# DYNAMIC APPLICATION SYSTEMS AND PROCESSES FOR DISTRIBUTED COMPUTER ENVIRONMENT

**Related Applications:** This application claims priority pursuant to 35 U.S.C. §119 of provisional application Ser. No. 60/126,928, filed on March 29, 1999, the entire contents of which are hereby incorporated by reference.

**Field of the Invention:** The present invention relates to dynamic application systems and particularly to dynamic application systems implementing behaviorally oriented object programming.

**Background of the Invention:** Presently, software applications are essentially lines of instructions coded to create specified features and behaviors. These applications have difficulty in adapting to the rapid changes in technology and tend to be relatively cumbersome in using system resources. Typically, traditional applications attempt to provide all features for all users, despite the fact that few users require all of the included features.

A prime example of these applications are in the field of information technology management. Information technology, the field of obtaining, managing and utilizing digital information, has become increasingly a vital and essential resource for businesses and other professional arenas. Information is now viewed as an important and abundant resource that must be handled as other resources. The raw data must be obtained, it must be stored and it must be processed into a usable form.

Before the advent of a global network, i.e., the Internet, managing information resources was relatively simple. Typically, data sources for such information consisted of static reports, fact sheets, or databases that were updated periodically. This data was normally maintained on a local area network. Now, with the interconnection of hundreds of millions of homes and businesses, with hundreds of thousands of data sources immediately available and with these data sources potentially changing every day, the old information management tools are not adequate. The presence of raw data in these multitude of data sources is useless if the data is inaccessible to potential users.

There is a desperate need for products and services that will make such information usable, that will gather the data in an efficient manner and process this data in a form that is accessible and informative to users. Often, even if raw data is accessible and gathered, the voluminous amounts of the raw data may keep the critical data hidden from the user.

Even more critical is the need for products and services that are able to work in an environment where the content, format, and access protocols associated with information resources change daily. Traditional applications for information gathering, and for most other purposes, consist of large numbers of lines of instructions that are bound together to create a set of specified features, behaviors and user interfaces that define the applications. These programs normally require a large expenditure of programming resources to implement even small changes in the applications. These applications are not able to quickly and efficiently adapt to the frequent and ongoing changes occurring in the formats and protocols in the data sources or other application environments. Even where an application may be adaptable, it requires the skill of a programmer proficient in that particular application to make the necessary changes.

The traditional applications normally are designed to include all features and behaviors that may or may not be required by each individual users. Thus, traditional applications typically are very large in size. Additionally, traditional applications are normally bound to a particular hardware platform, operating system or application domain. Thus the applications normally must be substantially revised for use on other platforms, operating systems or application domains.

There is presently a need for an efficient, reliable system for the collection, storage and processing of data from the multitude of data sources available, not only now but in increasing amounts in the future.

There is presently a need for a dynamic system that can be utilized in various environments and that can be easily implemented as required by a particular user.

There is presently a need for such a system that can be transparently implemented and upgraded so to be usable by non-programmers and that can verify itself of changing formats and protocols.

Another critical problem with traditional software programs is their inability to adapt or "learn" from their environment. Most programs, including those developed in object-oriented languages are goal-oriented. The objects are assigned a specific task to be conducted in a specified manner. The instantiated objects normally are expecting a specific message to proceed and the failure to receive such a message may result in the application breaking.

Thus, there is a need for a system and processes that allows the objects to be influenced by the environment to adapt to achieve the goals of the application.

There is presently a need for such systems, not only in the field of information gathering but also in other application environments.

## Summary of the Invention

The present invention accomplishes these needs by providing dynamic application systems that, in a preferred embodiment, utilize behaviorally-oriented objects to adapt to the operating environment of the applications. The dynamic applications only instantiate components necessary to perform the functionality required by the user.

The dynamic client application of the preferred embodiment of the present invention are able to be operated with a minimal operating system. These operating system can include handheld computers, PDAs, appliances and the like. The dynamic client application utilizes a "seed" program that includes minimal executable code to create a connection with a server, download a set of system components from the server and install those components to create a user interface. The seed application can go into a dormant phase at this point.

The user can then request desired functionality from the system. The client application will then download the components necessary for the client application to either perform the functionality or to access this functionality elsewhere in a distributed computer system.

The seed program, in a preferred embodiment, initially requests an identification from the user of the client computer. This identification is passed on to the server which determines the profile of the user assigned to that identification, the privileges and access allowed to that user, and downloads components for system components that the user is authorized to use. The profile

may also include such information as group membership indicating the name or identity of any user-groups to which the user belongs. If the user does not possess an identification, then the user is logged as a "guest" and a base level of components are downloaded and instantiated.

In the preferred embodiment, these system components include a component loader and an application, such as an XML (eXtensible Markup Language) parser, to determine the initial application structure and the attributes that are to be assigned to each instance of the components of the initial application structure and to verify other components. As the user requests additional functionality, the client application will make access requests to the server and download selected components based upon the user supplied information.

The components are combined with data from the user at runtime to provide application-specific functionality. Thus, complex sets of behavior can be created for each component while still being generic for application for many tasks.

In the preferred embodiment, the components include behaviorally object-oriented components. In particular, the components are able to provide rich messaging between components or objects. Prior objects, in traditional object-oriented systems, use static interfaces that require prior understanding of specific parameters of the objects interfaces before messaging between the objects can understand. Further, prior objects are created to expect certain responses from the objects with which they interact. Failure to receive or return expected responses causes the applications to break or fail.

The present invention, in a preferred embodiment, provides behavior-oriented objects. The components or instantiated objects are loosely coupled to one another to allow effective communication between the objects even when the interface of either or both objects changes substantially. The particular implementation of the preferred embodiment uses a component interface that provides a two stage interface. The first stage allows an object to perform a simple communication or "tap" of a second object. This tap includes identification of the first object and information as to the method of response to the first object. The second object responds by providing information as to the entry points of the second object. The first object then sends a rich message to

the second objects through those entry points. These behaviorally oriented objects are also designed not to expect messages but to receive them if they are "tapped". As a result, each object becomes dynamically modified to generate messages to any other object, or to no object at all.

