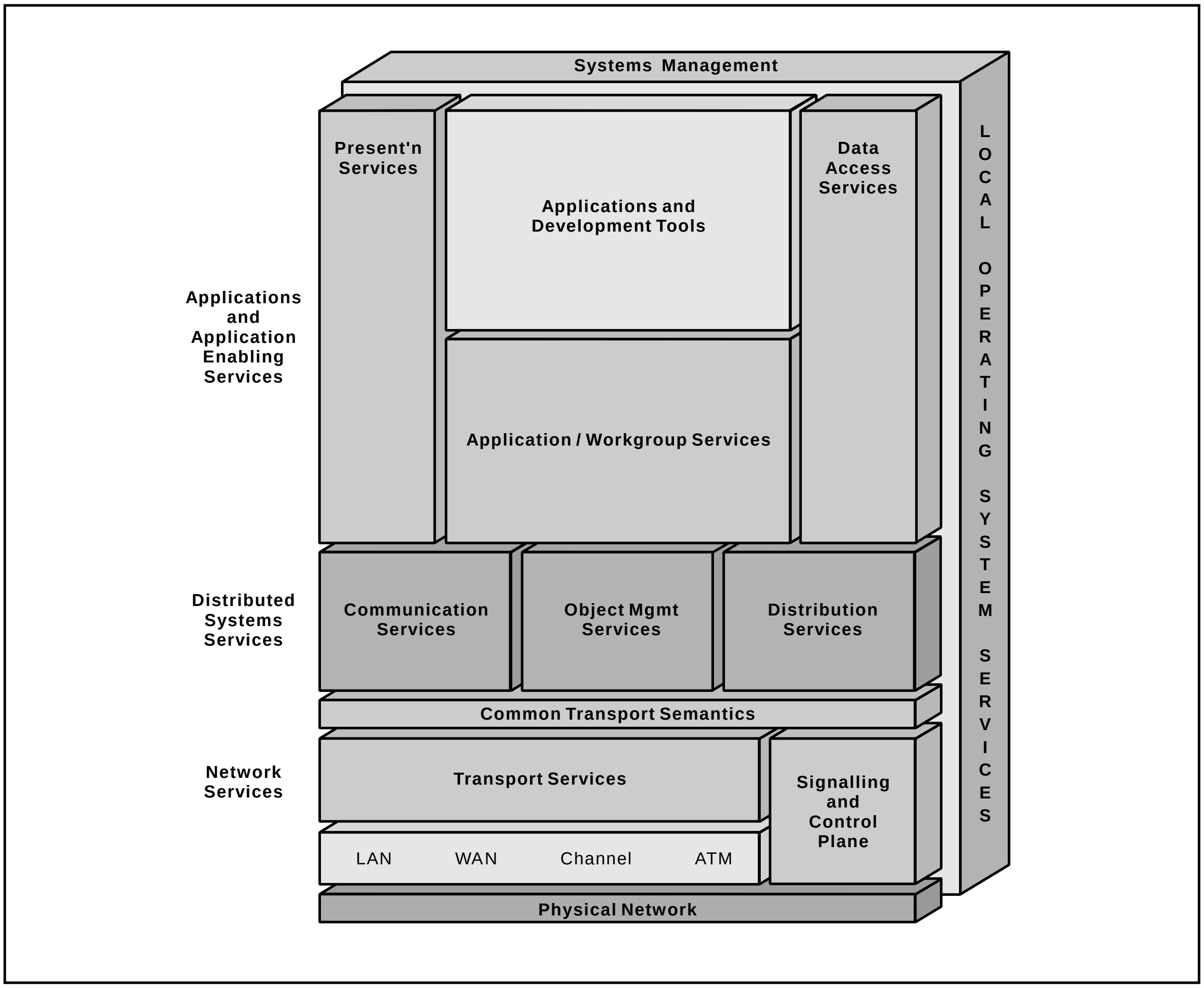
Open Blueprint

Intelligent Agent Resource Manager


Systems Management
Present'n Services
Applications and Development Tools
Data Access Services
Application / Workgroup Services
Applications and Application Enabling Services
Distributed Systems Services
Communication Services
Object Mgmt Services
Distribution Services
Common Transport Semantics
Network Services
Transport Services
Signalling and Control Plane
LAN
WAN
Channel
ATM
Physical Network
LOCAL OPERATING SYSTEM SERVICES

Open Blueprint

Intelligent Agent Resource Manager

G325-6592-00

About This Paper

Open, distributed computing of all forms, including client/server and network computing, is the model that is driving the rapid evolution of information technology today. The Open Blueprint structure is IBM's industry-leading architectural framework for distributed computing in a multivendor, heterogeneous environment. This paper describes the Intelligent Agent resource manager component of the Open Blueprint and its relationships with other Open Blueprint components.

The Open Blueprint structure continues to accommodate advances in technology and incorporate emerging standards and protocols as information technology needs and capabilities evolve. For example, the structure now incorporates digital library, object-oriented and mobile technologies, and support for internet-enabled applications. Thus, this document is a snapshot at a particular point in time. The Open Blueprint structure will continue to evolve as new technologies emerge.

