Generating XSLT with a Semantic Hub

Transformations for the Semantic Web

Keywords: Semantics, Semantic Hub, Semantic Model, Semantic Web, EAI, Information Model, Business Model, Ontology, XML, XSLT, XSL, XSLT-Generation, XSLT Development, XSD, XML Schemas, Transformation, Code Generation

Joshua Fox
Software Architect
Unicorn Solutions Inc.
New York
New York
United States
joshua@unicorn.com
http://www.unicorn.com

Biography

Joshua Fox serves as Software Architect at Unicorn Solutions, working on the Unicorn Coherence™ platform for unification of information in the enterprise. Fox's previous experience includes the design and development of large-scale distributed Internet systems for collaboration over the Internet. He earned a B.A. *summa cum laude* in Mathematics from Brandeis University and a Ph.D. in Comparative Semitic Philology from Harvard University, where he also served as Lecturer. He has published and lectured extensively in the field of software engineering.

Abstract

XSLT is the standard technique for integrating heterogeneous XML-based applications, whether in messaging environments like EAI or in request/response systems like the Semantic Web.

With the techniques commonly used today, writing XSLT to integrate multiple XML formats requires significant effort. The usual procedure requires an analysis of both the semantics and the structures of the source and target XML files, often by examining instance documents. Manual coding then follows this analysis.

A partial step towards in reducing the time and errors in this procedure is to use schemas or DTDs; software can use these to assist in developing the XSLT. Currently available graphical tools can do this, but they still require developers to manually indicate mappings for each source-target pair.

The heart of the problem is that schemas only formalize the documents' structure, not their semantics. As a result, the XSLT developer must make the effort in each case to re-analyze the "meaning" of each schema's XML tags. The complexity of this repeated effort in developing point-to-point XSLTs rises quadratically ($O(n^2)$) in the number of schemas that must be integrated. Such XSLTs cannot be maintained or reused as schemas change and as new schemas are added.

A new solution involves capturing the semantics (meaning) of the schemas and using these semantics to automatically generate the necessary XSLT's.

The developer first defines a rich information model using ontology, a formal technique representing real-life semantics through concepts such as classes, relationships, and inheritance. This model is itself valuable in clarifying the application domain. The developer then maps the schema's Elements, Complex Types, and Simple Types to the information model, thereby formally capturing the schemas' semantics.

In the next step, the active semantic hub is used to generate the XSLT based on the elements' meanings. The algorithm finds elements of the source and target that are mapped to the same ontological concepts, or to concepts that can be related to each other with encoded conversion rules. This gives the well-known advantage of linear (O(n)) complexity in a hub architecture as opposed to the quadratic complexity of point-to-point solutions.

In the final step, the XSLT is deployed for runtime, for example in an EAI message broker or in a Semantic Web application. This deployment can be manual or automated.

Table of Contents

Introduction

Writing XSLT today requires significant effort. The usual procedure requires an analysis of both the semantics and the structures of the source and target XML files, often by examining instance documents. Manual coding then follows this analysis. Currently available graphical tools can help, but still require developers to manually indicate mappings for each source-target pair.

The heart of the problem is that schemas only formalize the documents' structure, not their semantics. As a result, the XSLT developer must make the effort in each case to re-analyze the "meaning" of each schema's XML tags.

A new solution uses a central information model to eliminate this point-by-point development effort. This information model allows formalization of the semantics of XSD schemas. The software can then generate XSLT code, reducing the quadratic complexity of the point-to-point solution to the linear complexity of the star-shaped approach.

XSLT: Hard to Write

A Functional, XML-based Language

Cameron Laird, in a recent *Linux Journal* article**[CL02]** , described the difficulty of writing XSLT: "Another hurdle in XSLT's diffusion, along with its unconventional XML-based syntax and confusing deployment, is its functional or applicative semantics." These problems make XSLT not only difficult to develop: It makes XSLT difficult to modify as schemas change and difficult to reuse transforming to and from new schemas.

Martin Fowler, in a recent book on enterprise architecture**[MF02]** , reinforces Laird's sentiment :"XSLT can also be an awkward language to master, due [to] its functional programming style coupled with its awkward XML syntax."

XSLT is one of only two popular functional languages, along with SQL. As with SQL, it is considered hard to write and read large programs in XSLT.

Functional programming is not a popular paradigm.

- Many programmers do not understand functional programming.
- Functional programs can be harder to debug because of its non-sequential nature, and, in fact, few such tools are available for XSLT.
- Few functional languages are commonly used.

XSLT: XML-based Syntax

XSLT's XML-based syntax is an additional burden.

- It is harder than line-based syntax for humans to understand.
- Its nesting and tightly validated structure can be confusing: End-tags must always match exactly.
- It is hard to process in line-based development tools like *merge*.
- The XML tags make the language verbose.

XSLT Development Today

Today, XSLT development is characterized by schema-less or schema-to-schema development.

