are predicted to perform well along the qualities of interest to them, even if there is inadequate positive experience with such implementations.

Our evaluation shows that the agents are able to dynamically adjust their trust assignments and thus continually select the best available services for their consumers' needs.

### 1.2. Organization

The remaining of the paper is as follows. Section 2 gives an overview of our trust model and how it is used to solve the service selection problem. Section 3 briefly discusses the technical framework including highlighting previous results. Section 4 gives a conceptual evaluation of our model and presents the framework in the realm of autonomic computing. Section 5 gives a detail empirical evaluation showing the emergence of self-adjusting trust. Section 6 highlights various related work in the field and Section 7 concludes and gives some directions for future work.

## 2. Background

We developed an agent-based approach for service selection that includes a flexible notion of trust based on reputation. The agents transparently attach to existing services and enable their dynamic selection. We introduce a comprehensive *ontology* (roughly, a taxonomy with some additional features) for qualities of services (QoS). This ontology includes the well-known computing qualities such as throughput and latency, but provides hooks to include any application-specific or even idiosyncratic qualities that users may need, such as shipping delay. Our agent-based framework enables users to share information about any of the qualities. We developed algorithms by which user preferences regarding which qualities they consider more or less important can be applied (using the reputation data) to help each user select services that best meet his needs. The framework also has the ability to continually monitor services so that services that begin to perform poorly (relative to a particular user's preferences) are de-selected and those that begin to perform well are re-selected.

We now give an overview of how we model the service selection problem, our QoS-based trust model, and show how our model can be used as a solution to the service selection problem.

### 2.1. Service Selection

We model each Web service $s = (\iota, i)$ as a pair, where $\iota \in \Upsilon$ is the interface and $i \in I_\iota$ is an implementation of the service. $\Upsilon$ represents the set of all URIs and $I_\iota \subseteq \Upsilon$ is the set of all service implementations of interface $\iota$. For each service $s$ we associate an application domain $d \in \Delta$ where $\Delta$ is the set of all application domains. An example of an application domain is *Math* representing Web services for

mathematical calculations; another domain is *Finance* with services such as loan and stock quote.

With each application domain $d$ we associate qualities $Q \in \Phi_d$ representing nonfunctional attributes common to the Web services in the application domain $d$. $\Phi$ is the set of all qualities. For each quality $Q$ and service $s$ we let $\widehat{Q_s} = \{q_1, \ldots, q_n\}$ be the set of collected opinions, on quality $Q$. These opinions corresponds to $n$ selection of service $s$.

We assume without loss of generality that for each selection of service $s$ we obtain quality opinions (from the selection agent) for each quality in the domain $d$ of $s$. We can now formulate the service selection problem.

**Definition 1 (Service Selection)** *Let $P$ be the set of all providers with implementations for interface $\iota$. Our problem is to select the service implementation $i \in I_\iota$ of service $s$ from all service providers $p \in P$ such that:*

$$i = \arg \max_{i \in I_\iota} \{trust(i, \Phi_d)\} \quad (1)$$

*Where $trust() : I_\iota \times \Phi \mapsto \mathcal{R}$ is a service trust function.*

## 2.2. QoS Preferences and Advertisements

Each service consumer $c \in C$, where $C$ is the set of all consumers, will have service quality needs that are specific to its application. For instance, the consumer of a *StockQuote* service used to give quick security quotes on financial Web sites has different quality needs than for a consumer of this same quote service but within a brokerage application used to buy and sell securities. The latter's need for fast response time, high availability, and accuracy are critical to the brokerage application's success.

In order to accurately select services for consumers we first need a means to represent the consumer's needs for each quality exposed by the service.

**Definition 2 (Consumer Quality Preferences)** *A consumer's preferences for a quality $Q$ are given by $\pi = (\pi_{min}, \pi_{pref}, \pi_{max})$, where $\pi_{min}$ is the minimum value acceptable for the quality, $\pi_{max}$ the maximum acceptable, and $\pi_{pref}$ is the preferred value for quality $Q$ all from the perspective of the consumer. We require $\pi_{min} \leq \pi_{pref} \leq \pi_{max}$.*

A consumer's preference for a service $s$ is a collection of quality preferences, one for each quality needed by the consumer.

Since service providers may offer different services with specific targeted consumers, we also need a means to represent the providers' advertisements for each quality in the set of qualities exposed by a service.

**Definition 3 (Provider Quality Advertisement)** *A quality advertisement $\alpha$ is a proclamation by a provider for a particular quality $Q$. More specifically, we denote $\alpha = (\alpha_{min}, \alpha_{typical}, \alpha_{max})$, where $\alpha_{min}$ is the minimum advertised value for the quality, $\alpha_{max}$ the maximum, and $\alpha_{typical}$ is the typical value promised for the quality $Q$ by the provider. We require $\alpha_{min} \leq \alpha_{typical} \leq \alpha_{max}$.*

A provider's service advertisement for a service implementation is a collection of quality advertisements, one for each quality applicable to the service.

## 2.3. Trust Model

To provide a solution to Equation 1 we need to provide a $trust()$ function that uses the collected quality values while taking into account the quality preferences of the consumer and the advertisements of the provider. The resulting trust value for a service implementation would enable us to rank different service implementations according to how well they meet the consumer's quality needs.

Let us assume that the collected quality opinions values are normally distributed with minimum, maximum, mean, and variance which are inferred from domain experts and attached to the QoS ontology. We can then assume that the collected values $\widehat{Q}$ for each quality $Q$ can be normalized as Z statistics values [35, p. 173]. This enables us to meaningfully compare qualities and aggregate them into a single value.

With this assumption, we start by defining an aggregation of the collected quality values for a quality $Q$ that represents the general opinions of all agents that have selected the given service implementation.