Another feature of the behaviorally oriented objects is the capability to "grade" it's ability to implement useful functionality with it's programmed behavior. As the functionality of the component is used, or if the user is "pleased" with the performance of the component, the grade of the component is increased. If the user is displeased with the component or if the functionality is not utilized over time, the grade is decreased. Other components may use the grade of the component to determine whether to send messages to that component. The component may even be "encouraged" to mutate to more useful behavior by this capability. Useful functions will encourage the component to use those behaviors while unused behaviors will be dropped. The component may even be killed off if it's score drops to a certain level.

In a preferred embodiment of a specific implementation of the present invention, the client application becomes an information management tool. The seed application downloads components from the server to allow it to instantiate the components into a user interface which communicates to other servers and/or data miners. The user can also access a premium content database on the server, if the user's profile allows this access, to obtain preprocessed information. The user can access preferred portals that have specialized data miners to speed the search process. The data miners are collections of software objects that each apply sets of rules and/or strategies to determine if data contained at a data site fit the criteria. These miners employ fuzzy logic to determine if the data fits the criteria, and may seek verification from the user as well. The user may also seek information from preferred services, such as airline, train, and other travel sites, merchandising sites and other sites requiring transactions. The user will also have access to general or non-preferred data portals using non-specialized data miners.

An important feature of the preferred embodiment of the present invention is the ability to constantly track and verify changes in formats, protocols and access to the different sites. The present invention tracks and verifies these changes and upgrades the system and client application components

to maintain access to these sites without intervention of the user. Thus. even non-programmers will be able to utilize the present invention.

Additional features of the present invention include the incorporation of Personal Information Mangers. This enables personal information about the user to be incorporated into the client applications without the user having to frequently reenter this information.

Another feature is the data storehouse on the server. The user is able to store collected information for access anywhere. The user does not have to access the same client computer each time. The user is able to gain access to the collected information from any computer having communication with the server by simply inputting the user identification.

While access to the server is critical to maintaining the most current system components, the client application will cache components on the client computer to enable the user to utilize the client application when the connection between the client computer and server is broken. This allows laptop computers, handheld devices, appliances, even cellular telephones to function as the client computer.

These and other features will be evident from the ensuing detailed description of preferred embodiments and the drawings.

## BRIEF DESCRIPTION OF THE DRAWINGS

FIG. 1 illustrates an exemplary network system and processes in which the present invention is implemented;

FIG. 2 shows elements of an exemplary application environment in accordance with the present invention;

FIG. 3 shows a component of the application environment of FIG. 2 in greater detail;

FIG. 4 through FIG. 6 illustrate a particular implementation of the system and processes of FIG. 2 at various stages of implementation; and

FIG. 7 shows an application description document in accordance with a particular implementation of the present invention.

FIG. 8 shows an XML file declaring the constituent components of an exemplary component in accordance with a particular implementation of the present invention.

## DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENTS

The present invention provides a system and processes for creating dynamic relations between individual entities, such as bits of information spread across an infinite number of nodes connected together in networks. In the preferred descriptive embodiment, an information gathering application is described as an example of the usefulness of this system and processes.

It is to be expressly understood that the descriptive embodiments set forth herein are intended for explanatory purposes and is not intended to unduly limit the scope of the claimed inventions. Other embodiments and applications not described herein are considered to be within the scope of the claimed inventions.

The system and processes of a preferred embodiment of the present invention utilizes Dynamic Application architecture to create behaviorally oriented programs operating in a client/server relationship. A Dynamic Application exists as an unbounded set of components working together to form a software solution. Every Dynamic Application begins with a starting point, such as a "Seed". This seed creates the initial environment for the Dynamic Application.

One example of a very simple Dynamic Application is the World Wide Web. The seed for the World Wide Web would be the browser that provides access for a user to the World Wide Web. Browsers, however, are typically hybrids of old static architecture (using hard coded features) and new dynamic architecture. Thus, browsers typically contain a relatively fixed set of features and are quite large, requiring extensive resources in order to operate.

The system and processes of the present invention operates quite differently than the above-described browsers. The system and processes provides a completely dynamic seed. The seed of the preferred embodiment of the present invention acts as a "client" having no particular features of it's own. Instead, it requests from a server (or multiple servers) such features as necessary for it to perform a requested function. In the preferred embodiment, the system and processes is made up of the following components:

the Seed;

network connection (such as the Internet, other global network services, LAN, WAN, MAN, or other networks);

Membership Identification;

Server ( comprising a repository, database, and storehouse);

Client;

Preferred portals;

Preferred Services;

External Repositories;

Local Personal Information Managers (PIMs).

**The Seed**

The Seed, as described above, is a completely dynamic kernel. In the preferred embodiment, the Seed is approximately 75K bytes. This allows it to function on even the smallest handheld device or appliance. The first time that the Seed is run on a computer, it requests the user for disk space and the user's Membership Identification (described below). If the user does not yet have a Membership Identification, then the Seed will grant the user a limited guest membership. A specific implementation of the Seed is described in greater detail below.

**Membership Identification**

Users have a Membership Identification assigned to them. The Membership Identification provides a particular set of features to the user. For instance, different levels of services may be accessible to the user depending on their Membership Identification. These different levels of services may be available for fees, or assigned to the user on a need to access basis.

**Server**

The Server provides the functionality for the system and processes of the preferred embodiment of the present invention. The Server provides the connections, the hardware and powers the system and provides access to all of the features and membership privileges. The system and processes allows the Client (described below) to be disconnected from the Server, such as with a handheld device or appliance, a laptop computer, etc. for an indeterminate amount of time. The device then must rely on information that was cached prior to disconnection and on features and components previously obtained during

connection to the network. The Server, in the preferred descriptive embodiment, has three major components (in addition to the hardware and network connection): the Feature repository, the Premium Content Database and the User Storehouse.

