

CREATING NEW MODELING LANGUAGES

As stated, MDA does not require you to use UML for modeling. You can use other languages, as long as you supply a MOF metamodel for each of the languages. In many cases UML will be the language for specifying some but not all of the aspects of a system.

Tight Domain Focus

Some kinds of models are fundamentally different than UML models because their modeling constructs don't fit easily into any of the UML modeling paradigms. When creating a MOF metamodel to define the abstract syntax of such model modeling constructs, it often does not make sense to try to extend the UML metamodel.

For example, Figure 1¹ is a simple metamodel supporting graphs such as the graph in Figure 2.² Such graphs are not UML models. A graph metamodel defined as a heavyweight UML extension would carry the baggage of the UML metamodel and thus would have a large footprint.

On the other hand, the metamodel depicted in Figure 1 tightly focuses on the graph domain. Hence, the model that Figure 2 expresses has a much smaller footprint in a MOF repository or XMI document than it would if the metamodel were derived from the UML metamodel.

A specialized graph modeling tool that knows about these kinds of graphs would be able to support the special graph notation of Figure 2.

¹ Adapted, with permission, from [XMI], figure 5-1, p. 5-2.

² Adapted, with permission, from [XMI], figure 5-3, p. 5-4.

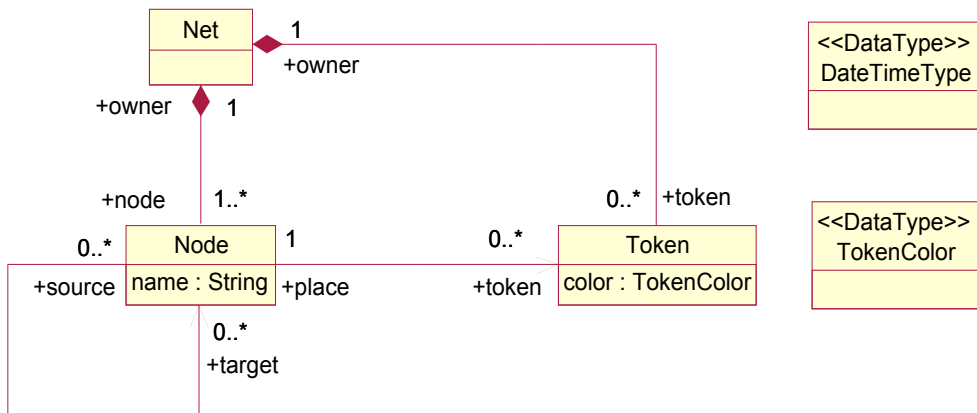


Figure 1: Simple Metamodel for Graph Modeling

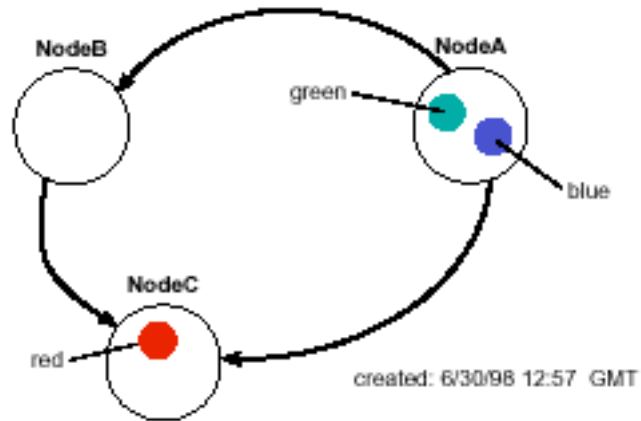


Figure 2: Sample Graph Expressed via Non-UML Notation

The Metamodel + Profile Strategy

There may be a requirement in some cases to use a generic UML tool to draw our graphs. In that case, a UML profile for graph modeling could supplement the graph metamodel. Such a profile could support rendering the graph of Figure 2 as in Figure 3, for example. This rendering is not as ideal as the rendering using the graph-specific notation, but it satisfactorily expresses the semantics. The formal definition of the profile would have to:

- Select a very small subset of the UML metamodel as its scope
- Highly constrain the use and interpretation of the UML constructs included in its scope
- Define the <<Node>> stereotype and the tagged value used to describe a Node's tokens.

