

CONFIGURATIONS FOR BINDING SOFTWARE ASSEMBLIES TO APPLICATION PROGRAMS

CROSS-REFERENCE TO RELATED APPLICATIONS

[0001] The present application claims priority to U.S. Provisional Patent Application Serial No. 60/199,227, filed Apr. 24, 2000, and is also related to copending United States Patent Application entitled "Isolating Assembly Versions for Binding to Application Programs" filed concurrently herewith.

FIELD OF THE INVENTION

[0002] The present invention is generally directed to computer systems, and more particularly to executable computer code such as application programs that utilize shared assemblies.

BACKGROUND OF THE INVENTION

[0003] At one time, computer applications were monolithic blocks of executable code and data, although some of their data such as variable settings could be maintained in separate files. This made tasks like moving or replacing the application simple. In contrast, contemporary computer applications and other executable code (such as an operating system component) bind to and make use of shared components, wherein in general a component is a self-contained software entity, offering a set of functions that can be used by a variety of applications. Such components include dynamic link libraries (DLLs) and objects such as OLE (Object Linking and Embedding) components and COM (Component Object Model) components, including ActiveX® controls. In turn, some of these shared components depend on other shared components.

[0004] On any given machine, at present there is one version of each of these components shared by applications, such as the most-recently installed version, although some mechanisms are known that replace an installed component only when an available replacement component has a higher version number. The metadata maintained for using these components is generally maintained in the system registry, and the application has the names of the needed components compiled into its binary code. Because in general the application does not change as components change, to function properly, global component sharing requires that any shared component function exactly like previous other versions of that component with respect to what an application expects. In practice, however, perfect backwards compatibility is difficult if not impossible to achieve, among other reasons because it is impractical to test the many configurations in which the shared component may be used. For example, both newer and older applications end up sharing the same component, whereby over time, fixing and improving the component becomes increasingly difficult. Moreover, the practical functionality of a component is not easily defined. For example, some applications may utilize unintended side effects in a component that are not considered part of the core function of the component, e.g., an application may become dependent on a bug in a component, and when the component publisher chooses to fix that bug, the application fails. Of course, on the other side, application writers cannot test future versions of components.

[0005] As a result, problems occur when a component is updated to its newer version, such as when a new application or operating system service pack is installed with updated copies of components, as the newly installed component versions become the ones used by other applications and components on the system. The sheer volume of applications and components that rely on other components magnifies this problem, which is sometimes referred to as "DLL Hell."

[0006] One mechanism that provided sharing for some applications while enhancing the stability of other applications was provided in Microsoft Corporation's Windows® 2000 and Windows® 98, Second Edition, operating systems. In general, this mechanism provided a way for an application to be bound to a local copy of a component instead of a shared copy. However, with this solution, a component needed to be isolated per application, which resulted in multiple copies of the same component version having to be maintained on the system. Additionally COM data was not isolated, limiting this mechanism's usefulness with COM objects.

[0007] At the same time, even if it was possible to permanently bind an application to one version of a shared component, it is not always desirable to do so. For example, a critical security fix may be made to a component, but if an existing application were permanently bound to an earlier version of that component, the application would not be protected by the security fix. In sum, the existing models for sharing components have many problems and shortcomings.

SUMMARY OF THE INVENTION

[0008] Briefly, the present invention provides a method, system and infrastructure that allow an application to run with specified versions of components bound thereto, while allowing the application author and/or component publisher to change the version as desired. In an alternative mode, an administrator may also have input (e.g., the final decision) as to specifying the version to be used. A component is often packaged with other components as an assembly, wherein an assembly is set of one or more component files that are versioned and ship as a unit, and thus as used herein a set of one or more components are also referred to as an assembly, and a component publisher an assembly publisher.

[0009] Each assembly may exist and run side-by-side on the system with other versions of the same assembly being used by other applications. To this end, the application provides an application manifest to specify any desired assembly versions. The application author may also provide (e.g., at a later time) an application configuration that overrides the binding information in the application manifest. The present invention also allows an assembly publisher to provide a publisher configuration that may similarly control which assembly version will be used. In a first alternative mode, the application configuration (when present) is applied after any publisher configuration is applied and thus overrides the publisher configuration's binding information. In a second alternative mode, the order of applying the configurations is reversed, whereby the publisher configuration may change the application configuration's binding override information. In this second alternative mode, the application configuration may have a setting therein that bypasses the publisher configuration, in a "safe" mode of operation. Lastly, (preferably also in this