Problem

Do I need to make the methods on my component thread-safe or is that handled for me automatically?

Solution

No, that is the responsibility of the component developer.

Background

The IDL methods of a component (or in general of any CORBA object) are called by the ORB according to some threading configuration policies. Normally, the ORB has available a number of threads to be used to dispatch calls "in parallel". See the language specific sections below for details about the ORBs we are using in the different languages.

This means that typically the developer of Components should make sure that each component method that can be called through an IDL interface (directly or indirectly) is reentrant. The same method can be called multiple times concurrently or in parallel with other methods from multiple threads.

While it is possible to make everything in the ORB single threaded (by assigning only 1 thread to the ORB at configuration time), that is not good for performance. For example, the Java ORB is configured to work with several threads, to improve response times.

Moreover, fully synchronising complete IDL methods is in many cases overkill and will heavily affect performance. It is usually much better to implement synchronisation at the resource access level with finer granularity.

Component Method Design/Implementation Considerations

So, generally all components should be coded so that they are thread-safe (reentrant). That is, they should gracefully deal with multiple ORB threads calling the same or different component methods simultaneously.

If just a few methods on a component are critical (e.g. with respect to accessing shared data), you can focus your attention on just those methods.

Be careful to think through your design, however, to avoid any deadlock scenarios. For example, a deadlock occurs if component A's method "A.a()" calls another component "B.b()" from within a synchronization lock, and B.b() calls back to A.a() or to another method of A that shares the lock. While in a monolithic application the last call would be done in the same thread and thus go fine, in a distributed environment such as the ACS component /container model a different thread is used, and the lock in A will cause a dead-lock. Obviously, you must prevent this in your code.

Language-specific concerns

- Java
  - You can use standard java synchronization techniques. You may do this either at the method level or, if performance concerns dictate, you may wish to consider narrowing the scope of the synchronized block. Please also consider using the more powerful synchronization classes from the concurrent library, see java.util.concurrent.locks and FAQJCompThreads.
  - All pitfalls (atomicity, ordering, visibility) of the Java memory model must be considered, see http://java.sun.com/docs/books/jls/third_edition/html/memory.html or http://puredanger.com/techfiles/JavaOne_ConcurrencyGotchas.pdf
  - Example of a method in the public IDL interface that is synchronized at the method level (generic CORBA example not specific to ACS component-container model)
  - Example of a method in the public IDL interface that is synchronized in a more precise way (generic CORBA example)
- C++
  - The CPP Container CDB contains a **`ServerThreads`** attribute that specifies the number of threads used by the ORB to dispatch calls in parallel.
    See the online schema documentation for details.
    If you set this value to 1 you will get a single threaded, fully synchronised Container.
    This technique is used for example to link with ACS non-reentrant AIPS++ code.
  - Consult your preferred C++ language reference for techniques to synchronize access to shared data.
    This may involve use of mutexes or semaphores, for example.
  - The ACE library provides very powerful synchronisation objects (mutex, semaphore and many other).
    See for example the ACE Tutorials and the ACE Programmer's Guide
  - ACS provides a very convenient **`ThreadSyncGuard`** class (built on top of ACE Mutex.
    `ThreadSyncGuard` objects can be used to solve in a very simple and elegant way many synchronisation problems.
    See the online documentation here.
    A simple example of usage can be found in the **`acssamp`** module in the **`acssamplImpl`** class. http://www.eso.org/projects/alma/develop/acs/OnlineDocs/ACS_docs/cpp/classACS_1_1ThreadSyncGuard.html
  - TAO ORB configuration options, including threading policies, are described here
- Python
  - In general, Python provides locks, semaphores, etc. Look at the following links for more info:
    - http://www.python.org/doc/current/lib/module-threading.html
    - http://www.python.org/doc/current/lib/module-thread.html

Related articles

- How can more people do development with ACS on the same machine without disturbing each other?
- Which ports are used by ACS?

- Problems connecting to ACS servers on a remote machine: bad /etc/hosts
- Why does the getComponent method of ZLegacy/ACS.ContainerServices return an object of type None?
- Why are some of my print statements not showing up in the container output section of acscommandcenter?