EXHIBIT A

(Part 3 of 3)

The next classifier event is removed from the classifier's event queue (**3601**). Since the classifier is subscribed to the notifier, it receives notifier events when objects are added to the system; when objects change their property values; and when objects are removed. In each of these cases the classifier is responsible for determining the set of containers (folders, collections, or other specific containers) to which the object belongs.

1) If the event is an object added event (**3602**), then the classifier determines the set of containers to which the object belongs and creates an added set and an empty removed set (**3603**).
2) If the event is an object changed event (**3604**), then the classifier performs the following procedure (**3605**). First, the existing container set is retrieved. Next, the object is classified, resulting in a set of containers to which the object should now belong. Next, these two sets are compared, resulting in the added set, which includes containers to which the object should be added; and the removed set, containers from which the object should be removed.
3) If the event is an object removed event (**3606**), then the classifier creates an empty added set, and sets the removed set to the object's container set (**3607**).
4) Finally, the object is added to the containers in the added set (**3608**), and removed from containers in the removed set (**3609**).

In this way, each object referenced in the classifier event queue ends up in the correct set of containers that select for its current property values.

Classification of a Single Object

The classifier determines container membership for an object through the process described in FIG. **37**:

Initially, the result set, which contains the set of containers to which the object should belong, is set to empty (**3701**). Then the classifier asks the object's source (e.g. the File or EMail domain) to perform an initial classification of the object (**3702**), resulting in a new result set. The Files domain, by way of example, would add a file object to its enclosing folder.

Objects can be classified into collections by specifying in each collection a list of key phrases whose occurrence in an object means that the object should be referenced in the collection. A collection may have many key phrases, and the same key phrases may be specified in many different collections. The MFS-configured computer system's key phrase classifier performs a single-pass, multiplex sorting of a given object into an unlimited number of collections based on the pKeyPhrases properties defined in those collections and the textual content of the object.

The classifier runs through each text property in the object (**3703**) and for each property goes through each key phrase in the classifier (**3704**) determining whether the key phrase exists in the property's value text (**3705**). If so, it adds the entire set of collections associated with the key phrase, since a single key phrase may be listed by multiple collections (**3706**).

The key phrase classifier is based on a novel use and implementation of the Aho-Corasick string search algorithm. The classifier begins by scanning each collection when MFS is launched, and adds each key phrase to the Aho-Corasick finite state machine. At the terminal nodes for each key phrase is a list of collections that specify that phrase; as the collections are scanned and each key phrase is added, the list of collections at each key phrase is kept up to date with all collections that specify it.

All objects maintain a list of collections in which they occur. Classification is accomplished by scanning the text body of an object using the Aho-Corasick algorithm. When a key phrase is found within the text, the list of collections for that phrase is fetched from the finite-state machine and united to a final (list or set) of collections in which this object should appear. When the entire text body has been scanned, the final set is compared with the initial set. For collections that appear in both sets, nothing is done. For collections that appear in the initial set but not the final set, the object is removed from those collections. For collections that appear in the final set but not in the initial set, the object is added to those collections. Finally, the object's collection set becomes the final set, reflecting that object's membership in those sets.

Next, the classifier goes through each collection (**3707**) and determines if the object satisfies the query specified by the collection (**3708**). If so, the collection is inserted into the result set (**3709**).

Finally, the result set is returned (**3710**) and the object is placed into the collections listed in same.

View by Reference

A container C (folder, collection, or any other container) is viewed by reference using the following process.

1) An empty result set R is created.
2) For each object in the container C, the set of collections to which that object belongs is added to the result set.
3) A new container V representing the reference view, is created.
4) For each collection A in the result set R, a new proxy collection P is created, whereby the contents of the proxy collection P is simply the objects in C that are also in the collection A; this is done through a set intersection of the collection A and the container C. Generally, this proxy collection is simply defined by an MFS metadata query on P which states that the contents of the proxy collection are the intersection of the contents of collection A and container C.
5) The final container V that is the reference view now contains a set of proxy collections, each of which holds a subset of the original objects in C.

