focus

service-centric software systems

Improving Web Service Discovery with Usage Data

Aliaksandr Birukou, Enrico Blanzieri, Vincenzo D'Andrea, Paolo Giorgini, and Natallia Kokash, *University of Trento*

A recommendation system to help service-based application developers discover appropriate services uses a task description and the history of previous decisions made for similar objectives.

Service-oriented computing and Web services are becoming more popular, enabling organizations to use the Web as a market for selling their own services and consuming existing services from others. Nevertheless, the more services are available, the more difficult it becomes to find the most appropriate service for a specific application. Existing approaches to Web service discovery tend to address different information-processing styles. For example, some approaches develop extensive service-description and publication mechanisms;[1] others use syntactic, semantic, and structural reviews of Web service specifications.[2]

However, Web services have functional and nonfunctional characteristics that can be difficult to present and control. Service behavior and quality-of-service (QoS) parameters can vary over time, and new services can emerge in certain business areas. So, despite the availability of various tools, service-based application developers who want to discover new Web services often base their selections on information from business partners, experts in the field, friends, or other people who have had experience with a certain service.

To support such information exchange, several proposals for applying recommendation systems to Web services discovery and selection have recently appeared.[3–6] Most recommendation systems manage information about clients and items by collecting feedback from clients and rating items. Existing recommendation-based approaches to rating Web service providers collect feedback based on explicit and often subjective opinions of service clients.[6] However, research has shown that people are not usually willing to actively provide feedback.[7]

We have developed a recommendation system that lets a service-based application developer benefit from other developers' experience without asking them personally to participate in evaluating services. The overall approach is to connect service requests with observation data from the service invocations and executions that follow such requests. Data collected during observations are the input to identify which services are considered relevant for specific requests of a particular developer community. Additionally, data about service execution can help rank services according to their QoS. The only effort requested of developers is to enable observations of the Web service invocations their applications perform. In exchange for this, they can access the history of service executions and obtain recommendations about which services to use for their tasks. This kind of information can be particularly useful for supporting self-healing behaviors in dynamically reconfigurable systems.

