

APL. As explained below, system functions like QueryRegEx and GetProfileString can be hooked so that each time they are invoked, another function or application intercepts the call. The Operating System Guard **100** of the present invention will hook each appropriate API function to service the request, if made by an application being actively managed or if made by an application against a configuration item being actively managed. In this way, unless explicitly configured to do so, the present invention can create the application environment without making any actual changes to the end-user's system. Also, any modifications made at run-time by the application can be persisted or removed easily.

[0014] As used herein the term "Operating System Guard" defines a layer between a running application and the operating system of a target computer or client computer that provides a virtual environment in which an application may run. This virtual environment has several purposes. First, it prevents a running application from making changes to the client computer. If an application attempts to change underlying operating system settings of a client computer, such settings are protected and only "made" in the virtual environment. For example, if an application attempts to change the version of a shared object like MSVCRT.DLL, this change is localized to the application and the code resident on the client computer is left untouched.

[0015] Second, the invention presents an environment to a running application that appears to be an installation environment without performing an installation, and is thus a "pseudo installation" or installation-like." All of the settings are brought into a virtual environment at the time the application being served runs, or just-in-time when the application needs the particular setting. For example, if a computer program such as Adobe Photoshop® expects to see a set of Windows Registry entries under HKEY_LOCAL_MACHINE\Software\Adobe and they are not there on the client computer since Photoshop was never installed, a system made in accordance with this aspect of the present invention will "show" those registry entries to the Photoshop programming code exactly as if they were resident on the client computer.

[0016] Next, the invention prevents information that may exist on the client/users machine from interfering with or modifying the behavior of an application. For example, if the user has already existing registry entries under:

[0017]
HKEY_LOCAL_MACHINE\Software\Adobe

[0018] for an older version of Photoshop, but now wishes to operate a newer version, these entries can be hidden from the new application to prevent conflicts.

[0019] Finally, the present invention unlocks application behavior that may not exist as the application is currently written. It does this through the ability to dynamically change the virtual environment according to administrative settings. For example, in a typical instance of an enterprise software application, a client application may expect to read a setting for the address of the database to which the user should connect from a setting in the registry. Because this registry key is often stored in HKEY_LOCAL_MACHINE, the setting is global for the entire client computer. A user can only connect to one database without reinstalling the client,

or knowing how to modify this registry key, and doing so each time they wish to run the application. However, by implementing the present invention, two instances of the application may now run on the same client computer, each connecting to a different database.

[0020] Contexts

[0021] In providing this functionality, each application is able to run in a private context within the system. To the application, it has its own private view of what the system looks like and its behavior. The present invention provides this by its inherent nature. Referring to **FIG. 2**, two separate applications **52,54**, or two instances of the same application (**50** illustrated in **FIG. 1**), can be provided private contexts in which they will appear to have separate or differing copies of system services, configuration and data. In the preferred embodiment, this is the default behavior of the system.

[0022] By extending this concept, the Operating System Guard **100** of the present invention can also provide shared, controlled contexts in which two or more applications **52,54** can share some or all of their virtual settings. This is important for application suites such as Microsoft Office, or for applications that perform differently in the presence of other applications. For example, many applications use Microsoft Word as an engine for doing Mail Merge or document creation functionality. The application must know about the installation or presence of Word and be able to tap into its functions. In the preferred embodiment, two instances of the same application will share a single context by default, while two separate applications will maintain private contexts. Referring to **FIG. 3**, the two applications **52,54** can run while the Operating System Guard **100** provides a shared view of the available system resources.

[0023] Design

[0024] As illustrated in **FIG. 4**, the Operating System Guard is comprised of the following subsystems: core **102**, configuration manager **104**, file manager **106**, shared object manager **108**, device manager **110**, font manager **112**, process manager **120**, process environment manager **114**, loader **116**, and recovery manager **118**. With the exception of the core **102**, the process manager **120**, and the loader **116**, all other subsystems are elements of the Virtualization System described in further detail below. The core **102** is primarily responsible for managing applications and their context as defined by the configuration files.

[0025] The process manager **120** provided by the Operating System Guard allows the core **102** to be informed of any process or thread event that may be of interest. It also provides an abstraction layer to the operating system-dependent implementations for managing a process space and handling thread processing. Processes may be grouped together into application bundles. An application bundle is a group of processes which all share their virtual resources with each other. For example, Microsoft Word and Microsoft Excel may want to share the virtual registry and virtual file system to be able to work together as an application suite. The process manager **120** calls these application bundles "applications". The information about an application exists until the process manager **120** is told to release the application. If another process needs to be loaded into the application bundle, it may do so as long as the application has not been released.