enterprise integrity

By DAVID McGOVERAN

Data Modeling

Where has all the data modeling gone? Over the last four or five years I've become aware that increasingly fewer IT projects bother with data modeling. That's a pretty serious mistake, especially where controlling new software costs and integrating inherited assets (the politically correct term for legacy systems) are concerned. Obviously, most IT projects involve either developing new software, or integrating a new application or new functionality with existing software.

True, there are lots of one-off development and point-to-point integration projects. But, hopefully, we are learning that the long-term costs of these projects seriously outweigh any perceived short-term savings from the "code it and deploy it" cowboy programmer approach, the current craze of extreme programming notwithstanding.

You need to understand logical data relationships—all of them

We're increasingly encouraged to use a model-driven approach. I wholeheartedly endorse that concept. Unfortunately, "model-driven" all too often means a little more than a variation on visual programming. It's hard for me to talk about development environment being "model-driven" with a straight face when the so-called model is just a set of visual diagrams and text specifications with little or no science behind them. Mostly, these are efficiency devices—engineering tricks that help a developer conceptualize the program and which will result in a working application.

That's certainly a good thing, but we need to step back and think about how to do better. Consider how data is handled in these tools, or perhaps I should say how data is not handled. In my experience, few developers understand the difference between conceptual, external, and internal (or to use another set of terms, conceptual, logical, and physical) data models or why anyone should even care. The tools they work with do not encourage them to learn. For example, while J2EE application servers have made a lot of the drudgery concerning performance and run-time resource management easier, they have failed to integrate with databases at the conceptual and logical level of design.

Here's the thing. In order to understand data semantics, you have to understand how each data element is intended to be used. And I mean functional or business use, not physical use like access patterns or storage methods. That is, you need to understand logical data relationships—all of them—in order to understand which data elements have natural cohesion and which are more loosely coupled. Dependency theory, and the normalization procedure based on it, is the bit of science that can tell you the inherent "clumping": It tells you which elements are properties of which entities and how those entities are related. To put this another way for my Java friends, it tells you what classes you need and how those classes are related before you make any implementation decisions or do any optimization for implementation-specific reasons. The resulting logical class model should not have any physical data types assigned to class attributes, as so often happens in a UML class model. And all too often, a much worse mistake is made because the class model is not consistent with the dependencies among attributes. The result is inconsistent data usage leading to higher costs for user training, application maintenance, component integration, and application integration.

If a logical data model (by definition, complete and consistent) existed for each application's data store, the many inconsistencies with which we deal during application integration (or mere enhancement) would be relatively easy to overcome. But when an application is designed without the benefit of a logical data model, or at least a class model that is consistent with the logical data model implied by dependencies, integration and enhancement are prone to costly errors. Why have we forgotten this? Did we really expect object orientation, component models, or service-oriented architectures to change these proven fundamentals? If so, how?

Recently a researcher that I generally respect pontificated on the future of XML schemas and enterprise information integration. He suggested that we focus on using XML schemas for federated data integration. Data modeling was just too hard for developers. The gist of the conclusion I drew was "schemas, schemas everywhere, and not a bit (byte?) with sense." It's a trivial exercise to come up with many data sources with very different semantics but with the same XML schema: The real world of business is a lot more clever at creating subtle, consequential nuances of meaning. Can we expect anything but confusion and high enterprise maintenance costs from development tools and strategies that ignore the fundamentals of enterprise data integrity? BIJ

David McGoveran is president of Alternative Technologies, Inc. He has more than 20 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.