0740-7459/07/$25.00 © 2007 IEEE

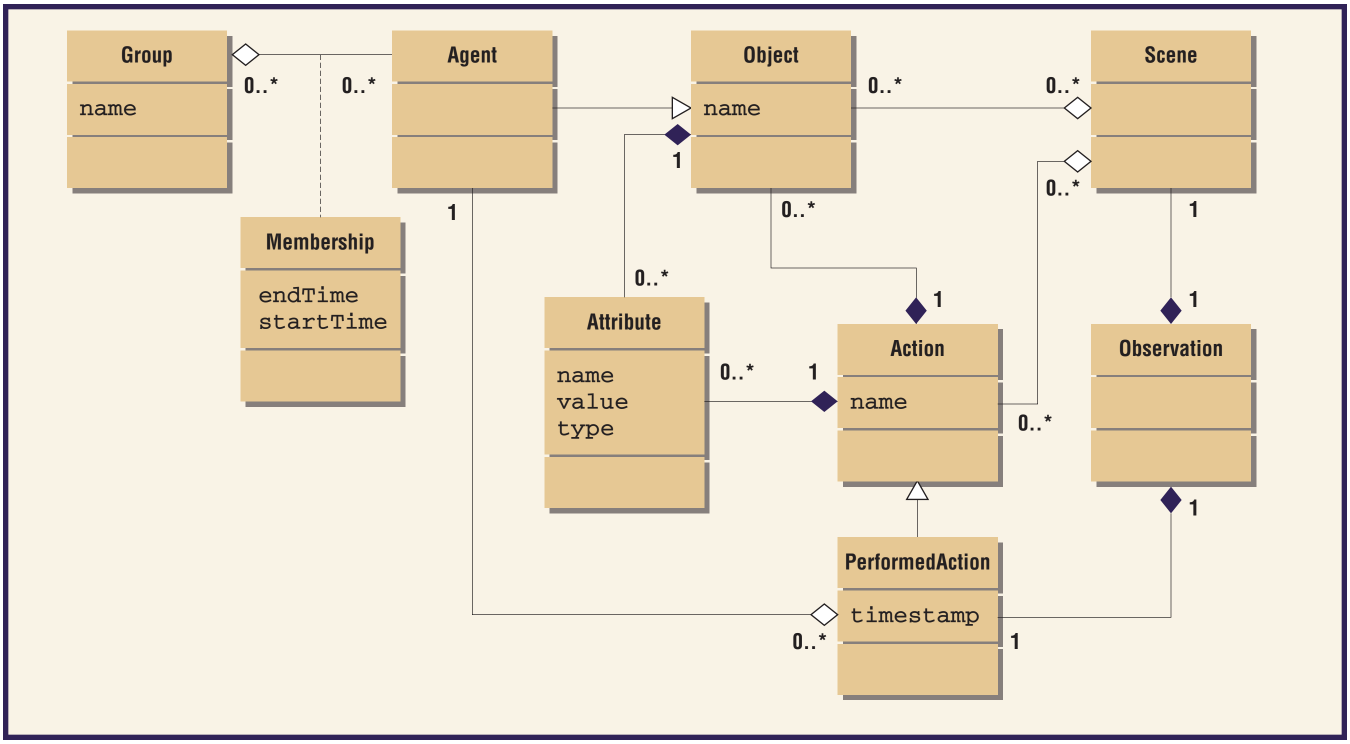

Figure 1. The metamodel of implicit-culture core concepts.

The recommendation system application we present here extends our previous work on *IC-Service*,[8,9] a general context-independent recommendation service. IC-Service, in turn, is based on our work in *implicit culture*, a concept that defines a relation between a set and a group of agents such that the set's elements behave according to the group's culture.[10] Specifically, we present details here about the conceptual and algorithmic basis of the Web service discovery method. We also present experimental performance results for the application using two similarity metrics: one syntactic and one semantic.

Implicit culture: Concepts and implementation

When searching for a service to perform a specific task, a developer or autonomous system might lack knowledge about available services and their actual behavior. Other users who have previously faced similar needs might know suitable services and have experience-based preferences about which of them to use. Such implicit knowledge exists in various application areas and can be used to make practical recommendations. The implicit-culture approach to decision support assumes that it's possible to elicit this knowledge by observing the behavior of the involved parties and then encouraging newcomers to behave similarly to more experienced members of a community.

Figure 1 shows the metamodel of the implicit-culture core concepts. It describes an environment in terms of agents that perform actions on objects. An *object* is defined by its name and a set of related attributes. An *attribute* represents additional information about objects, actions, or agents and consists of a name, a value, and the value's type. An *agent* is a particular type of object that can perform actions. Several agents can compose a *group*. The metamodel describes a possible restriction of an agent's *membership* in a group in time. An *action* is characterized by its name, a set of related attributes, and a set of related objects. Each *performed action* is a specific kind of action that contains a time stamp and the action's agent. The metamodel considers the actions in the context of situations—each of which is represented by a *scene* that includes the set of possible actions and the set of objects the agents can operate with. After the agent performs one of the possible actions, the performed action and the scene constitute an *observation*.

In our application, agents are developers or service-based applications that submit requests for Web-service operations represented as objects. The recommendation system stores Web services' names and information about their providers as attributes of operations, while it models client requests, service invocations, and corresponding responses as actions. For example, a scene could be a set of actions corresponding to the invocation of various service operations:

invoke(...; getWeatherByZip
(*service* = DOTSFastWeather); ...)

or

invoke(...; getWeather
(*service* = GlobalWeather); ...)

An example performed action could be

invoke(Peter; getWeatherByZip
(*service* = DOTSFastWeather);
25-Jun-07-14:22)

which states that Peter invoked the operation getWeatherByZip of the DOTSFastWeather Web service on 25 June 2007 at 14:22.

The developer community relies on information about actions and their relation to situations—namely, which actions the observed group usually takes in which situations. We designed the System for Implicit Culture Support (SICS, see figure 2) to use this information to give newcomers information about other community members' behavior in similar situations. When newcomers start to behave similarly to the community culture, a transfer of knowledge has occurred and is reflected in the implicit culture.[10] In the weather example, the implicit culture can contain the information about which services a community of service clients usually invokes for getting a weather forecast. The SICS performs the knowledge transfer that establishes the implicit culture. It consists of three main layers:

- The SICS Core stores observations, manages theory, and facilitates actions by suggesting scenes.
- The SICS Remote Module implements protocols for information exchange with the client and converts the SICS Core's objects into a format compatible with these protocols.
- The SICS Remote Client provides a simple Java interface for the remote clients, releasing them from dealing with information exchange protocols.

