

# Proactive Service Discovery and Execution Using Agents

M. Brian Blake  
Georgetown University  
Washington, DC  
[blakeb@cs.georgetown.edu](mailto:blakeb@cs.georgetown.edu)

David H. Fado and Gregory A. Mack  
Science Applications International Corp.  
Arlington, VA  
{David.H.Fado, Gregory.A.Mack}@saic.com

## 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 search vast distributed repositories at real-time. In addition, automated search tools may invasively present too much information. 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 and manages the incorporation of web services.*

## 1. Introduction

*Service-oriented Computing (SOC)* is a paradigm where the capabilities of network-accessible services (i.e. web services) can be easily searched and integrated among multiple organizations. Furthermore by adding semantics to the definition of the services (*semantic web services*), automated software can intelligently discover services and incorporate them into an existing systems context. This idea of service-oriented machine-to-machine integration is the premise for many related projects. However, perhaps the most difficult problem in this domain is the development and standardization of representations and general data integration approaches. Such approaches would enable unrelated organizations to advertise their services while allowing the underlying software mechanisms or software agents to interpret them. Although emerging standards and technologies [15][18] are beginning to support these notions, this is still a difficult problem [14]. Furthermore, organizational differences and politics tend to amplify the problem [3]. Understanding that data integration is still an open problem, the SAGE approach supports an agent-based human-in-the-loop approach to web service integration.

The motivation for this approach stems from two observations:

1. *Human users perhaps remain the best semantic matchers with regards to determining if a software capability is applicable.*
2. *Web services, if presented in a timely enough fashion, can support human users directly.*

Thus, our work leverages humans as a first step towards web service integration by incorporating external software capabilities into their daily activities. Agents in the SAGE architecture monitor the actions of their human counterparts. The agents extract text values and contextual information and use that information to search open repositories of web services in the background. When a relevant service is discovered, the user is presented with the capability (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 our agent architecture representing the primary contribution here, there are also several secondary research questions that we explore: (1) *What user actions or behaviors can be captured and used as the search criteria to discover relevant services* and (2) *What are the aspects of the user preference model that links users to web services to enable usage prediction?*

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 high-level SAGE architecture. In the subsequent section, related work is detailed. Following sections discuss the formal models that describe the operation of the SAGE agents (i.e. user profiling, service discovery, and user-agent interaction). The next sections discuss the requirements for the SAGE architecture in the intelligence domain and the resulting proof-of-concept system. The final sections present an evaluation of the system and conclude with plans for future work.

## 2. Sage Approach and Architecture

The SAGE approach promotes the development of general capabilities to allow human users to incorporate services into their group-oriented processes. In the context of the architecture, a human user performs his/her daily activities. SAGE agents maintain a knowledge base consisting of keywords and historical actions of the analyst. Either through the request of the analyst or proactively, the agent queries open service repositories to find information or relevant services either from other agents or from open service repositories. This architecture is illustrated in Figure 1.

## 3. Related Work

Projects related to the SAGE approach can be classified into two categories, agents for user profiling and agents for web service management. Our notion of *user profiling* 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, as known by the authors, 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 [2][10][13] and calendaring [11][12][9]. 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.

Using agents for web service discovery and composition is also a well-established area [6][14]. Some projects use agents to compose web services by semantically linking their underlying information [19]. 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 also related to the area of agents used in collaborative group work settings [4][17], 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.

## 4. Sage Agents

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 [7]. As such, SAGE agents communicate about the actions taken by their human counterparts and derive services and/or information that may be useful to those actions.

