



XML Purposes

White Paper

Abstract

The purpose of this document is to provide a general analysis of the uses for Xml (eXtensible Markup Language) for purposes in three categories: Communication, storage, and convergence. Each section will compare Xml to other technologies which attempt to solve the same problem in different ways, and provide comparisons for the benefits of each system. The primary allure of Xml is, in a word, readability. Data is stored in the clear, and may be read or modified by users with nothing more than an ordinary text editor. Because of this, the use of Xml bypasses many of the typical hazards of cross-platform compliance, such as byte ordering and other processor architecture differences. The combination of usability and portability has made Xml very popular for use in applications from web data access to the storage of application configuration data.

Tometa creates custom software for you

Tometa Software designs and develops robust software solutions for virtually all industries including in-house (vertical market) and retail software, some of which is on the shelves at your local software store. We focus our unique combination of creative, technical, and problem-solving skills on meeting our client's objectives. Because of our clarity of purpose, commitment to process, and broad professional skill sets, we are able to provide our clients with world-class solutions that are functionally superior and fully aligned with our client's strategic focus.

Balancing development speed, quality and cost is what we are all about. Tometa combines agile development practices with fixed pricing, so you know what the cost, end product, and delivery time table look like—up front. If we underestimate the effort, we complete the overrun on our dime. Simple as that. That's why large enterprise firms like Alcoa and NASDAQ choose Tometa.

Tometa's agile development expertise and low-overhead US location keep our prices comparable to offshore vendors – without offshore challenges. Using a fixed pricing model, we provide upfront visibility into a project's ultimate costs, end product and delivery schedule. Our clients like knowing that we have “skin in the game” – a fixed price that aligns our goals with yours, incenting us to get the job done right and fast.

Lastly, as a Microsoft Certified Gold Partner, Tometa Software, can customize its products or create custom web, client/server, and traditional applications. With programming experience in C#, C++, Visual Basic, CGI, HTML, RPG, Delphi, Java and many others; Tometa Software is uniquely positioned to meet your needs as a development firm.

[Check us out today](#)

Introduction

In the world of computing, with only a few exceptions, most applications are either built upon or forgotten. If the application is built upon, it will be used and find its use extended either explicitly by the programmer or database engineer, or implicitly by other applications. It has increasingly become the case that applications which lend themselves to extension have lasted longer than those which do not.

Enter Xml. Xml is a ready-made standard which allows the annotation and specification of human-readable data. The standard itself only specifies acceptable language constructs, but places very few restrictions on the way these constructs may be used. For this reason Xml may even be considered a data language, the way that C++ is a programming language. And just as C++ is most appropriate for the construction of programs, Xml is most appropriate for the construction of data protocols.

Communication

As a data protocol, it seems clear that Xml lends itself well to the transmission of data between two processes, which may potentially be running on different machines or even different platforms. Asynchronous JavaScript and Xml, or AJAX, is a programming technique which uses the Xml standard for object transmission. The Web Service Description Language, or WSDL, is a Microsoft standard which allows a programmer to specify an HTTP-based procedure's calling convention. Parameters, return values, and even the specification of the procedure itself are all defined using Xml.

Some applications go even further than this. It isn't rare to see management-style applications transmitting pure Xml back and forth to coordinate a transaction. The entire communication is wrapped in a "Connection" tag, or something similar, and the communication is considered finished when the connection tag is closed. In addition to being an aesthetically appealing and easy concept, this communication architecture method enables programmers to reuse the abundant and generally efficient Xml-parsing machinery present in just about every major language and platform in common use today.

In spite of its benefits, performance, security, and data compatibility are factors which must be considered before using Xml as a low-level communication protocol in the manner listed above.

Performance

An efficient Xml implementation is slower and consumes more space than an efficient packed data equivalent. This is an inescapable conclusion which stems from the fact that Xml data is encoded in the clear, requires closing tags, has language constructs which do not by themselves convey any additional meaning, and must be parsed. It would be foolish to underestimate the resulting overhead, especially in a performance-sensitive application.

Consider a server application which transmits commands to its clients. The command is identified by the name of the tag, and the command text is enclosed by the tag. In the case of a packed data implementation, four bytes of data in each message could encode the message type, followed by two bytes for the message length, followed by the length of the message itself. By comparison, the server implementation would use at least seven bytes just to encode the opening and closing tag, and would more likely require as many as fifteen. If the message text is binary, then an additional base-64 translation will be necessary, even further inflating the size.

The packed message's four-byte type identifier may be processed by a switch-case construct, whereas the Xml protocol's message type must be evaluated by a string searching routine after parsing takes place. Packed messages can be interpreted and acted upon in a single pass, whereas Xml data requires (at the least) two passes.

Measures do exist, however, which can increase the efficiency of Xml. Two implementations of binary Xml, for example, exist which can improve the processing and space efficiency of Xml: A raw binary implementation, and a state-machine implementation.

The state machine implementation runs a state machine on both the client and server, and uses a packed data protocol in between. Tags and attributes are processed before transmission, and tag names are given identifiers the first time they are encountered. The XML data is then reconstructed at the destination. The system's space efficiency is close to that of a packed data protocol, but the time performance is clearly not comparable. State machines are elegant in that they can easily replace a pure Xml transmission system, but do not escape the actual requirement that Xml data be parsed—they simply shift the task of parsing it from the client to the server.

The raw binary implementation uses identification numbers for everything except the actual data. It maintains no states, but because translation does not occur at the recipient back into Xml, it requires that the recipient know the ID numbers used in the place of tag and attribute names. This solution's space and speed efficiencies are very close to that of a packed data protocol, but the obvious problem is that there isn't actually any Xml in this implementation. In practice, raw binary implementations are little more than formalizations of packed data protocols, and because they cannot easily replace Xml systems, more forethought is required before implementation.