This paper is one in a series of papers available in the *Open Blueprint Technical Reference Library* collection, SBOF-8702 (hardcopy) or SK2T-2478 (CD-ROM). The intent of this technical library is to provide detailed information about each Open Blueprint component. The authors of these papers are the developers and designers directly responsible for the components, so you might observe differences in style, scope, and format between this paper and others.

Readers who are less familiar with a particular component can refer to the referenced materials to gain basic background knowledge not included in the papers. For a general technical overview of the Open Blueprint, see the *Open Blueprint Technical Overview*, GC23-3808.

Who Should Read This Paper

This paper is intended for audiences requiring technical detail about the Intelligent Agent Resource Manager in the Open Blueprint. These include:

- Customers who are planning technology or architecture investments
- Software vendors who are developing products to interoperate with other products that support the Open Blueprint
- Consultants and service providers who offer integration services to customers

Contents

Figures

Intelligent Agents in the Open Blueprint

Intelligent agents are software entities that perform a set of operations on behalf of a user or another program with some degree of autonomy, and in so doing, employ some knowledge or representation of the user's goals or desires. Intelligent agents help the user in the following ways:

- In the network computing environment, an enormous amount of information is available to the user from a wide variety of sources. Intelligent agents can filter this flood of information, passing on to the user only those items in which the user is interested. Intelligent agents can automate the retrieval and processing of information obtained from the network. In this way, intelligent agents reduce the information complexity of the network computing environment.
- The sophistication and complexity of computer hardware and software continues to increase, and the ever-expanding population of users (many of them novices) is clamoring for increased ease-of-use. Intelligent agent technology applied to user interface design can provide smart user interfaces that can detect when a user is having difficulty, and assist the user to resolve or bypass the problem. Smart interfaces can observe and learn a user's preferences and habits, and automate actions that the user routinely performs. In this way, intelligent agents reduce a task's complexity.
- Advances in technology have made mobile computing a reality. Mobile users are not always connected to the network, and, when they are connected, it can be over a variety of media with different bandwidth, reliability, and security characteristics. Intelligent agent technology can be used to create surrogates within the network that represent mobile users. These surrogates can do work on behalf of the mobile users they represent, even when the mobile user is disconnected. When the mobile user reconnects, agent technology can be used to tailor how data is transferred to the mobile user to match the characteristics of the connections. Intelligent agents can reduce the complexity associated with end user mobility.

The Intelligent Agent resource manager described in this document helps instantiate intelligent agents at initialization time, helps manage intelligent agents at run time, and provides common services that intelligent agents need.

Intelligent Agents

Intelligent agents can be characterized in terms of three dimensions: intelligence, agency, and mobility.

Intelligence reflects the degree of reasoning and learned behavior in an agent. Intelligence describes the agent's ability to accept the user's statement of goals and carry out the task the user delegated to it. The agent's goals and behaviors could be encoded in a simple script that is executed by an interpreter in response to an event. Or, the reasoning could be provided by a set of rules that encodes strategy and goals. Sophisticated agents could learn and adapt to their observed environment, both in terms of the user's objectives and in terms of the resources available to the agent to carry out its task.

