# VIRTUAL SINGLETON

Author: Gerard Meszaros
Object Systems Group
Gerard@osgcanada.com
Phone: 403-210-2967

## Context

Your are building a multi-user system or application that will provide service to many users concurrently, or you are converting a single-user system to support multiple users. The system may be either a multi-client server or a thin-client system such as a terminal server or mainframe.
Additionally, you may be retrofitting security onto an existing multi-user system.
Multi-user systems add a host of problems beyond those faced by single-user systems. These include

- the need to restrict the activities of users to those they have permission to carry out,
- the need to prevent accidental interactions or "race conditions" between the activities of the various concurrent users

Many of these activities require information about the user on who's behalf processing is taking place to ensure that their activities are kept separate.

## Problem

How do you access information specific to a particular user's session context from software that must be shared by various parts of the application?

## Forces

- The sessionContext is often required by infrastructure software in the deepest parts of the application. Passing the necessary information explicitly requires modifications to all the parts of the application between the parts aware of the sessionContext and the infrastructure that requires the information. This "signature pollution" adds significant overhead to each level of method call by adding one or more extra parameters and also requires that the application know which information will be required by the infrastructure software.
- Forcing the application to be aware of the needs of the multi-user part of the infrastructure increases the complexity of the application software which distracts the developer from the task of understanding the business processes being automated by the application.
- Using a Singleton [GOF] would provide an easy way for software deep in the system to access the information via a well-known global object, but the information to be accessed is unique to each user thus a normal singleton cannot be used.

## Solution

Define a Virtual Singleton which can be used by infrastructure software to access information about the current sessionContext. The Virtual Singleton is a global variable which holds an object which acts as a gateway to per-session information about the current user which is stored in a dictionary indexed by the current session, process or thread. Name the global variable as though it were truly a singleton. This hides from the application software (and its developer) whether a single or multi-user implementation of the infrastructure is behind the interface.

## Resulting Context

The infrastructure on which the application is built becomes more complex, but this should only have to be built or acquired once. Most application code is simplified because it does not need to be aware of the need for per-sessionContext information. This can be left to the parts of the system that care about this information. This simplifies the application software by removing the need to be aware of multi-user data access issues. Other issues related to multi-user servers are discussed in the Related Patterns section.

## Related Patterns

Virtual Singleton is an extension to the Singleton pattern to allow it to be used in multi-user systems for per-sessionContext information. The signature of the Virtual Singleton should be identical to the equivalent Singleton but the implementation deals with the issue of keeping separate information for each sessionContext. For example, the class method SoleInstance provided in Smalltalk by Singletons to return a reference to the only instance of the singleton could return the instance of the Virtual Singleton specific to the current sessionContext.

"Thread-Specific Storage for C/C++" is a C/C++ solution to the problem solved by Virutal Singleton. The solution described in [Schmidt97] combines aspects of Virtual Singleton and Property List (or Registry) to act as a central facility for storing various bits of sessionContext-specific information. It is not necessarily interface-compatible with the single-context version of the equivalent functionality.

This pattern is one of a set of patterns used to provide application developers with a much simpler environment in which to implement business logic. This environment strives to hide as much of the multi-user nature of the system as possible from application software and its developers. Other aspects of multi-user behavior needs to be handled to achieve a true "virtual single-user system" capability include:

- Object Concurrency protection (mutual exclusion for updates to objects)
- Private Object Copies for transactional changes prior to transaction commit.

.

Self-Protecting Object [undocumented] is used to provide concurrency management when objects are shared between user sessions. It replaces the concept of database locks in the object world.
Per-SessionContext Object Instance [undocumented] is used to provide each transactional context with its own copy of any objects which it modifies.

## Implementation Notes

The following describes ways to implement the Virtual Singleton in both OO languages and procedural langauges. In all cases, we care about the sessionContext using the current process or thread in which we are executing. This definition of "current" works equally well whether we are running in a single processor system or a multi-processor system and whether we are using a thread-per-session or thread-per-request model of distributed processing. We use the identity of the process/thread in which the Virtual Singleton is accessed as the key to locating the right copy of the Virtual Singleton. When using a Thread-Per-Object model (a.k.a. Active Object), we must be able to determine the sessionContext which initiated the request currently being processed

### *Object Systems*

There are two basic approaches (plus at least one hybrid) to implementing a Virtual Singleton in an Object Oriented programming language. These are:

1) Pure Object Approach: Use a real Singleton which looks up the current user or thread and uses it to determine to which per-sessionContext object it should delegate the request. The Singleton and the sessionContext-specific objects should both implement the same interface that a true singleton would have provided. Structurally, this looks like a limited form of Composite, but behaviorally, it is more of a Selector or Dispatcher.

2) Updated Global Variable Approach: Use a global variable which points directly to a per-sessionContext object. The global variable must be updated whenever a sessionContext swap is executed. This requires "hooking" the process scheduler to update the variable and it only works in single-processor systems. In multi-processor systems, the global variable would have to be at least one instance per processor.
3) Hybrid Approach: A hybrid of these two approaches involves extending the run-time environment to provide for per-sessionContext (either per-process or per-thread) variables.

