Learn Sesame - a Learning Agent Engine

Alper Caglayan, Magnús Snorrason, Jennifer Jacoby,

James Mazzu, Robin Jones & Krishna Kumar

Charles River Analytics

55 Wheeler Street

Cambridge, Massachusetts, 02138 U.S.A.

akc@cra.com, http://www.opensesame.com

Abstract

Open Sesame!® 1.0—released in 1993—was the world's first commercial user interface (UI) learning agent. The development of this agent involved a number of decisions about basic design issues that had not been previously addressed, including the expected types of agent and the preferred form and frequency of interaction. In the two years after shipping Open Sesame! 1.0, we have compiled a rich database of customer feedback. Many of our design choices have been validated by the general approval of our customers while some were not received as favorably. Thanks to the overwhelming amount of feedback, we were able to substantially improve the design for Open Sesame! 2.0, and develop a cross-platform learning engine - Learn Sesame- that can be used to add learning agent functionality to any third party application. In this paper, we present a summary of the lessons learned from customer feedback, an outline of resulting design changes, the details of the developed learning agent engine and planned research.

1. Background on User Interface Learning Agents

In our user interface learning agent paradigm, a learning agent sits in the background and observes user actions, finds repetitive patterns, and automates them upon approval. The learning agent paradigm uses the metaphor of a personal assistant because it is responsible for facilitating user tasks (such as opening and closing documents, performing desktop maintenance, creating aliases, etc.).

Our learning agent acquires its knowledge from the user by a hybrid neural expert architecture where neural networks perform knowledge acquisition, and expert system techniques perform inference and knowledge maintenance. Although other methods for learning information patterns do exist, we feel that hybrid neural expert system architectures are the best choice for UI learning agents. They take advantage of each technology's best features: neural networks add robust unsupervised learning capability to a hybrid system, while a knowledge based expert system interprets the neural network's results (Mazzu, Allen & Caglayan, 1991, Liebowitz, 1993).

In addition to their learning capability, learning agents have other features, advantages and benefits as described in Table 1.

Table 1: Features and Advantages of Learning Agent Technology

Feature	Advantage	Benefit
Learning	Identify tasks for automation	Reduced workload
Automation	Perform repetitive tasks	Increased productivity
Customization	Automatically set user's software preferences	Enhanced user experience
Tutoring	Provide in-context coaching	Reduced training costs

Learning agents can provide a quantum leap in user experience by customizing software to the user. Learning agents are particularly applicable when (Maes & Kozierok, 1993):

- The application domain contains a significant amount of repetitive behavior,
- The repetitive behavior is different for different users,
- The environment supports the monitoring of user events and actuation of user commands.

The human-agent interface is an important consideration in a learning agent design since human-computer interaction is inherently social as evidenced by individual emotions such as trust, anger, annoyance and forgiveness towards machines and applications (Nass, Steuer & Tauber, 1993). Based on research involving human-human interaction, findings from the social sciences reveal factors affecting human-agent interaction including:

- *Agent name* - an agent’s name (such as “personal assistant”) implies that the agent possesses certain abilities and characteristics (Laurel, 1990)
- *Agent form* - a form that is too human-like (for example, an animated face, or human-sounding natural language) may build up the user’s expectations of the agent in such a way that cannot possibly be fulfilled (Norman & Long, 1994)
- *Communication skills* - the agent’s language ability, the frequency with which it interrupts the user, and the language that it employs all play a significant role in the degree to which the agent is liked (Nass et al., 1993)
- *Trust* - trust must be built over time and the agent should never take expensive actions without permission (Norman & Long, 1994)
- *Believability* - the user’s degree of belief in what the agent says is another characteristic that needs to be built up over time

2. Open Sesame! 1.0

Open Sesame! is the first desktop agent that learns to automate the routine tasks of a user. Open Sesame! has been reviewed and covered in trade and mainstream publications worldwide from the New York Times to the Japanese edition of PC Week. Open Sesame! has been shipping since December 1993, and is distributed worldwide. Currently, we are shipping version 1.1 of

Open Sesame! which is specifically optimized for MacOS on PowerPC. (This version is also available for downloading from our Web page, http://www.opensesame.com)

Table 2 defines the terms required to discuss the interaction between the user and Open Sesame! without ambiguity:

Table 2: Open Sesame! UI Elements

Term	Definition
Instruction	A set of directions to the agent on how to carry out a task
Confirmation	A request for user's approval before the agent carries out an instruction
Observation	A behavioral pattern learned by the agent
Suggestion	An agent recommendation to the user

Open Sesame! version 1.0 watches for two kinds of tasks: *time-based* and *event-based*. A time-based task is something that the user does at a particular time. For example, opening electronic mail every day at nine o'clock is a time-based task. An event-based task is something that the user does in relation to another task. For example, opening the clock desk accessory before logging into an on-line database is an event-based task.

Open Sesame! is based on the hybrid neural expert architecture (Mazzu, Snorrason & Caglayan, 1994) shown in Figure 1.

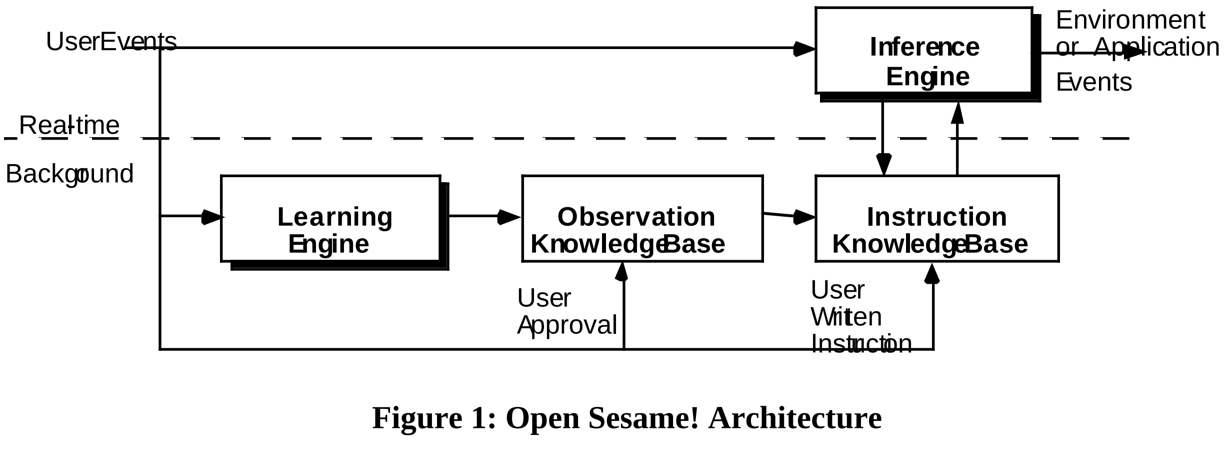
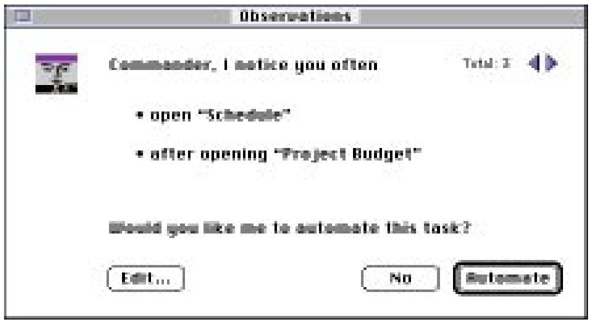


