Model Mapping in XML-Oriented Environments

Erik Wilde
ETH Zürich (Swiss Federal Institute of Technology)

July 2006 (TIK Report 257)

Available at http://dret.net/netdret/publications#wil06m

Abstract

XML and Service-Oriented Architectures (SOA) make it easier to develop loosely coupled systems, but they do not eliminate the fundamental necessity of communications that there must be an underlying shared model. Because of the popularity of SOA, it becomes increasingly important to be able to quickly and efficiently integrate information systems, rather than using an expensive top-down process. The XML landscape evolved bottom-up, and so far it has not yet reached a stage where XML is explicitly targeted in conceptual models. Filling this gap with guidelines and best practices thus is the most pragmatic approach. The approach presented in this paper is a structured view accompanied by guidelines for how interoperability can be achieved on the model level today.

Contents

1 Introduction

While XML has been very successful as a data format for use in information reuse and integration, it has become clear that a syntax alone is not sufficient for fully supporting these activities. For very simple scenarios, it may be sufficient to rely on an XML schema for specifying the data to be exchanged, but in more complex scenarios, it is crucial to properly understand the data. This involves two major relationships, the relationship of schema components to the model underlying the data exchange, and the relationship of this model to the model underlying the own system.

The basic problem is that of a model mismatch between the system model and (often implicit) exchange models. Attempts to address this problem can be made in different ways. *Model-Driven Engineering (MDE)* [2] aims at changing the whole focus of software engineering to a model-centered view. It is mainly a top-down approach and has a promising future for replacing the OO principle as the next paradigm for building software systems. In this paper, we describe a bottom-up approach for handling the model mismatch, based on the assumption that for already developed and deployed systems, the MDE approach is not applicable.

The model mapping approach described here has close links to the field of *Document Engineering* [4], where interaction between loosely coupled systems is described as exchanging documents. The challenge of this approach is how these documents and their relationships to system models can be described as accurately as possible. Basically, Document Engineering can be regarded as a special case of MDE, where the goal is to work with models which can be broken down into documents for implementing loosely coupled systems.

Even though XML in the past years has established itself as the most successful format for data exchange, there currently is no established way how to define models on a more abstract level than XML schema languages [8]. Therefore, the approach presented here cannot simply take an existing conceptual model for XML and use this as the foundation. Instead, the lack of modeling support for XML must be alleviated by using a disciplined approach of how to use XML schemas.

2 Model Types

Models in software engineering represent those aspects of reality which are considered relevant for the task at hand. In this very broad sense, models can represent any aspect of reality, but for the sake of simplicity and generalization, models are often designed in ways which classify certain facets of reality, so that they their representations can be grouped and handled uniformly. In the case of XML-oriented information exchange, there are two main aspects which are often captured in models.

These aspects are not fully disjoint or mutually exclusive, but for a closer look at models such as in Section 3, it is useful to have a broad classification of model types. When considering the popular modeling language UML [7], its structure diagrams are considered *inventory models*, and behavior and interaction diagrams are considered *event models*.

2.1 Inventory Models

An inventory model represents (mostly) static facets of reality which can be observed as objects, states, facts and similar things. While these static facets may change over time, they have a certain lifespan, and the model aims at capturing these facets during this lifespan.

Inventory models represent a snapshot of the realm they are concerned with, and this snapshot can be used to retrieve valuable information about current state of reality.

Inventory models do not capture how the transition is made between different states of reality, they only have the ability to also represent the new state after the transition took place. Consistency constraints may place limitations on which states are acceptable and which are not, but again this is only concerned with a static state, not with the transition triggering state change.

2.2 Event Models

An event model concentrates on events which change the state of an inventory model. As such it has to refer to certain aspects of the inventory model (most importantly to the facets affected by the state change), but the main focus is the event triggering the change. This event may also have facets which are not relevant for the inventory model, but which are nonetheless relevant for the event itself.

3 Models

When interpreted in its most general meaning, there is a model behind every artifact that somehow relates to reality. In the context of this paper, there are models being used by peers in B2B scenarios, and models underlying the data formats which are used for data exchange. These models typically represent similar facets of reality, but they may do so in slightly different ways, and they may differ in coverage. Since successful information integration critically depends on the ability to correctly understand data being processed, the models involved in B2B scenarios must be investigated carefully.