The Most Common Development Technique

The most common approach today is the manual development of XSLT. This approach does not use schemas (XSD or DTD) at all. Typically, the developer uses a sample instance document for the source of the XSLT and generates sample documents to test the result.

This does not imply that source and target documents are arbitrarily structured. Rather, the schemas for the source and target documents are implicit in the code, and must be understood by the developer.

Manual Schema-to-Schema Development

In a more sophisticated approach, a schema is used if available. The developer can refer to it while creating the XSLT.

Still, the schema has no formal role. It is used only by the human developer, who must analyze its meaning and apply it to the work of developing the XSLT. The schemas for source and target are not used formally in the development of the XSLT.

Schema-less XSLT Generation Tools

Martin Fowler (**[MF02]**) says "Tools for XSLT are, at least so far, much less sophisticated" than tools for other programming languages."

One of the most obvious limits in the sophistication of XSLT tools is the lack of use of schemas. Despite the constraints on source and target XML schemas that are assumed by every XSLT, the tools do not take into account explicitly declared XSDs or DTDs. With these tools, the developer uses the GUI to indicate a transformation. For example, the developer may indicate that the third position in each source document is to be copied into the fourth position in the target document. These definitions, however, are still on the level of instance documents, without formal schemas.

XSLT-Generation Tools for HTML

Some GUI tools are oriented towards generating HTML. As such, the format of the target document is implicit in its being HTML; if the target is XHTML, the target schema is even explicit.

These tools represent a step forward, yet they are suited only for user-interface generation.

Schema-to-Schema XSLT-Generation Tools

Some more advanced tools do go beyond HTML generation, and take source schemas into account. These tools allow developers to juxtapose a schema for the source and another schema for the target, where the target is not necessarily HTML but rather an arbitrary schema. The user connects elements in the source and target schemas to produce XSLT.

Message brokers are software components that aid in application integration by directing messages to the correct application and by translating between the messages used by different applicaitons. Message brokers are typically bundled with transformation design tools, although often they produce languages other than XSLT, and are designed to transform proprietary document formats. The problem with these techniques is that they require effort for each pair of source and target schemas.

The Problem: Point-to-Point Development

An Example

Let's look at a simplified example of three instance documents, each with its own schema.

Instance document 1

```
<?xml version="1.0" encoding="UTF-8"?>
<computer>
        <harddisk>
               <capacity>30</capacity>
        </harddisk>
</computer>
```

Instance Document 2

```
<?xml version="1.0" encoding="UTF-8"?>
<desktop_computer>
  <harddrive>
   <partition drive="c:" capacity="10" unit="GB"/>
   <partition drive="d:" capacity="20" unit="GB"/>
  </harddrive>
</desktop_computer>
```

Instance Document 3

```
<?xml version="1.0" encoding="UTF-8"?>
<unit type="desktop">
 <storage>30000</storage>
</unit>
```

Schema 1

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified">
```

```
 <xs:element name="capacity" type="xs:integer"/>
  <xs:element name="computer">
   <xs:complexType>
    <xs:sequence>
     <xs:element ref="harddisk"/>
    </xs:sequence>
   </xs:complexType>
  </xs:element>
  <xs:element name="harddisk">
   <xs:complexType>
    <xs:sequence>
     <xs:element ref="capacity"/>
    </xs:sequence>
   </xs:complexType>
  </xs:element>
</xs:schema>
```

Schema 2

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified">
 <xs:element name="desktop_computer">
  <xs:complexType>
   <xs:sequence>
    <xs:element name="harddrive">
     <xs:complexType>
      <xs:sequence>
       <xs:element name="partition" maxOccurs="unbounded">
        <xs:complexType>
         <xs:attribute name="drive" type="xs:string" use="required"/>
         <xs:attribute name="capacity" type="xs:integer" use="required"/>
         <xs:attribute name="unit" type="xs:string" use="required" fixed="GB"/>
        </xs:complexType>
       </xs:element>
      </xs:sequence>
     </xs:complexType>
    </xs:element>
   </xs:sequence>
  </xs:complexType>
 </xs:element>
</xs:schema>
```

