SYSTEM, METHOD, AND ARTICLE OF MANUFACTURE FOR ADDING OBJECT SERVICES TO A BINARY CLASS IN AN OBJECT ORIENTED SYSTEM

BACKGROUND OF THE INVENTION

1. Field of the Invention.

The present invention relates to information handling systems, methods and articles of manufacture, and more particularly to information handling systems and methods for facilitating code reuse in an object oriented system.

2. Prior Art.

In the prior art there are many techniques for improving object oriented programming systems (OOPS).

The following are examples of the prior art.

U.S. Patent 5,093,914 generally teaches a method used by a digital computer in controlling execution of an object oriented program to effect a defined action, for example, stopping the program when a specified virtual function is invoked on a specified object during execution of the program.

Although the patent generally deals with methods for controlling execution of object oriented programs, the patent does not teach nor suggest automatically inserting object services into binary classes in an object oriented system as is taught and claimed herein with reference with the present invention.

U.S. Patent 5,343,554 teaches a computing system in which problems are solved by a process which includes creating objects of

first and second classes wherein the second class objects are externally invocable and where the externally invocable sub-class objects include an indication of an internally invocable sub-class object and executing the class of objects wherein one externally invocable sub-object invokes the internally invocable sub-object and a new object of the first class is generated in response to the results.

Although the patent deals generally with the use of object oriented systems including classes and sub-classes for solving problems, the patent does not teach nor suggest automatically inserting object services into binary classes in an object oriented system as is taught and claimed herein with reference with the present invention.

There is a need to automatically add object services features, for example, persistence, recoverability, concurrency and security to a binary class. Sometimes the source code of a class is not available for modification. Even when the source code is available, a considerable reprogramming effort is required to add the object services features.

A user should be able to add object services features in a way that does not increase the size of the users class diagram with many variations on the original class. The user should be able to specify these object services features as constraints along with constraints that describe the function of the class when searching for the class.

An overall goal in object oriented programming systems is the development of program objects which can be reused easily.

The importance of binary code over source code increases with the degree of reuse. A main purpose of object oriented technology is code reuse. For object oriented technology to achieve large scale success, binary code reuse is essential. As the state of the art moves towards applications built from object formats which can be tailored and plugged into each other, binary code is critical.

The Object Management Group is currently defining a set of interfaces for object system services named Common Object Services.

SUMMARY OF THE INVENTION

It is an object of the present invention to automatically insert object services such as persistence, concurrency, recoverability, security and distribution into binary object classes.

Accordingly, a method for automatically inserting object services such as persistence, concurrency, recoverability, security and distribution into binary classes in an information handling system employing object oriented technology, includes the steps of registering a description of a class in a name service context, searching the name service context for a class object referred to as a factory with desired constraints on properties, whether the object is operable in a predetermined operational environment, where the search excludes object services constraints, for classes which are found by the search, first select those classes which already support the object services constraints, create in the operational environment a class object with the desired object services features, and adding a particular object services feature if the original class provide the desired feature, and returning the class object to the caller which can then use the class object to create multiple instances thereof.

It is an advantage of the present invention that no reprogramming effort is required for insertion of object services, source code is not required and, in fact, is not desired, and the user's class diagram and library do not increase in complexity or size.

BRIEF DESCRIPTION OF THE DRAWINGS

Figure 1 is a block diagram of a system for executing the method according to the present invention.

Figure 2 is a block diagram showing the operating system platform and system object model program supporting the present invention.

Figure 3 is a flow chart of the method in accordance with the present invention.

Figure 4 is a block diagram of an object name service in accordance with the present invention.

Figure 5 is a block diagram of a factory finder naming context example in accordance with the present invention.

Figure 6 is an object diagram showing the use of a BeforeAfter metaclass for providing object services features in accordance with the present invention.

DETAILED DESCRIPTION OF A PREFERRED EMBODIMENT OF THE INVENTION

Referring now to Figures 1 and 2, on information handling system 10 and an operating system environment for the present invention will be described.

Information handling system 10 may be a graphics work station or the like with a very powerful processor 12, a storage system 14, a system bus 16, display sub-system 18 controlling a display device 20, a cursor control device 22, and an I/O controller 24, all connected by system bus 16. The information handling system 10, shown in Figure 1 may be operated by a commercially available well known multitasking operating system such as OS/2® (OS/2 is a registered trademark of International Business Machines Corporation). Among many tasks which the OS/2 operating system controls an operating information handling system 10, is execution of a program SOMObjects, which is a commercially available product of International Business Machines Corporation.

The method and article of manufacture of the present invention may be incorporated in the SOMObjects program.

The object services supported by the object platform include the following:

Externalization, which is a technique for moving data into and out of an object for purposes such as copying an object across address bases, moving an object across address bases, caching an object in a client address base, passing an object by value across address bases or storing and restoring objects persistent state;

Persistence allows an object's state to be stored into a datastore mechanism such as a database or file system.