Note that in languages like Java which do not have global variables, we need to create a Class to act as the Singleton. This class should have the same name as we would have used for the global variable such as TheXxx or CurrentXxx. It would have a class variable which points to the "current" instance which might be of the same class (approach 2) or an interface-compatible class (approach 1). Each method on the class side would delegate to the corresponding instance method of the instance referred to by the class variable..

### *Procedural Systems*

When implementing the Virtual Singleton in a procedural language, we can use:
1) A global procedure which does the user lookup and continues processing using that data as a context.
2) Use a global variable to point to a particular record in a data structure which contains the bits of information we need to access about the user. Various procedures and functions could use the global variable as an implicit argument. The global variable must be updated whenever a sessionContext swap is executed. This requires "hooking" the process scheduler.
3) A variant of these two approaches involves extending language definition and the run-time environment to provide for per-sessionContext (either per-process or per-thread) variables. A variable declared with the "per-context" modifier would be guaranteed to have the desired semantics.

The best approach is determined largely by whether we have control over the runtime environment (required for approaches 2 and 3).

### *Deallocating Storage*

A key issue when storing information for each session (as opposed to just accessing information such as the sessionId) is when to throw away the information. (Creation is not so much of a problem since one can use Lazy Initialization pattern to create a new entry upon the first access on behalf of a user not already known.) Ideally, the global singleton would use the Observer Pattern to be informed whenever a sessionContext is terminated. It could then delete the corresponding entry in its dictionary. An alternative is to use an "aging" algorithm (i.e. timeouts) to discard entries that have not be accessed in last x minutes. This necessitates timestamping the dictionary entry each time it is accessed. This approach must be used when clients do not maintain a connection to the server between accesses such as in Web-based applications.

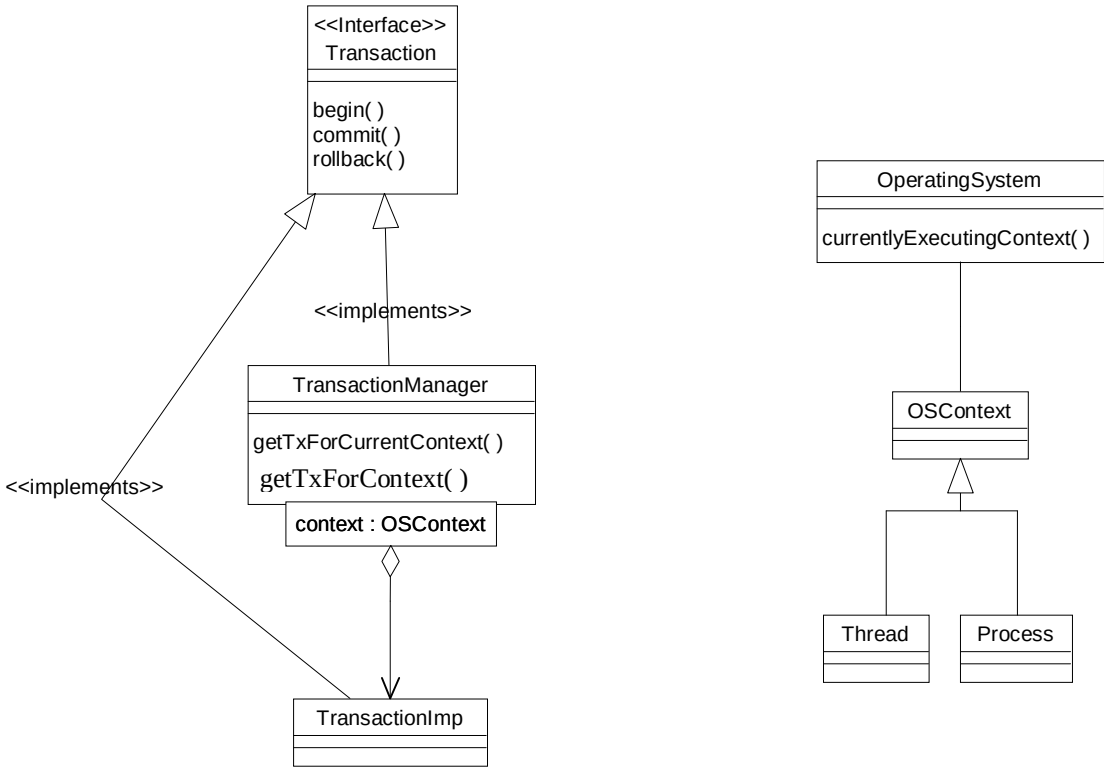
## Example

The following diagrams illustrates the Pure Object Approach to implementing the Virtual Singleton. We focus on the behavior when the Virtual Singleton is sent a message. (The other object approaches would be hard to differentiate from a normal singleton; most of the difference is hidden somewhere in the bowels of the runtime system.)

We illustrate the Virtual Singleton using the globally visible currentTransaction object which is a standin for the actual transaction associated with the current thread. We could have just as easily illustrated the pattern using currentUser to access the access permissions of the currently logged-in user.