Schema 3

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified">
 <xs:element name="storage" type="xs:integer"/>
 <xs:element name="unit">
  <xs:complexType>
   <xs:sequence>
     <xs:element ref="storage"/>
   </xs:sequence>
   <xs:attribute name="type" type="xs:string" use="required"/>
  </xs:complexType>
 </xs:element>
</xs:schema>
```

Quadratic vs. Linear Complexity

As seen in the following figure (**Fig. 1**), even three XSDs require six point-to-point development efforts.

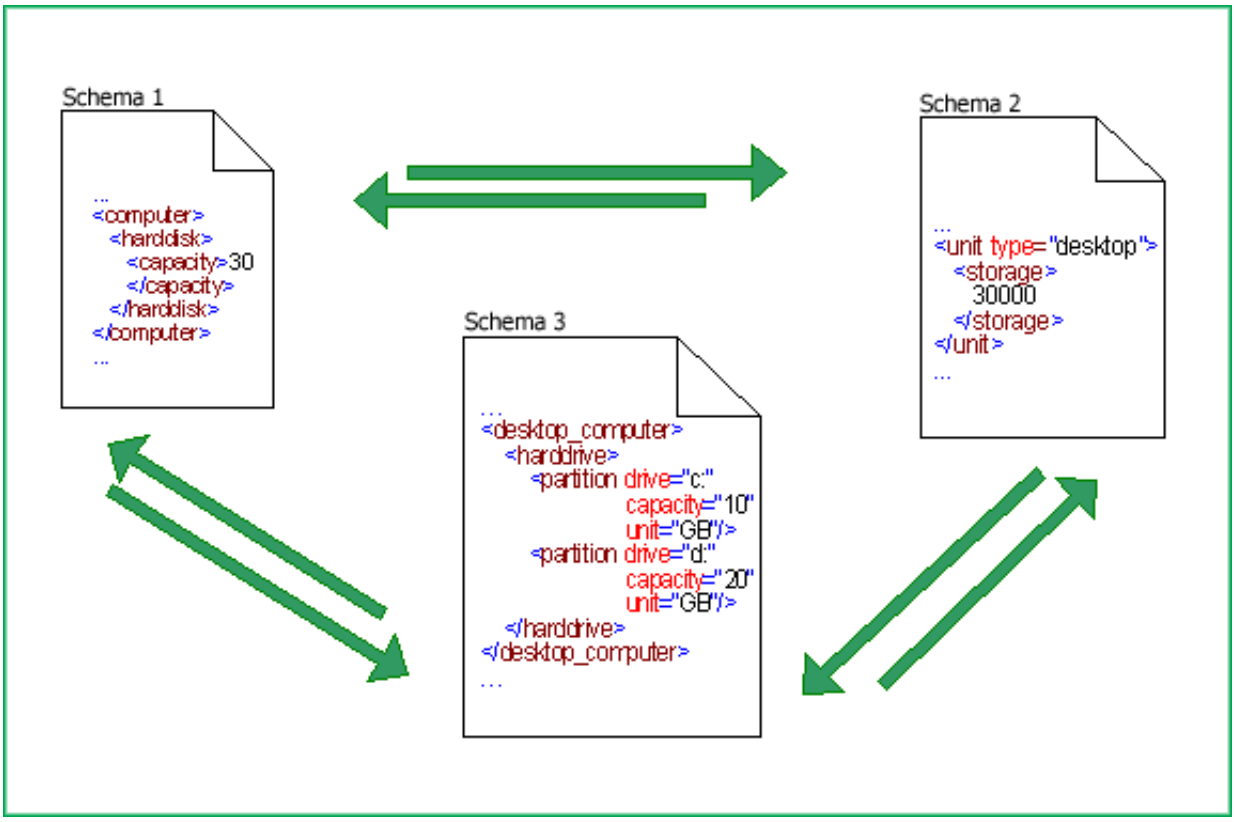

Fig. 1: Point-to-Point Transformation of Three XML Schemas

As the number of schemas increases, more and more manual effort is needed, rising quadratically, as $O(n^2)$ (**Fig. 2**).

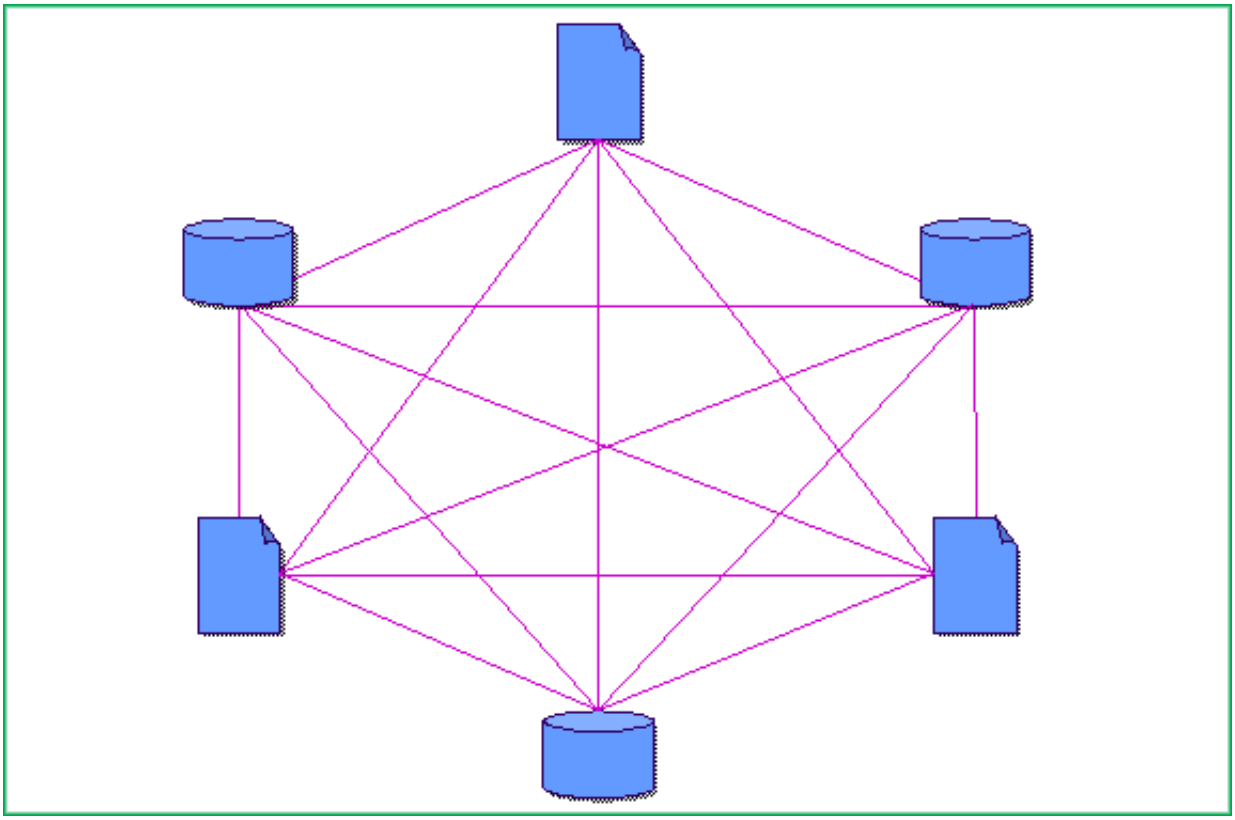

Fig. 2: Point-to-Point: Quadratic

Problem: Repeated Work per Schema-Pair

The work that must be done repeatedly for each source/target pair includes:

1. Syntactic analysis of schemas of instance documents, if not present
2. Semantic analysis of the meaning of schema elements
3. Development of XSLT

Beyond the needless $O(n^2)$ effort in point-to-point development, the most common development method involves other types of needless repetition: comprehensive analysis of the syntax and the semantics of both the source and the target documents must be done every time a new point-to-point transformation arises.

The Semantic Hub

