Lock Server

Robert Hirschfeld
hirschfeld@windwardsolutions.com

Jeff Eastman
jeff@windwardsolutions.com

28. July 1998

Abstract

In enterprise information systems based on a two-tier distribution architecture, there are several clients working with shared resources. When designing such a system one must ensure that each client that accesses such shared resources does not interfere with other clients accessing and modifying the same resources. If the resource in question does not have a thread safe interface and/or it does not provide concurrency control mechanisms, a *Lock Server* attached to that shared source can help provide controlled concurrent access which allows each client to work with a consistent view of the resource. This paper discusses the architecture of such a shared *Lock Server*.

Name

Lock Server

Aliases

Lock Manager

Context

You are developing an application for a distributed enterprise information system where some clients have to access or modify shared resources (Figure 1).

You have decided to design your application based at least on a two-tier distribution architecture. One tier holds the resources that may be shared by several clients and another tier holds the clients themselves.

Sometimes those shared resources do not provide any facilities to manage the concurrent access to them, or they do not provide an appropriate interface to their own concurrency control mechanism what makes it difficult to integrate them with the mechanism used by the clients (e.g. an operating system’s file system or proprietary databases).

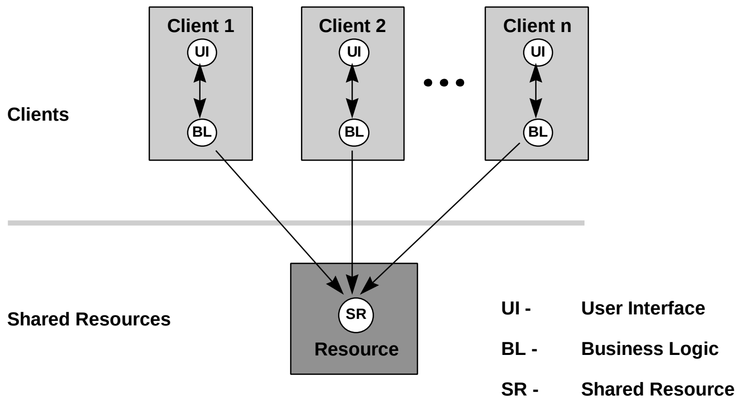

Figure 1: Distributed information system

Problem

How can you ensure that each client has the opportunity to coordinate its access to the shared resource with other clients to ensure an adequate level of resource consistency?

Forces

Some resources are not intended to be shared within a concurrent distributed environment, therefore, they don’t provide a thread-safe interface. That may cause inconsistencies to the resource when accessed by more than one client at a time. Examples of such inconsistencies are an invalid resource state or an invalid state of a resource team of which the resource in question is a member.

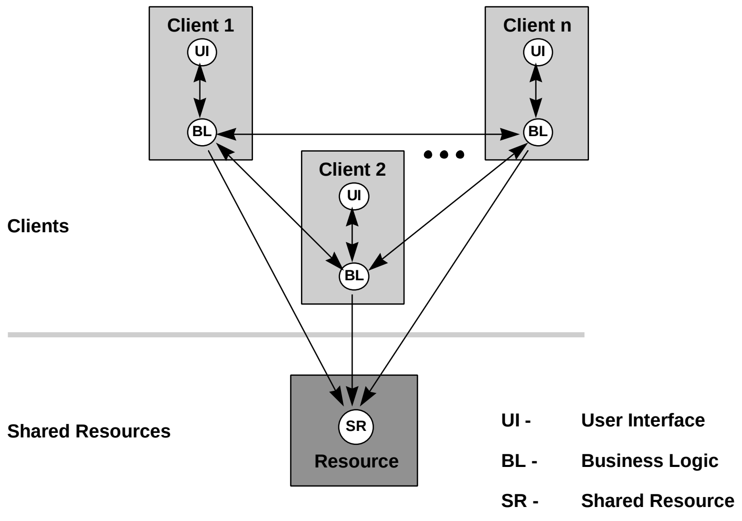

Figure 2: Client-side interconnection for access coordination

Sometimes resources rely on client side coordination to deal with concurrency control while accessing them. One way to provide this coordination would require the interconnection of each client with every other client that wants to access the same resource (Figure 2). Due to the fact that every client has to negotiate with all the others and therefore has to know all of them, it is very expensive to introduce new clients to or remove clients from the group. All clients have to agree about the resources to share to be able to negotiate about the required access.

