

## Context-aware agents for user-oriented web services discovery and execution

M. Brian Blake · Daniel R. Kahan ·  
Michael Fitzgerald Nowlan

Published online: 8 November 2006  
© Springer Science + Business Media, LLC 2007

**Abstract** Service-oriented computing (SOC) suggests that the Internet will be an open repository of many modular capabilities realized as web services. Organizations may be able to leverage this SOC paradigm if their employees are able to ubiquitously incorporate such capabilities and their resulting information into their daily practices. It is impractical to assume that human users will be able to manually search vast distributed repositories at real-time. This paper presents an architecture, Software Agent-Based Groupware using E-services (SAGE), that incorporates the use of intelligent agents to integrate human users with web services. SAGE provides background search and discovery approaches, thus enabling human users to exploit service-based capabilities that were previously too time-consuming to locate and integrate. We present a multi-agent system where each agent learns the rule-based preferences of a human user with regards to their current operational “context” and manages the incorporation of relevant web services.

**Keywords** User context and web service discovery · Intelligent agents · Service-oriented computing · Software agents and context for Web services personalization

---

**Recommended by:** Djamal Benslimane and Zakaria Maamar

M. B. Blake (✉) · D. R. Kahan · M. F. Nowlan  
240 Reiss Science Building, Georgetown University, Washington, DC 20057, USA  
e-mail: mb7@cs.georgetown.edu

*Present Address:*

M. B. Blake  
7515 Colshire Drive (M/S H317), The MITRE Corporation, McLean, VA 22102-7508

D. R. Kahan  
e-mail: drk8@cs.georgetown.edu

M. F. Nowlan  
e-mail: mfn3@cs.georgetown.edu

## 1 Introduction

*Service-oriented Computing (SOC)* [21] is a paradigm where the capabilities of network-accessible services (i.e. web services) can be easily searched and integrated among multiple organizations. SOC enables unrelated organizations to advertise their services while allowing underlying software mechanisms or *software agents* to interpret them. Although emerging standards and technologies [26, 28] are beginning to support these notions, lack of industry consensus [5] for defining services and their messages is a significant problem for automated software that attempts to reason about and match service offerings to appropriate consumers [25]. Although the addition of semantics to the definition of the services (*semantic web services*) is a promising solution to this problem [23], these approaches are in the early maturation phase. Furthermore, by reasoning on semantics that *just* describe the service offerings, these approaches tend to neglect the benefit of using the information that incorporates (1) the *preference* of the consumer and (2) the *context* in which the consumer operates.

Our work leverages humans as the consumers of web service offerings and attempts to incorporate these external software capabilities into their daily activities. More specifically, we use the reasoning abilities of humans as a first step towards web service integration. Agents in the SAGE architecture monitor the actions of their human counterparts. These agents extract text values with the humans routines based on *contextual* information. *Context* is the information that describes the environment in which a user does his/her daily routines (i.e. actions, current personal information, and the users' roles). The contextual information is later used to suggest services that may be relevant to that consumer. When a relevant service is discovered, the user is presented with the capability and/or the result of its execution (i.e. the agent automatically invokes the service by inserting the captured information and presents the resulting information). A feature of this system is the semi-automated user profiling that enables the SAGE personal assistant agent to predict what services and/or information to present to the human user in the future based on trends associated with past preferences. With the agent architecture representing the primary contribution here, there are also several secondary research questions that we explore: (1) *What context-based user actions or behaviors can be captured and used as the search criteria to discover relevant services and* (2) *What is the effective rule-based model that uses context and user preferences to predict service-based capabilities to human user/consumers?*

Throughout this paper, these questions are addressed with respect to investigations into the SAGE architecture, models, and implementations. In the next section, we describe the independent SAGE agent and the future high-level architecture considering the interaction of multiple agents. In the subsequent section, related works are discussed. Following sections discuss the formal models that describe the operation of the SAGE agents (i.e. service profile management and context management). Next is a discussion of the resulting proof-of-concept system. The final sections present an evaluation of the system and the paper concludes with plans for future work.

## 2 SAGE approach and agent architecture

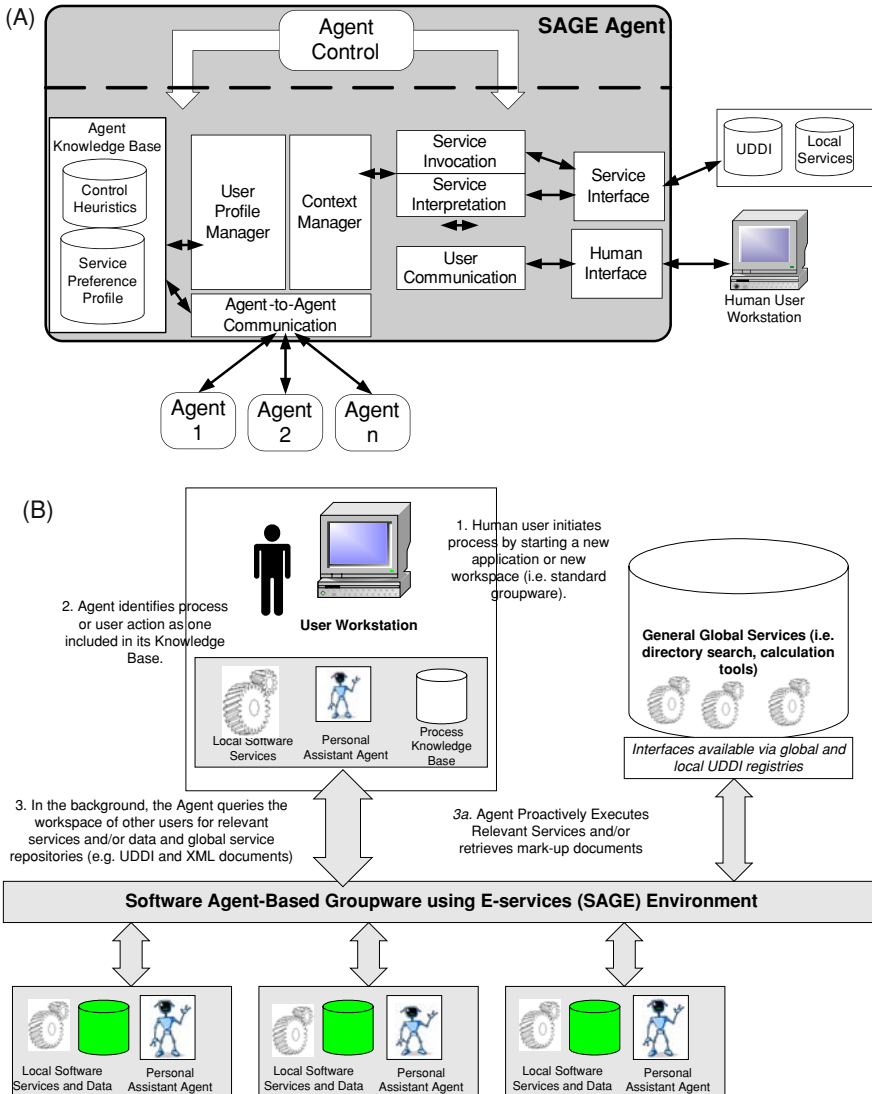
The SAGE approach [1, 2] uses general capabilities to allow human users to incorporate services into their group-oriented processes. Agents can interact with human consumers in their operational context and use that information to determine services appropriate for their tasks. Over time, these agents can ascertain and store usage preferences for the consumer. As semantic approaches mature, the knowledge attained by these agents can be used to develop fully automated approaches beyond the semi-automated approaches described here.

