

In a further alternative embodiment of the invention, a pinned version of a target object that is related to a source object is returned. A pinned object is a particular version of an object that has been specified as the default destination object in a relationship.

Workspaces in an Object Repository

The previous section described versioning of objects in an object repository. This section will describe embodiments of the invention that support workspaces within a repository that can be used to support working with versioned objects.

A system level overview of an embodiment of the invention supporting repository workspaces is shown in FIG. 9. The system includes a repository 250, one or more workspaces 908, a version aware application 902, and a non-version-aware application 904. Repository 250 is described in detail above with reference to FIG. 2, and by way of example includes repository objects 906. The objects 906 are versions 1–3 of an object X, and version 1 of an object Y.

Each of workspaces 908 is a logical repository session. However, unlike an ordinary repository session, a workspace is persistent. In other words, workspaces exist across repository sessions.

Versions of repository objects can be explicitly added to a workspace, thereby making them visible in the workspace. In the exemplary system shown, workspace #1 contains version 1 of object X, workspace #2 also contains versions 1 of object X, and workspace #3 contains version 3 of object X and version 1 of object Y. Objects can also be explicitly removed from the workspace. A version can be added to many workspaces. However, there can be at most one version of an object in each workspace. Thus, a workspace is a single-version view of a subset of the repository database.

Version-aware application 902 is an application that has been designed to take advantage of the versioning capability provided by repository 250. Version-aware application 902 establishes a session S with repository 250. In one embodiment of the invention where the repository is Microsoft Repository, application 902 accesses the repository via an IRepository2 interface. The IRepository2 interface supports versioning. After a session S has been opened, the application's context includes the entire repository. The application can then access a workspace W in S. In the example shown, application 902 has established a connection with workspace #1, using an IWorkspace interface. Workspaces 908 support the session interfaces, so a client can use a workspace as a logical, or virtual, repository session. Thus, a workspace can be viewed as a wrapper for the base repository which provides a context and filter mechanism. Operations on workspaces are delegated to the base repository object, with appropriate filtering applied to a subset of the object and relationship versions present in the workspace 908.

By executing operations in the context of a workspace instead of S, the client only sees objects that are in (i.e. were added to) the workspace, relationships on such objects, and those relationships' target objects that are also in the workspace. However, if required, the application can use S instead of W to access the entire repository.

An object (i.e., version) in a workspace can be updated only after it is checked out. It can be checked out to at most one workspace at a time. The checkout/checkin methods amount to long-term locks that are stored in the repository database and are used to implement long transactions. A typical long transaction would add some versions to a

workspace, check out the ones to modify, perform updates (under short transaction control), check them back in, and optionally freeze them. This has the benefit of controlling and managing changes to objects in the repository.

Non-version-aware application 904 is an application that has been designed such that it is not capable of recognizing multiple versions of an object. The application 904 may be one that was designed to access a repository before versioning capability was added, or it can be an application that does not require versioning, but wants to access objects in a repository containing versioned objects. In an embodiment of the invention where the repository is Microsoft Repository, the non-version-aware application is designed to use the IRepository interface. This interface does not support versioning in the repository.

In the example shown, non-version-aware application 904 has established a connection to workspace #3. The non-version-aware application 904 accesses (non-versioned) objects using a repository session as its context. The application 904 can still use session interfaces on those workspace objects, so no other changes to the application 904 are required. The resulting application only accesses those objects that are in the workspace.

Thus, the workspace's support of session interfaces provides the backwards compatibility necessary for non-version-aware applications such as application 904. This provides a way for non-version-aware applications to gain the benefits of long term locking provided by workspaces by opening a workspace. In addition, the workspace interface can be modified to add major new functionality (workspace scoping) while avoiding the major change in the programming model that would otherwise be necessary to set and reset scope.

After establishing a workspace connection, applications such as applications 902 and 904 can add versions to a workspace. In an embodiment of the invention where the repository is the Microsoft Repository, versions are added to a workspace using the IWorkspace.Contents.Add method. As noted above, a workspace includes a single version of each object. If a version of an object already present is included in the workspace, the newly included version replaces the previously included version in the workspace.

In addition, object versions can be removed from a workspace. In an embodiment of the invention where the repository is Microsoft Repository, objects are removed using the IWorkspace.Contents.Remove method. It is desirable that a version cannot be removed from a workspace while it is checked out to that workspace.

Each version in a repository maintains a context pointer. This context pointer indicates whether or not the version object is associated with a workspace or workspaces, and if so, which workspaces. The context pointer simplifies the addition of objects to a workspace, and also allows an application to copy or compare an object between workspaces, or between a workspace and the repository. The first advantage of an implicit context pointer is the simplification of the API (Application Programming Interface) for programs that manipulate versions vs. requiring the program had to explicitly specify workspace context on every object reference. The ability to add objects to workspaces, compare objects in workspaces and/or the repository, copy objects between workspaces and/or the repository etc. is more in the nature of a requirement for the API. By having separate ruining object instances, each with its own context, the system disambiguates cases where the same version of an object must be manipulated in multiple contexts simulta-