Digital Libraries ’97
Second ACM International Conference on
Digital Libraries
Philadelphia, Penn., July 1997

Please send correspondence to:

J. Alfredo Sánchez
Department of Computer Systems
Engineering
Universidad de las Américas–Puebla
AP 100
Cholula, Puebla
72820 México

Phone: +52-22-29-2657
Fax: +52-22-29-22-58
email: alfredo@cca.pue.udlap.mx

AGS: Introducing Agents as Services Provided by Digital Libraries

J. Alfredo Sánchez, John J. Leggett, John L. Schnase

DL 97 Philadelphia PA, USA

AGS: Introducing Agents as Services Provided by Digital Libraries

J. Alfredo Sánchez
Department of Computer Systems Engineering
Universidad de las Américas-Puebla
Cholula, Puebla, 72820 México
alfredo@udlapvms.pue.udlap.mx

John J. Leggett
Center for the Study of Digital Libraries
Texas A&M University
College Station, TX 77843 USA
leggett@csdl.tamu.edu

Abstract

This paper presents an architecture for digital libraries that introduces *user agents* as one of the services available to publishers, librarians and patrons. User agents are the fundamental component of an emerging style of human-computer interaction based on the concept of delegation and indirect management of tasks. In the agent-enabled digital library architecture, termed "AGS", service providers define classes of agents that describe helpful tasks for patrons. Patrons, in turn, delegate work by selecting agents from the available agent classes and assigning specific tasks to be performed. AGS enables the development of agents that rely on a wide variety of construction approaches while maintaining a unified view of an active environment. AGS is intended to serve as a testbed to investigate alternative user interfaces to digital libraries and, in particular, a host of unexplored issues raised by the introduction of user agents.

Keywords

User agents, interface agents, digital library interfaces, digital library architectures, open architectures, AGS, TAGS.

1. Introduction

It is clear that *agents* provide a useful abstraction to help system designers understand the functionality of complex, distributed systems such as digital libraries. In the eighties, for example, Kahn and Cerf [1] wrote about their vision of an electronic library that would consist entirely of autonomous entities or "knowbots." More recently, the University of Michigan Digital Library has been described almost completely in terms of agents [2, 3]. Just as designers benefit from using this abstraction, end users can take advantage of agents to cope with the volume, dynamism and complexity of the information universe and the tasks enabled by digital libraries. In this paper we are interested in the potential of agents to assist patrons in taking advantage of the vast resources available in digital libraries.

Following the taxonomy proposed by Sánchez [4, 32], we use the term *user agents* to refer to semi-autonomous entities that are directly perceived (and can be delegated tasks) by end users. User agents are the fundamental component of an emerging style of human-computer interaction [5, 6] based on the concept of delegation and indirect management of tasks. This promising paradigm for user interaction is expected to enhance and supplement current user environments based upon navigational and direct manipulation approaches. We see user agents as just one of the technologies that will contribute to realization of digital libraries.

In this paper, we present an architecture for a digital library that introduces user agents as one of the services available to publishers, librarians and patrons. In this agent-enabled digital library architecture, termed "AGS", service providers define classes of agents that generally describe helpful tasks for patrons. Patrons, in turn, delegate work by selecting agents from the available agent classes and assigning specific tasks to be performed. Agents at the user's service can transform an otherwise unwieldy and ever-changing environment into a cooperating system that will develop expertise in the user's needs and will adapt to those needs to provide personalized services.

The paper is organized as follows. Section 2 describes the context and major components of the proposed architecture. Section 3 presents a prototypical implementation of AGS. An evaluation of the architecture in terms of its relationships with existing agent implementations and other architectures is presented in Section 4. Finally, Section 5 presents an overview of current work in developing AGS more fully and the direction of future work.

2. The AGS Architecture

The design of the AGS architecture has been driven by three major objectives. First, AGS seeks to make agents available *as a service* provided to users of a distributed digital library system, along with other library services. Second, AGS aims at facilitating the design and implementation of user agents, as well as their incorporation into the complex information environment represented by digital libraries. Finally, AGS is intended to serve as a testbed to investigate alternative user interfaces to digital libraries and, in particular, a host of unexplored issues raised by the introduction of user agents. (For a detailed discussion of user agency issues, see [7-10]).

2.1 Base Architecture

Although various architectures have been suggested for digital libraries, a number of common characteristics are apparent. Digital libraries will be developed in highly distributed environments [2; 11-15]. Library objects will be *served* to (often remote) *client* processes which will send information requests and receive results employing varied *communication protocols*. Client processes will make diverse *interfaces* available to users. Repositories on the server side will rely on advanced database management systems (DBMSs) for object storage, indexing and selective retrieval. Layers of abstraction are possible between the basic storage substrate and the interfaces available to external entities, and abstraction mappers will translate between abstractions handled by the storage substrate and those offered by the various digital library services [15].

2.2 Design Abstractions

AGS makes the notion of agents available to users by introducing five abstractions into the base library architecture just described: *agent classes*, *agent instances*, *agent actions*, *user profiles* and *agencies*. Whereas an *agent class* defines a general behavior for user agents, an *agent instance* is associated with an agent class every time a user delegates a specific task to ("instantiates") a given agent. *Actions* are objects in the system which implement functionality used in the construction of agents. Actions can be reused as building blocks for constructing agents with diverse functionality. A *user profile* is associated with each AGS user and contains arbitrary attributes that describe the user's interests and preferences. *Agencies* refer to other AGS-enabled digital libraries known to a given AGS server.