**Definition 4 (Service Quality Reputation)** *We denote the reputation $R_Q^{(i)}$ of a service implementation, with respect to quality $Q$, as the aggregation of the quality opinions (i.e., quality values) for the service implementation $i$ of service $s$ over some time interval. Specifically,*

$$R_Q^{(i)} = \frac{1}{n} \sum_{k=1}^{n} q_k \delta^{-t(q_k)} \quad (2)$$

*where $n$ is the number of collected quality values, $\widehat{Q}_i = \{q_k\}_{k=1}^n$ is the set of quality values collected from service agents as they selected the service implementation $i$, $\delta \in R$ is the quality $Q$'s dampening factor, and $t() : \Phi \mapsto \mathcal{Z}^+$ is the time for which the quality value $q$ was collected. $t(q) = 1$ for the most recent collected value and $t(q) > 1$ for all other values.*

With $R_Q^{(i)}$ we have a representation of the general opinion on how well service implementation $i$ performs for a quality $Q$. We now need a means to derive the trust value that a prospective consumer of service $s$ should assign to

each service implementation $i$ of interface $\iota$ using the quality reputations for implementation $i$, the consumer's quality preferences, and the quality advertisements of provider $p$ of implementation $i$.

Since the quality advertisements and preferences are defined as points on the quality line of $Q$, we can calculate the moment of these points with respect to the $\pi_{pref}$ of the consumer preferences. In essence, the closer the advertised values and reputation are to the preferred value, the greater the degree of match (and of the resulting trust). Generally, Equation 3 shows the second moment of a vector $\vec{x} = \langle x_1, x_2, \ldots, x_n \rangle$ about some point $a$.

$$moment(\vec{x}, a) = \frac{1}{n-1} \sum_{i=1}^{n} (a - x_i)^2 \qquad (3)$$

We formulate the consumer's trust assignment for an implementation using Equation 3. However, since we want to match service implementations whose advertisement match the need of a service consumer, we start by defining a matching operator between quality preferences and advertisements.

**Definition 5 (Preference Matching Operator $\triangleright$)** *For each $Q \in \Phi_d$ let $\alpha_Q = (\alpha_{min}, \alpha_{typical}, \alpha_{max})$ is the advertisement of provider $p$ of service implementation $i$ for quality $Q$ and $\pi_Q = (\pi_{min}, \pi_{pref}, \pi_{max})$ be the consumer's preferences for quality $Q$.*

*Let $Q_{min} = \min(\alpha_{min}, \pi_{min})$ and $Q_{max} = \max(\alpha_{max}, \pi_{max})$.*

*Let $\vec{Q_i} = \langle Q_{min}, \alpha_{typical}, \pi_{pref}, Q_{max}, R_Q^{(i)} \rangle$.*

*We define the preference matching operator $\triangleright$ for $Q$ as:*

$$\phi \triangleright \varphi \quad = \quad \begin{cases} (\pi_{max} \leq \alpha_{max}) \wedge (\pi_{pref} \geq \alpha_{min}) \wedge \\ (\pi_{pref} \leq \alpha_{max}) \ \textit{if } dir(Q) = \uparrow, \textit{ and} \\ \\ (\pi_{min} \leq \varphi_{min}) \wedge (\pi_{pref} \leq \alpha_{min}) \wedge \\ (\pi_{pref} \geq \alpha_{max}) \ \textit{when } dir(Q) = \downarrow \end{cases}$$

*Where $dir(Q) : \Phi \mapsto \{\uparrow, \downarrow\}$ is associated with each quality $Q$, such that $dir(Q) = \uparrow$ indicates that the quality $Q$ is directionally increasing which means that higher values for $Q$ are generally preferred by service consumers. And $dir(Q) = \downarrow$ indicates that lower values are generally preferred.*

Using the $\triangleright$ operator we can derive the $trust$ function of Definition 1 as follows.

**Definition 6 (Service Trust Function)**

$$\begin{aligned} qTrust(\vec{Q_i}, q_{pref}) &= moment(\vec{Q_i}, q_{pref})^{-\frac{1}{2}} \\ &\quad \textit{where } moment(\vec{Q_i}, q_{pref}) \neq 0 \\ serviceTrust(i) &= \sum_{\substack{Q \in \Phi_d, \\ \phi \triangleright_Q \varphi}} qTrust(Q_i, q_{pref}) \\ trust(i_p, c) &= serviceTrust(i_p) \end{aligned}$$

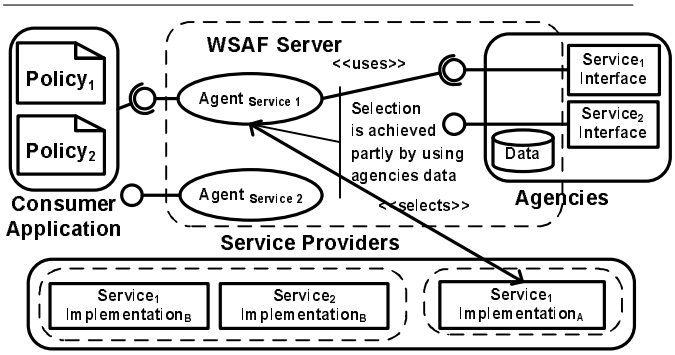


**Figure 1. Architecture overview.**

## 3. Framework

To evaluate our trust model and hypothesis of self-adjusting trust, we created a framework that augments a typical SOA with agents. The principal idea is to install software agents between service consumers and each service that they consume. These service agents expose the same interface as the service. However, they augment the service interface with agent-specific methods. An example of such a method is *setWsPolicy* which allows consumers to communicate their QoS preferences. By exposing the same interface as the service these agents are able to transparently and dynamically select the actual service implementation by considering the service consumer's quality needs. The consumer communicates its need via the augmented agent interface. Service method invocations are done via the service agent who in turn monitors and forwards all calls to the selected service.

Figure 1 shows a high-level view of the architecture; the details and runtime operation are described at length in [21]. Briefly, the consumer application makes use of *Service 1* which has three implementations (two by provider *B* and one by provider *A*). Instead of selecting the implementation directly, the application uses a service agent which expose the same interface as *Service 1* and selects, on the consumer's behalf, the implementation which best matches the consumer's policy. It is worthwhile to note that the service agents can be co-located with the service consumers or distributed to an agent server (application server) as shown in Figure 1. The advantage of decoupling the agents and the service consumers is to move the agents' processing from the consumer's applications and importantly to allow cross-platform consumer-to-agent interactions.