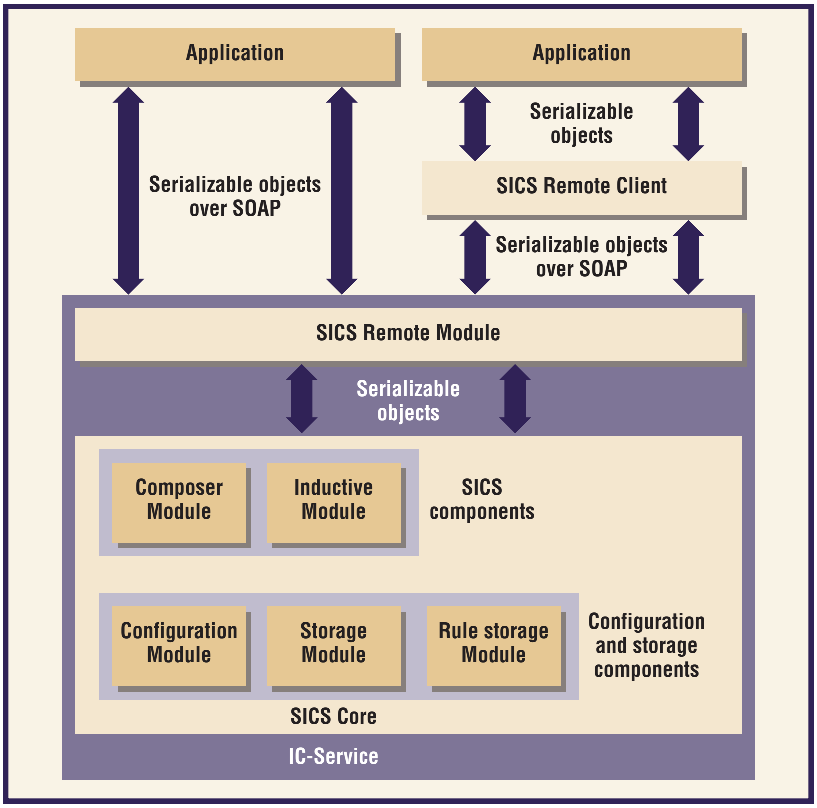

Figure 2. The System for Implicit Culture Support general architecture. The Composer Module provides recommendation facilities; the Inductive Module discovers a theory that expresses the community culture; the Configuration Module configures all parameters of a SICS instance; the Storage Module stores information about the application domain (agents, actions, observations, and so on); the Rule Storage Module manages the theory (for example, adding or removing theory rules).

Locally run applications can use the SICS recommendation facilities as a Java library. Distributed systems can access the SICS Core as an Enterprise JavaBeans component or as the IC-Service recommendation service. In the case of IC-Service discovery, the SICS is deployed as a Web service and accessed via the SICS Remote Client.

The SICS produces recommendations according to the specified rules-based cultural theory. The SICS administrator can adjust the recommendation strategy by configuring theory rules essentially defined in the form

if *antecedent* then *consequent*

where the consequent and the antecedent consist of one or several predicates. This rule reflects the intuitive notion that if the antecedent happened, then the consequent happened and will happen again. A cultural theory for Web service discovery consists of a rule

if *submit_request*(...) then *invoke*(...)

