XML, The Model Driven Architecture, and RDF

Uche **Ogbuji** <uche.ogbuji@fourthought.com>

Abstract

These are exciting times for application development: in many ways there is a fundamental shift in how developer perceive of applications. And XML is one of the main axes of this re-conception. XML promises to give a powerful framework for dealing with semi-structured metadata in applications. At the same time, the Object Management Group (OMG) is shifting its world view to a focus on the abstract models that underlie computer applications. This new approach to development: the Model-Driven Architecture (MDA) promises to build a bridge from successful design technologies such as UML to the very concepts that underpin the uses we have for computers. The Resource Description Framework (RDF) is another technology positioned to change the way concepts are synthesized into applications, both in localized data-driven systems and in the future Semantic Web. This presentation will discuss how these three technologies come together to provide powerful patterns for solid and maintainable software design.

Table of Contents

Modeling is one of the most important considerations of computer systems development. It is the process of finding the most effective representation of real-world concepts in the computer space for purposes of a software project. In most cases familiar with developers, the process of modeling involves aggressively cutting down the context in which these real-world concepts are considered, in order to allow the models to follow practicalities imposed by the expected inomemation. So, for instance, Object-oriented development might impose a particular set of distortions of the isea of an "employee" in order to meet rules and expectations driven by the resulting object-oriented language. Relational modeling (entity-relationship modeling) imposes its own, quite different set of distortions.

XML has come along bringing to development a novel new angle on modeling: the modeling of real world concepts according to how a human might express them in documents. And RDF brings an angle on modeling of how one might represent the concepts as sets of nodes, which are identified using URIs, and which are connected to other nodes with labeled, directed edges, to form a graph structure.

In parallel developments to the arrival of XML and RDF on the modeling scene, the Object Management Group (MDA), which defines the tool with which most developers now associate modeling representation: UML, has brought a new perspective to the role of modeling in development. The OMG Acknowledges that this is an age where systems integration often occurs at Internet scope, and where "virtual enterprises" often encompass the roles of multiple development with completely different chains of responsibility (for instance, each partner in a commerce supply-chain has to synchronize certain concepts according to contract, but develops separate software for doing this). Building on lessons from the pioneers of development within this expanded horizon, the OMG has developed the Model-Driven Architecture (MDA) as the method for projects in this new generation.

The MDA separates the fundamental understanding of concepts in development projects from the respresentation of these concepts for particular implementations technologies. This means even separating this abstract modeling layer from the object-oriented orthodoxy, which has been the bedrock of the OMG in the first place. Basically, the OMG is admitting that no model tainted with implementation considerations can survive the disjoint between multiple development groups, or different processing environments, as are common in Internet integration and establishing vertical alignment. The UML, despite its most common use in object-oriented modeling, is designed

to be a much broader modeling language, and has been put to the test in at least one very prominent Internet-scale model that requires far more sophistication that mere objects do not approach. The Common Information Model (CIM), which models concepts related to the management of technological assets, is expressed in a UML that would probably be entirely alien to most developers, and this flexibility of UML is what the OMG uses to justify its being the language of the most abstract modeling under the MDA.

This paper will take a brief look at how XML, RDF and MDA come together to revolutionize modeling for software development.

1. UML and XMI

UML defines models, and models have their own information, and they have meta-information pertaining to them. Although UML defines its own meta-model for expressing is meta-information, the OMG also has the Meta-Object Facility (MOF) as a standard meta-information management standard, which evolved from needs of expressing meta-data for CORBA in platform-independent ways. This unfortunately complicated arrangement is patched over a bit by a standard correspondence between MOF and UML meta-model (there is also a mapping from MOF to IDL). Much of the flexibility that UML has gained has been in the process of aligning it better with MOF, which is defigned for general information modeling, rather than object-oriented analysis and design.

