Document models and XML Vocabulary Building for Business Users

Susan L. **Spraragen**

Douglas **Lovell**

Abstract

Our work presents an experiment with a modeling tool that captures domain knowledge in a fashion natural to business users while producing formal models for use in IT processes. We demonstrate the use of this tool for designing XML Schemas.

Table of Contents

1. Introduction

Businesses rely on their enterprise data to make informed and correct decisions. XML Schema technologies can be used to help ensure the correctness and validity of enterprise data. How do business executives or content experts realize the benefits of a technology they can not immediately see and understand? How do they know their data is in line with their current required business constraints?

IBM Research is developing a tool to address this very issue. To make data and data modeling accessible to the non-technical community is a considerable task. There is a constant debate as to how much to expose to the non-technical end user while still providing a complete and correct XML Schema for their IT department. The goal of our tool is to enable and include all constituents in the process for accumulating the right data sources. For the executives or content experts, we don't want to burden them with syntactical specifications. At the same time for the IT professional we don't want to develop something so lightweight that the output from the tool, while containing all the business knowledge obtained from subject matter experts, is still basically so useless to them that they need to code from scratch as they would have done with out the use of any such tool. This paper will describe how we approach this challenge.

2. A data modeling tool

As a first approach to the problem of capturing and communicating data needs to others we chose a very open model centered on the terms. Figure 1, “Data Model” presents a UML class diagram that represents the underlying data model for the tool.