Figure 1: Open Sesame! Architecture

Open Sesame! compares the high level **User Events** (such as opening a folder or quitting an application) generated by the user's mouse clicks and keystrokes to information stored in its neural **Learning Engine** *in the background* and to information stored in its **Inference Engine** *in real-time*. The neural learning engine looks for repetitive patterns that haven't been automated. If it finds one, then Open Sesame! adds an observation to the **Observation Knowledge Base**, such as the one shown in figure 2.



Commander, I notice you often

• open "Schedule"

• after opening "Project Budget"

Would you like me to automate this task?

Figure 2: Observation Dialog Box

If the user gives her/his **Approval**, then Open Sesame! creates an *instruction* to automate the task that it observed and adds it to the **Instruction Knowledge Base**. The **Inference Engine** compares monitored **User Events** with patterns from the **Instruction Knowledge Base**, representing existing instructions. When it finds a match, the inference engine automatically sends a set of **Environment** or **Application Events** to the operating system to perform the instruction.

The user can either give the agent direct **Written Instructions** or accept the agent's offer of automation for an observed user work pattern. When the agent informs the user of an observation, the user has the following options:

- accept the observation and have the agent create an instruction
- decline the observation
- edit the observation to fine-tune the instruction

- postpone a decision until later

Open Sesame! 1.0 can learn the following patterns: opening and closing folders and documents, opening and quitting applications, hiding and showing applications, arranging application windows, and performing routine desktop maintenance.

In further detail, the Open Sesame! 1.0 learning engine architecture is based on both Macintosh domain specific knowledge and domain independent learning services. The domain specific knowledge is modeled using the object-oriented principles of Objects, Attributes and Events in addition to the learned Patterns and corresponding Rules (as shown in table 3).

Table 3: Learning Engine Knowledge Elements

Term	Definition
Objects	That which the user or system interacts with (documents, applications,..)
Attributes	Properties whose values define the state of an object (size, creation date,...)
Events	That which causes an object to change its state (open, close, shutdown...)
Patterns	Learned sequence of events that represent user behavior
Rules	If-Then statements which determine the Open Sesame! agent behavior

As the user interacts with the Macintosh, the monitored events are used as input to the learning engine services, which produce patterns to be interpreted. The learning engine interprets the patterns according to predefined interpretation knowledge and produces observations when the patterns represent useful information. The learning engine also provides validation services which manage new observations and existing rules. These learning engine services are summarized in table 4 below:

Table 3: Learning Engine Services

Term	Description
Pattern Recognition	Identifies event-based and time-based patterns from monitored objects/events using a neural network paradigm
Pattern Interpretation	Determines whether the recognized pattern is significant and decides how to provide the information to the user, possibly as an observation
Validation	Insures the uniqueness of learned observations and user-generated instructions
Rule Inferencing	Compares monitored events and objects to the instructions in the knowledge base, and executes any triggered instructions

The pattern recognition in Open Sesame! uses a neural network paradigm called GEN-ART (Snorrason & Caglayan, 1994; Caglayan & Snorrason, 1993), based on Adaptive Resonance Theory (Carpenter, Grossberg & Rosen, 1991) to categorize the high level events it monitors. One advantage of ART type networks is their provision for self-organized learning, which does not require prior knowledge of event-pattern categories. This is accomplished via bottom-up competitive filtering of patterns for finding the “best match” category, combined with top-down template matching for determining if the best match is "good enough." GEN-ART allows the use of customized distance metrics to handle input patterns with qualitative (nominal or ordinal) variables.

Because of Open Sesame's! predefined knowledge of its environment, *suggestions* can also be made about tasks that can not be learned from user behavior. For example, Apple recommends rebuilding the Macintosh Desktop once a month or so, hence Open Sesame! will offer to perform this task for the user. The user has the same options as with observations: to accept, decline, edit, or postpone.

3. Lessons Learned From Customer Feedback

Open Sesame! was announced at MacWorld Boston, 3 August 1993. Since its debut, we have profiled our customer base on a regular basis. Figure 3 depicts a breakdown of our customer base by Macintosh skill level. The breakdown has not changed significantly since the product began shipping.

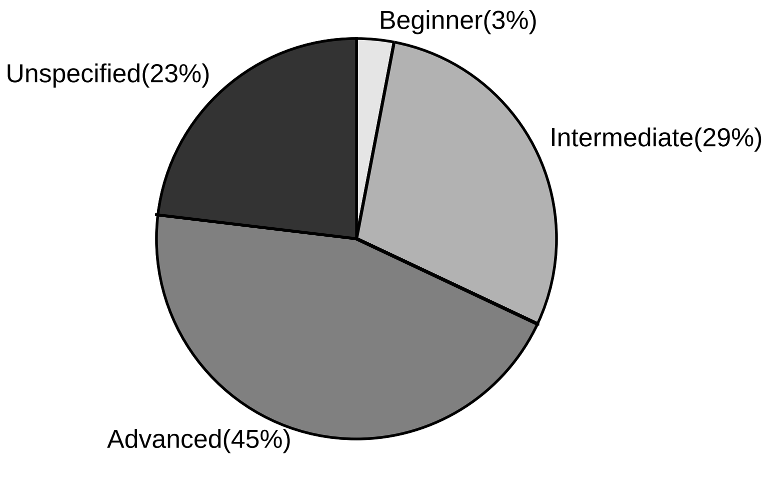


Figure 3: Breakdown of user base by skill level

Our market research reveals that the large percentage of advanced users can be attributed to our marketing emphasis on the novelty and "high-tech" nature of agent software. Open Sesame! was unlike any other product on the market, and we took the approach of educating the public about learning agent technology at numerous trade shows and user group presentations. According to our findings, Open Sesame! users are interested in agents that can increase productivity, function as a smart assistant, and retrieve information.