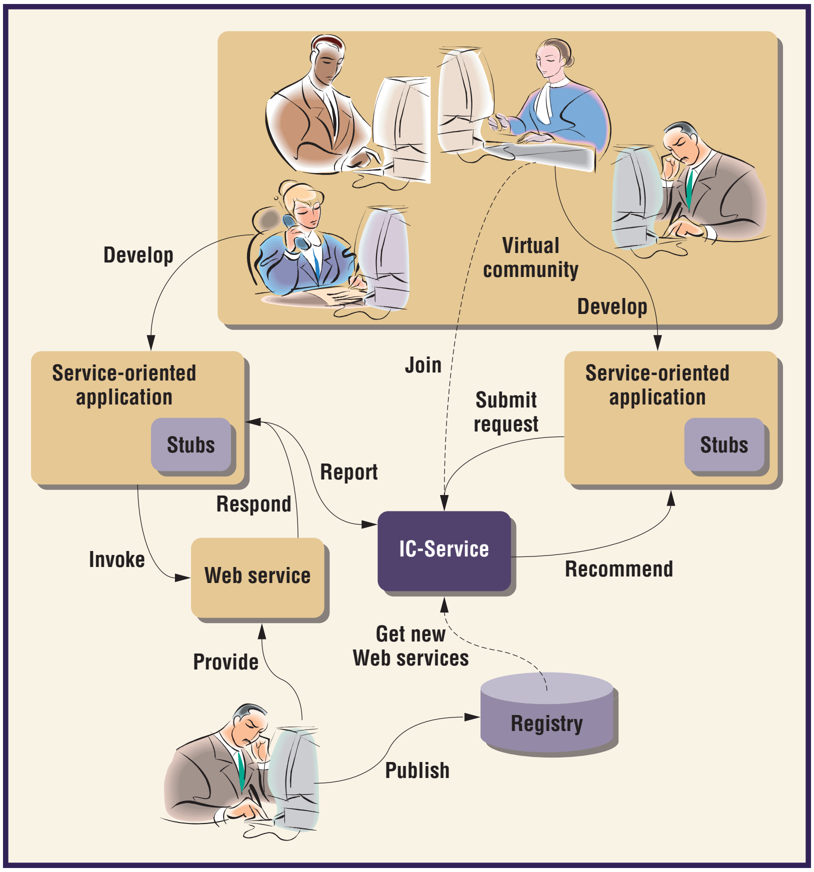

Figure 3. Web service monitoring and discovery with IC-Service.

where the dots refer to specific objects or attributes.

The Composer Module (CM) uses the described theory rules to analyze observations of agent behaviors. When an agent performs an action, the CM matches the observation corresponding to the action with the antecedent part of the theory. Basically, at this step, the CM selects the rule with the antecedent part most suitable for the current observation. Then it uses the information in the observation to instantiate variables in the corresponding rule's consequent. The consequent is called a *cultural action*, and the system uses it for recommending scenes where actions similar to the cultural action happened. The IC-Service provides a simple algorithm that calculates the similarity between pairs of actions using predefined similarity values for action names, time stamps, agents, objects, and attributes. A specific XML-based language lets us configure these values for each particular type (action, object, agent, or attribute) of each particular instance, element, or pair of elements. Moreover, if an application requires a custom algorithm for calculating similarity between certain kinds of elements, the SICS administrator can substitute the default similarity-assessment algorithm with the specialized one using provided configuration facilities.

The Web service discovery system

The IC-Service manages the history of Web services requests in our application system. It also collects reports about heterogeneous clients' service invocations and helps developers discover and select services suitable for their applications. Figure 3 describes the overall architecture, including the role of the IC-Service. To join a community that shares service usage experience, a developer must include the SICS Remote Client in his or her application to enable client-side monitoring of Web service invocations.

In general, creating a Web services recommendation system with the IC-Service requires the following steps:

1. Formalize the application domain in the implicit-culture terms.
2. Define the cultural theory.
3. Define similarity-calculation algorithms.

We now explain these steps in more detail before providing an example to illuminate how the system works.

Application domain

In our system, agents are software developers or service-based applications willing to find a Web service. Figure 3 shows a working scenario, in which an agent submits a request to the IC-Service, which returns a list of recommended services. The request contains a textual description of the goal, the desired operation's name, a description of its I/O parameters, a description of a desired Web service, and an optional list of preferred features, such as a provider. The system stores the request as an object of the submit_request action. It also collects the feedback via the optional provide_feedback action, which expresses the agent's level of satisfaction with the result, or via the invoke action, which marks a service as suitable for the request. If the agent decides to use one of the services, the system acquires further information. The get_response action marks a service as available and the raise_exception action signals that the service is either not available or faulty. Having received the response message, the application can generate a feedback on the basis of extra knowledge about the expected result. For example, the feedback is positive if the application obtains the correct output.

Cultural theory

The IC-Service processes the request from the system in two steps. First, it matches the submit_request action with the theory to determine the next action that must follow—that is, the invoke action. Second, the SICS finds situations where the invoke action was previously performed and determines Web service operations used for similar requests. In this step, the system calculates the similarity between the current agent's request and the previously submitted requests. As a result, the IC-Service returns a set of services that have been used for similar requests in the past. The cultural theory rule can be written as follows:

if *submit_request*(request-X) then *invoke* (operation-Y(service-Z), request-X)

This means that the invoke action must follow the submit_request action, and both actions are related to the same request. The administrator can use the IC-Service cultural theory definition language to specify other requirements as well. For example, the following rule

if *submit_request*(request-X(cost= "low")) then *invoke*(operation-Y(service-Z), request-X) ∧ *provide_feedback* (operation-Y(service-Z), cost = "low")

means that if the agent requests a service with a low cost, the system will recommend services that other clients considered cheap.

