Representing and Transforming Model-Based Information*

Shawn Bowers and Lois Delcambre
{shawn, lmd}@cse.ogi.edu
Computer Science and Engineering Department
Oregon Graduate Institute
20000 NW Walker Road, Beaverton, OR 97006

Abstract

There are a variety of ways to represent information and each representation scheme typically has associated tools to manipulate information so long as it is represented properly. We are particularly interested in representations for superimposed information, where supplementary information can be used to highlight, annotate, elaborate, and interconnect underlying information. The Resource Description Framework (RDF) and the Topic Map model are two such examples that can represent superimposed information.

We focus here on model-based information where the information representation scheme provides structural modeling constructs (analogous to a data model in a database). For example, the XML model includes elements, attributes, and permits elements to be nested. Similarly, RDF models information through resources and properties. We also consider other models such as a spreadsheet (with a model that includes cells organized into rows and columns) and a relational database (with a model that includes tables with attributes).

The goal of our work is to enable the user to apply tools of interest to the information at hand. Our approach is to represent information for a wide variety of model-based applications in a uniform way, using RDF, and to provide a mapping formalism that can easily transform information from one representation to another.

1 Introduction

In this research, we recognize that many distinct yet highly useful representation schemes have been proposed. Each representation scheme, such as the Extensible Markup Language (XML) [9], RDF [15] and Topic Maps [8], has various tools associated with it. Rather than promote the use of a single representation for information (to the exclusion of others), we propose to easily convert information from one representation scheme to another when needed. We enable useful tools to be exploited against existing information simply by converting the information from its original form to the form required by the tool of interest.

We focus in this work on information representation schemes that are model-based. The XML model includes elements with optional attributes and a relational database model represents information in tables, for example. A common feature of RDF, Topic Maps, and XML is their use of an optional schema to type information. For example, a particular RDF representation may conform to an RDF Schema [11], a topic map may conform to a topic map definition, and an XML document may be valid for a document type definition (DTD).

We present here a generic representation scheme for model-based information that allows the model, the schema, and the instance information to be represented explicitly. Our approach is to use a metamodel with an associated, underlying representation expressed in RDF and RDF Schema. This uniform, generic representation for information enables simple transformation from one form to another using our mapping rules.

This work is motivated by our general investigation into what we call superimposed information. We briefly present superimposed information in Section 2 and explain how it relates to this work. Section 3 describes the metamodel that provides a formalism to define various information models. Section 4 describes the associated representation for information, based on our metamodel using a flat representation scheme. In Section 5, we show how logical rules over the representation scheme can formally specify and implement superimposed-layer mappings from one representation to another. We discuss related work in Section 6 and present conclusions and future work in Section 7.

* This research is supported in part by the National Science Foundation, through grants IIS-98-17492, CDA-97-03218, and EIA-99-83518, and by the Defense Advanced Research Projects Agency, grant N66001-00-C-8032.

2 Superimposed Information: Motivation for this Work

Suppose you are planning a vacation using the Web. Imagine that you are able to drag and drop selected information from Web sites into a scratchpad tool that allows you to group information and attach annotations such as "Hotels in Vancouver," "Restaurants in Victoria," "Day trips from Nanaimo," etc. The scratchpad keeps links to the original information, allowing you to navigate back to it whenever you need to (e.g., you may have selected a regular rate, but wonder if the hotel has a weekend special).

To help you manage the cost of the trip, including exchange rates, lodging, and transportation expenses, you use a spreadsheet application, which automatically lifts the appropriate data from the scratchpad. In developing an itinerary, you select information from the scratchpad and place it into a calendar application, which also allows you to browse back over the original documents. You also decide to lift all of the address and phone numbers relevant to the trip into your address book (e.g., to load into your Palm Pilot) and integrate the payment information you've collected into your checkbook software to develop a budget for your vacation.

Each tool in this example leverages superimposed information [13, 16] to help manage and organize data. Superimposed information is a layer (the superimposed layer) of data placed over existing information sources (the base layer) to select, combine, highlight, supplement, and provide additional links among selected information elements within the underlying sources.

The basic arrangement of the superimposed and base layers is shown in Figure 1. The base layer consists of information sources that are referenced by marks in the superimposed layer. Marks reference information at various granularities within a source, as desired by the user and as permitted by the addressing scheme. Information-sources can be of many different types including HTML pages, XML documents, PDF files, Microsoft PowerPoint presentations, Excel spreadsheets, databases and so forth.

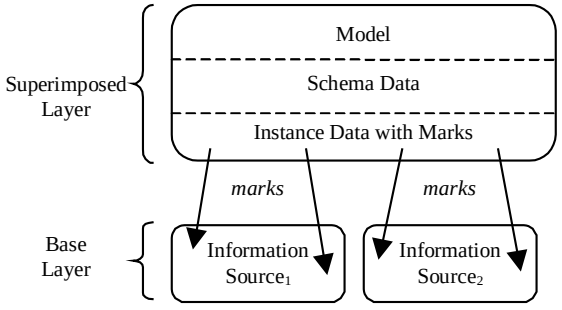

Figure 1. The superimposed and base layers.

Superimposed information has the following characteristics:

- It can contain additional information about elements in the base layer (e.g., by organizing, annotating, or highlighting elements).
- It can have varying degrees of structure.
- It does not modify the base layer.
- It can contain marks that connect the superimposed layer to elements within the existing information sources.

3 The Metamodel for Representing Model-Based Information

Information of interest in this research consists of three levels: model, schema, and instances as shown in Figure 2. All three layers are optional. Instance data may not have a schema and it may or may not have a model, although we focus in this research on information representation schemes that exploit a model. It is also possible to describe a model and a schema, without having instance data present.

To describe multiple information models, we define a level of abstraction above the model, called a metamodel, which is used to define the model of interest to the superimposed application. Figure 3 shows the role of the metamodel for three information representations: RDF, Topic Maps, and XML.

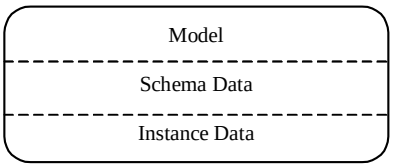

Figure 2. The three layers of information in a model-based information representation scheme.

The metamodel describes the basic abstractions used to define model constructs and their relationships. A model consists of schema and instance constructs that are used to define data. Additionally, the model describes the conformance relationship between instance-level data and schema-level data. Each level of the architecture can be viewed as an instantiation of the levels above it. More specifically, model constructs (i.e., schema and instance constructs) are particular instantiations of the abstractions defined by the metamodel, schema-level data are particular instantiations of the model's schema constructs, and instance-level data are instantiations of the model's instance constructs and can conform to the schema-level data.

In order to understand the three levels, Figure 4 shows an example of model, schema, and instance data for

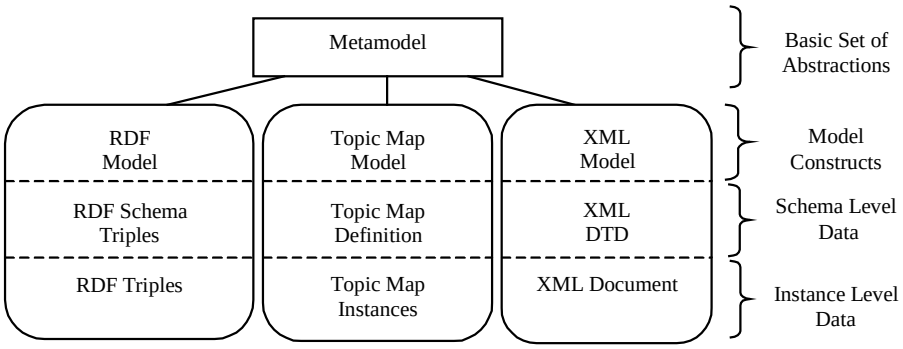

Figure 3. The RDF, Topic Map, and XML models within a superimposed layer.

