Theory and Basic Principles of MDE

Jean Bézivin
JBezivin@gmail.com
Lecture 2: Theory and Basic Principles of MDE

The second course will present the basic principles of MDE from a conceptual and theoretical point of view. MDE uses typed graphs for most of the representation tasks and rule-based declarative transformations for most of the operation tasks. It relies essentially on the concepts of metamodel and model transformation technologies called Modelware. MDE is thus a graph-based and transformation-based technical space. The notion of technical space will be defined and used to compare similarities and differences with other technologies like Grammarware. Metamodels provide some sort of typing of low level representation models themselves. The essence of MDE is however most related to the unification of abstract models. Terminal models, metamodels and metametamodels are the most important categories of abstract models. Different operations like storage, retrieval and transformation may be applied on all abstract models. One of the important characteristics of MDE is to represent systems by models. This relation of representation which is central to all modeling activities will be analyzed and discussed.

If metamodels provide a strong support for representation, they have however not the power, in reason or more generally to act upon these representations. Of course it is always possible to use an imperative general purpose language to put these metamodel-based representations to work, but it has many drawbacks that will be outlined. A more constructive MDE approach usually suggested consists in using model transformations as the core operation. This solution will be discussed; particularly in the case when the transformation language is declarative and rule-based. Additional advantages may also be reaped when the transformation program may be considered itself as a model. In this case it has much better potential with domain specific transformation languages, providing maximum impact to the regular MDE architecture based on metamodels and transformations.

- Advanced concepts
- Two important additional concepts
- Technical Spaces
- DSL
- Other concepts
- Metamodels
- Metametamodels
- Multimodels
- Transformations
- Correspondence Models
- Megamodels
- etc.
Technical Spaces

**Technical Space**: A representation system for models and a set of technical solutions to handle them. A framework usually based on some algebraic structures (relational tuples, trees, graphs, hypergraphs, etc.)

**System**: A group of interacting, interrelated, or interdependent elements forming a complex whole.

**Model**: An abstract representation of a system created for a specific purpose.
Structuring the solution space

Problem: How to Solve it?

- Java
- XML
- MDA™
- Ontologies
- DBMS
- etc.

Solution Space:

- Grammars
- XML Schema
- Metamodel
- Programs
- XML Documents
- Model

Problem Space:
Three representations for the same program

- Java Grammar
- Java Metamodel
- JavaML DTD

Program TS  MDA TS  XML TS
Three representations for the same program

Each of these representations may be more convenient to perform some operation on the program.
Multiple possible representation systems

- A Technological Space (abbr. TS) employs:
  - A set of concepts
  - A working context
  - A shared knowledge and know how
  - A set of associated tools
- It is usually related to a given community with an established expertise, research problems and known results.
  *Exemple*: the Petri Net community, the Coq community, the Prolog community.

But it is mainly based on some common representation system.
Classical TSs (examples)
Alternative TSs

- A technological space is organized around a set of concepts
- Spaces are connected via bridges
- Spaces are similarly organized and operationally interoperable.
The 3-level conjecture

- Each space may be viewed as organized in three engineering "metalevels"
- These levels have nothing to do with abstraction levels
- If we can assume this organization, this will make much easier the conceptual and operational bridging between different technical spaces
Projections between EBNF, MDE and XML
Technical Space Variants – Some Examples

- **MDE Modelware (Metametamodel)**
  - EMF (Ecore)
  - MDR (MOF 1.4)
  - UML Profiles*

- **Grammarware**
  - Yacc
  - BNF
  - EBNF
  - ANTLR

- **XMLware**
  - XSD
  - Relax NG
  - DTDs*

*Not 3-level based

Legend:
- Specializes

Principles and Applications of Model Driven Engineering

Lecture #2
\(\lambda\)-Models and MDE-Models

- Technical Spaces and their representation systems
  - XML uses a specific kind of trees (XML trees),
  - RDBMS uses the relational model,
  - Ontology uses a specific kind of graphs,
  - EBNF (or Grammarware) uses formal grammars and Abstract Syntax Trees,
  - Model Driven Engineering uses a specific kind of graphs.
- The \(\lambda\)-model notation, where \(\lambda\) is the Technical Space of the model:
  - XML-models,
  - RDBMS-models,
  - EBNF-models,
  - Ontology-models,
  - MDE-models.
- Each TS has its own engineering advantages and drawbacks.

