

# Ontologizing XML Using Mediation Patterns

Karthick Sankarachary

karthicks@email.com

**Abstract.** A prerequisite for carrying out semantic actions on an XML-based model is representing the defining meta-model in an ontology language. In this paper, we explore an approach towards expressing the meaning of an XML-based model, that applies the process outlined in the WSMX data mediation deliverable to not only specify abstract mappings which define its domain-specific ontology, but in the process also reveal an upper ontology for XML-based languages. In particular, we analyze each of the constructs of XML schema in turn and suggest how they might be mapped into the constructs of a WSML ontology. Structural constructs such as definitions and declarations in XML turn into concepts and attributes in WSML. Logical constructs such as content models and derivation in XML turn into axioms and relations in WSML. At the meta-model layer, the dynamism of the mapping definition calls for the inclusion of a WSML-specific upper ontology which formalizes the concept of concepts, and that is presented here. At the model layer, we specify a mapping that transforms an XML instance into a WSML ontology which is in keeping with the ontology of the corresponding XML schema. In closing, we discuss potential applications to industry verticals.

## 1 Introduction

To make perfect sense of an XML instance on a syntactic level, one refers to its XML schema. By the same token, to make perfect sense of a WSML instance on a semantic level, one needs its WSML definition (we assume, without loss of generality, that the language as described in [WSML] denotes a generic ontology.) However, the WSML definition in and of itself may not be self-describing enough to make sense in a vacuum. Specifically, a formal codification of the semantics of XML schema is called for, especially if we want to facilitate fully-automated reasoning. Typically, the semantics of XML schemas is implicitly built into an XML processor. However, when it comes to ontology languages, we might as well leverage its own constructs to express the semantics of XML schemas, which will serve to reduce the complexity of the ontology processing engine.

In this paper, we describe a pair of language-neutral mapping specifications that translates XML instances and schemas thereof into various ontology languages, including but not limited to WSML. Furthermore, we construct an upper ontology to represent the meta-meta-model layer of XML by applying those mappings to the XML schema for schemas (XSS). It will not only serve as ground rules that apply across the board of XML, but also allow for derivation of higher order theories such as XML compatibility, which might not be possible otherwise.

## 2 Mapping Language Requirements

Ideally, one would like to be able to specify a mapping between two or more models in as intuitive a way as possible. Specifically, it helps if the language used is axiomatized, modular, declarative, and most importantly, inductive. That would allow the designer of the mapping to take a bottom-up approach and define one context-free piece of mapping at a time, leaving it to the language to put it all together. Also, one would like to be able to execute the mapping in an ontology engine of one's choosing. This calls for an abstract mapping process which, as and when required, may be grounded to various concrete ontologies.

In that regard, the Abstract Mapping Language (AML) described in [WSMX] appears to fit the bill. In particular, its expressive power lets you define mappings between different kinds of entities in the ontology, thanks to its adoption of the mediation patterns described in [SEKT]. All in all, it is compact yet complete.

## 3 Serialization of XML to WSML

The AML makes an assumption about the source it takes as input and target that it outputs. Both of them must be instances of an ontology language, preferably WSML. However, in our use case scenario, the source takes the form of an XML document, at least to begin with. Therefore, as a necessary precursor to mapping, the XML document will first be serialized into a WSML ontology. At this stage, we avoid trying to be overly smart and do not define any semantics of XML at all. Instead, the goal is to generate an intermediate ontology which strictly treats the XML document as nothing more than a well-formed XML Information Set [XIS]. To that end, we lay down these rules of serialization<sup>1</sup> that,

1. Define the *PartOf* view of the WSML ontology as follows:
  - (a) An XML element is mapped to a WSML concept.
  - (b) An XML child element is mapped to a WSML attribute of the concept corresponding to the XML parent element. It has an inferring type of the WSML concept corresponding to the XML child element.
  - (c) An XML attribute is mapped to a WSML attribute of the concept corresponding to the XML parent element, with a string constraining type.
2. Define the *InstanceOf* view of the ontology as follows:
  - (a) An XML element is mapped to a WSML instance of the corresponding concept, whose ID is a globally unique, possibly anonymous, IRI.
  - (b) An XML child element (attribute) is mapped to a WSML attribute value of the instance corresponding to the parent XML element, whose value set includes the WSML instance (literal string value) corresponding to the XML child element (attribute).
3. Define the IRI naming convention as follows:
  - (a) The WSML concept's IRI is derived from the XML element's QName, after capitalizing the local name, but retaining the namespace URI.
  - (b) The WSML attribute's IRI is derived from the XML element's or attribute's QName, after capitalizing the local name and appending "has".