In addition, the service agents participate in common *agencies* where they share their quality opinions on the selected service implementations. An agency is simply a *rendezvous* node on the network where quality opinions are

shared and aggregated.

The agents share a conceptualization of quality in the form of an ontology. The ontology is partitioned into three parts. The *upper QoS ontology* contains basic definitions for all qualities, including modeling relationships between qualities. The *middle QoS ontology* extends the upper ontology and defines qualities that are applicable across different domains. *Lower QoS ontologies* are defined for specific domains by extending qualities in the middle ontology or creating new ones from the upper ontology. Service agent behaviors for quality monitoring can also be attached to the ontology and dynamically bootstrapped in the agents. Maximilien and Singh [21] give an overview of the upper and middle QoS ontology as well as discussing examples of lower ontology qualities and example usages of the framework.

We implemented this architecture in the Web Service Agent Framework (WSAF) and used simulation experiments on simple services as an initial evaluation [22]. The initial results showed that the service agents are able to accurately select service implementations according to the consumer's preferences and adjust the selection as service implementations' quality degrade. We further enhance this evaluation in Section 4 by conceptually showing how our notion of self-adjusting trust fits the challenges and goals of autonomic computing. In Section 5, we evaluate an approach to enable the service agents to adjust their service selection when a well-behaved service implementation starts degrading its exposed qualities and then again provides good qualities (or vice versa).

## 4. Conceptual Evaluation

A characteristic of our system as a whole is the emergence of a certain level of autonomy. This autonomy is the result of the shared knowledge of the consumers which allows them to make selections appropriate to their preferences and at the same time biasing their choices towards well-behaved services by taking into account the opinions of other consumers. Once a service starts behaving in a fashion that is not in accordance with its advertised QoS, it eventually collects low QoS values (low ratings) and therefore a low overall reputation. Unless that service starts behaving correctly, we expect consumer agents to be biased against selecting it. That is, our approach mimics social structures where experiences, knowledge sharing, and reputation are used to make decisions. The entire system regulates itself and therefore exhibits autonomy.

This emerging behavior of the system falls perfectly into the autonomic computing paradigm, as promoted by Horn [12]. According to Kephart and Chess [19] as well as Ganek and Corbi [9] an AC system manifests four main *self-** traits:

1. *Self-configuring*. The system dynamically configures and reconfigures itself as its environment changes.

2. *Self-optimizing*. The system tries to achieve optimal operational characteristics vis-à-vis the current conditions.

3. *Self-protecting*. The system protects itself from internal or external threats.

4. *Self-healing*. The system takes corrective actions to address, as well as prevent, failures during its operations.

A system that achieves some or all of the above traits is said to be autonomic and achieves some level of *self-management*. Such a system adjusts itself to be more resilient to its environment and to better support the goals and policies of its owners.

From the above descriptions we can say that our service agents exhibit dynamic configuration during runtime, in essence merging the traditional phases of configuration and execution. Using the preferences of its service consumer, a service agent is able to reselect and rebind to a new service provider and implementation, thereby reconfiguring the consumer's application. Since the agents participate in agencies that allow them to improve subsequent choices and collect overall historical perspectives on the various service providers and implementations, the system as a whole encourages good services while purging the ones that misbehave. This degree of self-regulation, self-optimization, and therefore self-management are key desired overall traits of AC systems. Our approach enables a new AC trait that we name *self-adjusting trust*. This emerges from the system as a result of the automatic adjustments to the trust levels that service consumers assign to service providers and implementations.

## 5. Empirical Evaluation

Our empirical evaluation consists of a series of simulations tailored to show the emergence of self-adjusting trust. The simulations results reveal how, empirically, our trust model yields a system that autonomically adjusts the level of trust for the service implementations depending on their past quality behaviors. Previous experiments showed that as service qualities drop, the consumers' agents select other service implementations and eventually converge to clean service implementations [22]. In the current experiment we expand on the previous one by adding *explorer agents*. The simulations reveal the importance of the explorer agents in the agent community and also how recent quality information can be captured and be benefited from.

### 5.1. Setup Summary

Briefly, we created three sets of a simple mathematical integer sorting service with one method to sort an array of integers. Each set contains five identical service implementations. Each implementation exposes the qualities: *MethodFaultRate*, *PercentAvailability*, and *MethodResponseTime*. The qualities represent the average fault-rate of service methods, the average service availability (as a percentage), and the average response-time of the service's method. In addition the implementations of each set have identical quality advertisement. We created three groups of five consumers; each group has quality preferences biasing the members to one of the set of service implementations. We named the groups of consumers: *Careful*, *Mellow*, and *Rushed*; indicating their general preferences biased for the three qualities.

We ran various simulations attaching a service agent to each consumer and collected the agent's selection. The service implementations in each group are numbered $0, \ldots, 4$ as are the service consumers. In some of the simulations, we artificially forced all service implementations of group, except the last one (numbered 4), to have its quality degrade. We term this *doping* the implementations. Our results show that the service agent is able to find and select the clean service implementation in a group after some number of iterations to build the necessary reputation of service implementations.

For this experiment we run simulations with a similar setup of service implementations and consumer service agents as in the previous experiment; however, for each simulation we:

- Added *explorer agents*. These are service agents deployed for a specific service interface that periodically run an exploring task on the service implementations. These agents' primary purpose is to explore the community of services regardless of a service's trustworthiness. That is, these agents would select a service implementation whose quality reputation has dropped to a point where it is no longer selected by regular consumer agents, even though more recently its quality characteristics may have improved. The explorer agents do not discriminate and have no preset quality preferences. They select the available service implementations in a round-robin fashion. For each implementation, they execute the same exploring task. They monitor the selected service for all of its advertised qualities and participate in appropriate agencies for these qualities. The net effect is that if a service implementation quality has improved, the explorer agents will help it gain positive reputation. The primary controlling factor for an explorer agent is how frequently it performs its exploring tasks. We name this parameter the *execution frequency* of the explorer agent. Essentially, this value denotes the wait period between each execution of the exploring task. Another controlling factor is the number of explorer agents as a ratio of the total number of agents in the community.
- Varied the service *doping policy*. As in the previous simulations we artificially control the qualities of certain service implementations with a quality doping mechanism. This allows us to have knowledge (as an oracle) to which service implementation should be selected by a service agent at any given time during the simulation. To show the effect of the explorer agents, we also control the duration of the doping. We introduce two new doping parameters:
  1. *Stop doping after*. Specifies the number of times a service will be selected and be under the effect of the quality doping. After the specified number of selections, the service doping will be turned off.
  2. *Restart doping after*. Indicates the number of times the service will be selected before quality doping is resumed. The restart of doping typically occurs in conjunction with stopping doping afterward to simulate the effect of a service that starts out with its quality doped, then stops having its quality affected, and then restarts with a poor quality period. In essence, this simulates a service that behaves incorrectly, then correctly, and then incorrectly once again.

