

the local namespace, then the global namespace). The system ignores duplicated objects with the global namespace enumeration found in the local namespace. Enumeration also ignores the objects marked as deleted from the local namespace and its corresponding global namespace object.

**[0066]** For applications that expect to share resources, the operating system may place the applications in the same virtualization application group (e.g., same isolation identity). Alternatively, the operating system may specify that a particular part of the namespace should not be virtualized. In yet another alternative, the applications specify a portion of their virtualized namespace that other applications may access. The client application specifies the applications for which access is desired. When the client application accesses a shared virtualized namespace, the operating system searches the corresponding exported namespace of the target applications.

**[0067]** In some environments, the operating system may want to have multiple virtualization layers. There might be a virtualization layer per user and virtualization layer per application group. Various ordering of the multiple virtualization layers are within the scope of this invention. In this example, the user virtualization layer takes precedence over the application virtualization layer. Therefore, query requests and open requests for an object first check the current user's virtualization layers, then the current application group's virtualization layer, and finally the global namespace. The operating system returns the first object found or no object if the object does not exist in any of the virtualization layers or global namespace. Likewise for write operations, the operating system first opens the object. If the object exists in the highest precedence layer, then the write operation occurs on that object. If the object does not exist in the highest precedence layer, then the object gets copied into the highest precedence layer and the write operation occurs on the copied object. Create operations occur at the highest precedence layer, though operating systems in some embodiments may allow code to specify a particular virtualization layer as a preference.

**[0068]** Similarly, when deleting an object, the operation occurs at the highest precedence virtualization layer, though operating systems in some embodiments may allow code to specify a particular virtualization layer as a preference. Once the exact object is found, the operating system checks if the object exists in any applicable lower precedence namespace. If the object does exist in a lower precedence namespace, the intended delete object is mark as "deleted" and stays in its namespace. If the object does not exist in a lower precedence namespace, the object is deleted and removed from that namespace. In some configurations, the operating system may delete corresponding object from higher precedence namespaces. The creator of the higher precedence object, however, may designate the object to not be deleted in that case.

**[0069]** When adding an object to a lower precedence namespace, the operating system removes all corresponding objects marked as deleted from the higher precedence namespaces. The search and removal starts from the target namespace up to the next applicable higher precedence layer until the search finds a corresponding object that is not marked as deleted or has searched all the applicable layers.

**[0070]** Enumeration operations account for all the applicable virtualization layers for the context and global

namespace. The enumeration starts from the highest precedence applicable namespace and moves down to the global namespace. As the enumeration encounters an object marked as deleted, the enumeration for that object is ignored in lower precedence namespaces. The enumeration also ignores corresponding objects found previously in higher precedence namespaces.

**[0071]** Internal Object Protection for the Operating System

**[0072]** The operating system creates various objects. Some of the objects are intended for access by applications and others (e.g., internal objects) are only accessible by operating system components. The operating system defines the access rights (e.g., open and read access) for the objects.

**[0073]** In one embodiment, internal operating system objects should only be accessible by internal operating system components. To prevent external code from accessing the internal objects, the operating system marks the internal object for access only by internal operating system components. The runtime objects, running as internal operating system code, get associated with the internal operating system identity. Therefore, when a runtime object attempts to access an internal object, the operating system checks if the runtime object is associated with the internal operating system identity. If the runtime object has the internal operating system identity, the operating system allows the access. Otherwise, the operating system implements appropriate action. Appropriate action may include rejecting the access, logging the access attempt, etc.

**[0074]** When an internal operating system component creates an object, the object is marked for access only by internal operating system components unless the creator specifically marks the object as available for external access. The operating system may mark internal objects offline using resource information from a store, manifest, configuration file, digital signature, etc.

**[0075]** Some operating system components are classified as middleware components, which means that even though they are part of the operating system, they should not access internal objects except for some special expectations that external applications are also allowed to access. The operating system in one embodiment would like the middleware components to stop using the special exception internal object and migrate over to external objects. To address this issue, the operating system associates a middleware application identity with the middleware components. The special exception internal objects are marked additionally with the deprecated attribute. When a middleware component accesses the deprecated object, the system responds with the appropriate action such as audit the access and/or block the access. The middleware deprecated resource detection may be applied more generally for deprecating external objects or other external objects or other internal objects.

**[0076]** Removal of Application Programs

**[0077]** Referring next to **FIG. 6**, an exemplary flow chart illustrates a method for performing an application undo using application identity information. In particular, the flow chart illustrates a method of completely removing an installed application program from a computing system via an application identifier associated with the application program and components (e.g., files and resources) thereof.