Each of these abstractions can be described in terms of its associated attributes. For example, an agent class is defined by a class name, references to actions implementing the agent functionality (*do* actions), actions defining specific user-agent interaction (*summons* actions) and, optionally, actions that counteract agent operations (*undo* actions). Lists of required and optional parameters for execution of each of these actions are also part of an agent class definition. One attribute indicates where an agent's actions can be executed, in the server's or in the client's environment. Additional attributes indicate the *language* in which the actions are defined. Actions may be binary executables or scripts in languages such as Java, Telescript, or Tcl. AGS classes also incorporate optional attributes which are designed to support the definition of agents that react to events in a digital library. Specifically, it is possible to specify actions an agent will perform upon access, change, addition or deletion of library objects. The remaining four abstractions can be described similarly.

2.3 Architectural Components

Figure 1 presents a simplified view of the general architecture for a distributed digital library as described in Section 2.1, including typical components at a server node, a client node from which patrons have access to library services, and a client node from which librarians have access to functions available for resource administration (this client can also act as a patron client). The components introduced to handle AGS's abstractions are shaded.

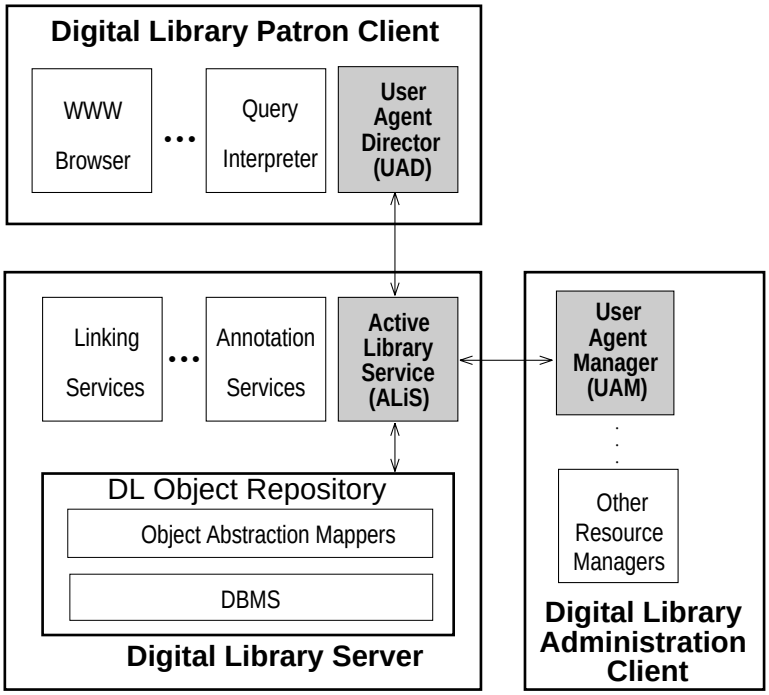

Figure 1. The Agent Services Architecture.

The heart of the AGS architecture is a new service module, the *Active Library Service* (ALiS), which augments the server side of a digital library architecture. ALiS provides functionality to manage the five AGS abstractions by utilizing and extending existing digital library infrastructure. Two components interact directly with ALiS: the *User Agent Manager* (UAM), which provides functionality for the administration and tuning of AGS resources; and the *User Agent Director* (UAD), through which end users define and control agents in the digital library. In Figure 1, only one library server is displayed along with a patron client and an administration client. In AGS, though, multiple users can access the library from a single client, each of them having the capability to use AGS services by starting a UAD process. A single UAD may also contact multiple library servers and their corresponding Active Library Services. Finally, user agents performing a task on the server or on a client node may also utilize ALiS functionality (e.g., to obtain or update user profiles) and may communicate among themselves. These interactions among AGS components are illustrated in Figure 2. For clarity, detail has been suppressed and only AGS server and patron components are displayed. The general functionality offered by each component is described below. A detailed specification of each function is provided in [4].

2.3.1 Active Library Service (ALiS)

ALiS relies on the available storage facilities in the digital library to provide a persistent representation of AGS abstractions and run-time data. In each library server, a database is defined for storing AGS objects. ALiS includes functions to facilitate the visualization and administration of each of the abstractions introduced by AGS. ALiS also play a central role in the delegation of tasks to agents. Upon instantiation of an agent class via the UAD, ALiS verifies the parameters

received and retrieves the appropriate agent class attributes. If agents in the class must be executed in the requesting client, ALiS retrieves the code referred to by the class' *do_actions* attribute and sends it to the UAD for remote execution. If the agent must be started on the server, the corresponding actions are retrieved and executed (using operating system calls or the language interpreter indicated in the class definition). The code is passed any parameters received from the client. If an agent may run on either the client or the server side, ALiS uses information on the task or current server load to decide whether to execute the agent's actions locally or remotely. Similar treatment is given to requests for execution of *summon* and *undo* actions. If an agent definition involves actions to be performed upon events occurring in the object repository, rules are installed upon instantiation in the underlying database that will trigger the actions when specified conditions are met. ALiS' clients may explicitly request that an agent (instance) be dismissed or its mission be suspended or resumed. In each case, ALiS sends a message (*terminate, sleep, awake*) to the appropriate instance and removes, deactivates or activates any associated rules in the database.