The Feature repository contains all the components required to enable a particular capability or feature (e.g., dynamic link library (DLL) files, data files and the like). It is contemplated that all features will evolve over time and so the Feature repository will evolve correspondingly. Thus, a reliable connection to the Server is necessary for the Client to remain fresh and vital. The Feature repository may be implemented using any available data store technology and may be accessed through a directory service or file system mechanism.

The Premium Content Database contains preprocessed data. This database is option, but is discussed herein as having particular utility in an Information Management application used as an exemplary embodiment. It is to be expressly understood that the present invention is not limited to this exemplary embodiment. In some cases it may be more efficient for the Server to provide access to such preprocessed data rather than requiring each Client to obtain and process this data itself. An example of such preprocessed data contained in the Premium Content Database would be customer marketing lists. Preprocessing may also be used to add value to the data by coordinating, arranging, and collecting disparate data.

The User Storehouse is used to store information that belongs to a particular user that has requested such a facility. The Storehouse allows a user to store information independent from a particular computer. Thus, a user is able to move from computer to computer without concern as to the location of stored data. A user needs only a computer, a network connection to the Server and a Membership Identification.

Client

The Client, in the preferred embodiment of the present invention, is an "organic" user interaction application. The Client begins with the Seed. As the Seed obtains the initial information from the user, such as the Membership Identification, the Seed then connects to the Server to download sufficient

components from the Server to begin the formation of a user interface. Then as the user begins to interact with the system through this user interface, additional components and forms are downloaded and cached as required. For example, once the Seed obtains a valid Membership Identification, then it will request from the Server the components for a user interface, such as a browser. The Seed/Server may also recognize that the particular Membership Identification is allowed or prefers a particular user interface. Also, the Seed/Server may recognize that the user associated with that Membership Identification requires a particular language, i.e., U.S. English, Japanese, French, etc.

The Client is organic in that it will constantly change as necessary depending on the function requested of it. For example, the Client may be used as a data search application to acquire data on a particular subject. The Client may then be used to parse that data or to associate that data with word processing, layout editor, or other applications. As the user requests a change in the functionality of the Client, additional components and/or forms may be downloaded from the Repository of the Server and previously downloaded components may be discarded.

A specific implementation of the Client is discussed in further detail below.

**Preferred Data Portals**

In the descriptive embodiment, the system and processes of the present invention has particular application to obtaining data from various sources. In particular, the present invention is able to efficiently perform data mining functions. Data mining is the extraction of data of particular interest from raw sources. Raw data is processed by collections of software objects ("Data Miners") which are organized to apply simple rules and strategies to the data. The Data Miners use fuzzy logic to do their tasks. This includes locating information that is "probably, likely, or possibly" something they recognize. In the preferred embodiment, the Data Miners also use "hints" that identify information that closely match certain criteria. Once a potential piece of information has been identified, the Miner "asks" a question to verify the information. A probability is assigned to such information which is applied to

other pieces of information. Information that has the highest probability is then stored and low probability information is discarded.

The system and processes of the present invention is able to mine most if not all data repositories or "Data Portals". However, a certain number of these repositories are regarded, in the descriptive embodiment as "Preferred Data Portals". These Preferred Data Portals include special Data Miners that know the specific formats and other details of these repositories. These special Data Miners perform fast and reliable extraction of data from their particular portal. Also, as the portals change the formats of the data contained therein, the Server will periodically validate the formats of the portals. This validation will typically occur in the background so the user is unaware of this operation.

**Preferred Services**

Often the user will request the Client to perform tasks on behalf of the user, such as searching for and booking travel arrangements, purchasing merchandise, and other operations requiring financial or time sensitive transactions. The system and processes of the present invention provides specific data miners able to interact with sites providing such services. Since the form of such transactions frequently change or updated, the data miners associated with these services are tested frequently. Validation and updating of the data miners is performed without no action required by the user.

**External Data Repositories**

External data repositories are defined as any node that is not classified as a Preferred Data Portal or Preferred Services. Since these repositories do not have any special data miner associated with them, more general strategies are used for data extraction. In the preferred embodiment, once a data miner has become assigned to mine a particular site, the miner will "evolve" quickly or be given such a low priority (as discussed in further detail below) to be effectively killed off. It should be understood that Preferred Data Portal and Preferred Services are desirable but optional. Although less efficient, generic components can usually access the Preferred Data Portals and Preferred Services. It is contemplated that when Preferred Data Portals and Preferred Services are

inoperable or unsuccessful for any reason, generic components will automatically fill in.

**Local Personal Information Manager (PIM)**

Personal Information Manager (PIM) products, such as Microsoft's Outlook, are useful to maintain and import personal information on the user to other applications. PIMs require more specific solutions than are required by the Preferred Data Portals and Preferred Services. Typically such products require specific components developed for that particular product. Integration with such services may be handled by including special purpose pluggable interfaces such as an outlet component. Often these components will include procedural code rather than a behavior-oriented components. This procedural code is specific to a particular PIM and interfaces with an API provided by the PIM. The present invention does not encourage, but can accommodate such specialized procedural code in a conventional object-oriented manner. Installation of these components is similar to the installation of behavior-oriented program components and occurs, automatic in a manner that is transparent to users.

**Implementation of Client Components**

The Client, discussed generally above, is a dynamic software application comprising a plurality of weakly linked components. The components comprise instantiated objects and are preferably implemented as hierarchical "container" objects (e.g., object that contain other objects). Since the client application is completely dynamic in the preferred implementation, each instance of the client will contain a different set of components and will vary from time-to-time as components are changed and updated. The Client application as it exists at any instant in time is the entry point for all user interaction with the system in accordance with the present invention.

In a particular implementation, the Seed comprises a minimal set of executable code from which the entirety of the client application is spawned. As initially installed, Seed may be the only component of the Client. The role of Seed is akin to a kernel in an operating system, the Seed represents code that

boots up, makes essential links to other devices and software, and executes a routine to enable more fully functional operation. Because the Seed is implemented as procedural code, the Seed preferably includes as little functionality as possible so that it does not need to be changed often, and will be robust in light of operating system changes and the like.

