Whole Platform Foundation

The Long Way Toward Language Oriented Programming
Outline

• Aim: Engineering the Production of Software
• Searching for a Common Denominator of Knowledge
• Looking for Answers within some Methodologies
  ♦ Object Oriented Programming
  ♦ Model Driven Engineering
  ♦ Language Oriented Programming
  ♦ Generative Programming
• From Methodologies to Technologies
• Whole Platform Overview
• Whole Platform Components
Aim: Engineering the Production of Software

• Raise the abstraction level to the *domain level*
  - Use domain languages for representing knowledge
  - Separate the knowledge from the implementation technologies
  - Involve domain experts in the development process

• Bridge the *Execution gap*
  - Rewrite programs into models and model transformations
  - Use generators to stay up-to-date with current technologies

• Bridge the *Data gap*
  - Specify the data formats using the data integration languages
  - Let the integration tools generate the models and parsers
KNOWLEDGE
comes in an almost infinite variety of forms

• Main sources of variability:
  • languages,
  • data formats,
  • notations

• What is the common denominator of knowledge?
  • Let us see a representative set of knowledge examples…
  • …and the answers of some methodologies

• ALERT: Definitions abound in literature
  • The followings reflect our understanding and vision
What is the common denominator of knowledge? (Java example)

```java
import java.util.Arrays;
import java.util.Comparator;

public class MainClass {
    public static void main(String[] args) {
        String[] sa = new String[] { "adf", "fdsa", "ASDF", "FSA", "r", "R" };

        System.out.println("Before sorting: " + Arrays.asList(sa));
        Arrays.sort(sa, new AlphabeticComparator());
        System.out.println("After sorting: " + Arrays.asList(sa));
    }
}

class AlphabeticComparator implements Comparator {
    public int compare(Object o1, Object o2) {
        String s1 = (String) o1;
        String s2 = (String) o2;
        return s1.toLowerCase().compareTo(s2.toLowerCase());
    }
}
```
What is the common denominator of knowledge? (csharp example)

```csharp
using System;

class RefTest {
    /* This method changes its argument.
       Notice the use of ref. */
    public void sqr(ref int i) {
        i = i * i;
    }
}

class MainClass {
    public static void Main() {
        RefTest ob = new RefTest();

        int a = 10;

        Console.WriteLine("a before call: " + a);

        ob.sqr(ref a); // notice the use of ref

        Console.WriteLine("a after call: " + a);
    }
}
```
What is the common denominator of knowledge? (Cobol example)

```cobol
DATA DIVISION.
WORKING-STORAGE SECTION.
01 IterCount PIC 99 VALUE ZEROS.
88 MaxCountReached VALUE 99.
01 UserInput PIC 99 VALUE ZEROS.
88 EndOfUserInput VALUE ZEROS.
01 RunningTotal PIC 999 VALUE ZEROS.
01 AverageValue PIC 999 VALUES ZEROS.

PROCEDURE DIVISION.
Begin.
PERFORM UNTIL IterCount = 5
   DISPLAY "IterCount = " IterCount
   ADD 1 TO IterCount
END-PERFORM
DISPLAY "Finished in line Perform."

INITIALIZE Itercount

DISPLAY "Enter a stream of up to 99 numbers."
DISPLAY "Each number must be in the range 1-99. Enter 0 to stop."
DISPLAY "Enter number :- " WITH NO ADVANCING
ACCEPT UserInput
PERFORM GetUserInput UNTIL EndOfUserInput OR MaxCountReached

DISPLAY "The final total is - " RunningTotal
DISPLAY "The final count is - " IterCount
COMPUTE AverageValue = RunningTotal / IterCount
DISPLAY "The average value entered is - " AverageValue
STOP RUN.

GetUserInput.
ADD UserInput TO RunningTotal
   ON SIZE ERROR DISPLAY "Error - new total too large for data-item."
   NOT ON SIZE ERROR ADD 1 TO IterCount END-ADD
END-ADD
DISPLAY "Total so far is - " RunningTotal
DISPLAY "Count so far is - " IterCount
DISPLAY "Enter number :- " WITH NO ADVANCING
ACCEPT UserInput.
```
What is the common denominator of knowledge? (CSV example)

