Concurrent Objects

by David G. Messerschmitt

Supplementary section for Understanding Networked Applications: A First Course, Morgan Kaufmann, 1999.

Because concurrency is often an application requirement for performance and/or intrinsic functionality reasons, application architects decomposing the application into interacting objects (or assembling interacting components) must often consciously take into account concurrency. Thus concurrent object interaction illustrates concurrency considerations and the challenges faced by the architect.

An object executes when one of its methods has been invoked and it is calculating return values. As discussed in "Timing and Concurrency" (Section on page 303), each object blocks from the time it invokes the method of another object until it receives return values. However, if every object only responded to method invocations, then nothing at all will happen—some object has to initiate things.

Analogy: If a group of people come to a meeting, each vowing to speak only when asked a question, then they will all be silent. Someone must take charge of the meeting by asking a question.

There must be at least one object that initiates activity, even without one of its methods being invoked. Called an *active* object, it constitutes an independent center of activity, as opposed to a *passive* object, which does nothing unless one of its methods is invoked. As shown in Figure 1., an active object collaborates with passive objects by invoking their methods. A passive object may in turn invoke the method of another passive object—thus, a passive object can be a client as well as a server.

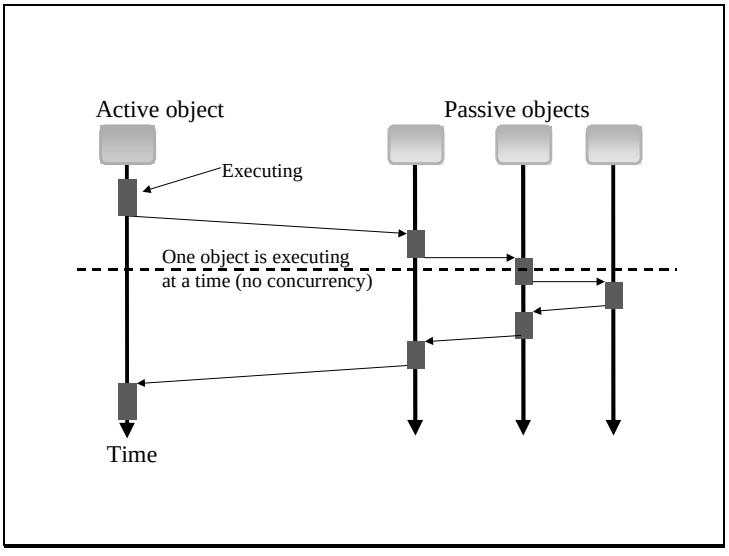

Figure 1. While an active object does not require a method invocation to execute, a passive object only responds to method invocations.

Analogy: Object collaboration by method invocation is like conducting a meeting through a series of questions and answers. Before answering a question, each person is free to ask another question, and take into account the answer. The first question is asked by a moderator, analogous to an active object.

During each method invocation initiated by an active object, many passive objects may execute. However, due to blocking on method invocations, only one object executes at a time. The active object cannot initiate a new method invocation until the previous one returns.

For this collaborating object model, there are several ways concurrency can be introduced, but the two listed in Table 1 are of particular importance. If two or more active objects are active—that is, working on independent tasks as independent centers of activity—then the tasks they are executing can be concurrent. Alternatively, one active object can introduce concurrency by using a message (rather than message with reply) service for interaction.

Table 1 Two ways to introduce concurrency among interacting objects

Approach	Description	Analogy
Two or more active objects	Each active object can work independently on one task. If there are two or more active objects, there can be two or more concurrent tasks.	Two waiters in a restaurant can wait on two tables concurrently, each serving one of those tables.
Interaction using messages	If a single active object collaborates by a message, rather than a message with reply, it doesn't block. Thus, it can induce other objects to work on multiple tasks by sending them messages, and it can move onto a different task after it sends a message.	If a waiter asks a busboy to clear a table, and waits for it to finish, there can be concurrency. If the waiter makes the request and immediately continues another task, the busboy can work concurrently.