If one has access to the source code of the resource, it may be possible to add required mechanisms like concurrency. This is not a good approach, however, as these modifications may cause inconsistencies with future releases of the resource implementation.

Encapsulation of the resource and adding of thread safety and concurrency control features to the capsulemay cause the loss of the identity of the resource. That in turn could create problems with other mechanisms in which the resource is involved.

Solution

Introduce a *Lock Server* into the system architecture (Figure 3). This *Lock Server* is a system infrastructure component that is separate from and independent of each client and the shared resources, too. It is known to all clients, so every client may access the *Lock Server* to claim a lock for accessing the related resource.

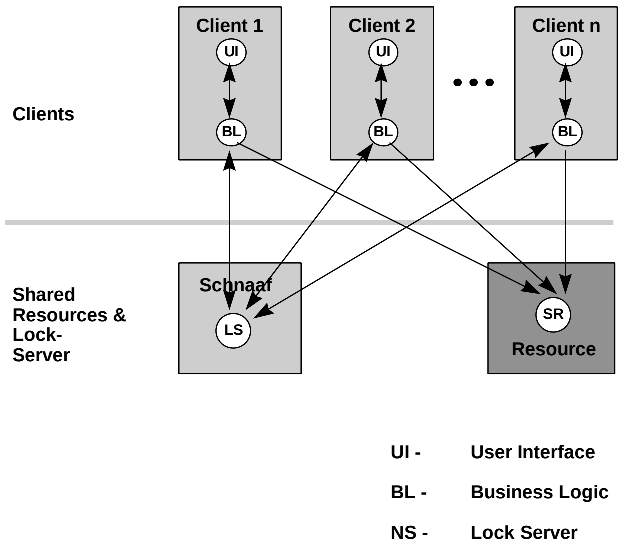

Figure 3: Lock-Server supporting access coordination

Before a client accesses a shared resource, it has to acquire a certain lock for this resource from the *Lock Server* (Figure 4). After the client has completed its work with the resource it should release the lock to allow other clients to access the resource fairly.

Depending on the lock model used, a request to acquire a lock may be rejected by the *Lock Server* if it conflicts with other locks already granted.

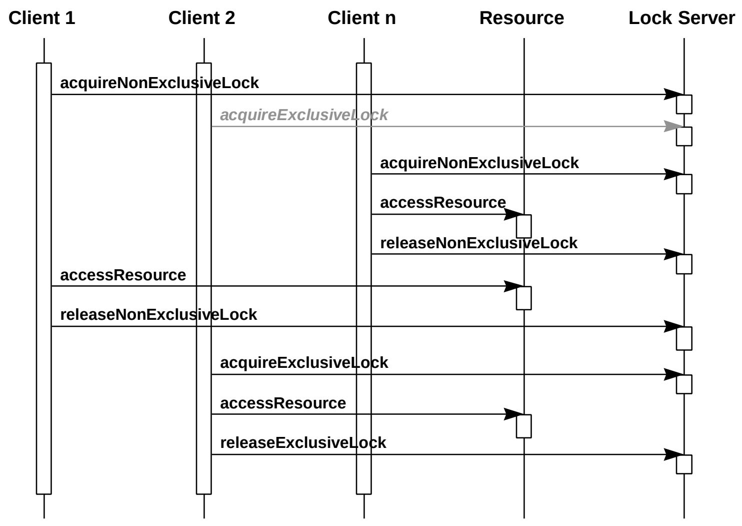

Figure 4: Sample scenario showing lock acquisition

Enforce all possible clients of a resource to use its associated *Lock Server* to avoid inconsistencies. Therefore all system parts sharing a resource have to agree on using its *Lock Server* to coordinate concurrent access.