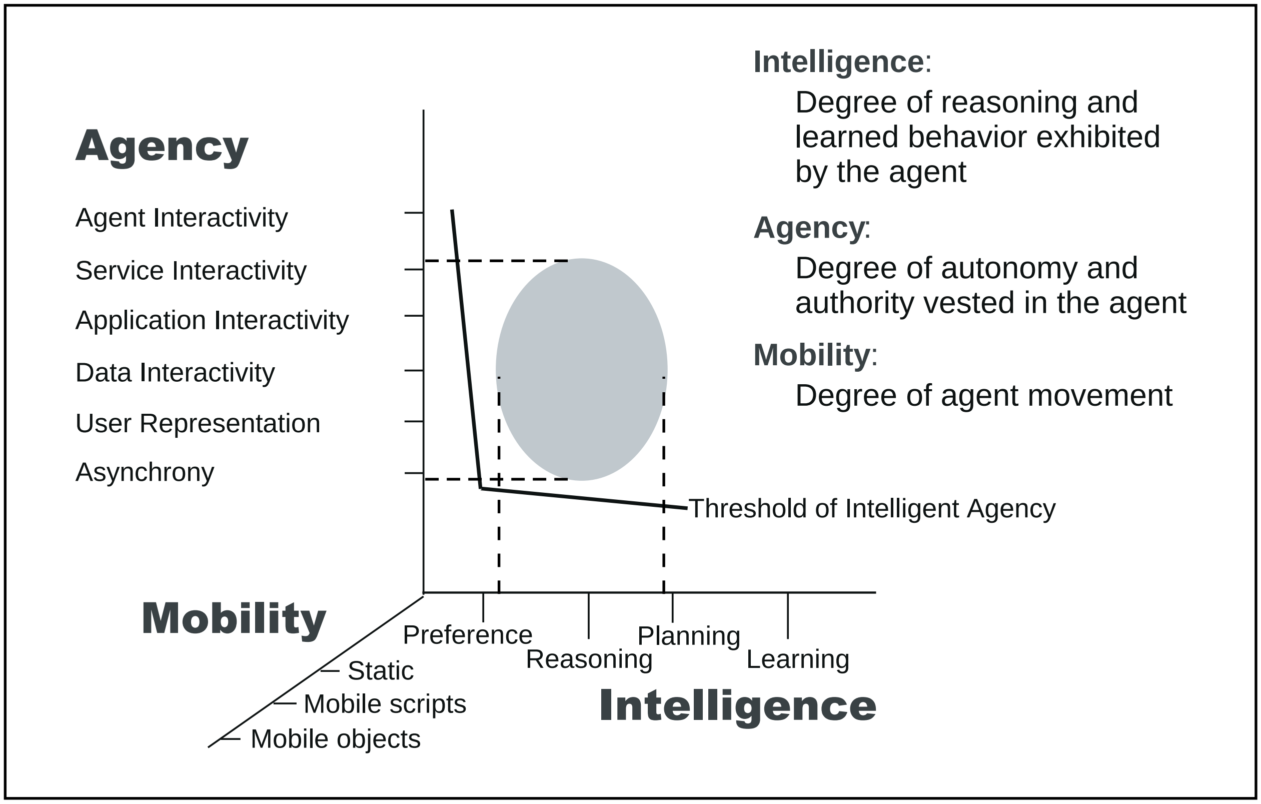


Figure 1. Intelligent Agent Scope

Agency is the degree of autonomy and authority that an agent possesses. Agents must run asynchronously, and can interact to varying degrees with other applications, databases, and resource managers. More sophisticated agents can actually collaborate and negotiate with other agents to accomplish complicated tasks such as scheduling meetings or conducting electronic auctions.

Mobility is the degree to which agents actually travel through the network. At the low end of the mobility axis are *static* agents, which do not travel. (The fact that an agent is static does not preclude it from being distributed.) At the next level, scripts can be composed on one machine and shipped to another machine for execution. Because the script travels before execution, no state information needs to accompany it. At the highest level are mobile agents that can suspend themselves in mid-execution, travel to another location, then resume execution where they left off. These agents must travel with state information.

Software must contain a minimal level of capability to be defined as an intelligent agent. This capability is depicted by the L-shaped line in Figure 1 above. The oval-shaped area shows the kind of agents that are commonly deployed (primarily static agents that contain some amount of reasoning). In the near future, however, agent technology will expand along both the agency axis and the mobility axis. The topics of agent interaction, collaboration, negotiation, and agent mobility are very active areas in both the research community and in standards bodies. Appendix A, “Future Extensions” on page 9 discusses these areas.

There are several stages in the life cycle of an intelligent agent. Initially, the various components that make up the agent need to be built (*agent build*). Later, the relationship among the various components that make up the intelligent agent needs to be defined (*agent bind*). This definition is used to bind the various components of the agent together when the agent is started up (*agent instantiation*). After instantiation, the agent is operational (*agent execution*), until it is deactivated for some reason (*agent deactivation*). The Intelligent Agent resource manager provides services to an intelligent agent at all stages of its life cycle.

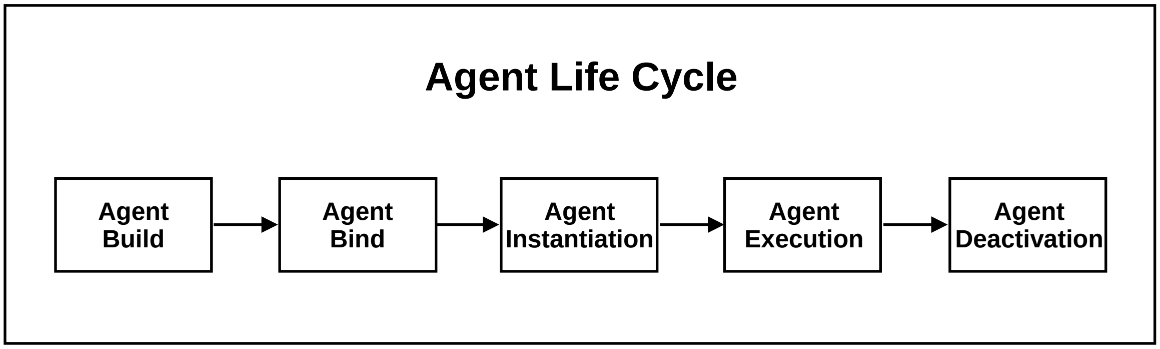


Figure 2. Intelligent Agent Life Cycle

The Intelligent Agent Resource Manager

The Intelligent Agent resource manager provides functions and facilities that an intelligent agent uses to do its job. Object oriented technology is the base for the resource manager's sub-components. Specific intelligent agents are composed of instances of the Intelligent Agent resource manager's subcomponents. Figure 3 below illustrates the structure of the Intelligent Agent resource manager and shows how the resource manager's subcomponents are also used as subcomponents of an intelligent agent. The five major sub-components (adapters, engines, knowledge, libraries, and views) are described in the next several sections.