Two active objects performing concurrent tasks are illustrated in Figure 2.. The subtle point illustrated is that a given passive object may participate in *both* concurrent tasks. For example, two active objects can invoke methods on the same passive object. This leads to possible resource conflicts—both active objects trying to change the same internal data in a way that conflicts. The solution—the same one discussed in "Resource Conflicts and Transactions" in Chapter 17—is *locking* the passive object, so it refuses to satisfy a second conflicting request until it completes the first.

The message interaction is illustrated in Figure 3.. Since an active object is not blocked waiting for a response from a passive object, it can resume execution immediately after sending a message. The active and passive objects are then executing concurrently. Similarly, an active object can send messages to two or more passive objects, which can then execute concurrently with one another.

Analogy: This is analogous to one worker delegating work to another, and then working concurrently. For example, a waiter may ask the busboy to clear a table, and then concurrently retrieve a clean tablecloth.

These options illustrate that the object decomposition of an application is consistent with achieving appropriate concurrency. The architect must simply determine the desired concurrency and

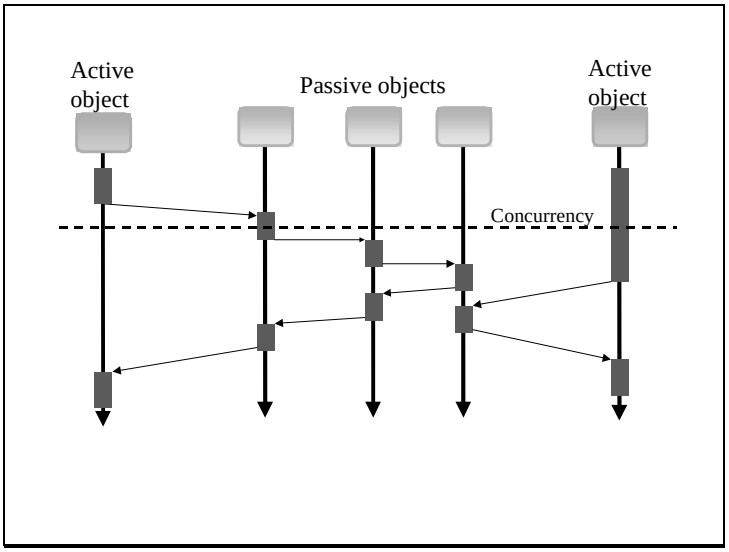

Figure 2. Two or more active objects can result in concurrency, as well as resource conflicts.

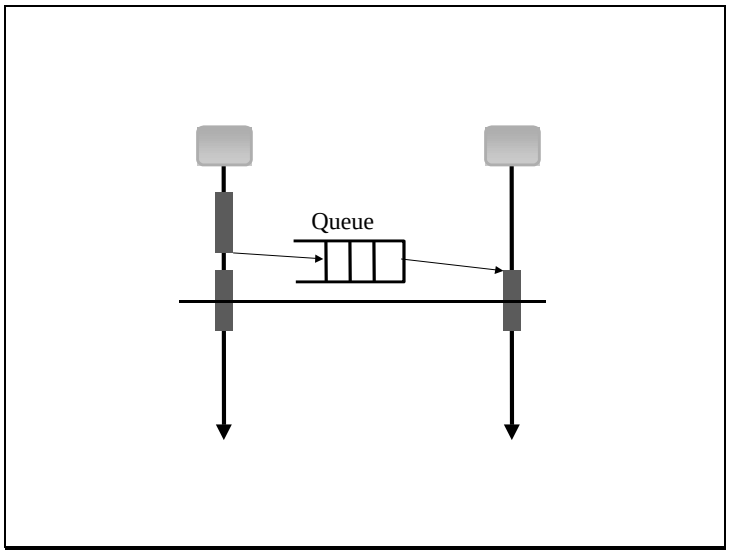

Figure 3. Object collaboration through messages can result in concurrency even when there is a single active object.

then choose an appropriate combination of active objects and object interaction models.

Review

In object-oriented programming, an object is active if it initiates activity even in the absence of an invocation of one of its methods; otherwise, an object is passive. Concurrency is achieved by having two or more active objects, or by a single active object interacting with two or more passive objects using messages. Each active object requires its own thread or process. Each (passive or active) client object that invokes the method of a server object is blocked until that invocation returns, slowing task completion time. Messages do not require blocking, so that an active object

can continue working immediately after sending a message.