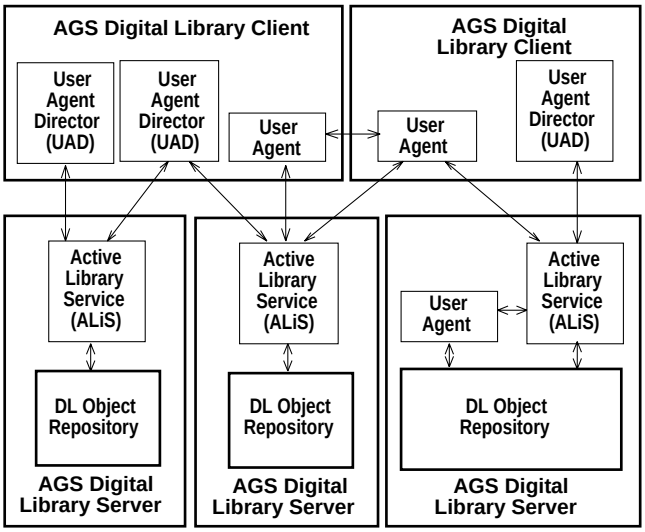

Figure 2. Interactions in the AGS architecture.

2.3.2 User Agent Manager (UAM)

The *User Agent Manager* (UAM) is a convenient interface that allows digital library administrators to utilize ALiS functionality. Although accessible only to authorized users, the UAM may also reside in any node in the network. The UAM presents its user with options to view, add and drop actions, user profiles, agent classes and agencies. The status of agent instances may be viewed and the librarian may temporarily suspend, resume or terminate an agent instance based on general server or network requirements if needed. The main role of the UAM is to receive user selections and issue the appropriate requests for ALiS services.

2.3.3 User Agent Director (UAD)

This component provides the means for end users to subscribe to AGS services, instantiate (i.e. delegate tasks to) agents of existing classes, and to monitor and control their performance. After the user is authenticated and has selected a library, the UAD presents all available agent classes

(obtained from ALiS) as well as information on any existing agent instances previously started by the user. If the user contacts other AGS libraries, any new library's agent classes and instances are added to existing available information for the user to control. If the user so desires, the UAD may arrange existing agent instances according to specified criteria (e.g., location, class or status). For each existing instance, the user may choose to suspend or resume assigned tasks, permanently dismiss the agent instance, invoke the undo option (if applicable) to counteract agent actions or inspect the status of an assigned task by summoning the agent. The UAD maintains up-to-date agent status information by periodically polling ALiS or by responding to explicit requests from the user.

Agent instantiation implies more involved work from the UAD. Using the agent class definition obtained from ALiS, the UAD generates an interface to allow the user to supply instance-specific information (name, task description) as well as any required and optional parameters. The information input by the user is submitted to the ALiS process residing on the digital library server associated with the appropriate class definition. ALiS may start the agent's actions or may send the code to the client for execution. In the latter case, the UAD receives and executes the code or passes it to the appropriate interpreter. Execution of binary code received in this fashion may optionally be disabled in order to reduce potential security problems (as in Java).

2.3.4 A Toolkit for Agent Construction

Developers are free to employ any suitable tools or algorithms in the implementation of user agents to be incorporated into AGS. However, in order to facilitate the development of agents in the proposed active digital library architecture, a *Toolkit for AGS* (TAGS) has been designed. The programming primitives offered by TAGS allow the agent developer to specify agent functionality using high-level abstractions involving AGS actions, digital library objects and DBMS features such as rules and event generators. TAGS also offers four major groups of functions for access by agent instances: user, agency, instance, and rule management functions. By using TAGS, the programmer does not need to know specific details of the DBMS underlying the digital library or the communication mechanisms being employed.

3. The AGS-1 Prototype

The conceptual design presented in the previous section is amenable to various implementations based on existing client/server technologies, database management systems and techniques for modeling agent behavior. A prototypical implementation of the AGS architecture has been constructed that demonstrates the viability of the proposed system abstractions and components, and provides a testbed in which further experimentation with user agents can be conducted. The major AGS components and TAGS primitives described in the previous section, as well as several experimental agents, have been implemented. These instantiations of the AGS architecture and its agent construction toolkit are referred to as *AGS-1* and *TAGS-1*, respectively.

