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)

```
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)

```
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)

```
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 99  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)

```
Ducati Marlboro Team;Ducati;Casey Stoner;1;Marco Melandri;33
Fiat Yamaha Team;Yamaha;Valentino Rossi;46;Jorge Lorenzo;48
Repsol Honda Team;Honda;Daniel Pedrosa;26;Nicky Hayden;69
Team Alice;Ducati;Tony Elias;24;Sylvain Guintoli;50
Honda Gresini;Honda;Shinya Nakano;56;Alex De Angelis;3
Yamaha Tech 3;Yamaha;Colin Edwards;5;James Toseland;52
Rizla Suzuki MotoGP;Suzuki;Chris Vermeulen;7;Loris Capirossi;65
Kawasaki Racing Team;Kavasaki;John Hopkins;21;Anthony West;13
Honda LCR;Honda;Randy De Puniet;14; ;0
Jir Scot Team;Honda;Andrea Dovizioso;3; ;0
```


What is the common denominator of knowledge? (FIN series 5 example)

```
{1:F01BCITITMMBXXX7798101545}{2:I535MSAMUS33XXXXN}{4:
:16R:GENL
:28E:1/ONLY
:20C::SEME//20050321HO000001
:23G:NEWM
:98A::STAT//20050321
:98A::PREP//20050321
:22F::STTY//CUST
:22F::SFRE//DAIL
:22F::CODE//COMP
:22F::STBA//SETT
:97A::SAFE//060236830200
:17B::ACTI//N
:17B::AUDT//N
:17B::CONS//N
:16S:GENL
-}{5:{MAC:CAF34EF9}{CHK:255950431A64}}
{1:F01BCITITMMBXXX7798101546}{2:I535PARBLULLXXXXN}{4:
:16R:GENL
:28E:2/MORE
:20C::SEME//20050321HO000004
:23G:NEWM
:98A::STAT//20050321
:98A::PREP//20050321
:22F::STTY//CUST
:22F::SFRE//DAIL
:22F::CODE//COMP
:22F::STBA//SETT
:97A::SAFE//073014120200
```


What is the common denominator of knowledge? (Manifest form example)

Overview

General Information

This section describes general information about this plug-in.

ID:	org.whole.lang.ui
Version:	1.0.0.v20081007-1544
Name:	%pluginName
Provider:	%providerName
Platform Filter:	
Activator:	org.whole.lang.ui.WholeUIPlugin [Browse...]