- **Everything is a model**
  - A \(\lambda\)-model
  - \(\lambda\) meaning the specific TS
  - An XML document is an XML-model
  - A Java source program is a Java-model
  - An UML model is a MDA-model
  - etc.
- Each TS is rooted in a metametamodel defining its representation scheme (representation ontology)
- Each is supposed to be organized in a 3-level architecture
- **Our default TS here is the MDE TS, but we can’t ignore the other TSs**
Intra and Inter-TS transformations

- Most Technical Spaces have internal model transformation solutions:
  - XSLT and XQuery for XML,
  - Compilers & Program transformation for EBNF,
  - QVT or ATL for Modelware,
  - etc.

- But there is also a need for inter-TS transformations.

- Many transformations may be expressed as a combination of intra and inter-TS transformations:

\[ t = f \circ g \circ h \]
Communications between technical spaces

Any software artifact can be injected into a model
Any model can be extracted to a software artifact

Cost of solving a problem inside a given TS vs. exporting it to another TS, solving it and importing back the solution.
Economic equation

XML TS

XSLT document

XSLT/XQuery transformation

XQuery document

MDE TS

XSLT model

ATL transformation

XQuery model

(1)

(2)

(3)
Collaborations between TS: Projections

MDA™

XMI → MDA™ → XML

JMI = JSR #30 from Java User Community
CMI = Corba Model Interchange (D. Frankell)
DI = Diagram Interchange for UML

Corba → CMI

SVG → DI
## Comparing spaces

<table>
<thead>
<tr>
<th></th>
<th>XML</th>
<th>MDE</th>
<th>Grammars</th>
<th>Ontologies</th>
</tr>
</thead>
<tbody>
<tr>
<td>Executability</td>
<td>Poor</td>
<td>Poor</td>
<td>Excellent</td>
<td>Poor</td>
</tr>
<tr>
<td>Aspect separation</td>
<td>Good</td>
<td>Excellent</td>
<td>Poor</td>
<td>Fair</td>
</tr>
<tr>
<td>Formalization</td>
<td>Poor</td>
<td>Poor</td>
<td>Excellent</td>
<td>Fair</td>
</tr>
<tr>
<td>Specialization</td>
<td>Fair</td>
<td>Good</td>
<td>Poor</td>
<td>Fair</td>
</tr>
<tr>
<td>Modularity</td>
<td>Good</td>
<td>Good</td>
<td>Poor</td>
<td>Poor</td>
</tr>
<tr>
<td>Traceability</td>
<td>Good</td>
<td>Fair</td>
<td>Poor</td>
<td>Fair</td>
</tr>
<tr>
<td>Transformability</td>
<td>Excellent</td>
<td>Fair</td>
<td>Fair</td>
<td>Fair</td>
</tr>
</tbody>
</table>

(NB: marks are *only indicative*)
Domain Specific Languages

- A domain-specific language (DSL) is a programming or modeling language dedicated to a particular domain, a particular problem, and/or a particular representation technique.

"DSLs are not a new idea. It is exactly what people called POL (Problem Oriented Languages) 50 years ago."

- DSLs or Special-purpose languages are becoming popular.
  - Several modeling languages are DSLs, some of them called specification languages.
The opposite of a DSL is

- a general-purpose programming language, such as Pascal, C, Java or Python,
- a general-purpose modeling language such as the Unified Modeling Language (UML).

Language-Oriented Programming considers the creation of special-purpose languages for expressing problems a standard part of the problem solving process.