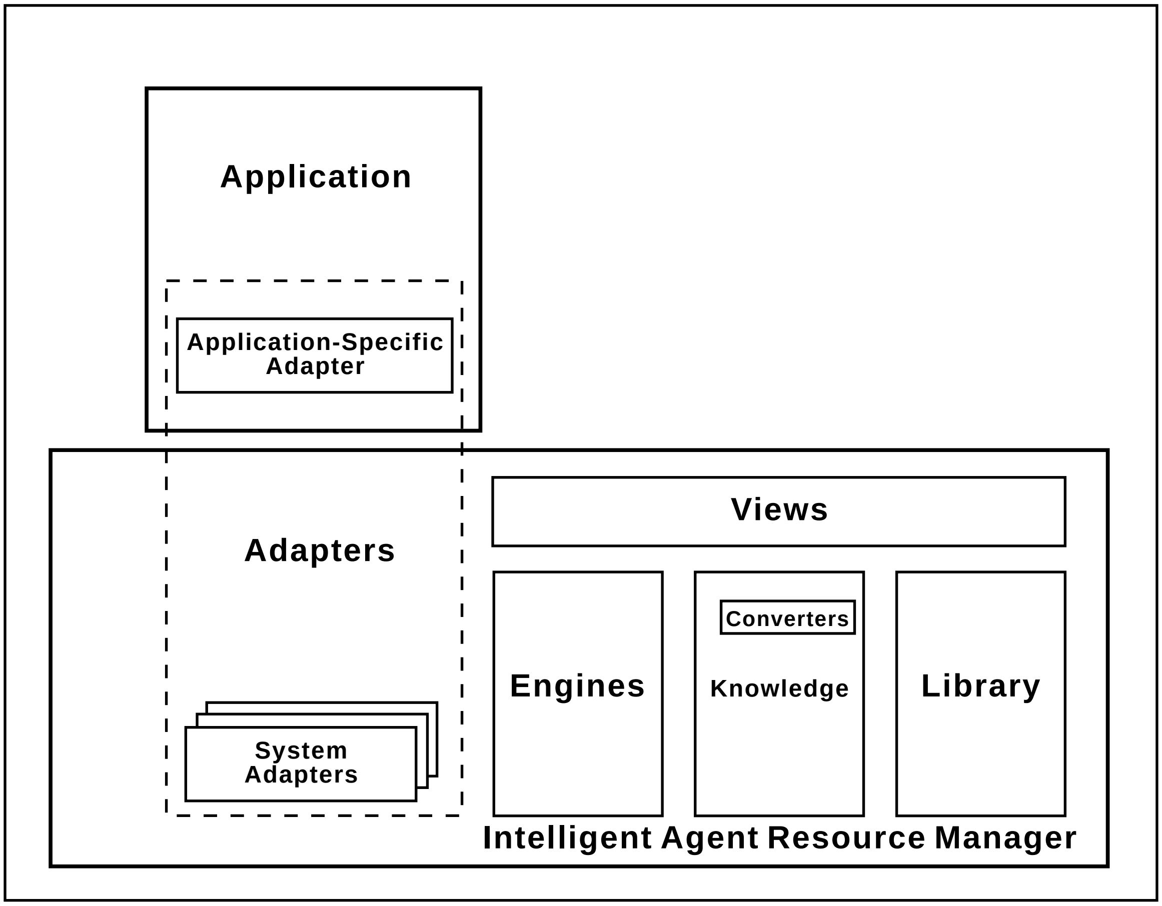


Figure 3. Sub-Components of an Intelligent Agent

Adapters

An intelligent agent receives events from the outside world, and, based on some level of intelligence, reacts to the event. The reaction could be as simple as "do nothing", or it could involve requesting more information (*sensing*), or initiating some action (*effecting*). In the intelligent agent architecture, software adapters interface to the outside world to communicate events to the agent, and to implement sensors and effectors for the agent. (For additional information about the adapter interface, see Appendix B, “Adapter Interface to the Intelligent Agent Resource Manager” on page 11.) In human terms, adapters are the eyes, ears, and hands of the agent. There are two broad categories of adapters:

- **Application-Specific Adapters**

  Detect events and perform actions within the domain of a particular application that an agent is augmenting. An example is an adapter that interfaces to an Internet news reader application, and provides news-related events such as "news article arrived" and actions such as "delete news", "save news", and so on. These adapters are considered part of the agent-enabled application. In fact, they are the glue used by the application to conform to the event/sensor/effector interface of the Intelligent Agent resource manager.

- **System Adapters**

  Provide connections to resources that are managed by other Open Blueprint resource managers such as file services, timer services, telephony services, paging services, and user interfaces.

Adapters are objects, and are instances of an adapter base class. Although the Intelligent Agent resource manager includes many adapter implementations, anyone (including application developers) can use the adapter base class to develop new adapters that operate with the Intelligent Agent resource manager.

Engines

Engines are the brains of an intelligent agent. There are several different types of engines. One type of engine is the *inferencing engine*. When notified of some event, inferencing engines operate on rule sets and perform complex symbolic reasoning to determine how to react to the event and what action to trigger.

