```
<Name><email></email></Name>  
<description><![CDATA[Architecting Tomorrow's Compu  
<Person><Name></Name></Person>  
<title>ACM Queue</title> <link><![CDATA[http://www.acmqueue.com/]]></link>
```

Managing Semi-

I vividly remember during my first college class my fascination with the relational database—an information oasis that guaranteed a constant flow of correct, complete, and consistent information at our disposal. In that class I learned how to build a schema for my information, and I learned that to obtain an accurate schema there must be a priori knowledge of the structure and properties of the information to be modeled. I also learned the ER (entity-relationship) model as a basic tool for all further data modeling, as well as the need for an a priori agreement on both the general structure of the information and the vocabularies used by all communities producing, processing, or consuming this information.

Several years later I was working with an organization whose goal was to create a large repository of food recipes. The intent was to include recipes from around the world and their nutritional information, as well as the historical and cultural aspects of food creation.

I was involved in creating the database schema to hold this information. Suddenly the axioms I had learned in school collapsed. There was no way we could know in advance what kind of schema was necessary to describe French, Chinese, Indian, and Ethiopian recipes. The information that we had to model was practically unbound and unknown. There was no common vocabulary. The available information was contained mostly

in natural language descriptions; even with significant effort, modeling it using entities and relationships would have been impossible. Asking a cook to enter the data in tables, rows, objects, or XML elements was unthinkable, and building an entry form for such flexible and unpredictable information structures was difficult, if not impossible. The project stopped. Years later I believe we still do not have such information available to us in the way we envisioned it.

Many projects of this kind are all around us. While the traditional data modeling and management techniques are useful and appropriate for a wide range of applications—as the huge success of relational databases attests—in other cases those traditional techniques do not work. A large volume of information is still unavailable and unexploited because the existing data modeling and management tools are not adapted to the reality of such information.

SEMI-STRUCTURED DATA

During the 1990s, the Web changed the digital information rules. The extreme simplicity of HTML and the universality of HTTP decreased the cost of authoring and exchanging information. We were suddenly exposed to a huge volume of information; this kind of information was, of course, not new, but the volume was unlike any-

Structured Data

DANIELA FLORESCU, ORACLE

Managing Semi-Structured Data

thing seen before. The impact in our daily lives was also tremendous. It became clear that this rich information could not be stored in relational databases or queried and processed using traditional techniques; we had reached the limits of what we could handle using the traditional rules. We needed new technologies.

In addition to the pure (unstructured) HTML data on the Web, more data was available in a form that did not fit the purely structured relational model, yet the information had a definite structure—it was not “just text.” This gray area of information was called semi-structured data. A lot of research has been devoted to this topic in the database community and elsewhere. Unfortunately, almost 10 years later we still do not have good solutions, software, tools, or methodologies to manipulate this kind of information. Computer science students still do not learn how to deal with it. We do not even agree on the shape of the problem—much less, good approaches to solving it.

The first part of the problem is the fuzzy definition of the term *semi-structured data*. I classify it as all the digital information that cannot be easily and efficiently modeled using traditional schema tools, software, or methodologies. Most such problems relate to the mismatch between the information we need to model and the current tools’ requirements for a priori simple schemas to describe the information. The information we currently need to handle has a complex and subtle relationship with schemas.

The reasons the information cannot be easily and efficiently modeled and processed using existing methodologies may be wildly different. Handling each such case of semi-structured data might require different techniques and solutions.

Probably the most frequent case of semi-structured information is simply *unstructured* information—that is, data embedded in natural text. This information has no simple structure associated with it—much less, schemas to describe such structures. A large percentage of the world’s information is contained in Word, PDF, TIFF, HTML, and other such file types. The constant evolution and improvement of search engines have made this information accessible (to a certain extent and with various

degrees of quality) to human readers. Yet automatically extracting information from it is a problem, as natural language understanding and information extraction tools are still simplistic. E-mail is a typical example. Despite the advances of natural language understanding, we still do not have good-quality tools to search, classify, and automatically process e-mail messages.

To a large degree, the reason we are unable to deal effectively with information buried in natural language is that most tools for automatically processing information require the information to be modeled under some variant of an entity-relationship schema. The ER model isn’t an adequate choice for modeling natural text: people communicate in sentences, not entities and relationships. For example, consider the content of a legal document. One can extract a subset of the information contained in the document under an ER form, yet any attempt to do so will degrade much of the original content.