- [x] Activate this plug-in when one of its classes is loaded
- [x] This plug-in is a singleton

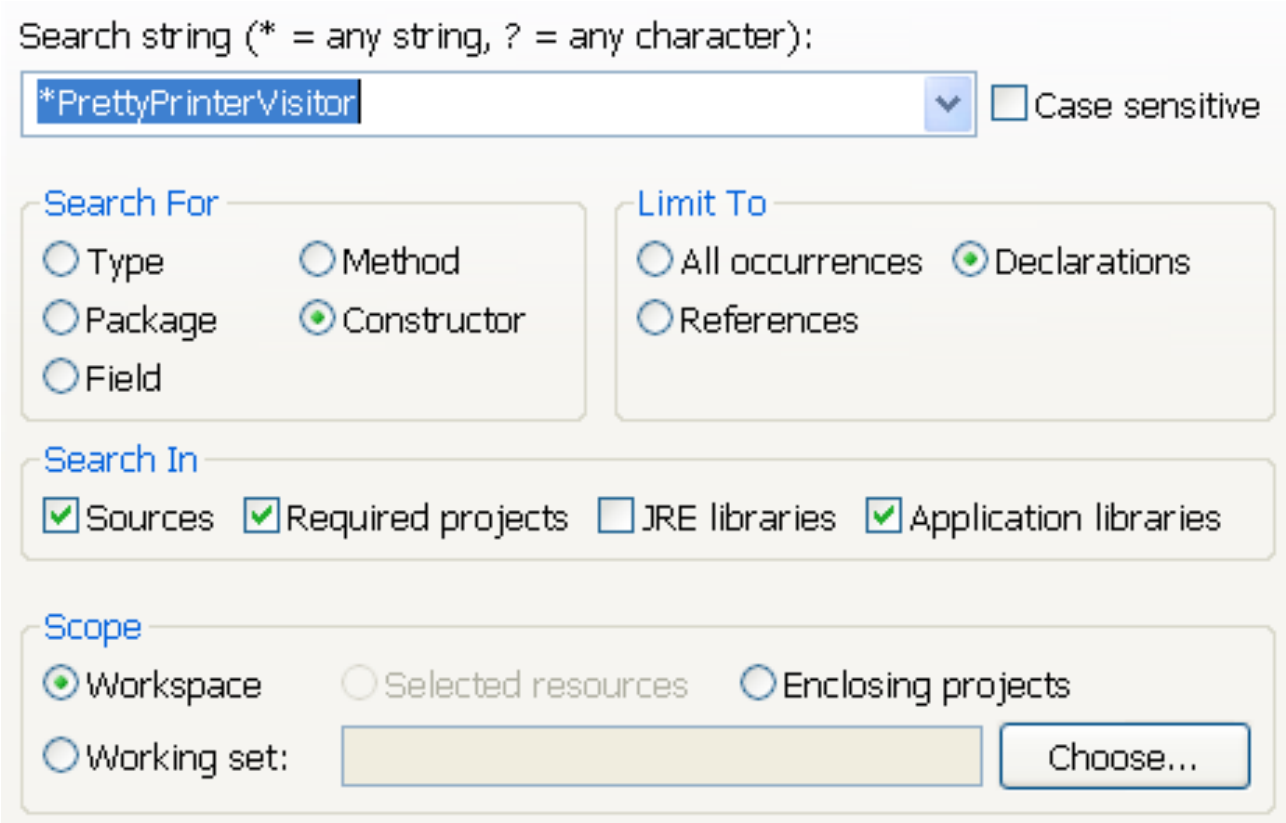
Execution Environments

Specify the minimum execution environments required to run this plug-in.

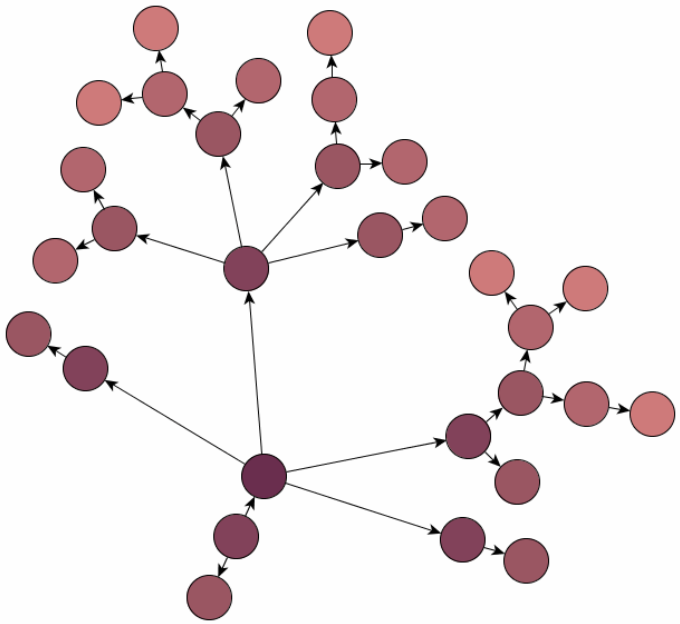
J2SE-1.5

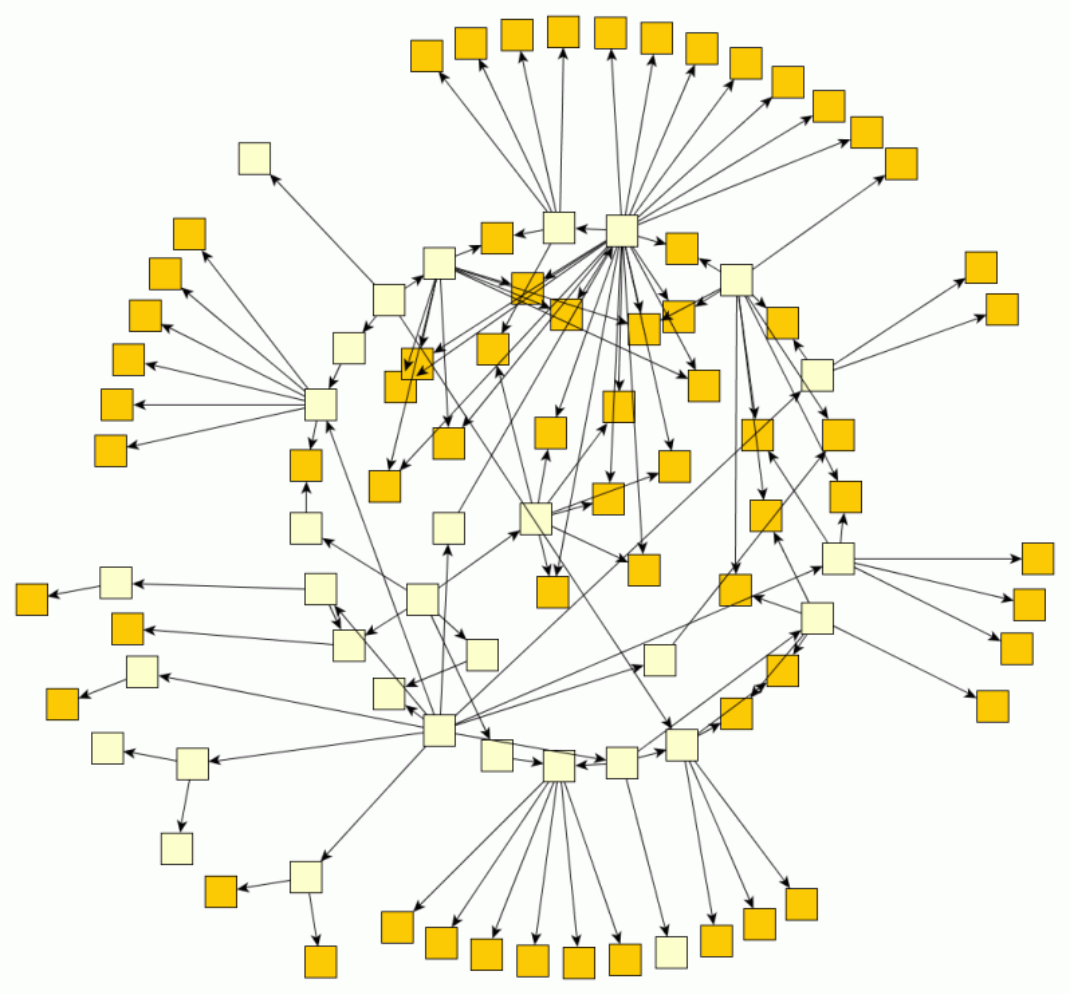
Add... | Remove | Up | Down

What is the common denominator of knowledge? (Search dialog example)

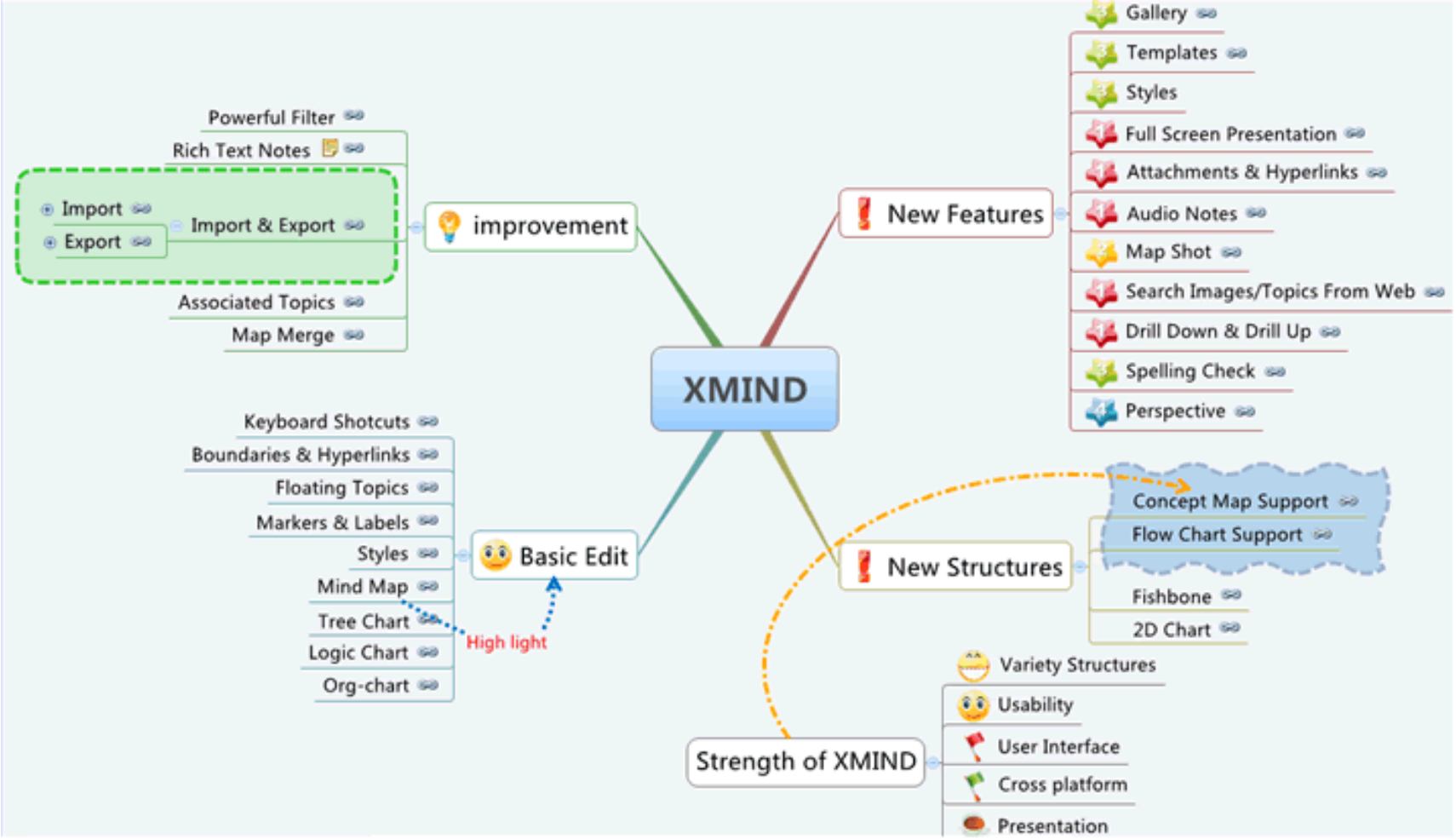


What is the common denominator of knowledge? (radial tree and circular layout graph examples)

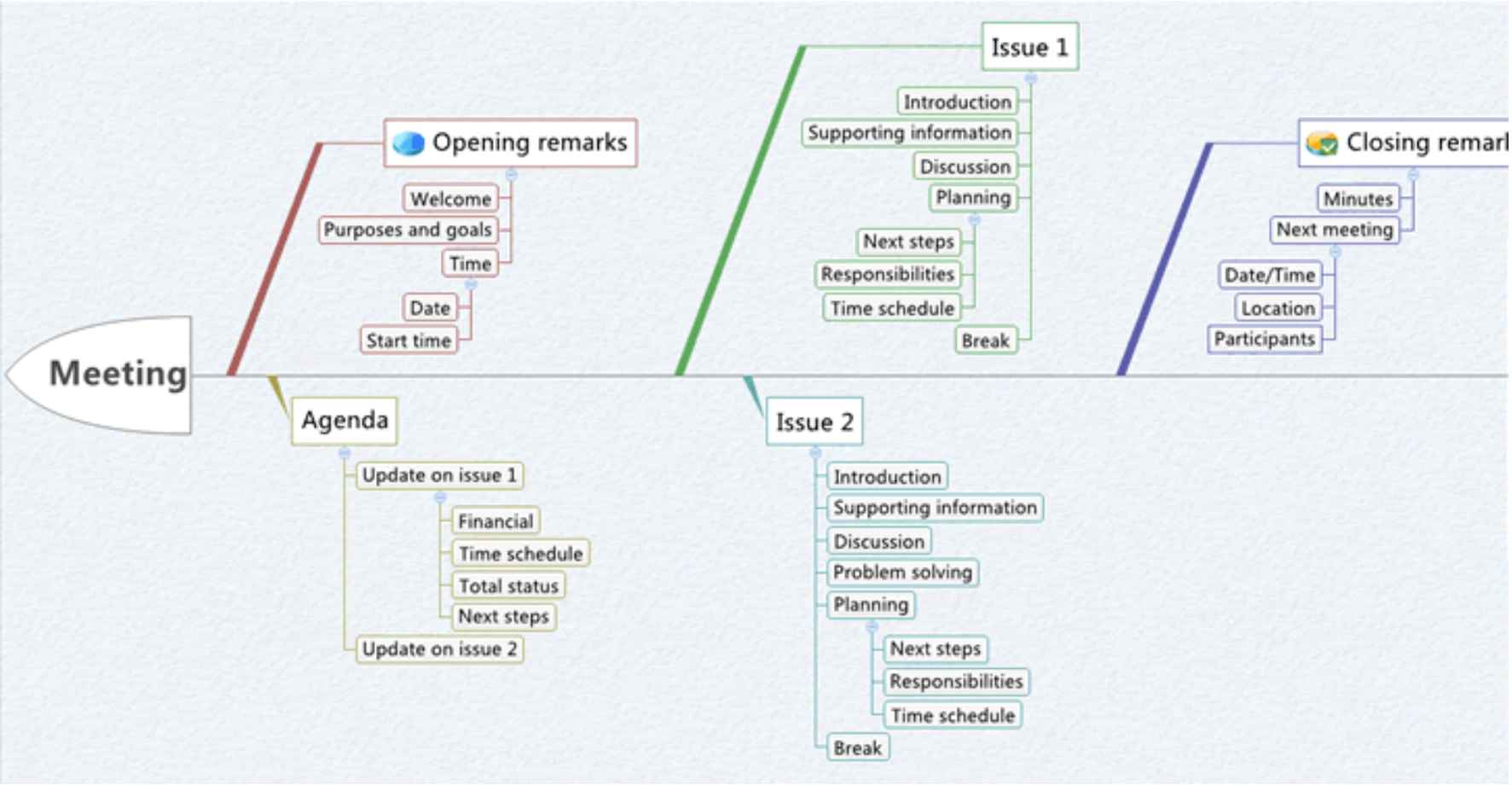




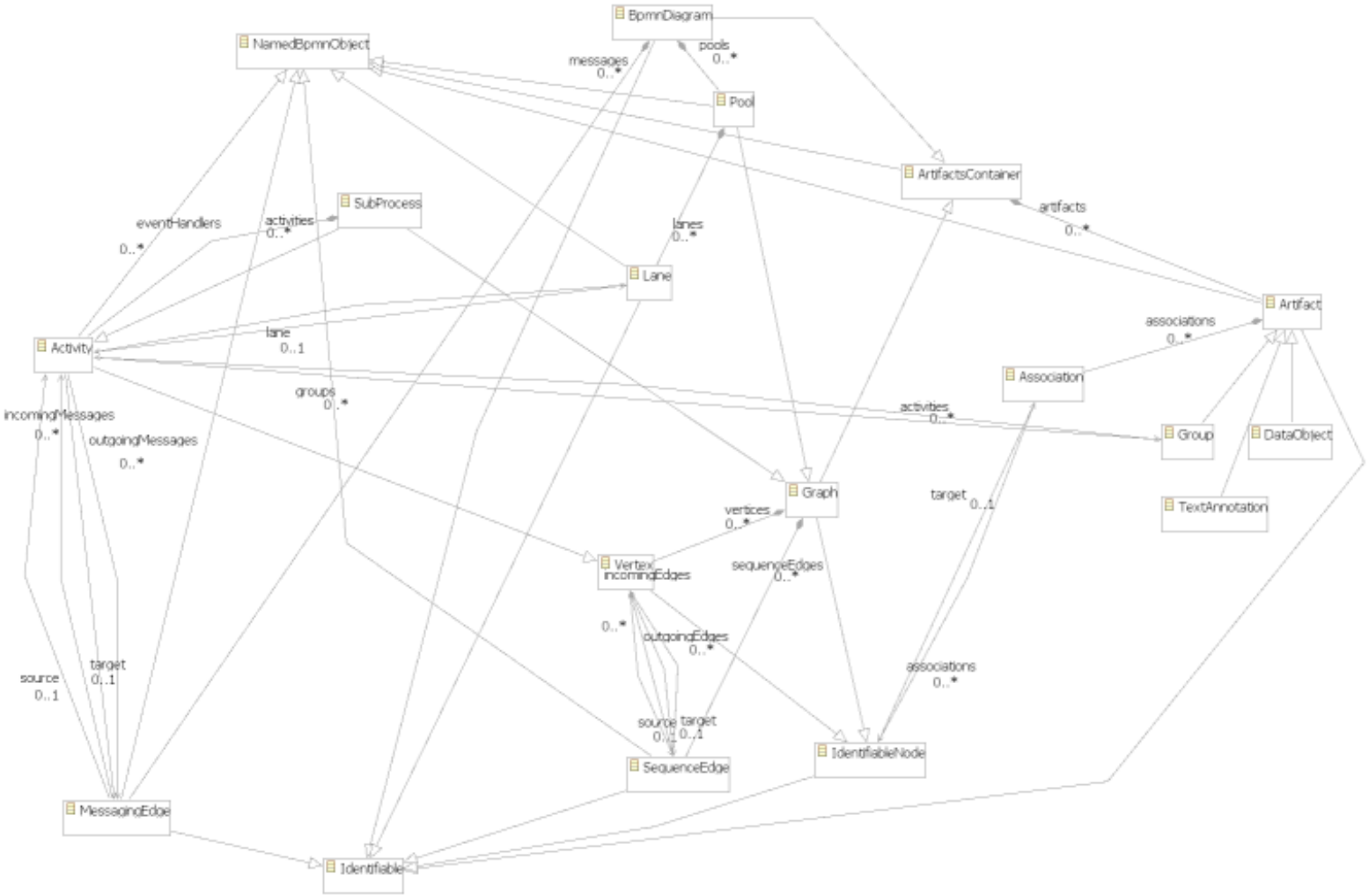
What is the common denominator of knowledge? (mind map example)



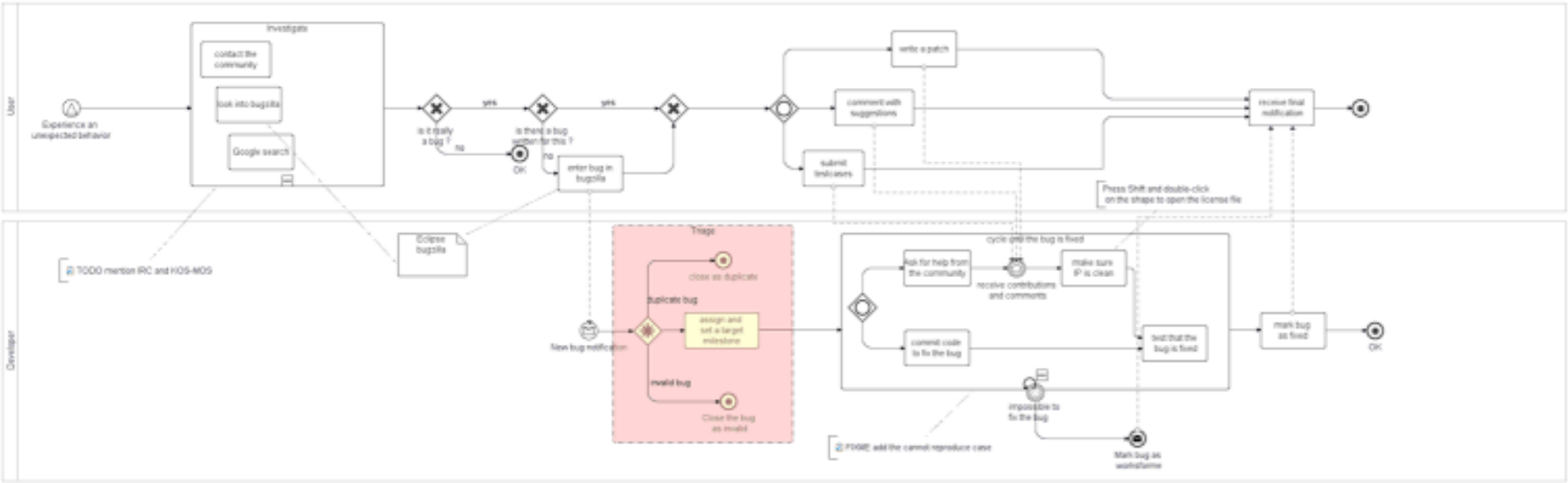
What is the common denominator of knowledge? (fishbone chart example)



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 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)