Another type of engine is the *executive engine*. Executive engines execute a preprogrammed response (reaction) to an event. (A preprogrammed response could be a LotusScript script or a Java program.) The executive engine is only a proxy within the Intelligent Agent resource manager—the executive engine either uses the Virtual Machine resource manager to support Java, or it invokes a script language interpreter such as the LotusScript interpreter.

Another type of engine is the *reflective engine*. A reflective engine observes events and reflects upon the state of the knowledge currently possessed by the agent.

An agent might depend on the services of multiple engines. For example, an agent might react to events based on the operation of an inferencing engine, while at the same time, a reflective engine observes the same events and modifies the rules that drive the inferencing engine (learning).

Like adapters, engines are objects. A specific engine implementation is derived from an engine base class. The Intelligent Agent resource manager includes several engine implementations, but anyone can use the engine base class to develop a new engine that conforms to all the Intelligent Agent resource manager interfaces, and plugs into the Intelligent Agent resource manager.

Knowledge

Engines rely on some representation of *knowledge* on which to operate. An inferencing engine's knowledge is the set of the agent's rules and beliefs that encode the preferences and intent of the users that the agent represents. An executive engine's knowledge is the script or program that procedurally encodes the agent's goals and behavior. Other forms of knowledge (URLs the user recently visited, inverted indexes containing subjects that a user has recently browsed, and so on) can be maintained by reflective engines, and can also logically reside in the knowledge sub-component of the Intelligent Agent resource manager.

Library

While active, an engine can maintain its current working knowledge in an internal, optimized form. However, this knowledge needs to be persistent so it can be recovered across agent activations. The library sub-component provides the facilities required to make knowledge persistent. To facilitate the sharing of knowledge among different engines of the same type, knowledge should be stored in the library in some standard format. For example, to promote the sharing of rule sets among different inferencing engines, rule sets are stored in the library in Knowledge Interchange Format (KIF) format. (KIF is a proposed ANSI X3T2 standard.) A converter associated with each engine converts knowledge to and from the standard KIF format stored in the library and the engine's optimized internal representation of rules.

Prior to execution time, the knowledge bases used by the various engines can be initialized and administered in several ways:

- Some base set of generally-applicable default knowledge can be provided as part of the agent's initial knowledge base.
- System administrators can add knowledge items to the defaults such as:
  - Knowledge that applies to groups of users
  - Knowledge that applies to a particular organization such as company or departmental policy
- Users can modify their (and, if they have the proper authority, other) knowledge bases to tailor them to their particular preferences.

The library sub-component of the Intelligent Agent resource manager relies on the Open Blueprint Security resource managers to prevent unauthorized access or changes to the knowledge base.

Views

Users need a way to browse and edit rule sets and other types of knowledge to describe their preferences and goals to the agent. These browse and edit capabilities are contained in the views sub-component of the Intelligent Agent resource manager. Because rule sets are stored in KIF format in the library sub-component, different rule editors and browsers are more likely to be compatible. The views sub-component also contains a graphical user interface (GUI) that enables users to browse and edit other types of knowledge, for example, the URL trails and inverted indexes that a reflective engine might maintain.

Intelligent Agent Operation

When an intelligent agent is started (usually but not always when an application is started), the Intelligent Agent resource manager creates the necessary association (binding) of engines and adapters for the agent. If some of the adapters and engines required by the agent run as separate system processes, the Intelligent Agent resource manager makes sure they are started.

Once the agent is active, the adapters associated with the agent can either wait passively for events of interest to the agent, or they can actively poll the environment to see if any events of interest have occurred. In either case, when an event is detected, the adapter calls the Intelligent Agent resource manager to start the associated engine(s). In response to the event, the engine can ask an adapter (either the adapter that originated the event, or another adapter associated with the intelligent agent) to either sense or effect.

If the requested action fails or cannot be completed (for example, if the engine asks the pager adapter to page an individual, and the operation is unsuccessful), the adapter that implements the requested action notifies the engine using a failure event. The engine processes this event just like any other event. For example, a rules-based inferencing engine processes its rule set to see if there are any rules (or combination of rules) that specify what backup actions should be taken when such a failure occurs.

A Simple Example

An Internet news reader application is an ideal application for agent technology. In the example (illustrated in Figure 4 on page 7 below), the timer adapter sub-component of the agent associated with the news reader application generates an event to indicate that it is half-past the hour. This event is passed to the engine associated with the news reader application which, in this case, is a rules-based inferencing engine.

While processing its rules, the engine discovers a rule that says it should check for new news every half-hour. Consequently, the engine generates a "refresh newsgroups" action and sends it to the application-specific adapter associated with the news reader application. The adapter works with the news reader application to refresh the newsgroups. While refreshing the newsgroups, the application detects that new news has arrived. The application-specific adapter creates an event object (the format of which is part of the definition of the interface to the Intelligent Agent resource manager) and passes it to the Intelligent Agent resource manager. The event object contains some standard information: What is the application domain of the event? (in this case, Internet news), what is the event type? (in this case, "news item arrived"), what time did the event occur?, and so on. Other domain and event-type-specific data is also provided in the event object which in this case is the name of the newsgroup to which the item was posted (comp.lang.java), the author of the news item (John Doe), and the subject line (Java Security) would be included.

