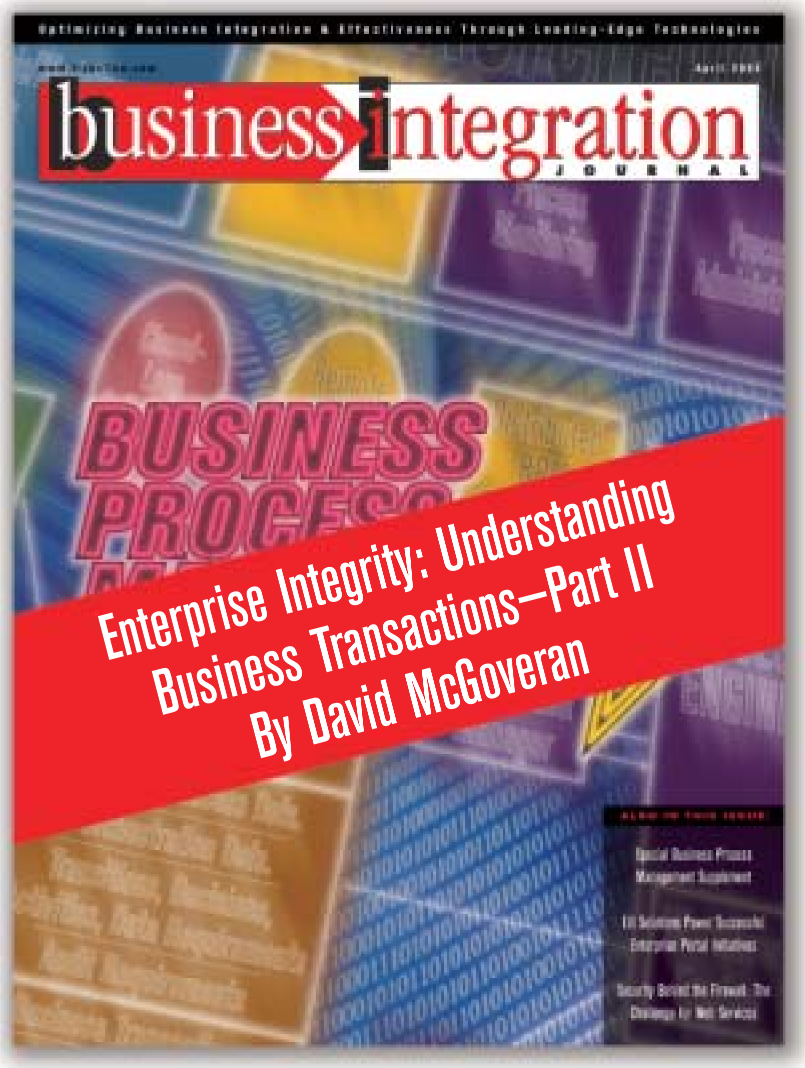
Enterprise Integrity: Understanding
Business Transactions—Part II
By David McGoveran

ENTERPRISE INTEGRITY

BY DAVID McGOVERAN

Understanding Business Transactions: Part II

We often think of business transactions in the sense of a purchase of goods—one party tenders the money and the other party tenders the goods. However, "business transaction" has acquired a broader meaning, evolving over a long period of use, eventually coming to mean almost any agreed upon group of activities between multiple business entities in which there is some quid pro quo among the parties. There is something profoundly special about business transactions; a set of properties that separates business transaction from other business activities. However, before we can explain these properties, we first need to understand some more transaction basics.

Note that consistency conditions are also known as *integrity constraints*, especially in the database world. In both the business world and the object-oriented world, they may also be referred to as *contractual conditions* or *compliance requirements*. Transactions can only legitimately begin when some set of conditions is satisfied and can only complete successfully when a set of conditions (often the same set) is satisfied. Furthermore, transactions can only perform certain transformations of state. Since computer-based transactions are usually defined with input values specified at run-time, some combination of values might cause a transition between two consistent states, but for which that particular change is disallowed. To prevent this, we can either define extra constraints on the permissible combinations of input values (and therefore on its initial state) or define transition constraints that preclude the change.

In practice, computer systems often avoid strict isolation property enforcement in an effort to improve concurrent access to state data or other transaction resources. If two transactions are completely isolated, then the results of running those transactions must be as if one (or the other) had run to completion prior to the second beginning (i.e., serially). Obviously, scheduling execution serially is a good way to waste resources and reduce throughput, so transactions are actually run concurrently with their steps interleaved using a protocol that guarantees the result is the same *as if* the transactions were run serially. The problem is that we can't tell what that specific equivalent order is, and so it's crucial that consistency is enforced at transaction boundaries to be certain of some correct result. An execution schedule of a transaction set that executes so as to ensure completely isolated behavior pair-wise for all transaction pairs is said to be serializable.

In some scenarios, every possible execution schedule of the applicable transactions may be serializable. No enforcement is then necessary. Likewise, some scenarios may permit almost all schedules to be serializable, so that only a degree of enforcement is necessary. These situations are legitimate uses of so-called "lower" isolation levels than serializable. Problems arise when no analysis is done to ensure that the chosen isolation level in conjunction with the scenario and transaction mix is equivalent to serializable execution, or when the transaction mix evolves without reanalysis. Even worse, using coding techniques to enforce transaction properties (instead of system facilities such as a TP manager) inevitably results in much higher costs of development and maintenance, and often leads to forms of data corruption that can be very subtle. Such data corruption is difficult to detect and expensive or even impossible to repair.

Properly speaking, a system that permits nonserializable execution schedules—as defined here—is not really executing logical transactions and so is a very high business risk, an accident waiting to happen. Such systems don't support HIPAA or Sarbanes-Oxley compliance. Consider a transaction that computes a number that must be reported according to Sarbanes-Oxley. If nonserializable execution schedules are permitted, it's impossible to say definitively that the transaction produces the intended number. The possibility always exists that some other, concurrent transaction interfered with the result. Exactly what can go wrong, and whether the error is tolerable or not, depends on the details of the concurrent transaction mix and the degree of isolation that's being enforced by the system. However, the more complex and varied the transaction mix, and the higher the load, the greater the likelihood of results for which there is *no definitive audit trail*. Thus, certifying the correctness of the number is impossible.

However achieved, transaction serializability is just not an option. While system-enforced serializability might be unnecessary for certain well-understood transaction mixes, the CEO asked to sign off on a Sarbanes-Oxley report should be skeptical. And, if data is being moved between two such systems for integration purposes, I would be seriously concerned about my *enterprise's integrity*. **bij**

About the Author

David McGoveran is president of Alternative Technologies. He has more than 25 years of experience with mission-critical applications and has authored numerous technical articles on application integration.
e-Mail: mcgoveran@bijonline.com
Website: www.alternativetech.com