XML and the future of risk management

Risk managers need to know how to use XML. Without it, they will be stuck in an era of constantly changing data formats, argues David Rowe

Truly enterprise-wide risk management continues to elude many large financial organisations. Why? In terms of technology, it is mainly because of fragmentation – disparate computer systems across both product and region. So far, the most common attempt to remedy this has been to build a central data warehouse that captures all the relevant data from these systems and stores them in a single consistent format.

The problem here, though, is that this is a never-ending task. This is especially true in the capital markets, where new products and new variations of old ones mean that the format of the data feeds into the central warehouse continually has to be revised and extended.

Now, a new technical innovation called eXtensible Markup Language (XML), offers the promise of improvement. Risk managers should understand this new technique and be aware of its potential.

FpML, NTM and all that

Several initiatives are under way to establish an XML-based standard for describing financial transactions and events. A consortium led by JP Morgan and PricewaterhouseCoopers is proposing one standard called Financial Product Markup Language (FpML). Infinity has proposed the Network Trade Model (NTM) and is also participating in the FpML consortium. Eventually, a consensus will emerge on a core of standard conventions. When this happens, it will be possible to move beyond the huge number of bilateral interface formats between specific systems.

Each system will only need to be able to send and receive data formatted according to the standard XML-based conventions. Financial data consolidation across internal systems will be significantly easier. Even more dramatic is the prospect of automated trade confirmations between otherwise incompatible systems. This would contribute to a meaningful reduction in the current level of operational risk.

So what exactly is XML and where does it come from? Both XML and the widely known HyperText Markup Language (HTML) are descendants of an earlier language called Standard General Markup Language (SGML). Unfortunately, using the term "mark-up language" for all three of these is misleading. SGML might better be called Standard General Meta Language, because it is a tool for creating specific mark-up languages. It exists at a higher level of abstraction than any specific language created by it.

HTML, in contrast, is an actual mark-up language created using SGML. It is the foundation for distributing information over the Internet. It allows a data packet to be self-describing in one important but limited respect; namely, how it is to be displayed by the receiving application (usually an Internet browser).

David Rowe is senior vice-president of risk management at Infinity, a SunGard company
e-mail: david_rowe@infinity.com

HTML does not permit self-description of the informational content of the data. Even more important, the "conventions of self-description" in HTML are static, not dynamic. This was the reason for the incompatibility among browsers a few years ago as new HTML "standards" emerged with uncomfortable regularity. Browser versions designed to deal with one set of self-describing conventions could not handle a later convention that was embedded in a particular Web site.

XML is also a descendent of SGML, but in a different sense. Like SGML, XML is not really a mark-up language at all, but a more restricted meta language based on the more general foundation of SGML. XML is a tool for creating specific mark-up languages that have various functions. Its most important use, from a risk management perspective, is to define self-describing conventions for various kinds of data transmissions. These conventions are contained in what are called document-type definitions (DTDs). A specific data transmission can include a DTD that defines the structure to which the data are supposed to conform.

Semantic consistency

We cannot construct an abstract language that describes all existing and possible future features of a security or derivative transaction. Therefore, we must fall back on mutually agreed conventions for how such features are to be represented. Using XML, such an agreement is manifested in a specific DTD. This defines the kinds of information that are either permitted or required, whether there are zero, one or many values of a given type. It also specifies tags used to represent specific concepts. Once such a DTD is defined, XML provides syntactical checking for any particular transmission. Matching opening and closing tags will be checked, the existence of unauthorised tags within any given part of an actual data transmission will be detected and the presence of values specified as required in the DTD will be assured.

Of more interest is semantic checking. Do the values, while syntactically correct, make sense individually and in combination? If a value is specified as a floating-point number, XML will not produce an error if the actual information that follows equals "ABC". If all elements in a numeric schedule are specified as having to total zero, XML itself will *not* check that this condition is satisfied by the actual data. Such semantic requirements can be specified in a DTD, but the logic to understand and enforce them must be engineered in each particular application.

Most important, such a DTD is extensible. New features can be added or new field types allowed, for example, without disrupting existing applications that do not utilise these new features. Just like the problem with the browser wars, however, an old application will not cope with a data feed that takes full advantage of extensions to the DTD. This is because it will not have the logic to deal with newly permitted features of the data. In effect, any given convention for describing financial transactions or life-cycle events is at the same level of abstraction as a particular version of HTML and can be correctly considered a mark-up language. As long as later versions are true extensions and do not change any conventions of an earlier version, there is no need to revise existing data feeds. ■

The author would like to thank Kerry Williams for his help in writing this article