Alternative Technologies

Enterprise Integrity: Data Integration II

Vol. 4, No. 7

As discussed in last month's column, we often treat the restructuring and reformatting of data as the work of data integration. This is deceptive because it ignores the very difficult problems of rectifying meaning among disparate sources. Fundamental to solving this problem is an understanding of the operational nature of semantics. Without going into a philosophical justification, I will simply assert that the meaning of data in a data store is determined by its permissible uses. The key phrase here is 'permissible uses.' Any actual uses are at best examples of permissible uses and should be treated as incidental.

Ideally, the permissible uses of data from each particular data source should be recorded explicitly in an open repository as declarative rules. It does not matter whether those permissible uses are recorded positively (what the data values *must* be) or negatively (what the data values *must not* be). Traditionally we call these rules integrity constraints. In practice, both positive and negative forms of integrity constraints are used, and are easily combined and correlated. For example, we may state that a data element must be a positive integer (and by inference not a fraction) and also that it must not be greater than twelve. Such constraints would, of course, be appropriate to a numeric representation of 'month.' As we shall see, there are many other types of constraints which together serve to define permissible use.

Unfortunately, the permissible uses of data are rarely recorded explicitly as independent constraints, which makes the task of semantic data integration a rather difficult one. Most often integrity constraints are implicit. Application code, whether Java, C++, HTML or XML, that manipulates a data element in a semantically correct manner clearly embodies a permissible use of that element. If such code is at all robust, it will contain code to constrain data values and enforce relationships among data elements. Some coded integrity constraints will, of course, apply to the use of a data element for a particular purpose or context, such as a particular transaction. Other coded integrity constraints are more general and apply to the data element in all contexts. Mining such implicit records for data semantics is a formidable task and one to which we will return next month.

Explicit recording of semantics is tantamount to maintaining a repository of constraints. Creating such a "constraint repository" might seem like a smaller task than creating a full metadata repository. In fact, it is not. Integrity constraints are not very useful unless strong data typing is enforced, and this cannot be done unless each data element is identified by its abstract data type (known as a relational domain or an object oriented class). When a particular abstract data type is used in a specific context with more restrictive integrity constraints, the abstract

data type is sub-classed. If fewer constraints apply, then generalization is required. A useful repository thus requires an enterprise data model that describes the semantic relationships among all abstract data types. I emphasize that an enterprise data model is required because data sources are likely to have been independently designed and populated. It is essential that the repository be able to capture a data model that represents all the data sources involved in data integration.

The problem of semantic data incompatibility is *fundamentally unsolvable* if there is no known and enforced model of the data sources. I don't mean that you can't analyze each data transformation requirement and, using the available tools, provide the necessary data integration. However, the task is laborious and oft repeated, sometimes resulting in inconsistent transformations and improper semantics. Once data is removed from the controlled environment of an application-specific data store, data integrity, and therefore data semantics, is in question. Part of the reason for this is the loss of transactional integrity control (I'll discuss transactional integrity and its impact on semantics in more detail in a future column).

Of course, developing an enterprise data model up front is a costly barrier to achieving any reasonable rate of return from data integration. The obvious solution is to follow a methodology that permits incremental development so that an incremental and continuous return on investment is achievable. Considerable discipline is required to maintain the independence (i.e., isolation) between conceptual, logical, and physical data models that is necessary for incremental design. Otherwise, semantic inconsistencies result that cannot be resolved without iterative re-integration. And then we pay and pay for want of a little *enterprise integrity*.

.