To show the emergence of self-adjusting trust, we conduct six simulations, varying the number of explorer agents and their execution frequency. We also simultaneously vary the above doping parameters to enable us to better predict and measure the effects of the explorer agents. Tables 1 shows an overview of the simulations with regard to the explorer agents and related parameters.

Table 3 shows the doping policy variation for each simulation. The *Doping* column indicates the overall service doping policy used. *Full doping* refers to the same service doping policy as in the previous experiment and summarized in Table 2. The *Stop doping* column indicates the number of service selections after which service doping is stopped. The *Restart doping* column indicates the number of service selections after which service doping is resumed after being stopped.

### 5.2. Results

We now further describe each simulation and its expected results. For each simulation we also show the results obtained in the form of service selection graphs. The simulations are designed to progressively show the impact of the

| Doped quality name | Provider sets | Parameters |
|---|---|---|
| *MethodInvokeTime* | $\{0,1,2,3\}$,$\{5,6,7,8\}$,$\{10,11,12,13\}$ | Random delay where, $10\ ms < delay < 20\ ms$ |
| *FaultRate* | $\{0,1,2,3\}$,$\{5,6,7,8\}$,$\{10,11,12,13\}$ | Random periodic with $period = 3$ |
| *PercentAvailability* | $\{0,1,2,3\}$,$\{5,6,7,8\}$,$\{10,11,12,13\}$ | Random periodic with $period = 4$ |

**Table 2. Full-doping. Doped quality names, doped provider sets, and any other additional doping parameters.**

| Sim | Iter. | Is doped? | No. | Ratio | Exe freq. (ms) |
|---|---|---|---|---|---|
| 1.0, 1.1 | 15 | No | 0 | 0.0 | NA |
| 1.2 | 15 | Yes | 3 | 0.2 | 4500 |
| 1.3 | 10 | Yes | 3 | 0.2 | 6000 |
| 1.4 | 10 | Yes | 3 | 0.2 | 5000 |
| 1.5, 1.6 | 10 | Yes | 3 | 0.2 | 4500 |

**Table 1. Simulations' explorer agents parameters. The columns show the simulation number, the number of iterations, whether the service implementations are doped or not, the number of explorer agents, the explorer agents ratio to the total number of agents in the simulation, and the explorer agents' execution frequency.**

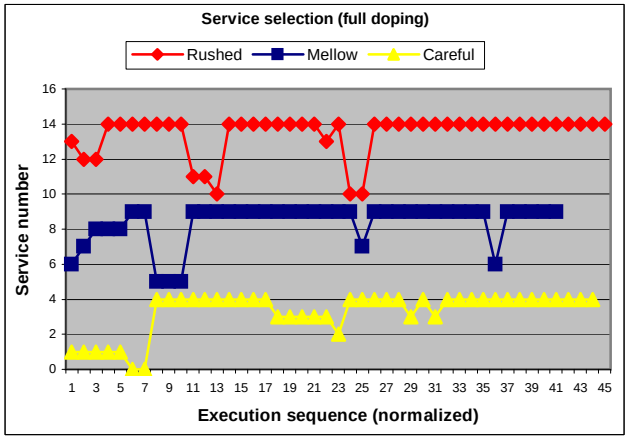


**Figure 2. Selection with full doping.**

| Simulation | Doping | Stop doping | Restart doping |
|---|---|---|---|
| 1.0 | NA | NA | NA |
| 1.1, 1.2 | Full doping | Never | Never |
| 1.3, 1.4, 1.5 | Full doping | After 6 | Never |
| 1.6 | Full doping | After 6 | After 15 |

**Table 3. Simulations' doping policy parameters. The columns show the simulation number, the doping policy, the number of selections after which to stop doping, and the number of selections after which doping is to restart when stopped.**

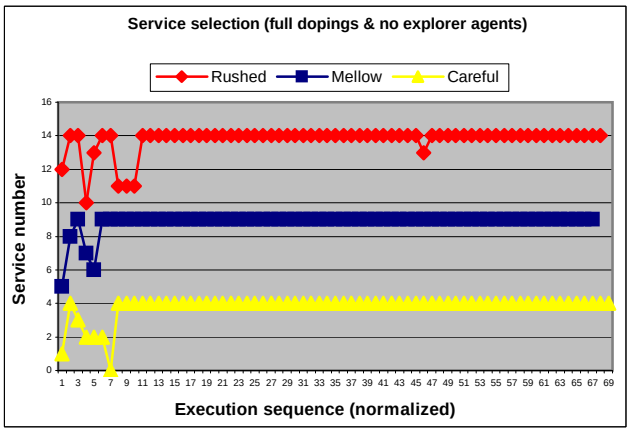


**Figure 3. Full service doping and no explorer agents (Simulation 1.0).**

explorer agents on the system as well as the emergence of self-adjusting trust.

**Simulations 1.0, 1.1, and 1.2: Base line service selection.** These simulations establish the base line service selection with and without explorer agents. As in the previous experiment we expect each group of consumers to eventually converge to the sole clean service implementation [22]. Our expected results for both simulations, with and without explorer agents, should be similar to Figure 2.

