

version as a default. In this manner, a configuration does not have to concern itself with assemblies whose versions are not to be changed.

[0063] Although it is possible to dynamically interpret the manifest and/or configurations to locate the appropriate version of an assembly each time an assembly is requested, assemblies are requested frequently, and thus it is more efficient to cache the information once. To this end, the activation APIs 300 cause an activation context 302 for that program 200 to be constructed if a valid one does not already exist for the application 300, (for example, when the application is being run for the first time or the activation context 302 exists but its information is invalid). Once created, the activation context 302 is maintained in a persistable binary form for caching. In general, there is an activation context for each application that has an expressed assembly dependency, and each activation context includes one or more mapping tables preferably hashed for quick lookup. As described below, the operating system (a runtime version matching mechanism therein) uses the activation context 302 to determine where to retrieve the version.

[0064] To construct the activation context 302, the activation APIs call (or otherwise include) a binding/initialization mechanism 304, wherein the call is generally represented in FIG. 3A by the arrow accompanied by the circled numeral one (1). If a new activation context 302 needs to be constructed, the binding/initialization mechanism 304 reads and interprets the application manifest 204 as represented in FIG. 3A by the arrows labeled with circled numerals two (2) and three (3). More particularly, as described above, whenever the operating system (e.g., a binding mechanism therein, including the binding/initialization mechanism 304) is asked to perform a bind for a shared assembly, the bind client (e.g., an API called to load a DLL) is required to provide a reference, which describes the requested assembly. The version of the assembly reference may then be altered by a series of configuration resolution stages, as described herein, by which the binding mechanism decides which version of the assembly to return to the bind client. Configuration resolution allows a reference to an assembly constructed at compile/link time to be modified after the application has been deployed, without re-compilation/re-linking of the assemblies involved.

[0065] As represented in FIG. 3A, in the first alternative mode described above, the first phase in bind configuration resolution is publisher configuration, by the arrows labeled four (4) and five (5). In general, publisher configuration allows shared-assembly vendors to make compatibility statements between different revisions of their software. These per-assembly configuration files are wrapped as strongly-named (e.g., COM+) assemblies, and are installed into the global assembly cache 212 (FIG. 2A), e.g., as part of a service-pack-style update. Because publisher configuration assemblies may affect all applications on the system, these assemblies should be installed separately from application installations, otherwise an application may “break” other applications simply through installation.

[0066] In one implementation, a publisher configuration assembly has the same name as the assembly it affects, but with a further extension appended to the name, (e.g., “config”). This publisher configuration assembly has a module reference to an XML configuration file that stores the actual

binding redirect information. Moreover, because assembly metadata contains a hash for the XML configuration file, it is possible to validate the integrity of the configuration file.

[0067] A publisher configuration assembly is created by authoring an XML configuration file (which may have any name), and using an assembly linker tool or the like to create the assembly. For example, as set forth below, a publisher configuration file, version 1.0.0.0, for an assembly named “test” is created, e.g., having an XML configuration file named

[0068] “test.[major].[minor].config”

[0069] where major and minor vary with the version.

[0070] Publisher configuration assemblies normally will be obtained directly from the publisher as part of a service-pack style update, intended to affect all applications on the system. Because there is no direct link between the publisher configuration and the applications it affects, version redirects specified by the publisher configuration file may contain a codebase to the targeted version in the configuration file, otherwise the operating system will not necessarily be able to locate the intended files. Another option is to install the redirected version of the assembly on the machine. Similarly, assemblies targeted by administrator configuration can either be installed into the global assembly cache 212, advertised to the user, or located through a codebase provided in the administrator configuration file.

[0071] In this first mode, the second stage of configuration resolution, after any publisher configuration is applied, comprises resolving any application configuration. As represented in FIG. 3A by circled numerals six (6) and seven (7), if an application configuration exists, the binding/initialization mechanism 304 reads and interprets the application configuration 216. To this end, before a bind to the assembly can proceed, the application configuration file (if any) is accessed, and analyzed. For example, the configuration data may be maintained as an application configuration file, and accessed via an application base (“appbase”) directory (e.g., the folder 202) or other suitable directory. A name/value pair in the application context 302 specifies the name of the configuration file.

[0072] For example, in an “.exe” runtime scenario, the file is named with the same name as the executable, but with a “.config” extension appended thereto, (e.g., “appname.exe.config”). An application author and/or deployer may choose to provide such a configuration file, thereby specifying version redirects for particular assemblies. For example, a configuration file may be written by an application author to specify that references to a common shared-assembly, whether directly provided by the application, or indirectly from a dependent assembly’s dependency, should use a particular version. As another example, once an application deployer is confident that the application works with a newer version of a shared assembly, the deployer can choose to change the application configuration file to automatically use the new version instead of the version set forth in the application manifest. When interpreting the configuration, if a relevant binding redirect statement is found in the application configuration file, the version of the assembly from the original reference is modified accordingly.

[0073] Whenever a binding configuration statement is made in a configuration file, it is the responsibility of the