Creating recommendations

The recommendation process consists of three steps:

1. Find the cultural theory rule that matches the current observation.
2. Find the corresponding cultural action.
3. Find the set of scenes where the cultural action is likely to be performed.

When an agent performs the submit_request action, the observation corresponding to the action is passed into the SICS where it is matched with the antecedent part of the theory. The CM uses the observation's information about the request to instantiate the consequent rule's variables. The corresponding cultural action could be, for instance, invoke(..., request-X). At the next step, for the given cultural action, the CM finds the set of agents that performed similar actions and the set of scenes where actions have been performed. Then it selects a set of agents most similar to the agent that submitted the request and updates the set of scenes accordingly. Next, it calculates the agents' similarity on the basis of their past actions. Finally, the CM selects scenes where the cultural action is most likely to occur and recommends Web services from the scenes to the request's originator.

Further details on this process are available elsewhere.[10]

The Composer Module selects scenes where the cultural action is most likely to occur and recommends Web services from them.

Similarity configuration

The similarities of names, attributes, and objects determines the similarity between observed actions (such as submit_request, invoke, and so on). The main element our system takes into account is the request represented as a sequence of terms $q = (t_1, t_2, \ldots, t_{|q|})$, where $|q|$ is the length of the request, $t_j \in T$, and

$$j \in \{1, \ldots, |q|\}$$

T is a vocabulary of all terms from the collection of requests $Q = \{q_1, \ldots, q_n\}$ that the agent submits to the system, where $|q|$ is the total number of requests. We use two different similarity metrics in the SICS Composer Module to calculate the similarity between requests:

- the Vector Space Model (VSM) with Term Frequency-Inverse Document Frequency (TF-IDF) metric and
- a WordNet-based semantic similarity metric.

To calculate the first metric, for each term t_j, n_{ij} denotes the number of occurrences of t_j in q_i, and n_j denotes the number of requests that contain t_j at least once. We can obtain the TF-IDF weight of the term t_j in the request q_i as follows:

$$w_{i,j} = \frac{n_{i,j}}{|q_i|} * \log\left(\frac{n}{n_j}\right)$$

where $|q_i|$ defines the length of the request q_i.

We use a WordNet-based metric to define a lexical similarity for all possible senses of two terms.

The similarity between requests q_i and q_k is defined using the cosine coefficient:

$$sim(q_i, q_k) = \cos(w_i, w_k) = \frac{w_i^T w_k}{\sqrt{w_i^T w_i}\sqrt{w_k^T w_k}}$$

Here, $w_i = (w_{i1}, \ldots, w_{i|T|})$, $w_k = (w_{k1}, \ldots, w_{k|T|})$ denote vectors of TF-IDF weights corresponding to the requests q_i and q_k, and $|T|$ is the length of the vocabulary T.

For the second metric, we define a similarity between requests

$$q_i = \left(t_{i1}, t_{i2}, \ldots, t_{i|q_i|}\right)$$

and

$$q_k = \left(t_{k1}, t_{k2}, \ldots, t_{k|q_k|}\right)$$

by first calculating the lexical similarity between any pair of their terms

$$t_{ij}, j \in \{1, \ldots, |q_i|\}$$

and

$$t_{kl}, l \in \{1, \ldots, |q_k|\}$$

We use the WordNet-based metric designed by Nuno Seco, Tony Veale, and Jer Hayes to define a lexical similarity for all possible senses of two terms:[11]

$$sim(t_{ij}, t_{kl}) = 1 - \frac{1}{2}\left(ic_{wn}(t_{ij}) + ic_{wn}(t_{kl}) - 2sim_{res'}(t_{ij}, t_{kl})\right)$$

where

$$sim_{res'}(t_{ij}, t_{kl}) = \max_{t \in S(t_{ij}, t_{kl})} ic_{wn}(t)$$

In this expression, $S(t_{ij}, t_{kl})$ denotes the set of concepts that subsume t_{ij} and t_{kl}, and *ic* denotes a WordNet concept's information content value, which is defined as

$$ic_{wn}(t) = 1 - \frac{\log(hypo(t) + 1)}{\log(\max_{wn})}$$

Here, *hypo* is the number of *hyponyms*—that is, words whose meaning is included within that of another word of a given concept, and $\max_{wn}$ is the maximum number of existing concepts.