The reference view may then be further refined by choosing a proxy collection P's contents (a subset of C's) to view by reference. This is done as follows:

1) The reference view V adds P to its prefix set.
2) V replaces its proxy collections with new proxy collections, using the same process as above, but with one difference: each proxy collection's MFS metadata query now states that the contents of the proxy collection are the intersection of the contents of collection A, container C, and all the collections in V's prefix set.

In this way, a view of the "Today" collection, which shows the objects modified today, can more easily be viewed by reference, which shows that (for example) the Received email collection was changed today, as well as the Documentation project. Clicking on the Received proxy collection in the view reveals email objects received today; further refining by Received will show the collections in which email was received today: typically a list of the contacts from whom email was received.

Display and Layout

MFS provides an architecture for display and layout of objects in a variety of ways. Individual objects are viewed in content viewers defined by each domain, which is responsible for the individual object types. Viewing containers of objects (e.g. collections or folders in icon or list views) is based on

three classes of objects: forms, figures, and scenes. A unique type of list view is implemented by MFS's sticky paths mechanism.

Content Viewers. For each specific type of object, a content viewer is available for viewing the actual object data. By way of example, contact objects are displayed in a window showing first and last names, addresses, and so on. Email messages have their own specific viewer, with the standard to, from, and body panes within the window. Text files or notes are displayed in a standard text-editing window.

In the case of content viewers that provide text fields, key phrases can be highlighted automatically when examining the object's contents and provide a hyperlink to the defining collection automatically. If multiple collections specify a given key phrase, the popup menu will list all collections that do so, allowing the user to choose which collection should be opened.

An object that has been classified into several different containers will indicate this in the Information window, where all of the containers are listed and may be opened.

Forms. A form is a 2-dimensional layout of property values of a single object. For example, a standard icon view includes two fields: the icon property situated and centered above the name property. A list view form will include a left to right arrangement of the object's icon, name, and additional properties as required by the display. Forms are used by figures to determine the appearance of the object in the window.

Figures. A figure is a drawable entity representing an object. Figures are linked to forms, which define how the figure should be drawn. Figures also provide the ability to be highlighted when clicked; to have their properties edited directly, such as the name in an icon view; and to be dragged from one place to another within the MFS interface. Figures are arranged within scenes, which determine where each figure should be located.

Scenes. A scene is an arrangement of figures in 2 or 2½ dimensions (2½ dimensions include a representation of depth). The scene is generally responsible for determining the form the figures within the scene should take; thus, MFS defines a small icon scene whereby the form defines a small rectangle for the icon property and a rectangle to its right for the name property; a large icon scene with the icon rectangle above the name rectangle; variants on the previous; and a preview scene where the object's preview property is drawn within a slide frame, along with the object's name, size and modification date; and various list views, among others.

The scene is also responsible for locating each figure within the scene based on certain conditions. For example, in the small icon scene the objects are sorted by a given property (chosen by the user) and then laid out top-down, left-to-right in the window; scrolling to the right shows additional figures. The large icon scene lays out the figures left-to-right, then top-down in the window; scrolling down shows additional figures.

The user typically chooses which scene to display objects in, by selecting an item in the View menu. Unique and specialized scenes may be defined by domains as well, if needed.

Sticky Paths

Sticky paths are a unique way of displaying hierarchies of objects within MFS. Often hierarchies of objects are displayed in a sort of outline view, whereby objects are listed in some order (typically alphabetical), and sub-objects that are contained in other objects may be displayed or hidden at the user's control. An object that contains other objects in this way may be either expanded (displaying its sub-objects) or collapsed (hiding them). Each object has a depth, a numeric value that describes how far down the hierarchy it exists; in particular, how many nodes down the hierarchy tree from the root. Objects at the same depth are known as siblings. The depth determines how far the object is indented to the right in the outline display.

When an object is collapsed, any object that contains others is indicated in some way with a clickable region, typically a symbol such as a + sign or a triangle, that may be clicked. Clicking on the region expands the object by displaying those objects which are contained within below the object and indented to the right by a specific amount, due to their depth being one greater than the depth of the parent object. Other objects that were at the same level as the object being expanded are moved down the display by the amount needed by the expanded object.

Objects within an expanded object may in turn be expanded, resulting in several levels of expanded objects and multiple indentations.

The path to an object is defined as the name of the object itself, prefixed by the names of the nested containers in which the object exists in outermost order. For example, if an object E is contained in an object D, and in turn D is contained in C, and C is contained in B, the path to the object E is generally described as B:C:D:E.