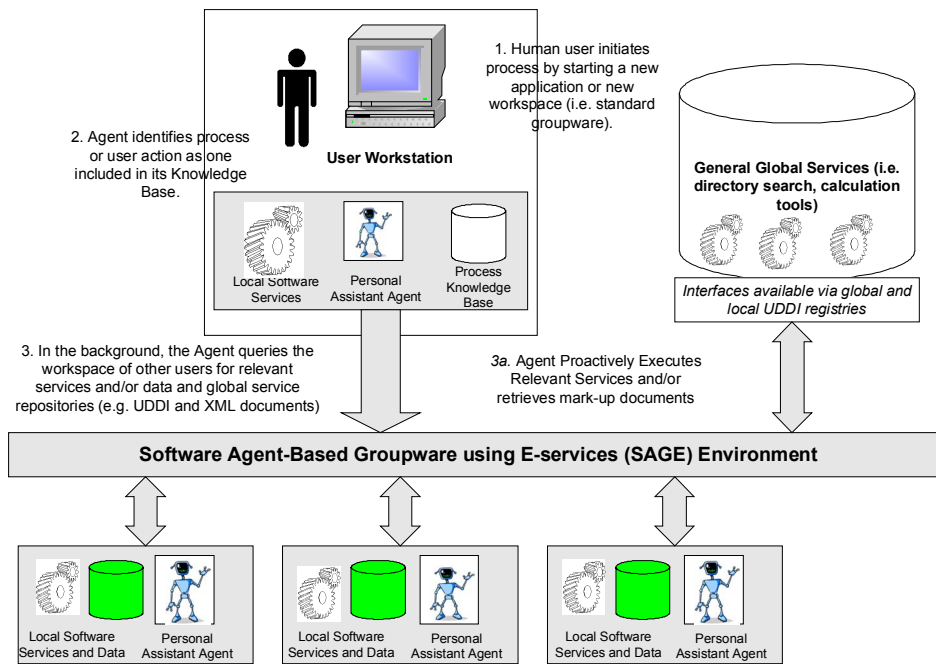


Figure 1. SAGE High-level System Architecture.

A major function of the SAGE agent is to search for services and information that may be relevant for the users. At this point in the project, we have concentrated on infrastructure development and human-agent modeling. Service discovery has been relegated to syntactical matching as this is yet an open problem in the research community. In future work, we intend to continually improve the architecture and infrastructure with semantics techniques, refer to previous work of the authors [19].

Although the architecture has many aspects, within the scope of this paper we concentrate on addressing the research questions in Section 1. In the following section, the user profile model is formally defined in addition to the human-agent interaction used to populate the profile. In the subsequent section, the service discovery technique is discussed with regards to the user actions and profiles. Finally, we introduce an approach whereas profiles can be used to predict if a service is relevant or not based on the past preferences of the user. Furthermore, we believe that future profiles will be *cast* upon users with similar affiliations to reduce the time it takes to program their personal SAGE agents.

#### 4.1. User profile model

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 preferences is defined as 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 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. Finally, the preferred action is *to execute*, *to not execute*, or *to defer the decision* the service-oriented capability. Considering the flexibility of dynamically defining a multi-dimension context vector, this approach allows a fine-grained definition of user preferences for service usage.

**4.1.1. Generating the profile.** The user profile can be completely generated by the user (*generated rules*) or *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 Figure 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  $\vec{C}_m = [c_1, \dots, c_n]$ . 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*.

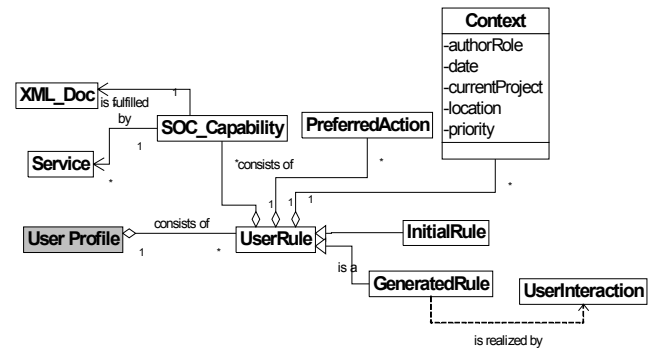


Figure 2. Structural Model of a SAGE User Profile.

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 Figure 3.

Once the SAGE application is initialized, SAGE agents continually monitor user actions 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 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 returned to the user in a non-intrusive manner. If the user profile does not have 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.

#### 4.1.2. Checking user rules.

The reader should note that, 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 and not directly to the *event* from which the service is derived. Example events may be typing a word of interest or accessing a specific file or application of interest. The authors made the decision to separate the user event from the services with regards to user preference. This separation maintains the extensibility of the system to adopt advanced service discovery techniques as they are devised. However, for prediction, human events are *required* as an attribute. In this paper, prediction is introduced but in-depth experimentation is an area of future work.

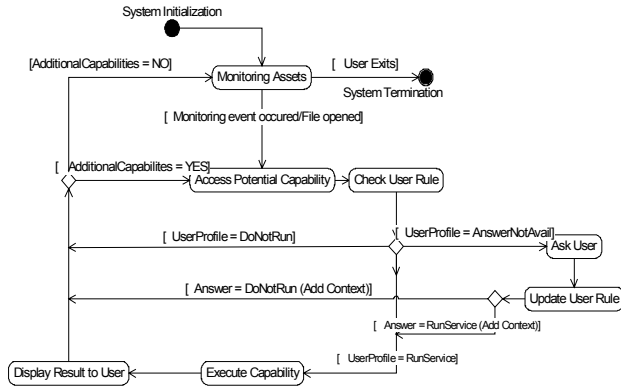


Figure 3. State Diagram of User Profile Generation.