### 2.1 A SAGE scenario

In a SAGE scenario, SAGE agents are personal assistant agents that reside on a human user's workstation or network systems. The human user might be running applications such as Microsoft Word, Excel, or PowerPoint. Alternatively, the user may be browsing the Internet or using other desktop or network applications. Consider a scenario where a user is searching for information on purchasing a server for his/her research group. Each time the user enters a new web page, the agent perceives text about computer dimensions and pricing. The agent then correlates this information and generates a query action against available web services locally or distributed in web service repositories. As a result, the agent finds a computer hardware pricing web service. The agent would then check its profile to see if the user has a preference *not* to use this service. If so, the agent can discontinue. If there is no current preference, then the agent can check with the user for direction and/or appropriate information. Optimally, the information is available and the agent can autonomously execute the service and present the user with the resulting information. In the event the agent is able to act autonomously, context is a vital source of information. The context information (e.g. user's name, zip code, job function) can be used as additional input to the web service. In addition, context information can be used by the agent to determine that a service is irrelevant (e.g. a web service that only looks for personal computers and the user is looking for a UNIX server).

### 2.2 SAGE agent conceptual design and multiagent architecture

A SAGE agent can be defined as a software mechanism that uses the knowledge of its environment to reactively and proactively assist a human user with regards to web service discovery and integration. These aspects are similar to the traditional definitions of agents [9]. As such, SAGE agents communicate to each other about the actions taken by their human counterparts and the resulting services and information that may be useful to those actions. A major function of the SAGE agent is to search for services and information that may be relevant for the users. The independent SAGE agent consists of several components shown in Fig. 1(A). Ultimately the agent is controlled by a *control* component which operates based on a heuristic stored in the agent's *knowledge base*. SAGE agents are distinctive considering the fact that they operate in the SOC arena. The SAGE agents have *service interfaces* to web service repositories and internal mechanisms to interpret web service descriptions and invoke their operations. In addition, these agents contain human interfaces to interpret feedback from human



**Fig. 1** (A) Single agent conceptual design (User Profile Manager and Context Manager are the concentration in this work). (B) SAGE high-level multiagent architecture

consumers. The SAGE agents have general *context management* components and *user preference management* components (*these two components are focus of the paper*) to marshal relevant information within the knowledge base that stores the consumer *service preferences*. Finally, the SAGE agents have an agent communication component that enables interaction with other SAGE agents representing other human consumers.

SAGE agents maintain a knowledge base consisting of keywords and historical actions of the users that they represent. In a large setting, multiple agent agents can

collaborate on relevant information for their users in a process-oriented setting as shown Fig. 1(B).

In the large setting, human users may have shared services and work on collaborative projects. In this setting, SAGE agents can correlate on the preference of users that work in the same context. In this way, SAGE agents can work together to create a higher-level of understanding by aggregating the information of similar users. The mode of operation is the same as for an independent agent, but in future work, we anticipate investigating the ability for agents to collaborate on preferences and context.

### 3 Related work in service-oriented user profiling and context management

Projects related to the SAGE approach can be classified into two categories, agents for user profiling and agents for web service management, specifically those incorporating context. Our notion of *user profiling* (i.e. service-oriented) is the use of agents to monitor human users, interpret important data, and proactively use that knowledge to act on the users' behalf. Although using agents for user profiling is not new, SAGE represents the first significant investigation that incorporates user profiling in a service-oriented computing context. Several projects have used agents to monitor human users and proactively act on their behalf. A common domain for this has been in the areas of web browsing [4, 14, 24] and calendaring [19, 22]. Although SAGE user profiling and learning are consistent with related work, innovation can be found in the customization that enables the usage of web services. For web services, the approach must consider 2 dimensions, the actual software capability *and* the information that the capability provides. This is a significantly different problem that requires a unique user profile model as discussed in later sections. In addition, the SAGE user profile embeds the context of the user as an additional predicate to service delivery.

Using agents for web service discovery and composition is also a well-established area [8]. Some projects use agents to compose web services by semantically linking their underlying information [30]. However these projects attempt to link machines to other machines whereas the approach in this paper is to link humans directly to the web service capabilities. Similarly, this work is related to the area of agents used in collaborative group work settings [6], particularly if services are exposed by other human users, such as services derived from desktop applications (i.e. Microsoft Excel, Visual Basic macros, etc.). It is the combination of the areas of web services, agents and group work that make our work unique.

Some initiatives [12, 29] present architectures to support context, but do not attempt to characterize context with regards to an operational setting. Alternatively, Maamar et al. [15] incorporate the use of context to enhance the degree of personalization in web service composition routines. Their approach considers context based on three aspects (i.e. user-, Web service-, and resource-context) which provides a model that covers discovery and composition. Other work [16] investigates conversation protocols between agents to implement the incorporation of context during service composition. The SAGE approach delves deeper into their notion of user-context by breaking it into three dimensions, *user information*, *user role*, and *user action*. These three dimensions focus the agents to act proactively for a human user. Our work concentrates on the

behavior of the proxy agent when storing context and the technique for discovering pertinent services with context as a search constraint. Therefore, in this work service discovery is emphasized with composition and resource-context (as defined by [15]) relegated to future work.

## 4 Agent-supported service preference profile

A major function of the SAGE agent is to search for services and information that may be relevant for the users. A significant feature in this work is the formalization of the profile management approach

### 4.1 User service preference profile overview

Each SAGE agent captures preferences about when a service should or should not be used on behalf of the human user. This list of rule-based service preferences is defined in the *user profile*. A *user rule* in the user profile can be decomposed into three aspects, the *service-oriented capability*, the *context* by which the capability was requested, and the *preferred action*. The service-oriented capability is a web service as defined with a Web Service Description Language (WSDL) document or information captured as an eXtensible Markup Language (XML) file. Each user rule can be further defined by the operational context of the actions that they perform that trigger the SAGE agents to look for relevant service-oriented capabilities. A non-exhaustive list of contextual descriptions is the role or project of the user, the location, the organization, the priority of the task, etc. Additional discussion of context is included in Section 5. Finally, the preferred action is *to execute*, *to not execute*, or *to defer the decision* of the service-oriented capability. Combining a multi-dimension context vector to these three actions facilitates a fine-grained definition of user preferences for a particular service usage (i.e. any combination of context settings can determine whether a service is executed or not).