```
<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>
```


What is the common denominator of knowledge? (SVG XML example)

```
<?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)

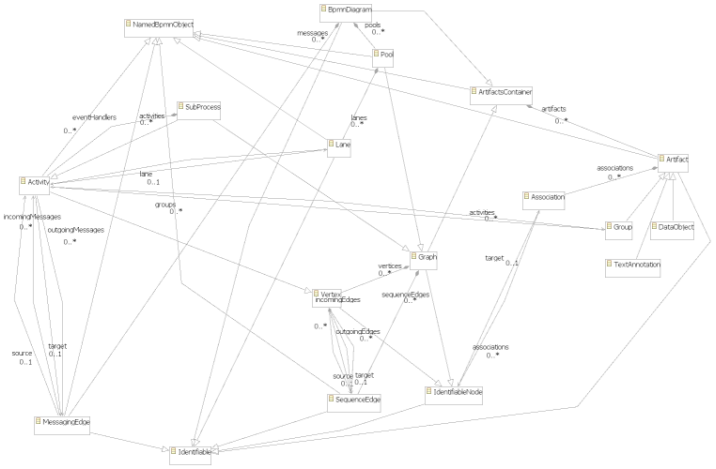
```
<?xml version="1.0" encoding="UTF-8"?>
<ns0:Document xmlns:ns0="urn:swift:xsd:$pacs.008.003.02" xmlns:xsi="http://
xsi:schemaLocation="urn:swift:xsd:$pacs.008.003.02 EPSCTPACS00800302M.xsd">
    <ns0:pacs.008.003.02>
        <ns0:CdtTrfTxInf>
            <ns0:PmtId>
                <ns0:InstrId>80131897065.0001</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:IntrBkSttlmAmt Ccy="EUR">2469.6</ns0:IntrBkSttlmAmt>
            <ns0:IntrBkSttlmDt>2007-12-15</ns0:IntrBkSttlmDt>
            <ns0:InstdAmt Ccy="EUR">123.23</ns0:InstdAmt>
            <ns0:ChrgBr>SLEV</ns0:ChrgBr>
            <ns0:ChrgsInf>
                <ns0:ChrgsAmt Ccy="EUR">123.45</ns0:ChrgsAmt>
                <ns0:ChrgsPty>
                    <ns0:FinInstnId>
                        <ns0:BIC>BCVSCH2L</ns0:BIC>
                    </ns0:FinInstnId>
                </ns0:ChrgsPty>
            </ns0:ChrgsInf>
```