<table>
<thead>
<tr>
<th>Team</th>
<th>Manufacturer</th>
<th>Name</th>
<th>Position</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ducati Marlboro Team</td>
<td>Ducati</td>
<td>Casey Stoner</td>
<td>1</td>
</tr>
<tr>
<td>Fiat Yamaha Team</td>
<td>Yamaha</td>
<td>Valentino Rossi</td>
<td>46</td>
</tr>
<tr>
<td>Repsol Honda Team</td>
<td>Honda</td>
<td>Daniel Pedrosa</td>
<td>26</td>
</tr>
<tr>
<td>Team Alice</td>
<td>Ducati</td>
<td>Tony Elias</td>
<td>24</td>
</tr>
<tr>
<td>Sylvain Guintoli</td>
<td></td>
<td>Nicky Hayden</td>
<td>69</td>
</tr>
<tr>
<td>Honda Gresini</td>
<td>Honda</td>
<td>Shinya Nakano</td>
<td>56</td>
</tr>
<tr>
<td>Alex De Angelis</td>
<td>Yamaha Tech 3</td>
<td>Colin Edwards</td>
<td>5</td>
</tr>
<tr>
<td>James Toseland</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Rizla Suzuki MotoGP</td>
<td>Suzuki</td>
<td>Chris Vermeulen</td>
<td>7</td>
</tr>
<tr>
<td>Loris Capirossi</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Kawasaki Racing Team</td>
<td>Kawasaki</td>
<td>John Hopkins</td>
<td>21</td>
</tr>
<tr>
<td>Anthony West</td>
<td>Honda LCR</td>
<td>Randy De Puniet</td>
<td>14</td>
</tr>
<tr>
<td>Honda</td>
<td></td>
<td>Andrea Dovizioso</td>
<td>3</td>
</tr>
<tr>
<td>Jir Scot Team</td>
<td>Honda</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
What is the common denominator of knowledge? (FIN series 5 example)
What is the common denominator of knowledge? (Manifest form example)
What is the common denominator of knowledge? (Search dialog example)
What is the common denominator of knowledge? (radial tree and circular layout graph examples)
What is the common denominator of knowledge? (mind map example)
What is the common denominator of knowledge? (fishbone chart example)
What is the common denominator of knowledge? (UML Class Diagram example)
What is the common denominator of knowledge? (BPMN example)
What is the common denominator of knowledge?
(BPEL XML example)

```xml
<?xml version="1.0" encoding="utf-8"?>
<bpel:process>
  <bpel:scope name="cook a cake">
    <bpel:sequence>
      <bpel:scope name="find a recipe">
        <bpel:sequence>
          <bpel:assign name="search on the web"/>
          <bpel:assign name="print recipe"/>
        </bpel:sequence>
      </bpel:scope>
      <bpel:assign name="shop for ingredients"/>
      <bpel:assign name="prepare the kitchen"/>
      <bpel:scope name="cook recipe">
        <bpel:assign name="melt flower with paper"/>
        <bpel:assign name="add ingredients"/>
        <bpel:scope name="bake cake">
          <bpel:sequence>
            <bpel:assign name="heat the oven at 300F"/>
            <bpel:assign name="leave the cake in the oven for 30 minutes"/>
          </bpel:sequence>
        </bpel:scope>
      </bpel:scope>
    </bpel:sequence>
  </bpel:scope>
</bpel:process>
```
What is the common denominator of knowledge? (Maven XML example)

```xml
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-v4_0_0.xsd">
  <modelVersion>4.0.0</modelVersion>

  <groupId>whole-platform</groupId>
  <artifactId>whole-platform</artifactId>
  <version>1.0.0-v20081007-1544</version>

  <packaging>pom</packaging>

  <name>Whole Platform</name>
  <url>http://whole.sourceforge.net</url>

  <build>
    <sourceDirectory>src</sourceDirectory>
    <outputDirectory>bin</outputDirectory>

    <plugins>
      <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-compiler-plugin</artifactId>

        <configuration>
          <source>1.5</source>
          <target>1.5</target>
        </configuration>
      </plugin>
    </plugins>
  </build>
</project>
```
What is the common denominator of knowledge?
(SVG XML example)

```xml
<?xml version="1.0"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN"
   "http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
<svg xmlns="http://www.w3.org/2000/svg" version="1.1"
     width="467" height="462">

<!-- This is the red square: -->
<rect x="80" y="60" width="250" height="250" rx="20" fill="red"
     stroke="black" stroke-width="2px" />

<!-- This is the blue square: -->
<rect x="140" y="120" width="250" height="250" rx="40" fill="blue"
     stroke="black" stroke-width="2px" fill-opacity="0.7" />

</svg>
```
What is the common denominator of knowledge? (Swift XML example)

```xml
<?xml version="1.0" encoding="UTF-8"?>
<ns0:Document xmlns:ns0="urn:swift:xsd:$pacs.008.003.02" xmlns:xsi="http://
  xsi:schemaLocation="urn:swift:xsd:$pacs.008.003.02 EPSCTPACS00800302M.xsd">
  <ns0:CdtTrfTxInf>
    <ns0:PmtId>
      <ns0:InstrId>80131897065.0001</ns0:InstrId>
      <ns0:EndToEndId>NOTPROVIDED</ns0:EndToEndId>
      <ns0:TxId>80131897065.0001</ns0:TxId>
    </ns0:PmtId>
    <ns0:PmtTpInf>
      <ns0:Svclvl>
        <ns0:Cd>SEPA</ns0:Cd>
      </ns0:Svclvl>
    </ns0:PmtTpInf>
    <ns0:IntrBkSttlmAmnt Ccy="EUR">2469.6</ns0:IntrBkSttlmAmnt>
    <ns0:IntrBkSttlmDt>2007-12-15</ns0:IntrBkSttlmDt>
    <ns0:InstdAmnt Ccy="EUR">123.23</ns0:InstdAmnt>
    <ns0:ChrgBr>SLEV</ns0:ChrgBr>
    <ns0:ChrgsInf>
      <ns0:ChrgsAmnt Ccy="EUR">123.45</ns0:ChrgsAmnt>
      <ns0:ChrgsPty>
        <ns0:FinInstnId>
          <ns0:BIC>BCV SCH2L</ns0:BIC>
        </ns0:FinInstnId>
      </ns0:ChrgsPty>
    </ns0:ChrgsInf>
  </ns0:CdtTrfTxInf>
</ns0:Document>
```
Outline