### 4.2 Generating the user service preference profile

The user profile can be completely generated by the user (*generated rules*) or alternatively *initial rules* (perhaps from other users) can be used to populate an initial profile, in advance. A structural model of the user profile is captured as a class diagram in Fig. 2. Formally, a user profile,  $\vec{P}_i$ , consists of multiple user rules,  $\vec{U}_i$ , such that  $\vec{P}_i = [\vec{U}_1, \vec{U}_2, \dots, \vec{U}_m]$ . The user rule,  $\vec{U}_m$ , is defined as  $\vec{U}_m = [\vec{S}_m, \vec{C}_m, A_m]$ , where the service-oriented capability,  $\vec{S}_m$ , can be a single capability,  $s_i$ , or a list of potential capabilities such that  $\vec{S}_m = [s_a, \dots, s_c]$ . Similarly the context,  $\vec{C}_m$ , by which the service-oriented capability is used can be defined as  $\vec{C}_m = [c_1, \dots, c_n]$ . The context is described in greater detail in the next section. Finally the preferred action,  $A_m$ , is stipulated by the user or inferred by the agent such that the service-oriented capability is set to either *run* or *do-not-run*. SAGE uses a systematic approach to updating the user profile by generating new rules. The user profile generation in context of SAGE operations is shown in the state diagram in Fig. 3.

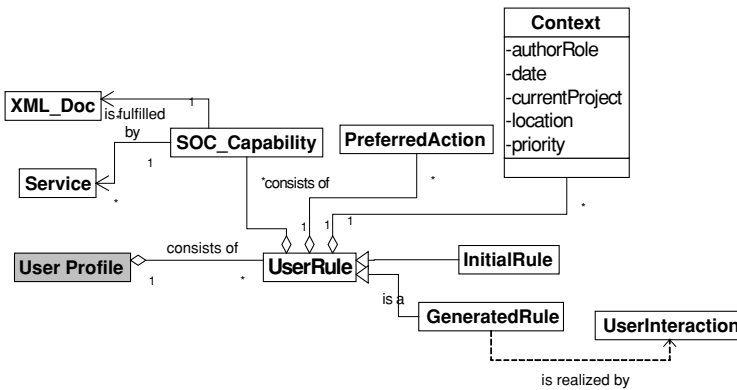


Fig. 2 Structural model of a SAGE user profile for service preferences

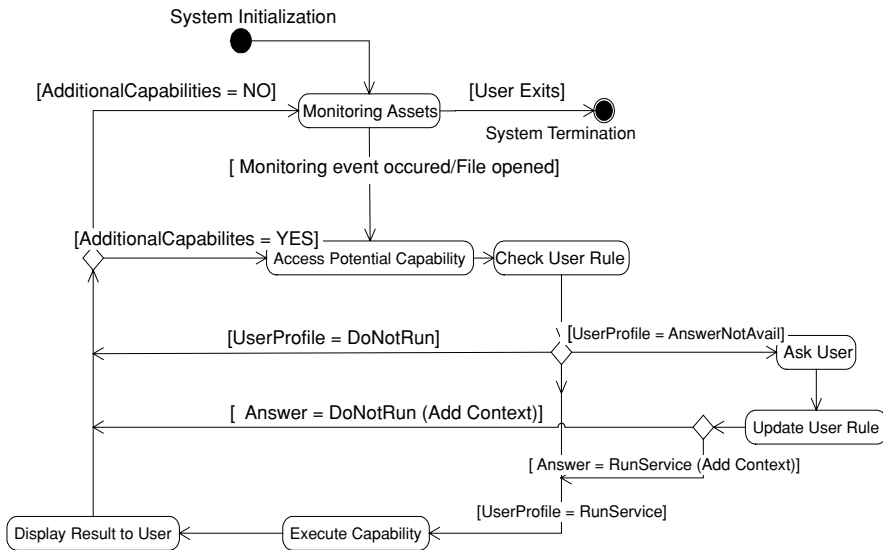


Fig. 3 State diagram of user profile generation

Once the SAGE application is initialized, SAGE agents continually monitor user actions acting as a background processing routine. When a particular action is executed by the user that correlates to an existing service-oriented capability(s), SAGE agents capture information about the potential capability. The agent then accesses its internal user profile to determine if the user has a previously saved preference for this capability. If the user has saved a preference to execute the capability, then the capability is executed and the results are returned to the user in a non-intrusive manner. If the user profile does not have a record of this particular capability or if it is listed but not defined, the user is provided with the opportunity to set the preference. A preference can be set to (1) run at that time and also in the future, (2) not to run at that time and not in the future, or (3) to run only under certain contextual conditions or defer until later.

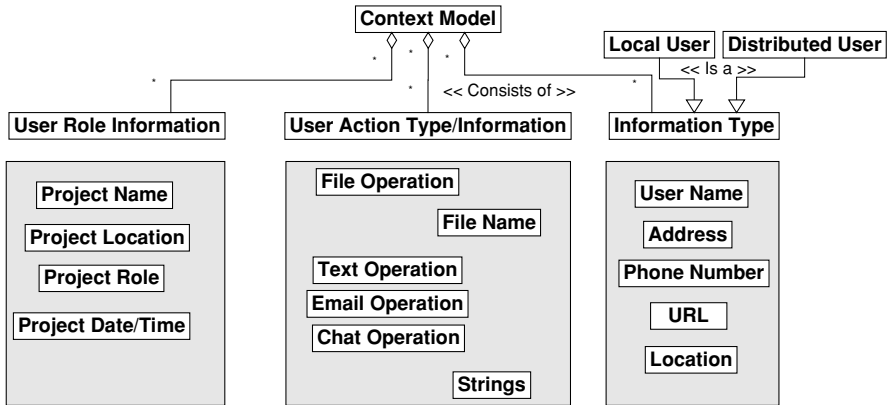


Fig. 4 Context model for the networked user domain

## 5 Agent-supported context management

The context of the user largely determines if a service is accessed or not and ultimately if service-based results are delivered or not. Within this paper, user context is determined based on the information surrounding a computer user (i.e. personal user information and their actions) in the networked environment. For example, if a user types an e-mail message, SAGE agents attempt to correlate the text of the message to a context model and uses the resulting instance information to search for pertinent services. Figure 4 shows the context model used to search for pertinent services. The context model is split into three types of information, *user role information*, *user action information*, and *personal information*, either local or distributed. The user role information describes the nature of the users' project or task. This information may consist of the name, location, role, or date of the project. The user action type/information describes a specific action performed by the user on his/her workstation. A user can manipulate files or communicate with other users via e-mail or chat. In each of these situations, strings can be extracted from the text in the body of the file or message and further from the name of the file itself. The third type of information is the affiliation information of the local user or the distributed user affiliated with the local user. This information consists of the user's name, address, phone number, URL, or current location.

In using the context model, we discovered that the user role and user action information can be used to determine the most relevant web service operation that may be helpful to the user. Alternatively, in many cases, the name of the action or task and user information are auxiliary information that is required as input when executing a web service. The SAGE agents extract information that is relevant to these three categories and looks for similar part names in web services messages to recommend specific capabilities to the user in real time.

### 5.1 Using context and similarity to recommend services

Although there is a large body of work towards the use of semantics for discovering services [17, 20, 23], currently most web services being used in practice are not defined

using related semantic techniques such as the Web Ontology Language, WSDL-S, or the Resource Description Framework (RDF) language. A recent analysis by the authors of web services available on the Internet [11] shows that the majority of services freely available use *only* standard WSDL descriptions. As a result, in this initial approach, SAGE agents use syntactical methods predominantly with WSDL messages to discover services that have a high probability of relevance to the user.