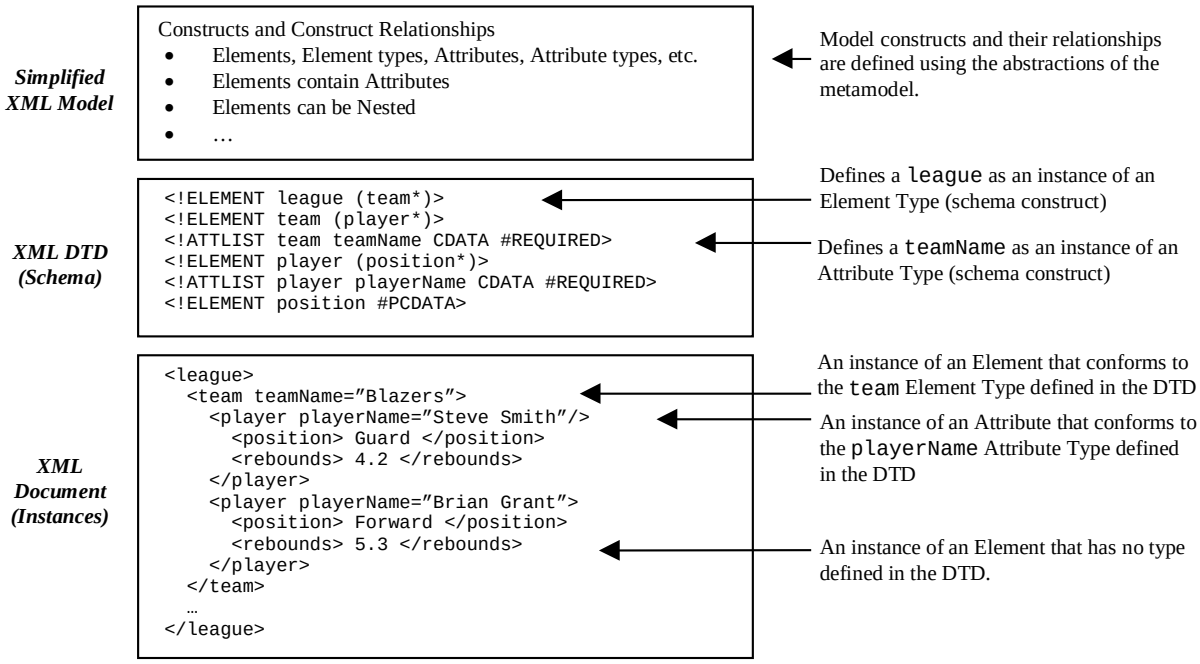

Figure 4. An example of each of the three levels (model, schema, and instance) for XML.

XML. Notice that we use an "open" DTD, which means that elements and attributes not defined in the DTD can be included in XML documents.

The definition of our metamodel is similar to other metamodel approaches in the object [18, 19] and database communities [1-5, 17] for describing structural models such as the entity-relationship model, the relational model, the hierarchical model, and the various semantic data models including the UML.

However, we support a number of unique features with our metamodel. One such feature is the inclusion of marks, which are references from the superimposed layer to elements within the base layer and may appear at various places within a superimposed model [13]. Another distinguishing characteristic of our metamodel is the relaxation of schema-first definitions that require schema to be created prior to instances. For example, in a relational database, a table (which represents schema information) must be created before any of its rows. Not only do we relax schema-first definitions, we also allow for data that is not explicitly typed. For example in a topic map, a topic can exist without being associated to any type even if there is a topic map definition.

The final unique characteristic of our metamodel representation is that it accommodates multiple levels of schema-instance relationships. In a topic map, topics can have a type that is also a topic, and so it too can have a type, resulting in two levels of schema and instance definition.

Figure 5 shows the abstractions of the metamodel. The two basic elements are the *construct*, which represents a basic structural definition within a model, and

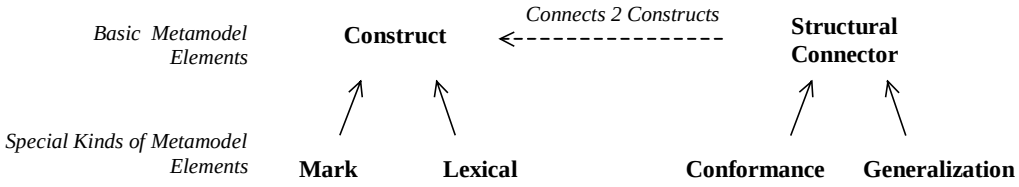

Figure 5. The elements defined by the superimposed-information metamodel.

structural connector, which represents a relationship between constructs.

There are two special constructs: mark and lexical. A *mark* describes a model construct whose instances represent connection-points to the base layer. A *lexical* describes a model construct whose instances contain primitive-value types (e.g., string). There are also two special structural connectors: *conformance*, which specifies a schema-instance relationship, and *generalization*, which specifies inheritance between constructs.

Table 1 shows an example of the XML model defined in terms of our metamodel. The XML model has been simplified to consist of Element Types, Elements, Attribute Types, Attributes, Primitive Content Type (e.g., PCDATA), and Primitive Content along with a minimal set of relationships between them.

Element constructs in XML form a hierarchy and are represented by the model connector Nested Element. By using multiplicity constraints, we can specify that an Element is either not nested or nested within one parent Element, and can have many Elements nested within it.

We use conformance connectors to specify schema-instance relationships between Attribute and Attribute Type, Element and Element Type, and Primitive Content and Primitive Content Type. We may also wish to apply constraints to the relationship. For example, by assigning the appropriate multiplicity constraints, we can specify whether an instance construct can be created prior to a schema construct.

The metamodel does not restrict model constructs to be at the instance- or schema-level. This allows models to have multiple levels of schema and instance definition. For example, in the Topic Map model we could define a Topic construct that has a conformance connector to itself.

Finally, we allow an Element Type construct to contain a Primitive Content Type construct. We define Primitive Content Type as a lexical construct, which means instances of the Primitive Content Type can be primitive types such as string, integer, or more specialized

Table 1. The XML model described in terms of the superimposed-information metamodel. The elements of the XML model (bottom) are instances of the corresponding Metamodel element definitions (top).

Metamodel Elements	**Constructs**	**Lexicals**	**Connectors**	**Conformance Connectors**
XML Model	Element Type Attribute Type Element Attribute Primitive Content Type Primitive Content	Primitive Content	Nested Element Type *Connects Two Element Types* Nested Element *Connects Two Elements* Element Content *Connects an Element to Primitive Content* Element Content Type *Connects an Element Type to Prim. Content Type* Element Attribute *Connects an Element to an Attribute* Attribute Element Type *Connects an Element Type to an Attribute Type*	Element Instance Of *Connects an Element to its Element Type* Attribute Instance Of *Connects an Attribute to its Attribute Type* Content Instance Of *Connects Prim. Content to its Primitive Content Type*

