Cooperation Support in a Software Development Environment

Jens-Otto Larsen*
Norwegian Institute of Technology, N-7034 Trondheim, Norway

Submitted for 9th Int'l Software Process Workshop (ISPW'9)

March 1, 1994

1 Introduction

A process centered software development environment (SDE) should support cooperation between groups of developers. We will distinguish between truly cooperative work where a group of people work together on a specific task, and situations where people working on different tasks need to coordinate work on software objects used in several development efforts.

Work on computer supported cooperative work (CSCW) has provided tools that let several users work concurrently on objects, e.g. collaborative editors. A general SDE will not include such tools, but it should allow and support use of CSCW tools on objects handled by the SDE. More important is support for situations where users knowingly work on common objects in isolation for a period before the updates are shared with the rest of the group. If several developers are going to work on the same objects over a period, their work needs to be coordinated to avoid conflicts. An SDE should allow several policies for coordination, ranging from strict scheduling of work to allow concurrent work but require merging before the results are made available to others. These policies can be supported by the system by mechanisms that prohibit concurrent updates and by giving one user access to the temporary copies of other users.

The other group of situations, where developers independently try to access the same objects, also should be supported, but conflicts are often not predictable and should be handled on an object or user basis rather than through a general policy. We can use the same mechanisms to support the concurrency management, but since the developers generally do not know about each others and their intentions, the SDE could store information about the developers intents and goals. An example is that a user could specify what objects will be updated and a time estimate for a given task. Other users who try to access an object, could ask the system who else are updating the object, get the information provided for the other users tasks and use it to make a decision on how to progress. It would also be useful if the SDE could allow the users to communicate about their work through the environment, so that users could more precisely reference the objects used.

A process centered SDE will usually associate such information with the process models used in the system. To support the situations described above, the SDE should provide users with a possibility to access those parts of the process models which describe other users enactment state and plans. If the process models are local, i.e. not accessible by all users, the developers must be able to communicate their plans to the other SDE users. The same applies if the SDE permit access from developers who are not following a process modelled and managed by the SDE.

*Detailed address: Dept. of Computer Systems and Telematics, Norwegian Institute of Technology (NTH), N-7034 Trondheim, Norway. Phone: +47 73 594485, Fax: +47 73 594466, Email: jensotto@idt.unit.no.
In addition, the SDE must offer mechanisms that allow concurrent work on software objects and an infrastructure to allow developers to communicate through the environment. The mechanisms provided to manage concurrency should be available to users at a proper abstraction level so that they can be used to enhance concurrency in the system. There are several reasons to provide a communication infrastructure within an SDE. The most important is that users need to communicate both objects stored in the system and information about objects, and the

In the following sections, we discuss approaches in the EPOS project towards providing cooperation support. We have assumed that the SDE includes a software engineering database to store software objects and possibly also other objects like process model representations. We have used the word database many places and it should be interpreted as the database within the SDE.

2 Concurrency Management

Most database systems offer a mechanism for controlling concurrent access to databases to ensure that the results are consistent. These systems guarantee correctness by forcing the access patterns to satisfy a correctness criterion, which usually is serializability. As a consequence, the database system will in some cases undo updates to enforce consistency.

These mechanisms cannot be used directly in a SDE since it is not acceptable to let the environment undo the results of people's work. It is often the case that the users involved in conflicting work can manually bring the database to a consistent state, provided they can access and distinguish between the different copies of an object.

On the other hand, the system may require that a set of operations is performed as a unit to guarantee consistency. Furthermore, the functionality of traditional transaction models is also used to write tools that perform operations in an atomic way. This is done both because it is difficult and time-consuming to let users manually re-establish database consistency after a tool has only half-way completed and because it is easier to write tools that can assume a consistent database.

Users interact with the environment at a higher level of abstraction than the database. What a user name and conceive as an object may be stored as several objects in the database and the user-level objects are generally not accessed through database operations but by tool invocations, tool abstractions or by abstraction provided in a process model. Thus, every SDE needs to provide a user-level interface to the environment.

As a result, an SDE should provide both a transaction model at the database interface level to support tools and a user controlled "long transaction" model at a higher level. The database level transaction model should provide correct concurrent database access to tools which need to perform operations in a serializable and recoverable manner. These tools need to synchronize their database operations and perform updates on a level beyond what could be made accessible to the SDE users.

The user-level transaction model is a natural basis for providing work contexts to users and it should be less restrictive in controlling object access. Only some operations need to be serialized, and in many cases, it is admissible to let several users work concurrently on an object, requiring integration of the changes later. It should be possible to incrementally make changed objects visible to other users, both by committing objects (cannot undo changes) and by letting other users inspect intermediate results. The model should also provide a flexible locking mechanism where locks are placed on user-level objects, and where some locks can be overridden so that real concurrent work is allowed. The SDE should record information on who placed a lock on an object and which users have copied objects into local transactions.

The user-level transaction model is the natural integration level between concurrency management and software process enactment in a process centered SDE.
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 previous section, user-level transactions were described as natural work contexts for users. Since user-level transactions may exist without an active user, they can be viewed as the environment’s stable user connection and they form a natural base for defining the communication system between the developer using the SDE.

In the rest of this section we present three communication facilities that could be supported. All the facilities are based on user-level transactions, which as pointed out above form a natural context for enacting software processes.

**Message Sending** provides communication between transactions offering the ability to send simple, typed messages to other transactions. Messages are stored as ordinary database objects and can be used both to provide tool-tool communication and for user communication through tools in the SDE. By integrating the message sending facility in the SDE database, we can let messages reference software objects stored in the database in a consistent way.

**Object propagation** lets a user send updated objects to other transactions, tools or users without making these changed objects available to all users. 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.

**Notification** allows asynchronous, implicit communication between users. Notifications are simple messages which are generated and sent by the SDE when certain events occur. These events may be any operation on the objects managed by the SDE. An example usage is to make the system generate notifications when other users try to access certain objects (locked or not).

### 4 Conclusions

We have presented a user-level transaction models and a set of communication facilities which serve as vehicles to enhance concurrency and support cooperative work within an SDE. We have assumed that software developers know much about the objects they are using and can interact more with the SDE in resolving conflicting access to objects.

Many ideas from traditional transaction models are useful in a user-level interface for interacting with the SDE. The main differences are relaxation of correctness criteria and manual control over how conflict situations are handled. Manual control may make the SDE more complex and more difficult to use, but in many cases we can still leave conflict handling to the environment if desired. To achieve manual control, users must be allowed access to concurrency information, e.g. who has locked this object, and access to other users temporary copies of objects.

A large part of the communication between developers is made up of objects (partial results and preliminary changes) and information about and related to objects managed by the SDE. The precision and availability of the communication can be enhanced by integrating the facilities in the SDE and its database.