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:

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

where  $\vec{C}_L$  is the context information defined in Section 4.1.2 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.

Although the overall implementation is discussed in a later section, the implementation of the rule engine is relevant to this section. The user profile operations were validated using a proof of concept implementation of the Jess rule engine [8]. 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 the Jess language is shown in Table 1. The reader should note the context information is named explicitly in this code snippet based on the intelligence domain discussed in later sections. 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.

#### 4.2. User profile model

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

**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))
```

## 5. Case Study

### 5.1. Collaborative Intelligence Environment

Under the sponsorship of the Air Force Research Lab (AFRL), new infrastructures, that support information sharing and advanced analytical collaboration, are being created to support intelligence analysts. Intelligence analysts typically issue or receive urgent requests for information. These requests spur the formation of a team of perhaps a dozen relevant experts who must construct accurate responses in a short period of time. Cognitive support tools employed will vary according to the shifting group membership. Although several forms of groupware software are used, the team formation and processes are relatively ad-hoc. Of benefit to this domain is the recent inception of SOC techniques. The SAGE architecture addresses the coordination that must occur to integrate on-demand group collaboration with the use of external cognitive support tools using SOC. Table 3 describes the steps that an analyst leading a study would take considering the existence of SOC tools.

**Table 3.** Initialization Routine.

1. Lead Analyst (LA) is notified of information request
2. LA queries and locates other specialists across organizations to contribute in the group setting.
3. LA establishes a collaborative workspace.
4. LA queries and locates services and informational streams that may be relevant to the study
5. LA establishes connectivity to the candidate services and information within the workspace

Table 4 shows the day-to-day requirements of a groupware system that incorporates services. Table 3 and Table 4 represent the core set of requirements that were used to determine an initial customization of the SAGE architecture.

**Table 4.** Concept of Operation.

<p><i>On-Demand Operational Requirements (Reactive)</i></p> <ul style="list-style-type: none"> <li>• Delivery of new information to the group</li> <li>• Analyst-initiated search for services or information</li> <li>• Standing request for information</li> </ul> <p><i>Proactive Requirements</i></p> <ul style="list-style-type: none"> <li>• Ubiquitous identification of human processes and correlation to relevant services or information</li> <li>• Execution of potentially relevant service and proactive delivery of the resultant information</li> </ul>