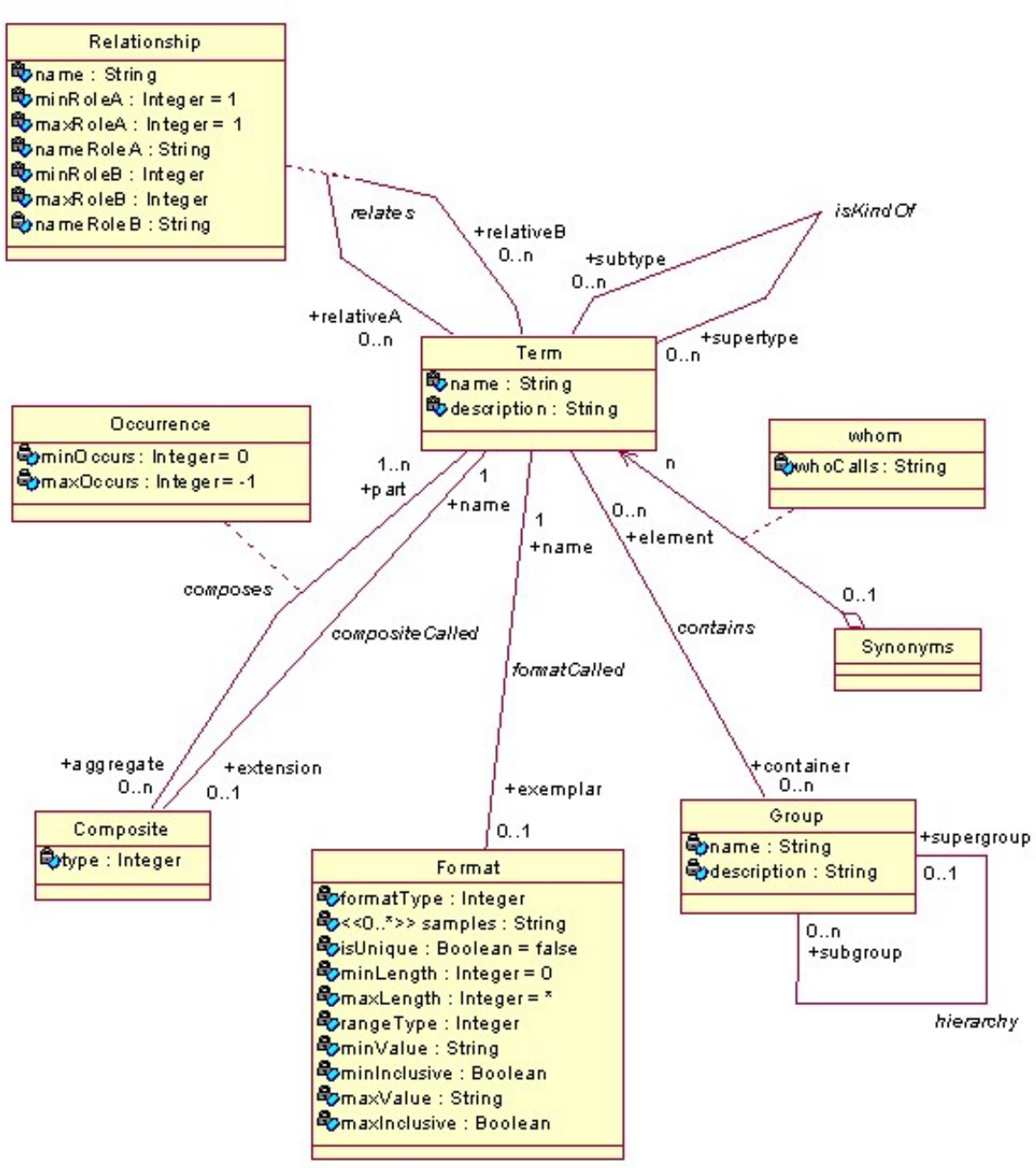

Figure 1. Data Model

Terms are the center of every interaction, and they are at the center of the model. The term, with its name and description is the primitive element in the model that requires no further elaboration. All of the other elements are optional. This way, modelers can easily enter terms without concern for other details.

The tool has six ways to elaborate further information and relationships among terms. Figure 1, “Data Model” shows them arranged around the term. In clockwise order they are the Format, Composite, Relationship, Kinds, Synonyms, and Group. The following sections explain these in further detail.

Our choice of names for these elaborations of terms was hotly debated. We attempted to respect the terms familiar to those in the industry, even when they differed or seemed contrary to computer science terms. Our primary goal was to lower the barrier to entry for data modeling and the power behind it. We strove to make early investigations as to what language is currently in place by people who actively perform data modeling. We believed that forcing a new terminology on the user would be just like having them learn any new syntax that a diagrammatic modeling tool would present.

2.1. Format

The format of a term captures information about values. If a term has a format, it is likely an atomic value. The `formatType` attribute captures whether the value is a number, some text, a date, time, currency, phone number, postal code, etc. Text values may have a minimum and maximum number of characters. Numeric values may have a minimum and maximum number of digits or values. The model is permissive in allowing any string to describe the minimum and maximum values. A modeler may enter the word “zero” or a “0” or “1.5” for example.

The `isUnique` field enables the modeler to specify whether this value is unique to other values that may occur in the data. We do not currently capture any scope for the unique property. A modeler may use the unique property as a reminder that this value is a kind of key.

A term can have only one format and the format, if specified, applies to only one term.

We added the format late in development, when we started to produce Schema from the tool. The XML Schema [3] is very rich in capabilities for expressing data formats and constraints. We believed some users would want to specify that a birth date field has exactly eight digits (works for the next eight millennia), that a text field may have zero to thirty characters, or that a gender field is one character limited to the values, M, F, U, C, A, X, 0, or 1.

2.2. Composite

The composite captures construction of data elements from other data elements. It captures containment or structure. This is the first elaboration we provided for terms because it captures information about what something is made of. It captures how the terms fit within one another. In the language of XML Schema [4], this is how we capture the document model, or structure of elements.

The composite has two relationships with terms. On one side, it has a one-to-one relationship with the term that names the composite. On the other, it has one or more component terms that form the internal structure or content of the composite. The model thus captures one or more terms that come together to form a higher level term.

The type of the composite captures whether the component terms should appear in order, any order, or whether they represent a choice. A choice presents a list of possible component terms, one of which must be present within the term that names the composite.

The occurrence annotation captures the multiplicity of each component term within the term than names the composite. A component may be optional in the composite or it may be required. It may appear only once or it may occur some number of times or any number of times.

2.3. Relationship

Relations between terms capture connections between data other than containment. A term need not participate in any relationships; but if it does, it participates with at least one other term. It takes two to make a relationship.

The model captures a name for the relationship. It also captures role descriptions and multiplicity for the two terms of the relationship.

Most of the lines in the UML logical class diagram (Figure 1, “Data Model”) represent relationships.

We added relationships based on experience we had with a user who was familiar with the construction of Entity-Relationship diagrams as an approach to modeling relational database tables. The Schema generation currently does nothing with this information, but we believe we can capture the semantics of relationships in a document Schema using `ID` and `IDREF` token data types.

2.4. Kinds

The tool captures an inheritance relationship between terms by allowing any term to refer to zero or more other terms as supertypes or subtypes. The model does not enforce a tree structure of inheritance. A modeler may introduce cycles in which a term directly or indirectly becomes a supertype and a subtype of itself.

We included kinds as a specialization-generalization mechanism for users familiar with object-oriented analysis. Kinds maps to the XML Schema as a union data type in which the subtypes are member types of the supertype.

2.5. Synonyms

Every term may join in zero or one synonym. A synonym groups some number of terms that are in some way equivalent. A modeler may use the “whom” annotation to annotate each term with a description of the context in which the term applies.