In a highly-hierarchical display with many objects that do not fit on a single screen, the user must scroll the hierarchy display in order to see objects lower down on the list. In particular, if some objects have many sub-objects which are in turn expanded to show their respective sub-objects, it is quite easy to forget what part of the hierarchy one is looking at, i.e., where the user is on the path, since the enclosing objects have scrolled off the top of the display.

Sticky paths are a mechanism by which a scrollable outline of this form is displayed in two dynamic parts: a path area and a scrollable area. Sticky paths provide the user with a constant awareness of his location in the hierarchy by:

1) constantly displaying the current path to the topmost item in the scrollable area above the scrollable area, and dynamically updating the path as the objects are scrolled up and down;
2) dynamically resizing the scrollable area to accommodate the path display.

Implementation Details. The sticky path scrolling mechanism is implemented, by way of example, in the following pseudocode:

1) Get the old path frame from the current display.
2) Get a list of container objects that comprise the path to the topmost figure in the outline. Do this by determining the object at the top of the scrolling region, and then walking up the outline item's parent tree until there are no more parents.
3) Set the path display by starting at the top of the list and drawing each parent in turn, indented appropriate to the parent's depth.
4) Get the size of the new path frame.
5) Determine the difference between the heights of the old and new frames.
6) If the difference is zero, then the size of the path hasn't changed, and the bits can be scrolled within the scrolling region.
7) If there is a difference in height, then we first adjust the size and location of the scrolling region based on the amount of the change.
8) If the difference is greater than zero (e.g. the path is smaller than it had been previously), then we don't

scroll, but we do have to refresh the topmost figures of the area that was vacated when the path region was made smaller.

9) If the difference is less than zero (e.g. the path is larger than it had been previously) then the resizing of the scrolling region is sufficient, and no scrolling is necessary since the topmost figure in the scrolling region will have been moved up into the vacated path area.

In this way, the current path to the topmost item is always visible.

Domains

Domains define an "area of expertise" for MFS. Typical domains include personal information management (appointments and contacts); file management (folders, files, disks); image file management (also known as digital asset management); and email, among others. Domains provide a way to extend MFS's capabilities and functions by leveraging MFS's architecture in new ways.

A domain is responsible for implementing the following procedures:

Registration of new object classes and properties for same;
Creation of new objects of specific classes when needed;
Creating and managing UUID mappings between reference objects and external data;
Adding metadata properties to objects;
Basic classification of objects by class and property values;
Updating of object metadata in response to changes in the operating environment; and
Performing basic operations on behalf of objects that the domain manages.

The following describes these procedures for domains defining file management, email handling, music organization, personal information management, image management, and organization by time.

The File Domain. This domain registers new object classes for disks, folders, and files. The properties that are registered include file and folder names; creation date; modification date; physical size; and permissions, among others.

The domain is also responsible for scanning folder and file objects, and resolving changes with the objects on the disk as the disk contents change. For example, when a folder's modification date differs from its corresponding object in MFS, the domain compares the folder's contents with the contents of the folder object, and creates or deletes file and folder objects in the folder object as required to match exactly the contents of the disk folder. Similarly, if a file's modification date changes, its corresponding file object is updated with the current filename, modification date, size, and so on in order to mirror exactly the file's property values.

Certain file types are handled specially by this domain. In particular, application and document files must have the appropriate icons associated with them, and behaviors such as opening a document must be defined to launch the correct application.

This domain provides window layouts for information about files and folders, and utilizes built-in MFS windows for displaying folder contents. Window layouts for certain types of files also are supported, including text files and clippings.

All sources provide the ability to scan external data to add information to the Working Set, and match the external data to update MFS's internal reference objects as the external data changes. By way of example we describe the File domain's implementation details of these two processes.

