XML is not Integration

Web Services has been much advertised as a standards based mechanism for cutting IT infrastructure costs, reducing application complexity, and enabling IT to fulfill the ever increasing needs of the businesses it serves. Many industry pundits claim that Web Services can accomplish these great feats because they will finally create a common language for all systems, enabling integration beyond anything that has ever been imagined before.

Well that's just hooey.

While XML is a promising development, it is nothing more than a structured file format. Those of us who have been in IT for any length of time (i.e., before 1995), know that common file formats are a grand service to understanding the way to get data into and out of systems, but it doesn't make the hard job of understanding the interfaces, complex applications, and architecture of those data containing systems any easier. XML doesn't address any of these issues.

Don't get me wrong. I've been a proponent of XML since 1998. XML is a magnificent approach to handling data, both within an application and between applications. Defining the data element within a self defining structure is tremendously beneficial to the developer, the DBA, and the parties responsible for troubleshooting when things go awry. It doesn't, however, answer the need for pre and post op edits on the data contained in it. It doesn't address the mapping of non-direct field definitions between systems, even when DTD's (the "map" of the XML file) are well defined.

XML's main value comes from the ability to easily parse the data contained, and to then manipulate it so that it can be used within the IT ecosystem of the company or companies that use its data. It is also an excellent mechanism for handling information from a very high level when you are not sure what format external systems will need when requesting information from you (i.e. – make them do the investigation and coding to transpose the data).

Integration, as defined from a business perspective, is achieved when systems interoperate in such a way that the work of information processing, mining and extrapolation is done by the system best suited to accomplish it, and the results of these actions is shared with those systems that need the information, but have not produced it themselves.

That heady statement can be boiled down into three functional elements. Integration of systems is basically made up of messaging, data transformation and process flow management

Web Services, through HTTP and XML, attempt to accomplish messaging and data transformation, but leave the job half done. What Web Services accomplish, is data "exposure", not integration. In order to transform data through Web Services, complex coding and deep understanding of the individual points of integration have to be understood by all parties. Only once the producing and consuming system's owners understand all of the others, can the effort of coding the ties between systems begin. This is hardly a paradigm for increased productivity.

Companies such as IBM, Sun Microsystems Oracle, and even Microsoft have created a variety of tools to make the "hard part in the middle" somewhat easier, while pure EAI vendors such as Vitria and WebMethods seek to create monolithic platforms wherein the integration code is developed and managed. These approaches follow the track of a defined EAI methodology similar to the App Server market of the late nineties, with the twist of using XML as the central way of defining data.

What is more useful, is an approach that has at its heart, a system that understands the systems which need to intercommunicate. More than just a data routing engine, such systems understand the complex edits of the systems they interoperate with, and provide a seamless transformation of the transactional and batch data feeds between systems that don't share common interfaces or formats.

SeeBeyond, Mercator Software, and BEA Systems are all aimed at this newly emerging architecture. Leaving traditional EAI mechanisms behind as "growth pains", the concept of an Integration Hub is rapidly growing in the market place. Through such an architecture, programmatic methods of dealing with external data and transactions take a back seat to core business functions and data transformation. Strangely, the value of messaging system investments (such as Tibco Rendezvous and WebsphereMQ) is increased by

decreasing the value of the work done by them, and transferring it to the integration hub. The integration hub greatly reduces the complexity of integration with the applications and databases that make up the company IT ecosystem, while the messaging hub acts as the mechanism to ensure all transactions get to their intended destination. What is most striking, is that integration hub architectures work best when the messaging system is not built in, but an external system altogether.

Through such a three tier architecture, XML interfaces become simply another, but not the core means of dealing with data. XML is then fed to those systems whose optimal input is tagged data structures through an integration hub. To those legacy (and modern) systems which operate with data in some format other than XML (such as COBOL Copybook, ANSI SQL, and structured transaction and file formats), the integration hub speaks to them in their native language.

A new paradigm in integration is rapidly emerging, based on an integration hub architecture. XML is a welcome player in this space, but XML is not integration.