

FLEXIBLE STATE SHARING AND CONSISTENCY MECHANISM FOR INTERACTIVE APPLICATIONS

CROSS-REFERENCE TO RELATED PATENT APPLICATION

The present invention is related to co-pending U.S. patent application Ser. No. 09/083,702, filed on May 22, 1998, entitled "Flexible Event Sharing, Batching, and State Consistency Mechanisms for Interactive Applications," by Bholra et al. This co-pending application, which is commonly assigned with the present invention to the International Business Machines Corporation, Armonk, N.Y., is hereby incorporated by reference in its entirety into the present application.

FIELD OF THE INVENTION

This invention relates to distributed applications and more specifically to a system and method for providing event and state sharing support for building object-oriented interactive collaborative applications in wide-area distributed environments such as an intranet or the Internet.

BACKGROUND

With the ability to extend web browser functionality using JAVA (a trademark of Sun Microsystems), plugins, ACTIVEX (a trademark of MicroSoft), etc., simple interactive groupware applications are being made available to a wide population of users. Examples include Internet chat and simple white boards. However, for building complex object-oriented collaborative applications, developers need a simple and powerful programming model, which at the same time should allow good response time and efficient implementation in a wide-area distributed environment. Appropriate support for sharing state is critical for supporting collaborative applications. In wide-area environments, replication of state is used to improve response time. Also, there are instances where a more primitive mechanism like event notification (also called event sharing) is a much better match for application requirements than state sharing. These high-level requirements for interactive groupware, coupled with replicated state, lead to three critical objectives that are addressed by the present invention.

OBJECTS OF THE INVENTION

1. Programming model: One objective, from a system design viewpoint is to decouple the programming model from any concurrency control implementation (including those of the present invention). For example, when an application issues an operation on a shared object, the system should have flexibility in scheduling this operation at the various replicas. This separation between the issuing of an operation and its execution leads to an asynchronous programming model that allows the system to employ a variety of concurrency control implementations such as pessimistic locking, ordering actions via a server, optimistic notification with automatic rollback and others. From the application developers viewpoint, the following requirements are considered:

Pipelining: It is desirable that an application be able to continue processing user input and initiate actions while its previous actions are being executed. This is useful, for example, when the user input is fine-grained, and a small response lag is acceptable, e.g., typing a few characters ahead of the echo. Depending on the

lookahead permissible to a user, and the actual response time, this can allow for faster user interaction. The asynchronous model motivated by system needs also meets this requirement.

Atomicity: As single-user applications have no source of contention, there is no need to distinguish between a group of operations which are part of a single action and a series of actions. However, with multiple users, it is necessary to specify some group of operations as indivisible so that they are scheduled for execution at the same time, and to ensure that other operations are not interleaved with them. Hence, the programming model should support atomic actions that access multiple shared objects.

Support for legacy applications: It should be easy to convert existing single-user application data-structures into shared data using class extensions.

The present invention has features that provide an asynchronous model for specifying atomic operations on the shared state using update events. This model only requires shared objects and events to implement a simple Marshallable interface. This interface allows for powerful state sharing semantics which are not possible to implement with simpler interfaces like JAVA™ Object Serialization.

2. Consistency of shared state: Potential inconsistencies in the replicas can arise due to different ordering of the events at different processes. Most systems take two extreme approaches when dealing with consistency of replicated state, (1) a fully optimistic approach with rollbacks and reexecution, and (2) a pessimistic approach utilizing locking. The first approach frees the application programmer from the burden of consistency maintenance, but the effect of jitters in the user interface due to automatic rollback could be a problem. The second approach does not allow enough freedom of interaction. It has been observed that strict locking is usually not necessary for collaborative applications because of implicit social protocols employed by collaborating users.

The present invention uses an intermediate approach of optimistic execution along with a combination of three mechanisms which flexibly expose the application writer to more of the distributed nature of the application:

- a. Global Locks: can be used by the application to enforce correct ordering;
- b. Detection of Conflicts: the system detects conflicts due to incorrectly ordered updates and informs the application; and
- c. Cloning and Re-initialization: the application can use this to construct an application specific conflict resolution policy.

3. Application specific event batching: Fine-grained user interaction with a GUI leads to a lot of updates on the shared state. For efficiency reasons, all these updates should not be propagated to the other users in the collaboration. The present invention uses a novel event batching technique (Late Event Modification) to resolve this tension between interactivity and performance.

SUMMARY

The foregoing and other objectives are realized by the present invention, which provides a system and method for event and state sharing support for building object-oriented interactive applications in wide-area distributed environments (such as an intranet or the Internet).

One embodiment of the present invention is implemented as middleware that provides support for different classes of