SAGE agents use a similarity measure to find pertinent strings in e-mail, document, and HTML files being manipulated by the user. When a string is similar to one of the words in the context model, the agent extracts that string and looks for potential proper nouns, names, or phrases within that block of text. The extracted strings are saved into a list of search terms. In earlier work, it was discovered that web service developers have a tendency to use similar word choices when naming messages for web service inputs and outputs [11]. SAGE agents leverage this knowledge by looking for input/output messages that are similar to the strings that match the context model. The process of discovering a relevant service is a statistical measure based on how many times a specific context-based string is matched to either the input/output part names of a WSDL file or to the operation name of the service. The focus of this paper is on the context-based infrastructure, however further details about the similarity approach can be found in related work [11]. In later sections, we evaluate how closely related the context model is to open services.

## 5.2 Controlling agent actions with contextual information

In the SAGE environment, an *event* is an action performed by a human user that the SAGE agent detects and uses to garnish a coherent package of information. There are many user actions that we have characterized as events (e.g. a web page opened, a HTTP put invoked, a return pressed in an application, a file saved, an e-mail sent/received, a chat message sent, or an application opened). The authors made the decision to separate the user event from the services with regards to user preference. In a SAGE user profile, the user preferred action (i.e. to execute a capability or not) and context information are associated with the service but the agent does not directly associate the event with the human preference. This notion is visualized in Fig. 5. This separation maintains the extensibility of the system to adopt advanced service discovery techniques as they are devised. However, in future work agents will be extended to use machine learning techniques to predict the preference of the human users and human events will be *required* as an attribute.

The human events that trigger the agent are predicates to determining user preferences. For example, if the human user event is the typing of a string that can be interpreted as a proper name, then there may be relevant web services such as *getAddressforName*, *getBooksAuthored*, or *getBackgroundInfo*. The importance of the user profile is to direct the agent when it is and is not appropriate to use these services.

For example, the human may not want to get the addresses for a specific proper name when that human user is not currently serving in an analyst role. In addition, the human user may never want to get background information when the priority is low.

Formally, an event,  $\vec{E}_L$ , can be defined as:

$$\vec{E}_L = [\vec{G}_L, \vec{C}_L],$$

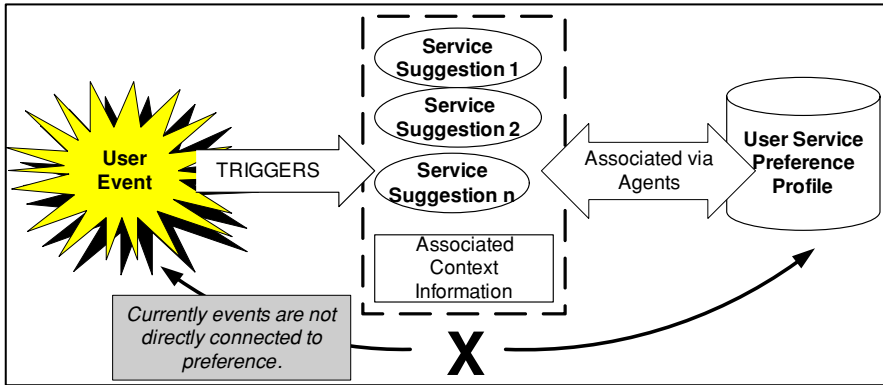


Fig. 5 Correlation of events, services, context, and preferences

where  $\vec{C}_L$  is the context information and the value grouping,  $\vec{G}_i$ , is defined as  $\vec{G}_i = [T_i, O_i, V_i]$ . The value grouping consists of the type of value,  $t_i$ , the orientation,  $o_i$ , and the actual value,  $v_i$ . The type of value can be defined as a proper name, URL, telephone number, address, filename, telephone number, location, and others. The orientation defines further where the value was captured such as accessing/modifying an existing file, writing an e-mail, interacting in a chat session.

Determining the action for a particular event is straight-forward once a service is suggested (The reader should note that the approach to suggesting the service is discussed in the following section). Once the SAGE agents capture the context of the user for the present event,  $C_u$ , and the context of the suggested service-oriented capability,  $C_c$ , then determining the action for that capability,  $A_c$ , can be performed using a matching approach. If the context of the present event is not defined or if the context is the subset of the suggested capability’s context, then the defined action should be used. Formally, if  $[(C_u = \{0\}) \vee (C_u \subseteq C_c)]$ , then the associated,  $A_c$ , represents the action for the proposed service.

The user profile operations were validated through the development of a proof-of-concept module. This module exploits the Jess rule engine [10]. The user preference rules are captured as *facts* in the rule engine. Once a SAGE agent suggests a capability, that capability is asserted with an associated context into agent’s internal Jess module. The assertion triggers the rule that searches the set of all facts to determine if there is a match as defined earlier. A snippet of agent code written in Jess language is shown in Table 1. The SAGE architecture supports flexible contexts that can be defined, in real time. The template for the facts that contain the user preference information is named *capability*. The rule, *run-service*, fires when the user’s preference dictates that the agent-suggested capability should be executed. Table 2 shows several sample facts that the SAGE agent stores in its persistent memory.

### 5.3 Using context and preference for agent recommendations

The SAGE infrastructure supports significant future innovations for proactively managing suggestions for services, rules, and profiles. The process for suggesting a service

**Table 1** Jess code for checking user preferences

---

```

; This template holds the capability, action, and context information
(deftemplate capability
  (slot service (type string))
  (slot project (type string))
  (slot org (type string))
  (slot analystname (type string))
  (slot analystrole (type string))
  (slot priority (type string))
  (slot reldatetime (type string))
  (slot location (type string))
  (slot action (type string)))

; This rule is fired when a service is found and the action "is" to run
; This rule reacts to the assertion defined as
(assert (inputservice <service-name> <project> <org> <analystname> <analystrole> <priority>
  <reldatetime> <location>))

(defrule run-service
  ?inputservice <- (inputservice ?service ?project ?org ?analystname ?analystrole ?priority ?reldatetime
  ?location)
  ?newserv <- (capability (service ?service) (project ?project) (org ?org) (analystname ?analystname)
  (analystrole ?analystrole) (priority ?priority) (reldatetime ?reldatetime) (location ?location) (action
  run))
  =>
  (store RETURN "run")
  (printout t "JESS: Found an existing service = " ?service crlf)
  (printout t "JESS: The user preference is to run the service! " crlf)
  (retract ?inputservice))

```

---

**Table 2** Sample facts for user preferences

---

```

(MAIN::capability (service getaddressforname) (project sage) (org georgetown) (analystname blake)
  (analystrole lead) (priority low) (reldatetime within_an_hour) (location dc) (action run))
(MAIN::capability (service getaddressforname) (project nil) (org nil) (analystname nil) (analystrole
  nil) (priority low) (reldatetime within_an_hour) (location nil) (action do-not-run))

```

---

follows closely to the formalization of the event in the previous section. The information captured in the event can be converted into ASCII strings and defined as the set of all relevant information,  $R_i$ , such that  $R_i = [T_i, O_i, V_i, C_i]$ . At this stage in our development, when a SAGE agent looks for a relevant service-oriented capability, it determines relevance by syntactically matching  $R_i$  to the predicates,  $In_i$ , of the web services (i.e. WSDL part names that serve as input messages to the web service) and of the XML file (i.e. element names in XML schemas available to the SAGE agents). The SAGE agents suggest all services and XML files where  $In_i \subseteq R_i$ . An innovation in this work is the separation between suggesting services and managing the user preferences. In this way, another innovation of the SAGE implementation is the ability to leverage and incorporate future approaches to service discovery.