- Creating a domain-specific language (with software to support it) for a repetitive type of problem.
- The cost of creating such a language should be light.
Examples of DSLs

- HTML for describing Web pages
- Verilog and VHDL for describing hardware
- Mata for matrix programming
- SQL for relational database queries
- YACC for creating parsers
- Csound for sound and music synthesis
- Graphviz for graph layout
- R for statistical computing
- Awk for Unix-based data transformation
- Make for software configuration
- Postscript for printer page layout
- Excel for basic accounting and spreadsheet calculation
- etc.
A very successful DSL (Visicalc, Excel)

Initial domain = accounting
Why Johnny can't program?

What is "programming"?

- The question really isn't "Why Johnny can't program" but rather "Why Johnny won't or doesn't choose to program".
- "Fill-in-the-forms and drop downs" and direct manipulation systems require less learning, testing of results, and debugging. There are fewer chances for error because of the constrained input choices. Rapid viewing of the results (in the domain vocabulary) makes it easier to make simple changes and see what happens.
- Programming languages like Assembler, FORTRAN, C, and Java are least likely to be well accepted by the masses.

Dan Bricklin's Web Site: www.bricklin.com

Why Johnny can't program

How the author constructs instructions to a computer, and how they can correctly anticipate the results, can affect acceptance of a system.
USA:
90 Millions computer users;
50 Millions Spreadsheet & DB users;
12 Millions self described programmers;
3 Millions professional programmers;
What is the relation between DSLs and MDE?
Both have a focus on domains
MDE is a branch of language engineering, very close to Language Oriented Programming.
In MDE, a language is created from a precise metamodel
More precisely a language is composed of:
  – Abstract syntax
  – Concrete Syntax
  – Semantics
In the MDE TS, these may mostly be represented by modeling artifacts
Definition: A DSL is a coordinated set of models
DDMM = Domain Definition MetaModel
Many models are mappings to other DSLs (semantic, concrete syntax, etc.)
The set of concepts and central relations in model-driven engineering and object-oriented technology are basically different.

Caution: Trying to use one set of relations, as a central interpretation, in the wrong context, may lead to serious problems:

- Example 1: A model being "an instance of" a meta-model
- Example 2: A UML profile "inheriting from" the UML meta-model etc.
Representation and Conformance

The two orthogonal dimensions of MDE

**Metamodel**

**System**

**Model**

repOf

wrt

conformsTo
Metamodels

A metamodel is also a graph. It is a simplified ontology, i.e. a set of concepts and relations between these concepts (bipartite graph).

A model is a graph composed of elements (nodes and edges). Each such element corresponds to a concept in the metamodel.
Structural definition of a MDE-model

Definition 1. A directed multigraph $G = (N_G, E_G, \Gamma_G)$ consists of a set of distinct nodes $N_G$, a set of edges $E_G$ and a mapping function $\Gamma_G: E_G \rightarrow N_G \times N_G$

Definition 2. A model $M = (G, \omega, \mu)$ is a triple where:

- $G = (N_G, E_G, \Gamma_G)$ is a directed multigraph
- $\omega$ is itself a model, called the reference model of $M$, associated to a graph $G_\omega = (N_\omega, E_\omega, \Gamma_\omega)$
- $\mu: N_G \cup E_G \rightarrow N_\omega$ is a function associating elements (nodes and edges) of $G$ to nodes of $G_\omega$ (metaElements)
The central role of metamodels

Each model conforms to its reference model (metamodel), but the metamodel is a model.
Definitions

- In most technologies, a three-level engineering organization has been chosen (see technical spaces later).