We extended this metric to deal with sets of words. We formulated the problem of calculating the overall similarity between requests q_i and q_k as the Maximum Weight Bipartite Matching problem in a complete bipartite graph, where terms from q_i and q_k define two partitions of vertices and the obtained lexical similarity values sim(t_{ij}, t_{kl}) define weights of edges. The goal is to find a matching—that is, a subset of edges $e_{jl} = (t_{ij}, t_{kl})$ such that no two edges in M share a vertex—with the maximum total weight. This weight defines the similarity between requests q_i and q_k. Specifically,

$$sim(q_i, q_k) = \max_M \sum_{j=1}^{|q_i|}\sum_{l=1}^{|q_k|} \sigma_{jl}\, sim(t_{ij}, t_{kl})$$

where

$$\sigma_{jl} = \begin{cases} 1, \text{ if edge } e_{jl} = (t_{ij}, t_{kl}) \in M \\ 0, \text{ otherwise} \end{cases}$$

Example

We can now illustrate how the search process takes place in practice. Suppose the SICS receives the following request:

goal: get weather forecast for Rome, Italy;
operation: get weather;
input: city name, country name;
output: weather forecast (temperature, humidity, and so forth).

The SICS matches the request action with the theory's antecedent and searches for scenes that have performed the invoke action.

Suppose it finds the following situations:

1. *invoke*(…; getWeather (*service* = GlobalWeather), *goal* = get weather report for all major cities around the world; …);
2. *invoke*(…; conversionRate (*service* = CurrencyConvertor), *goal* = get conversion rate from one currency to another currency; …);
3. *invoke*(…; getWeatherByZip (*service* = DOTSFastWeather), *goal* = return the weather for a given US postal code; …).

The SICS recommends invoking services that it considers relevant, according to previously observed requests that are most similar to the current request. So, in this example, SICS recommends the getWeather operation of the GlobalWeather Web service and the getWeatherByZip operation of the DOTSFastWeather service.