The lack of novice users reflects a real problem facing developers of commercial agent products. Two possible reasons for this asymmetry are intimidation by the concept of the agent "entity" and the need for a clear relationship with existing product categories. We have started the battle of increasing public awareness of what UI learning agents can do for the average user, but our data indicates that the appeal is still heavily skewed towards the expert user.

We have received formal and informal feedback from customers, researchers, user groups, and the software trade press about Open Sesame!. The feedback has been both positive and negative and it sheds light on ways in which user interface agents can be improved. Feedback is summarized in Table 3 (comments about Open Sesame!'s strengths are preceded by "+" and comments about its weaknesses are preceded by a "–").

Table 3: Summary of customer feedback

Topic	Definition	Feedback
Agent's form	The agent's appearance, its "persona"	– Needs more colorful and graphical interface – Needs more use of sound for feedback + Users like the ability to change its voice
Agent's communi-cation skills	The manner in which the agent communicates with the user	– Let the user customize agent language – Use fewer technical phrases – Use more natural sounding language – Allow option to communicate by sound, by dialog box, or by sound and dialog box + Users like the use of voice
Agent's task specific skills	The types of events that are monitored and actions that can be performed	– Too few types of actions – Some actions too limited in scope – Does not do enough for advanced users + Users like the automation of maintenance tasks that are easily forgotten
Agent's social skills	The "politeness" of the agent: when it interrupts, how frequently it interrupts	– Some users felt interruptions should happen less often – Others felt interruptions should happen more often (Open Sesame! talks more often when it is first installed) – Need control over timing of interruptions (i.e., not while logged into an expensive on-line service) – Interruptions should be prioritized + Users like the ability to defer reviewing observations
User Interface (UI)	The interface through which users access the agent's preferences	– Power users want access to more preferences + Users like the pull-down menu and sentence structure of instructions + Non-programmers like the fact that they do not have to learn a scripting language

Learning	Time-based and event-based learning	– Time-based observations, although correctly deduced, frequently are not useful – Should not offer to automate patterns performed by another application – Should not offer observations about topics the user has already rejected + Users like the novelty and usefulness of event-based learning

Most users like the idea of a personal agent that learns repetitive actions and makes the user more productive. The concept is intriguing and users look forward to giving Open Sesame! a try. Some users become disappointed by the product because they assume it can learn everything they do. When their agent does not learn a pattern, the user thinks there must be something wrong with it. What the user does not realize is that Open Sesame! has only a limited number of tasks and patterns that it can learn. After using Open Sesame! for a few months, we have seen that users either accept its limitations and continue to use and enjoy the product, or they become discouraged and stop using the product—they decide to wait until a more feature-laden version becomes available.

Our data also reveals that users who have highly customized computer work environments tend to like the product less. Open Sesame! 1.0 made a number of assumptions about the usage and organization of the "Apple Menu" and desktop: it offered to "manage" these areas for the user. We learned that without knowing the user's work habits and organizational preferences, it was presumptive of Open Sesame! to assume that all users work the same way. Ideally, a user interface agent should be flexible and should adapt to all work and organizational styles. It should also be a good citizen to the user and other applications in the environment and should not force the user to follow any one work style.

Users were in general agreement that time-based learning was not very useful. One of the problems is that time-based learning is often linked to other key causal and conditional events. When Open Sesame! presents time-based learning suggestions without the context of these other events, it looks less intelligent. For example, an observation such as:

I notice you often open your AppleCD Audio Player at 11:15 AM.

is meaningless without knowing whether a CD is mounted in the CD-ROM drive. A more appropriate observation would be:

I notice you open your AppleCD Audio Player when an audio CD is mounted.

In summary, we learned that time-based learning is brittle without a more complete picture of other events and conditions which are present at the same time.

One of the main lessons learned from Open Sesame! 1.0 is that while users approach any new software program with different skill levels, interaction preferences, work and organizational styles and assumptions, these issues are especially important to note in the case of user interface agents because of the social nature of the interaction between the software and the user. When a UI agent interrupts the user at the wrong time or makes a less than helpful suggestion, the mistake is more pronounced because the agent is more than *just* an application: because of their social nature, agents are subject to a more “human” level of criticism.

The above feedback represents a summary of the lessons learned from Open Sesame! 1.0. In the next section we describe how this feedback influenced the design of Open Sesame! 2.0.

4. Open Sesame! 2.0

Based on the feedback above and research in user experience, social interfaces, and intelligent agents, we have dramatically improved the design for Open Sesame! 2.0.

Table 4 summarizes the most significant design changes and the expected user benefits of those changes.

Table 4: Summary of design changes

Topic	Design Change	Expected Benefit
Agent’s form	Added user-selectable character options, graphical representations and sounds	Users can give their agents a personality, with more color and sound
Agent’s communication skills	Added ability to control interaction content (text before, during, or after performing tasks, and in case of errors)	Accommodate users who require significant control over the agent
	Allow option to communicate only by sound, not by dialog	Accommodate users who do not like visual feedback
	Let the user customize agent language and use fewer technical phrases	More natural sounding language
Agent’s task specific skills	Five times more types of events and actuations	Increased monitoring and actuation abilities make OS! act more intelligently, hence more attractive to advanced users
	Added some capability to monitor and perform tasks internal to 3rd party applications	
	Added the ability to specify conditions on instructions	
	Added in-context coaching	Teach novice users how to use features they have not yet discovered
Agent’s social skills	Added the ability to directly specify the observation frequency from the agent	Satisfies both users who wanted more frequent and less frequent interactions
	Added a “do not disturb” event	Users gain control over timing of interruptions
User Interface	Switched to multi-pane windows	Allows more preferences without cluttered windows
	Removed various options for customizing what types of events get learned	Simplify the interface by removing preferences no one used
Learning	More emphasis on event-based and less on time-based learning	More intelligent observations
	Switched from batch to incremental processing of data	Allows real-time processing of many more event types without slowing down other applications

In direct response to advanced user requests, one of the main enhancements in Open Sesame! 2.0 is its expanded skill set. Version 2.0 monitors and actuates more than five times the type of events, and one hundred times the number of events than version 1.0.

