# Cooperating Transactions in the EPOS Software Engineering Database

*Jens-Otto Larsen*

Div. of Computer Systems and Telematics (IDT),
Norwegian Institute of Technology (NTH), N-7034 Trondheim, Norway.
Phone: +47 7 594485, Fax: +47 7 594466, Email: jensotto@idt.unit.no.

## Abstract

This paper presents the EPOS software engineering database and its models and mechanisms to support cooperative work in a software engineering environment.

## 1 Introduction

One requirements for database support posed by advanced applications, like CAD, CIM and Software engineering (SE), is support for cooperation between ongoing activities.

Object-oriented database management systems offer a framework for modeling complex objects. Due to the duration and interdependence of development activities in these environments, mechanisms for communication, cooperation and synchronization are needed.

EPOS is a kernel software engineering environment covering both configuration management and software process management. The EPOS database, EPOSDB, has been developed to provide persistent storage of software products and processes. The EPOSDB is used by the EPOS applications, including a software process management framework, and therefore provides a framework for both data integration (through the data model) and inter-application communication.

First, we give a motivation for cooperation mechanisms from a software engineering view. We then briefly present the EPOSDB; the data model and transaction model. We then present the mechanisms that have been designed, and partially implemented, to support cooperative work.

## 2 Cooperation

We can identify some problems about the traditional model of serializable transactions, which we will attempt to solve:

- The serializability criterion cannot be used, as our transaction may be interdependent (as will be explained below), and because they last over a long time period. Serializability would mean that it should at least "look like" these were performed in some order, but this is not a god picture of the real world. We will have to allow conflicts to happen, without aborting or blocking transactions.

- We do not always want to have to wait for a long lasting activity to finish, before using any of the data produced by that activity. There is a need for some facility to communicate parts of your work for others to see, before you commit all your changes.
- A variant of the above arises when a user has started doing some modifications, and then want to "pass it on" to somebody else to do the finishing, while you continue with other work.

To solve these problems, we need some form of cooperation between ongoing transactions, so we can perform parallel work on the same object or represent dependencies between objects. The mechanisms should be flexible, so that one can choose the level of coupling (strong, loose) that is needed in each case.

We can support this by having these low level facilities available:

- Inter-transaction communication: A system for exchanging information (objects) between the clients of different transactions. This can provide *strong* coupling between the transactions.
- Notification: A system for asynchronous notification of clients about events in the database (message reception, check-in by other transactions). This provides *loose* coupling between the transactions.

## 3 The EPOS Database

The EPOSDB has been designed to support a software engineering environment and it offers a structurally object-oriented data model, versioning and a long transaction model. The database is realized in a client-server architecture.

In the following, we will give a brief presentation of the data and transaction models. The versioning will not be discussed here.

### 3.1 Data Model

The data model of the EPOS database, EPOS-OOER is based on the Object-Relation model, [6]. We will only give an overview of the most relevant characteristics of the data model.

- Objects have unique identities and are created as instances of object types. Objects must be related through external relations.
- Both object and relation types can be subtyped. Relation subtypes may restrict the types of the participating object types, which are inherited from the supertype.

- Objects are created as instances of a specific object type, but they may be dynamically converted to other super- or subtypes of the original type.
- The data model offers a standard set of value domains and a long-field domain for storing text files.
- The EPOSDB allows dynamic extensions of the type system, i.e. new subtypes of the existing type hierarchy can be added to the database at any time.

## 3.2 Transaction Model

EPOSDB offers a nested transaction model with long, non-serializable transactions. Transactions may survive several application sessions and they are represented in the database by special transaction objects. Transactions are started as sub-transactions of exiting ones and a "perpetual" *root transaction* forms the base of the transaction hierarchy.

The transactions are user-controlled – they may be started and terminated interactively. All database operations must be performed within the context of a long transaction and only one user can be working within one transaction.

Objects are checked out from and back into the parent transaction. If the requested objects are not present in the parent transaction, the request is passed up the transaction hierarchy. The EPOSDB provides a flexible locking scheme, including non-restrictive read and write locks, similar to the lock types in ObServer [2].

In addition to the functionality described above, transactions provide a framework for versioning[1]. All changes made within a transaction are uniformly versioned, implying that a transaction represents a consistent set of changes to the database. A transaction represents a physical change to the database. When the transaction is started, the range of the changes must be specified, i.e. the set of logical changes that are affected by the transaction. When later version selections are made, the changes made by a transaction are included if the version selection is within the range of the transaction.

[1]See e.g. [5] for more about our versioning.

## 4 Cooperation Mechanisms

In this section we present the cooperation mechanisms designed for the EPOSDB.

Communication between database applications is important for cooperation. The information which is communicated is related to the database contents, both high-level information interpreted by humans, formatted information interpreted by applications and direct relations to objects in the database. Since the database is common to the applications, it is the natural environment for a communication system.

In the EPOSDB, a long transaction is the fundamental work environment and its life-time may span several application sessions. Since long transactions exist without connected clients, they can be viewed as persistent database processes and they form a natural base for defining the communication system between the database users.