---

<sup>1</sup> Available from <http://purl.oclc.org/net/wsmo/serializers/xml-to-wsml.xsl>.

## 4 The Concept of Concepts

Typically, one has a priori knowledge of the target ontology at the time one designs a ontology-based data mediation solution. However, at times one may not be privy to the target, perhaps because the IDs of the target concepts are to be derived from dynamic values present in the source ontology. Unfortunately, WSML does not inherently allow dynamic definition of concepts and attributes because, unlike XML or RDF, it does not define its meta-model using its own constructs. So, what is required here is an ontology that defines the concept of concepts, a la the XML or RDF schema for schemas. For starters, we define concepts to represent the WSML entities of concept, attribute, relation and axiom. Then, we define the semantics of those entities, which for the most part, are taken from section 8.1 (Mapping Conceptual Syntax to Logical Expression Syntax) of [WSML]. The resulting ontology formally defines the concept of concepts, and is heretoforth referred to as the WSML Uppermost Ontology (WUO)<sup>2</sup>. The table below illustrates how WSML and WUO may be used interchangeably.

WSML Representation	WUO Representation
concept _#1 _#2 impliesType _#3 _#4 ofType _#5	instance _#6 memberOf Concept hasId hasValue _#1 hasAttribute hasValue {_#7, _#8} instance _#7 memberOf Attribute hasId hasValue _#2 hasImpliesType hasValue _#9 instance _#8 memberOf Attribute hasId hasValue _#4 hasOfType hasValue _#5 instance _#9 memberOf Concept hasId hasValue _#3
instance _#10 memberOf _#1 _#2 hasValue _#10	instance _#12 memberOf Instance hasType hasValue _#6 hasAttributeValue hasValue _#13 instance _#13 memberOf AttributeValue hasAttribute hasValue _#2 hasValues hasValue _#10

## 5 The Semantics of XML Schemas

In this section, we specify a meta-model (M2) mapping to transform the XML schema abstract data model into an ontology that embodies its meaning. At design-time, we define the content of the source in terms of XSS and the target in terms of WUO. We picked XSS, serialized into WSML, as it is a good representative sample of all XML schemas, in that it uses all of the constructs that it

<sup>2</sup> Available from <http://purl.oclc.org/net/wsmo/ontologies/wuo-wsml>.

defines and declares names that are globally unique. To allow for dynamism in the target, we elected to represent it in WUO, coupled with an ontology for XSS denoted as the XML Uppermost Ontology (XUO)<sup>3</sup>. At first blush, this may seem like the proverbial chicken-and-egg problem, but we get around that by taking the iterative approach. By running XSS through this mapping, we may extend the XUO further, which in turn could be used to enrich the mapping further.

It's a given that an XML instance comprises mostly XIS items and has few or none of the XSD entities. Consequently, to ontologize it such that it is aligned with its defining M2 ontology, simply entails its serialization (followed by a few minor structural changes.) It's well worth noting that the process of serialization maps XML's structural constructs (such as definitions and declarations) to WSML's structural constructs (such as concepts and attributes), and does not really depend on any of WSML's logical constructs (such as relations and axioms.) Given that, XML's structural constructs will not be mapped to WSML's logical constructs and XML's logical constructs will not be mapped to WSML's structural constructs, otherwise we run the risk of confusing one with the other.

The stage is now set to consider the M2 mappings, each of which takes on a certain logical component of XML schema, and which when iteratively applied to the source leads to the target. The mapping template comprises description elements of a logical *Name*, a focal *Feature*, a defining *Intent*, a prerequisite *Context* and an after *Effect*. For details, please refer to the solution scripted in AML<sup>4</sup>, which comes with comments for clarity. The resulting target, when redefined in terms of WSML<sup>5</sup>, is the end result of the M2 mapping process.

