Language Support for Megamodel Renarration

Ralf Lämmel and Vadim Zaytsev
Table of contents

- What’s renarration?
- What’s megamodeling?
- What language support?
Table of contents

• What’s renarration?
• What’s megamodeling?
• What language support?
What’s renarration?

- The process of converting a collection of facts into a story
- Fixed plot; derived story adapted to audience.
- Inspired by natural language engineering, computer-assisted reporting and database journalism
Table of contents

• What’s renarration?
• **What’s megamodeling?**
• What language support?
A megamodel is a model of which at least some elements represent and/or refer to models or metamodels.


We embrace this notion in the broadest sense: programs are models and grammars are metamodels (or v.v.) etc.
A megamodel

Bootstrapping a C compiler written in C, by compiling it using another compiler written in machine code.

Source:
http://en.wikipedia.org/wiki/Tombstone_diagram
Figure 1. An overview of model transformation

Figure 1 summarizes the full model transformation process. A model $M_a$, conforming to a metamodel $MM_a$, is here transformed into a model $M_b$ that conforms to a metamodel $MM_b$. The transformation is defined by the model transformation model $M_t$ which itself conforms to a model transformation metamodel $MM_t$. This last metamodel, along with the $MM_a$ and $MM_b$ metamodels, has to conform to a metametamodel (such as MOF or Ecore).

3 A simple transformation example

This section introduces the transformation example that is going to be developed in the document. The aim of this first example is to introduce users with the basic concepts of the ATL programming. To this end, this example considers two similar metamodels, Author (Figure 2) and Person (Figure 3), that both encode data relative to persons.

Both metamodels are composed of a single eponym element: Author for the Author metamodel and Person for the Person metamodel. Both entities are characterized by the same couple of string properties (name and surname).

The objective is here to design an ATL transformation enabling to generate a Person model from an Author model. The transformation to be designed will have to implement the following (obvious) semantics:

- A distinct Person element is generated for each source Author element;
- The name of the generated Person has to be initialized with the name of the source Author;
- The surname of the generated Person has to be initialized with the name of the source Author.

Source:
An uber megamodel

The upper frame uses the MegaL/yEd visual notation for megamodeling. The lower frame shows some linked artifacts explained later in the paper.

Fig. 1. The linguistic architecture of a software product when displayed with the MegaL/Explorer tool.

Technology for Object/XML mapping are clearly identifiable. Consider, for example, the fact that the class generator is not described as generating 'arbitrary' Cw. Instead, a designated subset, CSharpFromXsd, is used because the generator indeed produces very regular code whose regularity helps with understanding Object/XML mapping, as we discuss later.
A MegaL megamodel
http://softlang.uni-koblenz.de/mega/

See the MegaL MODELS 2012 paper.
Table of contents

• What’s renarration?
• What’s megamodeling?
• **What language support?**
Label = "Name of the milestone"
Operator = "Addition"

+ Add declaration of an entity or a relationship
+ ...
+ ...
+ ...

- Remove declaration of an entity or a relationship
- ...
- ...
- ...
An illustrative renarration: 
*Object/XML mapping*  
*(XML data binding)*
Label = "XML schema"
Operator = "Addition"

+ XSD : Language
+ ?anXmlSchema : File
+ anXmlSchema elementOf XSD
Label = "Object model"
Operator = "Addition"

+ Java : Language
+ ?anObjectModel : File+
+ anObjectModel elementOf Java
Label = "Schema first"
Operator = "Addition"

+ anXmlSchema mapsTo anObjectModel
+ anXmlSchema correspondsTo anObjectModel
Label = "Type-level mapping"
Operator = "ZoomIn"

+ ?aTypeMapping : XSD -> Java
+ aTypeMapping(anXmlSchema) |-> anObjectModel
- anXmlSchema mapsTo anObjectModel
Label = "O/X subset"
Operator = "Restriction"

+ OxJava : Language
+ OxJava subsetOf Java
+ anObjectModel elementOf OxJava
- anObjectModel elementOf Java
Label = "XML document"
Operator = "Addition"

+ XML : Language
+ ?anXmlDocument : File
+ anXmlDocument elementOf XML
+ anXmlDocument conformsTo anXmlSchema
Label = "Object graph"
Operator = "Addition"

+ JvmGraph : Language
+ ?anObjectGraph : State
+ anObjectGraph elementOf JvmGraph
+ anObjectGraph conformsTo anObjectModel
Label = "Instance-level mapping"
Operator = "Addition"

+ ?aDeserializer : XML -> JvmGraph
+ aDeserializer(anXmlDocument) |-> anObjectGraph
Label = "Code generator"
Operator = "Addition"

+ ?anOxTechnology : Technology
+ !anOxGenerator : Technology
+ anOxGenerator partOf anOxTechnology
Label = "Mapping dependency"
Operator = "Connection"

+ anOxGenerator defines aTypeMapping
+ !aTypeMapping : XSD -> Java
- ?aTypeMapping : XSD -> Java
Label = "O/X library"
Operator = "Addition"

+ !anOxLibrary : Technology
+ anOxLibrary partOf anOxTechnology
+ ?aFragment : Fragment
+ aFragment elementOf Java
+ aFragment refersTo anOxLibrary
Label = "Deserializer dependency"
Operator = "Connection"

+ aFragment defines aDeserializer
+ !aDeserializer : XML -> JvmGraph
- ?aDeserializer : XML -> JvmGraph
Label = "JAXB"
Operator = "Instantiation"

anOxTechnology => JAXB
anOxGenerator => JAXB.xjc
anOxLibrary => JAXB.jar

No use of deltas here!
Label = "Deserializer dependency"
Operator = "Backtracking"
Label = "Beyond Java"
Operator = "Parameterization"

Java => anOopLanguage
OxJava => anOxLanguage
Label = "Taxonomy"
Operator = "Restriction"

+ OopLanguage < Language
+ ?anOopLanguage : OopLanguage
+ ?anOxLanguage : OopLanguage
- ?anOopLanguage : Language
- ?anOxLanguage : Language
Questions?
Thank you!

http://softlang.uni-koblenz.de/mega
http://softlang.uni-koblenz.de/megal-renarration/
https://github.com/avaranovich/megal