SYNCHRONIZATION MECHANISMS FOR DISTRIBUTED PERSISTENT OBJECTS AND COLLABORATIVE USER VIEWS

Soon-Young Huh, Hyung-Min Kim
Graduate School of Management
Korea Advanced Institute of Science and Technology
207-32 Cheongryangri-dong, Dongdaemun-gu, Seoul, Korea
syluh@green.kaist.ac.kr, hmkim@green.kaist.ac.kr

ABSTRACT
The collaborative tasks performed by multiple users linked with computer networks require the coordination of the concurrent users’ activities. Such collaborative users usually share common objects but manipulate them with different views suiting individual needs. This paper proposes synchronization mechanisms for the shared objects and their dependent objects. In the proposed mechanisms, the shared object is a persistent object residing at a database server, while the dependent objects are transient objects such as various user views or persistent objects such as replicated objects in another databases.

1. INTRODUCTION
The systems for supporting collaborative tasks help group users to perform tasks individually while achieving enhanced productivity in group work by facilitating sharing information about the task. In an organization, group users interact with the shared objects on the basis of their own views by creating, deleting, or modifying the objects. Meanwhile, as distributed database systems become core underlying architecture of the organization, the shared object may be replicated to another remote sites. In such capacity, when some users make changes on the shared objects, other users may be left with unupdated views, or replicated objects may be inconsistent. The role of synchronization mechanisms is maintaining consistency between the shared object and its replicated objects or dependent user views on a real-time basis.

In this paper, we present an object-oriented database model for synchronization mechanisms that can manage dependency relationship between a persistent shared object and its dependent objects including transient user views and persistent replicated objects, and coordinate change notification between them. A prototype system is developed based on a commercial ODBMS called ObjectStore [4] with the C++ programming language.

2. CONSTRUCTS FOR SYNCHRONIZATION
2.1. Basic concept for synchronization
In this part, we describe the basic concept involved in synchronization mechanisms using an engineering design example. Consider a tree structure stored in database comprising an automobile as shown in Figure 1. The tree structure representing an automobile in database is a shared object in an organization, and accessed by the employees of design, production, and marketing departments through their views. Whereas, the tree structure stored in database 1 can be replicated to database 2.

![Figure 1. Shared objects and their views.](image)

The operational aspects of coordinated modification in the task performed in collaborative systems usually have two key components: shared object and its dependent objects. To adopt the two components in a generalized way, the synchronization mechanism uses two primary structural constructs: supporter and dependent. A supporter is a shared object or a collection of objects to which changes can be made. It is a persistent object residing at a database server. The tree structure representing an automobile shown in Figure 1 is a good example of a supporter. A dependent is an object that is affected by the supporter's change. In general, it provides a visual interpretation of the supporter or its intermediary object. The views used in three departments respectively in Figure 1 or replicated tree structure in database 2 can serve as good examples of dependents of the tree structure supporter in database 1.

2.2. Core constructs and mechanisms
The synchronization mechanisms have two primary tasks: (1) management of the dependency relationships and (2) notification of the changes happening at the supporter to its dependents. As a dedicated record-keeping tool to hold dependency relationship between supporter and dependents,
there is a dependency dictionary. Figure 2 shows the basic objects and their important operations for synchronization mechanisms using Rumbaugh’s Object Modeling Technique (OMT) [6].

As shown in Figure 2, there are two classes linked to the DependencyDictionary class: Supporter and Dependent. The key of the dependency dictionary is the supporter, and the values for the same key become the set of dependents. To manage the dependency relationships between supporter and dependents dynamically, three types of primary dependency manipulation operations are available in DependencyDictionary class: addition (addDep()), deletion (deleteDep()), and retrieval (getDepSet()) of dependents for a particular supporter. When a new dependent is created, the sendAddDep() operator is invoked to send a message to the supporter that new dependency relationship is occurred. Meanwhile, when a dependent is removed, to inform the supporter of disappearance of the dependency relationship, the sendDeleteDep() operator is invoked. When a change occurs at a supporter, the sendUpdate() operator of Supporter class is triggered to send an update request message to all the registered dependents. Upon receiving the requests, the Dependents execute the update() operator to modify their status accordingly.

The class definitions for Supporter and Dependent discussed above are generic, minimal, and suitable for managing the dependency relationships and change notification. Thus, they can be easily applied to application-dependent constructs. As superclasses, the definitions of Supporter and Dependent class can be inherited down to specialized subclasses playing a role of supporter or dependent. In addition, the attributes and operations for application-dependent task can be added to the subclasses.

