

custom object **472** “wraps” a generic repository object **474** in order to provide extended functionality through interfaces. In an interface-based binary object model with binary extensibility, the repository functionality remains available by interfaces accessed through the binary convention while type-specific functionality may be added, including specific executable software in order to support any required interfaces. It may be noted that the functionality of the repository object **474** may be directly available in the new custom object as shown by the extended access point **476** or may be completely internal to the custom object **472** with exterior client having access only to the type specific functionality through the access point **478**. The directly accessible functionality corresponds to the COM object model aggregation extensibility mechanism while the non-directly accessible wrapping corresponds to the COM object model containment extensibility mechanism. With some tool information models, repository activity and interaction may be completely hidden from the client.

Regardless of the extensibility mechanism, the repository object **474** will operate in the same way by interpreting information from the database **480** having type definition information **482**. In the currently preferred embodiment this type information is organized into a COM object hierarchy that can be accessed as necessary. Additionally, the generic repository object will read and write object state information in interface tables **484** as explained previously, including properties on newly added interfaces as defined in the type definitions.

In order to show an example of extension, FIG. **16** shows the address book example as initially shown in FIG. **4** except the contact class includes a second interface, IDial, with a new method DialNumber. In FIG. **4**, the contact class **134** supported only the IContact interface **136** while the contact class **486** of FIG. **16** supports both the IContact interface **136** and the IDial interface **488**. The DialNumber method **490** of the IDial interface **488** will access the phone property **162** of the IContact interface **136** and dial the number.

FIG. **17** shows the changes to the type definition object hierarchy that occurs by adding the new interface definition object **492** representing the IDial interface. Note also that there will be an additional IsScopeFor relationship **190** from PersonalInfoManager **188** to IDial **492** and Implements relationship **196** from CContact **194** to IDial **492**. Finally, FIG. **18** shows the addition of another browser (version **2**) **494** that implements the dialer interface and accesses the address book and contact repository **412**.

FIG. **19** is a diagram showing the custom object created by wrapping a generic repository object of type contact. The repository object supports the IContact interface **502** and the new contact object supports a new interface, IDial **498**. The COM object shown in FIG. **19** contains the outer contact object **496** that is instantiated using a DLL server called “Contact.dll.” The software in contact DLL allows the instantiated object to support the IDial interface **498** with the DialNumber method **500**. Using COM aggregation, the IContact interface **502** and all its associated methods supported by a generic repository object of type contact are made available to outside clients using the COM binary convention. The COM server for the repository object is called “reobj.dll.” By using COM aggregation, the IContact interface **502** is directly available or passed through the contact object to be made available to a client. Alternatively, a COM containment wrapping could occur which would require the contact object to access the IContact interface in order to support whatever functionality the contact object may support through the IDial interface or other interfaces.

Because type information has been entered into the repository type definition model for the interface IDial, should there be any properties associated therewith, they, too, would be stored into the SQL database.

The object state repository as described for the currently preferred embodiment may be created by a computer program product directing and controlling a general purpose computer. The computer program product will consist of some medium such as magnetic disks or CD-ROM having computer readable program code means to configure the computer. The program code means will configure or cause the computer to make the object state repository as described as well as the objects themselves. Furthermore, the program code means implementing the present invention will interact with existing program code means and additional program code means as a client in order to fully implement the configuration of the general purpose computer.

The present invention may be embodied in other specific forms without departing from its spirit or essential characteristics. The described embodiments are to be considered in all respects only as illustrated and not restrictive. The scope of the invention is, therefore, indicated by the appended claims rather than by the foregoing description. All changes which come within the meaning and range of equivalency of the claims are to be embraced within their scope.

What is claimed and desired to be secured by U.S. Letters Patent is:

1. A method of forming a repository for storing the state of a plurality of objects and for permitting retrieval of the stored state for use in later created objects that is independent of any underlying language used to form the objects, said method comprising the steps of:

forming one or more interface means, each interface means comprising (a) one or more properties that are capable of describing the state of an object, (b) one or more method means for accessing and modifying said properties and exposing other behavior, and (c) an interface identifier means for uniquely identifying an interface means so as to enable accessing of said one or more method means;

forming one or more class means, each consisting of executable code means that implements one or more said interface means and having a unique class identifier means;

forming one or more objects as instances of one or more said class means, the properties of the interface means implemented by said class means defining the state of the object, each object accessible through at least one of said one or more interface means; and

forming a repository of stored object states for each of said one or more objects by including in said objects a database interface means, said database interface means storing for each object: (a) said class identifier means used to instantiate said object, (b) said properties for each one or more said interface means defining the state of the object, (c) said interface identifier means for each interface means implemented in said object, and (d) a stored state identifier means for identifying the stored object state, thereby enabling said stored object state to be retrieved from said repository for use in a later created object, and thereafter enabling further use of said later created object in the state defined by said properties.

2. A method as recited in claim **1** wherein said state retrieval occurs implicitly during object navigation.

3. A method as recited in claim **1** wherein said state retrieval occurs explicitly by reference to said stored state identifier means.