

Model Driven Architecture, an Introduction

Goals

To clearly separate the problem definition/solution from the implementation, resulting in platform independence and greatly improved ability to maintain the resulting system and align it with the inevitable changes in requirements as it is developed.

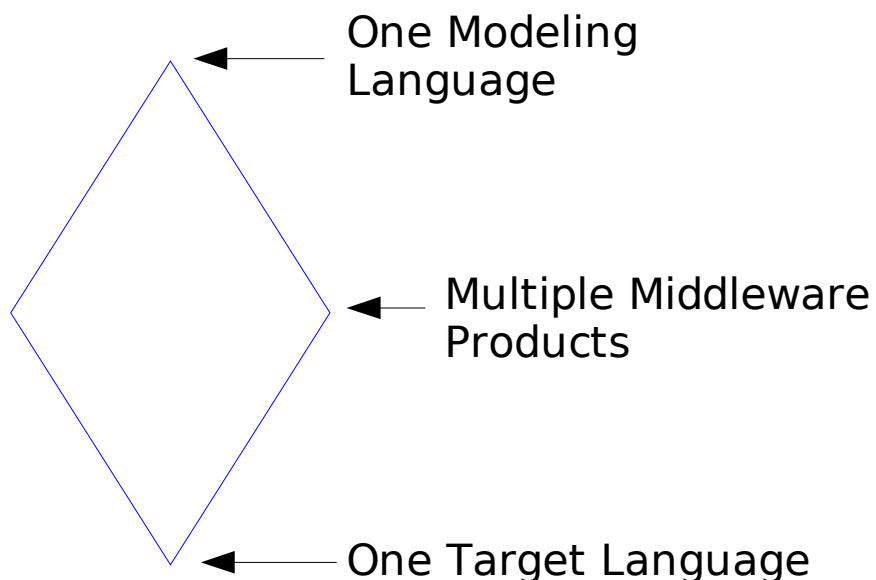
Modeling Language

The Object Management Group (OMG) has formalized UML2, a significant enhancement over the original UML. The most significant change is the idea of adding semantics to some of the drawings. This provides several benefits, including:

- ◆ The ability to check the semantics (correctness and consistency) of the drawings
- ◆ The ability to generate code from the drawings via transformations

Architecture

Unlike the traditional layered model (such as the ISO/OSI model, where there is a nexus (at layer 3)), leading to an X-shaped architecture, the proliferation of middleware products leads to a diamond shape.



Model Driven Architecture, an Introduction

Platform Independent Model (PIM)

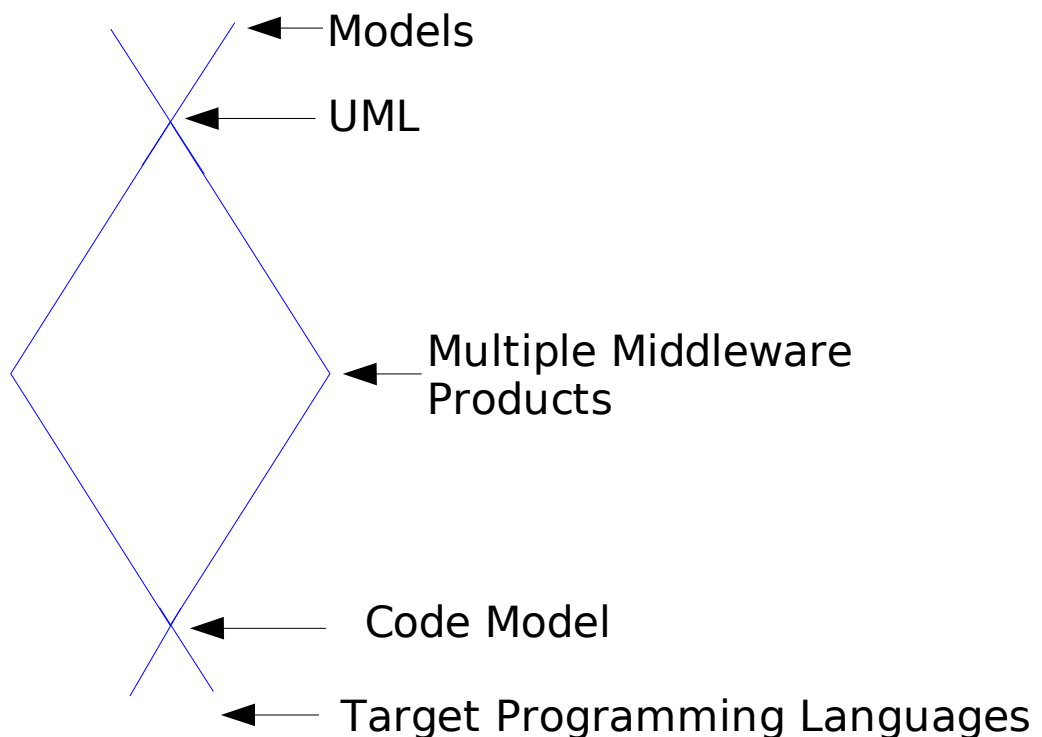
The models describing the problem domain are entirely independent of the target platform (middleware, programming language, communications, security, and any other implementation details irrelevant to the problem domain).

