Alternative Technologies

# Enterprise Integrity: Why Declarative?

## Vol. 4, No. 5

Since my XML databases article (eAI Journal, October 2001), I've received many letters and emails indicating a surprising lack of understanding of (and considerable controversy over) the concept of declarative languages. This month I'll explain why the concept is so important and what I mean by a purely declarative language.

Clearly, computer systems require some form of physical implementation. This physical implementation appears in many forms including, for example, the computer hardware used, the physical organization of data, and the algorithms used to service some particular user request. Though most of us realize that the physical implementation of a service is not the same as the service, we often forget. As soon as we depart from a limited, pre-programmed servicing of user requests, we have to provide access to a general purpose language for making requests.

Most general purpose languages are non-declarative, so that the user must know something about physical implementations and specify precisely how to accomplish each task. If they need to access existing data, they will need to know how that data is physically organized. Such languages have a procedural element to them, meaning that they must be able to take advantage of order. (If that's not obvious, try to imagine the concept of "next" or "previous" data element without those elements being ordered. Then try to imagine completely non-physical ordering.) Of course, the average user won't know how to use the procedural elements of a computer language.

Non-declarative languages have three major problems. First, the more procedural the grammar rules, the more difficult the language to learn *even* if the language is graphical (consider modal versus non-modal graphical user interfaces). Second and closely related, procedural elements tend to be used incorrectly much more often than non-procedural elements. A number of studies in the 1950s and 1960s identified the most frequently occurring programming errors. Elements having to do with order were the culprits: the exit conditions of loops, if-then-else sequences, sorting routines, control transfers ("go to"), and the like. Third, because procedural elements expose physical organization and structure to users, changes to that physical organization and structure cause costly and error prone maintenance efforts.

Consider the alternative, the creation of which was driven in part by consideration of these issues. Suppose that users could concentrate simply on what they wanted to achieve rather than on how to obtain it. They would simply *declare* the goal. This approach requires that the system translate the goal declaration into a set of component procedures that can be invoked as needed, and produce a result guaranteed to achieve the declared goal.

Purely declarative languages need to be semantically rich so that users can accurately express goals. Indeed, a declarative language is all about semantics or intended meaning. Since we don't have mind reading software that determines user intent, the language should be ambiguous. The software that processes a declarative user request must encapsulate physical data and hardware organization and the procedures that manipulate that organization. This means the fundamental operations must preserve information integrity – information is never augmented, altered, or lost except in ways that are specified explicitly by the user.

When a language exposes physical data organization to the user, its declarative power is degraded. For example, URLs are both hierarchical (and so have inherent order) and physical. Worse, there is no semantic model by which a content goal (based on meaning) can be translated into that physical location. XML and the languages and facilities that derive from XML mimic this organization. Query languages for XML are replete with the language of order: occurrences, sequences, paths, steps, descendants, children, and so on. Just because these languages "have no procedures" doesn't mean they are non-procedural (a naïve understanding of "procedural"): if operation order changes results, the language is procedural. Even SQL now has many procedural elements, its declarative power greatly diminished by the failure to implement physical and logical data independence.

Its not that procedural languages aren't useful, but we should limit their use because of the high price we must pay. We are now facing a future with high training and maintenance costs. The next time you encounter a broken Web link or a page that no longer contains the information pointed to, or have to change links or queries (whether SQL or X-Query) when you reorganize your data, you can dream fondly of declarative languages. They can have a positive impact on your *enterprise integrity*.

.


www.AlternativeTech.com mcgoveran@AlternativeTech.com