3.1 System Model

System models are the models used for planning and building IT systems. These models may be *prescriptive* or *descriptive*, meaning either created in advance and used for building a system, or created as a description of an already existing system. In either case, the model is an abstraction of the IT systems and represents the facets of reality which are represented with the IT system.

System models tend to be more inventory models (Section 2.1) than event models (Section 2.2). The reason for this is that IT systems usually try to represent states of reality as completely as required for the given application area, and therefore the representation of reality must include all relevant aspects that constitute these states. While state changes of course also are relevant and often are captured in the system model as well, there are also many cases where the system model ignores state changes, and the transition from one well-defined state to another takes place outside of the scope of the model.

3.2 Exchange Model

In the context of B2B, the most likely scenario is that the basic model unit is a workflow. A workflow in its most general sense is a composition of communicating peers, which interact by exchanging messages to accomplish a shared goal. While these messages in most cases have

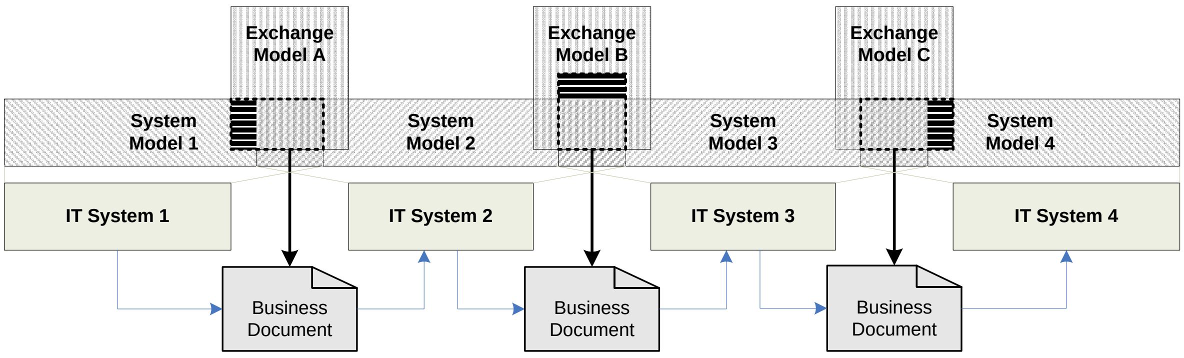

Figure 1: Model Chain of Typical B2B Scenario

references to some aspects of inventory models, they tend to have their focus on the process-relevant facets of the activity taking place. These facets on the one hand decide what the effect of the workflow on the affected inventory will be, and on the other hand may contain aspects which are irrelevant for the inventory as such, for example logging information or security-related information.

Therefore, exchange models tend to be more event models (Section 2.2) than inventory models (Section 2.1). As with inventory models, there will be hardly a case where an exchange model contains only event information and does not relate to any inventory at all. On the other hand, the continually growing set of WS-* standards can be seen as examples which often does not have anything to do with the system models of the communicating peers, but are essential for controlling the information exchange.

Exchange models often are not properly specified, and in many cases only exist in the minds of the designers of exchange mechanisms. Observing this, the *Universal Business Language (UBL)* [6] has been one approach to provide a common model for exchanging business information, defining some basic inventory facets (such as addresses or items) as well as some basic workflow patterns (such as purchase orders or invoices). UBL thus provides a common foundation from which exchange models can be constructed.

4 Model Chain

When taking the concepts presented in Sections 2 and 3, a complete B2B scenario can be regarded as a complex flow of information through a chain governed by different models. In most cases, there will be no “supermodel” spanning the full scenario (and thereby defining the common model of reality all participants of the transaction will be working with). With the advent of *Service-Oriented Architectures (SOA)* and their ability to construct loosely coupled systems, it becomes increasingly less likely that there is one unifying model, instead business transactions can be created spontaneously and without the usual modeling and planning and deployment stages that were the defining feature of earlier, more closely coupled architectures. Figure 1 shows the basic setup of this scenario, with a sequence of four systems being involved in a B2B workflow involving three exchanges.

The most important observation is that the system models of two systems engaged in an exchange must have an overlap of their models, otherwise they will not be able to communicate. This overlap may be part of the system models themselves, or it may be established by defining how certain facets of the peer model are mapped to the own model. At least part of this overlap must be contained in the exchange model (where it is represented in some form of which the two peers have the same understanding).