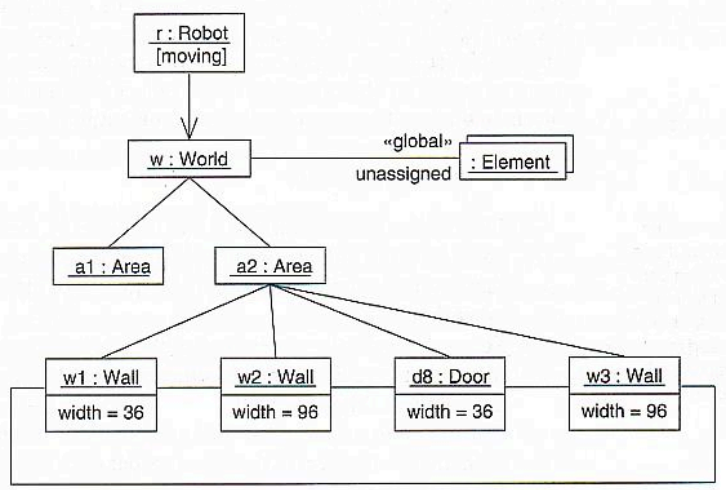

Outline

- 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

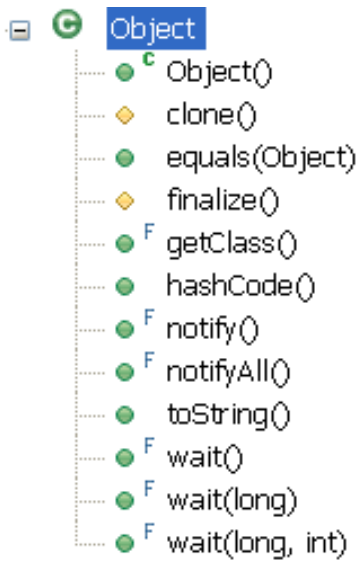
- Describe a Software System using a set of *classes*, their attributes, and the relationships between the classes
- An *Object* is an instance of a Class
- At runtime we *always* have "*aggregates of objects*"
- UML Class and Object diagrams are more abstract but help in visualizing the concepts





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 anwser

- 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

```
model            MotoGP
template types   ▫

▫ Teams: Team[ ordered ] <: ▫

▫ Team ⊟ <: ▫
    name          Name   ▫
    manufacturer  Name   ▫
    rider1        Rider  ▫
    rider2        Rider  ▫

▫ Rider ⊟ <: ▫
    name   Name   ▫
    number Number ▫

▫ Name: String <: ▫

▫ Number: int <: ▫
```

```
model            Java
template types   Expression

versionable JavaSystemSoftware: CompilationUnit[ unique ] <: ▫

versionable CompilationUnit ⊟ <: ▫
    package  PackageDeclaration  optional
    imports  ImportDeclarations  optional
    types    TypeDeclarations    ▫

▫ PackageDeclaration ⊟ <: ▫
    javadoc      Javadoc      optional
    annotations  Annotations  optional
    name         Name         ▫
```

```
▫ ForStatement ⊟ <: Statement
    initializers  Expressions  ▫
    expression    Expression   ▫
    updaters      Expressions  ▫
    body          Statement    ▫
```


Models and metamodels

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

```
abstract ModelDeclaration ⊟ <: ▫
    modifiers   EntityModifiers   optional
    name        SimpleName        id
    types       Types             optional

▫ SimpleEntity ⊟ <: ModelDeclaration
    features    Features          optional

▫ DataEntity ⊟ <: ModelDeclaration
    dataType    DataType          ▫
```

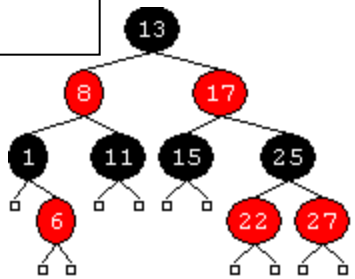
- (Model of) the RedBlackTree datatype

```
▫ RBNode ⊟ <: ▫
    value   Value    ▫
    color   Color    ▫
    left    RBNode   optional
    right   RBNode   optional

▫ Value: int <: ▫

▫ Color: ⊟ <: ▫
  BLACK, RED
```

- (Model of) an example of a RedBlack tree



- instance model of
- metamodel of

- instance model of
- metamodel of

- instance model of

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

```
package org.jug.test;

public class Hello {
    public static void main(String[] args) {
        System.out.println("Hello Java");
    }
}
```