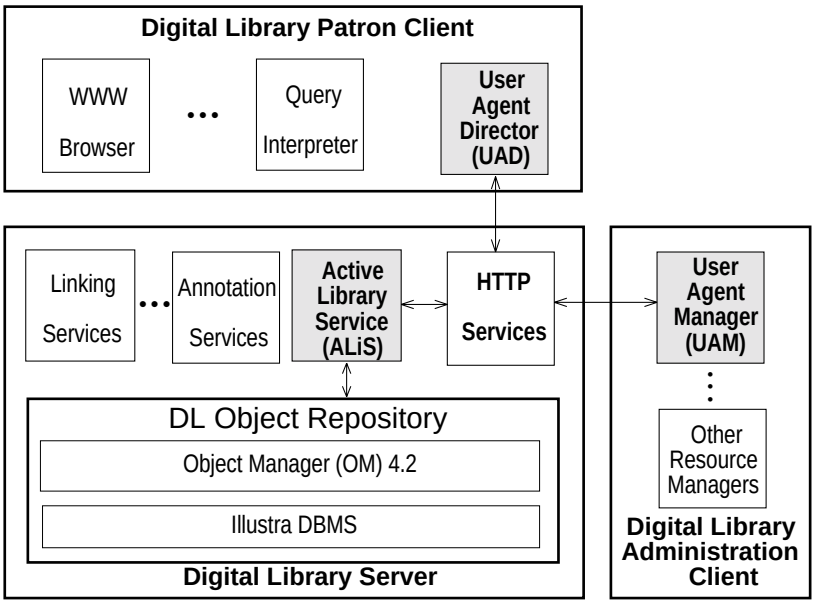

Figure 3. The AGS-1 prototype environment.

3.1 Prototype Environment

As illustrated in Figure 3, object repositories for prototypical digital libraries to be enabled by AGS-1 rely on the *Illustra* commercial DBMS [16] for storage management and on a locally developed server called the *Object Manager* (OM) [17], which provides a convenient abstraction layer for the manipulation of persistent objects. Abstraction layers and services other than user agents under development for this environment are documented in [18-19]. Indeed, one of the services available from the digital library server is a process that attends to requests from applications using the popular HTTP protocol. Client interfaces to services are provided via widely available WWW browsers, HTML forms, and in-house developed applications. The HTTP protocol and the Common Gateway Interface (CGI) [20] have been useful in implementing a highly portable interface to agent services. AGS-1 resources can thus be referenced using the common Uniform Resource Locator (URL) scheme [21]. In addition to HTML forms and documents, the components of AGS-1 have been written in C, with embedded SQL code via Illustra's application programming interface for direct access to Illustra databases. Inter-agent communication protocols can be specified using *H*, a general toolkit for inter-process communication developed locally [22] and available from ftp://bush.cs.tamu.edu/dist/hoss/H-4.2.0.tar.gz.

3.2 TAGS-1

TAGS-1 programming primitives are implemented in C. In addition to the programming primitives described previously, TAGS-1 also provides a number of implementation specific macros and functions. For example, Figure 4 shows the C/TAGS-1 code for a simple *PageMonitor* agent class. PageMonitor agents inform users whenever a specified object (e.g., a web page) in a digital library changes. While browsing a library, a user may find objects of interest that are likely to change. If the user wishes to keep abreast of any changes in such objects, he or she may periodically visit the

library site and check for any updates. Alternatively, an agent may be delegated the task of informing the user whenever objects are modified, thus saving user work and ensuring that no changes will be overlooked. PageMonitor agents use TAGS-1 functions and macros to obtain parameters supplied by ALiS and the user upon instantiation (Figure 4, line 18), to extract information from the received parameters (lines 20-22) and to convert between TAGS-1 and OM data formats (line 28). After the agent contacts the digital library repository (line 32), PageMonitors use one TAGS-1 primitive to register attributes, actions, and desired similarity level of objects of interest in the active repository maintained by ALiS (line 34).

```
1:       /* PageMonitor: INFORM USER UPON CHANGES TO A DL OBJECT */
2:
3:       #include "AGS.h"
4:       #include "TAGS.h"
5:
6:       void PageMonitor(AGS_Param *raw_params)
7:       {
8:          unsigned long no_params, no_urls, no_actions;;
9:          OM_Prop *params;
10:         AGS_RC rc;
11:         char  **url, **actions;
12:         char *strid, *method;
13:         AGS_ID agid;
14:         int likeness;
15:         AGS_CONN DL_conn;
16:
17:         /* GET INSTANTIATION PARAMETERS */
18:         TAGS_GetParams(raw_params, &no_params, &params, &rc);
19:         /* PARAMS TO USE: INSTANCE ID, ACTIONS UPON CHANGE, URL */
20:         TAGS_INST_ID(no_params, params, strid);
21:         TAGS_CHG_ACTIONS(no_params, params, no_actions, actions);
22:         TAGS_PARAM_VALS(no_params, params, "URL", no_urls, url);
23:
24:         /* INTERESTED ONLY IN EXACT, DIRECT ATTRIBUTE MATCH (URL) */
25:         likeness = 100;
26:         method = "direct";
27:         /* PUT INSTANCE ID IN AGS FORMAT */
28:         TAGS_STR_TO_ID(strid, agid);
29:
30:         /* CONNECT TO THE OBJECT REPOSITORY WHERE AGENT IS
31:            TO BE ACTIVATED */
32:         DL_conn = TAGS_CONNECT;
33:
34:         TAGS_OnChange(DL_conn, agid, no_urls, url, likeness,
35:                       method, no_actions, actions, &rc);
36:
37:         /* DISCONNECT FROM OBJECT REPOSITORY */
38:         TAGS_DISCONNECT(DL_conn);
39:      }
```

Figure 4. Code for sample agent using TAGS-1.