Apart from this overlap in the systems models contained in the exchange model, the exchange model may also contain facets only contained in one of the system models, or not contained in the system models of both peers. These parts of the system model are useless for the mutual understanding of both peers. They may, however, still be useful in the larger context of the workflow, where information carried along which can only be interpreted by a subset of the involved systems, and is transparently forwarded by the others. This situation is shown with the darker area of the exchange models, which is only contained in system models 1 and 4, but nevertheless forwarded by systems 2 and 3.

For B2B transactions it is necessary to have representations of the exchange models, so that the information can be encoded in business documents and forwarded along the workflow path. In the scenario discussed here, the B2B transactions are XML-oriented, and the usual practice for describing classes of XML documents is to use schemas.

5 Schemas

While every exchange has an underlying exchange model (as described in Section 3.2), in the majority of cases this model is not made explicit. Instead, in most cases the model is implicitly assumed, and the only thing made explicit is the structure of the information which may be exchanged. In case of XML, this is done with XML schemas, and depending on the design of the schema, it is more or less difficult to deduce the underlying model from the schema.

Most schema development for XML today is done with XML Schema [9]. This is a big improvement over XML DTDs, because XML Schema has an application-oriented set of built-in simple types, and because it better supports modeling through concepts such as type derivation and reusable groups. It is important to notice, however, that XML Schema is not a modeling language, it is primarily a language for defining XML vocabularies. The modeling aspects that have been included in XML Schema can be used to express some aspects of a conceptual model, but they still ar geared very much towards modeling parts of documents.

In addition, it has been shown that most schemas on the Web are structurally very simple [1], with many of them being simply generated from DTDs, and the vast majority not using any of XML Schema's more advanced features. This means that even though it would be possible to represent the exchange model at least partially in XML Schema, this is rarely done and in practice schemas are not much more than vocabularies for structured documents.

Even if schemas are carefully designed, using as much of XML Schema's modeling features as possible, they will always be insufficient for describing the exchange model for the following reasons:

1. Schemas only represent a part of the exchange model, representing the part which should be exchanged for a particular transaction. Designing a core schema and reusing components from it for designing the actual transactions schemas helps to make the shared concepts more obvious.

2. XML Schema's features do not support all aspects which are relevant for models, and some rather basic concepts such as the `all` group are severely limited.

3. Designing a schema introduces artifacts which are not part of the underlying model, but are required for markup design, such as the distinction between elements and attributes, or the question whether a hierarchic relationship is part of the model, or just XML structure. XML Schema has no way of distinguishing between "model structures" and "markup structures".

4. Naturally, schemas are focused on XML, and a conceptual model (whose primary focus should be an implementation-independent representation) should not be focused on one particular technology. This may change, however, if XML continues to penetrate applications and becomes even more dominant as the format for structured data.

This means that schemas are not the best way to describe an exchange model. In database terminology, a schema is on the level of a logical model, while the models that are described here are on the level of conceptual models. When trying to map logical models (such as database schemas to XML schemas), a number of problems occur because the models are already too implementation-specific [10].

It is thus advisable to treat XML Schemas not as conceptual models per se, but as useful logical models. Using terms from MDE, an XML Schema is a *Platform-Specific Model (PSM)*, while the exchange model itself should be a *Platform-Independent Model (PIM)*. So while in theory a schema should be derived from an exchange model, it often is created without such a foundation. This makes it harder to perform some of the typical configurations of exchange models, which are described in the following sections.

5.1 Model Parts

Often, schemas represent only parts of exchange models, because different interactions require different subsets of the model, and each of these interactions is defined by a schema. The best way to support this kind of dependence between schemas is to start with a core schema which defines the basic building blocks of the model, and to reuse these schema components in the individual interaction schemas. This may still fail to capture some of the exchange model aspects which cannot be represented in the schema, but the usage of a common foundation makes it easier to understand the shared model parts.

The UBL approach mentioned above is one such core schema, which can be used as the foundation for a more application-specific core schema. By reusing well-known models, the overall costs for understanding an exchange model can be reduced.

5.2 Model Stages

Another typical configuration are different processing stages of workflows, where the same aspects of reality should be handled with different constraints. For example, when processing input data, usually it has to be validated according to application-specific criteria, and there should be schemas describing the more lenient structure of the input data itself, and for the stricter structure of the validated input data. This pattern of increasingly strict constraints can occur in multiple stages.