To handle this event, the engine processes its set of rules to define an action to perform which in this example is to add the news item to a custom newsgroup for the user. The engine generates an "add to newsgroup" action and sends it to the application-specific adapter, which carries out the action.

Other new news items that are discovered as a result of refreshing the newsgroups are handled in a similar way.

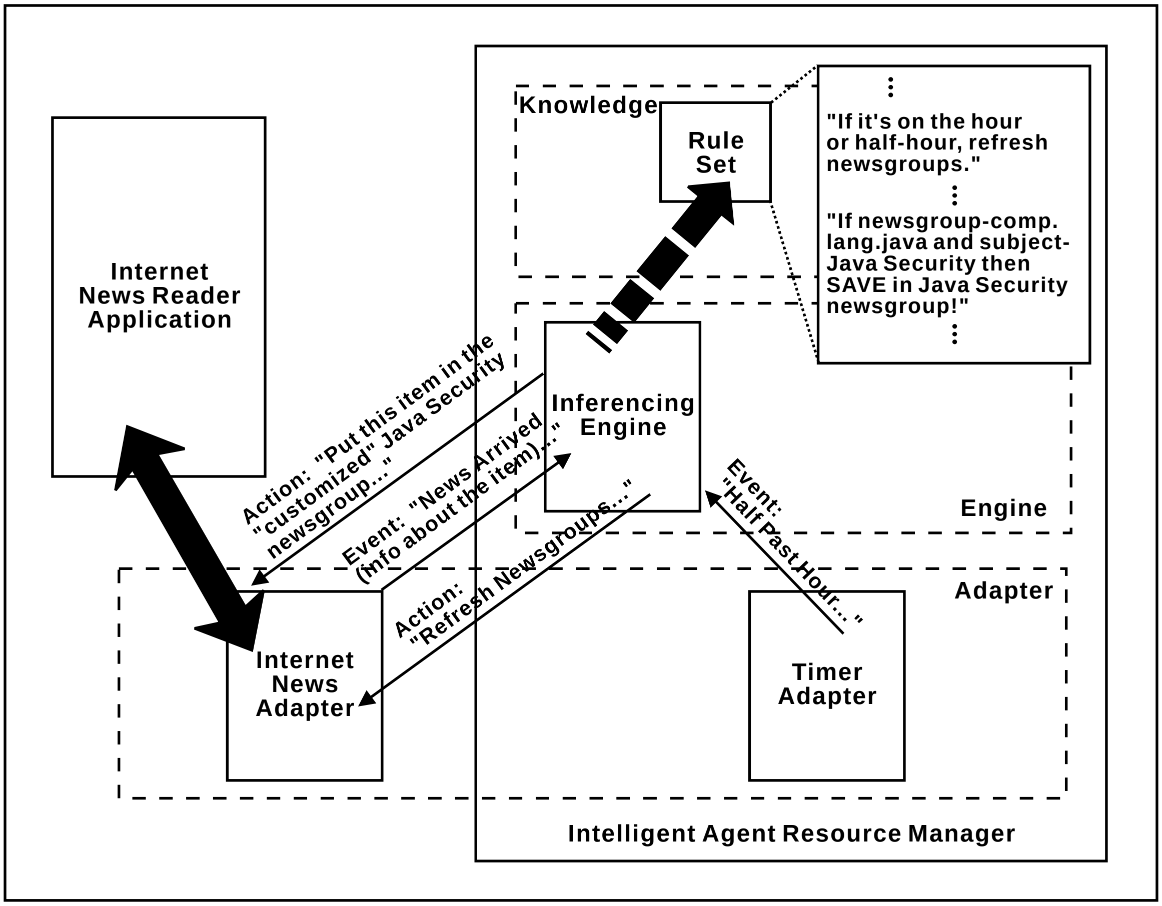

Internet News Reader Application
Knowledge
Rule Set
"If it's on the hour or half-hour, refresh newsgroups."
"If newsgroup-comp. lang.java and subject-Java Security then SAVE in Java Security newsgroup!"
Inferencing Engine
Engine
Action: "Put this item in the "customized" Java Security newsgroup..."
Event: "News Arrived (info about the item)..."
Action: "Refresh Newsgroups..."
Event: "Half Past Hour..."
Adapter
Internet News Adapter
Timer Adapter
Intelligent Agent Resource Manager

Figure 4. Internet News Reader Application

Distributed Intelligent Agent Support

In some cases, the agent-enabled application operates on a client system, and the Intelligent Agent resource manager operates on a different server system. The application-specific adapter in one system and the Intelligent Agent resource manager in another system can use the Object Request Broker resource manager on each platform to interact. Agent-enabled applications can also exploit the distributed environment.

Relationship To Other Open Blueprint Resource Managers

The Intelligent Agent resource manager uses several other Open Blueprint services. The library sub-component, which stores various forms of knowledge, uses the File resource manager. The library sub-component also uses the Identification and Authentication resource manager and uses the Access Control resource manager to control access to various forms of knowledge, and to prevent unauthorized modifications.

