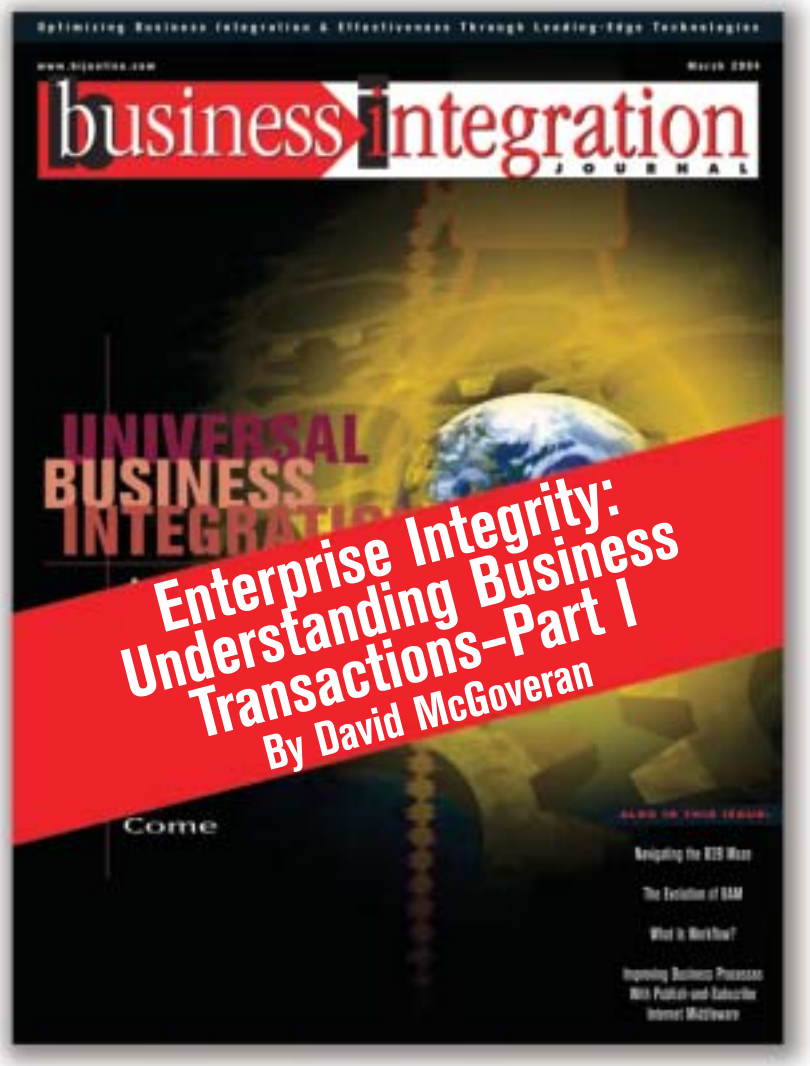
This article appeared in the
March 2004 issue of

business integration JOURNAL



Subscribe instantly at
www.bijonline.com

- Free in the U.S.
- $18 per year in Canada and Mexico
- $96 per year everywhere else

ENTERPRISE INTEGRITY

BY DAVID McGOVERAN

Understanding Business Transactions: Part I

Practically everyone has heard of "business transactions." The phrase permeates both business and technology language, especially that regarding B2B and business integration. Most people have some informal understanding of the phrase and know it refers to some kind of exchange or interaction between multiple entities. That's well and good, but we need something more precise. Can we give business transactions a formal definition that's compatible with our traditional, informal understanding? How do business transactions relate to transactions as implemented in online transaction processing (OLTP), application servers, transaction managers, and so on? This month, I'll begin explaining my answers to these questions, starting with some transaction basics.

There are many informal uses of the word "transaction." Most often, computer folks use transaction to refer to a unit of work that's made durable or otherwise confirmed as communicated to its destination. For example, both the communication of a message between two computers on a network and the writing of a related set of data to disk—each with some form of completion acknowledgement—are often referred to as transactions. This is a very physical notion of transaction in that it is not concerned with either consistency or isolation, and relies on the physical process of synchronization to establish transaction boundaries (so-called *synch points*). As such, they provide the basis for a unit of recovery. I prefer to call these *physical transactions.*

Formally, a transaction is *defined* as a group of actions that can be characterized by a state and that satisfy the ACID (atomic, consistent, isolated, and durable) properties. This definition includes important logical requirements and, for this reason, I refer to such transactions as *logical transactions*, distinguishing them from the less rigorous physical transactions. The transaction state is the set of values of a set of state variables at a point in time. State variables are any variables whose values have a potential effect on the final result of a transaction or which can be changed by the transaction. State variables fully characterize the portion of the system (e.g., database, business, application, etc.) that is acted upon, and transient or temporary variables are irrelevant.

A transaction is *atomic* in that it can't be partially completed and still retain its identity (i.e., intent or functional purpose). A transaction is *consistent* in that its state satisfies a set of predetermined consistency conditions at the start and at completion (including so-called transition constraints). Consistency conditions are also known as integrity constraints, especially in the database world. In both the object-oriented programming and business worlds, the phrases "contractual conditions" and "compliance requirements" are often used.

Transactions are *isolated* in that only the initial and final states may be known to other transactions, so they can't interfere with or influence each other's work except by one effectively preceding the other. Finally, transactions are *durable* in that the final state is not arbitrarily altered, but remains constant until altered by another transaction, unrecoverable system failure, or acceptable failure (e.g., system abandonment or media decay). Durability enables transaction recovery under most kinds of system failure. Later we'll discuss business parallels of these properties.

More than theoretical, ACID properties are prescriptive. They define transaction so as to guarantee highly desirable results in a predictable, and preferably automated, manner. Although these properties are defined relative to particular situations or contexts, and not in some absolute manner, it is important to note that they are necessary for predictable behavior. In effect, a transaction is a controlled transformation of state. Equally important, ACID property definitions enable automatic enforcement, permitting declarative system control in place of transaction-specific application coding.

Every logical transaction (and so every execution of it) has a functionality that can be declaratively specified by constraints. These obviously include constraints on the permissible starting states and completion states, often more restrictive than constraints applied to the entire system. However, a specific transaction performs only some of the transitions between the many combinations of consistent beginning and ending states. Restricting the permissible transitions effectively defines the transaction's intended functionality. Thus, an important aspect of consistency is the definition and enforcement of *transition constraints.*

Transition constraints are crucial in the business world and in modeling the world, generally. They prevent "working outside the system," "skirting the law," and "ill-gotten gains," by ensuring that "you can't get there from here." Our journey continues next month. For now, can you think of a company (or person) that lost its *enterprise integrity* because it didn't have adequate transition constraints? **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