3.3 ALiS

ALiS is implemented in AGS-1 as a server process that communicates with its clients utilizing the H toolkit. As shown in Figure 3, interactive sessions with ALiS in AGS-1 rely upon CGI scripts invoked using the HTTP protocol, which in turn issue calls to ALiS functions. This allows for fast prototyping of independent client interfaces that can be utilized in a wide range of hardware and operating system platforms. This implementation of ALiS includes provisions for server-side and client-side agent execution. Upon retrieving an agent class, if ALiS finds that it should be executed on the client side, its *do* actions are retrieved and attached to the reply sent to the UAD. In AGS-1, client-side agent classes can only be written as Java applets. When ALiS retrieves a client-side agent class for which the *language* attribute is “Java,” it generates appropriate markup referencing the class’ *do_actions* as part of the HTML document sent as the reply to the instantiation request. A Java-aware web browser will download and execute the actions referenced in this fashion.

3.4 End User Interfaces

The implementation of the UAM and UAD interfaces in AGS-1 takes advantage of the availability of a process servicing HTTP requests on the digital library server. Functionality for administration and utilization of AGS resources is thus accessible via popular web browsers. When accessing a digital library using a web browser, a patron may find links to available AGS services, which are suggested by the librarian at places where agents are considered potentially helpful, as illustrated in Figure 5 (this is a research version of the Bush Presidential Digital Library). Alternatively, a patron may directly start a session with the UAD by providing the URL for this service at an AGS-enabled digital library. In either case, the UAD allows the user to enter basic identification or register as a new user. The UAD obtains all available agent classes and any agent instances previously initiated by the user and presents the interface shown in Figure 6.

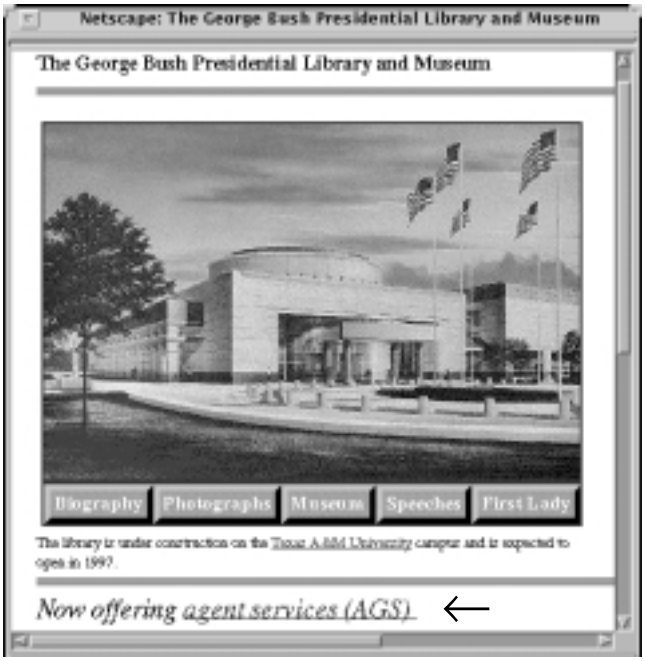

Figure 5. User access to AGS from a digital library.

The UAD's main interface is divided into two major sections. The top section presents the names of the user agents (agent instances) that have been assigned tasks by the user, along with their corresponding agent classes and current status. In Figure 6, two of the agents, *Bush Home Watcher* and *Speech Watcher* belong to the *PageMonitor* class defined previously and are active. *Desert Stormer* belongs to a *Topic Expert* agent class and its task has been suspended by the user. The user may select any of the available agents and, at the click of a button, summon the agent to examine its status, temporarily suspend an agent's task, resume agent tasks that have been previously suspended, or permanently dismiss one or all of the agents.

The bottom section of the UAD interface (Figure 6) presents the agent classes available to the end user for instantiation. Upon selection of an agent class, the UAD obtains the corresponding definition from ALiS and builds an HTML form that can be filled out by the user to specify the task the agent is to perform. Figure 7 shows the interface generated for the instantiation of *PageMonitor*

agents and the information provided by the end user to activate the *Bush Home Watcher* agent instance. As indicated in the class definition , the user may specify actions to be performed by the agent whenever a change occurs involving the specified object. The UAD obtains the applicable actions from ALiS and presents them to the user, who has selected *log* and *mail*. Upon receipt of this form, ALiS executes the class' *do* actions with the given parameters (since PageMonitor was defined as a server-side agent class) and stores a new instance object in its OM repository. The behavior of the agent in this case is determined by the code shown in Figure 4, which installs appropriate rules in the digital library repository. In the AGS-1 environment, e-mail notifications are also formatted as HTML documents so that any references to digital library objects via URLs can be immediately followed by the user.

Figure 6. Interface to AGS via the UAD.

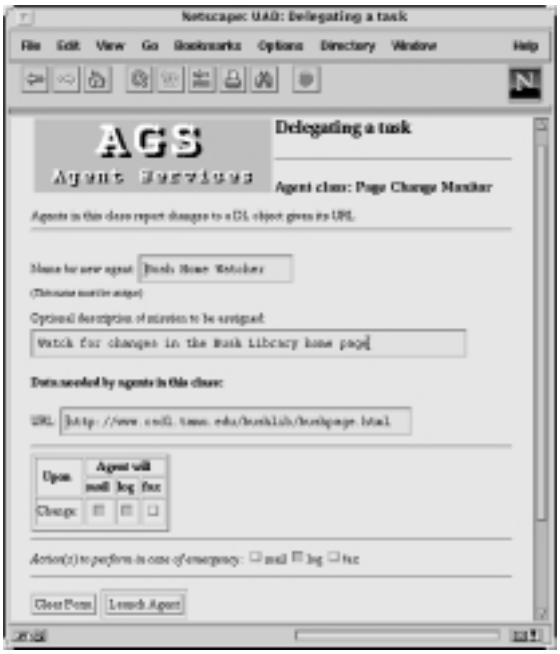

Figure 7. Instantiation of an agent class.