Figures 3, 4, and 5 show the results obtained. These

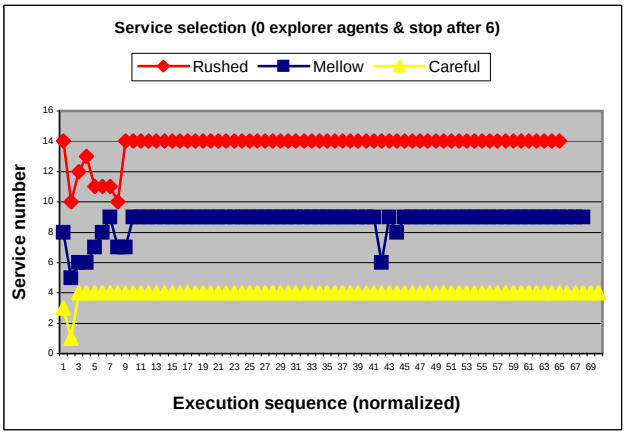


**Figure 4. Full doping, stop doping after six selections, and no explorer agents (Simulation 1.1).**

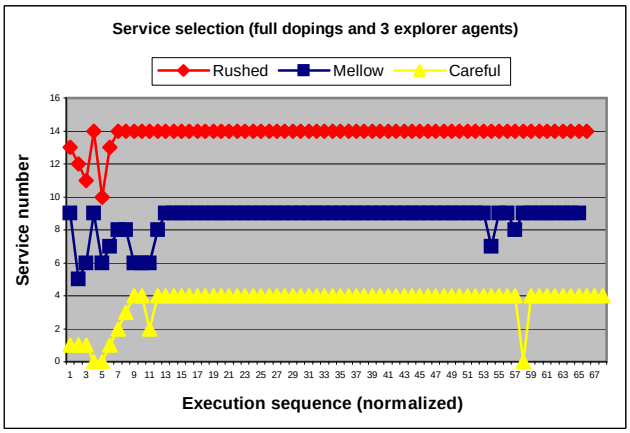


**Figure 5. Full doping and three explorer agents (Simulation 1.2).**

are in agreement with our expectations. The consumer agents, after a period of trying all service implementations in the same pool, eventually find the sole clean service implementation and converge to it. This comes about when no explorer agents are present and the doping lasts the entire simulation (Figure 3) and also when the doping stops (Figure 4). This occurs because the consumer agents never select any of the newly well-behaved service implementations—which accumulated a low reputation already and are never considered.

In Figure 5, even though there are explorer agents

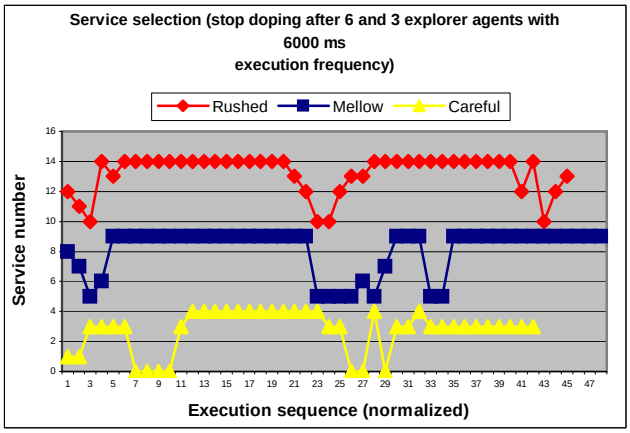


**Figure 6. Full service doping stopped after six and three explorer agents with 6000 ms execution frequency (Simulation 1.3).**

present, since the doping never stops, the sole clean service implementation remains the best alternative for selection during the entire simulation.

**Simulations 1.3, 1.4, and 1.5: Explorer agents (EAgents).** There are two principal goals for simulations 1.3 to 1.5. First, show that when we stop doping the service implementations pool, the use of explorer agents will eventually cause the consumer agents to reselect services that were avoided when they were doped. Second, understand the impact of the explorer agents' execution frequency on the reselection of these newly cleaned service implementations.

Figures 6, 7, and 8 show the service selection results with three explorer agents and execution frequency reduced progressively from 6000, 5000, and then 4500 ms. The first thing to notice is that in each case, the consumer agents first converge to the sole clean service implementation (as in simulations 1.0 and 1.1). Then as the stop doping takes effect—the service implementations start to have good quality—the effect of the explorer agents takes hold and the service implementations start to be selected randomly again. This is the emergence of self-adjusting trust—as these service implementations behave correctly (they are not doped) they are reconsidered for selection.

We also notice that with higher execution frequency, after the service doping is stopped, the newly well-behaved service implementations are more often selected than with lower execution frequency. This can be noticed by comparing Figure 8 with Figure 6 in the part of the graph where reselection is

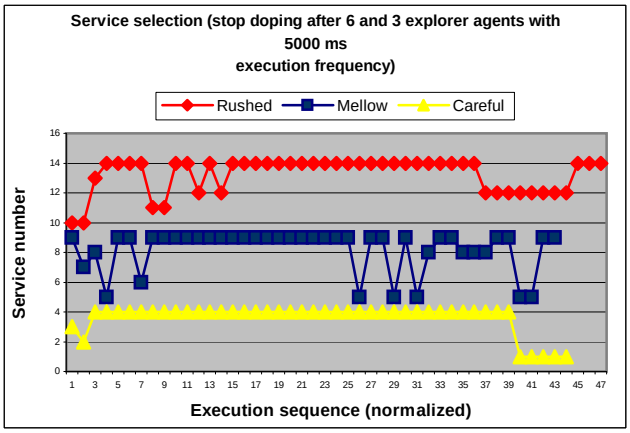


**Figure 7. Full service doping stopped after six and three explorer agents with 5000 ms execution frequency (Simulation 1.4).**

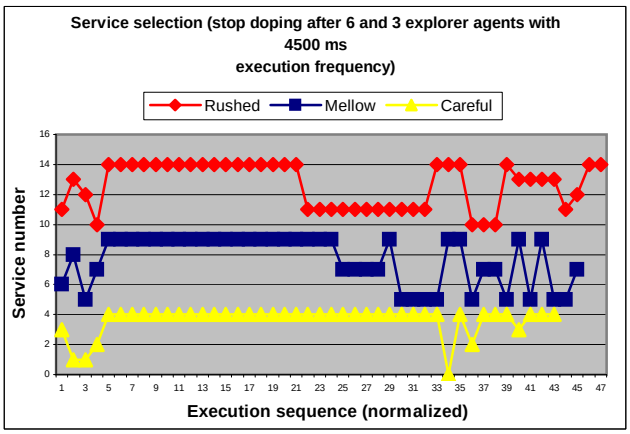


**Figure 8. Full service doping stopped after six and three explorer agents with 4500 ms execution frequency (Simulation 1.5).**

occurring. We conjecture that this is due to faster positive reputation accumulation for the well-behaved service implementations due to frequent explorer agent selection.