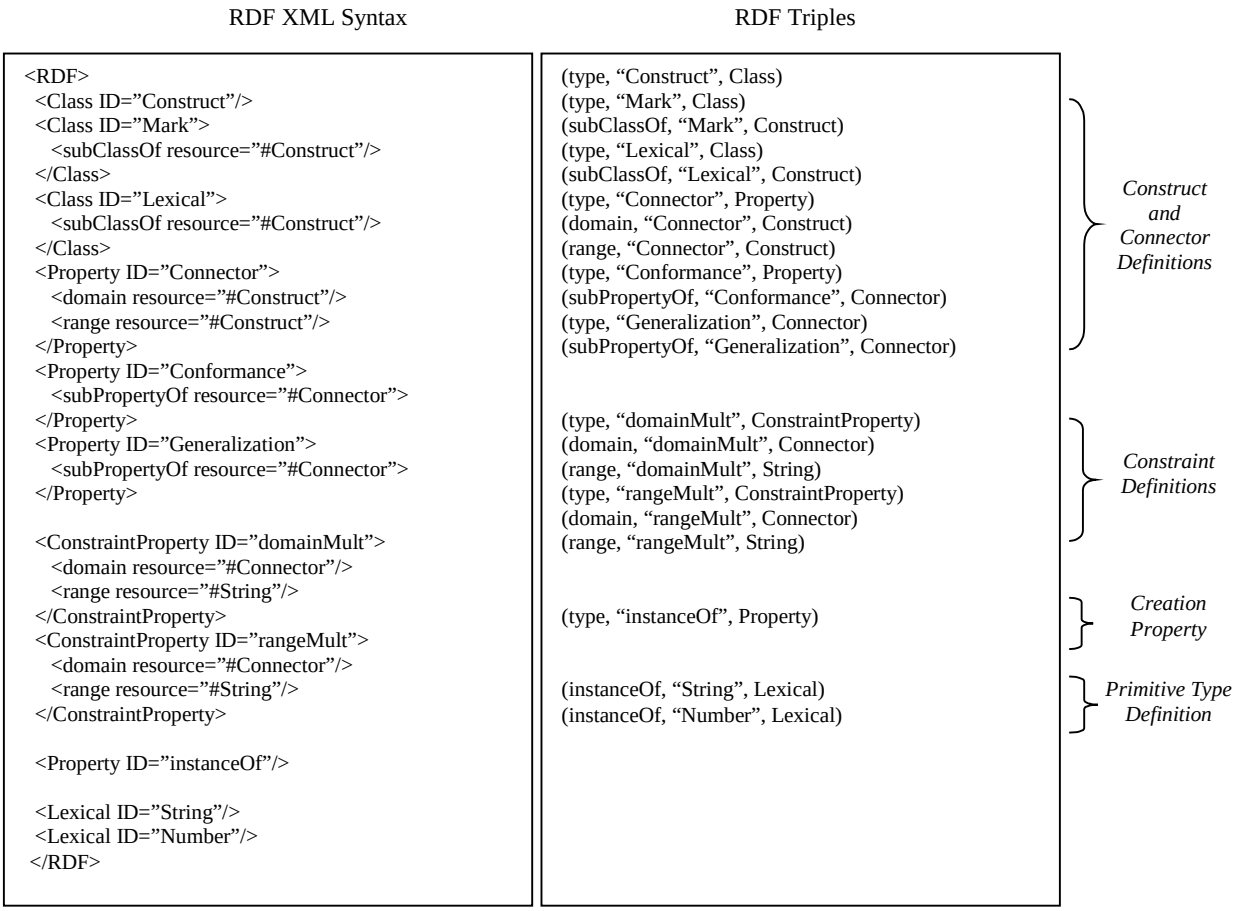

Figure 6. The superimposed-information metamodel represented in RDF XML and as RDF Triples.

types such as PCDATA for XML. Elements that conform to the Element Type must then contain Primitive Content with the type specified by the Primitive Content Type.

4 Representing Superimposed Models, Schemas, and Instances

Models defined by the metamodel are stored using a representation scheme based on RDF. Although model engineers can specify models directly using the RDF representation, we believe it is more convenient to define models visually. Therefore, we also define a visual representation of models using a subset of the UML.

4.1 The Resource Description Framework

RDF is a graph-based model for attaching metadata to information sources on the web (and can be itself considered a superimposed information model). It consists of a set of statements that are represented as triples. A triple denotes an edge between two nodes and has a property name (an edge), a resource (a node), and a value (a node). A value can be either a resource or a literal. Resources can represent anything from web pages to abstract concepts. A literal is a primitive type such as an integer or string. For example, the RDF triple (creator, "index.html", "Ora Lassilla") can be read as "the creator of index.html is Ora Lassilla" where "creator" is a property name, "index.html" a resource, and "Ora Lassilla" a string [15].

RDF Schema is a type system for RDF. It provides a mechanism to define classes of resources and property types, which restrict the domain and range of a property. The resource *Class* is used to type resources and the resource *Property* is used to type properties. Each Property consists of a *domain* and *range* constraint. In addition, RDF Schema defines the property *subClassOf* to represent a subset-superset relationship between classes, *subPropertyOf* for a specialization relationship between properties, and *type* to specify resource creation. The RDF and RDF Schema specifications use XML as an interchange format to exchange RDF and RDF Schema triples.

4.2 The Metamodel Defined using RDF

Figure 6 shows the definition of the metamodel using both the RDF XML syntax and RDF triples (for readability, the namespaces *rdf* and *rdfs* are not included). We represent

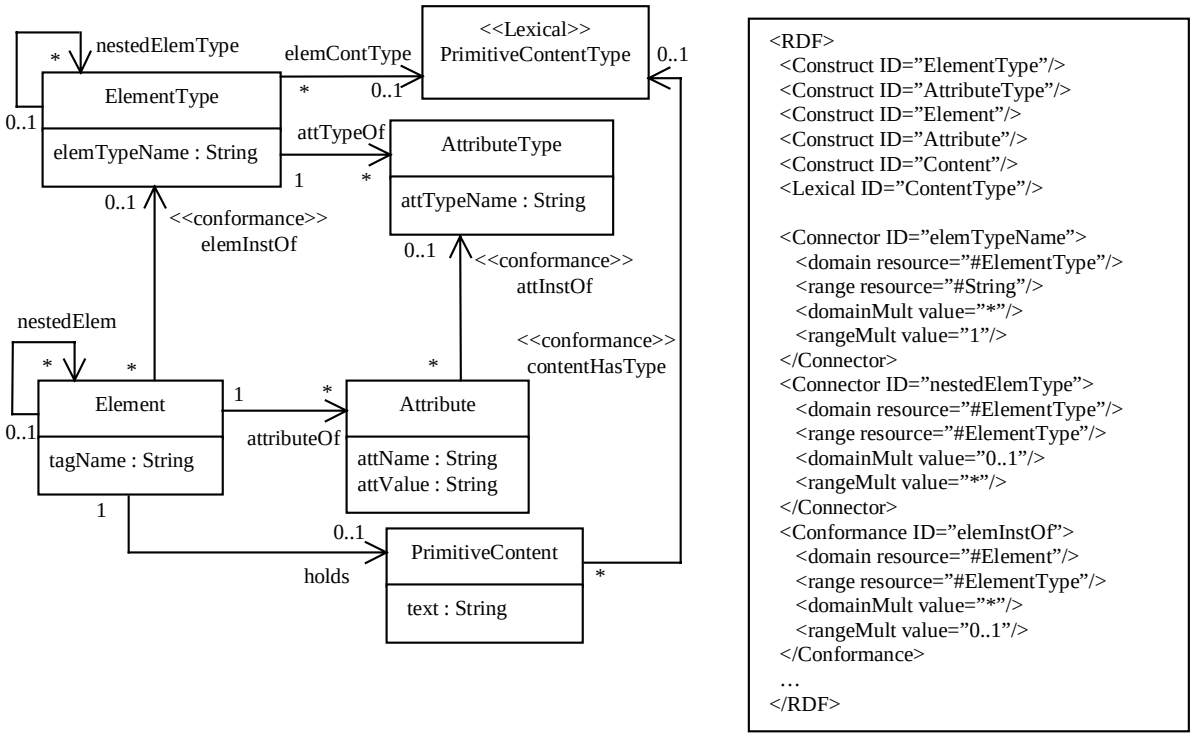

Figure 7. The XML model represented using UML with a sample of the RDF representation.

construct, mark, and lexical as RDF classes, where mark and lexical are sub-classes of construct. Similarly, we represent connector, generalization, and conformance as properties, each with a construct as domain and range. Generalization and conformance are both sub-properties of connector.