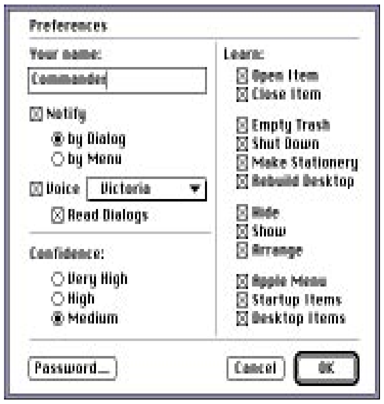
In addition to new events, Open Sesame! 2.0 also provides support for conditions in the instruction syntax. In order to counteract the time-based learning effects encountered in the previous version and give more flexibility to advanced users, version 2.0 offers the ability to narrow the scope of instructions by specifying the states under which instructions can fire.

A useful side effect gained from monitoring so many events and conditions is the ability to offer predefined tips to the user, especially novice users. In-context coaching functionality was added specifically to attract novice users, in response to the asymmetric skill level data discussed in the previous section. This feature allows Open Sesame! to coach the user about tips and tricks based on the user's current activity. Open Sesame! 2.0 is being bundled with a coach that teaches about the Macintosh operating system.

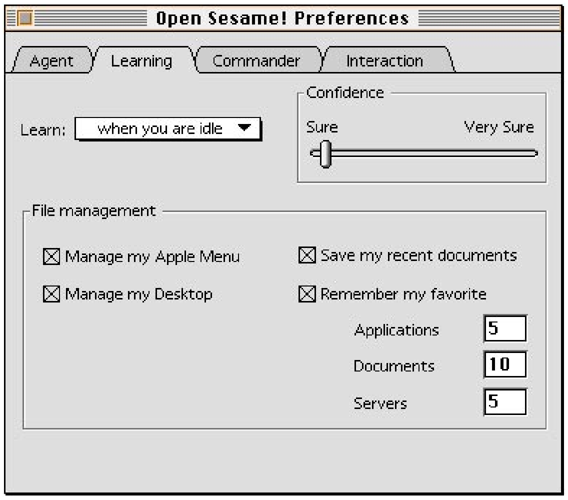
Open Sesame! 2.0 also includes a larger predefined knowledge base in order to provide more immediate benefits to the user. This came about from user feedback on redundant observations: we found that we can accelerate the agent's learning process by incorporating the knowledge learned from version 1.0 into the knowledge base of version 2.0. In essence, Open Sesame! 2.0 will holds the collective knowledge of all version 1.0 agents within its initial knowledge base.

Although this explosion in functionality complicates the simple user interface that version 1.0 boasted, initial feedback from advanced beta testers is positive. Figure 4 shows the user interface for setting preferences in version 1.0. There are more preference settings in version 2.0, but by using a multi-pane window interface the clutter is kept to a minimum, as can be seen in figure 5.

Comparing the two figures also shows that most of the options for customizing which types of events get learned were removed. In version 1.0, these options acted as an indirect control for observation frequency: if the user felt that too many “Open Item” observations were being generated, deselecting the “Open Item” learn option would stop similar observations from appearing in the future. Interestingly, very few users utilized this feature. Instead they asked for more control over the *overall* observation frequency. In version 2.0, through the Interaction pane in the Preferences application, the user is given greater control over agent interaction on both an *individual* instruction basis and an *overall* interaction basis.



Your name: Commander
☒ Notify
[✓ by Dialog] by Menu
☒ Voice: Victoria
☒ Read Dialogs
Confidence: Very High High [✓ Medium]
Learn:
☒ Open Item
☒ Close Item
☒ Empty Trash
☒ Shut Down
☒ Make Stationery
☒ Rebuild Desktop
☒ Hide
☒ Show
☒ Arrange
☒ Apple Menu
☒ Startup Items
☒ Desktop Items



Learn: when you are idle
☒ Manage my Apple Menu
☒ Manage my Desktop
☒ Save my recent documents
☒ Remember my favorite
Applications: 5
Documents: 10
Servers: 5

Figure 4: Preference dialog from Open Sesame! version 1.0

Figure 5: Preference dialog from Open Sesame! version 2.0

5. Learn Sesame - Learning Agent Engine

One of the lessons learned from our Open Sesame! product effort was the importance of third-party applications’ cooperation in providing monitoring and actuation support to a learning agent. After more than a decade of personal computing, it is important to note that the mainstream PC operating systems do not yet incorporate operating system services that would enable personalization such as user preferences, user model, or user work organization style. In response to this need, we developed a cross-platform learning engine - Learn Sesame - that can be embedded into any application such as desktop applications, Web servers, OLAP engines, etc. to provide learning agent functionality.

The learning engine, called Learn Sesame, provides a framework for modeling and learning sequences of events in a dynamically changing environment. A client application presents the engine with a continuous stream of time-stamped events that describe occurrences in the environment and the states of the objects affected by these events. The engine analyzes this stream of events, and incrementally identifies recurrent sequential patterns in the event stream. The engine outputs patterns, called facts, which are passed back to the client for domain specific interpretation. A fact contains information that will enable the client application to convert it into an if-then rule when such a conversion makes sense for the domain being monitored.

The learning framework itself is domain independent, and includes a language that provides a simple, structured vocabulary for clients to describe the domain objects and events. The application designer uses this language to construct a domain model that encodes client specific notions of comparability, similarity and proximity of objects and events. Using this model, the client converts application specific data into an event stream, and interprets the facts output by the engine. The engine uses the model to identify clusters of similar sequences of events in the event stream, to analyze the clusters, and to construct facts describing the similarities found between the sequences in the clusters. The clustering and cluster analysis algorithms used by the engine incorporate all the notions of similarity and proximity defined by the domain model. Thus the client application has considerable control over the learning process. Figure 6 shows the overall relationship between a client and the engine.