Implementation Details. The File domain is notified of files to add to the Working Set as follows. The user drags a folder to MFS's workspace window; this causes a reference object is created for the folder by the specific source handling the folder; in this case, File source, which is responsible for all file system objects. Next, a scanner thread is created with the reference object as a parameter. This thread performs the following functions, in order:

1) Traverse: A procedure recursively descends the folder hierarchy, creating data entries that are stored in an array. Each entry contains a file system specifier that represents the file; a depth in the folder hierarchy; and a flag that determines whether the file system specifier is for a folder or a file. The array is then sorted, deepest objects in the tree first (so that files within a folder are created before the folder is).
2) Annotate: Once this array has been populated, the entries are annotated with metadata that can be efficiently fetched "en masse", such as file and folder comments.
3) Create: The entries are fetched one by one from the array. For each entry, a reference object is created with the entry's information (e.g. the file specifier and any metadata that was previously fetched and added to the catalog). A new array of reference objects is created.
4) Classify: Each object in the array is then classified by examining its metadata and determining in which collections the object belongs, based on the collections' specifications. Every collection that is modified (e.g. that has received a new object through the classification process) is added to yet a third array for notification.
5) Notify: Finally, each collection that participated in the classify step is notified that it has been changed. This typically results in the collection updating dependent property values (e.g. count of contained objects), which are then updated in a separate thread.

Then, the folder reference object is then added to the HFS source's working set, which is the set of all folders that MFS should manage. The Workspace window is then updated, since the working set property determines which folders are shown in the window.

Because the file system data that the File source tracks changes over time, the File source has the ability to match these changes and propagate them throughout the catalog and object store. This is done as follows:

1) During the match process, MFS compares its stored information against the source's versions of the same information. If there is an indication of difference between a reference object in MFS and the actual external object (by noting a changed modification date, for example), MFS invalidates the reference object's metadata.
2) Once the metadata has been invalidated for all objects suspected of being changed externally, MFS puts each object on the updater queue.
3) While items exist on the updater queue, the updater does the following:
4) Takes the next item off the queue
5) Tells the object to update itself. This, in turn, causes the object to go to the source to determine what the true values should be for each of the invalid property values.

Once the values have been retrieved from the source (by asking the file system for file metadata such as name, modification date, etc.), the new values are set in the catalog and the property is validated.

6) The catalog then creates notification events based on the objects and values that were set, and enqueues them on the notifier queue
7) The notifier goes through each event, telling all of the object's dependents (any containers to which the object

belongs as a member, as well as any other dependent objects) that it has changed the given properties.

8) Those objects, in turn, determine whether any of their properties need invalidation. For example, if a file's size has changed, the containing folder's size property needs to be updated, since it is dependent on the sizes of all the files within the folder.

Using these two processes, scan and match, the File domain creates a Mirrored Object System within MFS that exactly represents the file and folder hierarchy that the Domain tracks, regardless of external changes.

The EMail Domain. This domain registers new object classes for mailboxes, which describe servers and passwords for retrieving mail; signatures, for signing messages; and email messages themselves. Properties for these objects include server names, addresses, and passwords, for mailboxes; a name and text string, for signatures; and the full suite of email properties for messages, including From, Date, Subject, and message body.

The domain is responsible for communicating with mail servers for both sending and receiving email; creating outgoing message objects; and for creating new received message objects as they are downloaded from the server. Attachments are handled by communication with the File domain for creating and linking to file objects as they are downloaded to disk. Finally, window layouts are provided for outgoing and incoming email messages; mailboxes; and signatures. Behaviors such as sending messages, forwarding, replying, and so on are also supported by the domain.

The Music Domain. This domain, a client of the File domain, registers a new object for a music file, generally in the MP3 format. Properties registered include the track's title, artist, album, genre, and comments.

The domain is responsible for extracting the property values from the file using the ID3 tags that are embedded in the file, and for setting the properties in the catalog for the object. The domain also creates predefined collections for titles by album, and albums by artist, based on the music files that are handled by the system.

The domain may also provide a music player for all files in a given container; in this way the user can play all the tracks on a given album, or tracks grouped together in an arbitrary collection.

Finally, a window layout is provided for information about the music track, showing album, artist, title, and so on in addition to the generic file information such as filename, file size, and so on.

The Personal Information Domain. This domain registers new objects for contacts, notes, appointments, projects, events, and tasks. Specific properties are registered for each object: for contacts, the standard list of contact information such as first and last name, email address, phone, and so on; for notes, the note text; for appointments, the date and time, repeat interval, description, contacts, and so on; for projects, the project name and description; for events, the event name, date, and so on; and for tasks, the task description, priority, and the like.