6 Model Mapping

To achieve interoperability, the models involved in a workflow must be mapped, so that semantic interoperability can be achieved. For this task there are two "good" solutions, which unfortunately both are not feasible in many scenarios. The best solution would be to have a "super-model" spanning all involved models, and this model could then be used for defining the mapping. This is the model-driven approach advertised by MDE. The downside is that this approach only works in an environment where all peers agree to use it, which somehow contradicts the loose coupling of SOA. A more modest approach in this area would be to see all peers agreeing on "micro-models" (often called "micro-schema" when being used with XML), so that at least parts of the overall model can be represented using common models.

In the case of using schemas rather than models, it is advisable to augment the schemas with information which they cannot represent themselves, but which is considered relevant from the model point of view. Most schema languages today provide mechanisms for schema annotations, so that a schema can be improved. One possibility is to add references to a data dictionary, where the components of the schema are semantically described. Another possibility is to add structural information to the XML Schema, for example

- whether `maxOccurs` > 1 should be treated as set or sequence,
- additional constraints using *Schematron* [5],
- marking hierarchical relationships as model-relevant or pure XML markup design, or
- any "identity constraints" involving information outside the scope of a single document.

Carefully designed and annotated schemas can provide substantial improvements over tool-generated and undocumented schemas, and given the fact that schemas typically have a rather long life span, is well worth the effort. Schema design guidelines have been appearing in many XML-oriented communities mostly for this reason (the other main reason being that XML Schema is hard to master).

Using this pragmatic approach of the "schema as the model", it becomes much easier to manage the cases described in Sections 5.1 and 5.2. Given sufficient annotations, specialized schemas can then be derived automatically from the "super schema", for example by using transformations. However, this strategy in most cases only applies to one model (typically exchange models, which are often based on XML Schemas), so when working with an exchange model defined in this way, the question is how to align the system model.

6.1 Model to Schema Mapping

For mapping the information from a system model to an exchange schema, it is necessary to identify the overlap in the two models. Figure 2 shows this situation, where the overlap between the two models is identified, and the system can then encode this information as defined by the schema.

From the information integration point of view, the identification of the overlap is the critical part. From the software engineering point of view, it is also relevant to think about how the actual data binding takes place. Depending on the system platform, various data binding are available, and because of XML's popularity, this area is very active and new

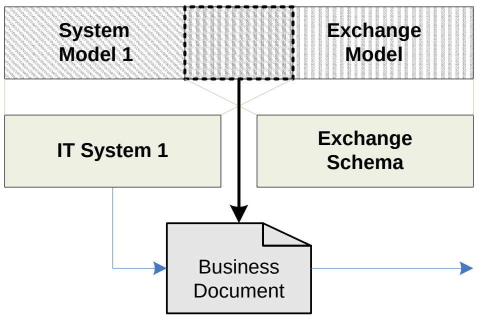

Figure 2: Model to Schema Mapping

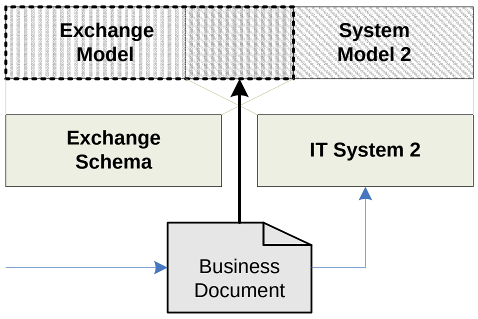

Figure 3: Schema to Model Mapping

technologies appear regularly. Data binding can be achieved with databases, programming languages, or simply other XML schemas, and for all these cases, solutions are available.

While data binding is addressed for many different scenarios, the ability of most technologies to deal with changes in the schema is very limited. Schema extensibility and versioning are two issues which are not well studied. XML schema has weak built-in mechanisms for supporting extensibility, and no built-in mechanisms for supporting versioning, so designing schemas in a future-proof way today is left to schema developers.

6.2 Schema to Model Mapping

For incoming information, the schema of the respective exchange model must be mapped to the system model. This situation is depicted in Figure 3, which also shows the slight difference to the model to schema mapping described in the previous section. Schema to model mapping must take into account the fact that the received documents may use the full exchange model (unless some form of content negotiation is available).