Domain and range multiplicity constraints are defined as RDF *Constraint Properties*. A Constraint Property is a higher-order property that can be used to add constraints (beyond domain and range) to a Property. To create model constructs and connectors as well as schema and instance data, we define a property called *instanceOf*. Notice that the domain and range of instanceOf is not defined, which means that it can be used over any resource and contain any value. We also create two lexical constructs: string and number. If desired, a model engineer can specify a new primitive type (e.g., PCDATA) using a similar definition, providing new primitive types are primitive (i.e., data of the type must have some form of string representation like a float, integer, or date).

4.3 The RDF and Visual Model Representation

Figure 7 depicts both the RDF and visual representations of the XML model of Section 3 using the metamodel. In the visual description, UML classes are mapped to constructs; relationships and class attributes are mapped to connectors. Attributes must have a primitive type as a range (e.g., string or number) and implicitly have a domain multiplicity of zero-or-one and a range multiplicity of one. UML stereotypes are used to distinguish marks and lexicals from constructs, and conformance connectors from regular connectors. Additionally, UML generalization relationships require a name (which is not a general requirement of UML).

The schema-level constructs of Figure 7 are Element Type, Attribute Type, and Primitive Content Type. The instance-level constructs are Element, Attribute, and Primitive Content. The conformance connector between Element and Element Type, Attribute and Attribute Type, and Primitive Content and Primitive Content Type specify the schema-instance relationships. The conformance connectors only represent a structural connection between the constructs and do not fully define the meaning of the conformance (e.g., if an Element conforms to an Element Type, then each nested Element should conform to the appropriate nested Element Type).

Figure 8 shows example schema- and instance-level data that represent the XML document excerpt of Figure 4. (See [10] for more detailed examples of the triple representation). Notice that RDF provides a uniform representation of model, schema, and instance by allowing the model, schema, and instance data to be described using only RDF triples.

Figures 9 and 10 describe the Structured-Map and Structured-Bundle models using the UML visual

RDF XML Syntax

```
<RDF>
  …
  <ElementType ID="player_type">
    <elemTypeName value="player"/>
    <nestedElemType
resource="#position_type"/>
    <attTypeOf resource="playerName_attr"/>
  </ElementType>
  <ElementType ID="position_type">
    <elemTypeName value="position"/>
  </ElementType>
   <AttributeType ID="playerName_attr">
    <attName value="playerName"/>
  </AttributeType>
  <Element ID="player1">
    <elemInstOf resource="#player_type"/>
    <tagName value="player"/>
    <attributeOf resource="#playerName1"/>
    <nestedElem resource="# position1"/>
    <nestedElem resource="#rebounds1"/>
  </Element>
  <Attribute ID= "playerName1">
    <attInstOf resource="#playerName_attr"/>
    <attName value="playerName"/>
    <attValue value="Steve Smith"/>
  </Attribute>
  <Element ID="position1">
    <elemInstOf resource="#position_type"/>
    <tagName value="position"/>
  </Element>
  <Element ID="rebounds1">
    <tagName value="rebounds"/>
  </Element>
</RDF>
```

RDF Triples

```
…
(instanceOf, "player_type", ElementType)
(elemTypeName, "player_type", "player")
(nestedElemType, "player_type",  position_type)
(attTypeOf, "player_type", playerName_attr)

(instanceOf, "position_type", ElementType)
(elemTypeName, "position_type", "position")

(instanceOf, "playerName_attr", AttributeType)
(attName, "playerName_attr", "playerName")

(instanceOf, "player1", Element)
(elemInstOf, "player1", player_type)
(tagName, "player1", "player")
(attributeOf, "player1", playerName1)
(nestedElem, "player1", position1)
(nestedElem, "player1", e5)

(instanceOf, "playerName1", Attribute)
(attInstOf, "playerName1", playerName_attr)
(attName, "playerName1", "playerName")
(attValue, "playerName1", "Steve Smith")

(instanceOf, "position1", Element)
(elemInstOf, "position1", position_type)
(tagName, "position1", "position")

(instanceOf, "rebounds1", Element)
(tagName, "rebounds", rebounds)
```

Figure 8. Schema and instance data of the XML model.

representation. The Structured-Map model is a simplified version of the Topic-Map model, which uses a single level of schema and instance definition to allow Structured-Map data to be easily stored in a relational database. The model is designed for CARTE [12], which is a program that dynamically creates Web pages to navigate Structured-Map data. In CARTE, marks are represented as URLs. The user navigates through Topic Types, Topic Instances, and Topic Relations to reach Anchors, which contain marks. When a mark is selected, the referenced URL is displayed in a new Web browser window.

TopicType, TopicRelType, and AnchorType represent the schema constructs of the model. TopicInstance, TopicRelInst, AnchorInst, and Address represent the instance constructs of the model. CARTE requires schema-first definitions. For example, a TopicType (perhaps named "painter") must exist prior to creating a conforming TopicInstance (e.g., with the name "Van Gogh"). We express this requirement through the use of range multiplicity constraints on each conformance relationship.

The Structured-Bundle model is used by SLIMPad (*Superimposed-Layer Information Manager scratchPad*), which is a scratchpad application being built for the Traces project [20]. The goal is to develop specialized applications similar to SLIMPad that enable medical experts to organize important facts and issues excerpted from a base layer of digital medical documents. Currently, users interact with SLIMPad by selecting content from any of a number of information sources including XML documents, Microsoft PowerPoint slides, Excel spreadsheets, and PDF files, and dragging it into the scratch pad. Once content is placed into SLIMPad, a *scrap* is created that contains a mark with a reference back to the content. Scraps can be organized into *bundles* and bundles can be nested. By selecting a scrap, the content referenced by the corresponding mark (inside the scrap) is displayed and highlighted at the information source.

In SLIMPad, users can create and use templates as schema-level data. By instantiating a template, the user is provided with a set of default bundles organized hierarchically within the scratch pad. However, as shown by the multiplicity constraints, bundles can be created without an associated template. Within the medical domain, we see a number of templates being used for specialized tasks (e.g., templates for making drug-

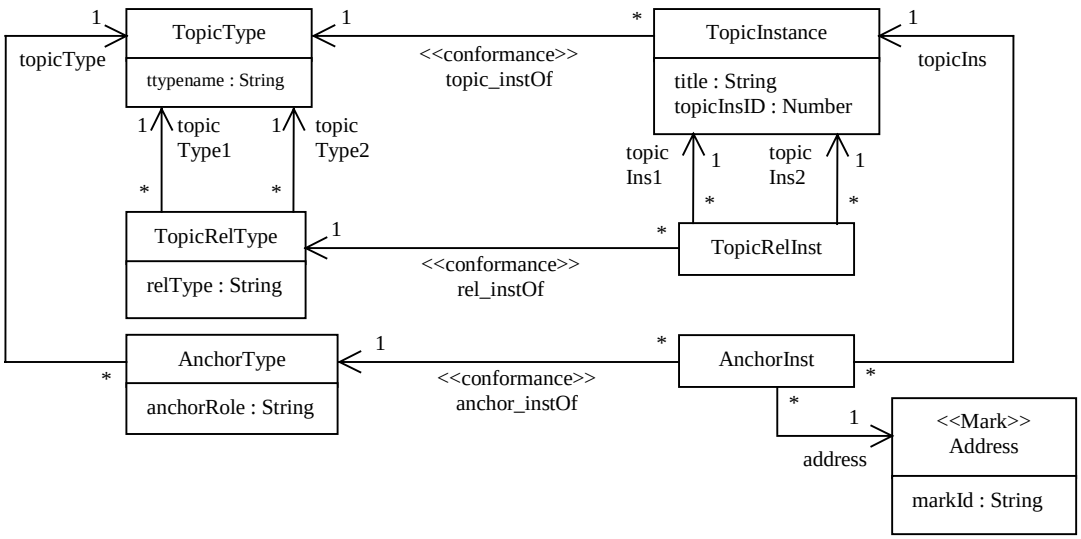

Figure 9. CARTE's Structured-Map model.