Consequences

All clients have to agree about the resources to share to be able to negotiate about the required access with the *Lock Server*.

You also need to deal with the situation where a client locks a resource and then either forgets to unlock it or dies.

Since the *Lock Server* may be located anywhere in the distributed environment, the additional communication between a *Lock Server* and its associated clients may introduce an additional performance reduction.

Implementation

Granularity. The application of a *Lock Server* is not limited to a shared resource as a whole. Instead you could divide the resource conceptually into smaller parts that can be treated as resources with their own *Lock Servers*. You should try to divide every resource into disjoint parts to avoid additional locks that cause other clients to wait even if they sometimes don't have to. If you are not able to obtain a disjoint division, you have to ensure that clients working with such resources take care about the locks of the joint set of resources.

Several Resources. A *Lock Server* may be responsible for managing concurrency to more than one resource. Here the resources managed by the server must be distinguishable by all clients and the server. This could be achieved by introducing a name for each resource. All clients have to agree about which name is associated with which resource and they have to provide the proper name to the *Lock Server* while acquiring and releasing a certain lock.

Communication. If your applications are intended to run within a distributed heterogeneous environment, you should decide to apply standards like CORBA and CORBAservices from the Object Management Group (OMG) as much as possible ([OMG94, OMG95, OMG96]). So you can keep your application and the *Lock Server* interfaces independent from the constraints usually introduced by the environment like the programming language and the network protocol.

Initial Connection. If a desktop client wants to establish an initial connection to the *Lock Server*, it has to look for it. Here some sort of naming service may help by supporting the client to access the *Lock Server* by a symbolic/meaningful name. On solution could be the application of the OMG Naming Service ([OMG96]).

Sharing & Concurrency. A *Lock Server* is usually shared among many clients. Therefore you are responsible for coordinating concurrent access to the shared *Lock Server* to avoid race conditions. Since you are the developer of the server, it should not be that hard to add concurrency control and thread safety features. One solution could be the application of the OMG Concurrency Control Service to serialize access ([OMG96]).

Location. A *Lock Server* should be located on a reliable system because it must be available as much as the resources themselves. It could be placed nearby the resources (e.g. the same network node or even the same object space).

Related Patterns

Notification Server[1] ([Hirsch96]): Like the Notification Server the *Lock Server* is a server that is attached to a certain kind of resource without affecting the interface or implementation of that resource.

[1] *Notification Server:* In enterprise information systems based on a multi-tier distribution architecture, there are several clients working with shared resources. When designing such a system you have to ensure that each client has a consistent up-to-date view on the actual contents of the shared resource. If the resource in question is passive, i.e. the resource is not able to notify interested clients about changes of its (internal) state, a Notification Server, attached to that passive resource, helps achieving such a consistent view for each client ([Hirsch96]).

Known Uses

A prototype of a Manufacturing Control System at a big German automobile company uses the Lock Server for coordinating access to shared resources located in the middle tier within a Three-Tier Distribution Architecture ([Hirsch97]).

Hewlett-Packard's DistributedSmalltalk introduces a resource manager (DSTResource-Manager) that is used in the Presentation-Semantic-Split framework to manage resources that are shared between several clients ([HP95]).

Acknowledgments

Thanks are due to Steven Abell and Antonio Rito da Silva for giving helpful hints for improvement.

References

[Hirsch96] Hirschfeld, R.:
Notification Server.
Workshop on Patterns in Systems Architecture,
OOPSLA'96, San Jose CA, 1996

[Hirsch97] Hirschfeld, R.:
Three-Tier Distribution Architecture.
In: Collected papers from the PLoP '96 and EuroPLoP '96 Conferences,
Washington University, Department of Computer Science,
Technical Report No. wucs-97-07, February 1997

[HP95] *HP Distributed Smalltalk User's Guide.*
Hewlett-Packard, October 1995

[OMG94] *Object Services Architecture.*
Object Management Group, December 1994

[OMG95] *The Common Object Request Broker: Architecture and Specification*
Object Management Group, July 1995

[OMG96] *CORBAservices: Common Object Services Specification.*
Object Management Group, March 1996