For example, the Seed can implement a command line or dialog box type interface to get rudimentary user information such as name, account number, password and the like and makes a link to the Server to downloaded enough components for the formation of a user interface. The user interface enables the Client to begin interaction with a new member. Other components that would be loaded upon initialization might include components for accessing a mass storage device such as a datastore or for establishing and operating a cache structure. As the Client begins interacting with server, new components and data are downloaded and cached as required. Once initialized, the Seed becomes dormant and the Client is controlled by application components.

In a particular initialization procedure, the Seed passes user ID and password information to the Repository Server which returns a user identification, a "profile", an initial code library, and a start-up XML file. The user identification is a unique code or number assigned by the Server to that particular user ID and is used throughout the system to uniquely identify the user.

The profile contains user-specific information such as a list of code or modules available to the user (based upon permissions or privileges determined from the user ID), a list of components that should be loaded, group membership information, and the like. The initial code library will include code for performing basic system-type functions such a system object for interacting with the operating system and performing system-type behaviors such task scheduling. Other components include a component loader for loading new components and installing them in software application at runtime, and a component for parsing XML documents.

Once an XML parsing component is installed, the startup XML document can be read to determine the initial application structure. The preferred implementation uses a declarative application construction method in which application components are declared dynamically at runtime. An XML document is conveniently used to make this declaration. XML is preferred

because it is rigidly formatted and highly transportable over existing public communication links such as the Internet. The startup XML document essentially contains a list of components that are to be included in the Client application along with attributes that are to be assigned to each instance of the component.

Although the components describe above provide system-type functionality such as user interface implementation, XML parsing, class loading, and the like, additional components are downloaded to implement application-specific functionality. In a broad sense, any type of functionality can be implemented by selection of components from feature repository including text processing, spreadsheet, internet browsing and the like. In the particular implementations, the Client serves as an information tool and so will selectively download components that enable the Client to specify desired information, retrieve the user-specified information, and present the user-specified information in a user-specified manner.

Because the Client does not have any prior knowledge of what information will be requested or where the information will reside, components are generally "generic" components that define rudimentary behaviors, but do not have enough structure as downloaded to implement any particular functionality. In other words, a single component may include behaviors for accessing a web site at a uniform resource locator (URL) and returning an HTML document, but as downloaded does not specify which URL or know anything about the content of the HTML document. This component's behavior is customized at runtime to provide application-specific functionality by providing the component with data specifying a URL, for example. This distinction between generic behavior and specific functionality is important to a complete understanding of the present invention.

Components are preferably complex combinations of other components and so provide more complex functionality than, for example, just accessing a web site and returning the HTML response page. Hence, although components are generic in that they are not data store specific, the particular combination of components contained within any other component makes each component an individual. For example, a component that is individualized for address retrieval may include a component for accessing a company web

page, another object for identifying a zip code in the returned HTML page, another object selecting text from the returned page surrounding the zip code, and yet another object for formatting the selected text into tab-delimited address information. These complex sets of behavior make each component highly functional but still generic as they could be applied to any company web page.

It is important to note that although components comprise generic elements and behavior-oriented objects, once a component is instantiated and given a specific task, its functionality becomes application specific. A particular instance of a component can be saved and reused to implement the specific functionality of its component objects. A particular instance is not, however, sub-classed or extended to provide other functionality. Once instantiated, a component will "know", for example, a specific URL to access and will know specific information to select from the return HTML page. As described below, the component is preferably given means to evolve so as to improve its performance or it be given such a low priority as to be effectively killed off.

An important feature of the present invention is that components comprise a set of instantiated behavior-oriented objects. In other words, components are not extensions, sub-classes, or over-ridden versions of objects or classes stored on server. Instead, they are instantiated directly from classes, objects, or object descriptions on the Server. Components are high-level objects that implement a dynamic interface supporting rich messaging. Only after an instantiated object receives a message can it respond to perform application-specific functionality. Components are almost purely behavior. Behavior given context results in application-specific functionality.

FIG. 3 illustrates a simplified component 209 configured to select data from a URL. The component comprises a set of other components, including thread component 301, a URL component 302, a data port component 303, a page grabber component 304, and a select text component 305. Each component 301-305 implements a more primitive behavior than does the container component. Thread component 301 serves to manage message passing within component. In general, any message passed to component is broadcast to all of its sub components 301-305, and any message generated by

any sub-component 301-305 is broadcast to all other sub-components, but thread component 301 may alter that general message passing.

**Behavior Object-Oriented Components**

The above-described embodiment can be implemented in presently available object-oriented programming paradigms. One difficulty in prior object-oriented systems is that objects, while implementing behavior, encapsulate conventional procedural code. This results in static interfaces such that an object requires specific parameters to be passed through the interface before the object can function. To establish an interaction between two objects both objects must have an a priori understanding of the other's interfaces. This requirement forces programmers to rigidly define the interfaces. However, such rigid definition leads to many of the same problems as conventional (non object oriented) procedural code. Specifically, when the interfaces change, the application breaks. Also, objects are created to expect certain responses from the objects with which they interact. If an object fails to return an expected response, the application breaks.

Objects receive context in two manners. Upon instantiation, attributes can be assigned to the object. Additionally, messages passed to the object at runtime provide context. Together these context mechanisms give the component, and all components that are contained therein, the information needed to determine their functionality. Their functionality will be different if they are instantiated within a different parent, or if they are passed a different message.

The present invention provides a unique environment for providing dynamic object interfaces is an important feature of the components to make them "behavior-oriented". Hence, instantiated objects within components are "loosely coupled" to each other. This means that an object's interface can change substantially or entirely and still effectively communicate with other objects. The particular implementation uses an component interface architecture that provides a two stage interface. In a first stage, a component "taps" another component with which it wishes to communicate. The tapped component responds with a message or message object that indicates its entry points. The first component can then use the newly gained knowledge of the second

component's entry points to send a rich message. In this manner, the only procedural interface that needs to be hard coded into both components is the simple tap interface. The dashed-line interconnects in FIG. 3 are used to suggest this loose coupling mechanism.