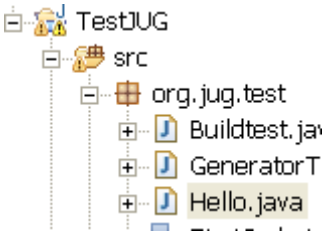
No standard

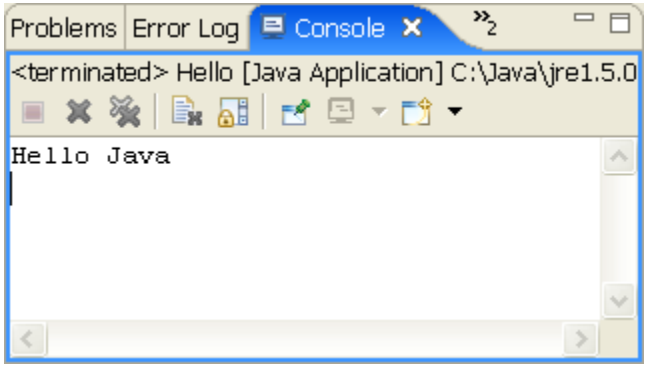
CONCRETE SYNTAX

ABSTRACT SYNTAX

SEMANTICS

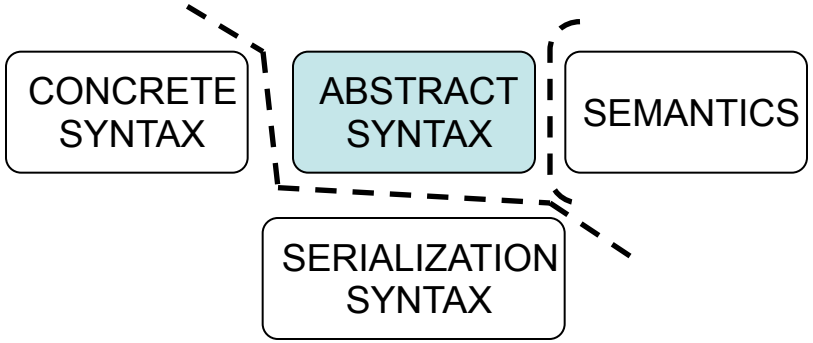
SERIALIZATION SYNTAX



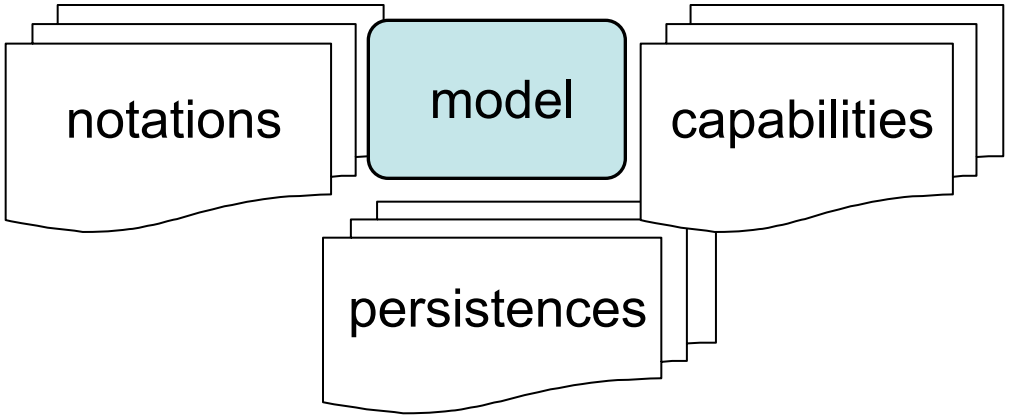


MDE reverses the language design focus

- 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

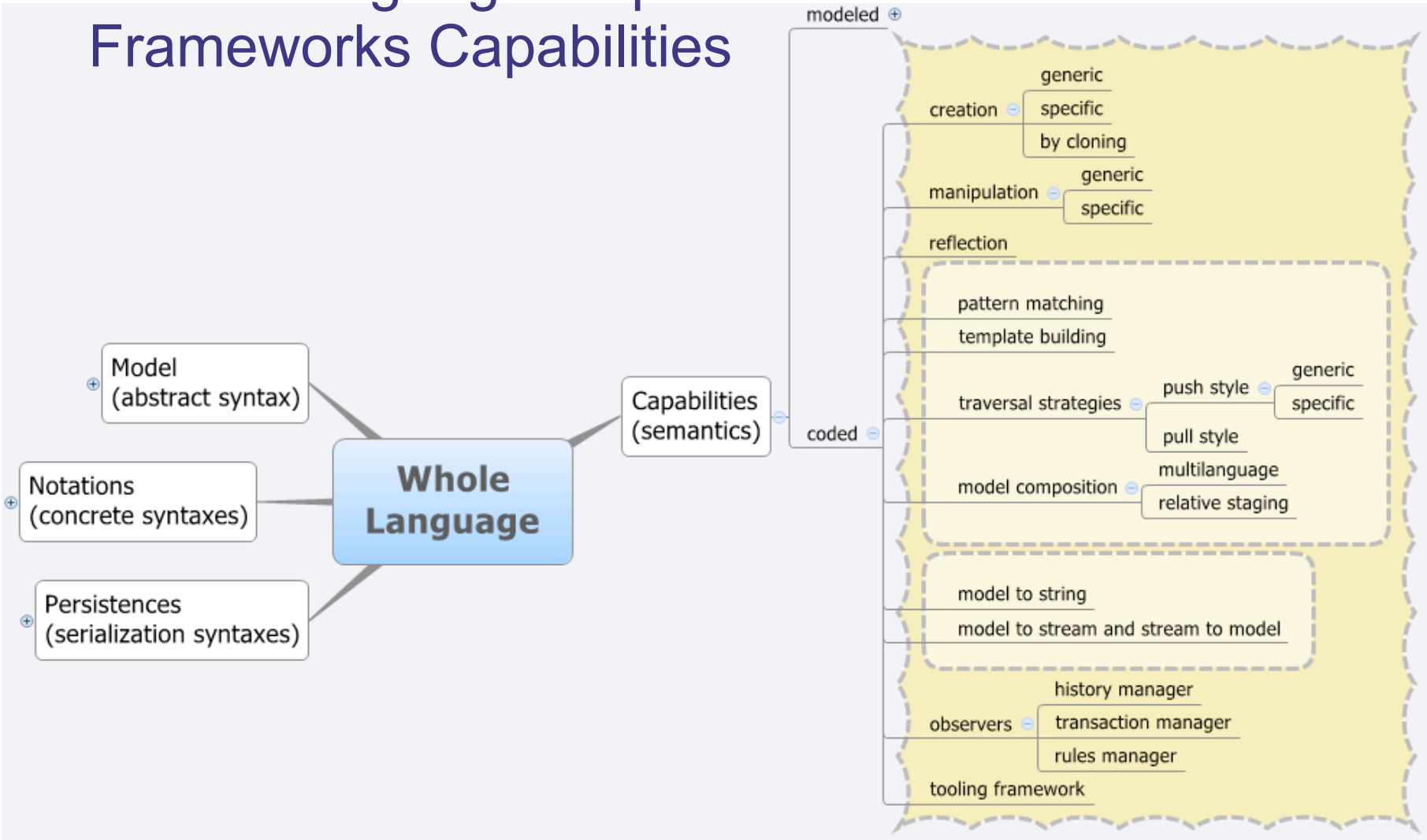
MDE maximizes the capabilities shared by all languages

- 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

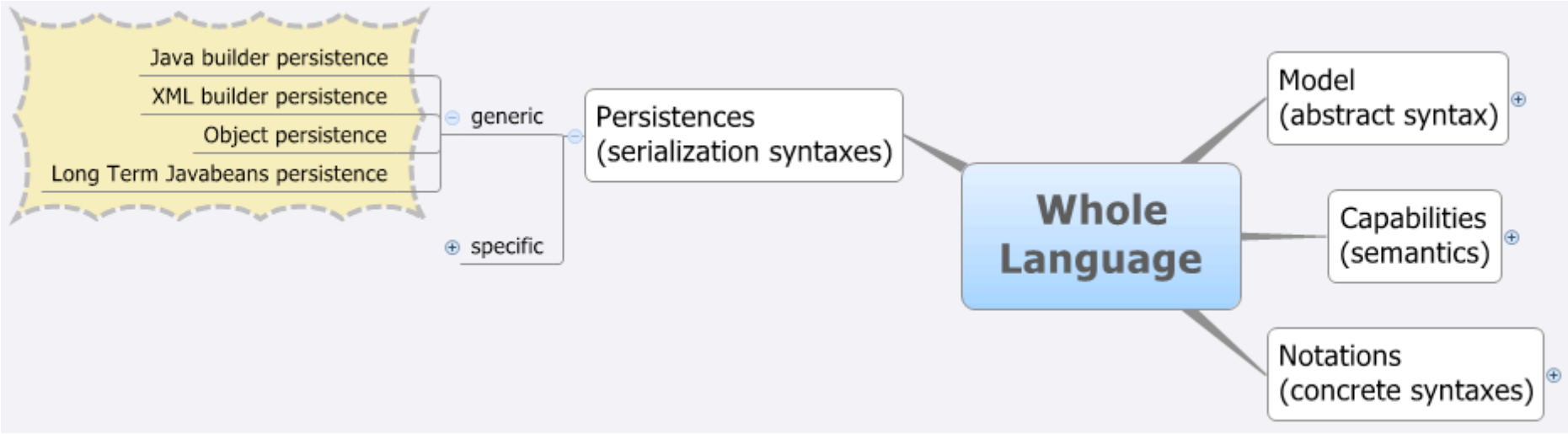
- With more out-of-the-box capabilities, the user can concentrate himself on business behavior