The domain is responsible for scanning and matching with system-level address book databases, creating, deleting, and modifying contacts as required. Depending on the domain implementation, it may also match with other PIM databases such as Outlook and Palm in the same way, by creating mirror objects in MFS for each object in the target database.

The domain creates predefined collections for all notes, all contacts, and so forth, as well as predefined collections for each contact that collect all objects that reference the contact's name.

Finally, window layouts are provided for each type of object to allow display and editing of the object's data.

The Image Management Domain. This domain, a client of the File domain, registers new objects for file types that store images. Properties registered include image resolution, width, height, depth, and the like.

The domain is responsible for extracting the attribute from the file and attributing the MFS object appropriately, as well as for reading the file data and displaying it as an image within an MFS window.

The domain creates predefined collections for all images of various types (e.g. JPEG files, GIF files, Photoshop files, etc.), as well as a single Images collection for all images.

Finally, a window layout is provided for the display of image files.

The Time Domain. This domain provides no new object classes, but creates and maintains a set of dynamic collections that are based on relative time. For example, the domain creates and keeps up to date a Today collection that changes each day; similarly, Last Week, Last Month, Last Year, and collections created on demand by the user are handled by this domain.

The domain provides a root collection called Time; in this collection the various other collections are created and stored. A collection for the current year contains collections for each month of the year to date; in turn, each month has collections for each week of the month.

Finally, the domain provides window layouts for unique views of objects by time, including a Timeline view where documents are arranged by a date property within a given range, among others. This domain is particularly adaptable to use in the legal field where extensive docketing systems are required.

Additional Domains. It should be understood, as will be evident to one skilled in the art that a wide variety of other Domains may be added, e.g., Location, Space, Event, Symptom, Cause of Action, etc., as the Domains described above are merely exemplary and not limiting of the scope, nature and character of domains that the inventive system can employ. The Domains can be special in nature, as noted by the Symptom for those in the medical profession, and Cause of Action for those in the legal services profession. Another Domain could be "MO" for modus operandi, for use by investigators and police, which can be set to automatically group in collections sets of facts (objects representing text narratives of criminal activities, images and the like) based on similar MOs. This automatic building of collections could be a powerful tool in the criminal justice field. Likewise, engineering professions can build collections with similarities in

data trends or values, e.g., temperatures, materials values, velocity, concentrations of chemical components, etc. for analytic purposes.

INDUSTRIAL APPLICABILITY

The inventive data storage organization, archiving, retrieval and presentation system architecture and technology can be used in a wide variety of applications; the primary being desktop file organization and server data management. The inventive system is remarkably robust, yet is a relatively small application program that can function with any type of Operating System: Microsoft Windows, Windows NT, Windows 2000, and Windows XP; Apple Macintosh OS 9 and OSX; BSD Unix, HP-UX, Sun Solaris, Linux, and the like. Currently the inventive technology is preferably implemented in its current best mode in a form that is executable on the Apple Macintosh OS9 and OSX operating systems.

As to Desktop Organization, the invention is useful as an improved desktop organization application for all types of data, limited only by the domains that can be conceived-of. Domains may be easily created to extend the MFS capabilities to new areas of expertise.

As to File Organization, similar to the Apple Macintosh Finder or Microsoft Windows Explorer, the inventive MFS system provides basic disk navigation and display features. In addition, the File domain allows additional properties to be specified for files, including: a due date; a file species (e.g. an application, a bookmark, a text-readable file, an image file, a font file, etc.); and a file path. Folders have additional properties that are maintained automatically: the size of the contents of the folder; and the depth of the folder from the root directory of the disk, among others.

As to Image Cataloging, image asset management is easily implemented as an extension of the MFS File domain. A domain that can extract relevant information from images found on disk (e.g. size, type of image, colors used, resolution, and so on) is created as a representative object within MFS that has the given properties. Users can then view and select the images that satisfy certain collection criteria. Comments on the objects can also be used to describe and/or define the content of the images, and collections can be created to organize all images based on their described/defined content.