The views sub-component uses Open Blueprint Presentation Services to enable the user to view and modify various forms of knowledge. Views also interacts with the library sub-component and Open Blueprint Security resource managers.

Adapters can use the Open Blueprint resource managers that provide the function they need. For example, a relational database system adapter could use the Relational Database resource manager to enable an agent to detect events in or to manipulate a relational database.

In another example, a telephony system adapter could use the Telephony resource manager to enable an agent to call users when a particularly urgent piece of mail arrives.

A third example is a dialog adapter that can interact with a user. The dialog adapter might be a simple pop-up dialog box, or as sophisticated as a "talking head" whose lips move in synchrony with computer generated speech, and whose facial expressions (happy, sad, perplexed) indicate whether the agent is able to meet its goals. The dialog adapter would use Open Blueprint Presentation Services.

The Intelligent Agent resource manager also interfaces to the Open Blueprint Local Operating System Services to access functions such as memory management and process start-up and to the Systems Management backplane to allow the Intelligent Agent resource manager to be managed.

Appendix A. Future Extensions

Intelligent agent technology is changing rapidly. This section describes IBM's current thoughts on how the Intelligent Agent resource manager might be extended to accommodate higher levels of agent function.

This paper describes the first level of agent technology, that is, intelligent, stationary, non-interacting agents. The next level of agent function, *agent interaction*, will support more advanced applications. However, this area is not mature, and there are several different approaches to agent interaction that are being explored by the agent research community.

Using agent interaction, a user could subscribe to a new Internet news group. When a news article for that newsgroup arrives, the agent would be informed. If the rules governing the operation of the agent are comprehensive enough, the agent can determine what to do with the article. But this is a new newsgroup, and suppose that the article contains a previously-unencountered subject line? What can the agent do in response to this event?

The agent could ask the user what to do with the article. However, a more proactive agent could *ask other Internet news agents* that represent other users with similar professions, backgrounds, and interests, how they might handle news articles on this subject. Perhaps some of them have subscribed to this newsgroup in the past and have some experience and advice they can share.

Several important questions in the area of agent interaction are being resolved by agent research. For example, there are at least two ways agents can interact. One way is for Agent A to invoke methods on Agent B, and vice versa. If this approach to agent-to-agent interaction eventually emerges as the preferred approach, the existing CORBA ORB capabilities specified in the Open Blueprint can provide the required support.

An alternative way for Agent A to interact with Agent B is to exchange messages in some high-level declarative language that expresses definitions, assumptions, beliefs, and so on. A leading candidate in this area is the Knowledge Query Manipulation Language/Knowledge Interchange Format (KQML/KIF) work that is part of the Defense Advanced Research Project Agency (DARPA) Knowledge Sharing Effort. If this approach were to emerge as the preferred approach to agent interaction, IBM's current thinking is to add agent interaction adapter function to the Intelligent Agent resource manager to support a function protocol between Intelligent Agent resource managers instances. The function protocol would enable an Intelligent Agent resource manager to *ask* other Intelligent Agent resource managers questions, to *tell* them facts and rules, to request that they *perform* certain actions, and to *subscribe* to certain events. (Ask, tell, perform, and subscribe are all KQML performatives).

As the preferred mode of agent interaction emerges, additional services will be provided by the Intelligent Agent resource manager. For example, agents that can interact will need to be able to advertise their existence, and to look each other up (using a Yellow Pages type of function). When two agents want to interact, they might want to authenticate each other, and (if the mode of interaction is message passing) to cryptographically protect their message exchange to prevent spying or tampering. Agents representing *servers* or *sellers* will want to charge agents representing *clients* or *buyers*, so some electronic commerce services will be needed.

These services need not be implemented by the Intelligent Agent resource manager. Agent authentication would be based on Open Blueprint Security Services. The agent "Yellow Pages" will be based on the Open Blueprint Directory Services, which include the Object Naming Services. The Intelligent Agent resource manager might provide a higher level of agent-specific abstraction upon these services, however.

Mobile agents are another very active area, both in the research community and in various standards bodies. IBM has conducted research and, based on that research, IBM has submitted a proposal for a mobile agent facility standard to the Object Management Group (OMG). This mobile agent facility would provide the capability to:

- Support the dynamic registration of various agent language interpreters such as Java, Telescript, and TCL
- Flatten mobile agents, transport them through the network (encrypting them if secrecy is needed), and reconstitute them at their destination
- Authenticate mobile agents that arrive, and cryptographically detect any tampering
- Monitor the execution of mobile agents for several purposes:
  - billing
  - preventing malicious or poorly-programmed agents from using all the system resources
  - logging
- Provide status to the originator of a mobile agent when it terminates (normally or abnormally)

IBM is working with other leaders in the intelligent agent field to develop and promote standards that will enable agents to operate across systems from different vendors. As these standards emerge and mature, they will be incorporated into the Intelligent Agent resource manager.

Appendix B. Adapter Interface to the Intelligent Agent Resource Manager

This appendix describes how an adapter communicates with the Intelligent Agent resource manager. In the IBM implementation of the resource manager, both the adapter and the Intelligent Agent resource manager are objects, so the two communicate using method invocations. (The information in this appendix is subject to change as the Intelligent Agent resource manager evolves.)