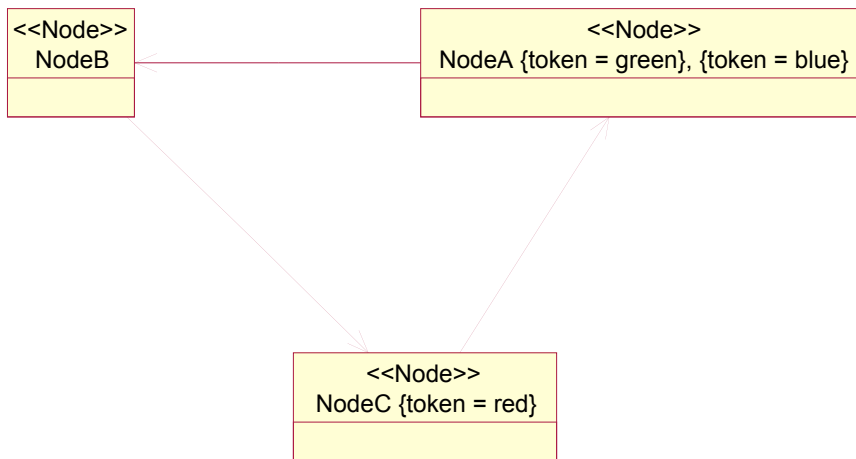


Figure 3: Sample Graph Expressed via a UML Profile

Our profile is rather crude. It's possible to do a more elegant job that would better mimic the special graph notation of Figure 2, but it still would fall short of completely reproducing that notation.

Given the "Metamodel + Profile" strategy, there are two tactics for representing Figure 3 as MOF metadata:

- *Render it as a UML model.* Using this tactic, the APIs that represent the model in a MOF repository will be those generated from the UML metamodel. When the model is encoded in an XMI document the document would conform to the XML DTD and Schema generated from the UML metamodel. The advantage of this approach is that a UML tool that knows how to render UML models as MOF metadata can render the graph model accordingly without having to add any new functionality or configuration scripts that know about the graph profile or metamodel. The UML metamodel defines the abstract syntax of the stereotype and tagged value constructs. The MOF technology mappings translate the abstract syntax into concrete representations—e.g. DTD elements and Java

interfaces—for representing a stereotype such as <<Node>> and a tagged value such as `red` in XML documents and as Java objects, as part of a UML model.

- *Render it as a Graph Model.* Using this tactic, the APIs that represent the model in a MOF repository will be those generated from the graph metamodel. When the model is encoded in an XML document the document would conform to the XML DTD and Schema generated from the graph metamodel. The advantage of this approach is that the metadata foot print is small and the semantic information is more precisely aligned with the graph domain concepts. In order to use this tactic, the generic UML tool would have to have the configuration knowledge that allows it to do metamodel-specific rendering of the metadata. Such configuration knowledge would be part of a modeling framework for the graph domain that plugs in to the generic tool.

There are some notable instances of the "Metamodel + Profile" strategy in the software standards arena. In each case the metamodel is not an extension of UML:

- *UML Profile for EJB:* The OMG has adopted a MOF metamodel of Java and EJB³ to complement the UML Profile for EJB⁴ that the Java Community Process is working on.
- *UML for EAI:* This specification, mentioned earlier, actually defines both a MOF metamodel and a UML profile for modeling enterprise application integration.⁵
- *UML Profile for CORBA:* The OMG has defined a MOF metamodel of CORBA⁶ and a UML Profile for CORBA⁷ as well.

I should have also included in this list the Component Collaboration Architecture (CCA), which has a metamodel and a corresponding profile (part of the OMG's EDOC spec)--dsf

³ [UML4EDOC], Chapter 5

⁴ [JSR26]

⁵ [UML4EAI]

⁶ [OMG ptc/01-11-03]

⁷ [OMG ptc/01-01-06]