---

### 5.2. 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 Cougaar agent infrastructure. The SAGE implementation follows a strict object-oriented design as shown in Figure 4. *SageAgent* is the main class that contains the control for the agent. It would

not be practical to discuss each class in detail, but of importance are the aggregates of the SageAgent class because these classes represent the main components of a general SAGE agent. These components are the *UserProfileMgr*, *SageQueryIP*, *EventDaemon*, *WSMgr*, *SageUI*, and *CommunicationMgr*. The *UserProfileMgr* encapsulates the user profiling functionality describe 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. In this initial implementation, the *SageUI* contains text fields customized for the intelligence domain.

## 5.2. 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 [16][5]). By incorporating SAGE into a groupware environment, events can be monitored and captured relatively predictably.

A sample interaction among SAGE components is illustrated in Figure 5. 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 *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 had

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 6 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.

## 6. Evaluation

Although the implemented the 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 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. The results of the experimentation are shown in Table 5 and Figure 7. The reader should note that the times in Figure 7 are not cumulative since the operations occur separately, but the data are presented in the bar chart for comparison purposes. The first observation was that the SAGE Agent ran out of heap space for any number of facts more than ~41,000 user preference rules (for standard PC operational settings).

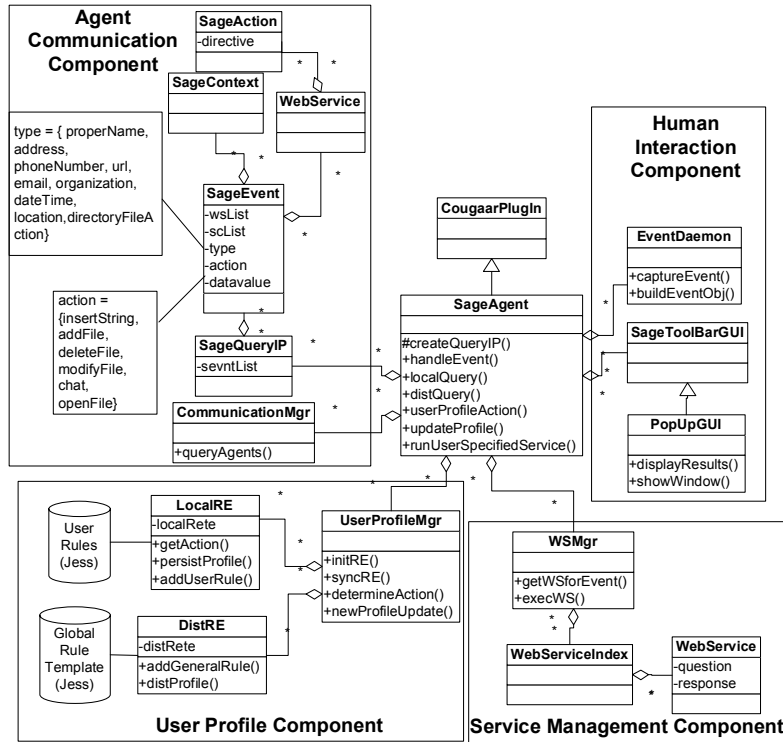


Figure 4. SAGE Object-Oriented Software Design.

The authors believe 40,000 facts seem like a reasonable ceiling for use by an individual user profile. We also determined that the UserProfileMgr component that incorporates the Jess module consistently performed memory operations within 10ms (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.

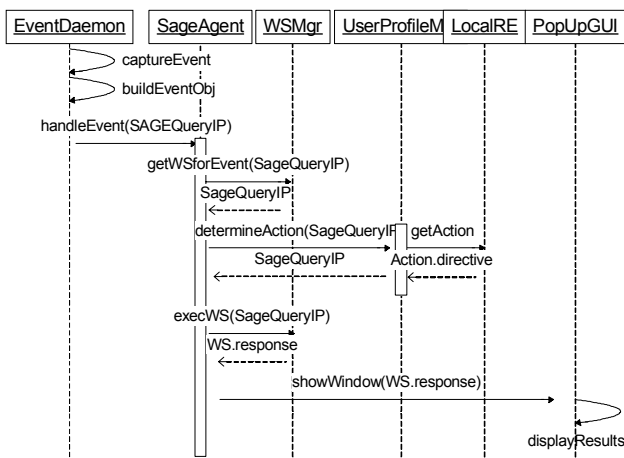


Figure 5. Sequence Diagram of Sage Interaction

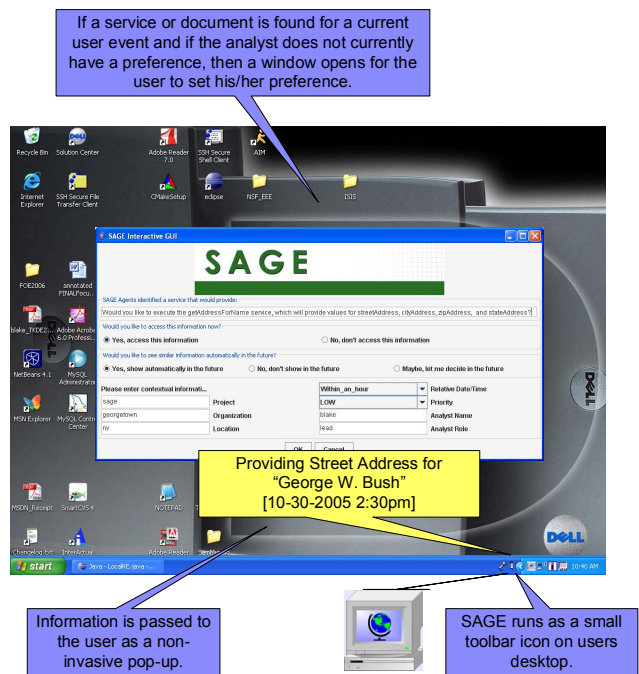


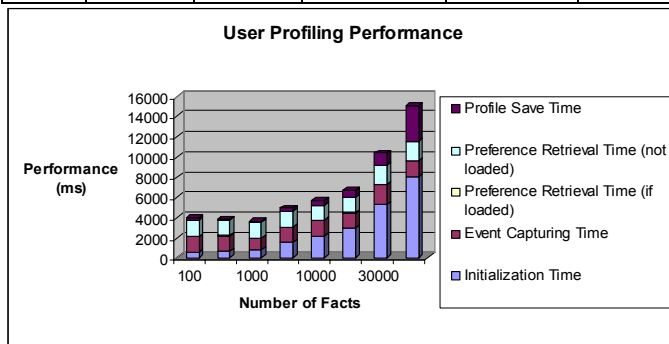
Figure 6. Screenshot of SAGE Graphical Interface.

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 seconds 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.

**Table 5.** 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



**Figure 7.** User Profiling Performance Decomposition.

Other results show that the application is affected by the size of the profile. The initialization time increases proportionally with the introduction of rules at approximately 1 second per 5000 user preference rules. However, initialization time is only incurred at startup. As one would suspect, saving the profile to the file system also increases proportionally to the number of user preference rules. 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.