4. Evaluation of AGS

AGS enables the development of agents that rely on a wide variety of construction approaches while maintaining a unified view of an active environment for both digital library administrators and patrons. The generality, openness and flexibility of the AGS architecture are discussed below by showing how it relates to other specific agent implementations and agent architectures.

4.1 Incorporating Specific Agent Implementations

A large number of user agents with specific functionality have been designed and implemented. A survey of recent work can be found in [4]. Experimenting with specific user agents such as those described in the literature is not only possible in the context of an AGS-enhanced digital library, but

advantages also exist in incorporating such agents into an environment like AGS. These programs can be defined as agent classes, allowing users to experiment with different parameter values and providing persistency for search goals such as those required by WebWatcher [23]. Modules that are commonly used in the construction of information agents (e.g., implementation of information retrieval techniques) can be registered as AGS actions, shared and experimented with by different agent classes, and used in agent construction via TAGS primitives. Agents such as Video Navigator [24] and the Media Lab's Mail and News agents [25] could share user profiles and can take advantage of TAGS primitives designed for user profile manipulation.

Agents that are designed to communicate with other agents may do so in AGS by taking advantage of ALiS services, which may be requested by using TAGS primitives. Protocols for inter-agent communication can be defined for each agent class and their complexity may range from the simple, special-purpose messages utilized by the Expertise Locators [26], to the richer, typed exchanges of the Collaborative Mail Agents [27], to a full-fledged protocol such as that required by agents based on KQML communication [28]. In AGS-1, the specification of the communication protocol for each agent class utilizes the facilities provided by the H toolkit.

4.2 Relationship with Other Architectures

Distinctive characteristics of AGS in relation to other agent architectures are its orientation to *user agents*, its emphases on *user control*, the *utilization of existing active features* on behalf of the user, and the provision of *agents as a service* to which users can subscribe. Indeed, AGS also distinguishes itself by enhancing emerging views of digital libraries with the addition of agent services for patrons. AGS is committed to make agents available as an abstraction to the end user and to provide a testbed for the exploration, by both users and developers, of agents as a basis for human-computer interaction. Although other notions of agents may be useful in the design and implementation of user agents, AGS neither requires nor rules out the utilization of those related notions. Specific differences and similarities between AGS and agent architectures which have implications for the implementation of user agents are further discussed below.

4.2.1 SodaBot

The view offered by SodaBot [29], is that of an agent "engine" associated with each user on a computer network. Each of these engines, referred to as *Basic Software Agents* (BSAs) is capable of executing programs (called *agents*) written in the SodaBotL language. SodaBotL includes support for specific *information agent* functionality such as mail filtering rules and *network agent* functionality, supporting the movement of code between BSAs and inter-agent communication. End users are presented with an interface to their personal BSA via which general agent behavior can be configured, as well as an editor for the manipulation of SodaBotL code that can be used for the definition of new agents.

Like AGS, SodaBot is extensible via tools specifically provided for agent development. In both AGS and SodaBot, agents can communicate among themselves, and users can exert control over agent activity. Unlike AGS agents, however, SodaBotL agents can only be written in the supplied

agent language and use only a pre-defined communication protocol. Control mechanisms provided by SodaBot refer to all agents executed by a given BSA. For example, all agents working for a given user may be prevented from modifying mail files. AGS's open design allows for the incorporation of diverse tools in the construction of agents. Each agent class may define its own communication protocol (to be used by its instances). Using the UAD interface, end users may control individual agent instances. Each class of agents may define specific behaviors to be performed when the user summons, suspends or terminates agent instances. Whereas in SodaBot end users wishing to define their own agents are required to know the SodaBotL language, AGS takes advantage of its particular digital libraries setting to divide this task into agent class definition, to be performed by service administrators (librarians), and agent instantiation and customization, to be performed by end users (patrons). AGS also includes provisions for the construction of interfaces that will allow end users to build their own agent classes based on well-defined actions available from the AGS server.

4.2.2 Open Agent Architecture

The Open Agent Architecture (OAA) [30] makes delegation facilities available to the end user via a natural language interface. OAA is viewed by the user as a single agent with multiple capabilities. Functions that have been integrated into OAA include e-mail, news, and calendar management. The *developers* have conceived of this architecture as a collection of agents which a server process may select or combine according to the tasks delegated by the end user. Even the programs accepting natural language queries or the text-to-speech output system are considered "agents" in OAA. The architecture is open in that it allows for the incorporation of new (programmer) agents and agent design can rely on functionality provided by existing stand-alone applications.