The SAGE infrastructure is the foundation for several other future innovations, discovering rules and sharing profiles. A future area of research that the SAGE infras-

structure facilitates is the automatic enhancement of user preferences by discovering rules. At the most basic level, SAGE agents can encapsulate heuristics that develop trends statistically based on the number of times and in what context the user requires a capability. In future work, we plan to experiment with user preferences and rule prediction. We believe that another benefit of the SAGE infrastructure, in the future, will be the ability to share partial or full profiles among multiple users.

## 6 Implementation

The SAGE architecture was validated through the development of a proof-of-concept research application, implemented using the Java, Apache Axis, the Jess rule engine, and WSDL4J (for integration with web service descriptions) and the Cougar agent infrastructure.

### 6.1 SAGE low-level software design

The SAGE implementation follows a strict object-oriented design as shown in Fig. 6. *SageAgent* is the main class that contains the control for the agent. Other components are the *UserProfileMgr*, *SageQueryIP*, *EventDaemon*, *WSMgr*, *SageUI*, and *CommunicationMgr*.

The *UserProfileMgr* encapsulates the user profiling functionality described in previous sections. This component contains two instances of the Jess rule engine as encapsulated in the *LocalRE* and *DistRE*. One instance of the rule engine (i.e. *LocalRE*) contains the local preferences of the user while the other rule engine (i.e. *DistRE*) is a proxy component that allows SAGE agents to share preferences among users with similar interests. The *SageQueryIP* component encapsulates the data that is passed within the components of the Sage agent and to other agents using the *CommunicationMgr*. The *EventDaemon* component acts as an interface to the user's desktop such that user events are monitored by the SAGE agents. The *WSMgr* is the software that scans service-oriented capabilities that will be used by the SAGE agents. Finally, the *SageUI* is the graphical user interface component that allows user preferences to be passed from the human to the agent.

### 6.2 SAGE operations and graphical view

Considering the intelligence domain for which the SAGE architecture was customized, the data and services of distributed intelligence analysts and other global services are exposed in open folders held in collaborative groupware applications [such as *Vignette* or *Groove* 7, 27]. By incorporating SAGE into a groupware environment, events can be monitored and captured relatively predictably. A sample interaction among SAGE components is illustrated in Fig. 7.

The *EventDaemon* monitors and captures human events and their context in the background. When an event of interest is captured, the event is passed to the *SageAgent* class and a *SageQueryIP* object is constructed containing all of the relevant information. This *SageQueryIP* object is passed to the *WSMgr* which searches through available services and documents to find a potential capability to assist the user. The

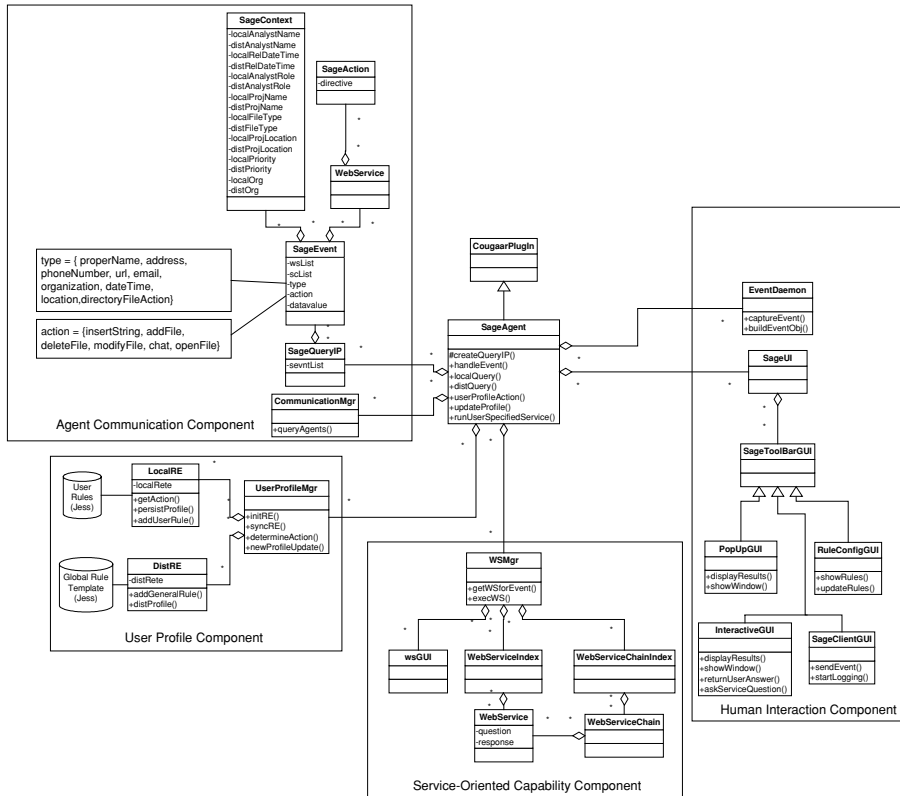


Fig. 6 SAGE object-oriented software design

WSMgr returns all potential capabilities as unique identifiers. The unique identifiers representing the capabilities are passed to the UserProfileMgr to determine if the user has pre-established preferences with regards to the suggested capabilities. In this case, the human user has either set the preference to allow the capability to execute or the agent had included inferred rules to execute the service. The capability is executed and the resultant information is passed to the user as a pop-up bubble on the toolbar. Figure 8 shows a screenshot of the SAGE Interactive View.

In the monitoring state, SAGE operates as a small toolbar icon at the bottom right corner. When a capability, that is not recognized or specified, is suggested the user is presented with a window to enter his/her preferences. The resulting information from the capability is passed back as a pop-up bubble in the bottom right corner.