The Synonym model is the latest elaboration we have added in response to the requests of our users. In many organizations data formats have proliferated through growth or through merger such that one format gives one name to a data element and another format uses a different name. Our users wanted to capture that relationship among multiple terms, expose it, and document it.

2.6. Groups

The last of the information maintained in our model is the group hierarchy. The group hierarchy is like a directory of folders that may contain terms or other groups. The XML Schema output from the tool does not in any way show the use of groups. We have had some discussion about using groups to manage namespaces; but, have so far restrained from attaching any semantic meaning to groups. Groups remain a convenience mechanism only, for separating a large space of terms into smaller areas of concern. They are especially useful when the model contains a large number of terms.

A term may belong to any number of groups. A group may contain any number of terms or other groups. A group may belong to only one parent group, thus enforcing the tree structure of groups.

3. Drawing a Picture

All of the user interaction occurs through forms, buttons, dialogs, and menus. We chose this deliberately over diagrammatic manipulation as a mode of interaction most familiar to people who spend their time on a computer working with spreadsheet, word processing, and web applications. Figure 2, “Entering Terms” shows a screen shot of the interaction

used to capture terms as one example. The user enters one or more terms using the dialog, each of which appear in the tabular view of terms shown above the dialog.

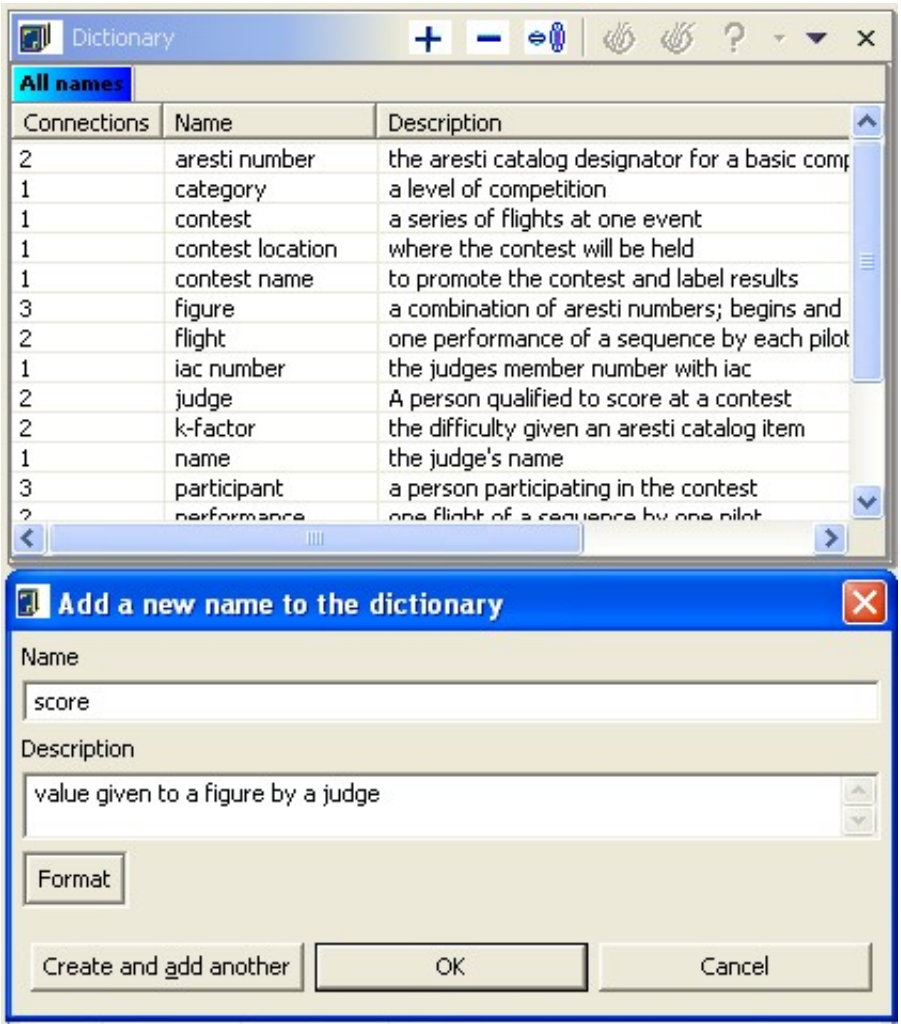

Figure 2. Entering Terms

The tool captures and displays connections and elaborations of the data such as composition, kinds, and relations with methods similar to that used for the terms.