3. SYNCHRONIZATION MECHANISMS IN DISTRIBUTED ENVIRONMENTS

3.1. Management of dependency relationship

Applying the synchronization mechanisms described in previous section to distributed computing environments, calls for an extension of the mechanisms so that it can facilitate the activities between the supporters and dependent objects spread around the networks. To manage dependency relationships in distributed environments we employ a dependency relationship management mechanism with two tiers. At the lower tier, an internal dependency dictionary controls the dependency relationships between a supporter and dependents in one process. At the upper tier, an external dependency dictionary manages the dependency relationships between a supporter in one process and its dependents in other processes. On the other hand, there can be two types of dependents: transient dependents and persistent dependents. Transient dependents are the views in dispersed clients whereas persistent dependents are the replicated objects in the current or remote databases.

To apply the dependency dictionary proposed above to the distributed environments, it is necessary to classify the dependency dictionary into four types according to their roles. (1) **Internal Transient Dependency Dictionary (ITDD)** manages the dependency relationships between a supporter and its transient dependents that reside in the same process. (2) **External Transient Dependency Dictionary (ETDD)** manages the dependency relationships between a supporter and the processes managing its transient dependents and residing outside. (3) **Internal Persistent Dependency Dictionary (IPDD)** manages the dependency relationships between a supporter and its persistent dependents that reside in the same process. (4) **External Persistent Dependency Dictionary (EPDD)** manages the dependency relationships between a supporter and the processes managing its persistent dependents and residing outside. Among the four types of dependency dictionaries, ITDD and ETDD are used for managing transient dependents, while IPDD and EPDD are used for persistent dependents.

3.2. Management of change notification process

In this section, we describe how the change notification process occurs in distributed environments. The mechanism
to support the two-tiered dependency relationships is embodied in two change notification objects, Change Notification Server and Change Notification Client. These two objects have operations to support communications between servers or between server and client. And Change Notification Server is the object for detecting and notifying a change at server side while Change Notification Client is the object for receiving and reflecting a change at client side. When a change is encountered, Change Notification Server uses an IPDD to find persistent dependent internally and EPDD to find other databases containing persistent dependent. And to find the clients maintaining user views it uses ETDD. Meanwhile, Change Notification Client uses its ITDD to know what kinds of views are created in the client.

![Change notification process (event trace diagram)](image)

To describe the change notification process, we can depict the process using event trace diagram [6] as Figure 4 and it can be presented in two phases. Phase I pertains to the synchronization activities between databases, i.e., the change notification processes to persistent dependents. Especially, the entire process in Phase I should be dealt with within the scope of a single transaction for ensuring persistent dependent synchronized and consistent. To this end, the two-phase commit protocol [5] is adopted in the synchronization mechanisms. As shown in Figure 4, In the first step, when changes are made to a supporter in database, corresponding ChangeContent object is created and the contents of the change are added to the ChangeContent object. In the second step, when the locally committed change contents are released to the Change Notification Server in the database 1, the server identifies the external dependent databases containing persistent dependents using its EPDD. Then the Change Notification Server sends the change contents to the individual dependent databases with a change update request. In the third step, upon receiving the change contents, the Change Notification Server in database 2 identifies the persistent dependent using its own IPDD. Then, it performs appropriate updates according to the requested change contents. Meanwhile, during the Phase II of synchronization, the change notification is propagated down to the transient dependents in clients. As depicted in Figure 4, in the first step of Phase II, when changes are made in a supporter and the update operations are terminated successfully in database, the Change Notification Server in the database identifies the dependent clients by way of its ETDD. Then the Change Notification Server generates a list of change contents and sends them to the individual clients with a change update request. In the second step of Phase II, the client in turn receives the change contents and performs appropriate updates for the individual changes, consulting its ITDD to find specific user views.

4. CONCLUSION

In this paper, we propose synchronization mechanisms to maintain the replicated objects and user views synchronized and consistent as the shared objects are evolving in distributed computing environments. The proposed mechanisms built on an object-oriented database system are capable of managing the dependency relationships between the shared object and dependent objects, and automatically notifying the changes made on the shared objects to their dependents. There are two distinct directions for future research. First, the mechanisms proposed in this paper can be applied to the various collaborative systems requiring consistency maintenance functionality. Second, the knowledge-base containing application-dependent consistency maintenance rules can be included in the proposed mechanisms.

REFERENCES