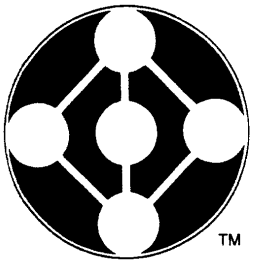

Alternative Technologies

Enterprise Integrity: Compensation

Vol. 2, No. 4

All too often, the newest wave of technology is accompanied by a lot of vendor hand waving. I think this phenomenon currently surrounds many integration products and explanations of product support for error or failure recovery. Some vendors even go so far as to suggest that the system resulting from integrating (transactional messaging) a set of applications which already have error recovery mechanisms will itself recover from errors. Nothing could be further from the truth. As more and more business functions are integrated, the problems of automated error recovery become increasingly important. It's only natural that many of the systems we integrate first are mission critical. Such integration demands the reliability we associate with transaction management if error recovery is to be robust. Unfortunately (and as noted in an earlier column), integration based on asynchronous messaging also demands a more sophisticated transaction model and recovery based on compensating transactions. I promised some guidelines for designing compensating transactions, and that is the subject of this month's column.

The basic idea is that each transaction has a matching compensating transaction that will "undo" any work that the transaction does. When transactions are treated in isolation or are applied sequentially, its pretty easy to come up with compensating transactions. All we need is the state of the system saved from the beginning of the transaction and a function to restore that state. In essence, this is how we recover a database using a backup copy. When transactions become interleaved, the rules become a little more stringent and the implementation a bit trickier. For example, the compensating transaction is applied by the transaction manager before the transaction is committed. Still, the process is much the same: the system is returned to an earlier state.

Compensating transactions require that each transaction to register its corresponding compensating transaction with an error management system so that recovery can take place automatically and consistently. The rules for using compensating transactions become more complex as the transaction model departs further from the familiar "flat" model. Formally, compensating transactions should always return a system to a prior state. If multiple systems are recovered, they are all recovered to prior states that share a common point in time. If the atomic actions that make up a transaction can be done in any order, and if each of these has an undo operation, then such a compensating transaction can always be defined. So here are three guidelines. *(1) Try to keep the overall transaction model as close as possible to the traditional "flat" model or else a simple hierarchy of strictly nested transactions. (2) Design the atomic actions so that order of application within a transaction does not matter. (3) Make certain that compensating transactions are applied in the right order.*

All this is well and good, but we know that business processes do not always lend themselves to such simple models: often they involve interleaved multi-hierarchies and networks. While the guidelines still apply, we need to understand that the processes a business uses to correct for errors do not always return the business to a prior state. Rather, the business is transitioned to some acceptable state and the nature of this state make available to those portions of the business that have some dependence upon it. Notice that I said "some acceptable state" instead of the more familiar transactional notion of an internally consistent state. Obviously, businesses do not follow a rigid set of rules of consistency as a database might. However, it should be equally obvious that some action will be taken if the business is not in an acceptable state.

This informal understanding of "acceptability" is the business equivalent of consistency. Although space does not permit, I would also argue that it is the appropriate one in a world in which the transaction model is extremely complex (and far from flat). In effect, businesses operate in an environment in which significant portions of the business are likely to be in error and undergoing correction at any particular time. While actions that have consumed few resources may be started over on error, more often corrective actions are taken in an attempt to preserve resources and to be as productive as possible. These corrective actions are real-world compensating transactions. They are more closely akin to "re-targeting" business systems than they are to an "undo" action. Think about it: how efficient would a business be if every error resulted in the work having to be redone from the start?

This leads us to two important conclusions, expressed as additional guidelines: (4) Consider using traditional compensating transactions when the combined cost of undo followed by redo has relatively small cost and minimal impact on the rest of the system. (5) If an undo followed by redo is likely to cause errors in other portion of the system (given the resource cost and especially in terms of time delays), design the compensating transaction to be "corrective." Transition directly to an acceptable state, noting that this need not be the original target state. In fact, I prefer to call such general compensating transactions *corrective transactions.*

Now just so you don't think compensating transactions are esoteric, consider a few familiar examples. In manufacturing: rework. In funds transfer: message repair. In billing: collections. In retail: spillage. In telecommunications: re-routing. Get the picture? Compensating and corrective transactions are old friends to businesses worldwide… and absolutely essential to the integrity of the enterprise.