• Aim: Engineering the Production of Software
• Searching for a Common Denominator of Knowledge
• Looking for Answers within some Methodologies
  ♦ Object Oriented Programming
  ♦ Model Driven Engineering
  ♦ Language Oriented Programming
  ♦ Generative Programming
• From Methodologies to Technologies
• Whole Platform Overview
• Whole Platform Components
OBJECT
is the Object Oriented Programming’s answer

• Describe a Software System using a set of classes, their attributes, and the relationships between the classes
• An Object is an instance of a Class
• At runtime we always have “aggregates of objects”
• UML Class and Object diagrams are more abstract but help in visualizing the concepts
Object Oriented Programming is not enough

• **Object** is only the *lowest common denominator*
  - For instance the Java Object gives:
    - Object identity
    - Support for multithreading
    - Basic support for cloning
    - Basic support for reflection  
      (but difficult to use)

• Can we find a greater common denominator?
  - We are searching for the *greatest common denominator*
  - Similarity between knowledge forms is not helpful!
MODEL
is the Model Driven Engineering’s answer

• The *Model Driven Development* of a Software System consists of identifying a set of domains, defining the corresponding models and model transformations.

• A *Domain* is a knowledge area
  ✷ i.e. concepts, terminology, activities

• A *Model* is an abstraction over (part of) a domain
  ✷ A model collects information about a domain by defining: *entities*, their *features* and the relationships between the entities.

• A *Metamodel* is a model used to define other models

• At runtime we *always* have instances of models
Examples of defining models

... using the Models language
Models and metamodels

• (Model of) the Models language used to define the RedBlackTree model

• (Model of) the RedBlackTree datatype

• (Model of) an example of a RedBlack tree
MDE is a big step over OOP

• MDE introduces new names for existing concepts
  - metamodel stands for the classes composing a (sub)system
  - model stands for an “aggregate of objects”

• New terminology is just the first contribution

• MDE’s fundamental contributions are:
  - a new perspective: the language design focus is reversed
  - more services: the capabilities shared by all languages are maximized
The Java developer’s perspective on languages

• (Programming) languages are *general purpose* (GPLs)
• The *fixed* syntax(es) and semantics give a language an identity
  - The abstract syntax is not even standardized
  - Designing and implementing an object structure representation of a program is a tool responsibility

```java
package org.jug.test;

public class Hello {
    public static void main(String[] args) {
        System.out.println("Hello Java");
    }
}
```
MDE reverses the language design focus