- In MDE, the following is assumed:
  - **Definition 3.** A *metametamodel* is a model that is its own reference model (i.e., it conforms to itself).
  - **Definition 4.** A *metamodel* is a model such that its reference model is a metametamodel.
  - **Definition 5.** A *terminal model* is a model such that its reference model is a metamodel.
Abstract models

Abstract model

- Metametamodel conformsTo
- Metamodel conformsTo
- Terminal model conformsTo
A more complete view

Abstract model

Reference model  Terminal model

MetaMetaModel  MetaModel

Transformation Model  WeavingModel

Measurement  Verification
Collections of metamodels

How to organize collections of reusable metamodels?

Several possible answers

- Atlanmod zoo with 300+ unorganized metamodels of unequal quality (empirical material)
- To a regularly organized lattice of metamodels
- With all the intermediate possibilities

Zoo Federation - All In one Zoo.

Atlantic Zoo - A zoo of metamodels expressed in KM3. Basic measurements available at KM3

AtlanEcore Zoo - An ATL-auto-generated mirror of Atlantic zoo expressed in EMF XMI 2.0.

Atlantic Microsoft SQL Server Modeling "M" Zoo - An ATL-auto-generated mirror of AtlanEco version only uses the MSchema part of the language.

AtlanticClojure Zoo - An ATL-auto-generated mirror of Atlantic zoo expressed in Clojure.

AtlanticSBVR Zoo - An ATL-auto-generated mirror of Atlantic zoo expressed as SBVR vocal

Atlantic UML 2 Zoo - An ATL-auto-generated mirror of Atlantic zoo expressed in UML 2.1,

Atlantic yUML Zoo - An ATL-auto-generated mirror of Atlantic zoo expressed in yUML, confc

Atlantic Raster Zoo - An ATL-auto-generated mirror of Atlantic zoo expressed in PNG bitmap

Atlantic GraphML Zoo - An ATL-auto-generated mirror of Atlantic zoo expressed in GraphML

Atlantic Measurement Zoo - An ATL-auto-generated mirror of Atlantic zoo presenting measures

Atlantic MOF/MDR Zoo - An ATL-auto-generated mirror of Atlantic zoo expressed in MDR XM

Atlantic UML Zoo - An ATL-auto-generated mirror of Atlantic zoo expressed in MDR XM 1.2

Atlantic SQL DDL Zoo - An ATL-auto-generated mirror of Atlantic zoo expressed in SQL DDL
The vision: a lattice of metamodels

A collection of several hundreds of metamodels (DSLs)
of low granularity and high abstraction power.
Utilization definition

Two orthogonal definitions for a model
- Structural definition (language dimension)
- Utilization definition (ontology dimension)
Utilization definition

The objective here is to define the possible usages of a model. Consequently, in all the present subsection, model will mean "terminal model".

- **Definition 6.** A system $S$ is a delimited part of the world considered as a set of elements in interaction.

- **Definition 7.** A model $M$ is a representation of a given system $S$, satisfying the substitutability principle (see below).

- **Definition 8.** (Principle of substitutability). A model $M$ is said to be a representation of a system $S$ for a given set of questions $Q$ if, for each question of this set $Q$, the model $M$ will provide exactly the same answer that the system $S$ would have provided in answering the same question.
Principle of limited substituability

- We use the term "model" in the following sense: To an observer B, an object A* is a model of an object A to the extent that B can use A* to answer questions that interest him about A. ...

- It is understood that B's use of a model entails the use of encodings for input and output, both for A and A*. If A is the world, questions for A are experiments. ...

- A* is a good model of A, in B's view, to the extent that A*'s answers agree with those of A's, on the whole, with respect to the questions important to B. ...

Marvin L. Minsky
Matter, Mind and Models Semantic Information Processing,
MIT Press, 1968
"What about the [relationship between model and real-world]? The answer, and one of the main points I hope you will take away from this discussion, is that, at this point in intellectual history, we have no theory of this [...] relationship".

Brian Cantwell Smith *The Limits of Correctness*  

See also “*On the origin of objects*”  
ISBN 0-262-69209-0
On the origin of models

Naïve answer: from Rational Rose

a system S

repOf

a model M

Where are models coming from?
How to assess the fidelity (or faithfulness) of models?