The reason for walking this tangle is that we want to examine XML Medtadata Interchange (XMI), which is OMG's standard facility for exporting MOF meta-models to XML format. Because of the strong alignment between UML and MOF, XMI has found an emergent use as an exchange mechanism, for UML models between tools, and this is how most developers know of XMI.

As an example, let us use the example from Objects by Design's interesting tutorial on converting XMI to HTML. The following is a graph of a UML model of graphics objects.

Figure 1. A UML model of grapics objects
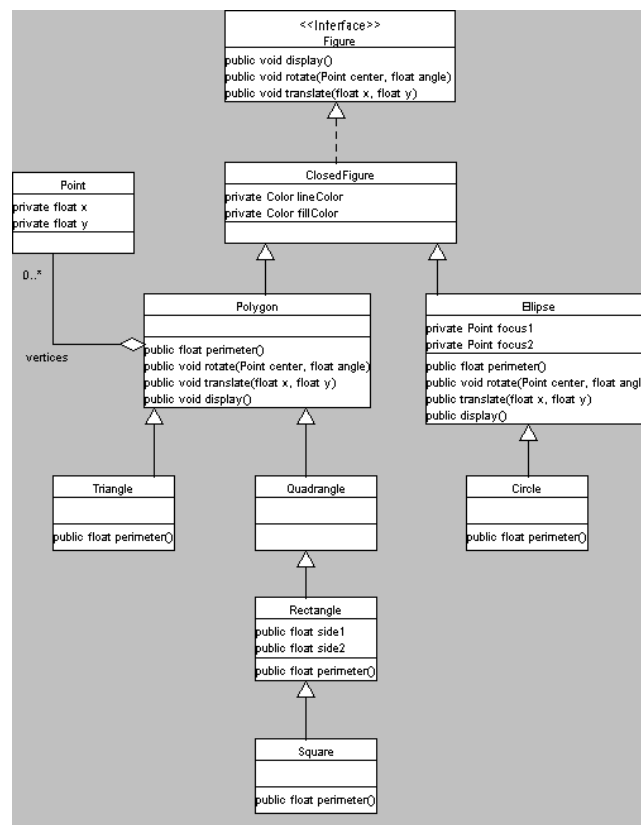
And the following listing gives an abbreviation of its relresentation in XMI.

2

```xml
<?xml version="1.0"?>
<XMI xmi.version="1.0">
  <XMI.header>
    <XMI.metamodel xmi.name="UML" xmi.version="1.3"/>
  </XMI.header>
  <XMI.content>
    <Model_Management.Model xmi.id="xmi.1"
      xmi.uuid="-106--106-25-11--7e352540:e17838870b:-7fed">
      <Foundation.Core.ModelElement.name>Graphics Editor
        Model</Foundation.Core.ModelElement.name>
      <Foundation.Core.ModelElement.isSpecification
        xmi.value="false"/>
      <Foundation.Core.GeneralizableElement.isRoot
        xmi.value="false"/>
      <Foundation.Core.GeneralizableElement.isLeaf
        xmi.value="false"/>
      <Foundation.Core.GeneralizableElement.isAbstract
        xmi.value="false"/>
      <Foundation.Core.Namespace.ownedElement>
        <Foundation.Core.Class xmi.id="xmi.2"
          xmi.uuid="-106--106-25-11--7e352540:e17838870b:-7fec">
          <Foundation.Core.ModelElement.name>Point
            </Foundation.Core.ModelElement.name>
          <Foundation.Core.ModelElement.isSpecification xmi.value="false"/>
          <Foundation.Core.GeneralizableElement.isRoot xmi.value="false"/>
          <Foundation.Core.GeneralizableElement.isLeaf xmi.value="false"/>
          <Foundation.Core.GeneralizableElement.isAbstract
            xmi.value="false"/>
          <Foundation.Core.Class.isActive xmi.value="false"/>
          <Foundation.Core.ModelElement.namespace>
            <Model_Management.Model xmi.idref="xmi.1"/>
          </Foundation.Core.ModelElement.namespace>
          <Foundation.Core.Classifier.feature>
            <Foundation.Core.Attribute xmi.id="xmi.3">
              <Foundation.Core.ModelElement.name>x
                </Foundation.Core.ModelElement.name>
              <Foundation.Core.ModelElement.visibility
                xmi.value="private"/>
              <Foundation.Core.ModelElement.isSpecification
                xmi.value="false"/>
              <Foundation.Core.Feature.owner>
                <Foundation.Core.Class xmi.idref="xmi.2"/>
              </Foundation.Core.Feature.owner>
              <Foundation.Core.StructuralFeature.type>
                <Foundation.Core.DataType xmi.idref="xmi.4"/>
              </Foundation.Core.StructuralFeature.type>
            </Foundation.Core.Attribute>
            <Foundation.Core.Attribute xmi.id="xmi.5">
              <Foundation.Core.ModelElement.name>y
                </Foundation.Core.ModelElement.name>
              <Foundation.Core.ModelElement.visibility
                xmi.value="private"/>
              <Foundation.Core.ModelElement.isSpecification
                xmi.value="false"/>
              <Foundation.Core.Feature.owner>
                <Foundation.Core.Class xmi.idref="xmi.2"/>
              </Foundation.Core.Feature.owner>
              <Foundation.Core.StructuralFeature.type>
                <Foundation.Core.DataType xmi.idref="xmi.4"/>
              </Foundation.Core.StructuralFeature.type>
            </Foundation.Core.Attribute>
```