A tap message preferably includes the requesting object's name or address (so the return message will reach the correct object). The requesting object also implements an interface sufficient to receive the response message as a parameter and procedural logic sufficient to use the response message to implement the rich interface. The tap interface is so simple that it can be used without modification among any number and variety of components without regard to their fundamental behavior. The more complex rich interface is determined dynamically at runtime. Because the rich interface can be implemented in a very context-specific manner, the messaging between components can be richer than is practical with conventional object-oriented designs.

Object-oriented designs often require that a particular object receive a message for the application to proceed. If the message does not come, the application can break. Behavior oriented design relies on each object to generate messages to appropriate objects, but do not specify what those appropriate objects are. That decision is left to the object generating the message. For example, it is common to implement objects that send a message then either stall waiting for a return message, or notify another object that it should expect a message. In either case, if the message does not occur, the application breaks.

In a behavior-oriented system in accordance with the present invention, objects do not expect messages but will receive them when tapped as described above. As a result, each object can be dynamically modified to generate messages that are directed to any other object (or no other object in the case of terminal behaviors such as displaying information in a GUI). When an object's behavior is terminal it will affect the application's behavior, but it will not break the application.

Unplanned terminal behavior is a fact of life in software systems. What is important is implementing a graceful mechanism for handling such events. The behavior-oriented model set out above could lead to a proliferation of

dysfunctional objects that fail to generate useful messages. In a particular implementation of the present invention an application component has two simple, built-in goals to be used by the user or by another object as often as possible and to avoid being deleted or hidden. In trying to achieve these goals, the information object learns to "please" the user with useful information; the user "teaches" the object by using it.

This feature of the present invention is implemented as a mechanism contained within each object that grades the ability of the object to implement useful functionality with it's programmed behavior. For example, each component maintains one or more "primary value" registers that hold a value indicating the objects viability or worth to the system. When a component is used the object contains primitive methods for increasing the primary value. Optionally, positive reinforcement can be given when a component receives useful information
(e.g., information meeting specified value criteria) triggering the recipient to reward the sending object with primary value credits. Primary value credits can be depleted implicitly as a function of time, or explicitly as a result of issuing a poor message (i.e., a punishment).

In this manner, objects that fail to produce valuable results and/ or fail to direct their results to an appropriate object will have a net depletion of their primary value. Objects succeeding in producing valuable results directed to an appropriate recipient object will have a net increase in their primary value. In one implementation, objects self destruct when their primary value falls below a specified minimum. In another embodiment, objects implement a primary value test in their tap sequence so that an object wishing to send a message can select among a plurality of potential target objects based upon the primary value data returned during the tap sequence.

Referring again to FIG. 3, component 202 is accessed by a message from, for example, a user interface component 306 or an automated scheduling component 307. The message object provided to component may include specific information needed by a "post" or "get" message that will be used by components 301-30 to access specific information. When component 202 receives a message, URL object 302 responds by composing an HTTP message with a target URL including the post or get

information derived from the input message to a data port component 303. Data port component 303 interfaces with hardware and/or operating system resources to send the HTTP message and receive an HTTP response including, for example, an embedded HTLM page. Data port component 303 has no knowledge of where it is sending a message or the message content, or whether a message will be returned. It performs a specific function at runtime by executing generic behaviors on runtime specified data provided in the message it receives.

The response message is sent to page grabber 304 which, for example, is configured generically to strip off HTML tags from a message it receives. Again, page grabber 304 has no a priori expectation of the message it is to receive. It has an interface that accepts an HTTP or HTML message. If no message is ever received, page grabber 304 simply remains dormant. When page grabber 304 receives an HTTP or HTML message it strips off the tags and sends a plain text message to one or more select text component (s) 305. A select text component can be configured to, for example, find the first string including a "$" and convert that string to a value.

Another select text component 305 can be configured to, for example, find the first string value following a specified string (e.g., "Company Name:"). Select text component (s) 305 has no knowledge of the received message and may not find the text to be selected in which case it simply becomes dormant.

If select text component 305 finds the text, a message is generated to component. Component can compile the messages generated by a variety of components into a message that it then sends to, for example, a company datastore object. A company datastore object 308 is a generic object that can store company-specific information retrieved from the page using, for example, the company name text as an index.

**Data Mining Application**

The broad concepts of the present invention are usefully described in terms of a specific application. The particular implementation of present invention described herein is an environment for business and other professionals who need to obtain, manage, and use digital information and services. Data is an abundant resource that can be mined, stored, and processed. This processed

data can be used much more profitably than the raw data from which it comes. For purposes of illustration, a specific implementation for monitoring financial information is described below.

The present invention enables a unique application that makes digital information and services easier to find and use. To achieve this goal, the present invention includes a specific application in which a membership-based behavior-oriented system and processes is configured to act as an intelligent digital assistant. The digital assistant processes, organizes, and manages digital information, learning from the user's instructions. This embodiment of the present invention assists the user by:

Obtaining digital information from a variety of heterogeneous sources;

Managing the information once it is found; and

Integrating disparate information and services into a single entry point.

Digital information is often hidden in raw data deposits, which the user can access using, for example, an Internet search engine (sometimes called a Data Portal). Even though the search engine provides a portal to the data, the meaning or significance of the data may be difficult for the user to discover on her own or using conventional data access technology including search engines. Sometimes the mere amount of data returned by a search engine obscures the significant information.

The specific implementation of the present invention uses a process called "data mining", as discussed above. In data mining, raw data is processed by collections of instantiated software objects which are organized into "data miners" 409 (shown in FIG. 5). Data miners 409 process raw data by applying simple rules and strategies to the data. Each miner applies a different rule or variant on a rule, to the work to be done. When a user interacts with the information the miners have produced, the interaction will either encourage or restrict both the miner that produced the information, and the information object itself.