- Two possible sources for models (only two, initial and derived)
  1. Transformation of other models
  2. Observation of a system

- Because we gave too much emphasis to our technology and not enough to its applications, computer science has got a bad reputation of buggy and loose engineering.

- The real question of the origin of objects has never been seriously addressed by the community.

- A software can be written in the most wonderful language and completely misunderstand the reality it is supposed to control and miss its goal.

- We have never seriously invested time and energy in seriously studying the faithfulness or fidelity of models to systems and this is one reason why we deserve the bad reputation of our software systems. It’s about time we start studying ways to evaluate how accurate a copy is to its source.

- If we want to improve the public image of software engineering, this is an important field on which we should invest in future research.
Multimodeling is the joint exploitation of different models representing the same system. These models usually conform to different metamodels. Multimodeling suggests to manage complex systems by collaborative reasoning based on multiple models, each one encompassing a specific type of knowledge (e.g. structural, behavioral, functional) and representation. N.B.: the different models (Ma, Mb, Mc, ...) may also be linked together.
A usual program is an example of a composite model, representing at the same time the domain, the execution platform, and the application user requirements.
Model transformations

Transformation Rules Based on Meta-modeling

An Object Meta-model (UML)

Classifier

Feature

owner

feature

Address

number
street
city

An Object Model

A Relational Meta-model

Table

Column

owner
column

Address

number
street
city

A Relational Model

Transformation rules

based on

Object meta-model

Relational meta-model

based on

Object model

A Relational model

transformation
Endogeneous and exogeneous transformations

- Endogeneous: MM\(_a = MM\(_b\)
- Exogeneous: MM\(_a \neq MM\(_b\)
  - Changing the source and target metamodels to MM\(_a \cup MM\(_b\) may change the type from exogeneous to endogeneous
Model transformations as models

Model-to-Model transformation DSLs

- A transformation ($Mt$) builds a target model ($Mb$) from a source model ($Ma$)
- Model transformations are transformation models
  - Each model conforms to a metamodel
  - A transformation is a model ($Mt$) conforming to a metamodel ($MMt$)

\[
\text{conformsTo} \quad \text{transformation}
\]
Experiments in Model Transformation

Model

TerminalModel

Model Transformation

Verification

Measurement

ViewExtraction

MetaModel

Empirical research on model transformation

Model Weaving

MegaModel

etc
Example: UML to Java

UML Metamodel

Java Metamodel
Example: Java to Excel

```
public class FirstClass {
    public void fc_m1() {
    }
    public void fc_m2() {
        this.fc_m1();
        this.fc_m1();
    }
}

public class SecondClass {
    public void sc_m1() {
        FirstClass a = new FirstClass();
        a.fc_m1();
    }
    public void sc_m2() {
        this.sc_m1();
    }
}
```

```
<table>
<thead>
<tr>
<th></th>
<th>FirstClass.fc_m1</th>
<th>FirstClass.fc_m2</th>
<th>SecondClass.sc_m1</th>
<th>SecondClass.sc_m2</th>
</tr>
</thead>
<tbody>
<tr>
<td>FirstClass.fc_m1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>FirstClass.fc_m2</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>SecondClass.sc_m1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>SecondClass.sc_m2</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
```
Example: XSLT to XQuery

### XSLT

```xml
<xsl:stylesheet ["...]
  <xsl:template match="/"
    <emps>
    <xsl:apply-templates
     select="employee"/>
    </emps>
  </xsl:template>
  <xsl:template match="employee">
    <xsl:if test="salary>2000">
      <emp>
      <xsl:value-of select="name"/>
      <xsl:value-of
       select="firstname"/>
      </emp>
    </xsl:if>
  </xsl:template>
</xsl:stylesheet>
```

### XQuery

```xml
define function fctemployee($paramVar) {
  for $var in $paramVar
  return
  let $var := $var
  where $var/salary>2000
  return
  <emp>{$var/name}{$var/firstname}</emp>
}
for $var in document("xmlFile.xml")/
return
<emps>{fctemployee($var/employee)}</emps>
```

Note: TS crossing needed
Example: Tool Interoperability

About 30 open source tools to choose among for bug tracking (Mantis, Bugzilla, etc.)