Diagrams provide a great way to concisely visualize the network of relationships among terms. We therefore determined early-on to provide diagrams and dynamically update them as users make changes to their data models.

Figure 3, “Generated Diagram” shows an example diagram generated by the tool. It maps the connections to a currently selected term, “pilot.” At a glance we can see that the “pilot” is one of the elements contained within a “performance;” it is a kind of “participant;” and it has a relationship with “score.”

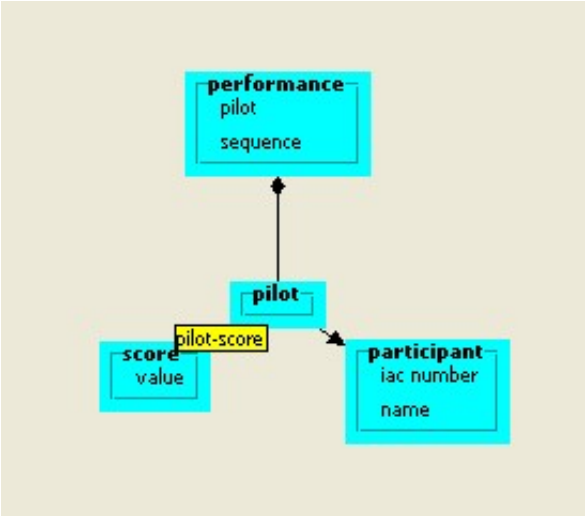

Figure 3. Generated Diagram

Our dynamic drawing display updates with the term currently selected for work rather than with a display all of the relationships with all of the terms. There are a number of reasons for this:

- It is clean. The term of concern is simply presented without irrelevant clutter.
- It's the easiest thing to do. It is hard to create an automatic layout of all of the terms and relationships in the model without overlaps and crossing lines.
- It fits within limited screen real-estate. Complete diagrams require scrolling about when viewed at a scale that makes text readable.
- It keeps focus where focus is wanted. The user sees only what is immediately relevant to the selected term.

We found that users appreciated having the diagrammatic form for their terms. Some users have requested an output that would provide the complete diagram as documentation of their work. Others have suggested a more interactive form of the diagram. Exactly how useful these diagrams are for understanding the model still requires some research [5], but our goals at this stage was to keep our diagrams syntactically simple so as to not distract the modeler as they build their dictionary of terms.

4. What we observed

By working with and observing people use our tool to solve a problem in their work, we have learned the following about how to design a successful and useful data modeling tool.

First, we take a flexible design approach that will accommodate the needs of our users as they become more proficient at using our tool. Initially we like to keep the interface simple, but it is important to recognize that soon more advanced features, that may add complexity to the screen, may be desired. For example, we debated about where to present the feature for adding format information to the terms. As designers we did not want the user to feel required to add in all details about data types just as they were entering their initial thoughts, but soon enough data types were desired by our user and as such we made the feature more readily available. Now the user may add format information while adding terms to the dictionary and while modifying terms in the dictionary, if they select the format feature button in the interface (shown in Figure 2, "Entering Terms"). This is just one of many examples where we let the user decide how they would like to advance and work with the data modeling tool. Incorporating the idea of offering various staging areas for reaching the same goal is a sound design methodology for building successful user interfaces [1]

As a model grows it is helpful to keep track on which terms were linked to other terms. First we had several columns with different icons in each column to represent whether or not each term had a relationship, or a hierarchy, or a composition created with it. When users saw this they first asked – what do these columns and icons mean? Clearly this was a distraction instead of an aid. So we removed the columns and put in one column labeled connections that displays a numeric value for the number of times each term is used in a connection. As such with a quick glance down this column one can check for any "orphans" to make sure all terms are placed appropriately within the model.

As our tool evolves, we learn just what features are useful and what additional features may be desired. Specifically, we found that the HTML output was used for presentation purposes, communication purposes, and for viewing the model in its entirety. How the XML Schema was reviewed also revealed aspects of model usage as the model grows. Users become curious about the Schema generator and as they become more familiar with Schema, they may wish to integrate clues into the model that would be used to generate better Schema. For instance, providing a method for differentiating between elements and attributes became necessary.

To address this need requires care. Do we change the user interface to enable users to specify whether a term should appear as an attribute or as an element; or, do we use heuristics within the Schema generator to infer whether terms are elements or attributes?

Questions like this are the fulcrum for the balance we are maintaining between the simplicity in the user interface and the functional completeness of the schema. It is encouraging to see how users who may have had minimal exposure to Schema develop a curiosity about it through the use of our tool. As such we like to support that curiosity.