## 5.1 The Ground Rules

<i>Name: Primitive Types</i>	<i>Feature: Built-in Datatype Hierarchy</i>
<i>Intent: Bridge the gap between the ground facts of XML and WSML. Use direct counterparts, wherever possible. The QName of an XML type definition or declaration turns into the IRI of the corresponding WSML entity.</i>	
<i>Context: The primitives in the source are XML data types, typically strings.</i>	
<i>Effects: Primitives are WSML based. IRI naming conventions are in force.</i>	

## 5.2 The Basic Concepts

<i>Name: Type Definitions</i>	<i>Feature: Simple, Complex, Schema Type Definitions</i>
<i>Intent: The schema component turns into an ontology whose ID takes on the value of the target namespace. A simple or complex type component turns into a concept, whose ID is the type's name, if present, or else it's anonymous IRI.</i>	
<i>Context: The type has a name. If not, it's instance's ID acts as its name.</i>	
<i>Effects: A concept whose ID is the name of the type is generated.</i>	
<i>Name: Item Declarations</i>	<i>Feature: Element &amp; Attribute Declarations</i>

<sup>3</sup> Available from <http://purl.oclc.org/net/wsmo/ontologies/xuo-wsml>.

<sup>4</sup> Available from <http://purl.oclc.org/net/wsmo/mappings/complete-aml>.

<sup>5</sup> That is, the WUO instances are converted into WSML concepts and relations.

*Intent:* Define an XML element (attribute) declaration as a WSML attribute.  
*Context:* The XML element (attribute) has a name or references a global one.  
*Effects:* The WSML ID is derived from the XML name. Local names are added to the enclosing complex type's concept, and global names to the range of schemaTop, which is an instance of Group. The inferring type of the WSML attribute is the concept representing the (named or anonymous) XML type.

*Name:* *Occurrence Constraints*      *Feature:* *Element & Attribute Occurrences*

*Intent:* Map XML occurrence constraints to WSML attribute cardinalities.  
*Context:* Item declarations may or may not have occurrence constraints.  
*Effects:* In the case of an XML element, its min and max occurs properties are used to derive the min and max cardinalities of the corresponding WSML attribute. In the case of an XML attribute, its use property is employed.

*Name:* *Annotations*      *Feature:* *Annotation User Information Items*

*Intent:* Define XML annotations as WSML non-functional properties.  
*Context:* The annotation describes an XML definition, declaration or instance.  
*Effects:* The annotation is attached to the corresponding WSML concept, attribute or instance. To be precise, the documentation property of the XML annotation goes into the description property of the WSML entity.

*Name:* *Content Models*      *Feature:* *Element & Attribute Model Groups*

*Intent:* Define the XML Model Group using the XUO Group concept.  
*Context:* Named groups are global. Constraints or references thereof are local.  
*Effects:* The XML group, all, choice, sequence and attributeGroup models are represented using the XUO Group concept, whose domain is taken to be the concept denoting the complex type in which the model is defined, and whose range has its attributes populated with the XML elements contained in the model group. Model constraints are captured as axioms defined on it. Occurrence constraints trickle down to the attributes for each particle in the group.

*Name:* *Nil Values*      *Feature:* *Signalling Empty Content In Complex Types*

*Intent:* Express nillability of elements using the XUO Nillable relation.  
*Context:* The element declaration has a "xsi:nil" attribute set to true.  
*Effects:* This comes into play when applying the M1 mapping on an XML element whose "xsi:nil" is set to true, which then implicates its non-existence.

*Name:* *Simple Type Derivation*      *Feature:* *Simple Type Variety & Facets*

*Intent:* Define list, union and facet restrictions on simple types as relations.  
*Context:* The facet has a value and the list and member have base types.  
*Effects:* A list type constrains the concept to take on values that is a comma-separated list of atomic values of the item type. A union type constrains it to take on values whose type is one of those in the member type set. A facet constrains the concept to take on values that belong in the facet's value space.

### 5.3 Modular Schemas

*Name:* *Multiple Documents*      *Feature:* *Schema Include, Import & Redefine*

*Intent:* Map XML import to WSML import. Merge included/redefined XSDs.