• A language can be general purpose or domain specific
• A DSL does not require an executable semantics
  ♦ even data formats are (domain) languages
• Semantics does not care about concrete and serialization syntaxes
  ♦ algorithms operate on data structures not language sentences
• MDE focuses on the design of the abstract syntax of a language: the model
Consequences of the focus on models

- Only the model (abstract syntax) is fixed in a language.
- Any number of notations (concrete syntaxes), persistences (serialization syntaxes) and capabilities (semantics) can be defined.
  - These enable both more specificity and more commonality.
  - To achieve more commonality is the MDE goal.
MDE maximizes the capabilities shared by all languages

• MDE requires a *modeling framework* to implement models
  - An Entity is basically an Object with more capabilities
  - Every model, for instance, exposes a common *generic* API
  - The generic API enables the definition of *cross-language* behavior and of *language-neutral* syntaxes
  - A modeling framework promotes uniformity and accelerates application development

• With more out-of-the-box capabilities, the user can concentrate himself on business behavior

• With out-of-the-box persistences and notations, the user defines a model and obtains a language
Whole Language Map of Generic Persistences
Example of the Xml Builder generic persistence

```xml
<?xml version="1.0" encoding="UTF-8" ?>

<actions:LanguageActionFactory xmlns:actions="http://lang.whole.org/Actions">
  <actions:Name value="RedBlackTreeActions"/>
  <actions:URI value="http://datatypes.examples.whole.org/RedBlackTree"/>
  <actions:ToolbarActions>
    <actions:SimpleAction>
      <actions:Label value="validator"/>
    </actions:SimpleAction>
    <commons:StageUpFragment xmlns:commons="http://lang.whole.org/Commons">
      <queries:Block xmlns:queries="http://lang.whole.org/Queries" initialCapacity="7">
        <queries:QueryDeclaration>
          <queries:Name value="addError"/>
          <queries:Name initialCapacity="2"/>
          <queries:Name value="message"/>
          <queries:Name value="location"/>
        </queries:QueryDeclaration>
        <queries:Sequence initialCapacity="1"/>
        <commons:SameStageFragment>
          <java:ExpressionStatement xmlns:java="http://lang.whole.org/Java">
            <java:MethodInvocation>
              <java:SimpleName value="decorationManager"/>
            </java:MethodInvocation>
          </java:ExpressionStatement>
        </commons:SameStageFragment>
      </queries:Block>
    </commons:StageUpFragment>
  </actions:ToolbarActions>
</actions:LanguageActionFactory>
```
Example of the Java Builder generic persistence

```java
public class RedBlackTreeActions extends AbstractTemplateFactory<LanguageActionFactory> {

    public void apply(IBuilderOperation op) {
        IActionsBuilder b0 = (IActionsBuilder) op.wGetBuilder(ActionsLanguageKit.ID);
        b0.LanguageActionFactory_();
        b0.Name("RedBlackTreeActions");
        b0.URI("http://datatypes.examples.whole.org/RedBlackTree");
        b0.ToolbarActions_();
        b0.SimpleAction_();
        b0.Label("validator");
        ICommonsBuilder b1 = (ICommonsBuilder) op.wGetBuilder(CommonsLanguageKit.ID);
        b1.StageUpFragment_();
        IQueriesBuilder b2 = (IQueriesBuilder) op.wGetBuilder(QueriesLanguageKit.ID);
        b2.Block_(7);
        b2.QueryDeclaration_();
        b2.Name("addError");
        b2.Names_(2);
        b2.Name("message");
        b2.Name("location");
        b2._Names();
        b2.Sequence_(1);
        b1.StageUpFragment_();
        IJavaBuilder b3 = (IJavaBuilder) op.wGetBuilder(JavaLanguageKit.ID);
        b3.ExpressionStatement_();
        b3.MethodInvocation_();
        b3 SimpleName("decorationManager");
        b3.Types();
    }
```
Example of uniformity given by a generic persistence
Whole Language Map of Generic Notations
Examples of the **Table** generic notation