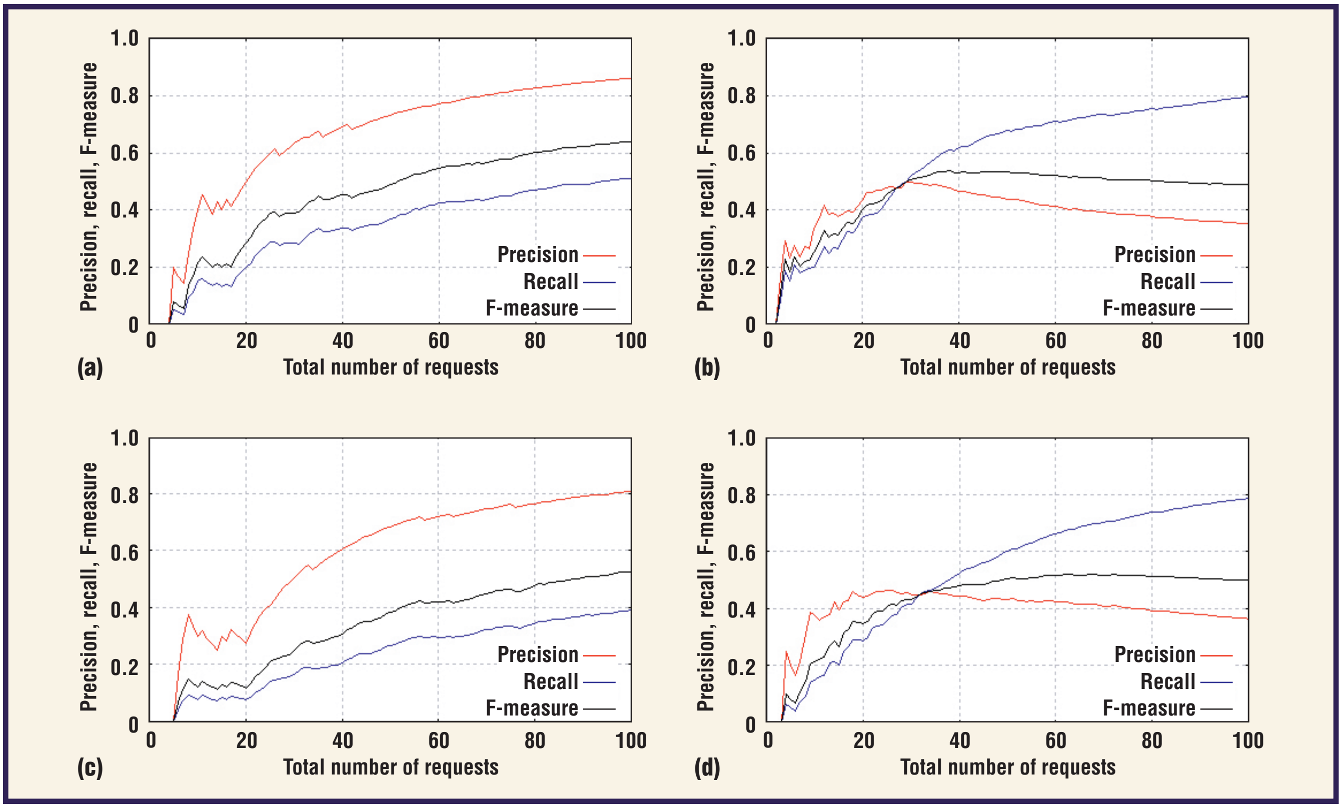

Figure 4. System performance in four scenarios: (a) VSM with TF-IDF, four clients; (b) WordNet-based semantic similarity metric, four clients; (c) VSM with TF-IDF, 20 clients; (d) WordNet-based semantic similarity metric, 20 clients.

Suppose that after analyzing the proposed results, the agent invokes the getWeather operation. After observing the invoke action, SICS will mark this service as suitable for the submitted request and, in particular, for the lexical terms that occurred in the request. Now, given a request for a service providing information about Italy, the system is likely to suggest the GlobalWeather Web service.

Experimental evaluation

We performed preliminary experiments to evaluate system performance in terms of precision, recall, and F-measure. Precision measures the fraction of relevant items among those recommended, recall measures the fraction of the relevant items included in the recommendations, and F-measure is a trade-off between precision and recall:

$$\text{Precision} = (\text{Relevant} \cap \text{Retrieved}) / \text{Retrieved}$$

$$\text{Recall} = (\text{Relevant} \cap \text{Retrieved}) / \text{Relevant}$$

$$F = (2 * \text{Precision} * \text{Recall}) / (\text{Precision} + \text{Recall})$$

We used a collection of 20 Web services from xMethods service registry (http://xMethods.com) and divided them into five topic categories. For each category, we found four semantically equivalent operations and formed 20 requests based on their short natural language descriptions from Web Service Definition Language files.

We defined user profiles to simulate real users' behavior. A user profile contains a set of requests and a set of Web service operations relevant to these requests. We simulated a user's request-generation behavior by choosing and submitting one of the requests randomly. We simulated the result-selection behavior by choosing and invoking one of the service operations to perform the task. The intuition behind user profiles is that users will submit a request for a service operation and, after receiving suggestions, will invoke one of the operations they consider to be relevant. The SICS Remote Client monitors the invocation. During the simulation, we used a random selection to choose which user submitted a request to the system in a given moment.

Figure 4 shows precision, recall, and F-measure results for 100 recommendation requests submitted to the system under four scenarios: two for each similarity metric with four and 20

About the Authors

Aliaksandr Birukou is a PhD candidate in the University of Trento's Department of Information and Telecommunication Technology. His research interests are in data mining, multiagent systems, and recommendation systems, and his professional experience includes banking software development. He received his MS with distinction in applied mathematics and computer science from the Belarusian State University, Minsk. Contact him at the Dept. of Information and Communication Technology, Univ. of Trento, via Sommarive 14, 38050 Povo (Trento), Italy; aliaksandr.birukou@dit.unitn.it.

Enrico Blanzieri is an assistant professor in the University of Trento's Department of Information and Telecommunication Technology. His research interests include soft computing, artificial intelligence, and bioinformatics. He received his PhD in cognitive science from the University of Turin. Contact him at the Dept. of Information and Communication Technology, Univ. of Trento, via Sommarive 14, 38050 Povo (Trento), Italy; enrico.blanzieri@dit.unitn.it.

Vincenzo D'Andrea is an associate professor in the University of Trento's Department of Information and Telecommunication Technology. His research interests include service-oriented computing, free and open source licensing, and virtual communities. He received his PhD in information technology from the University of Parma. He's a member of the IEEE Computer Society and the ACM. Contact him at the Dept. of Information and Communication Technology, Univ. of Trento, via Sommarive 14, 38050 Povo (Trento), Italy; vincenzo.dandrea@dit.unitn.it.