Major similarities between OAA and AGS are their openness and extensibility. Both environments are designed to take advantage of functionality provided by existing applications in the design of agents and both provide tools to facilitate agent construction. OAA's function library provides architecture-specific methods for blackboard access, vocabulary handling, and inter-agent communication. As in SodaBot, however, communication among agents is based on a fixed protocol that uses notions from Speech Act Theory.

The most salient differences between OAA and AGS are derived from their distinct research goals and the views that each architecture presents to the end user. OAA focuses on issues that are considered central to *programmer agents* and agent-based system design. Deriving appropriate goals from natural language requests and defining plans for goal accomplishment using available agents are among the issues investigated by OAA. Since the agents implementing delegated tasks in OAA are actually hidden from the end user, addressing research issues of paramount importance for *user agents* such as agent presentation, user control, and inspectability, is not straightforward or not possible. AGS is explicitly designed to make agents visible to end users and to foster the interaction between users and multiple agents in such a way that the investigation of agency as the basis for a user interface paradigm is enabled.

4.2.3 UMDL

As noted in the introduction, the University of Michigan Digital Library (UMDL) is distinctive because it has been described consistently in terms of agents [2, 3]. Agents in UMDL are a pervasive abstraction used to represent such diverse notions as resources, tasks and interfaces which are planned as part of the projected digital library. User interface agents (UIAs) will provide search strategies to users accessing UMDL. Mediator agents will perform tasks required to refer a query from a UIA to a collection. Collection interface agents (CIAs) will publish the contents and capabilities of collections. Work has been conducted that conceptualizes system components as task planning agents (TPAs), which decompose tasks and form teams to accomplish them [31].

The differences between AGS and UMDL with respect to agency issues are clearly related to their research emphases. Whereas UMDL will attempt to utilize the abstraction of *agents* (for the designer's benefit) in the construction of digital library functionality, AGS focuses in making agents available to end users as a service and an alternative interface to digital library resources. UIAs are perhaps one design feature that has the potential to play the role of *user agents* as proposed by AGS.

5. Future Work

Much effort is being focused on producing a more complete and functional implementation of the AGS architecture. Notification mechanisms have been planned for inclusion into the Object Manager abstraction layer of the digital library repositories. This would allow one to implement TAGS primitives with considerable independence from the underlying storage manager (Illustra in AGS-1). More sophisticated and more helpful agents are on schedule. In particular, agents are under development which will cooperate with users in the construction of large object repositories. Another class of agents will provide users (mainly taxonomists) with alternative organizations of objects in digital libraries. As digital libraries evolve and patrons begin to take advantage of the services offered, potential applications and new requirements for AGS will become apparent.

Acknowledgments

This work has been partially funded by the Mexican National Council for Science and Technology (Conacyt), the Center for Botanical Informatics of the Missouri Botanical Garden, and the Institute for Research and Graduate Studies (INIP) of the Universidad de las Américas-Puebla.

References

1. Kahn, R., and Cerf, V. 1988. *The Digital Library Project. Volume 1: The World of Knowbots*. Corporation for National Research Initiatives, Reston, VA.

2. Birmingham, W., Drabenstott, K., Frost, C., Warner, A., and Willis, K. 1994. The University of Michigan digital library: This is not your father's library. In *Proceedings of Digital Libraries '94* (College Station, Tex., June). Hypermedia Research Laboratory, Texas A&M University, College Station, Tex., 53-60.

3. Atkins, D., Birmingham, W., Durfee, E.,, Glover, E., Mullen, T., Rundensteiner, E., Soloway, E., Vidal, J., Wallace, R., and Wellman, P. 1996. Toward inquiry-based education through interacting software agents. *IEEE Computer 29*, 5 (May).

4. Sánchez, J. A. 1996. *Agent Services*. Ph.D. Dissertation. Department of Computer Science, Texas A&M University, College Station, Tex.

5. Baecker, J. Grudin, W. Buxton, and S. Greenberg (Eds.) 1995. *Readings in Human-Computer Interaction: Toward the Year 2000*. San Francisco: Morgan Kaufmann Publishers.

6. Laurel, B. 1990. Interface agents: Metaphors with character. In *The Art of Human-Computer Interface Design*, Brenda Laurel, Ed. Addison-Wesley, Reading, Mass., 355-365.

7. Sánchez, J. A. 1993. HyperActive: Extending an open hypermedia architecture to support agency. M.S. thesis. Department of Computer Science, Texas A&M University, College Station, Tex., December.

8. Sánchez, J. A. 1994. User agents in the interface to digital libraries. In *Proceedings of Digital Libraries '94* (College Station, Tex., June). Hypermedia Research Laboratory, Texas A&M University, College Station, Tex., 217-218.

9. Norman, D. 1994. How might people interact with agents. *Commun. ACM 37*, 7 (July), 68-71.

10. Sánchez, J. A., Azevedo, F. S., and Leggett, J. J. 1995. PARAgente: Exploring the issues in agent-based user interfaces. In *Proceedings of the First International Conference on Multi-Agent Systems (ICMAS-95*) (San Francisco, Calif., June), 320-327.

11. Crane, G. 1996. Building a digital library: The Perseus Project as a case study in the humanities. In *Proceedings of the 1st ACM International Conference on Digital Libraries* (Bethesda, Md., March). New York: ACM Press, 3-10.