<table>
<thead>
<tr>
<th>Tool</th>
<th>Lang</th>
<th>Ver</th>
<th>Cust</th>
<th>Temp</th>
<th>Search</th>
<th>RSS</th>
<th>Not</th>
<th>Rep</th>
<th>Hist</th>
<th>Attac</th>
<th>Updated</th>
<th>Demo</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASP.NET Starter</td>
<td>C#/VB</td>
<td></td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>No</td>
<td>2</td>
</tr>
<tr>
<td>Bug-a-Boo</td>
<td>CGI</td>
<td></td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Yes</td>
<td>4</td>
</tr>
<tr>
<td>Bug Base</td>
<td>Java</td>
<td></td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Aug 03</td>
<td>0</td>
</tr>
<tr>
<td>BugIn</td>
<td>PHP</td>
<td></td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>May 04</td>
<td>0</td>
</tr>
<tr>
<td>Bugs Online</td>
<td>ASP</td>
<td></td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Yes</td>
<td></td>
<td>Jan 02</td>
<td>3</td>
</tr>
<tr>
<td>BugTracker</td>
<td>Java</td>
<td></td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Apr 01</td>
<td>0</td>
</tr>
<tr>
<td>BugTracker.NET</td>
<td>C#</td>
<td></td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td></td>
<td></td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Jan 03</td>
<td>No</td>
<td>2</td>
</tr>
<tr>
<td>Bugzilla</td>
<td>Perl</td>
<td></td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Jan 03</td>
<td>Yes</td>
<td>2</td>
</tr>
<tr>
<td>Eventum</td>
<td>PHP</td>
<td>1.5.4</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td></td>
<td></td>
<td></td>
<td>Yes</td>
<td>Yes</td>
<td>Jun 05</td>
<td>No</td>
<td>7</td>
</tr>
<tr>
<td>EZ Ticket</td>
<td>PHP</td>
<td></td>
<td></td>
<td>Yes</td>
<td>Yes</td>
<td></td>
<td></td>
<td></td>
<td>Yes</td>
<td>Yes</td>
<td>Jan 04</td>
<td>No</td>
<td>1</td>
</tr>
<tr>
<td>Flyspray</td>
<td>PHP</td>
<td></td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td></td>
<td></td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Jan 05</td>
<td>Yes</td>
<td>4</td>
</tr>
<tr>
<td>GNATS</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Yes</td>
<td></td>
<td></td>
<td></td>
<td>Yes</td>
<td>Yes</td>
<td>Mar 05</td>
<td>Yes</td>
<td>1</td>
</tr>
<tr>
<td>Issue Tracker</td>
<td>PHP</td>
<td></td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td></td>
<td></td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Feb 04</td>
<td>Yes</td>
<td>4</td>
</tr>
<tr>
<td>JTracker</td>
<td>Java</td>
<td></td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td></td>
<td></td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Aug 04</td>
<td>Yes</td>
<td>2</td>
</tr>
<tr>
<td>JTracker</td>
<td>Python</td>
<td></td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td></td>
<td></td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>May 04</td>
<td>No</td>
<td>0</td>
</tr>
<tr>
<td>Mantis</td>
<td>PHP</td>
<td></td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>May 05</td>
<td>Yes</td>
<td>10</td>
</tr>
<tr>
<td>Midge</td>
<td>Python</td>
<td></td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td></td>
<td></td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Oct 04</td>
<td>No</td>
<td>0</td>
</tr>
<tr>
<td>OpenPGA</td>
<td>PHP</td>
<td></td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td></td>
<td></td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Jan 05</td>
<td>Yes</td>
<td>3</td>
</tr>
<tr>
<td>Support</td>
<td>PHP</td>
<td></td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td></td>
<td></td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Oct 04</td>
<td>No</td>
<td>6</td>
</tr>
<tr>
<td>OTRS</td>
<td>Perl</td>
<td></td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Nov 05</td>
<td>Yes</td>
<td>5</td>
</tr>
<tr>
<td>phpBugTracker</td>
<td>PHP</td>
<td></td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Apr 04</td>
<td>No</td>
<td>6</td>
</tr>
<tr>
<td>PhoneCollector-NG</td>
<td>Python</td>
<td></td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Feb 05</td>
<td>No</td>
<td>7</td>
</tr>
<tr>
<td>Request Tracker</td>
<td>Perl</td>
<td></td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Mar 05</td>
<td>Yes</td>
<td>7</td>
</tr>
<tr>
<td>Roundup</td>
<td>Python</td>
<td></td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Mar 05</td>
<td>Yes</td>
<td>0</td>
</tr>
<tr>
<td>sBugs</td>
<td>Java</td>
<td></td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td></td>
<td></td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Jan 02</td>
<td>Yes</td>
<td>0</td>
</tr>
<tr>
<td>Scarab</td>
<td>Java</td>
<td></td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Apr 04</td>
<td>Yes</td>
<td>9</td>
</tr>
<tr>
<td>Subissue</td>
<td></td>
<td></td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>N/A</td>
<td>No</td>
<td>0</td>
</tr>
<tr>
<td>SuperCRM</td>
<td></td>
<td>5</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Jun 05</td>
<td>No</td>
<td>5</td>
</tr>
<tr>
<td>Trac</td>
<td>Python</td>
<td></td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Nov 04</td>
<td>Yes</td>
<td>6</td>
</tr>
<tr>
<td>Whups</td>
<td>PHP</td>
<td></td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Apr 02</td>
<td>No</td>
<td>1</td>
</tr>
<tr>
<td>Workbench</td>
<td>PHP</td>
<td></td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Dec 03</td>
<td>Yes</td>
<td>1</td>
</tr>
<tr>
<td>Zope Issue Tracker</td>
<td>Python</td>
<td></td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Apr 04</td>
<td>Yes</td>
<td>2</td>
</tr>
</tbody>
</table>