Security

In any communication system, security should always be a consideration. Security is typically provided by a combination of obfuscation and cryptography, though the strongest implementations do not gain anything from obfuscation.

Before proceeding, it should be made clear that a determined hacker will be able to breach a weak cryptographic protocol, whether it is implemented in

Xml or by a packed protocol. Ultimately, the security of a system should be rooted firmly in the security of the cryptographic algorithms used to implement the system. It should also be noted, however, that this kind of determined hacker is rare.

Except for cases where both peers share the same key, at least some of the communication between them will need to be in the clear. If a packed data protocol is being used, it takes time for an adversary to discover the protocols being used, and then even more time for the adversary to determine what attacks can be made on the system. Even if all of the communication between the client and the server is made in the clear, discovery of the data protocols can be a very difficult task.

With an Xml -based communication system, however, all of the security offered by protocol obscurity is lost. The system itself is exposed to the adversary with little or no analysis required. This is not a failing of Xml: Quite the opposite, it is one of its features. The standardization is intended to make it easy for unlike applications to communicate, even without documentation on the communication protocol being used.

Interoperability

The same transparency which makes it possible to easily exploit flaws in an Xml Xml-based protocol also makes it very easy to extend (thus, the name Xml). It could be said that it is just this transparency is the key justifying principal for the use of the Xml format.

Because protocol information is provided with the protocol itself, it is not necessary to document an Xml protocol the same way that it would be necessary to document a packed data, tokenized, or other proprietary protocol. Projects involving many members would benefit from Xml -based protocols because the sorts of protocol issues which typically arise when many members are participating can usually be recognized with much less expenditure of effort.

It is for this very reason, however, that the use of Xml in production communications systems is sometimes considered to be a strategy used where proper documentation and source maintenance would be more appropriate. Protocol conflicts and problems which require protocol analysis on a product created by the same organization requiring the analysis usually indicates a severe issue with the development strategy chosen for the development of the product. If the decision *is* made to use Xml in such a circumstance, however, then typically the raw Xml is replaced before release with a binary- Xml equivalent.

Storage

The use of Xml as a persistent storage format is becoming increasingly common for configuration information, as an intermediate database format, and for semantic annotation.

Configuration Files

Xml lends itself well to small configuration files, or files which are loaded infrequently. The first, and most obvious, benefit is that an Xml-based configuration file can be modified in emergencies where the normal configuration utility may not be available. This may be especially important if the configuration file is so corrupted that the utility is unable to open it, or if settings must be changed that cannot be changed with the utility due to user interface inadequacies.

Xml as a configuration format also allows a software developer to completely eliminate the need for a configuration utility in the first place. Provided that the target audience is capable enough to modify Xml, the raw Xml configuration file may even lend itself more easily to such kinds of modification.

Additionally, there are hundreds of software utilities available for the direct modification of Xml data. Internet Explorer has a built-in parser for Xml, and almost every software development IDE supports Xml syntax highlighting.

On the negative side, however, is that Xml format can only store tree-type data natively. This means that a configuration file which must represent a graph—for example, a topology or a relationship table—must perform searches to reconstruct the graph. Using sequential ID numbers is a potential solution, but this damages the format's human modifiability.

Intermediate Database Format

Xml is a lowest-common-denominator format which can represent very complex data. Database architectures are hierarchical, and can therefore be translated without searches into Xml. If data must be migrated from extremely dissimilar systems, Xml may prove to be the most easily implemented option available to a programmer.

Xml's linear nature, however, makes it unsuitable as a primary database format. All efficient static database formats implement highly optimized searching tables or organizational systems which allow traversal through the database by accessing elements randomly. For large databases, the search speed can be millions of times slower.

Semantic Annotation

Because of the flexibility of Xml, it is possible to use it to provide semantic information for human-readable data. The most appropriate case for such a use would be any information repository designed to amass. Examples of such systems would be online news sources, encyclopedias, and information-centered forums.

It is possible to provide extra information about the data being represented in any of these systems by using Xml without hampering the user's ability to read such data. The purpose of this type of annotation would be to improve the accuracy of searching systems. It typically falls upon the person generating the

data to annotate the data properly, but as Wikipedia's success has demonstrated, proper annotation can be accomplished with the adequate application of feedback mechanisms.

Convergence

The process of standardization is itself an extremely difficult one. If one proposes a standard too early, the standard doesn't consider enough information and is therefore not adopted. If the standard is proposed too late, too many people will have adopted their own techniques, and the standard will be ignored. Some examples of standards which came too early or too late would be JavaScript, the OSI layers, and the rewritable DVD.

Xml escapes some of this difficulty by allowing a standard to evolve within the context of an Xml format. Certainly, this isn't a process that can occur on its own, but Xml provides the general framework and establishes the very basic things necessary for meaningful communication to occur.

An example of a format which has converged just this way is the RSS format. RSS is an Xml-based format which provides news-streaming services to its clients. Before RSS existed, there were several formats already available for syndication. This standard went through many changes, and was even orphaned by its creator, Netscape. After five years of update and modification, the New York Times offered its news services in RSS format, and shortly thereafter RSS 2.0 became a de-facto standard.

Conclusion

Xml is suitable for any case where compatibility is important, and the entire nature of the data being transmitted is uncertain. Xml parsing, however, is not a trivial task, and adds significant overhead to data processing. Xml should only be used for data interchange after it has been determined that performance is not as important as broadness of service.