As to Personal Information Management the inventive MFS system is useful for scheduling, organization and tracking of appointments, contacts, events, notes, projects, and tasks as typical kinds of objects that are defined by the PIM (Personal Information Management) domain. What makes the inventive system of particular interest to industry is that the PIM Domain functionality provides a new feature: the ability to organize information objects by person and by project.

As to EMail, a basic embodiment of the inventive MFS system provides basic EMail client services. The objects that the basic EMail domain defines include: mailboxes, email messages, and signatures. This can be expanded to include all types of e-business trust services, including: signature legalization; payment transfers; electronic record retention and verification; electronic filing of documents, including formal/legal documents, applications, forms and the like; privacy and confidentiality guards; identity verification and authentication; access guards; time verification and authentication, including times of sending, acceptance, receipt and performance; and the like.

As to EMail Notification, the inventive MFS system permits the user to watch all postings and create emails describing when a message will be classified into a given collection (automatic forum). This results from the functionality of the classifier; as it is data-driven, it classifies all collections simultaneously. When a key phrase is found it lists all collections from all users that specify that phrase. The phrases can be defined in Boolean search terms for the broadest inclusive categorization and inclusion.

As to Custom Desktop Client, the inventive MFS system includes, by way of example, the useful functionality of Portfolio management by which the user, as a customer of a financial services site, such as Marketocracy.com, can communicate with the site server to synchronize data, e.g., to provide automatic portfolio updates, stock quote display, market, sector and stock performance graphing, and the like.

As to Personal Finance, a personal finance domain may be implemented in MFS to provide objects for checks, checkbooks, receipts, invoices, stocks, funds, and so on. The normal accounting principles apply; the value propagation feature mechanism is used to ensure that the checkbook properties (e.g. balance) are computed from the check properties (e.g. amount of check). Stocks and funds can be updated over the network and their values displayed in MFS.

It should be understood that various modifications within the scope of this invention can be made by one of ordinary skill in the art without departing from the spirit thereof and without undue experimentation. It is apparent to those skilled in the art that many features and functionalities of the inventive MFS can be enabled and realized separately. For example, sticky paths or drag and drop link creation can each be coded in a separate application or applet, or can be provided as incremental upgrades to a program, or as plug-ins or add-ons to existing other productivity, organizational or creativity application programs or applets. That is, MFS can include less than all the dozen or more features described above, and, conversely, MFS is extensible to include additional features and is adaptable to co-operatingly interact and enhance other applications. This invention is therefore to be defined by the scope of the appended claims as broadly as the prior art will permit, and in view of the specification if need be, including a full range of current and future equivalents thereof.

The invention claimed is:

1. A computer data processing system having a central processing unit configured with an integrated computer control software system for the management of informational objects relating to multiple levels of objects organized in at least one of an expandable hierarchy structure and a group structure which include said objects, multiple branches, groups, and/or levels of said objects and their container relationships, and names, identifiers or location paths of said objects within said structure, said data processing system comprising:

a) a computer readable memory including a storage structure for storing information relating to said objects selected from at least one of said objects, said object metadata and said object names, identifiers or location paths;

b) a computer display connected to said memory for displaying said objects, said object metadata, and said object identifiers from said storage structure in a view;

c) a computer-user interface for selective display for viewing of at least a portion of said expandable hierarchy or group structure in said view;

d) an applications program having component architecture code processed by said central processing unit so as, upon user scrolling of said structure, to continuously

render visible, with dynamic updating, in at least one dynamically-updating sticky path display portion at a margin of said view; and

e) said dynamically-updating of said component architecture code being processed so as to provide the function of at least one of:

i) when the scrolling enters the beginning of a new open branch, group or level of said structure, an identifier thereof is rendered in a sticky path display area as a new identifier, said new identifier being visibly displayed in said area adjacent any visible identifier of a previously-entered and still open branch that directly contains said new branch, and if there is no visible identifier of a previously-entered and still open branch, said new identifier is visibly displayed by itself; and

ii) when the scrolling continues past the end of the entries of said new branch, group or level of said structure to exit it such that the entries of said new branch are no longer visible in said view, said new identifier of said new branch is removed from said dynamically-updating sticky path display area, while any identifiers of previously-entered branches remain until they in turn are exited;

thereby to provide continuous automatic dynamically-updated display, in said at least one sticky path display area of said view, of at least one branch identifier of said structure.

* * * * *