**Simulation 1.6: EAgents with doping stopped and restarted.** In this simulation, we use the best setup from Simulations 1.3 to 1.5, but in addition we use a doping restart policy. That is, we simulate service implementations behaving incorrectly again. The restart is set up to occur after the 15th selection. We expect the consumer agents to begin again converging to the sole clean service implementation.

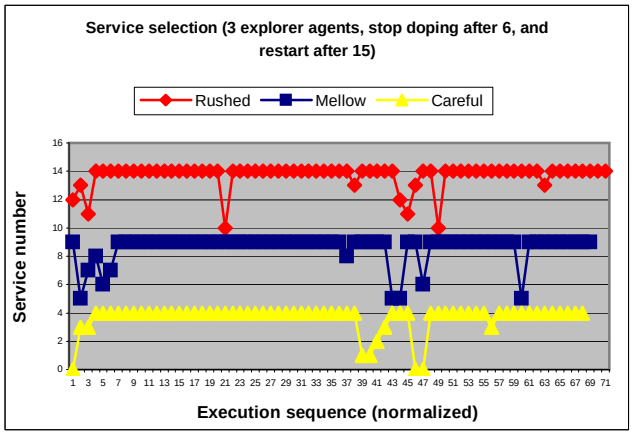


**Figure 9. Three explorer agents with full service doping stopped after six selections and restarted after 15 (Simulation 1.6).**

Figure 9 shows the results of this simulation. As expected, the consumer agents first converge to the sole service implementation. Then as the doping stops they start selecting other service implementations—since the explorer agents help these service implementations rebuild their quality reputation and thus their trustworthiness. As the services doping restart, the consumer agents again gradually converge to the sole clean service implementation. This result is in complete agreement with the emergence of self-adjusting trust.

## 6. Related Work

We divide the works related to ours into three broad categories: QoS-based service selection, trust and trustworthiness, and autonomic computing and multiagent systems (MAS). For each category we discuss the ones that are closest to ours.

### 6.1. QoS-Based Service Selection

Service selection approaches fall into two primary categories: design-time and runtime. In design-time selection, the application designer or architect use service registries coupled with service descriptions to select and test binding to a service. Nonfunctional characteristics are considered during trial and error tests of the selected services. Newer techniques using richer semantic descriptions of services can help in the discovery of service interfaces. OWL-S [30] is an example of a rich service ontology used for semantic service discovery.

In runtime service selection, which is of greater relevance to our work, the service interface is already discovered, but the service implementations must be discovered, selected, and bound to, based on nonfunctional service attributes. Work in this area encompasses QoS requirements, models and metrics, and middleware. The W3C QoS Requirements for Web Services [34] gives an overview of the requirements. The QoS UML profiles described in [1] and [6] are examples of QoS models. Various brokering and middleware architectures [27, 24, 25, 31, 37] have been proposed for using QoS in Web services. Also, in the realm of Grid Services, QoS is an important characteristic used for differentiating the available services on the Grid [2].

Interestingly, van Moorsel et al. also consider higher level of qualities such as Quality of Experience (QoE, the subjective quality perception of the end user) and Quality of Business (QoBiz, the economic characteristic of the service to the service provider) [32]. This work also stresses the importance of relationships among qualities. Similarly, we model relationships between qualities and use such relationships to improve service selection by better computing the preferences of the consumers.

Other researchers have also proposed using QoS for service selection [24, 31, 16, 42]. However, whereas these works address some form of service selection, they do not address it adequately for open environments, such as using trust and reputation and a decentralized multiagent architecture as we are proposing. Wohlstadter et al. [37] propose a policy language for advertising the QoS needs of clients and to allow the middleware to match servers and clients. However, their work lacks a complete conceptualization of nonfunctional attributes for Web services. Further, their matchmaking techniques is not geared to enable dynamic evolution of the QoS exposed by the services. In other words, it does not exhibit autonomic characteristics.

Casati et al. [5] present a dynamic service selection using a data mining approach on the service conversation logs. They assume that conversation logs are collected for each business process execution and that service consumers specify quality goals that can be measured from the logs. Using data mining they provide a middleware that can dynamically analyze the logs of various conversations and determine the services best matching the service consumer's goals. Though similar in intent, our approach differs in two respects. First, in our architecture, we use a multiagent system with an agreed upon QoS ontology, whereas Casati et al. use a centralized middleware and their definitions of qualities is derived from conversation logs. Second, our selection is based on the trust value assigned to a service implementation, which itself is based of a combination of the reputation of the consumer's quality needs. Casati et al.'s selection is based on a service ranking which is derived from the decision trees constructed from the conversation logs.

### 6.2. Trust and Trustworthiness

The literature on trust in the context of Web systems and Web services has been growing lately. The literature closest to our work is summarized in [11]. Though a bit dated, most of the techniques described are applicable to our framework. It is well-known that no security techniques can apply in a purely open environment where identities can be readily created and discarded. In such cases, the best we can do is to use social approaches based on open reputation management. Yu and Singh have examined the robustness of such approaches under certain threats [40].

Other relevant trust approaches are the Pretty Good Privacy (PGP) Web of Trust and the Platform for Internet Content Selection (PICS) [33]. However, these works are mainly in the realm of creating trust systems and platforms for human users and human-facing applications. Our goal is to create trust where human involvement is limited or completely absent.