In the specific implementation, a seed program 411 is delivered to a user. In an initial state, seed program is essentially equivalent to client application 409 because no other components have been downloaded. The seed program comprises executable code on the user's system. The seed code may be written to a specific platform such as UNIX or a Windows operating

system. or may be written to a platform independent environment such as the Java programming environment. Java is a trademark of Sun Microsystems, Inc.

Upon execution, the seed program implements a command line or graphical user interface such as a dialog box. The user supplies identification information (e.g., user name and password). The seed program then creates a link to server 401. Server 401 downloads a startup message including a unique user ID assigned by server 401, a list of modules to load, and a set of startup components. The remainder of system and processes 400 includes dynamic data and service resources such as a news service 416, government information data store 412, and a financial service 413. Government information store 412 is an example of a preferred portal 212 (shown in FIG. 2) whereas financial service 413 is an example of a preferred service 213 shown in FIG. 2. Server 401 is coupled to a feature repository that includes a variety of generic components.

Referring now to FIG. 5, software application 402 downloads and installs system components such as a system object 501, a class loader 502, and a user interface 503. The particular implementation includes a component 504 called an object inspector that enables a user to interact with individual components and their constituent components and objects. This interaction enables a user to examine a component to determine its properties and in some cases to directly manipulate those properties. Preferably, GUI component 503 supports a "drag-and-drop" style interface enabling components to be visually represented, selected using mouse/screen interaction, and moved into or onto other components.

Client application 402 also includes a profile 506 that holds the user ID, a list of components to load, and a group membership list indicating the name or identity of any user-groups to which the user belongs. For example, one user may belong to a group called "paid subscribers" which provides access to certain features that another group called "free subscribers" cannot access. A startup XML document 507 is also included provides a hierarchical text-based description of the client application 402. An example startup XML file is shown in FIG. 7. XML parser object can read the application description document 507 and interacting with system object 501 and class loader component 502 access and download components necessary for operation. At this point

system object 501 (or an equivalent component) implements a link to a file system enabling access to datastore 414 and cache 404.

Seed code 411 can become dormant once client 402 has downloaded system components sufficient to support sustained operation. It should be noted that during subsequent restarts, seed 411 will look for already installed components before requesting downloads from server 401. Because components can be replaced dynamically at runtime, seed 411 can actually boot up using old components, then the client application 402 can replace those older components with fresher versions at runtime in the background.

Client 402 reads XML application description document 414, checks it profile document 506 to determine if it already has the required component. If the component is not already present, application 402 makes access requests to server 401 and downloads selected components 409 based upon the user supplied identification information. For new users (i.e., non-authenticated users) or anonymous users a base level of components 409 may be downloaded. Seed 411 then instantiates components 409.

The preferred implementation of the present invention uses declarative application construction techniques for some if not all application functionality. Declarative application construction controls the use of procedural code in the application by defining the software application as a set of objects (or components) having attributes. The application can thus be defined as a hierarchical listing of objects and a listing of attributes for those objects, without procedural code used to define the application itself.

By way of example, a document authored using extensible markup language (XML) and stored as XML file 507 is used to describe applications. XML documents are a useful format because they provide a standard grammar that is well understood. Moreover, the XML language is extensible, actively developed, and readily transportable through a variety of communications media using commonly available HTTP transport mechanisms. Routers, switches, network ports, and other network devices handle XML formatted documents and are configured to handle them appropriately and reliably. It is contemplated that other formats and transport mechanisms may be used such as HTML forms, CORBA objects and the like.

The application description and/or component (s) 409 are listed an XML document called an "application description document" 701 and 801 shown in greater detail FIG. 7 and FIG. 8 respectively. An application description document 507 may be defined by a "well formed" XML document or a "valid" XML document associated with a data type description (DTD). The application description documents 701 and 801 comprises a plurality
of nested hierarchically arranged elements where at least one of the elements corresponds to a component object 409 in the client application 402.

Fig. 7 shows a start-up application description document 701 listing components that are instantiated during start up or boot operations. In the particular implementation shown in Fig. 7 containers for local data, object storage, and windowing are specified. The containers are defined to have specified behaviors and an interface defined by listed nodes. Primitive behaviors are included for connecting to the server, loading new classes, and managing update services. The XML document 701 also identifies the code file (e.g., "BIN/GUI" where the startup components are located. Each element declaration can include attributes that are assigned to that instance of the named object. Attributes comprise key=value pairs where the key is a parameter used by the associated component and the value is a value given to that parameter in the particular instance of the component 409. For example, the NODE NAME declarations in Fig. 7 are attributes.

Fig. 8 illustrates an XML file 801 declaring the constituent components of an exemplary component 409 in accordance with the present invention. As shown in Fig. 8, the XML file 801 includes a hierarchical listing of nested components. In accordance with the present invention, each behavior object includes a globally unique identification. For example, object identifications are indicated as "BEHAVIOR ID" and "EVAL_ID" in FIG. 8. Optionally, an element declaration includes a version identifier (not shown), although new versions can be indicated also by assigning the new version a new object ID value. The ObjectID information uniquely identifies each component object instance among every component object instance of that type that has ever been created using the system in accordance with the present invention. This global identifier enables every object to know exactly what

object has sent a message. This information can be important in a behavior-oriented system as, like biological systems, a components behavior can be modified based on who has asked the behavior to be carried out. Because objects within components are high level, somewhat generic objects they appear indistinguishable from each other except for this global identification feature.

It is contemplated that as an application such as client 402 evolves, the components 409 therein will evolve with new versions implemented to improve performance or offer additional functionality. Also, components 409 will be added and removed dynamically. Accordingly, the application description document can specify a specific version of an object or component 409 that is to be instantiated. By default, for example, the most current version from server 401 can be used, or the most current version available in cache structure 404. However, an application author can specify an older version using the ObjectID and versionID attributes shown in FIG. 7.

Application changes are readily implemented by changing the application document description instance (e.g., documents like 701 and/or 801) and downloading the new instance to client application 402. The components 409 listed in the newly downloaded application description document are instantiated with the attributes declared in the application description. Because the components 409 are not tightly connected, the new components can be instantiated dynamically without affecting other components.

