

144, application programs 145, other program modules 146, and program data 147 are given different numbers herein to illustrate that, at a minimum, they are different copies. A user may enter commands and information into the computer 20 through input devices such as a keyboard 162 and pointing device 161, commonly referred to as a mouse, trackball or touch pad. Other input devices (not shown) may include a microphone, joystick, game pad, satellite dish, scanner, or the like. These and other input devices are often connected to the processing unit 120 through a user input interface 160 that is coupled to the system bus, but may be connected by other interface and bus structures, such as a parallel port, game port or a universal serial bus (USB). A monitor 191 or other type of display device is also connected to the system bus 121 via an interface, such as a video interface 190. In addition to the monitor, computers may also include other peripheral output devices such as speakers 197 and printer 196, which may be connected through an output peripheral interface 190.

[0035] The computer 110 may operate in a networked environment using logical connections to one or more remote computers, such as a remote computer 180. The remote computer 180 may be a personal computer, a server, a router, a network PC, a peer device or other common network node, and typically includes many or all of the elements described above relative to the computer 110, although only a memory storage device 181 has been illustrated in FIG. 1. The logical connections depicted in FIG. 1 include a local area network (LAN) 171 and a wide area network (WAN) 173, but may also include other networks. Such networking environments are commonplace in offices, enterprise-wide computer networks, intranets and the Internet. When used in a LAN networking environment, the computer 110 is connected to the LAN 171 through a network interface or adapter 170. When used in a WAN networking environment, the computer 110 typically includes a modem 172 or other means for establishing communications over the WAN 173, such as the Internet. The modem 172, which may be internal or external, may be connected to the system bus 121 via the user input interface 160 or other appropriate mechanism. In a networked environment, program modules depicted relative to the computer 110, or portions thereof, may be stored in the remote memory storage device. By way of example, and not limitation, FIG. 1 illustrates remote application programs 185 as residing on memory device 181. It will be appreciated that the network connections shown are exemplary and other means of establishing a communications link between the computers may be used.

[0036] Isolating and Binding Assembly Versions

[0037] The present invention is generally directed to binding application programs to configuration (also referred to as policy) determined, isolated versions of components, (including code and/or data), per-application program or the like, in a manner that allows multiple versions of the same component to exist and operate side-by-side on a system. For practical purposes, components are often collected into an assembly, which, when referring to items such as a component, is the lowest unit of storage packaged for activation, distribution and versioning. Rather than deal with individual components, of which there may be a relatively large number, many of the actions regarding components that are grouped together can be handled by referring to their

assembly. For example, rather than list in the manifest 204 the dependencies on a large number of individual components that are packaged together in a component assembly, the manifest may simply list a dependency on the assembly. As used herein, the term “assembly” will refer to one or more components, whether referring to a single component (e.g., one contiguous DLL) or to a plurality of components grouped together.

[0038] Assemblies can be shared, such as when more than one application or the like needs an instance of the assembly’s code. To provide significant flexibility while being transparent to existing and newly-developed applications, the present invention has been implemented in an operating system, with applications being run via the operating system. As will be understood, however, the present invention is not limited to applications and/or an operating system implementation, but rather is capable of being implemented by virtually any mechanism internal or external to executable code (e.g., an application) that needs or wants to use a specific version of an assembly. Note that as used herein, an application program is not limited to any particular type of software product, but includes any executable code such as operating system components, drivers and so on that in turn use other assemblies. Notwithstanding, the present invention will be primarily described with an application that uses assemblies such as DLLs and objects.

[0039] FIG. 2A shows an application program 200 maintained, for example, as an executable file in a file system folder 202 in a non-volatile storage (e.g., hard disk drive 141) of the computer system 100 (FIG. 1). To identify specific versions of one or all of the specific assemblies that the application prefers to use, an application such as the application 200 of FIG. 2A is associated with an application manifest 204. For example, one way in which an application may be associated with a manifest is to store the manifest in the same folder 202 with the application executable, named with the same filenames but with different file extensions (e.g., “.exe” versus “.manifest”). Alternatively, the application manifest 204 may be compiled into the application’s binary code/data, as long as it can be easily accessed. Note that other applications (typically in different folders) may or may not have application manifests associated therewith.

[0040] In general, an application manifest is an XML (extensible Markup Language) formatted file or other suitable file that comprises metadata (e.g., 206) describing an application’s dependencies on shareable assembly versions, (sometimes referred to as side-by-side assemblies), and also includes metadata to describe any privatized assemblies (described below). For example, the application manifest 204 specifies in its dependency data 206 a dependency on a particular version (e.g., v1.0.0.0) of a shared assembly, assembly 208₁, as represented in FIG. 2A by the arrow between blocks 206 and 208₁. Note that the application manifest 204 may also specify dependencies on other assemblies. Further, note that other data structures including a configuration, (described below), which are not ordinarily considered side-by-side assemblies, each may be wrapped as an assembly to thereby obtain the benefits that an assembly may have, such as versioning, naming and so forth. The wrapping of a configuration as an assembly including associated version information is generally represented in FIG.