The proposed solution is a semantic hub, a semantic software dictionary which unites syntactical schemas. This software can assist an EAI message broker or a Semantic Web server by producing XSLT to automatically transform the output of one application into the expected input of another.

Structure of the Hub

The semantic hub includes an information model at its core, supplemented by mappings to the schemas:

1. The infrastructure is an information model, at the center of the semantic hub.
2. Layered on top of this is a mapping between the schemas and the ontological concepts.
3. Automated generation of XSLT is the final output of the semantic hub. (See **Fig. 3**)

The following sections of this presentation will delve into each of these layers in greater detail.

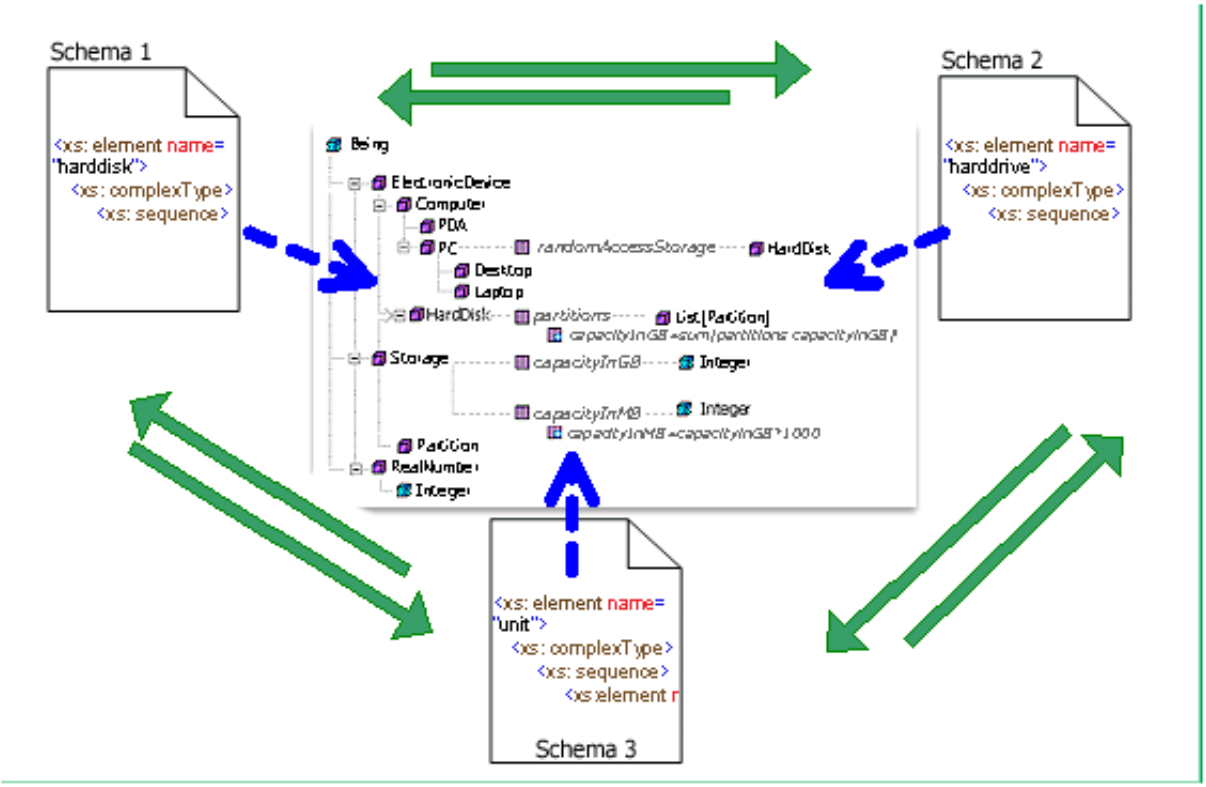

Fig. 3: The Structure of the Semantic Hub

Ontology: A Formal Technique for Building Information Models

The information model lies at the core of semantic hub. The model is structured according to the techniques of ontology (see an example in **Fig. 4**), which allow the formal expression of sets of entities, the relationship between these entities, and the constraints on the relationships. Unlike natural-language documents produced by systems analysts, ontological models express the business domain formally and precisely, and so can be used by the other layers of the semantic hub for functionality, such as generating XSLT.

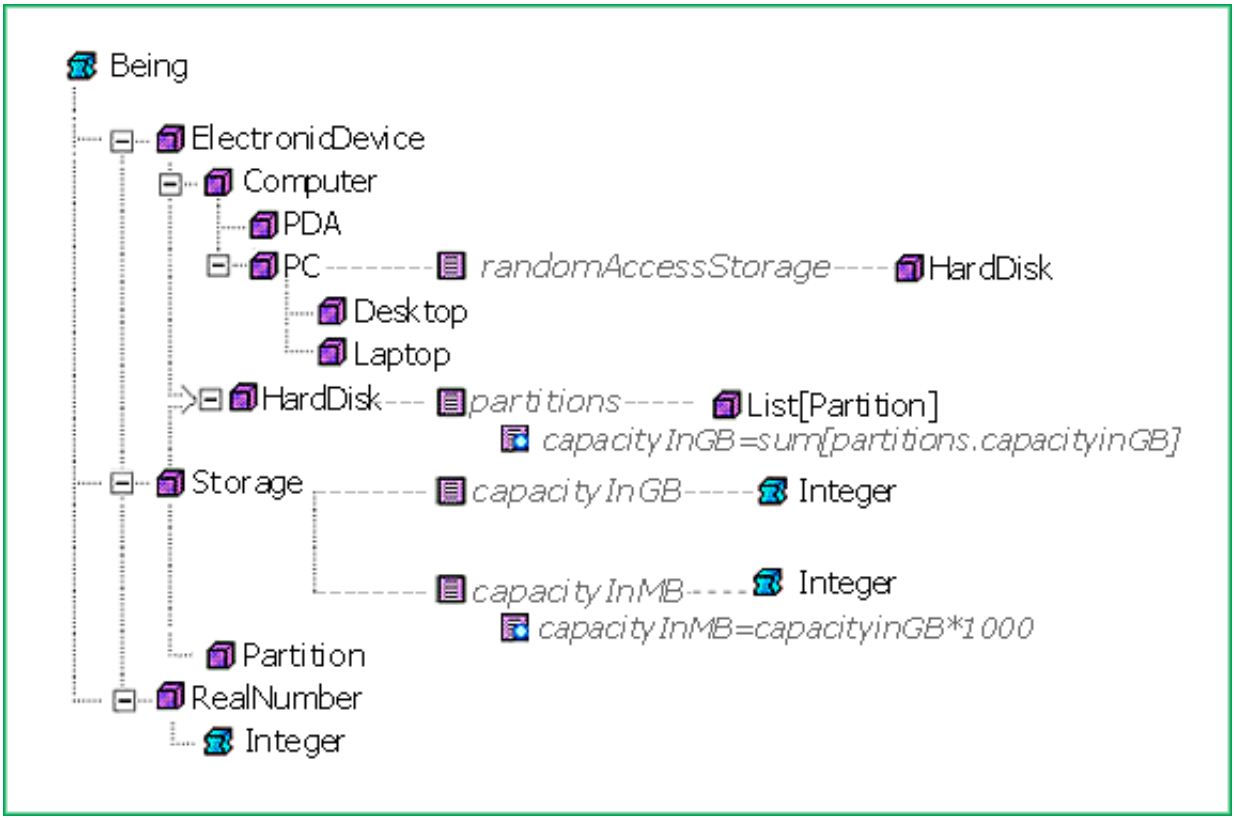