### 7 Evaluating the prototype performance

Although the implemented system validates the functionality of the SAGE user profile model, a further requirement is for the SAGE agents to be able to make suggestions in a timely fashion. For example, if a user types a proper name such as a fellow

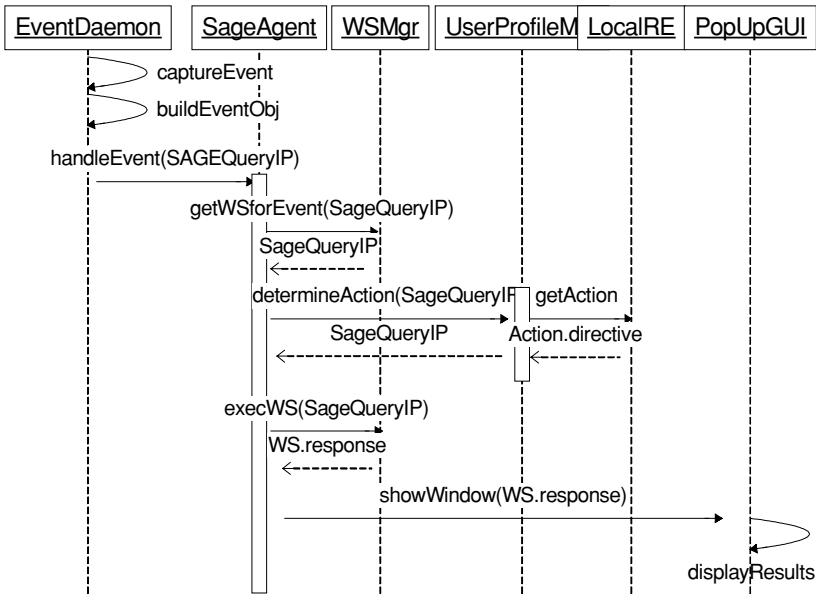


Fig. 7 Sequence diagram of sage interaction

co-worker's name, the SAGE infrastructure should be able to quickly return pertinent capabilities (e.g. co-worker contact information or availability) before the user moves on to the next task. The SAGE implementation is unable to control how fast distributed services can be discovered, however the SAGE implementation is directly responsible for the speed at which the user preference is processed and determined. The SAGE application is designed as several modules and information is passed as objects. This design supports effective message passing across agents, however distribution and modularity can also increase overhead. The purpose of the evaluation was to determine the extent to which these design decisions affect the performance of the user profiling functionality.

A 1.4 GHz, 512 MB, Dell Latitude D600 was used in the experimentation. For the experiment, the number of rules maintained by an independent SAGE agent was varied from 100 to 40,000 facts (i.e. user preference rules). The SAGE agent encapsulates a running instance of the Jess Rule Engine. At agent initialization time, all rules are loaded into memory (40,000 rules is within the default heap space capacity). The results of the experimentation are shown in Table 3 and Fig. 9.

The results show that the UserProfileMgr component that incorporates the Jess module consistently performed memory operations within 10 ms (i.e. operations such as retrieving the user preference (when a rule exists) and updating the profile in memory), regardless of the size of the rule-base. This was a promising result with regards to our user profiling model and implementation. The intent of the second observation was to see how another component of the SAGE Agent performed as the UserProfileMgr component consumed more memory. The EventDaemon is responsible for capturing events and creating information objects of the information. Performance results show that the event capturing time remains fairly constant at approximately 1.5 sec even

as the rule base grows. This is also promising considering the future addition of new service management tools that will be competing for process time and memory.

Other results show that the application is only affected by the size of the profile at initialization time. The initialization time increases proportionally with the introduction of rules at approximately 1 sec per 5000 user preference rules. However, initialization time is only incurred at startup. Another interesting observation is that preference retrieval is much slower and increases with size, when a rule is not present. This may be an area of future work to determine if this can be attributed to the SAGE user profile module. The percentage of change in the completion time remains constant in event capture and preference retrieval, irrespective of the profile size. The completion time for saving updated profiles increases approximately 1500% when the profile increases to 40000 rules. This is still far less by an order a magnitude than the corresponding change in the amount of rules. Although it is unlikely that a single profile will reach 40,000 rules, the profile save function can run concurrently with

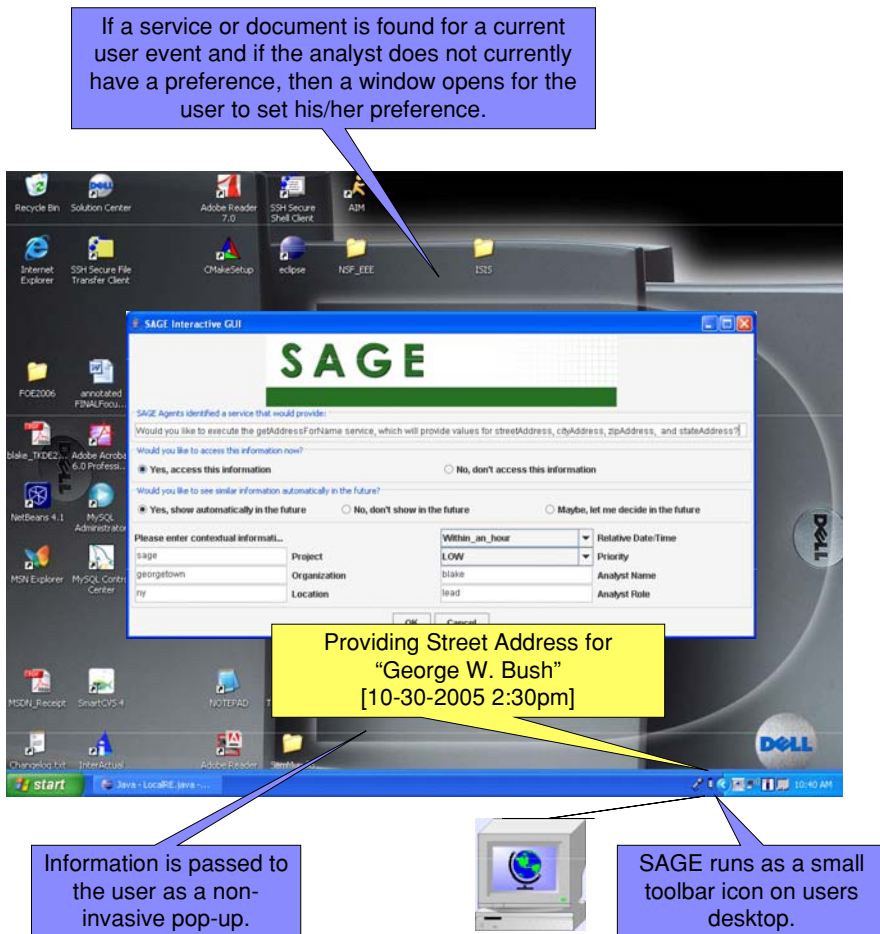
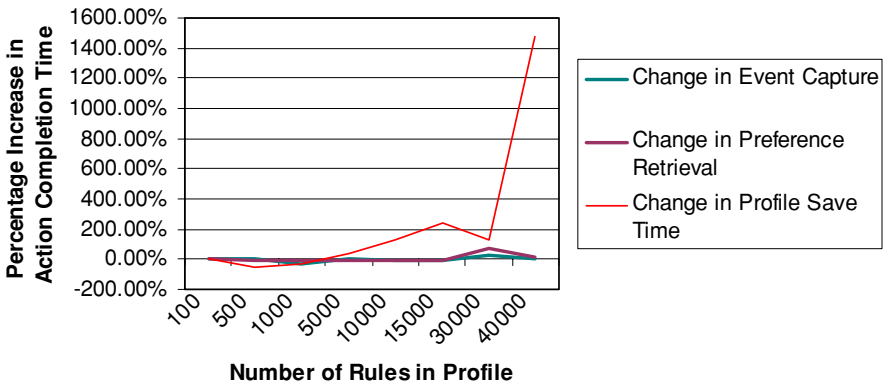


Fig. 8 Screenshot of SAGE graphical interface

**Table 3** Performance evaluation of user profiling tasks

Num of facts	Initialize time (sec)	Event capture time (sec)	Preference retrieval time (ms) (rule exists)	Preference retrieval time (sec) (no rule)	Profile save time (sec)
100	.591	1.533	10	1.632	.220
500	.661	1.562	10	1.493	.110
1000	.821	1.122	10	1.572	.150
5000	1.562	1.542	10	1.522	.321
10000	2.223	1.502	10	1.472	.501
15000	2.984	1.502	10	1.493	.742
30000	5.288	1.953	10	1.893	1.232
40000	8.062	1.542	10	1.963	3.465

**Percentage that Operation Times Increase with Larger Profile Size**



**Fig. 9** User profiling performance decomposition

the other operational functions to mitigate any potential risk associated with slower response times.