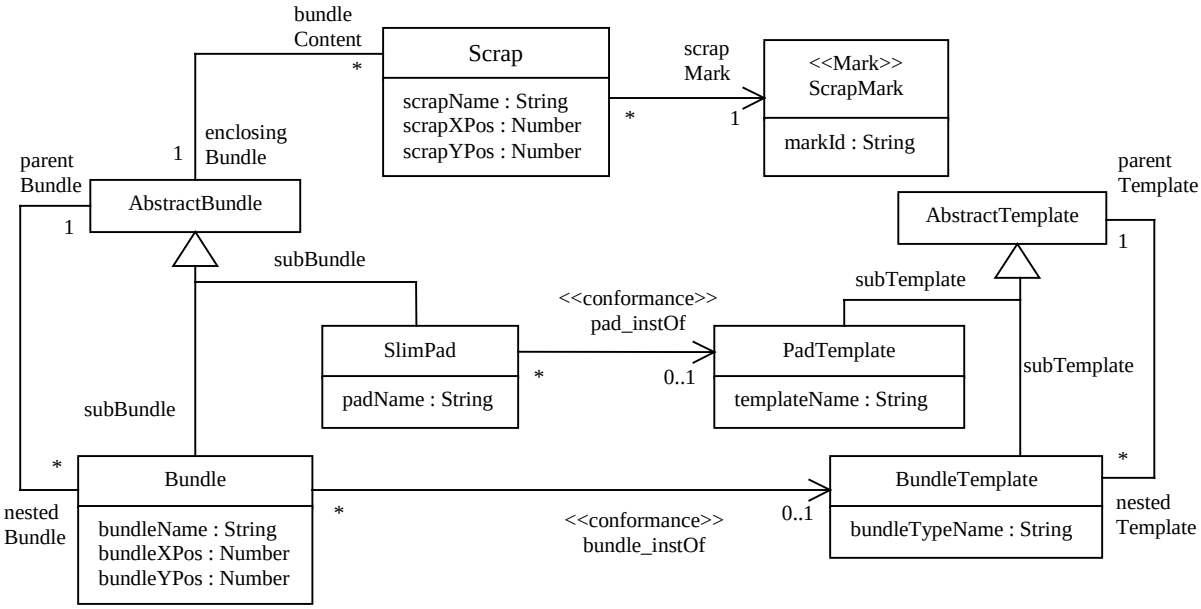

Figure 10. SLIMPad's Structured-Bundle Model.

interaction decisions for individual patients differ from templates for managing the state of an Intensive Care Unit). While SLIMPad and CARTE handle multiple schemas, we envision a number of schema-specific versions of SLIMPad designed around a single template (schema), in which the schema-level data is designed by an application developer.

Note that in Figure 10 we use the UML subclass symbol to denote generalization. However, we require that the relationship be named, since we are creating a new connector instance in the model. Additionally, arrows are not included in the association between a Scrap and an AbstractBundle since the association is bi-directional.

5 Mappings

In this section, we describe an approach to transforming information from one representation to another. Note that we can transform information between tools that use different models and schemas. In general, there will be various ways to map between information representations, depending on the user's desires and the applications of interest. Therefore, instead of trying to automatically generate mappings, we provide a technique for mappings to be specified manually, and perform the conversions as needed.

Table 2. Predicate and mapping rule definitions.

Symbol	Definition
τ	A predicate that represents an RDF triple, for example τ('creator', 'url', 'person').
L	A set (i.e., database) of triple predicates τ for superimposed layer l.
S	A predicate of the form $S(L, \tau)$ that is true if $\tau \in L$.
M	A mapping that consists of a set of mapping rules.
m	A mapping rule with the form: $T \Rightarrow T'$, where T, T' are sets of S predicates. The rule can be read as follows: if the left hand side matches (i.e., each $S \in T$ is true) then for each $S(L, \tau) \in T'$ add τ to L.

5.1 Mapping Rules

Mappings are specified using production rules. The rules are defined over triples of the RDF representation for superimposed information. Since a triple is a simple predicate (e.g., "triple(creator, index.html, Ora Lassilla)"), we specify mapping rules using a logic-based language such as Prolog, which allows us to both specify and implement the mappings. We do not require mappings between superimposed layers to be complete, since only part of a model or schema may be needed while using a specific tool.

Table 2 contains the basic definitions used to specify mappings. RDF triples are represented with the predicate τ. Quotes are used to denote constants and upper-case letters denote variables. For example, the predicate τ('creator', X, Y) is used in a mapping rule to match all triples that are related through the property "creator" (since X and Y are variables). The predicate S is true if its τ-predicate is in a superimposed layer L. For example, S('xml', τ('instanceOf', 'Element', 'Construct')) would be true if there were an "Element" construct defined in a superimposed layer named "xml." Finally, we define a mapping rule as a production in which the left- and right-hand sides consist of S-predicates. The left-hand side S-predicates must be true in order to generate the right-hand side S-predicates (i.e., the right-hand side S-predicates are produced by the rule). For example, the mapping rule S('source', τ('creator', X, Y)) $\Rightarrow$ S('target', τ('owner', X, Y)) would add a triple τ('owner', X, Y) to the superimposed layer named "target" for every triple that matched τ('creator', X, Y) in the superimposed layer named "source." Therefore, if τ('creator', 'index.html', 'Ora Lassilla') is a triple in the superimposed layer named "source," then the triple τ('owner', 'index.html', 'Ora lassilla') will be added to the superimposed layer named "target."

Table 3 describes the functions that are used to perform mappings. The conversion function applies a set of mapping rules to a source and target superimposed layer. Conversion generates a new superimposed layer that contains the generated triples of the mappings. In our current implementation, Prolog performs the conversion function.

The extract model function is used to extract model information from a superimposed layer. It can also be applied to the result of a conversion. For example, in an inter-model mapping, we want the resulting superimposed

Table 3. Functions used to provide mappings.

Function	Definition
Conversion : $M \times L_s \times L_t \rightarrow L_r$	Conversion takes a mapping M and applies the mapping rules of M to a source layer L_s and a target layer L_t, and returns a new layer L_r. Conversion is a basic rule-based algorithm that computes the fixed-point of applying mapping rules to the source and target superimposed layers.
ExtractModel : $L \rightarrow L'$	$L' = L_1 \cup L_2$ where: L_1 = {t | t$\in L$ and t=τ('instanceOf',X,Y} where X='Construct', 'Mark', 'Lexical', 'Connector', 'Conformance', or 'Generalization'} L_2 = {u | u,t$\in L$ and t=τ('instanceOf',X,Y) where Y='Connector', 'Conformance', or 'Generalization' and u=τ(P,X,Z)}.
ExtractSchema : $L \rightarrow L'$	$L' = L_1 \cup L_2$ where: L_1 = {v | t,u,v$\in L$ and t=τ('instanceOf',P,'Conformance'} and u=τ(P,Y,Z) and v=τ('instanceOf',Z,X)} L_2 = {u | u $\in L$ and t1,t2$\in$ L_1 and t1=τ('instanceOf',S1,X) and t2=τ('instanceOf',S2,Y) and u=τ(P,S1,S2)}.
guid $\rightarrow x$	A 0-ary Skolem function that returns a unique identifier x.