Fig. 4: An Example Information Model

Classes

The fundamental unit in ontology is the **class**, a set of real-world entities, which hold in common certain defined relationships to other entities.

For example, we might define the class `DesktopComputer` to represent the set of real-world Desktop Computers. Although ontological classes are reminiscent of the classes of Object Oriented programming, it is important to understand that ontological classes represent already-existing real-world entities, rather than serving as a factory of software constructs, as in object-oriented programming.

A class is a set of **instances**. An instance is a real-world entities that is declared to be a member of the class and conforms to the class definition. For example, the desktop computer with the DNS address `joshua1.unicorn.com` is an instance of `DesktopComputer`.

Inheritance

Classes can be related in an **inheritance** relationship (see **Fig. 5**). A subclass is a subset of the superclass, and every statement that can be made about a superclass's instances can also be made about a subclass's instances

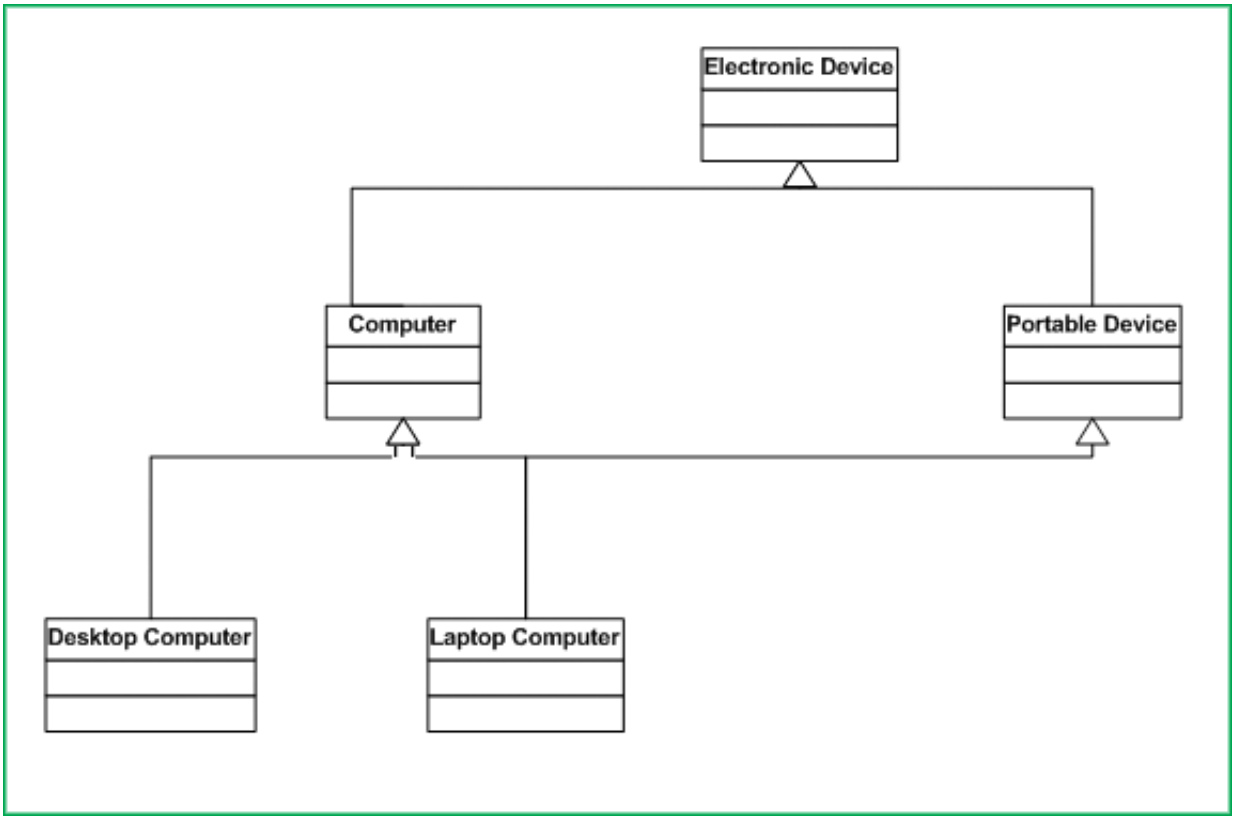

Fig. 5: Inheritance

Properties

A class can have **properties** that relate it to other classes. A property is a function in the mathematical sense, relating each instance of the source class to an instance of the target class (or more generally, to a set of instances of the target class).