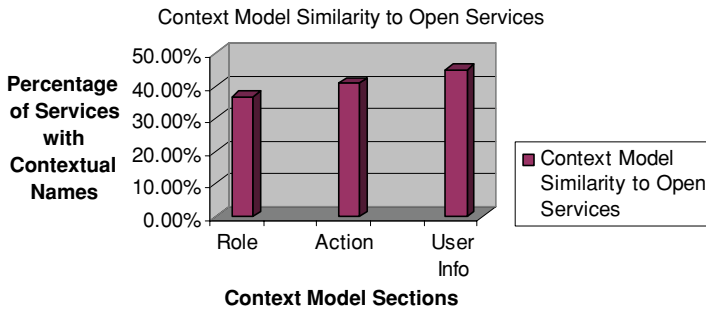
### 8 Evaluating context model effectiveness on real services

SAGE agents recommend services based on the similarity of the input and output messages and the operation name to the context-based information extracted from the user. In order to evaluate the effectiveness of the context model, experimentation was performed to see how closely related the context model is to message names of web services currently available on the Internet. To perform the experimentation, the authors spent 40 hours downloading WSDL files from various service repositories on the Internet (i.e. [31]). The corresponding web services were tested for functionality using the SoapScope tool [18]. Table 4 shows the statistics of the resulting repository.

In early evaluation of the repository, we found that developers have the tendency to use message names that are ambiguous (i.e. “parameters” as the descriptive name for input part names or “return” or “result” for output part names) (Kahan et al. 2005).

**Table 4** Detailed information about the repository

Statistic description	Input	Output	Total
Number of web services			490
Gross number of part names			12,187
Number of part names (unique within each WSDL)	1816	674	2,490
Overall unique part names (23 names overlap inputs and outputs)	798	182	957



**Fig. 10** Percentage of services that have messages related to the context model

Message names in this manner disable the ability to extract syntactic *or* semantic meaning. We found that approximately 50% of the most common part names (i.e. web service messages) in the repository used ambiguous part names. Figure 10 shows the percentage of services by context model component that intersect with strings that are similar to the attributes in the context model. The user role, user action, and user information components of the context model had similarity to the messages of at least one operation of 36.7%, 40.9%, and 45.1% of the web services in the repository, respectively. Considering the maximum possible percentage of services with descriptive message names is 51%, the context model intersects with 95% of the “descriptive” services that we randomly captured from the Internet. Another promising result is that the attributes of the user information aspect of the model have the largest percentage of intersections to message names within the open repository. This result was anticipated, because the user action information and user role information components are less correlated to the message names but more closely correlated to the web service operation name.

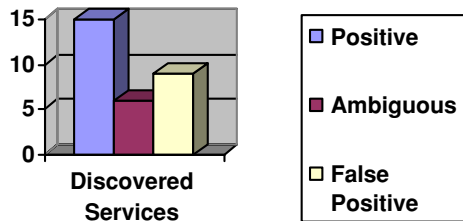
In another experiment, the SAGE agents were evaluated on their ability to find services that would be relevant to a human user in his/her daily routines. To emulate a human user’s daily routine, a random document was extracted from one of the author’s desktop. This document contained an itinerary from Washington, D.C to Orlando, Florida. Related projects have adopted the idea of extracting composition requests from natural language documents [3]. The SAGE agent extracts all the strings related (i.e. similar) to the context model. Figure 11 shows a portion of the discovery request that was generated from the raw itinerary document. When the discovery request was executed on the entire repository using our software, 29 services were returned as results (Table 5 and Fig. 12). While automated characterization of the results is

**Fig. 11** Generated discovery request

```

<DiscoveryRoutine>
  <Provided> Dr, Blake, The, flight, rental,
confirmations, attached, The, information, follows, I,
mailed, couple, parking, passes, campus, The,
schedule, e-mailed, completion, Holiday, Inn,
Select,12125, High, Tech, Ave, Orlando, FL,
32817407-275-9000, 407-381-0019, (Fax),
AssistantSchool, EECS, University, Central,
Florida, 407.823.3925407.823.5419,
(fax),Passenger(s), M ,BRIAN, BLAKE, US,
Airways, record, locator, COYAEETicket,
requested, electronic, (e-ticket), Orbitz, record,
locator, EU8AC8IFAirline, ticket, number(s),
0372301926423,Thursday, January, 26, 2006US,
Airways, #, 1631, Operated, US, AIRWAYS,
EXPRESS-MIDATLANTIC, AIRWAYS, -, Please,
operating, carrier, Washington, Ronald, Reagan,
National, (DCA), Orlando, International,
(MCO)Departure, (DCA), January, 26, 6:20, AM,
EST, (morning)Arrival, (MCO), January, 26, 8:35,
AM, EST, (morning)Class, From, "Avis, Rent, A,
Car", avisreservations@avis.com, Date, Thu, 5,
Jan, 2006, 15:28:18, -0500To, Avis, Real-Time,
Reservation, Confirmation, 45017313US5,
Reservation, Terms, &, Conditions: </Provided>
</DiscoveryRoutine>
    
```

**Fig. 12** Categorization of discovery results on open repository



not possible, a manual analysis of the results shows that at least 15 relevant WSDL files (positive results such as phonebook.wsdl and peoplesearch.wsdl), 6 ambiguous WSDL files (possible false positives such as icd9toicd10.wsdl), and 9 false positives (such as VideoGames.wsdl). The reader should note that the names represent the WSDL files, although the more relevant operations are contained within the file.

The SAGE agents’ ability to suggest 29 candidate services from a repository of 490 can significantly enhance future SOC routines. Although there were a relatively large

**Table 5** WSDL files targeted by SAGE agents

bnprice, CSV2XMLService, DictService.xml, emailservices,  
 EMBLNucleotideSequenceWebService, GBEAWHolidayDates,  
 GBNIRHolidayDates, GBSCTHolidayDates, IADPNNewHotService, icd10, icd9toicd10, imstatus,  
 kayakpaddlelen, personlookup, phonebook,  
 pukiQueryDeathIndex, RssToHTML  
 Server, Stockquote, StockQuotesCurrent, USHolidayDates, VideoGames, WSAmazonBox  
 WSCustNews, WSElectronics, WSSportingGoods, WSVideoGames, xmltracking

number of false positives, these generally were a result of the discovery request that was generated. Many misleading strings were included in the request when gathering proper names that just surround context strings. Leveraging more sophisticated natural language approaches to extracting the request would enhance the accuracy of our approach. Table 5 shows the services discovered and Fig. 12 evaluates the results. These statistics only account for the raw agent discovery features. The non-relevant services are further filtered once the agent checks the user preference database. This represents an innovative multi-dimensional selection technique.