3

```xml
            <Foundation.Core.Operation xmi.id="xmi.6">
              <Foundation.Core.ModelElement.name>translate
              </Foundation.Core.ModelElement.name>
              <Foundation.Core.ModelElement.visibility
                xmi.value="public"/>
              <Foundation.Core.ModelElement.isSpecification
                xmi.value="false"/>
              <Foundation.Core.BehavioralFeature.isQuery
                xmi.value="false"/>
              <Foundation.Core.Operation.isRoot xmi.value="false"/>
              <Foundation.Core.Operation.isLeaf xmi.value="false"/>
              <Foundation.Core.Operation.isAbstract xmi.value="false"/>
              <Foundation.Core.Feature.owner>
                <Foundation.Core.Class xmi.idref="xmi.2"/>
              </Foundation.Core.Feature.owner>
              <Foundation.Core.BehavioralFeature.parameter>
                <Foundation.Core.Parameter xmi.id="xmi.7">
                  <Foundation.Core.ModelElement.isSpecification
                    xmi.value="false"/>
                  <Foundation.Core.Parameter.kind xmi.value="return"/>
                  <Foundation.Core.Parameter.behavioralFeature>
                    <Foundation.Core.Operation xmi.idref="xmi.6"/>
                  </Foundation.Core.Parameter.behavioralFeature>
                  <Foundation.Core.Parameter.type>
                    <Foundation.Core.DataType xmi.idref="xmi.8"/>
                  </Foundation.Core.Parameter.type>
                </Foundation.Core.Parameter>
                <Foundation.Core.Parameter xmi.id="xmi.9">
                  <Foundation.Core.ModelElement.name>x
                  </Foundation.Core.ModelElement.name>
                  <Foundation.Core.ModelElement.isSpecification
                    xmi.value="false"/>
                  <Foundation.Core.Parameter.kind xmi.value="in"/>
                  <Foundation.Core.Parameter.behavioralFeature>
                    <Foundation.Core.Operation xmi.idref="xmi.6"/>
                  </Foundation.Core.Parameter.behavioralFeature>
                  <Foundation.Core.Parameter.type>
                    <Foundation.Core.DataType xmi.idref="xmi.4"/>
                  </Foundation.Core.Parameter.type>
                </Foundation.Core.Parameter>
                <Foundation.Core.Parameter xmi.id="xmi.10">
                  <Foundation.Core.ModelElement.name>y
                    </Foundation.Core.ModelElement.name>
                  <Foundation.Core.ModelElement.isSpecification
                    xmi.value="false"/>
                  <Foundation.Core.Parameter.kind xmi.value="in"/>
                  <Foundation.Core.Parameter.behavioralFeature>
                    <Foundation.Core.Operation xmi.idref="xmi.6"/>
                  </Foundation.Core.Parameter.behavioralFeature>
                  <Foundation.Core.Parameter.type>
                    <Foundation.Core.DataType xmi.idref="xmi.4"/>
                  </Foundation.Core.Parameter.type>
                </Foundation.Core.Parameter>
              </Foundation.Core.BehavioralFeature.parameter>
            </Foundation.Core.Operation>
          </Foundation.Core.Classifier.feature>
        </Foundation.Core.Class>
        <Foundation.Core.DataType xmi.id="xmi.4">
          <Foundation.Core.ModelElement.name>float
            </Foundation.Core.ModelElement.name>
          <Foundation.Core.ModelElement.isSpecification
            xmi.value="false"/>
          <Foundation.Core.GeneralizableElement.isRoot
```

```
        xmi.value="false"/>
      <Foundation.Core.GeneralizableElement.isLeaf
        xmi.value="false"/>
      <Foundation.Core.GeneralizableElement.isAbstract
        xmi.value="false"/>
      <Foundation.Core.ModelElement.namespace>
        <Model_Management.Model xmi.idref="xmi.1"/>
      </Foundation.Core.ModelElement.namespace>
    </Foundation.Core.DataType>

    [  ...  SNIP  ...  ]

    </Foundation.Core.Namespace.ownedElement>
   </Model_Management.Model>
 </XMI.content>
</XMI>
```