Paolo Giorgini is a researcher at University of Trento. His research interests include the development of requirements and design languages for agent-based systems and the application of knowledge representation techniques to software repositories and software development. He received his PhD in computer science from the Computer Science Institute of University of Ancona, Italy. He's co-editor in chief of the *International Journal of Agent-Oriented Software Engineering.* Contact him at the Dept. of Information and Communication Technology, Univ. of Trento, via Sommarive 14, 38050 Povo (Trento), Italy; paolo.giorgini@dit.unitn.it.

Natallia Kokash is a PhD candidate in the University of Trento's Department of Information and Telecommunication Technology. Her research interests include information retrieval, service-oriented computing, and software engineering. She received her MS in information science from the Belarusian State University, Minsk. She's a member of the ACM. Contact her at the Dept. of Information and Communication Technology, Univ. of Trento, via Sommarive 14, 38050 Povo (Trento), Italy; natallia.kokash@dit.unitn.it.

clients each. For the TF-IDF syntactic similarity metric, the results show performance increases for all measures as the number of user requests increases. For the WordNet-based semantic similarity metric, precision decreases slightly after some point. This is because the lexical similarity used to match requests is too generic, so the system produces false positive recommendations. However, the semantic metric's recall is significantly higher than that of the syntactic metric.

We plan to extend our application to deal with other important information about Web services, such as QoS parameters. Also, we have not specifically addressed security and privacy issues in the work presented here; instead, we've assumed reciprocal trust relations between developers and the proposed system. However, the European Union's ongoing Serenity project includes security and dependability research related to recommendation systems based specifically on IC-Service; see www.serenity-project.org for more details.

Acknowledgments

This work is partly funded by the following projects: QUIEW (Quality-Based Indexing of Web Information, funded by Provincia Autonoma di Trento), Serenity (System Engineering for Security and Dependability, funded by the European Commission), and Mensa (Methodologies for the Engineering of Complex Software Systems: Agent-Based Approach, funded by the Italian government).

References

1. D. Martin et al., "OWL-S: Semantic Markup for Web Services," W3C member submission, Web Ontology Working group, 2004; www.w3.org/Submission/OWL-S.
2. U. Keller, R. Lara, and A. Polleres, "WSMO Web Service Discovery," working draft, WSML Working Group, 2004; www.wsmo.org/2004/d5/d5.1.
3. R. Bova et al., "Task Memories and Task Forums: A Foundation for Sharing Service-Based Personal Processes," *Proc. Int'l Conf. Service Oriented Computing* (Icsoc 07), LNCS 4749, Springer, 2007, pp. 365–376.
4. M. Kerrigan, "Web Service Selection Mechanisms in the Web Service Execution Environment," *Proc. ACM Symp. Applied Computing* (SAC 06), ACM Press, 2006, pp. 1664–1668.
5. U.S. Manikrao and T.V. Prabhakar, "Dynamic Selection of Web Services with Recommendation System," *Proc. Int'l Conf. Next Generation Web Services Practices* (NWESP 05), IEEE CS Press, 2005, pp. 117–121.
6. W. Sherchan, S.W. Loke, and S. Krishnaswamy, "A Fuzzy Model for Reasoning about Reputation in Web Services," *Proc. ACM Symp. Applied Computing* (SAC 06), ACM Press, 2006, pp. 1886–1892.
7. M. Claypool et al., "Implicit Interest Indicators," *Proc. Int'l Conf. Intelligent User Interfaces*, ACM Press, 2001, pp. 33–40.
8. E. Birukou et al., "IC-Service: A Service-Oriented Approach to the Development of Recommendation Systems," *Proc. ACM Symp. Applied Computing* (SAC 07), ACM Press, 2007, pp. 1683–1688.
9. N. Kokash, A. Birukou, and V. D'Andrea, "Web Service Discovery Based on Past User Experience," *Proc. Int'l Conf. Business Information Systems* (BIS 07), LNCS 4439, Springer, 2007, pp. 95–107.
10. E. Blanzieri et al., "Implicit Culture for Multi-agent Interaction Support," *Proc. 9th Int'l Conf. Cooperative Information Systems* (CoopIS 01), LNCS 2172, Springer, 2001, pp. 27–39.
11. N. Seco, T. Veale, and J. Hayes, "An Intrinsic Information Content Metric for Semantic Similarity in WordNet," *Proc. European Conf. Artificial Intelligence* (ECAI 04), IOS Press, 2004, pp. 1089–1090.