- With out-of-the-box persistences and notations, the user defines a model and obtains a language

Whole Language Map of Frameworks Capabilities



Whole Language Map of Generic Persistences



Example of the **Xml Builder** generic persistence

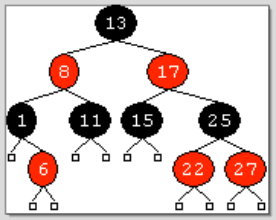
```
<?xml version =" 1.0 " encoding =" UTF-8 " ?>
▫
actions:LanguageActionFactory xmlns:actions="http://lang.whole.org/Actions"
  actions:Name value="RedBlackTreeActions"
  actions:URI value="http://datatypes.examples.whole.org/RedBlackTree"
  actions:ToolbarActions
    actions:SimpleAction
      actions:Label value="validator"
      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:Names initialCapacity="2"
              queries:Name value="message"
              queries:Name value="location"
            queries:Sequence initialCapacity="1"
              commons:SameStageFragment
                java:ExpressionStatement xmlns:java="http://lang.whole.org/Java"
                  java:MethodInvocation
                    java:SimpleName value="decorationManager"
```


Example of the **Java Builder** generic persistence

```
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.SameStageFragment_();
        IJavaBuilder b3 = (IJavaBuilder) op.wGetBuilder(JavaLanguageKit.ID);
        b3.ExpressionStatement_();
        b3.MethodInvocation_();
        b3.SimpleName("decorationManager");
        b3.Types();
```


Example of uniformity given by a generic persistence



MotoGP_Teams08_out.csv

```
Ducati Marlboro Team;Ducati;Casey Stoner;1;Marco Melandri;33
Fiat Yamaha Team;Yamaha;Valentino Rossi;46;Jorge Lorenzo;48
Repsol Honda Team;Honda;Daniel Pedrosa;26;Nicky Hayden;69
Team Alice;Ducati;Tony Elias;24;Sylvain Guintoli;50
Honda Gresini;Honda;Shinya Nakano;56;Alex De Angelis;3
```

pom.xml

```
<?xml version="1.0"?>
<project>
    <modelVersion>4.0.0</modelVersion>

    <groupId>
        it.nch.fwk.eai.servers.sepa.
    </groupId>
```

EntityUtils.java

```
public class EntityUtils {
    public static final boolean isSimple(EntityDescriptor<
        return ed.getEntityKind().isSimple();
    }
    public static final boolean isComposite(EntityDescript
        return ed.getEntityKind().isComposite();
    }
    public static final boolean isData(EntityDescriptor<?>
```

```
<redblacktree:RBNode xmlns:redblacktree="http://datatypes.e
    <redblacktree:Value value="13"/>
    <redblacktree:Color value="BLACK"/>
    <redblacktree:RBNode>
        <redblacktree:Value value="8"/>
        <redblacktree:Color value="RED"/>
        <redblacktree:RBNode>
```

MotoGP_Teams08_out.xml

```
<text:Document xmlns:text="http://lang.whole.org/Text" init:
    <text:Row initialCapacity="1">
        <text:Text value="Ducati Marlboro Team;Ducati;Casey
    </text:Row>
    <text:Row initialCapacity="1">
        <text:Text value="Fiat Yamaha Team;Yamaha;Valentino
```

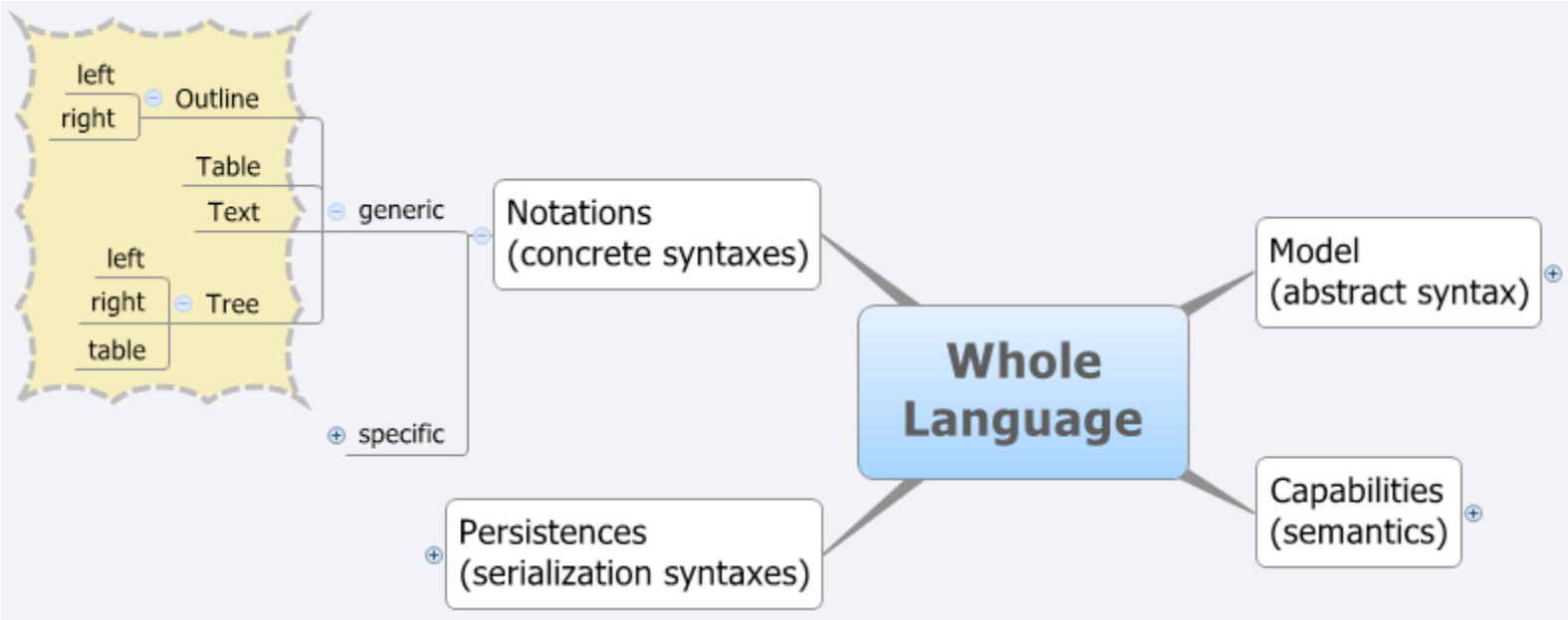
pom_xml.xml

```
<wxml:Document xmlns:wxml="http://lang.whole.org/Xml">
    <wxml:Prolog>
        <wxml:XMLDecl>
            <wxml:Version value="1.0"/>
            <wxml:Encoding value="UTF-8"/>
            <wxml:Standalone value="false"/>
        </wxml:XMLDecl>
```