Another common problem in managing today’s information is the lack of agreement on vocabularies and schemas. Existing information-processing methodologies require that all the communities involved in generating, processing, or consuming the same information agree to a given schema and vocabulary. Unfortunately, different people, organizations, and communities have inherently different ways of modeling the same information. This is independent of the domain to be modeled or the target abstract model being used (e.g., relations, Cobol structures, object classes, XML elements, or RDF [Resource Description Framework] graphs). Reaching schema agreements among different communities is one of the most expensive steps in software design. Database views have been designed to alleviate this problem, yet views do not solve the schema heterogeneity problem in general. We need to be able to process information without requiring such a priori schema and vocabulary agreements among the participants.

Traditional tools require the data schema to be developed prior to the creation of the data. Unfortunately, sometimes the data schema emerges only after the software is already in use—and the schema often changes as the information grows. A typical example is the informa-

tion contained in the item descriptions on eBay. It seems impossible for the eBay developers to define an a priori schema for the information contained in such descriptions. Today, all of this information is stored in raw text and searched using only keywords, significantly limiting its usability. The problem is that the content of item descriptions is known only after new item descriptions are entered into the eBay database. EBay has some standard entities (e.g., buyer, date, ask, bid...), but the meat of the information—the item descriptions—has a rich and evolving structure that isn't captured.

Traditional software design methodology does not work in such cases. One cannot rigidly follow the steps:

1. Gather knowledge about the data to be manipulated by the software components being designed.
2. Design a schema to model this information.
3. Populate the schema with data.

We need software and methodologies that allow a more flexible process in which the steps are interleaved freely, while at the same time allowing us to process this information automatically.

Often the structure of the information evolves as the information progresses through various stages of processing, and the process cannot be anticipated statically. Imagine, for example, a medical form containing the results of laboratory tests and medical examinations, filled in by successive medical investigations. The results of some tests trigger new tests. It is impossible to know the information structure until the process ends; as the process unfolds, the information is filled in. We need methodologies that are able to capture and effectively exploit these kinds of dependencies between schemas and processes.

The information structure often evolves and is refined over time, as the information ages and is better understood. Let's consider as an example the data obtained as a result of scientific experiments. Over time, the scientists' understanding of a certain scientific fact are refined, and as a consequence the schema describing this information evolves. The difficulties encountered while processing this kind of iteratively refined information motivated the original work on semi-structured databases, which still constitutes a significant effort on the part of the database community.

The popular schema languages are generally too simplistic to model the increasingly complex and dynamic information structures. Because of this mismatch, in some cases, even if schemas exist, the result is unfortunately the same as in the previous cases: "rich structure" often translates in practice to "no structure." For example, the

commonly used relational and object-oriented schema languages lack adequate support for describing alternative structures (e.g., authors or editors for books), and for conditional and correlated structures. Examples of such correlations that are difficult to model in existing schema languages are co-occurrence constraints (e.g., if the attribute *employer* is present, then the attribute *salary* is also present) and value-based constraints (e.g., if the attribute *married* has value *yes*, then the person also has to have an attribute called *spouse-name*). Very often such cases are represented using a union of all known properties or, even worse, as a global lack of structure.

These are only a few of the schema-related challenges that we face while modeling and processing nontraditional information. There are, of course, many more. A natural question arises: Aren't those just traditional cases of schema evolution? Indeed, semi-structured data can be seen and explained as the extreme case of schema evolution, wherein the data has a complex relationship with the schemas describing it. The data may or may not have

We have to learn to use
and exploit schemas as
helpers, but not rely on their
existence or allow them to be
constraining factors.

a schema or multiple schemas; schemas can be unknown statically; schemas can change over time, or change while the data is processing, or simply change extremely fast. Schemas can be very rich and might be difficult to model using the ER model. Schemas can be derived from the data instead of driving the data generation, or schemas can be a posteriori overlaid existing data.

So why bother having schemas at all? The reason is simple: Schemas assign meaning to the data and so allow automatic data search, comparison, and processing.

Managing Semi-Structured Data

While it is true that imposing schemas in the traditional sense limits the evolution of the data and the code that manipulates the data, completely eliminating schemas does not seem to be the right solution either. A balance has to be found; we have to learn to use and exploit schemas as helpers, but not rely on their existence or allow them to be constraining factors.

The reality is that we do not possess good tools, software, and methodologies to deal with semi-structured information. Very often such cases are simply not solved—or they are solved in complex and expensive ways. In many cases information is stored in flat files and then extracted and processed with code. In other cases databases are developed that use schemas in minimal ways, hiding the intelligence of handling the data in the applications that manipulate it. Hiding the intelligence of data manipulation in the programs has many negative consequences, mostly in terms of the cost of building and maintaining such applications, and of the fragility of the resulting code.