Work on reputation systems is also closely related to our work. Our proposed reputation calculation extends that of Zacharia and Maes [41] and applies it in the world of Web services by making considerations of QoS explicit. Huynh et al. [14] calculate interaction trust as a reputation value but their approach is limited when applied to service selection (they do not consider nonfunctional characteristics of services) and they also have limited prospect to creating a system that is autonomic. Giorgini et al. [10] present an agent-based social and individual trust model. Their model is mainly in the realm of requirements engineering and gathering; however, the resulting model is a step in the right direction to achieving a formal agent-based trust model and could be used to completement our own. Kalepu et al. [16] extend our own approach to consider variance in collected ratings values and it is also specific to reputation for qualities that are subjectively rated.

Kashyap [17] argues that trust models for information systems and service composition must have a verification process. He gives a partial taxonomy of trust that includes transitive trust (endorsements), reputation (indirect trust), and *a priori* beliefs. Kashyap sketches how a composed trust quality value can be computed from the various trust notions. We differ primarily with this work in that we compute an aggregate trust value from QoS but agree with the fact that a complete model for trust must be multidimensional.

Sloman [29] gives a trust definition that can be used in decision making. Like in our approach, Sloman considers reputation a basic component of trust. He observes that trust is dynamic and context specific and argues that contextual information should be included in the trust model. Sloman's work differs from our approach primarily in that he does not consider qualities as a basis for determining the trustor's

needs. Further, since our quality ontology allows domain-specific qualities to be included in the trust calculation, our trust model addresses the contextual requirement that Sloman raises.

### 6.3. Autonomic Computing and MAS

The overall goal of autonomic computing is outlined by the Autonomic Computing Manifesto [12]. Five levels of AC are outlined with the goal of eventually building systems reaching the highest level: self-management. Section 4 summarizes this vision as it applies to our approach.

Though AC has a well documented vision, actual results in the field remain somewhat scarce. Yellin [39] describes an algorithm to dynamically choose between software component implementations. However, Yellin's approach is limited to closed systems and only considers the case of two competing implementations. Ours is applicable for open systems with a potentially large number of alternative implementations.

Our agents are decision-theoretic in design [26, pp. 413—419]. They extend and apply decision theory as documented in [8]. Our approach determines complex consumer preferences and utility drawing on the seminal work by [18] on making decisions with multiple objectives. We make use of this work in the modern landscape of Web services.

The literature on agents and multiagent systems is vast and varied. Works by [13, 7, 36, 38] give good introductions to agency. More recently [23] and [28] discuss agency in the context of Web services and are thus closest to our framework and approach.

Other multiagent frameworks are also being considered in the Web services arena. These include the well-established JADE platform [15, 3] and the lesser known Bond [4] MAS framework. We see two key differences between these works and ours. First, our agents are solely designed with Web services in mind, as such, the Web services infrastructure is built-in. In fact, the Web services infrastructure constitutes the core of our framework. Second, our framework and agents are geared toward creating a trust model that achieves intelligent selection where autonomy is an emergent property. This is a novel approach to service selection and novel application of agency.

## 7. Conclusions

The autonomic characteristic of *self-adjusting trust* is crucial in open systems. We showed how explorer agents enable our trust model and multiagent system to exhibit this characteristic. In this manner, we can enable the benefits of autonomic computing for a wide range of settings.

Further aspects of the emergence of self-adjusting trust remain to be studied. These include, for instance, understanding the density of explorer agents needed to achieve a certain level of self-adjusting trust in the presence of services advertising some qualities. Since explorer agents consume precious resources, an analytical model of their behavior in the community with respect to the resulting perceived level of self-adjusting trust would be highly valuable. Such a model would depend upon a characterization of the variability in practice in the behavior of service implementations.

The full exploitation of explorer agents may require the emergence of new standards. Specifically, explorer agents would not automatically be able to use services that require a consumer's private data or require payments. Agents that can perform such tests would be difficult to set up and specialized agencies may emerge to evaluate expensive services on the behalf of prospective consumers. However, if such services expose an *exploration interface* that enables exploration and evaluation of their offerings but without requiring payment, they would facilitate explorer agents. An exploration interface would be useful to a provider, because it would yield greater exposure to their offerings. An associated aspect is that the consumers will have to trust that the behavior obtained during exploration is representative of the behavior to be expected during the actual interaction.

Finally, a direction for our work is to improve the trust model to take into account provider's trustworthiness. Providers can be trusted by consumers *a priori* using implicit trust as well as indirect and transitive trust such as endorsements from trusted third parties [20].

## References


[1] J. Ø. Aagedal, M. A. de Miguel, E. Fafournoux, M. S. Lund, and K. Stolen. UML Profile for Modeling Quality of Service and Fault Tolerance Characteristics and Mechanisms. Technical Report 2004-06-01, Object Management Group, June 2004.

[2] R. J. Al-Ali, O. F. Rana, D. W. Walker, S. Jha, and S. Sohail. G-QoSM: Grid Service Discovery Using QoS Properties. *Computing and Informatics Journal*, 21(4):363–382, 2002.

[3] F. Bellifemine, A. Poggi, and G. Rimassa. JADE: A FIPA-compliant agent framework. In *Practical Application of Intelligent Agents and Multi-Agent Technology (PAAM)*, pages 97–108, 1999.

[4] Bond. Bond: A Multi-agent System, 2002.

[5] F. Casati, M. Castellanos, U. Dayal, and M.-C. Shan. Probabilistic, Contex-Sensitive, and Goal-Oriented Service Selection. In *Proceedings of 2nd International Conference on Service Oriented Computing (ICSOC)*, pages 316–321, New York, Nov. 2004. ACM Press.

[6] V. Cortellessa and A. Pompei. Towards a UML Profile for QoS: A Contribution in the Reliability Domain. In *Proceed-*

*ings of the fourth International Workshop on Software and Performance*, pages 197–206. ACM Press, 2004.

[7] J. Ferber, editor. *Multi-Agent Systems: An Introduction to Distributed Artificial Intelligence*. Addison-Wesley, Great Britain edition, 1999.

[8] S. French. *Decision Theory: An Introduction to the Mathematics of Rationality*. John Wiley and Sons Inc., West Sussex, England, 1986.

[9] A. G. Ganek and T. A. Corbi. The Dawning of the Autonomic Computing Era. *IBM Systems Journal*, 42(1):5–18, Jan. 2003.