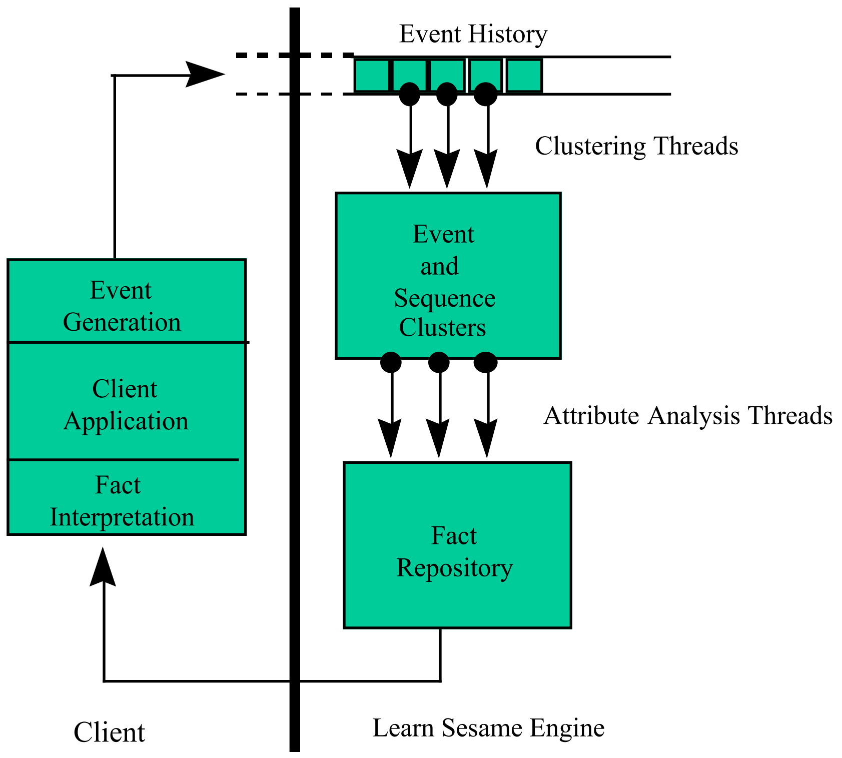


Figure 6: Relationship between a client and the Learn Sesame engine.

The following are the main features of the Learn Sesame engine. A more detailed description of the concepts and architecture of the learning engine can be found on the products page at the Open Sesame Web site (Kumar, 1996).

1. *Domain independence*: The domain model is explicitly defined by the client application. The engine works in the realm of abstract events and objects whose domain specific properties are encoded in the model.

2. *Efficient incremental learning*: The engine operates in an incremental manner: events and sequences are clustered and analyzed as they are inserted and in a multi-threaded environment such as Windows NT, the processes described in figure 6 can all be carried out concurrently in independent threads.

3. *Configurable clustering algorithms*: The model definition language and the clustering algorithms operate in conjunction to allow domain specific control of the learning process. This allows the application designer to control what kind of facts will be learned.

4. *Encompasses functionality of attribute based learning algorithms*: Sequence learning is proper generalization of traditional attribute based learning — the special case of learning sequences of length one provides functionality fully equivalent to traditional attribute based clustering algorithms.

5. *Robust, scaleable implementation*: The architecture of the engine uses efficient database techniques and a threading model that fully exploits the inherent concurrency in the learning process. The architecture is designed to scale smoothly to multi-processor platforms with robust preemptive threading, while providing full functionality (albeit, at some cost in performance) in single threaded or cooperatively threaded environments. The engine can also be configured to account for available time and memory constraints of the application at hand.

The incorporation of the Learn Sesame learning engine into a third-party application involves the development of a monitoring knowledge base and actuation components for the selected application. Developing an application monitor is analogous to factoring an application to make it scriptable or recordable. This process involves:

- identifying an event model for the application
- determining an object model encapsulating the user action, application state and environment events of significance for each event
- implementing the event object model and notification code

In general, the event dictionary is an appropriate subset of the user actions that the application user interface allows such as menu commands, dialog buttons and controls, and so on. To integrate Learn Sesame into an application, the domain is first modeled as a collection of objects, attributes and events. An object represents a real-world entity such as a document on the desktop or an HTML page on a Web server. Attributes specify the structure and properties of an object. An event is modeled by the change in the value of an object attribute.

An object type definition defines a category of modeled objects for the domain. This definition specifies the possible attributes of the object and establishes the criteria for distinguishing between distinct objects of the given object type. For example, the following defines a document type:

ObjectType *Document*

{

Attributes:

DocumentName: TypeName;

DocumentCreator: TypeApplication;

DocumentSize: TypeFloat

CreationDate: TypeDate;

Forms:

FormName: {DocumentCreator};

}

A form is defined as a named subset of the possible attributes of a given object type that is used in controlling the comparability of objects in the domain. Forms enable multiple views of the same object for clustering purposes. Essentially, object types define which objects can be compared, and object forms define the criteria for object similarity. An object instance represents a snapshot of an object in the domain. Every object instance has unique object type and form identifier associated with it and defines a value for each of the attribute names in that form. Two object instances are said to have the same *identity* if they have the same object type and form identifier, and agree on the values of all the attributes in the form.

Occurrences in the domain that affect domain objects are called events. An event has zero or more targets, each of which is an object instance, and one or more named attributes called the parameters of the event. The target of an event specifies an object whose attribute value changes due to this event. An event type defines a category of related events. For example, the following defines the print document event in Learn Sesame.

EventType *PrintDocument*

{

TargetTypes:

Document;

Parameters:

Application: TypeApplication;

Printer: TypePrinter;

};

The definition above sets up an event type whose targets are of type Document defined earlier, and has two parameters the application, an attribute of type *TypeApplication,* and the printer, an attribute of *TypePrinter.*

The Learn Sesame engine processes the stream of events to find recurrent patterns. The event model described is used to define the meaning of comparability, similarity, and proximity of events in the application domain. Comparability is defined by object type, similarity is defined by object form, and proximity is defined by the attribute metrics dependent on the domain. The learning engine uses this model to identify clusters of events, analyze the clusters, and generate facts describing the similarity found between the clusters. In general, the clusters can be composed of event sequences in which case the engine finds recurrent sequential patterns such as repetitive navigation patterns of a user at a Web site . When specialized to the case of sequences of length one, the engine can find events with common attribute values such as the set of users with common information interests at a Web site.