WHERE SHOULD WE START?

The academic database community has done a lot of work on semi-structured information over the past decade. Among the proposals are new (graph-based) data models, as well as schema-independent query languages and new storage and indexing methodologies—all entirely schema-agnostic. New methodologies in which schemas are automatically derived from data have also been investigated, resulting in several prototype systems.

Several other fields related to semi-structured data management have reached industrial maturity. As the Internet has grown, so has the importance of good search engines. Today excellent engines offer good-quality answers to simple keyword-based queries over large data volumes. Search engines might be considered the answer to the semi-structured data problem in that they mostly ignore the potential structure of the underlying information. But are search engines the answer to the management of semi-structured information? They may be *an* answer but they are not *the* answer.

Unfortunately, in their current incarnations the search

engines have too many limitations to be the answer to this problem. The data and the queries that such techniques have been designed for are very simple and do not gracefully scale while gradually increasing the degree of structure in the data, the degree of schema knowledge, and the degree of structured search in the queries. In addition to offering human access to this information, it must be available for automatic processing. Programs have to be able to update the data, use complex logic that handles the data, and take automatic actions based on the content. Existing search engines weren't designed with such goals in mind.

The Semantic Web set of technologies developed as part of the W3C is a good starting point for automatic processing of the world's semi- and unstructured data. Data models such as RDF, enriched with declarative ontology descriptions such as OWL (Web Ontology Language) and with automatic classification and inferencing mechanisms, have been specially designed to address such problems. The goal is to add meta-data to the world's structured and unstructured content, and be able to process the meta-data to infer and extract the necessary information. Will the Semantic Web be the solution to managing the world's semi-structured information? My answer is the same as before: While ontologies, classification, and inferencing are essential tools for the management of semi-structured information, they aren't the *only* essential tools, and techniques from other fields will also be required.

An older technology is another possible solution to the semi-structured data problem. After XML was proposed by the W3C in 1998, the academic work on semi-structured data was almost halted, as there was hope that XML was the answer. Almost a decade later, XML is universally accepted and embraced by a variety of communities for many reasons, yet it is now clear that while XML solves a large number of schema-related challenges, it does not solve the general problem of semi-structured information. (In this article, XML refers to the entire standard XML infrastructure developed by W3C, including a set of technologies and languages designed in a consistent fashion: abstract data models, such as Infoset, XQuery 1.0, and

XSLT 2.0; XML Schema; and the declarative processing languages such as XPath, XQuery, and XSLT.)

XML offers some major advantages. As a standard syntax for information, XML is able to model the entire range of information, from totally structured data (e.g., bank account information) to natural text. Having a single model for the entire spectrum of information has tremendous benefits for modeling, storage, indexing, and automatic processing of such information. There is no need to switch from system to system and make inconsistent systems communicate with each other while we increase or decrease the level of structure in information.

Another major advantage of XML is the ability to model mixed content. Having an abstract information model that goes beyond the entity-relationship model opens the door to a large volume of information that was impossible to model with prior techniques. The fact that XML schemas are decoupled from data is also essential for data and schema evolution; data can exist with or without schemas, or with multiple schemas. Schemas can be added after the data has been generated; the data and schema generation can be freely interleaved.

While providing significant advantages for managing semi-structured information, XML-based technologies in their current form and in isolation are not the magic bullet. Most information is still not in XML form, and some of it never will be. The advantages of XML (e.g., complex schemas, mixed context, schema-independent data) unsurprisingly bring an extra level of complexity and many challenges. Finally, XML-related technologies today don't offer a complete solution. For example, while XSLT and XQuery provide good query and transformation (i.e., read-only) languages, there is still no good way of expressing imperative logic over such schema-flexible data, or language to describe complex integrity constraints and assertions. Such limitations will eventually be eliminated, but not immediately.

A solution to the general problem of semi-structured information will need to draw ideas and techniques from many fields, including knowledge representation, XML and markup documents, information retrieval, the Semantic Web, and traditional data management techniques. No single method will provide the answer to this problem.

WHAT'S NEXT?

Much work remains to be done to solve the semi-structured data management problem. We are just at the beginning of a long journey. Here are some of the obvious tasks ahead of us.