Agent Binding and Instantiation

The Intelligent Agent resource manager reads a configuration file to associate the adapter and engine that comprise the agent and starts up the components. For each adapter, the resource manager calls the identify() method of the adapter object, asking the adapter to register all its sensor and effector functions. The adapter registers each function it provides by invoking its registerProcedure() call once for each function.[1] Each function is identified by a token defined by the adapter writer. When all the components have been started and all the adapter's have registered their functions, the Intelligent Agent resource manager calls the start() method on each adapter, telling the adapter that it can start generating events.

Agent Runtime

An adapter can generate an event and pass it to the resource manager to be processed by invoking its notify() function. The event is a parameter on the notify() function. The information contained in the event is described in “Events” on page 12.

In response to an event, the engine in the Intelligent Agent resource manager can request more information (sensing), or it can initiate some action (effecting). For sensing, the resource manager can invoke either the testCondition() method or the provideFacts() method on the adapter. The first method, as its name implies, is boolean in nature - it requests the adapter to test some condition in its domain of expertise, and to return TRUE or FALSE.

The second method asks the adapter to gather some information within its domain and return it to the engine in the resource manager. The token passed in the method call identifies which condition to test or what information to gather. These are the same tokens that the adapter registered with the Intelligent Agent resource manager at agent start time using the registerProcedure() calls.

For effecting, the Intelligent Agent resource manager calls the performAction() method of the adapter object. The call to performAction() specifies a token that tells the adapter what action to perform. These are the same tokens that the adapter registered with the resource manager at agent startup time using the registerProcedure() calls.

The Intelligent Agent resource manager calls the eventComplete() method on the adapter when all processing of a previously-generated event is complete. This tells the adapter that no further sensing or effecting will occur for the event, and that it can discard any data associated with the event.

[1] . Methods that are used to communicate with the Intelligent Agent resource manager such as registerProcedure() and notify() are actually *adapter* methods that are inherited from the adapter base class. The low-level details and mechanics of interacting with the Intelligent Agent resource manager are hidden from the adapter writer.

Agent Termination

When the Intelligent Agent resource manager decides to terminate an agent, it invokes the stop() method on all adapters. This tells an adapter to stop generating new events (but says nothing whether the resource manager can send the adapter additional performAction(), testCondition(), or provideFacts() calls). Eventually, the resource manager calls the adapter's shutdown() method, which tells the adapter to cleanup and terminate.

Events

An events (passed from the adapter to the engine in the Intelligent Agent resource manager) has two parts - a standard part which is called the event header, and an event-specific part which is called the event body.

The event header contains information that is required for all events. For example, the domain of the event (news, Web, and so on), the type of the event ("news arrived", "Web page changed"), a timestamp, and a unique event ID are all carried in the event header.

The contents of the event body depend on the specific event. In the case of the "Web page changed" event, the event body could contain information such as URL of the page that changed, the value of various HTML tags associated with the page such as title, content type, content length, date of modification, and so on. In the case of a "news arrived" event, the event body could contain information such as the name of the newsgroup that an article belongs to.

Appendix C. Notices

References in this publication to IBM products, programs, or services do not imply that IBM intends to make these available in all countries in which IBM operates. Any reference to an IBM product, program, or service is not intended to state or imply that only IBM's product, program, or service may be used. Subject to IBM's valid intellectual property or other legally protectable rights, any functionally equivalent product, program, or service may be used instead of the IBM product, program, or service. Evaluation and verification of operation in conjunction with other products, except those expressly designated by IBM, is the user's responsibility.

IBM may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
500 Columbus Avenue
Thornwood, NY 10594
USA

Trademarks

The following terms are trademarks of the IBM Corporation in the United States or other countries or both:

IBM
IBMLink
Open Blueprint

The following terms are trademarks of other companies:

CORBA	Object Management Group, Incorporated
Java	Sun Microsystems, Incorporated
LotusScript	Lotus Development Corporation

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Sun Microsystems, Inc., in the U.S. and other countries.

Appendix D. Communicating Your Comments to IBM

If you especially like or dislike anything about this paper, please use one of the methods listed below to send your comments to IBM. Whichever method you choose, make sure you send your name, address, and telephone number if you would like a reply. Feel free to comment on specific error or omissions, accuracy, organization, subject matter, or completeness of this paper.

When you send comments to IBM, you grant IBM a nonexclusive right to use or distribute your comments in any way it believes appropriate without incurring any obligation to you.

- If you prefer to send comments by FAX, use this number:

  United States and Canada: 1-800-227-5088.

- If you prefer to send comments electronically, use one of these ID's:
  - Internet: **USIB2HPD@VNET.IBM.COM**
  - IBM Mail Exchange: **USIB2HPD at IBMMAIL**
  - IBMLink: **CIBMORCF at RALVM13**

Make sure to include the following in your note:

- Title of this paper
- Page number or topic to which your comment applies

Printed in the United States of America
on recycled paper containing 10%
recovered post-consumer fiber.

G325-6592-00