

The PDFs of *eAI Journal's* Data Integration department are sponsored by:



**Data Junction** sets the standard for data integration with award-winning technology, customer support and technical services. Right now, more than 40,000 customers invested in Data Junction technology to solve their most pressing integration concerns, making our software the most widely deployed in the world. What integration challenges are you facing? Discover how Data Junction's solutions fit into your integration world. For an on-line demonstration of the power and versatility of Data Junction, contact us at 800.580.4411 or [info@datajunction.com](mailto:info@datajunction.com)

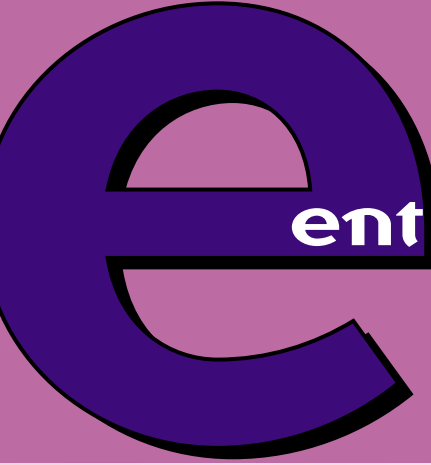


Data Junction Corporation  
5555 North Lamar Blvd.  
Ste. J-125  
Austin, TX 78751  
tel: 512-452-6105 ext. 256  
fax: 512-467-1331  
[www.datajunction.com](http://www.datajunction.com)



**This article is a PDF version of the one that appeared in a recent issue of *eAI Journal*, the leading resource for e-business, application integration, and Web services.**

Integrating the Interconnected World™



# enterprise integrity



By DAVID MCGOVERAN

## Data Integration, Part IV

Discovering data element semantics is a daunting task that was all too familiar to enterprise data modelers and database designers a decade ago. Insufficient attention to data semantics exacerbates common integration problems, which include its high initial cost and, often, inadequate return on investment. If all applications were designed, developed, and operated based on an explicit data model, we'd then have a hope of developing an enterprise data model with consistent semantics. The model would guide data transformations between applications. Semantic data integration would be easy. Unfortunately, reality is unkind; such models rarely exist and we're usually forced to investigate the correct uses of a data element to discover data semantics in existing applications.

Data semantics are often recorded implicitly as constraints and relationships. For example, validation checks on data entry fields serve to constrain the domain of the data type. Similarly, transactions that read from and write to a data store generally enforce data semantics to some degree. If integrity constraints are fully defined and the data store is transactionally consistent from an external perspective, then semantics are preserved. However, we cannot know those semantics unless we know the definition of each integrity constraint and transaction. Well-designed *logical* transactions provide important clues to data semantics. In the ideal transaction, all encapsulated data elements are logically related and change together as a unit of consistency.

Again, reality intrudes to make our task more difficult. Application developers often combine some logical transactions into a single *physical transaction* (a unit of recovery) and split others into multiple physical transactions. Worse, transaction code may be interleaved with other code or may not even have explicit boundaries. The result is that it's difficult to discern with accuracy either the semantic relationships among data types or when constraints on values pertain to the data type vs. a specific use of that type.


Too often, application-specific data stores use application code to ensure transactional consistency from an internal (i.e., application-specific) perspective, rather than enforcing it within the Database Management System (DBMS) for all potential users and applications. The result is that an unconstrained user, such as an Enterprise Application Integration (EAI) adapter, can access data in an intermediate (i.e., inconsistent) state of processing. Even an omniscient analyst cannot specify the semantics of such data precisely because the

meaning of transactions leaves such intermediate states undefined. This is terrible news for the data integrator attempting to ensure consistent semantics between applications.

Many commercial applications as well as some custom applications restrict data access to an Application Program Interface (API) library to guarantee transactional integrity of part of the data. Such applications either hide transaction management — so each API procedure contains one or more logical transactions — or else provide special API procedures to begin, end, and abort a transaction. Both API approaches make it difficult to discover transactional integrity constraints (and therefore data semantics).

If each API access guarantees transactional integrity, transactional semantics are embedded within each API procedure. Unfortunately, that guarantee cannot extend across multiple accesses without true transaction management. For example, consider an API library for a purchasing application. The API may easily enforce the semantic rule that each line item must be associated with a purchase order and an approved vendor (e.g., by requiring this information as procedure arguments). It can also enforce credit limit semantics. Suppose further, however, that the application also provides API-based transactions that edit an existing purchase order incrementally, adding new line items or deleting existing line items in any order. Without transactions that span multiple API accesses, a reporting function will then produce false (i.e., intermediate) purchase order totals if run between line item edits.

Using special API procedures for beginning and ending a transaction places a significant burden on the programmer to design transactions correctly. Assuming the developer was successful, the data integrator must minimally examine the API code between each transaction (begin and end) to determine transaction semantics. Such APIs are among the most difficult for the data integrator to analyze.

Code analysis is a costly, risky approach to documenting data semantics. To avoid it, aggressively pursue semantically consistent data models for every application and from every vendor. Along with well-designed, encapsulated logical transactions, data models can go a long way toward reducing the data integration costs of enterprise integrity. 

*David McGoveran is president of Alternative Technologies, Inc. He has more than 20 years' experience with mission-critical applications and has authored numerous technical articles on application integration. e-Mail: [mcgoveran@alternativetech.com](mailto:mcgoveran@alternativetech.com); Website: [www.alternativetech.com](http://www.alternativetech.com).*