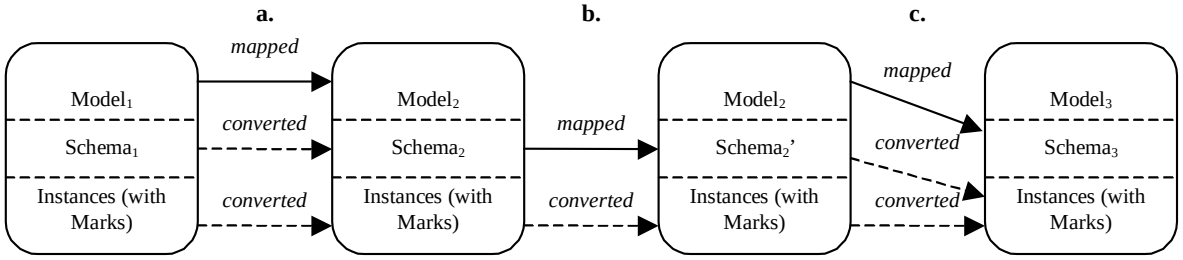

Figure 11. Three mappings: (a) inter-model, (b) inter-schema, and (c) model-to-schema.

layer to contain the model of the target layer. By using the extract model function, we can add the appropriate triples of the target layer to the superimposed layer returned by the conversion function.

Similarly, the extract schema function gathers schema information from the target layer. Extract schema returns the construct instances that are at the schema-ends of conformance connectors along with the connections between the schema construct instances. (Note that this approach works for cases where there is only one level of schema and instance). To add target model and schema information to the result of an inter-schema mapping we would use the extract model and schema functions as follows: conversion(M, L_s, L_t) ∪ extractModel(L_t) ∪ extractSchema(L_t), where L_t is the target layer, L_s the source layer, and the result of the clause is a new superimposed layer.

5.2 Inter-Model, Inter-Schema, and Model-to-Schema Mappings

Figure 11 illustrates three types of mappings. Each example shows information from a source layer being mapped to a target layer to convert data from the source layer into data that conforms to the target layer. Although we focus on conversion, it is also possible to perform integration between superimposed layers. Integration goes a step further by combining the source and target data. The mapping rules can be used to provide integration at both the schema and instance levels.

Figure 11(a) is an inter-model mapping in which the schema- and instance-level data of the source superimposed-layer are converted to valid schema- and instance-level data of the target superimposed-layer. Figure 12 shows an inter-model mapping between the Structured-Bundle model and the Structured-Map model. The goal of the mapping is to allow SLIMPad data to be used with CARTE.

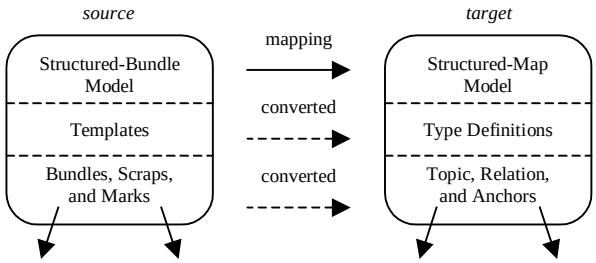

Figure 12. An inter-model mapping between the Structured Bundles and Structured Maps.

Figure 13 illustrates four mapping rules between the Structured-Bundle model (shown as the source) and the Structured-Map model (shown as the target) using the UML model representation. The first mapping rule, Figure 13(a), specifies a mapping between the Bundle Template schema-construct and the Topic Type schema-construct. That is, all Bundle Templates in the source

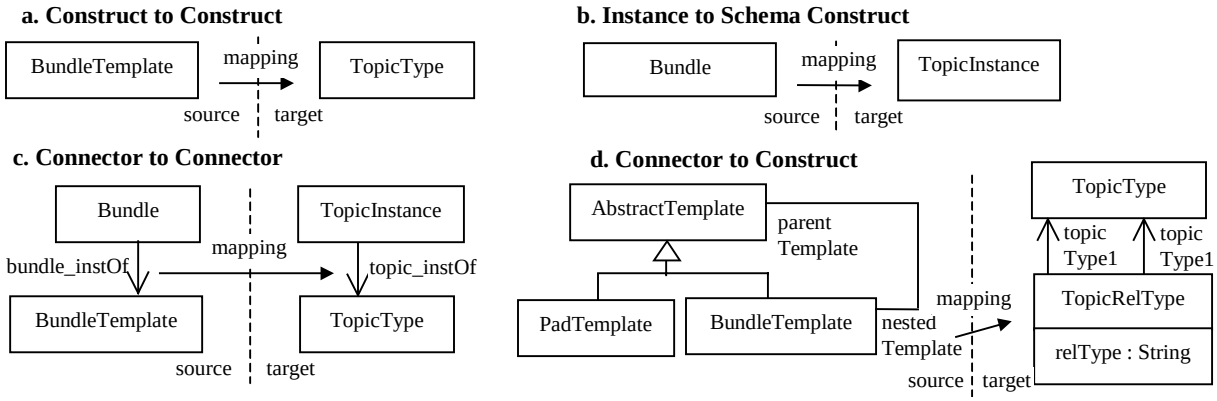

Figure 13. Inter-model mappings from Structured Bundles to Structured Maps represented visually.

a. Construct to Construct
S('source', τ('instanceOf', X, 'BundleTemplate'))
⇒
S('result', τ('instanceOf', X, 'TopicType'))

b. Instance to Schema Construct
S('source', τ('instanceOf', X, 'Bundle'))
⇒
S('result', τ('instanceOf', X, 'TopicType'))

c. Connector to Connector
S('source', τ('bundle_instOf', X, Y)),
S('result', τ('instanceOf', X, 'TopicInstance')),
S('result', τ('instanceOf', Y, 'TopicType'))
⇒
S('result', τ('topic_instOf', X, Y))

d. Connector to Construct
S('source', τ('nestedTemplate', P, B)),
S('result', τ('instanceOf', P, 'TopicType')),
S('result', τ('instanceOf', B, 'TopicType')),
X = guid()
⇒
S('result', τ('instanceOf', X, 'TopicRelType')),
S('result', τ('relType', X, 'nested_template')),
S('result', τ('topicType1', X, P)),
S('result', τ('topicType2', X, B))

Figure 14. Inter-model mappings from Structured Bundles to Structured Maps.

superimposed-layer will be converted to Topic Types in the resulting superimposed layer. Figure 13(b) is a mapping between Bundles and Topic Instances, which are both at the instance-level. Figure 13(c) is a mapping between two conformance connectors: bundle_instOf and topic_instOf. The mapping states that each bundle_instOf relationship in the source-layer should be converted to a topic_instOf relationship in the target-layer. Finally, the mapping in Figure 13(d) shows the nestedTemplate connector being mapped to a Topic Relationship Type. As the data is converted, a new Topic Relationship Type will be created with the relType attribute set to the string "nested_template," the domain of the nestedTemplate connector assigned to the range of topicType1, and the range of the NestedTemplate connector assigned to the range of topicType2.

Figure 14 shows the mapping rules that specify the mappings of Figure 13. We use the constant 'source' to represent the Structured Bundles superimposed layer, the constant 'target' to represent the Structured Maps superimposed layer, and the constant 'result' to represent the new superimposed layer that is created from the mappings. (See [10] for a more detailed exposition of the mappings).

Figure 11(b) is an inter-schema mapping in which the source and target models are the same, but two distinct schemas are mapped so that the source instance-level data can be converted to data that conforms to the target schema. Figure 15 illustrates an inter-schema mapping using the XML model. The source is an animal taxonomy DTD containing Element Types such as genus and species. The target is a bookmark list DTD with Element Types such as folder and bookmark. One reason to perform this type of mapping is to reuse existing tools for browsing bookmark lists on taxonomies.

Figure 16 demonstrates three rules that are used to perform part of the mapping of Figure 15. The first rule takes content that is designated as genus (e.g., "homos") and maps it to a folder (titled "homos"). Similarly, species content (e.g., "sapiens") is mapped as a nested folder (titled "sapiens") within a genus folder.