One of the best testimonials we received about our tool was how easy it was to go back and make modifications. As the modeler delved deeper into the model and checked it against requirement documents he needed to adjust his initial thoughts about grouping terms into compositions. He realized that in order to properly represent a business rule more completely, he had to join two existing compositions by creating a third composition that included references of the other compositions. Our tool does not require the user make changes or additions graphically and as such our user noted: "I did not dread doing it as I would have in other modeling tools…"

A clear snapshot on how successful a tool may be is how willing a user is to go back and continue using the tool and how much they consider and look forward to building more models with the tool. When the user gets feedback about his model, he will be more willing to go back to the tool and change the model to reflect the updates. As such the model becomes an ongoing tool for tracking communication, as well as a Schema generator. Hence the impact of a successful modeling tool is felt beyond the keyboard of a single user. Even with a small sample size, this kind of feedback sets the direction of the development process for building tools that are truly useful and usable.

5. Where our data modeling tool may be applied

We attempt to improve the accuracy of the data gathering process by putting a tool in the hands of the domain expert, thereby capturing the kinds of knowledge based on both the user's experience with data and the facts associated with the data [6]. Our tool supports these concepts by offering a natural interface for composing the domain dictionary. Future research and observations will lead to determining the efficacy of the models and role they play in true domain understanding [7].

We have already observed how our tool can be used for consolidating several applications within an enterprise. We see that once a model is developed, the various outputs of the model become a vehicle for communicating key issues within the group whereby the same set of people who set out to establish a set of requirements can determine if all that was discussed was indeed understood. By documenting the data requirements with a model, the modeler, or designer, or IT architect has a chance to organize all the ideas into a meaningful way. He can then verify his process by sharing the HTML or Schema output from the model with his organization.

Other scenarios where we see this data modeling tool having a role include:

1. Capture knowledge from a subject matter expert during an interview process

2. Facilitate communications about a system when people are widely dispersed and have varied technical backgrounds.

3. To capture decisions made at data management workshops

4. As a guide during standards discussions

5. To share knowledge about a domain with others unfamiliar with it

6. As a shared development platform for investigating all nuances of the domain and data elements

7. To enable a non technical user to begin working with their data in a formal way, as they wish to start the ball rolling - while they wait for the IT architect or developer – so when they finally do meet their time will be better spent.

To further determine where the best match may be would require further field studies with users trying to solve real data problems in their organization [2].

6. Conclusions

Working with real users on a real task is a critical element for designing better data modeling tools. Following user centered design practices proved to be invaluable to shaping the data modeling tool into an effective communication tool as well.

We target this tool towards the initial phases of software development: interviews, requirements meetings, and recordings of the initial discussions where the accuracy of data and knowledge capture is critical. We hope the tool will continue to enhance and advance these early stages so that we can effectively measure the impact we might have on IT development costs.

Bibliography

[1] Spraragen, Susan, and Ribak, Amnon Flexible Software Interface Design, Ergonomics in Design, Vol.7, No. 4,4-8, October 1999.

[2] Greene, S.L. et al. Iterative development in the field, IBM Systems Journal, Vol. 42, No. 4, 594-612, (2003).

[3] XML Schema Part 2: Datatypes. W3C Recommendation 02 May 2001. http://www.w3.org/TR/xmlschema-2/

[4] XML Schema Part 1: Structures. W3C Recommendation 2 May 2001. http://www.w3.org/TR/xmlschema-1/

[5] Tilley, Scott and Huang ,Shihong A Qualitative Assessment of the Efficacy of UML Diagrams as a Form of Graphical Documentation in Aiding Program Understanding Proceedings of SIGDOC03, 184-191, October 12-15 2003.

[6] Robillard, Pierre N., The Role of Knowledge in Software Development, Communications of the ACM, Vol 42, No.1, 87-92, January 1999.

[7] Gemino, Andrew and Wand, Yair Evaluating Modeling Techniques Based on Models of Learning, Communications of the ACM, Vol.46, No. 10, 79 84 October 2003.

Biography

Susan L. **Spraragen**
IBM Research [http://www.ibm.com]
Hawthorne
New York
United States of America

Susan is a usability engineer at the IBM TJ Watson Research Center. Susan is a driver for usability and human factors applied to engineering of user systems and interfaces for better user experiences and marketability.

Douglas **Lovell**
IBM Research [http://www.ibm.com]
Hawthorne
New York
United States of America

Douglas is a software engineer with IBM Research. He has worked on W3C XML recommendations, including the XSL Formatting Objects for which he developed an implementation, authored a book, and provided a tutorial at the 2003 conference of the W3C; and MathML for which he served as an alternate member of the working group and provided tutorials.