<table>
<thead>
<tr>
<th>Model</th>
<th>Libraries</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>name</td>
</tr>
<tr>
<td></td>
<td>templateTypes</td>
</tr>
<tr>
<td></td>
<td>declarations</td>
</tr>
<tr>
<td></td>
<td>SimpleEntity</td>
</tr>
<tr>
<td></td>
<td>modifiers</td>
</tr>
<tr>
<td></td>
<td>name</td>
</tr>
<tr>
<td></td>
<td>Library</td>
</tr>
<tr>
<td></td>
<td>type</td>
</tr>
<tr>
<td></td>
<td>versions</td>
</tr>
<tr>
<td></td>
<td>features</td>
</tr>
<tr>
<td></td>
<td>Features</td>
</tr>
<tr>
<td></td>
<td>modifiers</td>
</tr>
<tr>
<td></td>
<td>type</td>
</tr>
<tr>
<td></td>
<td>name</td>
</tr>
<tr>
<td></td>
<td>URI</td>
</tr>
<tr>
<td></td>
<td>uri</td>
</tr>
<tr>
<td></td>
<td>Name</td>
</tr>
<tr>
<td></td>
<td>name</td>
</tr>
<tr>
<td></td>
<td>Version</td>
</tr>
<tr>
<td></td>
<td>version</td>
</tr>
<tr>
<td></td>
<td>Optional</td>
</tr>
<tr>
<td></td>
<td>Tags</td>
</tr>
<tr>
<td></td>
<td>tags</td>
</tr>
<tr>
<td></td>
<td>Imports</td>
</tr>
<tr>
<td></td>
<td>imports</td>
</tr>
<tr>
<td></td>
<td>Rules</td>
</tr>
<tr>
<td></td>
<td>rules</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Grammar</th>
<th>Productions</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>name</td>
</tr>
<tr>
<td></td>
<td>rule</td>
</tr>
<tr>
<td></td>
<td>template</td>
</tr>
<tr>
<td>MotoGP_TeamsCSVGrammar</td>
<td>NL</td>
</tr>
<tr>
<td></td>
<td>SEP</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Productions</th>
<th>name</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>rule</td>
</tr>
<tr>
<td></td>
<td>template</td>
</tr>
<tr>
<td>Teams</td>
<td>Repeat</td>
</tr>
<tr>
<td></td>
<td>lowerBound</td>
</tr>
<tr>
<td></td>
<td>upperBound</td>
</tr>
<tr>
<td></td>
<td>separator</td>
</tr>
<tr>
<td></td>
<td>NL</td>
</tr>
<tr>
<td>Rider</td>
<td>As</td>
</tr>
<tr>
<td></td>
<td>rule</td>
</tr>
<tr>
<td></td>
<td>Name</td>
</tr>
<tr>
<td></td>
<td>name</td>
</tr>
<tr>
<td></td>
<td>SEP</td>
</tr>
</tbody>
</table>
Examples of the **Tree** generic notation
Examples of the **Outline** generic notation

- **Teams**
  - **Team**
    - name Ducati Marlboro Team
    - manufacturer Ducati
      - rider1
    - rider2
  - **Team**
    - name Fiat Yamaha Team
    - manufacturer Yamaha
      - rider1
    - rider2
  - **Team**
    - name Repsol Honda Team
    - manufacturer Honda
      - rider1

- **Model**
  - name
  - templateTypes
    - declarations
  - Libraries
    - **TemplateTypes**
    - **ModelDeclarations**
      - **SimpleEntity**
        - modifiers
        - name
        - types
        - features
      - **Features**
        - modifiers
        - **FeatureModifiers**
          - type
          - name
        - **FeatureModifiers**
          - type
          - name
        - **FeatureModifiers**
          - type
          - name
        - **FeatureModifiers**
          - type
          - name
Example of uniformity given by a generic notation
MDE reduces a DSL into a library

• Forget the examples of multiple generic syntaxes
• MDE promotes a more conservative solution
  ✷ Implement the DSL as a library of a given base language
  ✷ Use the underlying “method invocation” syntax as a generic syntax for the DSL
  ✷ Retain the full power of the base language including tools (IDE) without effort
• This kind of solution is called *internal* DSL in contrast to *external* DSL that is a completely autonomous language
Model Driven Engineering is not enough

• Domain Specific Languages are not exploited by MDE
  ✷ More specificity is not achieved
  ✷ More commonality is limited to capabilities
  ✷ With Internal DSLs
    ▪ The specificity is limited by the syntax and semantics of the base language
    ▪ The base language must also be understood
Fundamental advantages of External DSLs

• All of the code is abstracted at the problem domain level
  ◦ Integration with technologies is a separate problem
  ◦ Code generators are easier to write and maintain

• DSLs are a direct coding of a domain knowledge
  ◦ Speak the same language as business people
  ◦ Show the code to clients for verification
  ◦ Involve business analysts in writing the code
LANGUAGE is the Language Oriented Programming’s answer

- For each domain introduce a domain specific language and use it to represent knowledge
- Build software around a set of domain specific languages
- Match existing domain notations

- Employ a Language Workbench tool to simplify the creation and use of new domain languages
Why use language as an abstraction?