For each new event (e.g. print Excel document, click on link Products on the Home Page of a Web site), the engine forms sequences by concatenating the event to existing sub-sequences in the event history. The maximum and minimum lengths of sequences constructed are configurable parameters of the engine.

For each new sequence, the engine identifies the unique collection of sequences with the same identity as the sequence and inserts the new sequence into this collection to form clusters (e.g. all excel documents printed in printer Money, all users who clicked on Products from .edu domain). The unique collection of sequences that have the same identity as a given sequence is called the primary cluster for the sequence.

So a primary cluster is the set of all sequences derived from the event history that share the same identity. The use of the term cluster in this context is consistent with traditional statistical clustering, where a cluster usually denotes a set of vectors in a metric space that are close to each other according to a distance metric.

The clustering in the learning engine incorporates both bottom-up and top-down techniques in the formation and analysis of clusters. The primary clusters are produced bottom-up by grouping all sequences with several attributes in common by virtue of their identity (e.g. all Word documents created this year, all users who clicked on a particular page from a .com domain). The engine then groups primary clusters according to a configurable set of criteria, into a hierarchy of clusters called secondary clusters (e.g. all documents created this year or all users from any domain who clicked on a particular page in a Web site) as shown in figure 7.

Secondary clusters are formed by taking the union of primary clusters or other secondary clusters.

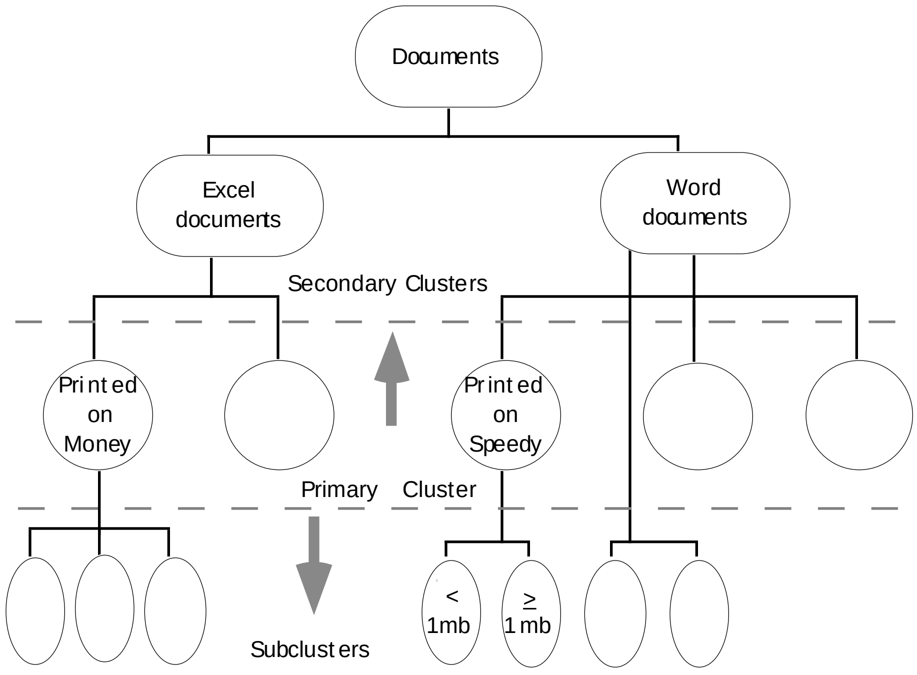


Figure 7 Primary and Secondary Clusters

The criteria for the formation of primary and secondary clusters are structural - the primary level groups sequences together by identity, and secondary clusters may group sequences together by the object and events types of the events in a sequence. The form construct enables a measurement of proximity between events. For example, if a floating point value appears as the value of a parameter of an event, the model designer could “force” events with different values for this attribute to belong to the same primary cluster as an attribute of an object with a null form. Then, sequences containing event instances with any value for this parameter will be forced into the same primary cluster. Similarly, when primary clusters are grouped together into secondary clusters the criteria used are typically the types of the events, and the presence or absence of attributes rather than the proximity of attribute values.

Figure 8 shows the event processing in the engine. When a cluster (e.g. all Word Documents with size > 1 Mb. all Word documents with size <1Mb), whether primary or secondary, is analyzed, it may be broken down into sub-clusters based on the proximity of events according to the metric defined in the domain model. In general, the facts produced by analyzing the sub-clusters are more specific than those obtained by analyzing different clusters.