The last mapping of Figure 11, shown as Figure 11(c), is called a model-to-schema mapping. Here, the model of the source layer is mapped to schema-level data in the target layer, which allows the schema- and instance-level data of the source to be converted to valid instance-level data in the target. Figure 17 shows a model-to-schema mapping in which the Structured-Map model is mapped to an XML DTD. The Structured-Map schema-level and instance-level data are converted to an XML document. The benefit of doing this mapping is to use XML as the interchange format for Topic Maps.

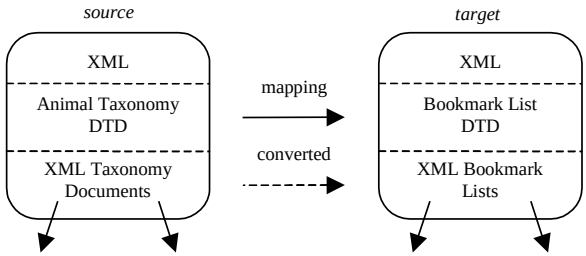

Figure 15. An example of a schema-to-schema mapping between two DTDs.

To specify the mapping, we could map a Topic Instance construct in the Structured-Map model to an Element Type in an XML DTD with an Attribute Type titled "name". Then, for a particular Topic Instance (e.g., "painter"), the conversion would result in the XML tag `<topic name="painter"/>`. Figure 18 shows two rules to perform the mapping. Notice that the first rule has an empty left-hand side. Rules without left-hand sides automatically match, which means that the right-hand side triples are always added.

Source Schema

τ(‘instanceOf’, ‘genus_type’, ‘ElementType’)
τ(‘elementTypeName’, ‘genus_type’, ‘genus’)
τ(‘instanceOf’, ‘species_type’, ‘ElementType’)
τ(‘elementTypeName’, ‘species_type’, ‘species’)
τ(‘nestedElement’, ‘genus_type’, ‘species_type’)

Target Schema

τ(‘instanceOf’, ‘folder_type’, ‘ElementType’)
τ(‘elementTypeName’, ‘folder_type’, ‘folder’)
τ(‘nestedElement’, ‘folder_type’, ‘folder_type’)

Mapping Rules

S(‘source’, τ(‘elemInstOf’, X, ‘genus_type’))
⇒
S(‘result’, τ (‘elemInstOf’, X, folder_type))

S(‘source’, τ(‘elemInstOf’, X, ‘species_type’))
⇒
S(‘result’, τ(‘elemInstOf’, X, ‘folder_type’))

S(‘source’, τ(‘nestedElement’, X, Y))
⇒
S(‘result’, τ(‘nestedElement’, X, Y))

Figure 16. Example of a schema-to-schema mapping between two XML model schemas.

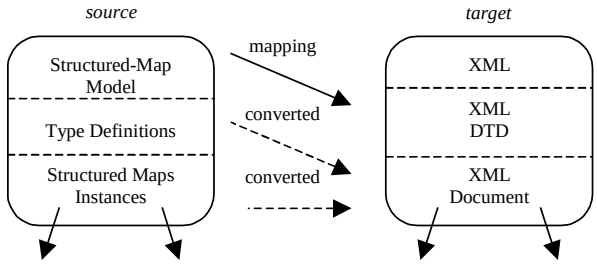

Figure 17. Example of a model-to-schema mapping.

6 Comparison of Related Work

A number of metamodels have been developed (see Atzeni and Torlone [1-4], Barsalou and Gangopadhyay [5], McBrien and Poulovassilis [17], and the Meta Object Facility [18]) with the primary focus on supporting interoperability. Our metamodel is different from these approaches because we do not require model-first nor schema-first definitions. Rather, we support the independent specification of model, schema, and instance and we permit the application to explicitly specify the relationship between schema and instance. Metamodels for describing both database data models [1-4, 5, 17] and object-oriented models [18-19] require that instances be the extension of schema in which schema must be defined first. By not enforcing schema-first definitions and allowing instances to be independent of schema, our metamodel is able to accurately define superimposed models such as XML and Topic Maps. Additionally, by explicitly representing the relationship between schema and instance we can specify more complex situations such as multiple levels of schema-instance relationships.

Another major difference between our approach and other metamodel approaches is that we employ a single, generic representation scheme for model, schema, and

Source Model

S(‘source’, τ(‘instanceOf’, ‘TopicInstance’, ‘Construct’))
S(‘source’, τ(‘instanceOf’, ‘title’, ‘Connector’))
S(‘source’, τ(‘instanceOf’, ‘t1’, ‘TopicInstance’))
S(‘source’, τ(‘title’, ‘t1’, ‘painter’))

Target Model

S(‘target’, τ(‘instanceOf’, ‘ElementType’, ‘Construct’))
S(‘target’, τ(‘instanceOf’, ‘AttributeType’, ‘Construct’))
S(‘target’, τ(‘instanceOf’, ‘elementTypeName’, ‘Connector’))
S(‘target’, τ(‘instanceOf’, ‘attTypeOf’, ‘Connector’))
S(‘target’, τ(‘instanceOf’, ‘attTypeName’, ‘Connector’))
S(‘target’, τ(‘instanceOf’, ‘Element’, ‘Construct))
S(‘target’, τ(‘instanceOf’, ‘Attribute’, ‘Construct’))
S(‘target’, τ(‘instanceOf’, ‘elemInstOf’, ‘Conformance’))
S(‘target’, τ(‘instanceOf’, ‘attInstOf’, ‘Conformance’))
S(‘target’, τ(‘instanceOf’, ‘tagName’, ‘Connector’))
S(‘target’, τ(‘instanceOf’, ‘attName’, ‘Connector’))
S(‘target’, τ(‘instanceOf’, ‘attValue’, ‘Connector’))
S(‘target’, τ(‘instanceOf’, ‘attributeOf’, ‘Connector’))

Mapping Rule:

⇒
S(‘result’, τ(‘instanceOf’, ‘topic_inst’, ‘ElementType’)),
S(‘result’, τ(‘elemTypeName’, ‘topic_inst’, ‘topic’)),
S(‘result’, τ(‘instanceOf,’ ‘topicInst_att’, ‘AttributeType’)),
S(‘result’, τ(‘attTypeName’, ‘topicInst_att’, ‘name’)),
S(‘result’, τ(‘attTypeOf’, ‘topic_inst’, ‘topicInst_att’))

S(‘source’, τ(‘instanceOf’, X, ‘TopicInstance’)),
S(‘source’, τ(‘title’, X, Y))
⇒
S(‘target’, τ(‘instanceOf’, X, ‘Element’)),
S(‘target’, τ(‘tagName’, X, ‘topic’)),
S(‘target’, τ(‘elemInstOf’, X, ‘topic_inst’)),
Z = guid(),
S(‘target’, τ(‘instanceOf’, Z, ‘Attribute’)),
S(‘target’, τ(‘attName’, Z, ‘name’)),
S(‘target’, τ(‘attInstOf’, Z, ‘topicInst_Att’)),
S(‘target’, τ(‘attributeOf’, X, Z)),
S(‘target’, τ(‘attValue’, Z, Y))

Figure 18. A model-to-schema mapping rule between the Structured-Map model and XML.

instance data. The representation scheme allows mappings to be defined in a uniform way between models (inter-model), schema (inter-schema), model and schema (model-to-schema), and any mixture of the three levels. McBrien and Poulovassilis use the Hypergraph Data Model (HDM) to store schemas defined in diverse models. Their primary goal is to perform inter-model transformations, which are specified using first-order logic expressions, to map between semantically equivalent schemas. They also specify transformations from the extent of a schema in one model (e.g., the relational model) to the extent of a similar schema in a different model (e.g., the entity-relationship model). However, both types of transformations are considerably more difficult to specify, when compared to our mapping rules, because they do not explicitly represent models or instances using the HDM. Atzeni and Torlone employ procedural inter-model mapping specifications. Similar to our approach, their specifications can be used to implement the conversions. However, they require complete mappings between models, whereas we allow partial mappings to allow for a wider range of cases, and they do not provide support for other types of mappings (e.g., inter-schema or model-to-schema mappings).