Object transaction service provides a two-phase commit coordination among recoverable objects, where the transaction service is responsible for passing the transaction context between a caller and a callee, reliably accepting registration of recoverable resources, and coordinating two-phase commitment;

Concurrency service provides a lock manager which controls locks on behalf of either transactions or threads, where in dealing with transaction locks the concurrency handles locks inheritance for nested transactions and supports explicit locking and unlocking for applications where lock contention is a bottleneck;

Object name service (See Figure 4) allows an object to be bound to a directory (a naming context) with a human friendly name;

A special case of the object name service is a factory finder (See Figure 5), wherein a class description can be bound to a directory along with properties describing class, including functional features, class library, cost, instruction set type, etc.

To support binary class reuse, the contract between the following two human roles must be addressed:

- The object provider designs the class external interface, implements its methods and packages it as a product including its IDL (interface definition language) description of the external interface and its DLL (dynamic link library) containing its binary code.

- The object consumer buys the above binary object and adapts it for use in his application. This includes,

among other things, adding whatever object services necessary.

Object services are supported by the object platform (e.g., SOM). However, the binary class must cooperate with the object platform to enable these object services to be applied to the binary class. This requires some work by both the object provider and consumer. This invention minimizes the work required by both of these human roles.

In accordance with the present invention, the object provider adds the following in the binary class IDL and DLL:

- An external data schema for the object that includes the external view of all data needed for the object's methods to work correctly.

- The externalize_to_stream() and internalize_from_stream() methods that move data out of and into the object.

This is used in conjunction with the Externalization Service to move or copy an object, caching an object in a client address space, pass an object by value, store/restore an object's persistent state from/to a datastore, object query, recoverability, initialization, export/import, and other object services.

The object provider may also provide a method to check the consistency of the object, which is called during the transaction commit process. The object provider usually does nothing else to provide object services.

In accordance with the present invention, the object consumer requests the object platform to add various object services to the

binary class, producing a new class that fits more precisely into the object consumer's application. The enhancements to the binary class to facilitate object services include the following:

- Making the object recoverable, so it can participate in a transaction involving multiple recoverable objects coordinated the two-phase commit process of the Transaction Service. This includes registering the object with the transaction manager of the Transaction Service in a before method.

- Making the object concurrent, so that multiple users of the object can be serialized correctly. This involves getting a lock on the object using the Concurrency Service in a before method.

- Making the object persistent, so that its state can be saved in a datastore at commit and later restored to where it was earlier in a before method. If a database schema exists where the object's data is to be stored, the object consumer provides a schema mapping between the object's external data schema and the database schema.

- Making the object secure, so that object/method granularity access control can be enforced. This involves checking access control in a before method.

In general, it is expected that the class provider, the entity which provides the abstracted data type referred to herein as a class, will construct the class without concern for object services. When a user wants an instance of a class, the combination of desired services are specified.. A new binary class

is constructed from the original which adds the desired object services.

Two mechanisms are employed in accordance with the present invention to accomplish the automatic insertion of object services.

First, a new class (sub-class) is created by mixing in the original class and a special class that provides the additional function.

Next, a BeforeAfter metaclass (See Figure 6) inserts a method before and a method after the methods of the original class. Further description of the BeforeAfter metaclass may be found in a paper entitled "Composition of Before/After metaclasses SOM" by Forman, Danforth, and Madduri, published at OOPSLA 94. The identified paper is incorporated by reference herein.

As an example, if a user desires a concurrent, persistent and recoverable version of class A, to each of class A's original methods, the following needs to be added:

1. The before method ensures the object is locked;

2. The before method ensures the object is registered as a recoverable resource with the transaction manager;

3. The before method ensures that the object's state is loaded from its datastore if its state is out of date

If information is available which indicates whether the method is read only, the above identified before methods can be optimized. The following methods also must be added:

4. Pre-prepare: Store the object's state in its datastore.

5. Prepare: Call the object's method for checking integrity constraints.

6. Commit: Mark that the object's state as out of date and release locks.

7. Abort: Mark that the object's state as out of date and release locks.

The object services constraints that are specified as part of the predicate when a factory is found in a Name Service (NS) ExtendedNamingContext (ENC) for factories are:

name: persistent
value: A boolean indicating whether the object should be persistent, which means that the object's state can be saved and later brought back to where it was even if the object's server goes away.

name: PO
value: If persistent==TRUE, a boolean indicating whether the PersistentObject (PO) interface is wanted.

name: recoverable
value: A boolean indicating whether the object is recoverable, which means that it can participate in transactional two-phase commit.

name: concurrent
value: A boolean indicating whether the object is locked.

name: secure
value: A boolean indicating whether the object is protected by access control checking.

name: caching
value: A boolean indicating whether the object's state can be cached within its ORB proxies.