[10] P. Giorgini, F. Massacci, J. Mylopoulos, and N. Zannone. Modeling Social and Individual Trust in Requirements Engineering Methodologies. In *Third International Conference on Trust Management*, Lecture Notes on Computer Science. Springer Verlag, 2005. To appear.

[11] T. Grandison and M. Sloman. A Survey of Trust in Internet Application. *IEEE Communications Surveys & Tutorials*, 3(4):2–16, 2000.

[12] P. Horn. Autonomic Computing: IBM's Perspective on the State of Information Technology, 2001.

[13] M. Huhns and M. P. Singh, editors. *Readings in Agents*. Morgan Kaufmann, San Francisco, CA, 1998.

[14] D. Huynh, N. R. Jennings, and N. R. Shadbolt. Developing an Integrated Trust and Reputation Model for Open Multi-Agent Systems. In *Proceedings of 7th International Workshop on Trust in Agent Societies*, pages 66–74, New York, July 2004. ACM Press.

[15] JADE. Java Agent DEvelopment Framework (JADE), 1999.

[16] S. Kalepu, S. Krishnaswamy, and S. W. Loke. Verity: A QoS Metric for Selecting Web Services and Providers. In *Proceedings of the Fourth International Conference on Web Information Systems Engineering Workshops (WISEW'03)*, pages 131–139, Rome, Dec. 2003. IEEE Computer Society.

[17] V. Kashyap. Trust, But Verify: Emergence, Trust, and Quality in Intelligent Systems. *IEEE Intelligent Systems*, 19(5):85–87, Sept. 2004.

[18] R. L. Keeney and H. Raiffa. *Decisions with Multiple Objectives: Preferences and Value Tradeoffs*. John Wiley & Sons, Hoboken, NJ, 1976.

[19] J. O. Kephart and D. M. Chess. The Vision of Autonomic Computing. *IEEE Computer*, 36(1):41–50, Jan. 2003.

[20] E. M. Maximilien and M. P. Singh. Reputation and Endorsement for Web Services. *SIGecom Exchanges*, 3(1):24–31, Dec. 2001.

[21] E. M. Maximilien and M. P. Singh. A Framework and Ontology for Dynamic Web Services Selection. *IEEE Internet Computing*, 8(5):84–93, Sept. 2004.

[22] E. M. Maximilien and M. P. Singh. Toward Autonomic Web Services Trust and Selection. In *Proceedings of 2nd International Conference on Service Oriented Computing (ICSOC)*, pages 212–221, New York, Nov. 2004. ACM Press.

[23] OWL-S. OWL-Service Ontology 1.0, 2004.

[24] S. Ran. A Framework for Discovering Web Services with Desired Quality of Service Attributes. In L.-J. Zhang, editor, *Proceedings of the International Conference on Web Services*, pages 208–213, Las Vegas, NV, June 2003. IEEE Computer Society.

[25] S. Ran. A Model for Web Services Discovery with QoS. *SIGecom Exchanges*, 4(1):1–10, 2003.

[26] S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, Upper Saddle River, NJ, 1st edition, 1995.

[27] A. Sheth, J. Cardoso, J. Miller, and K. Kochut. QoS for Service-Oriented Middleware. In *Proceedings of the 6th World Multiconference on Sytemics, Cybernetics, and Informatics (SCI02)*, volume 8, pages 528–534, Orlando, FL, July 2002.

[28] M. P. Singh and M. N. Huhns. *Service-Oriented Computing: Semantics, Processes, Agents*. John Wiley & Sons, Chichester, UK, 2005.

[29] M. Sloman. Trust Management in Internet and Pervasive Systems. *IEEE Intelligent Systems*, 19(5):77–79, Sept. 2004.

[30] K. Sycara, M. Paolucci, A. Ankolekar, and N. Srinivasan. Automated Discovery, Interaction, and Composition of Semantic Web Services. *Journal on Web Semantics*, 1(1):27–46, Sept. 2003.

[31] M. Tian, A. Gramm, T. Naumowicz, H. Ritter, and J. Schiller. A Concept for QoS Integration in Web Services. In *Proceedings of the Fourth International Conference on Web Information Systems Engineering Workshops (WISEW'03)*, pages 149–155, Rome, Dec. 2003. IEEE Computer Society.

[32] A. van Moorsel. Metrics for the Internet Age: Quality of Experience and Quality of Business. Technical Report HPL-2001-179, Hewlett-Packard, Erlangen, Germany, July 2001.

[33] W3C. Platform for Internet Content Selection (PICS), 1998. Recommendation.

[34] W3C. QoS for Web Services: Requirements and Possible Approaches, Nov. 2003. Note.

[35] D. D. Wackerly, W. M. III, and R. L. Scheaffer. *Mathematical Statistics with Applications*. Duxbury, Pacific Grove, CA, 6th edition, 2002.

[36] G. Weiss, editor. *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence*. MIT Press, Cambridge, MA, 1999.

[37] E. Wohlstadter, S. Tai, T. Mikalsen, I. Rouvellou, and P. Devanbu. GlueQoS: Middleware to Sweeten Quality-of-Service Policy Interactions. In *Proceedings of 26th International Conference on Software Engineering (ICSE 2004)*, pages 189–199, Edinburgh, May 2004. IEEE Computer Society.

[38] M. Wooldridge. *An Introduction to Multiagent Systems*. John Wiley & Sons, West Sussex, England, 2002.

[39] D. M. Yellin. Competitive Algorithms for the Dynamic Selection of Component Implementations. *IBM Systems Journal*, 42(1):85–97, Jan. 2003.

[40] B. Yu and M. P. Singh. An Evidential Model of Distributed Reputation Management. In *Proceedings of Autonomous Agents and Multi Agent Systems (AAMAS)*, pages 294–301. ACM Press, 2002.

[41] G. Zacharia and P. Maes. Trust Management Through Reputation Mechanisms. *Applied Artificial Intelligence*, 14:881–907, 2000.

[42] L. Zeng, B. Benatallah, A. H. H. Ngu, M. Dumas, J. Kalagnanam, and H. Chang. QoS-Aware Middleware for Web Services Composition. *IEEE Transactions on Software Engineering*, 30(5):311–327, May 2004.