In cases where the new application description declares components or versions that are not available locally in cache 404 the specified components 409 are downloaded from server 401. All of the update process will typically occur in the background. Preferably server 401 includes the ability to push application description documents out to client 402 so that the update process is initiated by server 401 rather than by any action of client 402.

**System Implementation**

The present invention, in a descriptive preferred embodiment, is described in terms of a distributed system and processes such as an enterprise computing system or framework using public communication channels such as the Internet. However, an important feature of the present invention is that it is readily scaled

upwardly and downwardly to meet the needs of a particular application. Accordingly, unless specified to the contrary the present invention is applicable to significantly larger, more complex network environments as well as small network environments such as conventional LAN systems.

As discussed above, instead of defining a static set of features, the present invention is completely dynamic, requesting features from a server at runtime as needed. It is contemplated that the present invention will be particularly useful in environments that need to be extensible and support a wide variety and dynamically changing set of application behaviors. The present invention provides a method and mechanism for defining application components and so can be adapted to new components (and the behaviors implemented by those components) that were unavailable or unknown when the software application is first implemented.

One configuration of the implementation is shown in FIG. 1. illustrating an exemplary system and processes 100. Essentially, a number of computing devices and groups of devices are interconnected through a network 101. For example, a LAN 102 and a LAN 103 are each coupled to network 101 through gateway machines 104 and 105 respectively. LANs 102 and 103 maybe implemented using any available topology such as a hub and spoke, topology of LAN 102 and a loop topology of LAN 103. LANs 102 and 103 may implement one or more server technologies including, for example a UNIX, Novell, or Windows NT, or peer-to-peer type network. Each network will include distributed storage implemented in each device and typically includes some mass storage device coupled to or managed by a server computer. Network 101 comprises, for example, a public network such as the internet or another network mechanism such as a fiber channel fabric or conventional WAN technologies.

LAN 102 includes one or more workstations such as personal computer (PC) 106. LAN 102 also includes a server machine 107 and one or more shared devices such as printer 108. A hub or router 109 provides a physical connection between the various devices in LAN 102.

Router 104 is coupled through gateway 109 to provide shared access to network 101. Gateway 109 may implement any desired access and security protocols to manage access between network 101 and devices coupled to network 102. Similarly, network 103 comprises a collection of workstations

III, 112 and 113 that share a common connection to network 101 through gateway 105.

Distributed computing environment 100 further includes a wide variety of devices that have a logical connection to the network supported by a physical connection to network 101. For example, a stand alone workstation 114 may. couple to network 101 through a modem or other suitable physical connection. Likewise, notebook computer 11 and palmtop computer 116 may connect to network 101 using known connection technologies. It is contemplated that a wide variety of devices may join the distributed network 100 including mobile phones, remote telemetry devices, information appliances, and the like. An important feature of the present invention is that it tolerates and adapts to an environment filled with heterogeneous hardware devices coupled to the network 101 from a variety of physical locations.

Each of the devices shown in FIG. 1 may include memory, mass storage, and a degree of data processing capability sufficient to manage their connection to network 101. The computer program devices in accordance with the present invention are implemented in the memory of the various devices shown in FIG. 1 and enabled by the data processing capability of the devices shown in FIG. 1. In addition to local memory and storage associated with each device, it is often desirable to provide one or more locations of shared storage such as disk farm 116 that provides mass storage capacity beyond what an individual device can efficiently use and manage.

Selected components of the present invention may be stored in or implemented in shared mass storage such as disk farm 116. FIG. 2 illustrates components of a system and processes 200 in accordance with the present invention. Server 201 it powers computing environment 200 and provides access to all features as well as managing membership privileges. At various times during operation a the Client connects through network 203 to server 201.

Network 203 comprises any type of packet switched, circuit switched, or cell switched network architecture and may be implemented using copper, fiber optic, radio frequency or other available data communication technology. Network 203 may comprise a local area network (LAN), wide area network (WAN), or public network such as the Internet.

Although it is important that the Client have access to server 203 so that client 202 has access to the freshest versions of components 209, Client 202 can remain disconnected from server 201 for a indeterminate amount of time, such as in the case of a handheld or laptop computer that can used for unspecified periods of time away from a connection to network 203. While disconnected from network 203, client 202 relies on information and components held in client cache 204, and features and components previously obtained while connected to network 203.

Although the invention has been described and illustrated with a certain degree of particularity, it is understood that the present disclosure has been made only by way of example, and that numerous changes in the combination and arrangement of parts can be resorted to by those skilled in the art without departing from the spirit and scope of the invention, as hereinafter claimed.

## CLAIMS




**WHAT IS CLAIMED IS:**

1. In a distributed computing system comprising at least one client computer in communication with at least one server, a method for operating a client application
comprising the steps of:

executing a seed program on the client computer to establish a connection with the server;

downloading a set of system components from the server to the client computer;

initializing said set of system components using said seed program;

using said set of system components to interface with a user at the client computer to determine the required functionality of the client application;

requesting additional components from the server that are necessary to perform the required functionality; and

installing the additional components dynamically into the software application.

2. The method of claim 1 wherein said step of executing a seed program on the client computer includes the steps of:

requesting an identification from a user at the client computer;

forming a connection with the server;

passing said identification from the user to the server;

receiving components from the server to enable the user identified by said identification to interface with the client application.

3. The method of claim 2 wherein said step of receiving components from the server includes:

receiving a profile assigned to said identification, said profile including user-specific information available to the user linked to said identification.

4. The method of claim 2 wherein said step of receiving components from the server to enable the user identified by said identification to interface with the client application includes:

said components including an initial code library to enable the client application to perform basic system functions.

5. The method of claim 1 wherein said step of downloading a set of system components from the server to the client computer includes:

downloading a component loader for loading new components into the client application.

6. The method of claim 1 wherein said step of downloading a set of system components from the server to the client computer includes:

downloading components for an application to determine the initial application structure and the attributes that are to be assigned to each instance of the components of the initial application structure.