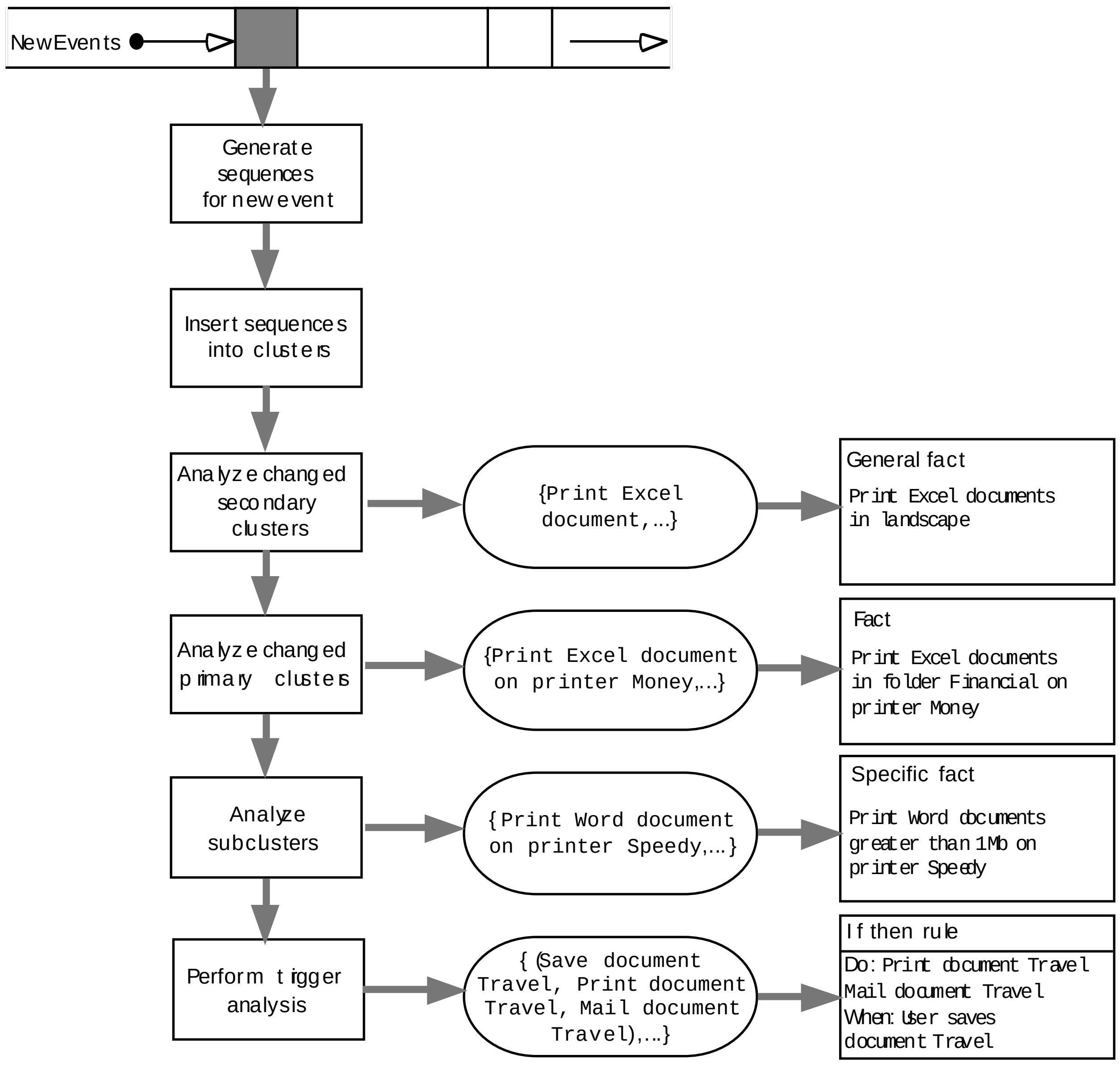

NewEvents
Generate sequences for new event
Insert sequences into clusters
Analyze changed secondary clusters
{Print Excel document,...}
General fact
Print Excel documents in landscape
Analyze changed primary clusters
{Print Excel document on printer Money,...}
Fact
Print Excel documents in folder Financial on printer Money
Analyze subclusters
{Print Word document on printer Speedy,...}
Specific fact
Print Word documents greater than 1Mb on printer Speedy
Perform trigger analysis
{(Save document Travel, Print document Travel, Mail document Travel),...}
If then rule
Do: Print document Travel Mail document Travel
When: User saves document Travel

Figure 8 Event Processing in the Learning Engine

The output of the learning engine is a fact. A fact is a logical expression that states a common set of properties shared by sequences in a cluster. For example, a fact could signify the identities of the targets of two events are the same which implies that the value of the form attributes of these targets agree. On the other hand, not all the attributes need to be in agreement. The non-form attributes on targets and object valued parameters may vary quite widely between the sequences. The properties shared in common by the sequence is described by a sequence pattern, which may be viewed as a boolean expression on the event attributes of events in the sequence.

In the cluster analysis phase, a cluster with a given base pattern is analyzed to produce a fact. The base pattern is guaranteed to satisfy the property that every sequence in the cluster satisfies the pattern. The fact output by the analysis will consist of

- A sequence pattern is satisfied by a predetermined percentage of sequences in the cluster
- A trigger (if any)

Figure 9 shows the cluster analysis steps. Triggers are used to convert a fact derived from a cluster into an if-then rule. The trigger (e.g. Select All event in a Word application) of a cluster (e.g. the set of sequences (Select All, Copy) in a Word application) is determined by considering a complementary cluster. A complementary cluster, consists of all possible sequences in the event history which do not belong to the given cluster but also satisfy this prefix of the pattern in the analyzed cluster (e.g. the set of all other sequences that start with the Select Allevent: {(Select All, Cut), (Select All, Clear)}. If the ratio of the complementary cluster size divided by the analyzed cluster size is smaller than a predetermined fixed percentage, is unique according to a predetermined criterion, then the prefix is used predictive trigger for the rest of the sequence pattern. Thus, the trigger is used convert the fact into an if-then form where the if part is the set of conditions in the prefix and the then part is the set of conditions in the rest of the pattern. Depending upon the domain, the then part can be interpreted as an action to be taken, or a conclusion to be made, or can be used as a rule in a rule-based system.