The location constraint that is specified as part of the predicate when a factory is found in a NS ENC for single-class factories is:

name: InstanceManager
value: The caretaker of the object, which provides the object's identity, its activation, caching in memory, etc.

Information about whether an IDL operation on an object might update its persistent state can allow a significant optimization for some implementations of persistence, concurrency control and recoverability. For persistence, this can be used to determine whether the method is going to make the object's persistent state out of sync with that in its datastore. For concurrency, this can be used to determine whether to get a read or write lock before allowing the operation to be executed. For recovery, this can be used to determine whether to make a copy of the object's state before allowing the operation to be executed. This information should be stored in the IR. The per IDL operation option is:

name: CONST
value: A boolean indicating whether the operation may update the object's state. The default is FALSE

(may update). This information should be stored in the IR.

Referring now to Figure 3, the method according to the present invention will be described.

A class description is registered in a name service (NS) context. This description typically includes a description of the class' function, for example a speaker driver, its characteristics, for example output frequency is 10 CPS, its cost, what object services features are already supported, what processor type it is compiled for, for example Intel X86, and the file name for its load module.

Next, a search is made of the name service context for a factory with the desired constraints on properties, for example a speaker driver that can provide an output to a frequency less than 20 CPS and a cost of less than $5.00. The search would also include a determination of whether the object can run in the machine where the object's InstanceManager lives, for example, Intel X86 binary. Object services constraints such as persistent and recoverable would be excluded from the search.

For classes that are found in the above search, a first selection is made of those classes which already support the object services constraints. Next, a class object is created in a desired server with the desired object services features such as persistent and recoverable. If a particular object services feature is provided in the original class, no action is taken. However, if the feature is not provided, it is added to the new class object.

Lastly, the new class object is returned to the caller who then can use it to create multiple instances of the class.

Many of the object services features can be provided by mixing the original class with the newly created class. Using run time inheritance, the original class' implementation may be overridden by the newly created class. The creation of the new class object with the desired properties can be accomplished in a manner that is transparent to the user.

Many of the OS features can be provided by using the BeforeAfter metaclass as illustrated in Figure 6. A metaclass can be created as a subclass of BeforeAfter with the before and after methods overridden. A class with this new metaclass has the before method and after method executed before and after each method of the class. A single pair of before/after methods is driven by a usage object that describes what object services to add.

For object systems that do not support inheritance and before/after metaclasses, a delegate class can be automatically constructed instead. The delegate class instance object references the original class instance object. The delegate class has the same additional methods added that were described above (e.g., preprepare, prepare, commit, abort). The delegate class also supports all of the operations of the original class but implements each method as follows:

```
DelegateClass :: methodA (...) {
     before method work;
     OriginalClass_instance_object->methodA (...);
     after method work;
}
```

It will be appreciated that although a specific embodiment of the present invention has been described herein for the purposes of

illustration, various modifications may be made without departing from the spirit or scope of the invention.

Accordingly, the scope of this invention is limited only by the following claims and their equivalents.

WHAT IS CLAIMED IS:

1. A method for automatically inserting object services into binary classes in an information handling system employing object oriented technology, comprising the steps of:

registering a description of a first class in a name service context;

searching the name service context for a class object referred to as a factory;

creating a second class object with one or more object services features;

adding one or more object services features to the second class object if the first class did not provide the desired features; and

returning the second class object to the caller.

2. A method according to claim 1, wherein said searching step further comprises the steps of:

identifying constraints on object properties;

determining whether the object is operable in a predetermined operational environment;

excluding object services constraints, for classes which are found by the search; and

selecting those classes which support the identified object properties constraints.

3. A method according to claim 1, further comprising the steps of:

creating a class comprising a before method;

executing said before method on said second class object.

4. A method according to claim 1, further comprising the steps of:

* C L M S *

creating a class comprising a before and an after method;

executing said before and after methods on said second class object.

5. A computer readable medium comprising means for automatically inserting object services into binary classes in an information handling system employing object oriented technology, comprising:

means for registering a description of a first class in a name service context;

means for searching the name service context for a class object referred to as a factory;

means for creating a second class object with one or more object services features;

means for adding an object services feature to the second class object if the first class did not provide the desired feature; and

means for returning the second class object to the caller.

6. A computer readable medium, according to claim 5, wherein said searching means further comprises:

means for identifying constraints on object properties;

means for determining whether the object is operable in a predetermined operational environment;

means for excluding object services constraints, for classes which are found by the search; and

means for selecting those classes which support the identified object properties constraints.

7. A computer readable medium, according to claim 5, further comprising:

means for creating a class comprising a before method;

means for executing said before method on said second class object.

8. A computer readable medium, according to claim 5, further comprising:

means for creating a class comprising one or more before and after methods;

means for executing said one or more before and after methods on said second class object.