<i>Context:</i> The include, import and redefine items refer to an external schema location, which in the case of an import may have a different target namespace.
<i>Effects:</i> In the case of include and redefine items, all of the concepts and relations representing the referenced schema become part of containing ontology. In the case of an import item, an importsOntology property is generated.
<i>Name:</i> <i>Complex Type Derivation</i> <i>Feature:</i> <i>Base Type Restriction &amp; Extension</i>
<i>Intent:</i> Define the restricted or extended type as a sub-concept of the base type. A type that has simpleContent is forbidden to have child elements or attributes.
<i>Context:</i> Only complex types may be extended, but any type may be restricted.
<i>Effects:</i> The concepts denoting extended types are allowed to contain attributes not present in the base type, whereas those for restricted types are constrained such that its attributes take on only those values permitted by the base type.
<i>Name:</i> Substitution Groups <i>Feature:</i> Substitution Group Affiliation
<i>Intent:</i> Allow certain head elements to be substituted by other elements.
<i>Context:</i> The head element, which may be abstract, must be globally declared.
<i>Effects:</i> Concepts having an attribute that denotes the head element may have instances in which that attribute actually denotes the substituting element.
<i>Name:</i> <i>Controlling Derivation</i> <i>Feature:</i> <i>Final &amp; Block Properties</i>
<i>Intent:</i> Prohibit derivation of final types and substitution of blocked types.
<i>Context:</i> The complex or simple type has a final or block property.
<i>Effects:</i> A final (block) property on a type prevents derivation (substitution) of the concept by restriction, extension or both. Facets may be similarly fixed.

#### 5.4 Primary and Foreign Keys

<i>Name:</i> <i>Specifying Uniqueness</i> <i>Feature:</i> <i>Identity-Constraint Definitions</i>
<i>Intent:</i> Define unique, key and keyref constraints in terms of a XUO Group.
<i>Context:</i> The constraints select a key base, which go into the Group's range.
<i>Effects:</i> The field nodes are checked to see if they are unique, null and/or referentially integral, within the scope of the set of selected elements.
<i>Name:</i> <i>Any Items</i> <i>Feature:</i> <i>Wildcard Schema Components</i>
<i>Intent:</i> Define the any wildcard properties in terms of a AnyAttribute relation.
<i>Context:</i> The wildcard declarations are part of a complex type definition.
<i>Effects:</i> Instances of the enclosing concept may have attributes of any type.

## 6 The Semantics of XML Instances

In this section, we specify a model (M1) mapping which transforms an XML instance into a WSML instance, in keeping with its M2 ontology. Firstly, we serialize the XML instance into WSML. Secondly, we realign the type of its concepts by replacing declared names with their constraining types, and recasting primitive attribute values to specific simple types. Thirdly, we handle [XSI] attributes that [XSD] defines for use in an XML instance, by capturing those runtime type information items as axioms. Note that both the source and the target of this mapping are ontology instances. Furthermore, both ontologies are

dynamic domain-specific languages and not known a priori at design-time. As in the case of the M2 mapping, we deal with this by redefining the serialized source in terms of WUO just before the map, and redefining the resulting target in terms of WSML right after the map. The end result of the M1 mapping execution conforms to the end result of the corresponding M2 mapping execution.

*Name:* Instance Alignment | *Feature:* M2 Alignment & XSI Attributes

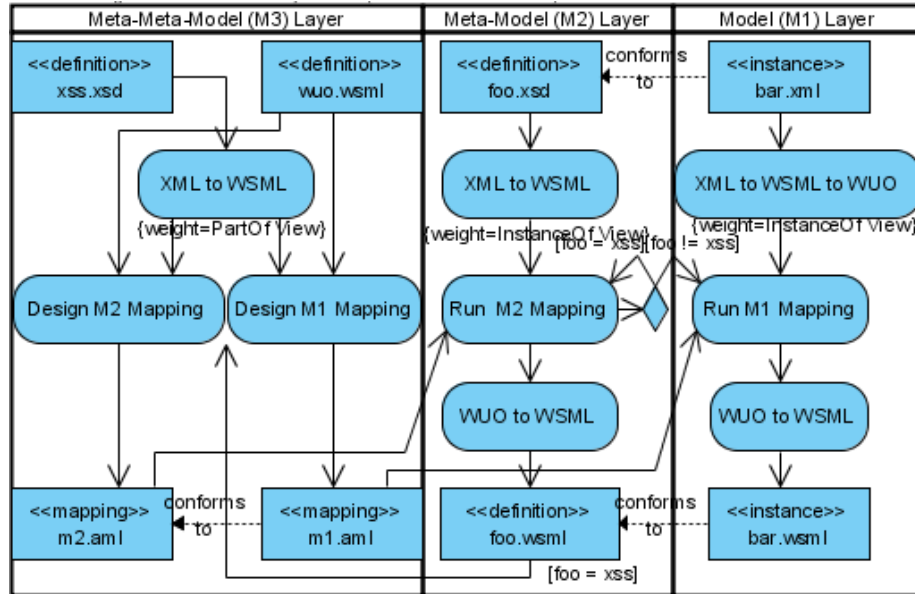
*Intent:* Align concept IDs and attribute values. Transform the XSI nil, type and schemaLocation attributes on XML elements into appropriate axioms.