The classes TransactionManager and RealTransaction both implement the Transaction interface (see Figure 1.) The Transaction Manager implements the interface by finding the RealTransaction for the current OSContext and delegating the action to it. We don't care whether the OSContext is a Process, Thread, VirtualMachine or some other concurrency concept; we only care that it is unique.

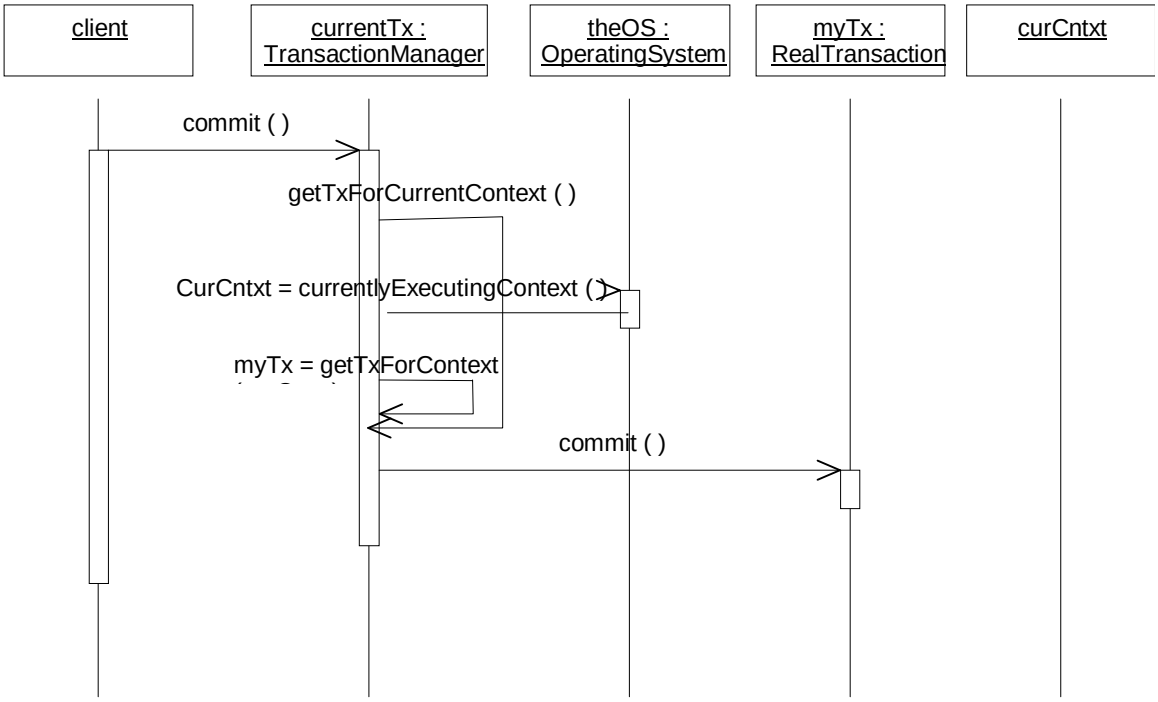
**Figure 1. Structure of the Virtual Singleton implemented using the Pure Object Approach**




In Java, we would invoke the methods on the TransactionManager as class (static) methods since we have no way of refering to an instance. These class methods would delegate to instance methods of the appropriate RealTransaction.

The sequence diagram in Figure 2 shows the flow of control from the client to the RealTransaction for the clients OSContext via the TransactionManager.

**Figure 2. Flow of control using Pure Object Approach**




## Known Uses

In the OMG's Object Transaction Service, the "psuedo object" called "current" is a Virtual Singleton which gives Transactional Servers access to the current transaction context. Similarly, "control" is a Virtual Singleton which gives application software access to the outer-most transaction for the current processing context. It is left to the ORB vendors to decide which implementation approach to use to implement these Virtual Singletons.

In the DMS-100 telephone switch, all three procedural language approaches were used at least as far back as 1990 with several approaches already being present in 1981.

1. Multi-user security was added to a previously internal-only user interface for modifying switch data by using a Virtual Singleton procedure to access the userid of the logged-in user so that security checks could be made thus restricting visibility of certain data to authorized users. This implementation corresponds fairly closely to the approach described in Thread-Specific Storage for C/C++.
2. The global variable CCB refers to the Call Control Block for the call being processed in the current thread. Any software which wants access to the current call context merely "USES" the module CCBINST which contains the global variable CCB. Call process scheduling software ensures that this variable is updated to point to the appropriate instance of CCB whenever the call processing context is changed.
3. Per-Process variable declaration is supported by the Protel language. This avoid passing commonly used data structures with every procedure call.

In the Jericho Business Object Framework developed at Nova Gas Transmission in 1995-1998, Virtual Singletons were used to provide access to the current transaction and the security permissions of the current user.

## Acknowledgments

Thanks to Erich Gamma for his helpful comments on early versions of this pattern description and for taking on extra shepherding load to allow it to be considered for PloP98.