## 9 Conclusion

The approaches presented in this paper introduce the combination of context-based discovery and user preference filtering as an innovative and effective technique for user-based web service delivery. The use of intelligent agents supports a distributed environment where users and the services pertinent to their daily routines can be located at any networked location. We conceptualized a contextual model that agents can use as a guide to determine user context and as a schema for syntactically matching relevant services. Through experimentation and evaluation, we found that the attributes of the context model intersect with a majority of a set of services downloaded randomly from the Internet. Furthermore, using the context model, agents found 29 potentially relevant services out of close to 500 services where 15 services were proven relevant. Combining the agent discovery feature with user preferences increased the precision of the services that agents discovered for their corresponding human users. In future work, we intend to extend our similarity techniques for web service discovery with other natural language approaches and semantic approaches. In this paper, we explore users that perform their daily routine in an Internet-based environment. We plan to extend our context model by exploring other operational domains with different types of users.

**Acknowledgments** The authors recognize Jerome Butcher-Green for his assistance in implementing the rule-based software. In addition, discussions with David H. Fado and Gregory A. Mack of SAIC were helpful in developing the SAGE approach. The service repository and certain parts of the service discovery software used in this work were partially funded by the National Science Foundation under award number 0548514.

Portions of this work were benefited by discussions with Dr. Mark G. Matthews of The MITRE Corporation. The author's affiliation with The MITRE Corporation is provided for identification purposes only, and is not intended to convey or imply MITRE's concurrence with, or support for, the positions, opinions or viewpoints expressed by the author.

## References

1. M.B. Blake, D. Kahan, D.H. Fado, and G.A. Mack, "SAGE: Software agent-based groupware using E-services," in Proceedings of the ACM Conference on Supporting Group Work (GROUP 2005), Sanibel Island, FL, November 2005 (poster).
2. M.B. Blake, D.H. Fado, and G.A. Mack, "Proactive service discovery and execution using intelligent agents," in Proceedings of the IEEE International Conference on Web Services (ICWS 2006), Chicago, IL, September 2006 (to appear).

3. A. Bosca, A. Ferrato, D. Corno, I. Congui, and G. Valetto, "Composing Web services on the basis of natural language requests," in Proceedings of the 3rd IEEE International Conference on Web Services (ICWS 2005), Orlando, FL, June 2005, pp. 817–818.
4. L. Chen and K. Sycara, "Webmate: A personal agent for browsing and searching," in Proceedings of 2nd International Conference on Autonomous Agents, New York, NY, USA, ACM Press, 1998, pp. 132–139.
5. G. Clark, "Politics hurting web services," Gartner, May 2005.
6. C. Ellis, J. Wainer, and P. Barthelmess, "Agent-augmented meetings," In Agent supported cooperative work, Yiming Ye, Elizabeth Churchill ed. Kluwer Academic Publishers, 2003.
7. Groove, 2006, <http://www.groove.net/home/index.cfm>.
8. M.N. Huhns, "Agents as web services," Internet Computing, vol. 6, no. 4, pp. 93–95, 2002.
9. N.R. Jenni00ngs, K.P. Sycara, and M. Wooldridge, "A roadmap of agent research and development," Journal of Autonomous Agents and Multi-Agent Systems, vol. 1, no. 1, pp. 7–36, 1998.
10. Jess Rule Engine, 2006, <http://herzberg.ca.sandia.gov/jess/>.
11. D.R. Kahan, M.F. Nowlan, and M.B. Blake, "Taming web services in the wild," in Proceedings of the 2006 International Conference on Web Services (ICWS 2006) (to appear).
12. M. Kiedl and A. Kemper, "Towards context-aware adaptable web services," in Proceedings of the WWW2004, New Your, NY, May 2004, pp 55–65.
13. R. Kozierok and P. Maes, "A learning interface agent for scheduling meetings," in Proceedings of the 1993 International Workshop of Intelligent User Interfaces, 1993, pp 81–88.
14. H. Lieberman, "Letizia: An agent that assists web browsing," in Proceedings of the 14th International Joint Conference on Artificial Intelligence (IJCAI-95), Montreal, Quebec, Canada, 1995, pp 924–929.
15. Z. Maamar, G. AlKhatib, S.K. Mostefaoui, M.B. Lahkim, and W. Monsoor, "Context-based personalization of web services composition and provisioning," In Proceedings of the 30th Euromicro Conference (EuroMico'04), 2004, pp. 396–403.
16. Z. Maamar, S.K. Mostefaoui, and H. Yahyaoui, "A web services composition approach based on software agents and context," in Proceedings of the 19th Annual ACM Symposium on Applied Computing (SAC'2004), Nicosia, Cyprus, 2004, pp. 1619 – 1623.
17. S. McIlraith, T. Son, and H. Zeng, "Semantic web services," IEEE Intelligent Systems, vol. 16, no. 2, pp. 43–56, March/April 2001.
18. Mindreef SoapScope, 2006, <http://www.mindreef.com/products/soapscope/index.php>.
19. T.M. Mitchell, R. Caruana, D. Freitag, J.P. McDermott, and D. Zabowski, "Experience with a learning personal assistant," Communications of the ACM, 1994, pp. 80–91.
20. OWL-S, 2006, <http://www.daml.org/owl-s/>.
21. M. Papazoglou, "Service-oriented computing: Concepts, characteristics and directions," in Proceedings of the 4th International Workshop on Web Information Systems Engineering, 2003, pp. 3–12.
22. S. Sen, T. Haynes, and N. Arora, "Satisfying user preferences while negotiating meetings," International Journal of Human-Computer Studies, vol. 47, pp. 407–427, 1997.
23. E. Sirin, J. Hendler, and B. Parsia, "Semi-automatic composition of Web services using semantic descriptions," in Proceedings of Web Services: Modeling, Architecture and Infrastructure workshop in conjunction with ICEIS2003, 2002. (<http://www.mindswap.org/papers/composition.pdf>).
24. G.L. Somlo and A.E. Howe, "Using web helper agent profiles in query generation," in Proceedings of the 2nd Int. Joint Conference on Autonomous Agents and Multiagent Systems, New York, ACM Press, 2003, pp. 812–818.
25. B. Srivastava and J. Koehler, "Web service composition – current solutions and open problems," ICAPS Workshop on Planning for Web Services, 2003.
26. W.M.P. Van der Aalst, "Don't go with the flow: Web Services composition standards exposed," IEEE Intelligent Systems, February 2003.
27. Vignette, 2006, <http://www.vignette.com/>.
28. Web Services, 2006, <http://www.w3.org/2002/ws/desc/>.
29. Web Service Context (WS-Context), 2006, <http://developers.sun.com/techtopics/webservices/wscsf/wscstx.pdf/>.
30. A.B. Williams, A. Padmanabhan, and M.B. Blake, "Experimentation with local consensus ontologies with implications to automated service composition," IEEE Transactions on Knowledge and Data Engineering, vol. 17, no. 7, pp. 1–13, July 2005.
31. XMethods, 2006, <http://www.xmethods.com/>.