The Meta Object Facility (MOF) defines an architecture that uses a metamodel to enable the sharing of information between object-oriented applications. Currently, the main application of the MOF architecture is to store and interchange UML class diagrams between analysis and design tools. The MOF uses the XML Metadata Interchange (XMI) as a representation scheme for exchange. XMI prescribes a method to generate an XML DTD to represent a model. (Note that the XML DTD is generated by hand and UML is the only version currently available). XML documents that conform to the DTD represent schema-level data. Unlike our approach, there is no way to represent instance-level data. Also, MOF does not provide any support for mapping between models. The Microsoft Repository [6-7] is similar to the MOF, except it does not define a metamodel. Instead, a global model called the Open Information Model is used to define schemas. However, our approach provides a mechanism to represent various models precisely to leverage available tools that are based on a particular model.

7 Conclusions and Future Work

The metamodel and representation as well as the mapping approach presented here have been deployed in a project that provides additional indexing for a base of XML documents. By first parsing the base documents using a standard XML parser, we generate the equivalent triples that represent the XML. A mapping is then used to automatically generate an equivalent Topic Map, based on the XML. The resulting Topic Map is then presented to users to navigate and search through items of interest in the XML document base.

We are also developing generic technology to manage information, e.g., to store, create, manipulate, and retrieve information, based on our representation. Our goal is to allow an application developer to specify a model or model-schema combination from which we can automatically generate application-specific APIs to store and manage superimposed information. Thus, developers aren't required to use the generic representation directly (instead they can use a tailored API) and still have the benefits of generic representation beneath the hood [14]. We have also successfully integrated our superimposed information management approach into both CARTE and SLIMPad, which allows the two applications to share information via mappings.

The contributions of our work are listed here:

1. **The Superimposed-Information Metamodel:** A unique metamodel for describing multiple superimposed models, which allows for the explicit specification of the relationship between schema and instance and allows marks to be placed anywhere within a model.
2. **A Generic Representation Scheme for Superimposed Information:** A single representation schema, based on RDF, that leverages the superimposed information metamodel, to uniformly represents model, schema, and instance data and can be used generically by diverse superimposed applications.
3. **Visual Superimposed-Model Definition:** The ability to define superimposed models visually by using a subset of the UML. We find that being able to define and view models visually is a benefit, and in addition, model engineers can leverage a number of existing tools that support UML.
4. **Information Sharing:** Using a common representation format for superimposed information makes it possible for applications to dynamically share data.
5. **Superimposed-Layer Mappings:** A formal method, based on production rules, to uniformly specify superimposed-layer mappings. We allow partial mappings as well as mappings between multiple levels of superimposed information.

Acknowledgments

The authors would like to thank David Maier for his helpful discussions, contributions, and comments on this

work. We would also like to acknowledge the other SLIM project members Longxing Deng and Mathew Weaver for their contributions to this research, including their work on the development of the SLIMPad application.

References

[1] Paolo Atzeni and Riccardo Torlone. A metamodel approach for the management of multiple models and the translation of schemes. *Information Systems* 18(6), pages 349-362, September 1993.

[2] Paolo Atzeni and Riccardo Torlone. MDM: a multiple-data-model tool for the management of heterogeneous database schemes. *ACM SIGMOD Conference*, Tucson, Arizona, May 13-15, 1997.

[3] Paolo Atzeni and Riccardo Torlone. Management of multiple models in an extensible database design tool. *5th International Conference on Extending Database Technology EDBT '95*, Lecture Notes in Computer Science Volume 1057, Avignon, France, March 25-29, 1996.

[4] P. Atzeni and R. Torlone, Schema translation between heterogeneous data models in a lattice framework. *6th IFIP TC-2 Working Conference on Database Semantics (DS-6)*, Atlanta, Georgia, May 30-June 2, 1995.

[5] Thierry Barsalou and Dipayan Gangopadhyay. M(DM): an open framework for interoperation of multimodel multidatabase systems. *Eighth International Conference on Data Engineering ICDE'92*, Tempe, Arizona, February 3-7, 1992.

[6] Philip A. Bernstein and Thomas Bergstraesser. Meta-data support for data transformations using Microsoft Repository. *IEEE Data Engineering Bulletin* 22(1), pages 9-14, March 1999.

[7] Philip A. Bernstein, Brian Harry, Paul Sanders, David Shutt, and Jason Zander. The Microsoft Repository. *Proceedings of the 23rd International Conference on Very Large Databases VLDB '97*, Athens, Greece, August 25-27, 1997.

[8] Michel Biezunski, Martin Bryan, and Steve Newcomb, editors. ISO/IEC 13250, *Topic Maps*, URL:http://www.ornl.gov/sgml/sc34/document/0058.htm.

[9] Tim Bray, Jean Paoli, and C.M. Sperger-McQueen, editors. Extensible Markup Language (XML) 1.0, W3C Recommendation 10-February-1998, URL:http://www.w3.org/TR/REC-xml.

[10] Shawn Bowers. A generic approach for representing model-based superimposed information. Oregon Graduate Institute of Science and Technology, Technical Report Number CSE-00-008, May 1, 2000.

[11] Dan Brickley and R.V. Guha, editors. Resource Description Framework Schema (RDFS), W3C Proposed Recommendation 03 March 1999, URL:http://www.w3.org/TR/PR-rdf-schema/.

[12] Lois Delcambre, David Maier, Radhika Reddy, and Lougie Anderson. Structured maps: modeling explicit semantics over a universe of information. *International Journal on Digital Libraries* 1(1), pages 20-35, 1997.

[13] Lois Delcambre and David Maier. Models for superimposed information. *Advances in Conceptual Modeling ER '99*, Lecture Notes in Computer Science Volume 1727, pages 264-280, Paris, France, November 15-18, 1999.

[14] Lois Delcambre, David Maier, Shawn Bowers, Longxing Deng, Mathew Weaver, Paul Gorman, Joan Ash, Mary Lavelle, and Jason A. Lyman. Bundles in captivity: an application of superimposed information. Oregon Graduate Institute of Science and Technology, Technical Report Number CSE-00-010, June, 2000.

[15] Ora Lassila and Ralph R. Swick, editors. Resource Description Framework (RDF) Model and Syntax Specification, W3C Recommendation 22 February 1999, URL:http://www.w3.org/TR/REC-rdf-syntax.

[16] David Maier and Lois Declambre. Superimposed information for the Internet. *ACM SIGMOD Workshop on The Web and Databases WebDB'99*, pages 1-9, Philadelphia, Pennsylvania, June 3-4, 1999.

[17] Peter McBrien and Alexandra Poulovassilis. A uniform approach to inter-model transformations. *11th International Conference on Advanced Information Systems Engineering CAiSE'99*, Lecture Notes in Computer Science Volume 1626, pages 333-348, Heidelberg, Germany, June 14-18, 1999.

[18] Object Management Group. *Meta Object Facility (MOF) Specification*. OMB Document ad/99-09-04. URL:http://www.omg.org/cgi-bin/doc?ad/99-09-04.

[19] James Rumbaugh, Ivar Jacobson, and Grady Booch. *The Unified Modeling Language Reference Manual*. Addison Wesley, 1999.

[20] Tracking footprints through an information space: leveraging the document selections of expert problem solvers. NSF Grant IIS-9817492. Digital Libraries Phase 2. Paul Gorman, Lois Delcambre, and David Maier.