• People are used to dealing with DSLs
  ✷ All cafes, pubs, restaurants have their own DSL
  ✷ All shops have their own DSL
  ✷ All sports have their own DSL
  ✷ All businesses have their own DSL

• People are really good at recognizing implicit context
  ✷ Sentences written with a DSL have an *implicit context*
  ✷ A domain specific notation can be much more *concise* with respect to a generic notation
Example of APIs vs. DSLs

```java
Coffee latte = new Coffee(Size.VENTI);
latte.setFatContent(FatContent.NON_FAT);
latte.setWhip(Whip.NONE);
latte.setFoam(Foam.NONE);
latte.setTemperature(Temp.EXTRA_HOT);
latte.setStrength(5);
```

Venti half-caf, non-fat, extra hot, no foam, no whip latte
LOP is a refinement over MDE

• Both MDE and LOP
  ♦ Put the model at the center of the DSL’s design
  ♦ Use a *Modeling Framework* to factor out all common behavior and obtain uniformity over DSLs

• MDE emphasizes *commonality*
  ♦ A general metamodel is enough (i.e. UML Class Diagram)
  ♦ Internal DSLs are enough (i.e. Java with modeling APIs)

• LOP emphasizes *specificity*
  ♦ The metamodel is specific to the domain
  ♦ Full External DSLs with domain specific notations are used
Modeling behavior

- So far, we have detailed the changes to software development due to modeling the structure of a software system.

- All of the semantics can be rephrased using model transformations.
  - To convince itself is just a matter of applying the MDE terminology to functions.
Generative Programming

• Both MDE and LOP deal also with modeling behavior

• ...
Outline

• Aim: Engineering the Production of Software
• Searching for a Common Denominator of Knowledge
• Looking for Answers within some Methodologies
  ♦ Object Oriented Programming
  ♦ Model Driven Engineering
  ♦ Language Oriented Programming
  ♦ Generative Programming
• From Methodologies to Technologies
• Whole Platform Overview
• Whole Platform Components
From Methodologies to Technologies

- MDE, LOP, and GP are just methodologies
- Technologies are needed to apply them
  - MDE requires a *Modeling Framework*
  - GP requires at least a model transformation language
  - LOP requires a full-fledged *Language Workbench*

- The *Whole Platform* aims to be a mainstream open source technology for application development involving these methodologies
Whole Platform Overview

• A *modeling framework* to concentrate development on the business knowledge factoring out technologies

• A *language workbench* to leverage the expertise required to participate in the development process

• A *software factory* to guide the organization of knowledge and processes to achieve systematic reuse
Whole Platform Components

• Whole Frameworks
  ♦ Common infrastructure for working on languages and tooling
  ♦ A Java library supporting Language Oriented Programming

• Whole Languages
  ♦ An extensible set of languages acting as a family
  ♦ Include Modeling, data integration, and transformation languages
  ♦ Also include popular languages and data formats

• Whole Language Workbench
  ♦ A development environment to program at the domain level
  ♦ Based on Eclipse
Whole Languages Map

Whole Language

Notations (concrete syntaxes)

Persistences (serialization syntaxes)

Capabilities (semantics)

Model (abstract syntax)

Libraries
- structure
- constraints

Grammars
- EBNF grammar

XSD - XML schema
RDB - Database schema
Pojo - Java API

Models

Components
- mappings
- rules

Queries
- model queries

Workflows
- processes

Tests
- unit testing

Actions
- tooling

Java
- Java 5

SQL
- database queries

Scripts
- Java 6 scripting

Editors
- text
- Grammars
- text
- Text
- XML
- XSD
- XML
- XML
- database
- RDB
- properties
- Properties
- file system
- Artifacts

modelled

coded
Whole Language Map