First, we need better information-authoring tools. We need to lower the cost of information generation and at the same time increase the quality and degree of structure in the data, which in turn would increase the probability of this information being automatically processed. The current information-authoring tools include document generators (e.g., Word), forms, and XML editors (e.g., XMeta). They are either too simplistic or too difficult to use and only exacerbate the semi-structured data problem. The next generation of such tools not only must be able to generate the information in a form that can be automatically processed (semantically meaningful XML is probably the most appropriate), but also must be easy to use. Finally, they shouldn't impose limitations on the types of information that can be modeled and its potential structure or content. The authoring process can be driven by underlying schemas or not, and/or helped by existing dictionaries and standard vocabularies.

Even with the use of the most sophisticated information-authoring tools, much information will be in pure text. Information extraction techniques will always be important. Research on this topic skyrocketed in the '90s with the desire to extract information automatically from HTML pages and process it with Web applications. This work became unfashionable, as many people believed that XML and Web Services made it irrelevant. Not only is this work still relevant, it is also a major piece of the puzzle in handling semi-structured information. Such techniques include pure extraction; extracting and marking portions of text that refer to certain entities (e.g., names, addresses, companies, cities); de-duplication (e.g., the ability to discover that several such extracted entities refer to the same real-world entity); and correlation, (e.g., the ability to discover that certain marked entities are related through a certain kind of relationship).

More work should be devoted to creating and reusing standard schemas and vocabularies. The role of community-created schemas is drastically increasing. RSS is a typical example of such community-based schemas. Initially proposed by a single person, the RSS schema was refined and embraced by an entire community. When communities decide to author their information according to a common vocabulary and a common schema, the value of the resulting information increases dramatically.

Although the community-based schema-design process worked well for RSS, it might not work for other communities and domains. We may need organisms and processes in place to create, search, and register standard schemas and vocabularies. This was one of the original goals of UDDI (Universal Description, Discovery, and

Managing Semi-Structured Data

Integration) registries; unfortunately, they still haven't materialized.

Even if the process of generating and reusing community-based schemas is in place, there will always be cases in which different people and communities will use different schemas to model the same information domain. We will always need to deal with legacy and independently designed schemas. We need to understand how to automatically map such schemas and vocabularies to each other, and how to automatically rewrite code written for a certain schema into code written for another schema describing the same domain. Interesting research is being done in this area, yet more effort will be needed before the problem is understood and we have usable tools.

In addition to the automatic (or semi-automatic) schema-to-schema mapping tools, we need ways to link existing data automatically to an existing but independently designed schema or ontology. This will eliminate the dependency between data generation and schema and ontology generation. We also need to extract good-quality schemas automatically from existing data and perform incremental maintenance of the generated schemas to fulfill the goal of achieving schema and data independence.

Search engines will improve solving one aspect of the semi-structured data problem: the human consumption of information. Contextual, semantic, and structural information can be exploited to increase the relevance of results of simple textual queries.

Decoupling data from schemas also has a large impact on all the aspects of data processing: storage, indexing, querying and updating, providing transactional support, and so on. Most current techniques rely on static schema information to achieve performance for such tasks. We need to revisit such techniques to guarantee their correctness and performance, even in the absence of schema information or with constantly evolving schema information.

Last but not least, we need to be able to process semi-structured data automatically—in other words, write programs to manipulate it. Currently, most popular

programming languages tightly couple code and schemas, and we need tools and methodologies that can separate them. Several decades ago, relational databases introduced the idea of separating code from the physical organization of the data, so that the physical organization could evolve independently without requiring changes in the code.

Today we need to go one step above—we need to be able to separate the code from the *logical* structure of the data. Similar to the database optimizers that bridge code to physical data organizations, we need a new component that links the code written against a logical data representation to the current logical structure of the data. These structures will be partial, nonstandard, and constantly evolving. Programs need far more data independence.

CONCLUSION

Semi-structured information exists all around us, yet often we are unable to process and use it. The high cost of processing information with existing techniques—because of the current requirement for tight (inflexible) coupling of data, schemas, and code—creates a natural barrier for using this information. We need to find a compromise to the tension between the advantages of having schemas, in terms of better understanding and automatically processing the data, and disadvantages imposed by schemas, in terms of inflexibility and lack of evolution.

Enabling semi-structured information processing that is flexible, cheap, simple, and effective is an important goal. We are still at day one. ☐

LOVE IT, HATE IT? LET US KNOW

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

DANIELA FLORESCU is a consulting member of the technical staff at Oracle Corporation. She was a senior software engineer at BEA Systems and CTO of XQRL Inc. prior to its acquisition by BEA. Together with Jonathan Robie and Don Chamberlin, she developed Quilt, the core language used as the basis for the W3C XML Query Language (XQuery). She is currently an editor of XQuery.

© 2005 ACM 1542-7730/05/1000 \$5.00