This raises the question how to proceed with information which is received, but cannot be mapped onto the system model. Extensibility and versioning as discussed in the previous section are one possible source for this, another one is an exchange model which contains a lot of information which is irrelevant for the system. A common practice in schemas is to mark information with a special `mustUnderstand` flag if it must be interpreted by the receiver. This makes it possible to enforce the interpretation of some schema parts. Thus, when a system received unmapped information, it may

- discard the unmapped information,
- throw an exception that has to be handled on the application level (in particular if the unmapped information is marked as `mustUnderstand`), or
- transparently forward the unmapped information to the next peer (as shown in Figure 1).

Data binding applies to schema to model mapping in the same way as it does to model to schema mapping. Depending on the system platform, some data binding technology must be chosen. Some data binding technologies are read-only (for example XQuery [3], which currently does not support updates), in this case a different technology must be chosen for schema to model mappings. For extensibility and versioning robustness, the data binding technology should support these issues, otherwise the loose coupling vision of SOA can be compromised by the need to update a system if the exchange schema changes.

7 Future Work

The approach presented in this paper can be used for model mapping in XML-oriented environments and is based on currently available technologies. It is inspired by ideas which are central to MDE, and is focused on the loose coupling scenario of SOA.

Document engineering is a central part of this approach, the idea to carefully design the business documents that are being exchanged in SOA environments. Loose coupling is the current trend in building distributed systems, and will probably remain the guiding principle for some time to come. The more traditional ways of building distributed systems are being replaced by document-oriented loose coupling, the more important it will become to be able to develop and describe the underlying models.

Data modeling for XML-oriented architectures still its in its beginnings and a very promising research area. In the long run, the pragmatic approach presented here will be replaced by a conceptual modeling language for XML [8], supporting schema derivation, model evolution, the derivation of versioned schemas, and annotations which can completely control how the model is mapped onto XML structures.

8 Conclusions

While building loosely coupled systems based on XML technologies has become easier than with any other family of technologies. For scenarios where the complexity goes beyond trivial micro-schemas, however, the task of mapping the models of two different applications still involves considerable work. This task of aligning different model worlds, however, will never disappear, regardless of the technology. In XML-oriented scenarios, it is possible to make this task a bit easier by using a disciplined approach to defining and using schemas. By structuring, annotating, deriving, and transforming schemas, the vision of MDE can be implemented today, not quite on the ideal model level envisioned my MDE, but at least on a level which already provides some level of abstraction.

References

[1] Geert Jan Bex, Wim Martens, Frank Neven, and Thomas Schwentick. Expressiveness of XSDs: From Practice to Theory, There and Back Again. In *Proceedings of the 14th International World Wide Web Conference*, pages 712–721, Chiba, Japan, May 2005. ACM Press.

[2] Jean Bézivin. On the Unification Power of Models. *Software and Systems Modeling*, 4(2):171–188, May 2005.

[3] Scott Boag, Donald D. Chamberlin, Mary F. Fernández, Daniela Florescu, Jonathan Robie, and Jérôme Siméon. XQuery 1.0: An XML Query Language. World Wide Web Consortium, Candidate Recommendation CR-xquery-20060608, June 2006.

[4] Robert J. Glushko and Tim McGrath. *Document Engineering.* The MIT Press, Cambridge, Massachusetts, August 2005.

[5] International Organization for Standardization. Information Technology — Document Schema Definition Languages (DSDL) — Part 3: Rule-based Validation — Schematron. ISO/IEC 19757-3, April 2006.

[6] Bill Meadows and Lisa Seaburg. Universal Business Language 1.0. Organization for the Advancement of Structured Information Standards (OASIS), Committee Draft, September 2004.

[7] Object Management Group, Framingham, Massachusetts. *UML 2.0 Superstructure Specification*, October 2004.

[8] Arijit Sengupta and Erik Wilde. The Case for Conceptual Modeling for XML. Technical Report TIK Report No. 244, Computer Engineering and Networks Laboratory, ETH Zürich, Zürich, Switzerland, February 2006.

[9] Henry S. Thompson, David Beech, Murray Maloney, and Noah Mendelsohn. XML Schema Part 1: Structures Second Edition. World Wide Web Consortium, Recommendation REC-xmlschema-1-20041028, October 2004.

[10] Erik Wilde. Tables and Trees Don't Mix (very well). In *Poster Proceedings of the 15th International World Wide Web Conference*, Edinburgh, UK, May 2006. ACM Press.