Platform Specific Model (PSM)

By predefining platform specific models, a transform can be applied to the PIM to produce a working system on the platform(s) of choice. All the common middleware platforms have patterns that can be consistently applied to produce the required transformations.

Reality

Modelers like to create custom extensions to the modeling language, and in the real world there are many target languages. The OMG has defined a meta drawing facility that allows drawings themselves to be transformed.



Model Driven Architecture, an Introduction

Domain View

The traditional modeling cycle goes from Use Cases, where useful information can be captured, to Class Diagrams, etc. UML2 has a drawing called a class structure diagram that captures the hierarchy of classes in the problem domain in a static view.

Code generated from the domain view

There are many standard patterns for manipulating objects, such as create, destroy, insert, remove, and navigate, where all the code can be generated automatically. Traditionally, other methods on objects are coded in the target language by hand.

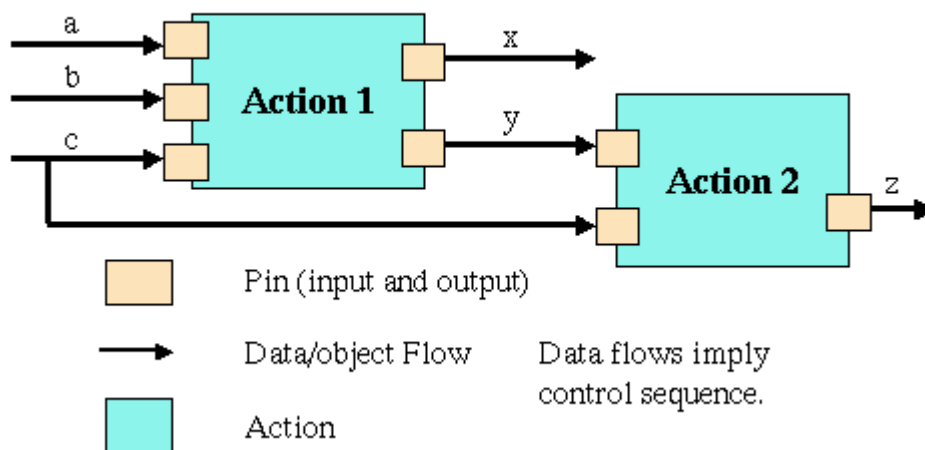
Action View

UML2 defines a new drawing style, Action Semantics, that capture the dynamic properties of a system. Components have input and output ports with well defined protocols, and can be nested to make it possible to do a top down decomposition of a system to a very fine level of detail. Action Semantics are programming language independent. No two tool vendors use the same visual style, and not many support it at all. Notably, Telelogic Tau2 and IBM Rational Real time are the only ones we know of that do, with a target application space primarily in embedded systems.

Other language agnostic approaches include the Systems Description Language (SDL, an ISO approved standard) and Sun's Architecture Independent Language.

Again, the key is that code can be generated from Action Semantics, and if taken to the full, make it possible to completely specify a system without conventional programming at all. Most IDEs will already generate template code from sequence diagrams and state charts, paving the way for complete code generation from the models. Action Semantics completes the picture. Here is an example by Kabira:

Actions generate results on *pins* connected by *data/object flows*.



Model Driven Architecture, an Introduction

Limitations of the Action Semantics

As currently defined, the models say nothing about the priority of messages entering and leaving ports, creating a significant ambiguity (non determinism). Since OMG doesn't define a representation, each vendor has a completely different way of drawing these models. Of the two known vendors supporting Action Semantics, each has a different way of specifying detailed behavior – Telelogic Tau uses SDL, and IBM/Rational Realtime uses special symbols to indicate flow. Compuware Optimal/J and IO-Software Arcstyler both specify action in a target programming language, which means you have to recode for every underlying target language. Sun's Netbeans, augmented with the Architecture independent Language, represents yet another path.

Results

EDS published a paper on results using Compuware's Optimal/J, reporting 610 vs. 3474 lines of Java (other artifacts unknown) to implement Sun's Java Pet Store. The Middleware Company found 330 Hours Vs. 507.5 hours to do the same; the 330 hours included the Optimal/J learning curve.

Using another MDA tool (Hamilton Technology's 001) which uses class structure diagrams coupled with functional programming like Action Semantics, ApogeeCT's results shows a ten times improvement in productivity: 550 Lines Axes Vs. 5258 C (Axes lines are drawing artifacts).

It should be noted that in an empirical (Source: <http://www.ipd.ira.uka.de/EIR/>) comparison of C, C++, Java, Perl, Python, Rexx, and Tcl for a search/string-processing program, the number of lines of code (or artifacts) to implement a particular algorithm varies, but MDA drawings are far more efficient. However, OMG Action Semantics are not good for describing procedural algorithms, and an SDL like addition makes for even better performance.