## 7. Acknowledgements

Several undergraduate research assistants helped to develop the SAGE implementation, Daniel Kahan, Daniel Colligan, and Jerome Butcher-Green. This work is partially funded under the AFRL's Topsail program in cooperation with SAIC. The service discovery functions of SAGE were partially funded by the National Science Foundation under award number 0548514.

## 8. References

- [1] Blake, M.B., Kahan, D., Fado, D.H., and Mack, G.A. "SAGE: Software Agent-Based Groupware Using E-Services", Proc. of the ACM Group Conference (GROUP 2005), Sanibel Island, FL, November 2005 (poster)
- [2] Chen, L. and Sycara, K. "Webmate: a personal agent for browsing and searching". In *Proceedings of 2<sup>nd</sup> International Conference on Autonomous Agents*, pp 132-139, New York, NY, USA, 1998. ACM Press.
- [3] Clark, G. "Politics Hurting Web Services", Gartner, May 2005
- [4] Ellis, C., Wainer, J., Barthelme, P. "Agent-Augmented Meetings" In: *Agent supported cooperative work*, Yiming Ye, Elizabeth Churchill ed. Kluwer Publishers, 2003
- [5] Groove(2006): <http://www.groove.net/home/index.cfm>
- [6] Huhns, M.N. "Agents as Web Services," *Internet Computing*, 6(4), pp. 93-95, 2002.
- [7] Jennings N.R., Sycara K.P., Wooldridge M., "A Roadmap of Agent Research and Development" *Journal of Autonomous Agents and Multi-Agent Systems*. 1(1): 7-36, 1998
- [8] Jess Rule Engine (2006): <http://herzberg.ca.sandia.gov/jess/>
- [9] Kozierok, R. and Maes, P. "A Learning Interface Agent for Scheduling Meetings" In *Proceedings of the 1993 International Workshop of Intelligent User Interfaces*, pp 81-88
- [10] Lieberman, H. "Letizia: An agent that assists web browsing" *Proceedings of the 14th Int. Joint Conference on Artificial Intelligence (IJCAI-95)*, pp 924-929, Montreal, Quebec, Canada, 1995. Morgan Kaufmann
- [11] Mitchell, T.M., Caruana, R., Freitag, D., McDermott, J.P. and Zabowski, D. "Experience with a Learning Personal Assistant" *CACM*, pages 80-91, 1994.
- [12] Sen, S., Haynes, T., and, Arora, N. "Satisfying user preferences while negotiating meetings". *International Journal of Human-Computer Studies*, 47:407-427, 1997.
- [13] Somlo, G.L. and Howe, A.E. "Using web helper agent profiles in query generation". In *Proceedings of the 2nd Int Joint Conference on Autonomous Agents and Multiagent Systems*, pp 812-818, New York, 2003. ACM Press.
- [14] Srivastava, B. and Koehler, J. "Web Service Composition - Current Solutions and Open Problems". *ICAPS Workshop on Planning for Web Services*, 2003.
- [15] Van der Aalst, W.M.P. "Don't go with the flow: Web Services composition standards exposed", *IEEE Intelligent Systems*, February 2003
- [16] Vignette (2006): <http://www.vignette.com/>
- [17] Wainer, J., Braga, D.P. "Symgroup: applying social agents in a group interaction system." In *Proceedings of the ACM 2001 International Conference on Supporting Group Work (GROUP 2001)*, ACM Press, pp. 224-231, 2001
- [18] Web Services (2006): <http://www.w3.org/2002/ws/desc/>
- [19] Williams, A.B., Padmanabhan, A., and Blake, M.B. "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