```
Class A ---property---> Class B
```

For example, a Desktop Computer has a monitor

```
Desktop Computer ---displayedBy ---> Monitor
```

Constraints

To indicate restrictions on the possible values for properties, a **constraint** can be defined.

- A cardinality constraint can limit the number of values of a multi-valued (set-valued) property. For example, a constraint can state that the target of the property called `central_processing_units` of the class `Computer` is a set with cardinality greater than 0 and less than or equal to 64. In other words, each computer has from 1 to 64 CPUs.
- A constraint may enumerate permitted values for a property. For example, only the values `"Alaska", "Alabama", "Arkansas", ..., "Wyoming"` are allowed for values of the property `name` of the class `State`.
- A constraint may express a mathematical expression or any other expression of a relationship (written in an expression language). For example, for properties `capacityInGB` and `capacityInMB` of the class `MemoryCapacity`, `capacityInGB = capacityInMB/1000`.

- A constraint may relate two properties as mutual inverses: as in properties `displays` and `displayedBy` for classes `Desktop Computer` and `Monitor`.

Ontology Initiatives

This approach is being implemented with widely accepted standards. Because the vision of the Semantic Web requires interoperability between Web Services with different output and input interfaces (see **[TBL01]**), organizations such as the W3C are now developing standards for sharing ontologies.

- OWL (Web Ontology Language) , formerly known as DAML+OIL (DARPA Agent Markup Language/ Ontology Inference Language) , is the leading XML-serialization format for ontologies. (See **[OWL02]** .)
- RDF (Resource Definition Framework) is a format for expressing triplets to allow semantic relationships to be expressed. A higher-level standard, RDF/S (Resource Definition Framework/Schema) provides an alternative for serialization of ontologies.

Semantic Mapping

The information model expresses an understanding of the real world. With this powerful tool for integration in place, the next step is to attach the syntactic elements of the schemas to these ontological concepts, thus expressing the semantics of the schemas. This can be partially automated, by arbitrarily declaring an ontological concept per schema component; for example, an ontological class `DesktopComputer` can be automatically produced based on a schema complex type `desktopComputer`. After this step, a human intervention information modeler continues the mapping process.

Semantic mapping does require some up-front effort, which is not negligible when schemas were not developed in coordination with the information model. However, even in the common development methods, semantic analysis cannot be neglected, but the semantic analysis is done informally and must be repeated for each development effort. With formal mapping, the semantic interpretation, once encoded formally, need never be done again. Even before the stage of automated XSLT-generation the formal semantic mapping is of great value: the schemas' semantics are precisely and formally encoded, removing uncertainties that are always present when syntax, but not semantics, are defined. Moreover, since mapping must be done once per schema, the complexity rises linearly with the number of schemas, rather than quadratically, as when each schema must be treated for each XSLT-development effort.

Correspondence between XSD Components and Ontological Concepts

In the mapping process, the modeler maps:

- An XSD **complex/simple type** to an ontological **class**.
- An XSD **element/attribute** to an ontological **property**.
- An XSD **inheritance by extension** to ontological **inheritance**.
- An XSD **inheritance by restriction** to ontological **constraints**.

Mapping Example

Returning to our earlier example of schemas for computers:

```
<xs:element name="harddisk">
 <xs:complexType>
```

```
   <xs:sequence>
    <xs:element ref="capacity"/>
    </xs:sequence>
  </xs:complexType>
 </xs:element>
```

In this case, the complex type for `harddisk` is mapped to the class `HardDisk`. If we imagine other schemas that express a `HardDisk` with complex types called `HardDrive`, `FixedDrive`, `HD`, or `MainStorage`, the value of mapping to an information model becomes clear.

The element referred to here as `capacity` is mapped to the ontological property `capacityInGB`. Here, the ambiguity of dual units of measurement, gigabytes and megabytes (and further ambiguity over the use of quasi-decimal gigabytes as against true-binary gibibyte is resolved through the ontological mapping. Moreover, ontological constraints such as `capacityInMB = capacityInGB * 1000` explain the numeric values and allow the semantics software to readily convert between all these units of measurement.

Transformation

The next step is an automatic one. Using the model and the mappings, the semantic hub can generate XSLT to transform from one schema to the other on demand.

The Algorithm

The transformation algorithm searches for components in the source schema that can be used to produce each component of the target schema. In the simplest case, if an element or attribute in the source is mapped to the same ontological property as an element or attribute in the target, then value of the source element is copied in the XSLT directly to the target.

Even when source and target are mapped to different elements, a transformation may still be possible. Inheritance by extension allows the use of any type in place of its supertypes. Also, if a constraint connects these two properties (the property mapped to the source and the property mapped to the target) then it can be used to generate a transformation.

In our computer example, constraints state that `capacityInMB = capacityInGB * 1000`. Different XML documents express harddrive's capacities in MB or GB, but since these differences are mapped to the information model. Since the information model contains these constraints, the semantic hub can generate the transformation.

Furthermore, where properties do not provide sufficient information to generate a transformation, properties can be chained; in mathematical terms, a compound function can be built from a sequence of functions. For example, one XML document may express the capacity of a harddrive (and the mappings indicate these semantics). Another XML document expresses the capacities of each of the partitions. The class `HardDisk` has a set-valued property `partitions`, and `Partition` has a property `capacity`. A constraint tells us that the sum of the capacities of a harddrive's partitions equals the total capacity of the harddrive. Thus, the compound property `partitions --> capacity` of a `HardDisk` can be converted into the property `capacity` of that `HardDisk`.

In some cases, multiple source elements can be transformed into single target elements. For example, if a computer has multiple storage devices (not shown in the model above), their capacities can be added to determine total capacity.

When some information needed for the target document is simply not available in the source document, the semantic hub creates XSLT that leaves these elements unfilled.

This algorithm can generate all possible XSLT to convert from the source to target, finding the most efficient form of the transformation. This algorithm goes beyond simple identification of equivalence classes (schema types mapped to the same ontological concept) to identify schema types that can be transformed using constraints and inheritance.

An Example

To return to our example, not all of the information is possible to fill all the elements in each of the six possible transformations. For example, schema 2 separately states the capacity of each of the hard disk partitions, whereas schemas 1 and 3 give the total hard disk capacity. The algorithm, however, generates XSLT that fills XML elements for which the information is available. For example, the following XSLT will convert schema 3 to schema 1.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
 <xsl:template match="/">
  <xsl:apply-templates select="unit[1]"/>
 </xsl:template>
 <xsl:template match="unit">
 <computer>
  <harddisk>
   <capacity>
     <xsl:value-of select="floor(storage/text() div 1000)"/>
   </capacity>
  </harddisk>
 </computer>
 </xsl:template>
</xsl:stylesheet>
```