The cooperation model in the EPOSDB defines a set of mechanisms to enable:

- Message Sending
  This is a system for asynchronous communication between transactions offering the ability to send simple, typed messages to other transactions, possibly at different sites.
- Notifications
  Asynchronous notifications about events in the database. An example usage is to get notifications about checked-out objects which have been modified by others.
- Object Propagation
  A framework for propagating changed and new objects to designated ongoing transactions, without making these object publicly available.

### 4.1 Message Sending

A central point is the information carried by a message. There is a wide variety of possibilities, and we have settled for an extensible solution. We have base the design on a general type `Message`, which can be subtyped on demand (adding attributes carrying information).

The functionality of the interface operations is:

- Sending messages: This operation will send a message of the specified type (or an existing message object) to a specified transaction. Relationships to other objects will also be included, but not the objects referenced by the relationships.
- Test for incoming messages: Checks if there are any messages waiting, but not received. Succeeds if there are any messages in the mailbox.

- Receive message: The first message is read from the mailbox and a new message object is created in the transaction. This object will have an identity distinct from objects in the sending transaction. If the type of the message object is not defined in the schema of the receiving transaction, the message object will will be converted.

A transaction can allow or deny sub-transactions access to received messages. A transaction can also forward messages to specified sub-transactions.

## 4.2 Notifications

Objects can be checked out and locked with notification locks attached to them. When other applications try to access these objects, messages will be sent to the transaction that owns the lock. The notification locks resemble the communication modes defined in ObServer [2].

A set of basic message types for notification messages is provided: `Update-Ntf`, `Read-Ntf`, `Write-Ntf` and `Read-Write-Ntf`, describing respectively an update of an object, or requests for reading, writing, or reading or writing. The messages will be related to the object for which the notification was sent and to the transaction that caused the notification.

## 4.3 Object propagation

The object propagation mechanism is used when an application wants to propagate changed objects or new objects to designated other transactions without checking them into the parent transaction.

We distinguish two forms of object propagation: In the first case, the sending transaction keeps the locks and access rights and the receiving transaction gets a copy of the object without any locks. In the second case, the receiving transaction also gets the "ownership" of the object, i.e. locks and access rights.

With this mechanism, check in and check out can be viewed as object propagation between a transaction and its parent transaction. Only newly created objects and objects which are checked out for writing can be propagated to other transactions.

When the object propagation operation is performed, a "virtual" sub-transaction of the receiving transaction will be created and the object to be propagated is inserted into this sub-transaction. The receiving transaction will get a *propagation message* and can select when the object is received, i.e. checked in from the sub-transaction.

## 5 Implementation Status and Further Work

The EPOSDB as defined in section 3 has been implemented on top of the C-ISAM package [4]. Database clients sends requests through an RPC protocol to a central database server. The EPOSDB offers programmatic interfaces in C, C++ and Prolog.

The message sending mechanism has been implemented and the notification mechanism is under implementation. The system for object propagation is not completed: it needs to handle the cases when two transactions have different schemas. We will design a model in which both object conversion and automatic schema propagation is supported.

Due to the client-server architecture of the EPOSDB, we have not designed an interactive notification mechanism, i.e. asynchronous communication from the database server to clients. The current approach results in applications polling the database for incoming messages, but in some cases an immediate transfer and processing of messages in connected applications is desired. Work in this area will be necessary.

The nested transaction model lends itself to distribution with several database servers, each hosting a number of long transactions. We foresee a system with a central name service and server-server communication.

## References

[1] Stuart I. Feldman, editor. *Proceedings of the Fourth International Workshop on Software Configuration Management (SCM-4)*, Baltimore, Maryland, May 21–22, 1993.

[2] M. F. Hornick and S. B. Zdonik. A Shared Segmented Memory System for an Object-Oriented Database. *ACM Transactions on Office Information Systems*, 5(1), January 1987.

[3] Jens-Otto Larsen, Bjørn P. Munch, and Reidar Conradi. Cooperating Transactions in the EPOS Software Engineering Database. In *Proc. 3rd ERCIM Database Research Group Workshop on Updates and Constraints Handling in Advanced Database Systems, Pisa, Italy*, pages 61–67, September 1992. Also EPOS TR 157, NTH, 31 March 1992, 6 p.

[4] Bjørn P. Munch. *Versioning in a Software Engineering Database — the Change Oriented Way.* PhD thesis, DCST, NTH, Trondheim, Norway, August 1993. 265 p. (PhD thesis NTH 1993:78).

[5] Bjørn P. Munch, Jens-Otto Larsen, Bjørn Gulla, Reidar Conradi, and Even-André Karlsson. Uniform Versioning: The Change-Oriented Model. In *[1]*, pages 188–196, 1993.

[6] James Rumbaugh. Relations as semantics constructs in an object-oriented language. In *Proc. ACM SIGPLAN Conference on Object-Oriented Programming Systems, Languages and Applications (OOPSLA'87)*, pages 466–481, Kissimmee, Florida, October 1987. In ACM SIGPLAN Notices 22(12), Dec. 1987.