12. Fox, E., D. Hix, L. Nowell, D. Brueni, W. Wake, and L. Heath 1993. Users, user interfaces, and objects: Envision, a digital library. *Journal of the American Society for Information Science 44*, 8 (Sept.), 480-491.

13. Gladney, H., E. Fox, Z. Ahmed, R. Ashany, N. Belkin, and M. Zemankova, 1994. Digital library: Gross structure and requirements: Report from a March 1994 workshop. In *Proceedings of Digital Libraries '94*, (College Station, Tex., June). College Station, Tex.: Hypermedia Research Laboratory, Texas A&M University, 101-107.

14. Kahn, R., and R. Wilensky 1995. A framework for distributed digital object services. Tech. Rep. TN95-01. Reston, Va.: Corporation for National Research Initiatives.

15. Nürnberg, P., R. Furuta, J. Leggett, C. Marshall, and F. Shipman 1995. Digital libraries: Issues and architectures. In *Proceedings of Digital Libraries '95* (Austin, Tex., June). College Station, Tex.: Center for the Study of Digital Libraries, Texas A&M University, 147-153.

16. IUG 1995. *Illustra User's Guide*. Release 3.2. Oakland, Calif.: Illustra Information Technologies, Inc.

17. Sánchez, J. A. 1995. The Object Manager 4.2 interface specification. Draft available from Center for the Study of Digital Libraries, Texas A&M University, College Station, Tex. (Also available from http://www.csdl.tamu.edu/~joseash/om4.2.html).

18. Furuta, R., C. Marshall, F. Shipman, and J. Leggett, J. 1996. Physical objects in the digital library. In *Proceedings of the 1st ACM International Conference on Digital Libraries* (Bethesda, Md., March). New York: ACM Press, 109-115.

19. Nürnberg, P., J. Leggett, E. Schneider, and J. Schnase 1996. Hypermedia operating systems: A new paradigm for computing. In *Proceedings of the Seventh Conference on Hypertext* (Washington, D.C., March). New York: ACM Press, 194-202.

20. NCSA 1996. The Common Gateway Interface. Specification available from National Center for Supercomputing Applications, University of Illinois at Urbana-Champaign , Ill. (Also available from http://hoohoo.ncsa.uiuc.edu/cgi/).

21. Berners-Lee, T., L. Masinter, and M. McCahill 1994. Uniform Resource Locators (URL). Internet Request for Comments No. RFC 1738. Available from InterNIC Directory and Database Services, AT&T, 5000 Hadley Road, Room 1B13 South Plainfield, N.J. (Also available from gopher://ds0.internic.net:70).

22. Nürnberg, P. 1994. Implications of an open, extensible, and distributed hypermedia information system architecture for inter-process communication subsystem design. M.S. Thesis, Department of Computer Science, Texas A&M University, College Station, Tex.

23. Armstrong, D. Freitag, T. Joachims, and T. Mitchell 1995. WebWatcher: A learning apprentice for the World Wide Web. In *Information Gathering from Heterogeneous, Distributed Environments: Papers from the 1995 AAAI Spring Symposium* (Menlo Park, Calif., March), C. Knoblock and A. Levy, Eds. Menlo Park: AAAI Press, 6-12.

24. Burke, R., and Hammond, K. 1995. Combining databases and knowledge bases for assisted browsing. In *Information Gathering from Heterogeneous, Distributed Environments: Papers from the 1995 AAAI Spring Symposium* (Menlo Park, Calif., March), C. Knoblock and A. Levy, Eds. AAAI Press, Menlo Park, Calif., 25-29.

25. Maes, P. 1994. Agents that reduce work and information overload. *Commun. ACM 37*, 7 (July), 31-40.

26. Kautz, H., A. Milewski, and B. Selman 1995. Agent amplified communication. In *Information Gathering from Heterogeneous, Distributed Environments: Papers from the 1995 AAAI Spring Symposium* (Menlo Park, Calif., March), C. Knoblock and A. Levy, Eds. Menlo Park: AAAI Press, 78-84.

27. Lashkari, M. Metral, M., and P. Maes 1994. Collaborative interface agents. In *Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI '94)* (Seattle, Wash., Aug.). Menlo Park: AAAI Press, 444-450.

28. Genesereth, M., and Ketchpel, S. 1994. Software agents. *Commun. ACM 37*, 7 (July), 47-53.

29. Coen, M. 1994. SodaBot: A software agent environment and construction system. Tech. Rep. 1493, Artificial Intelligence Laboratory, Massachusetts Institute of Technology, Cambridge, Mass.

30. Cohen, P., Cheyer, A., Wang, M., and Baeg, S. 1994. An open agent architecture. In *Software Agents: Papers from the 1994 Spring Symposium* (Menlo Park, Calif., March). AAAI Press, Menlo Park, Calif., 1-7.

31. Vidal, J., and Durfee, E. 1995. Task planning agents in the UMDL. In *Proceedings of the CIKM'95 Intelligent Information Agents Workshop* (Baltimore, MD) (Dec.).

32. Sánchez, J. A. 1997. A Taxonomy of agents. Tech. Rep. ICT-97-1. Laboratory of Interactive and Cooperative Technologies, Department of Computer Systems Engineering, Universidad de las Américas-Puebla. Cholula, Puebla, México.