Different data models and functionalities
Mantis metamodel (simplified)
Bug control metamodel (pivot)

Several other types of control could be added by creating new classes that inherit of the abstract class "ControlType"...
Excel-to-Bugzilla and Excel-to-Mantis bridges

- Microsoft Office Excel
  - XML Schema

- SpreadsheetML
  - XML
  - Excel (SpreadsheetML Simplified)
  - Software Quality Control

- Bugzilla
- Mantis
  - XML

- Ecore
  - C2

- Eclipse EMF

- Model
  - InjectXML
  - XML2SpreadsheetMLSimplified
  - SpreadsheetMLSimplified2SoftwareQualityControl

- Model
  - SoftwareQualityControl2Bugzilla
  - SoftwareQualityControl2Mantis

- Model
  - Bugzilla2XML
  - Mantis2XML

- Model
  - XML2BugzillaText
  - XML2MantisText

- Model
  - XML document

- XML document

- XML document
Partial Correspondences or weavings

1. How to represent the partial correspondences? (M c2 an extensible MM)
2. How to compute the partial correspondences? (model matching)
3. How to use the partial correspondences? (HOTs)
Principles of Model Weaving

- Fixed mapping metamodels are not sufficient
- We need support for extensible variable metamodels
Need for a global view: Megamodels

Global view

OMG/MDA recommendations

MOF
UML
SPEM
CWM
WfI
SysML
BPMN
OCL
e tc.
Megamodeling

- With megamodeling, a given model may describe a set of other models, their metadata and the mutual relationships between them.