*Context:* The meta-model must allow for the presence of XSI attributes.

*Effects:* The WSML concepts are aligned and XSI constraints are applied.

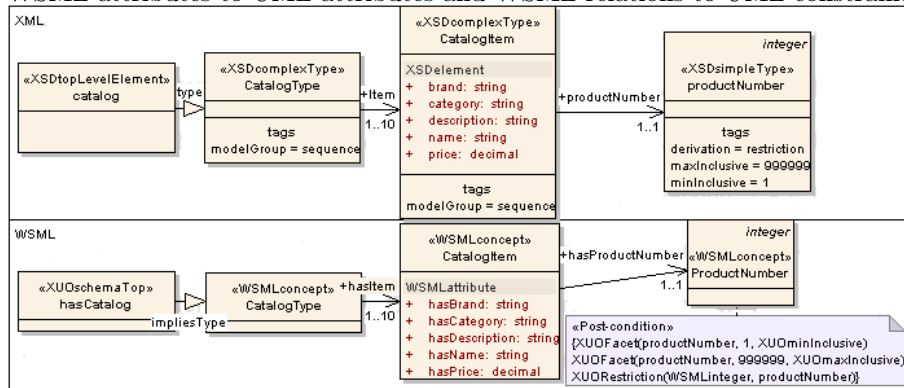
## 7 The Big Picture

Let's recapitulate the key activities of the design-time and run-time phases, which are partitioned into three swim lanes, one for each layer of abstraction.



In the M3 swim lane, we define the M2 and M1 mappings using the *PartOf* view of XSS, serialized into WSML, along with the WUO ontology. In the M2 swim lane, we execute the M2 mapping on the *InstanceOf* view of an XML schema, serialized into WSML, and finally convert the resulting WUO to WSML ontology. In the M1 swim lane, we execute the M1 mapping on the *InstanceOf* view of an XML instance, serialized into WSML and then WUO, and finally convert the resulting WUO to WSML ontology. Prior to running the M1 mapping on an XML instance, we import the WUO ontology of the corresponding XML schema. Optionally, before we run the M2 mapping on an XML schema, we may import the WUO ontology of XSS. Optionally, to iteratively enrich the mappings, the WSML ontology of XSS may be fed back into the design actions.

To illustrate this process, let us consider a sample catalog of products<sup>6</sup>. If the XML swim lane depicts the UML profile for its XSD, then the WSML swim lane is the UML profile for its WSML, which maps WSML concepts to UML classes, WSML attributes to UML attributes and WSML relations to UML constraints.



In case you're wondering why the mappings are two-way, it is to allow the lowering of XUO-aligned ontologies back into XML. In case you're wondering what an XUO-aligned ontology is, it is one that either results from the mappings described here or an alignment with XUO using a matching process that leverages and conforms to its structure and logic. By using two-way mappings, we have implicitly specified an inverse mapping from XUO-aligned to WSML-serialized ontologies. The latter may be lowered back into XML by applying the inverse of the function that is the XML to WSML serializer. That concludes this description of a process that lets you round-trip from XML to WSML and back.

## 8 Conclusions

In this paper, we have given serious consideration to using AML as a means for interpreting the meaning of XML, which can be used for anything from specifying domain-specific naming and design rules such as those of FpML or OAGIS, to ensuring that XML schemas such as those of RosettaNet when constrained by partners remain valid. The abstraction of AML not only simplifies the mapping specification, but also allows for it to be grounded to multiple ontology platforms. We feel that it holds a lot of promise, and hope this paper bears that theory out.

## References

- [XIS] XML Information Set (2<sup>nd</sup> Edition). W3C Recommendation. February 2004.
- [XSD] XML Schema Part 1 & 2: Structures & Datatypes 2<sup>nd</sup> Edition. October 2004.
- [WSML] Jos de Bruijn, editor. The Web Service Modeling Language WSML. D16.1v0.2
- [WSMX] Adrian Mocan, Emilia Cimpian. WSMX Data Mediation. D13.3v0.2.
- [SEKT] Jos de Bruijn, et al. Ontology Mediation Patterns Library V2. D4.3.2. SEKT.

<sup>6</sup> All artifacts available under <http://purl.oclc.org/net/wsmo/samples/wsiscm/>.