Deployment

The final stage is deployment. Just like manually developed transformations, the auto-generated transformations can be deployed into an XSLT engine within an EAI message broker or a Semantic Web server, then executed as needed. (See **[JF02]** .)

A semantic hub can significantly simplify XSLT-development. Moreover, once schemas are mapped, new possibilities open up for XSLT-generation at run-time. For example, a message broker can request a transformation for a given source and target schema, and as long as these schemas have been mapped, the integration of two applications can occur automatically at runtime. Likewise, a Semantic Web server can get transformation code to integrate semantic information for mapped schemas, even though the schemas were developed separately. Since total coordination cannot be expected, no two systems will have identical XML schemas for their expected transmissions, and so transformation will always be necessary. Dynamic generation of the relevant XSLT, based on an information model, can bring us a step closer to total runtime integration of enterprise applications, and to the future reality of the Semantic Web.

Use Case

A use case may help demonstrate a typical business application of the semantic hub. An international

manufacturing company discovers that it is incurring inventory excesses and stock shortages due to discrepancies between the different systems used to plan its various manufacturing processes and flow of goods. The company cannot monitor these discrepancies, as the dissimilar systems transmit data in entirely different XML schemas.

One system manages real-time operations while another manages weekly planning across time zones and rollover from the previous week's results. Another applications summarize monthly data across geographical areas, and a number of other applications vary similarly. The information in all these systems must be coordinated and accessible in a uniform format. Moreover, the systems must interact with other systems that act differently at different times, depending on the shipping status or other knowledge about a product and its progress in the production line. Business logic embedded in one process must be visible to the other processes.

The company uses a semantic hub to create an information model for its manufacturing planning, which is mapped to and from each of the data systems. Besides immediately exposing hidden discrepancies and improving the quality of the company's information, the semantic hub now allows data to be easily transformed from one system to another, with no human intervention beyond the initial modeling and mapping work.

The resulting XSLT code is of high quality, since it precisely reflects the underlying semantics of the XSD's in the simplest possible way, and when schemas are changed or new schemas are added, developing new XSLT is a simple matter of generating it automatically.

Conclusion: A Semantic Hub

By giving semantics to XSD-elements, we can significantly reduce the complexity of developing XSLT. In addition to the design-time simplicity, the runtime generation of XSLT for mapped schemas is an important advance in the direction of truly dynamic integration.

Acknowledgements

I thank Joram Borenstein and Zvi Schreiber for their valuable comments.

Bibliography

[TBL01]
Tim Berners-Lee, "The Semantic Web," *Scientific American*, May 2001 (also at www.sciam.com/2001/0501issue/0501berners-lee.html).

[JF02]
Joshua Fox, "Central Information Models for Data Transformation," *EAI Journal*, Fall 2002.

[CL02]
Cameron Laird, "XSLT Powers a New Wave of Web Applications", *Linux Journal* 95, Sep. 2002 (also at http://www.linuxjournal.com/article.php?sid=5622).

[MF02]
Martin Fowler, "Patterns of Enterprise Application Architecture", forthcoming in 2002 (also at http://www.martinfowler.com/isa/transformView.html).

[OWL02]
OWL Web Ontology Language 1.0 Reference, ed. Mike Dean *et al.*, (http://www.daml.org/2002/06/webont/owl-ref-proposed)