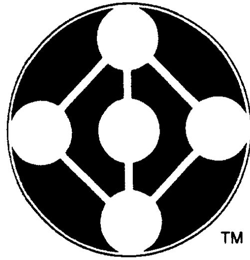


Alternative Technologies

Enterprise Integrity: Ready to Commit?

Vol. 2, No. 2

Forever in pursuit of the silver bullet, we often complicate the very problems we seek to solve. Common culprits are e-business (especially B2B) and integration policies with a singular goal of higher efficiency, usually understood as "faster". There are (at least) two traps in this thinking. First, as well-known since the old time and motion studies, faster processing is not necessarily better and often increases errors. People, and businesses, need unscheduled time to respond to and repair errors, to adapt to change, and to innovate. Second, cumulative local schedule optimization often leads to a bad overall schedule. Conclusion? The blind pursuit of "zero latency" is simply a mistake leading to inefficiencies and rigid business processes.

Please don't misunderstand: agendas like Straight Through Processing are important and increasingly necessary. Moreover, reducing latencies in business processes by an order of magnitude is a great goal. Unfortunately, we often confuse effective methods with solutions. Case in point: asynchronous messaging and the loosely coupled systems are flexible, aiding IT agility and therefore business agility (inasmuch as that business depends on IT). However, few have thought through the implications of a zero latency asynchronous infrastructure.

Let's consider the implications for maintaining correctness of data and applications via transaction management. Traditional application design relies on the assumption that the individual transaction steps are in lock step, have no latency, and the success or failure of each step is communicated synchronously. Synchronous communication gives the transaction manager (or application code) a chance to prevent partial completion of the transaction. Early in my career, we had to write each application so as to insure transaction atomicity, consistency, isolation, and durability. Most programmers delivered less than perfect solutions, unaware that correctness assumptions for one transaction mix often fail as new transactions are included.

Years of research gave us (provably correct) automatic methods to maintain transactional correctness without the need for application, process, or message specific coding. The results were encapsulated as TP (transaction processing) monitors, often associated with DBMSs. Although better distributed applications are written to take advantage of a TP monitor or DBMS managed transactions, most applications still attempt to maintain transactional integrity through code. Older mainframe applications may have very complex transaction structures in which transaction boundaries (synchronization points or synchpoints) are managed by a combination of code, control language, and TP monitor. Packaged application software is often just as complex. Whether legacy or packaged application, the relationship between application interfaces and transaction requirements is seldom documented. When an API exposes data or events, API boundaries are not necessarily coordinated with transaction boundaries. Therein lies a host of problems for all enterprise integration efforts.

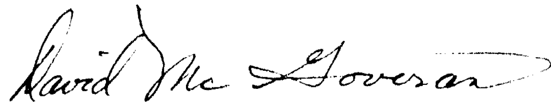
13150 Highway 9, Suite 123, Boulder Creek, CA 95006 Telephone: 831/338-4621 FAX: 831/338-3113

www.AlternativeTech.com mcgoveran@AlternativeTech.com
Copyright © Alternative Technologies, All Rights Reserved

With asynchronous messaging, maintaining known and coordinated synchronization points becomes extremely difficult. By definition, it permits latencies. This is problematic for the correctness (integrity) that transactions are intended to enforce, and we are forced to mix models of transaction isolation. "Synchronous" mode is commonly used within applications, typically using a pessimistic approach (locking) to prevent errors, isolating resource access and modification by distinct transactions. The result is a more rigid, closely coupled sequence of events. By contrast, while asynchronous messaging frees us to optimize error handling and change dynamically (and permits latencies in which to do this), it also forces an optimistic (no locks) approach to isolation. If something goes wrong, we *assume* we can detect and correct the problem without residual negative side effects. Usually completed steps are simply rolled back on detection of an integrity problem.

Optimistic approaches work reasonably well as long as the transaction mix is inherently isolated. Otherwise, residual effects will ultimately corrupt the application. When multiple asynchronous steps affecting distributed resources are involved in each transaction (welcome to typical *eAI* scenarios), the familiar rollback mechanism is not available and *compensating transactions* are needed. A compensating transaction is simply a transaction that "undoes" or compensates for changes already made. If both the transaction mix and the individual transaction steps are inherently isolated, a set of compensating transactions will let us repair errors in an asynchronous world.

We must be able to design correct compensating transactions… no easy task, but a solvable problem! (I'll provide some compensating transaction design guidelines in a future column.) Asynchronous messaging is a good thing and current approaches to integration have tremendous benefits. Nevertheless, given the code in legacy applications (and even in new many ones), we also need to reduce costs and the attendant business risks by insisting on good *transactional* design. Adopting asynchronous messaging with a focus on removing latencies can result in unanticipated errors and business rigidity. Gluing your enterprise into a tight monolith will hamper its agility… and your integrated enterprise has little value if you give up its integrity.



Copyright © Alternative Technologies, All Rights Reserved