7. The method of claim 1 wherein said step of using said set of system components to interface with a user at the client computer to determine the required functionality of the client application includes:

determining the functionality required by the user; and

selecting the components necessary to perform the user-selected functionality.

8. The method of claim 1 wherein said step of installing the additional components dynamically into the software application includes:

incorporating data received from the user during said step of using said set of system components to interface with a user at the client computer to determine the required functionality of the client application into said downloaded components during runtime to provide application-specific functionality.

9. The method of claim 1 wherein said method further comprises the step of:

causing said seed program to enter a dormant state after the system components are initialized.

10. The method of claim 1 wherein said components include:

loose coupling of instantiated objects within said components to allow the interfaces of the instantiated objects to change substantially and still effectively communicate with other objects.

11. The method of claim 10 wherein said interfaces of the instantiated objects include:

a first stage wherein a first object communicates with a second object through a simple interface to allow said second object to indicate the particular entry points of said second object to said first object; and

a second stage wherein said first object sends a rich message to said second object through said entry points of said second object.

12. The method of claim 1 wherein said components include:

value registers to grade the ability of the component to implement it's intended functionality.

13. The method of claim 12 wherein said value registers include:

increasing the value in said value register each time the component is utilized.

14. The method of claim 12 wherein said value registers include:

increasing the value in said value register each time the component provides a useful function.

15. The method of claim 12 wherein said value registers include:

decreasing the value in said value register as a function of time.

16. The method of claim 12 wherein said components include:

selecting a component to send a message to based upon the value in the components value register.




17. The method of claim 1 wherein said server includes:

a feature repository for maintaining components for the client application.

18. The method of claim 1 wherein said server includes:

a data storehouse for storing information for the user of the client application.

19. The method of claim 1 wherein said server includes a content server for maintaining preprocessed data.

20. The method of claim 1 wherein said client application includes:

components for seeking additional components to upgrade said client application;

a component loader for loading the additional components into said client application;

installing the additional components to upgrade said client application.

21. A computing system for dynamically providing functionality to a client application, said system comprising:

a client computer in communication with at least one server;

said server having a component storage facility;

a client application installed on said client computer, said client application initially includes a seed program on said client computer, said seed program having minimal executable code to establish communication with said server to download and install a set of system components from said component storage facility to said client computer; and

said installed set of system components include a user interface between a user on said client computer and said client application to determine the functionality desired by the user and to download and install additional system components from said component storage facility to perform the required functionality.

22. The system of claim 21 wherein said installed set of system components includes:

components for an application to determine the application structure and the attributes that are to be assigned to each instance of the components of the application structure.

23. The system of claim 21 wherein installed set of system components include:

components for an application to determine the functionality required by the user; and

selecting and installing components necessary to perform the user-selected functionality.

24. The system of claim 21 wherein said seed program includes:

means for requesting an identification from a user of the client computer;

means for passing said identification to said server;

means on said server for determining the profile of the user assigned to said identification and downloading the components that are authorized for that user.

25. The system of claim 21 wherein said seed program includes:

means for causing said seed program to enter a dormant state after the system components are initialized.

26. The system of claim 21 wherein said components include:

loose coupling of components to allow the interfaces of said components to change substantially and still effectively communicate with other components.

27. The system of claim 21 wherein said components include:

interfaces on each of said component to allow communication between said components;

a first stage on each of said interfaces allowing simple communication; and

a second stage on each of said interfaces including the entry points of the component, wherein said first component communicates through said first stage on a second component to determine the second stage of said second component to allow said first component to send a rich message to said second component.

10

28. The system of claim 21 wherein said components include:

value registers to grade the ability of the component to implement it's intended functionality.

29. The system of claim 28 wherein said value registers include:

means for increasing the value in said value register each time the component is utilized.

30. The system of claim 28 wherein said value registers include:

means for increasing the value in said value register each time the component provides a useful function.

31. The system of claim 28 wherein said value registers include:

means for decreasing the value in said value register as a function of time.

32. The system of claim 28 wherein said components include:

means for selecting a component to send a message to based upon the value in the components value register.

33. The system of claim 21 wherein said server includes:

a feature repository for maintaining components for the client application.

34. The system of claim 21 wherein said server includes:

a data storehouse for storing information for the user of the client application.

35. The system of claim 21 wherein said server includes a content server for maintaining preprocessed data.

36. The system of claim 21 wherein said client application includes:

components for seeking additional components to upgrade said client application;

a component loader for loading the additional components into said client application;

installing the additional components to upgrade said client application.

37. An information gathering system for use on a client computer in communication with at least one server, said information gathering system comprises:

a client application on the client computer having a user interface;

one or more collection of software objects having a set of rules for determining if data fits within said set of rules;

interfaces on each of said software objects to allow communication between said software objects;

a first stage on each of said interfaces allowing simple communication; and

a second stage on each of said interfaces including the entry points of the component, wherein a first software object communicates through said first stage on a second software object to determine the second stage of said second software object to allow said first software object to send a rich message to said second software object.

37. The system of claim 36 wherein said software objects include:

value registers to grade the ability of each of said software objects to implement it's intended functionality.

38. The system of claim 37 wherein said value registers include:

means for increasing the value in said value register each time the software object is utilized.

39. The system of claim 37 wherein said value registers include:

means for increasing the value in said value register each time the software object provides a useful function.

40. The system of claim 37 wherein said value registers include:

means for decreasing the value in said value register as a function of time.

41. The system of claim 37 wherein said software objects include:

means for selecting a software object to send a message to based upon the value in said value register of the software object.

42. The system of claim 36 wherein said server includes:

a feature repository for maintaining components for the client application.

43. The system of claim 36 wherein said server includes:

a data storehouse for storing information for the user of the client application.

44. The system of claim 36 wherein said server includes a content server for maintaining preprocessed data.

45. The system of claim 36 wherein said client application includes:

components for seeking additional components to upgrade said client application;

a component loader for loading the additional components into said client application;

installing the additional components to upgrade said client application.