The Cost of Data

Chris Suver, Microsoft

In the past few years people have convinced themselves that they have discovered an overlooked form of data. This new form of data is *semi-structured*. Bosh! There is no new form of data. What folks have discovered is really the effect of economics on data typing—but if you characterize the problem as one of economics, it isn't nearly as exciting. It is, however, much more accurate and valuable. Seeing the reality of semi-structured data clearly can actually lead to improving data processing (in the literal meaning of the term). As long as we look at this through the fogged vision of a “new type of data,” however, we will continue to misunderstand the problem and develop misguided solutions to address it. It's time to change this.

For data to be operated on reliably, either in an application or in a tool (such as a database), it must be typed, or for high reliability, it must be strongly typed. This is necessary because at some point the underlying hardware has to choose the right circuit to process the bits. It has long since been demonstrated that any data can be typed—in fact, strongly typed. Still, today most data is not typed (i.e., structured). This is simply because it is not worth spending the resources to apply typing to the data (i.e., the value of the data simply does not justify the investment).

Typing data incurs a number of costs:

- The design time cost, or the cost actually to determine and apply the types (schema design) to the data, is higher.
- The cost in runtime resources increases. This includes, but is not limited to, CPU costs (i.e., complex joins can be slow).
- Application development is more expensive when we use tools that require the precision necessary for handling strongly typed data. This often includes the cost of understanding (or almost decoding) the type description.
- Last, but by no means least, the cost to the user to actually query the data is higher, as the queries themselves are much more complex (to the point that the majority of users can't actually formulate structured queries with languages such as XQuery or SQL).

A cost/benefit analysis is the real way to differenti-

Semi-structured data IS THE RESULT OF ECONOMICS.

ate data; how valuable is it? If there is sufficient value, someone will type it. If not, no one will type it. This decision has

been made since written records have existed. It hasn't changed with the advent of XML. Simple economics controls whether data is typed or untyped—and simple economics we can understand.

From this premise I contend the term *semi-structured data* can be misleading. Too often I have seen *semi-structured* taken to imply that the data is intrinsically different. I've watched many folks treat this as a proper noun, as a “new thing.” Perhaps if we had chosen the description of *incompletely structured* or *semi-typed*, there would be less confusion (sadly, these don't roll off the tongue as smoothly). The essential point is that the structuring (or typing) of the data is only partially complete. It's not that the data is intrinsically different, nor that it doesn't have type (or can't be typed). It is simply that the effort to fully define the type of the data is not worthwhile. As a result, the author adds only as much type information as necessary to satisfy the immediate needs. Keep in mind that semi-structured data is a task left uncompleted rather than something fundamentally new. It is simply that the information has only part of its type information in place.

By looking at data from this alternative point of view—that typing is something you do as necessary to realize value—we can now look at our tools, features, and products to see how the cost/benefit matches the user's needs.

We can look at examples of how this plays out with some of today's data technologies that we all know and love. We can also look at technologies that have been successful and see why.

At one extreme is the data that is stored today in SQL databases, such as accounting data (demonstrably high-value data). Clearly, the cost of authoring the SQL schema, the queries, and even entering the information is very high. But the value of the data and the value of the

Continued on page 62

Continued from page 64

query results are even higher. High authoring costs, but substantial value in the results, have made these databases a huge success.

At the other extreme is data on the Web. On the whole, this is very low value (exactly how valuable is yet

**We need to use
a cost/benefit
analysis when we look
at data technologies.**

another site for performance-enhancing drugs?). But for the reader, there is virtually no authoring cost. Search engines such as Yahoo, Google, and MSN make the cost to query this data virtually nothing, either in terms of authoring the query or the time to execute. The cost to

“prepare” and to query the data properly reflects its value. Search is heavily used, even though the bulk of its raw data is of no value, and the results are often noisy. Low-value data, but aligned with low cost, makes this a cost/benefit win. As a result, this is one of the most widely used tools on the Web, another clear success.

Between these two extremes is XML, used for text markup—à la HTML and SGML (Standard Generalized Markup Language)—and data transport. Both of these uses need low authoring cost and a soft type system. I refer to XML typing as a soft system because the typing is applied to the data when the XML is processed. Often the types used by the sender and the receiver are different. Sometimes these are large differences, sometimes small, but any difference means that the transport itself must be softly typed so that it can be easily adapted to the different uses.

So, XML’s authoring cost is low. The author can choose to add as much meta-data (i.e., structure) or as little as appropriate. Here, again, is a case where we have an excellent match between the technology and the user. From this point of view it makes sense that XML has been well-received.

It is also informative to look at querying XML in this

light. XPath (version 1.0) is a successful tool to query XML. The use of DTDs (document type definitions) is a popular way to describe XML documents. What about XQuery and XSD (XML Schema Definition)? They have not been as well-received. They have not taken off the way that DTDs and XPath did when they were first introduced. Part of this is timing, but I contend the ultimate reason is that both XQuery and XSD are overly complex. As a direct result, they are overly expensive to use on data that is of modest value. They do, and will, find use with XML as a transport of expensive data—for example, with commercial Web services. Then the cost/benefit analysis works out.

Another example of when the cost doesn't justify the effort is document data. The low data value in most documents can explain why documents are rarely loaded into highly structured stores. While it is clearly possible, in the majority of cases the cost involved simply isn't justified by the return. (There are exceptions to this: Law firms and other users whose business is essentially documents regularly store them in quite sophisticated repositories. This is simply a case where the documents really carry the value necessary to justify this substantial investment. For the typical user this isn't true.)

We should not be talking about the nature of this “new” semi-structured data. Instead, we need to use a cost/benefit analysis when we look at data technologies—including addressing the needs of data without a full type description. What is the cost, how valuable is the data we are targeting, and, when all is said and done, do the users come out ahead? If they do (and it should be very clear), then we have a winner. If they don't, or it's not clear, then we don't. This is the hard and somewhat mundane question we have to ask to be successful. The good news is now, I contend, we have a solid framework to use when evaluating these choices. Q

LOVE IT, HATE IT? LET US KNOW

feedback@acmqueue.com or www.acmqueue.com/forums

CHRIS SUVER is a software architect with Microsoft working on data access and XML in the SQL Server group. His involvement with databases started in the 1970s and encompasses database applications, libraries, tools, languages, and engines. This includes work at Microrim, dbVista, and Cascade/DB, as well as the two startups he founded, Precedent Systems and MicroQuill.

© 2005 ACM 1542-7730/05/1000 \$5.00