- Since a megamodel is itself a model, this allows to represent arbitrarily deeply nested systems of systems.

- MDE solution for complex systems management
A model map (megamodel)
A model

Element

Association

Model

Correspondence
General organization

```
context MetaMetaModel inv: self.conformsTo = self
context MetaModel inv: self.conformsTo.oclIsKindOf(MetaMetaModel)
context TerminalModel inv: self.conformsTo.oclIsKindOf(MetaModel)
```
Megamodels for complex systems

Complex System

Metamodel

repOf

repOf

S

M

MM

M1

M2

M3
Software Project

UML Modeling
Improve modeling efficiency with this easy-to-use, feature-rich and reliable UML 2.1 modeling tool.

Database Modeling
Enhance database documentation quality with sophisticated ERD and Object-Relational Mapping diagrams.

Object-Relational Mapping
Know how to manipulate objects in Java, .NET and PHP. Then you now know how to create access to your data from more than one database programming! Just leave it to us.

Interoperability
Exchange UML diagrams and models with other industrial standard representation.

IDE Integration
Support full software development life-cycle, from analysis to design, and from design to implementation with your most favorite IDE.

Requirement Modeling
Capture requirements with SysML Requirement Diagram, Use Case Modeling, Textual Analysis, CRC and more.

Process Modeling
We understand and improve your business process with the most comprehensive BPMN tool.

Team Collaboration
Perform modeling collaboratively and concurrently with the VP Teamwork Server, CVS and Subversion.

Code Engineering
Keep model and code in sync with over 10 programming languages of your choice.

Documentation Generation
Share your design with your customers in popular document formats (PDF, HTML and MS Word).
How to define a metametamodel?

- Several examples
  - MOF, EMOF, CMOF, SMOF, ECORE, etc.

- Two main approaches
  - Universalist
  - Minimalist

- The design of a metametamodel is a difficult exercise in engineering (many tradeoffs)

- Two examples
  - KM3
  - sNets
The KM3 metamodel (compromise)

KM3 is a DSL for defining metamodels;
It's a M3 notation (self-defined)
[The presentation above is simplified.]
Self-definition of the KM3 metamodel

package KM3 {

    class Package extends ModelElement {
        reference contents[*] ordered
            container : ModelElement oppositeOf "package";
    }

    class Classifier extends ModelElement {}

    class DataType extends Classifier {}

    class Class extends Classifier {
        attribute isAbstract : Boolean;
        reference supertypes[*] : Class;
        reference structuralFeatures[*] ordered
            container : StructuralFeature
            oppositeOf owner;
    }

    ...
}
The sNets minimalist approach

sNets: Typed Partitioned Reflective Semantic Networks.

$$\exists x, \exists y : \text{Cat} (x) \land \text{Mat} (y) \land \text{on} (x,y)$$
A minimal M3

M3

M2

M1

Principles and Applications of Model Driven Engineering  Lecture #2
Explicit built-in modularity
Alignment on OMG standards

∃ x, ∃ y : Cat (x) ∧ Mat (y) ∧ on (x, y)
Main messages of the lecture

- Technology evolution may be characterized by different Technical Spaces (TSs).
  - Based on one representation system, these TSs may communicate and offer various cost alternative for similar solutions.
  - MDE is one of these TSs, based on graphs and performs well in some operation, less on others

- Domain Specific Languages (DSLs) are opposed to General Purpose Languages on their special purpose and their focus on a given domain.
  - DSLs

- Metamodels are the keystone of MDE
  - They are type graphs, playing the role of grammars or Schemas in other TSs
  - The MDE TS is organized around one central representation principle (models and reference models)
  - As most other TSs, MDE is organized in three layers

- Transformations are the main operation on models
  - Various kinds of transformations
  - Purely intuitive conjecture: 80% of the needs of an IT design could be covered with pure model transformations.
Thanks

• Questions?
• Comments?

JBezivin@gmail.com