

## VERSIONS AND WORKSPACES IN AN OBJECT REPOSITORY

### RELATED FILES

This application claims the benefit of U.S. Provisional Application No. 60/122,939, filed Mar. 5, 1999, which is hereby incorporated herein by reference.

### FIELD OF THE INVENTION

This invention relates generally to object repositories, and more particularly to maintaining versions and workspaces in an object repository.

### COPYRIGHT NOTICE/PERMISSION

A portion of the disclosure of this patent document contains material which is subject to copyright protection. The copyright owner has no objection to the facsimile reproduction by anyone of the patent document or the patent disclosure as it appears in the Patent and Trademark Office patent file or records, but otherwise reserves all copyright rights whatsoever. The following notice applies to the software and data as described below and in the drawing hereto: Copyright© 1999, 2000, Microsoft Corporation, All Rights Reserved.

### BACKGROUND

The number of applications that use object-oriented techniques and languages continues to increase at a rapid pace. This growth in object-oriented applications has resulted in a corresponding growth in the use of object databases and repositories. Object databases and repositories provide for the persistent storage of object data in the same way that a conventional database provides for the storage of tables containing data. Object repositories and object-oriented databases are similar in that they both store data in an object format, however repositories in addition typically provide for the storage of metadata, that is, data about the object data, along with the object data. This metadata typically comprises information such as object formats and interfaces, object versions, check-in/check-out dates and personnel, database schemas, etc.

An object, as is known in the art, is a data structure that has a persistent state. The persistent state consists of attributes, which comprise scalar values and object references. A scalar value is a value such as a string, integer or boolean. An object reference specifies one side of a binary relationship between two objects that refer to each other. In other words, the reference is to another object, which in turn refers back to the referring object. Each attribute is identified by a name, and each attribute has a data type. The data type for an attribute identifies either the type of scalar value for the attribute or the type of relationship defined by the attribute.

In addition to attributes, the state of an object includes structures. A structure contains a group of attributes that are organized according to a particular data structure. This data structure can be a collection (also referred to as a set), sequence, array, table, or record structure. Each structure conforms to a named structure type, which defines the particular data structure (collection, sequence, array, etc.) and the types of attributes the structure can contain. Like any attribute, an attribute in a structure can be a scalar value or object reference. A structure that contains object references is called an object structure.

Each object conforms to one or more types, where each type is identified by a name. An object type defines a set of

attribute types and/or structure types that an object of the given type can contain.

An object is typically an instance of a class. A class is a body of code that implements one or more object types. The class includes code to produce new objects of each type that it implements and code to perform various operations on objects of types that it implements and on attributes and structures of such objects. The types of operations performed vary depending on the class, and generally include read and write operations for the attributes and structures of an object.

The life cycle of a software development project typically includes multiple design changes, both before and after release of the software. These design changes include changes in the definition and relationships between objects. As a result it is desirable for object oriented environments to provide the ability to version objects and relationships between objects in the repository.

Previous systems have provided rudimentary versioning capability. In these systems, when a new version of an object is created, a copy of the old version is made, and changes are applied to the copy, which becomes the new version. While this mechanism does provide versioning ability, it has significant disadvantages. First, copying objects is very inefficient in terms of both time and computer resources. Each copy consumes memory, which can be costly given that a typical project will have many different objects, with each object having multiple versions.

A second drawback relates to the versioning interface. It is generally the case that multiple software applications will require access to an object repository. These applications may or may not be "version-aware." In other words, some applications may recognize that various versions of objects exist in the repository, and have interfaces designed to work with the various versions. These applications are known as version-aware applications. Other applications may be designed assuming that one, and only one version of an object exists. These applications are therefore not version-aware. Object repositories implemented by previous systems either provide a version-aware interface or an interface that is not a version-aware interface, but not both.

A third drawback relates to management of relationships between versions of objects. Previous systems apply an all or none approach to relationships between versions of objects. In other words, either all of the relationships from a previous version are included in the new version, or none of the relationships are included. This is undesirable, because it results in the need for a manual fixup of the relationships whenever a new version is created.

Therefore, there is a need in the art for a system to provide efficient versioning for objects in a repository. The system should only copy object properties and relationships when necessary. Furthermore, the system should provide a mechanism to control whether or not relationships are copied when a new version is created. In addition, the system should provide interfaces to applications that are version-aware, and those that are not version-aware.

### SUMMARY

The above-mentioned shortcomings, disadvantages and problems are addressed by the present invention, which will be understood by reading and studying the following specification.

The systems and methods presented maintain versions and workspaces in an object repository. One aspect of the system is that objects and properties are only copied when