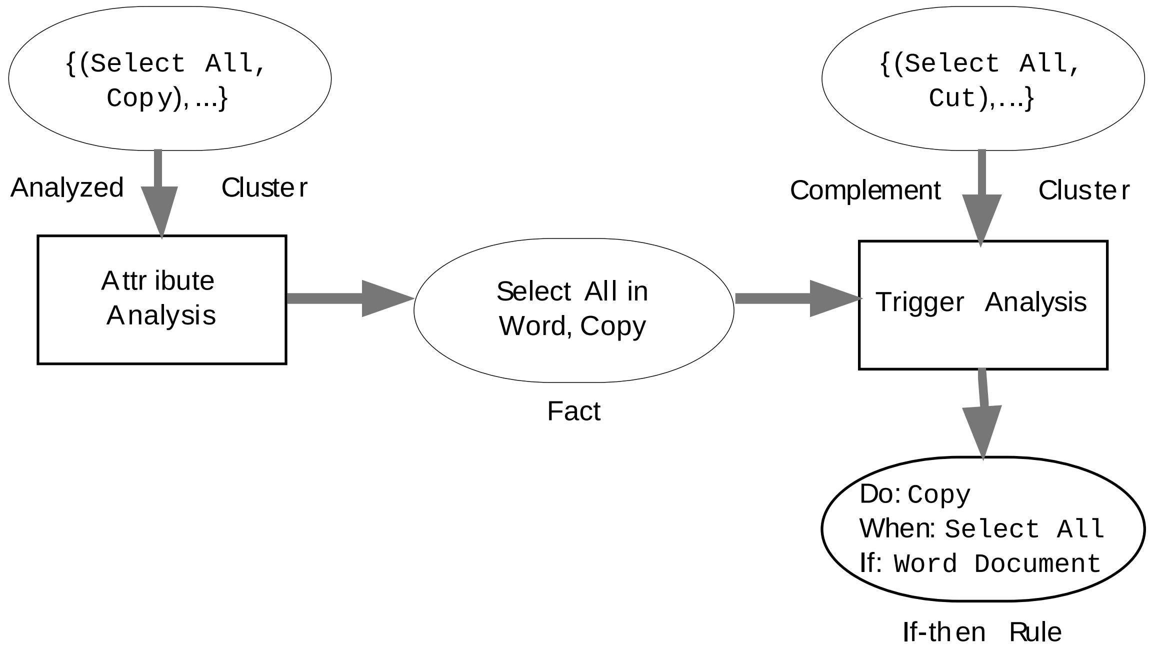


Figure 9 Cluster Analysis Steps

During attribute analysis the learning engine examines the sequences in a cluster on an attribute by attribute basis. The values of a given attribute are compared to determine if a suitable attribute template can be generated for the attribute type definition. User defined distance functions and comparison operators are used during this process. The order in which the engine learns the values for the attribute is summarized below.

- Learn an exact value for the attribute:
    - If the attribute behavior is Crisp, then an attribute template is learned if a predetermined percentage of the attributes have the same value.
    - If the attribute behavior is Fuzzy, the distance and comparison functions are used for learning. A fuzzy values is learned if a predetermined percentage of the values the distance to the median is less than fixed threshold.
- If an exact value cannot be learned then the engine tries to learn a range using the specified distance functions and range widths.

- If an exact value or range for an attribute cannot be learned, then the engine tries to learn an object template such that a fixed percentage of the object values satisfy the template. This process recursively tries to learn the values of the attributes of the objects, attribute by attribute.

The facts generated by the learning engine are interpreted by predefined domain dependent rules. The main objective of fact interpretation is to validate, maintain, and decide on the course of action for facts. The validation typically involves filtering out trivial facts (similar to ignoring 'the', 'a', etc. in document filtering) and those determined inappropriate for the domain (e.g. repeated user mistakes such as {(copy document A, delete copy of document A),...}. In addition, the fact interpretation involves knowledge maintenance functions.

The development of a knowledge base inference mechanism for an inference agent consists of:

- encapsulating application domain knowledge into rules
- determining rules for validating agent instructions
- figuring out rules that specify the execution schedule

The selection of an application to incorporate the functionality of the learning engine requires a user study. Such a study should evaluate the tradeoffs between:

- *In-house vs. third-party application*
- *Real-time vs. non-real-time application*
- *Large vs. small event dictionary*
- *Hourly vs. daily vs. weekly vs. monthly patterns*
- *High vs. low frequency repetitive user behavior*
- *Individualized vs. homogeneous repetitive behavior*

- *Multi-user vs. single user platform*

The user study should attempt to determine the degree of fit of the application to an interface agent metaphor. For instance, to determine the potential return on the deployment of a mail agent, it is advisable to determine the average number of hours spent on reading and filing mail messages, and the number of repetitive steps performed in filing mail messages.

6. Conclusions and Areas for Future Research

A wise business person once said, “Customers never tell you about problems you do not already know. What customers do tell you is how to prioritize your problems.” While this is certainly true for issues such as Open Sesame! 1.0’s task specific abilities (it came as no surprise that customers wanted to be able to automate practically everything), it was more interesting to witness the level of disagreement between users with varying skill levels and organizational styles on issues like the desired frequency and timing of interaction.

Such conflicting feedback led us to the conclusion that despite our intentions to make Open Sesame! as simple as possible to use, the user must still be given a fair amount of control over the agent’s social and communication skills. Giving users the ability to directly control agent interaction *and* monitor a significant number of new events increased the functionality and intelligence of the product at the cost of increased complexity, size, and time to market.

An important area that we have addressed only partly in version 2.0 is the user’s work style. It is clear that the optimum form of social and communication skills for an agent is a function of the user’s level of expertise. However, it also seems that knowing the user’s level of expertise is not sufficient; factors like the user’s style of file management are also important.

Finally, our Open Sesame 2.0 effort taught us that agents as stand-alone entities must wait for the development of infrastructures that support such agents. Currently, the best way to incorporate agent functionality into an application is to integrate a learning engine into the application with the active cooperation of the application developer. Such integration casn enable third-party applications provide automation, customize user preferences and provide in-context coaching. One of the more promising applications of learning agents is the customization of Web sites based on the learned preferences of users. The use of our learning engine on Web servers to provide customized content delivery is one of our current research projects.

References

Caglayan, A. K., & Snorrason, M. (1993). "On the Relationship Between the Generalized Equality Classifier and ART2 Neural Networks". *World Congress on Neural Networks*, Portland, OR.

Carpenter, G. A., Grossberg, S., & Rosen, D. B. (1991). "ART2-A: An Adaptive Resonance Algorithm for Rapid Category Learning and Recognition". *Neural Networks*, 4, 493-504.

Kumar, K. (1996). "Smarts by Open Sesame! - Conceptual Overview". Charles River Analytics technical report, Cambridge MA. A copy can be downloaded from www.opensesame.com /products/smarts/smartswhitepaper.html.

Laurel, B. (1990). *The Art of Human-Computer Interface Design*. Reading, MA: Addison-Wesley Publishing Company.

Liebowitz, J. (1993). "Roll Your Own Hybrids". *BYTE*, July.

Maes, P., & Kozierok, R. (1993). "Learning Interface Agents". *Int. Workshop on Intelligent User Interfaces*, Orlando, FL.

Mazzu, J. M., M. S. Snorrason & A. K. Caglayan (1994). "A Hybrid Intelligence Engine for Nuclear Monitoring", Department of Energy Contract No. DE-FG02-ER81178.

Mazzu, J. M., Allen, S. M., & Caglayan, A. K. (1991). "Neural Network/Knowledge Based Systems for Smart Structures". *Materials and Adaptive Structures Conf.*, Alexandria, VA.

Nass, C. I., Steuer, J. S., & Tauber, E. (1993). "Computers are Social Actors". *Proceedings of the CHI Conference*, Boston, MA.

Norman, T. J., & Long, D. (1994). *Goal Creation in Motivated Agents*. London, UK: University College London

Snorrason, M., & Caglayan, A. K. (1994). “Generalized ART2 Algorithms”. *World Congress on Neural Networks*., San Diego, CA.