This listing was cut down to represent just the Point class, and the data type definition for float. Of course, the mind boggles. "All that, for one class and one data type?". Well, this is the good and the bad of XMI. It carries about all the baggage of the UML meta-model itself, which makes it a very rigorous interoperability tool, but this causes tremendous bloat. This is where simplifications from the pure XML and RDF considerations of the model in question can prove valuable, and probably more useful outside the sphere of OMG-aligned tools.

2. XML and RDF

One possible model for this in XML is represented by the following DTD potrion:

```
<!ELEMENT Polygon (Point*)>
<!ELEMENT Point>
<!ATTLIST Point
    x CDATA #REQUIRED
    y CDATA #REQUIRED
>
```

There are many issues that come up right away. Do we instead interpose a "vertices" element between the Polygon and the Points to explicitly map the "vertices" association in the UML? This translates to the good old "do I use wrapper elements around similar, multiple children" XML modeling question that has no definitive answer. In this case, I choose to keep things simple, since, after all, the goal is the most direct XML representation, and I would not like the details of the UML model to seep into the XML.

Another matter is that the x and y attributes are typed as float in the UML, and just CDATA in the XML. It is in order to bring XML and traditional modeling in more concert that much of the recent work in XML data-typing has emerged, and one way to have dealt with this is would have been to use XSD rather than DTD. However, for those of us who are skeptical of the idea of global, interoperable typing, this is just a dangerous. "int" versus "float" captures but a fraction of the nuance behind how people separate things with data types, and it is unrealisting to think we can just call a value "float" in XMI and "float" in UML and have a correspondence. This is where MOF's insistence (via XMI) to bundle all the meta-model baggage along with the model itself makes eminent sense.

One more matter (and not the last, by any means) is how we model the operations. In some cases, we could do so by writing embedded XSLT right into the model, but because XSLT is hardly amenable to every task for which we turn languages that implement UML, this is probably not a general solution.

Using a natural representation of RDF Schema to represent this model, we might have:

```
<?xml version="1.0" encoding="UTF-8"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
         xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
         xmlns:g="http://rdfinference.org/eg/mda/graphics/"
         xml:base="http://rdfinference.org/eg/mda/graphics/">

  <rdfs:Class rdf:ID="ClosedFigure"/>

  <rdfs:Class rdf:ID="Polygon">
    <rdfs:subClassOf rdf:resource="#ClosedFigure"/>
    <g:Method rdf:resource="#perimieter"/>
    <g:Method rdf:resource="#rotate"/>
    <g:Method rdf:resource="#translate"/>
    <g:Method rdf:resource="#display"/>
  </rdfs:Class>

  <rdfs:Class rdf:ID="Point"/>

  <rdfs:Property rdf:ID="vertex">
    <rdfs:domain rdf:resource="#Polygon"/>
    <rdfs:range rdf:resource="#Point"/>
  </rdfs:Property>

  <rdfs:Property rdf:ID="x">
    <rdfs:domain rdf:resource="#Point"/>
  </rdfs:Property>

  <rdfs:Property rdf:ID="y">
    <rdfs:domain rdf:resource="#Point"/>
  </rdfs:Property>

  <rdfs:Class rdf:ID="Method"/>

</rdf:RDF>
```

This example covers a bit more of the original UML model than the DTD example, to illustrate the RDFS support for inheritance. Also, in this case, it is more natural to the core technology to represent the vertices association more directly. Another approach would have been to define a single "vertices" property that points to an RDF container, but I think that these are poorly designed, and prefer to just use multiple statements.

I attempt to illustrate how methods might be specified. Since RDF can point to any URI, we could make the actual definitions of the methods, omitted here for simplicity, point to, say an on-line specification of a code module (or one on an object repository), or even a Web services request end-point. The fact that the x and y properties are private could be expressed as a custom ststement on the RDFS definition for each, to the extent that specifying this makes sense outside the object-oriented implementation.

We could have gained even more facilities for direct expression fo the UML model, such as data typing and cardinality of associations, if we used DAML+OIL rather than RDFS, but I stick to the latter for simplicity, and because, as I stated, I am a skeptic of generic data typing, anyway.

3. Conclusion

This paper took a brief look at the interplay between MDA, in the manifestation of XMI representation of UML models, XML and RDF. Theer is much more discussion and work to be done in this area. For instance, how do we convert one to the other (Objects by Design have an XMI to HTML transform that could be the basis of more general transforms to other XML-based information modeling formats). How does one formalize the differences

and issues that I mentioned in passing? In my experience, modeling has already expanded to the point where XML and general information modeling affect common decisions of developers, and this is a good thing, as it fosters the development of solutions with much more flexibility and applicability. Understanding the interplay of these various modeling systems will make this conjunction even more commodious.

Bibliography

[RDF] Uche Ogbuji, *An introduction to RDF*, http://www-106.ibm.com/developerworks/xml/library/w-rdf/

[MDAHOME] MDA home page, http://www.omg.org/mda/

[XMI] The XMI specification, http://www.omg.org/technology/documents/formal/xmi.htm

[XMIHTML] Transforming XMI to HTML, by Objects by Design, http://www.objectsbydesign.com/projects/xmi_to_html.html

Biography

Wednesday, 22 May 16.45

Uche **Ogbuji**
    Principal Consultant
    Fourthought, Inc.
    Boulder
    USA
    Email: uche.ogbuji@fourthought.com

Uche Ogbuji is a Computer Engineer, co-founder and CEO of Fourthought, Inc., a software vendor and consultancy specializing in open, standards-based XML solutions, especially as applicable to problems of knowledge management. He has worked with XML for several years, co-developing 4Suite, a open-source platform for XML and RDF applications, written in Python and C. He writes articles on XML for ITWorld and IBM developerWorks, Intel Developer Services, Application Development Trends, and elsewhere. He also speaks extensively at various conferences.

Mr. Ogbuji is a Nigerian immigrant with a B.S. in Computer Engineering from Milwaukee School of Engineering. He currently resides in Boulder, Colorado where he enjoys playing amateur soccer in the summer and snowboarding in the winter. His main interest is literature, and poetry in particular.