EntityUtils.xml

```
<java:CompilationUnit xmlns:java="http://lang.whole.org/Java
    <java:PackageDeclaration>
        <commons:Resolver xmlns:commons="http://lang.whole.
        <commons0:Resolver xmlns:commons0="http://lang.whole
        <java:QualifiedName value="org.whole.examples.notat:
    </java:PackageDeclaration>
    <java:ImportDeclarations initialCapacity="13">
```


Whole Language Map of Generic Notations



Examples of the **Table** generic notation

Model

name	Libraries
templateTypes	▫
declarations	**SimpleEntity**

SimpleEntity

modifiers	▫
name	Library
types	▫
features	**Features**

Features

modifiers	type	name
▫	URI	uri
▫	Name	name
▫	Version	version
optional	Tags	tags
optional	Imports	imports
▫	Rules	rules

CompositeEntity

modifiers	▫
name	Tags
types	▫
componentModifiers	ordered
componentType	Name

Grammar

name	MotoGP_TeamsCSVGrammar
targetLanguage	http://dataintegration.examples.whole.org/MotoGP
startSymbol	Teams
delimiter	▫
lexicalStructure	**Productions**

Productions (lexicalStructure)

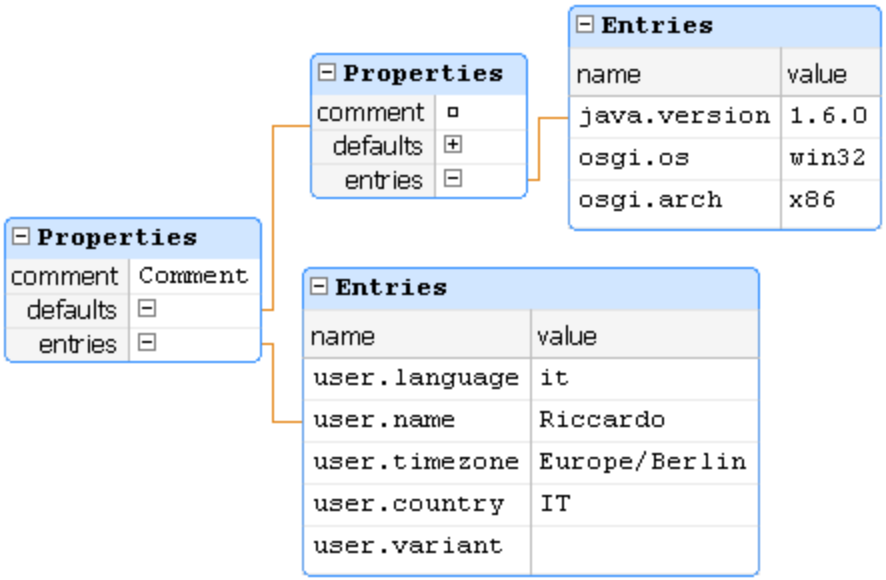
name	rule	template
NL	**LiteralTerminal**	▫
SEP	**LiteralTerminal** pattern: ; category: delimiter literal: ;	▫

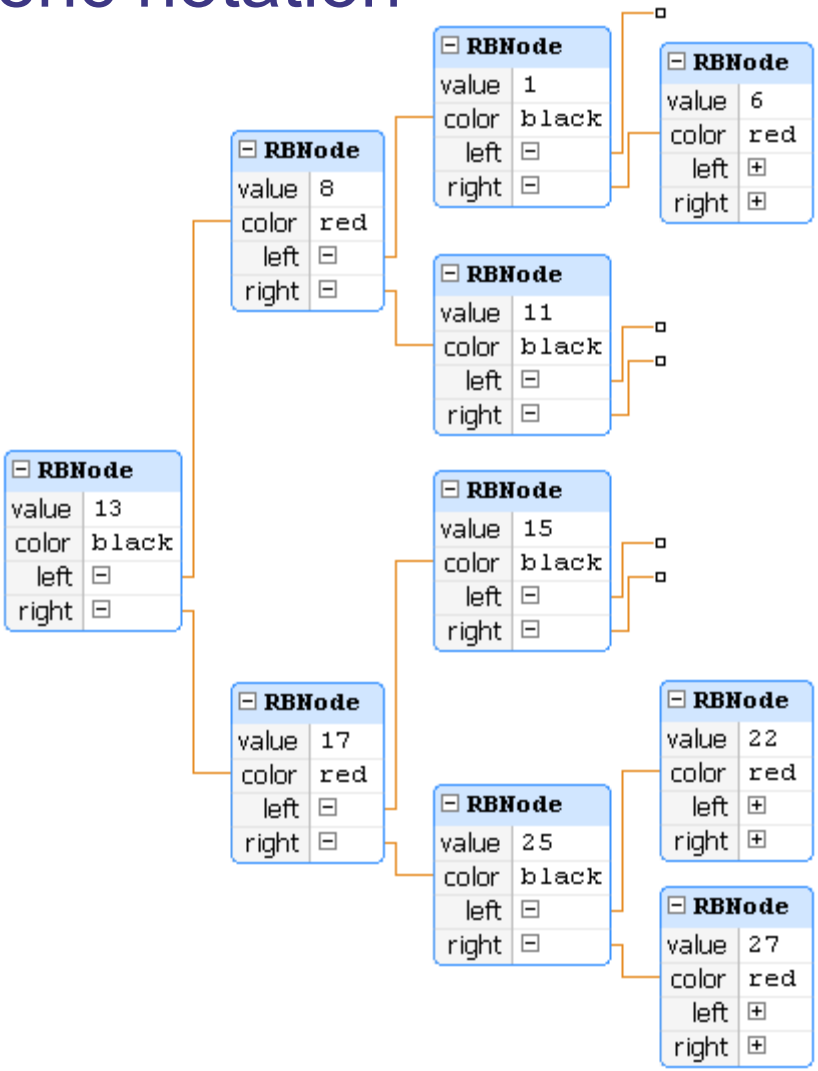
phraseStructure	**Productions**

Productions (phraseStructure)

name	rule	template
Teams	**Repeat** lowerBound: 2 upperBound: **Unbounded** separator: NL rule: **Production**	▫
Rider	**As** rule: Name name: name SEP	▫

Examples of the **Tree** generic notation



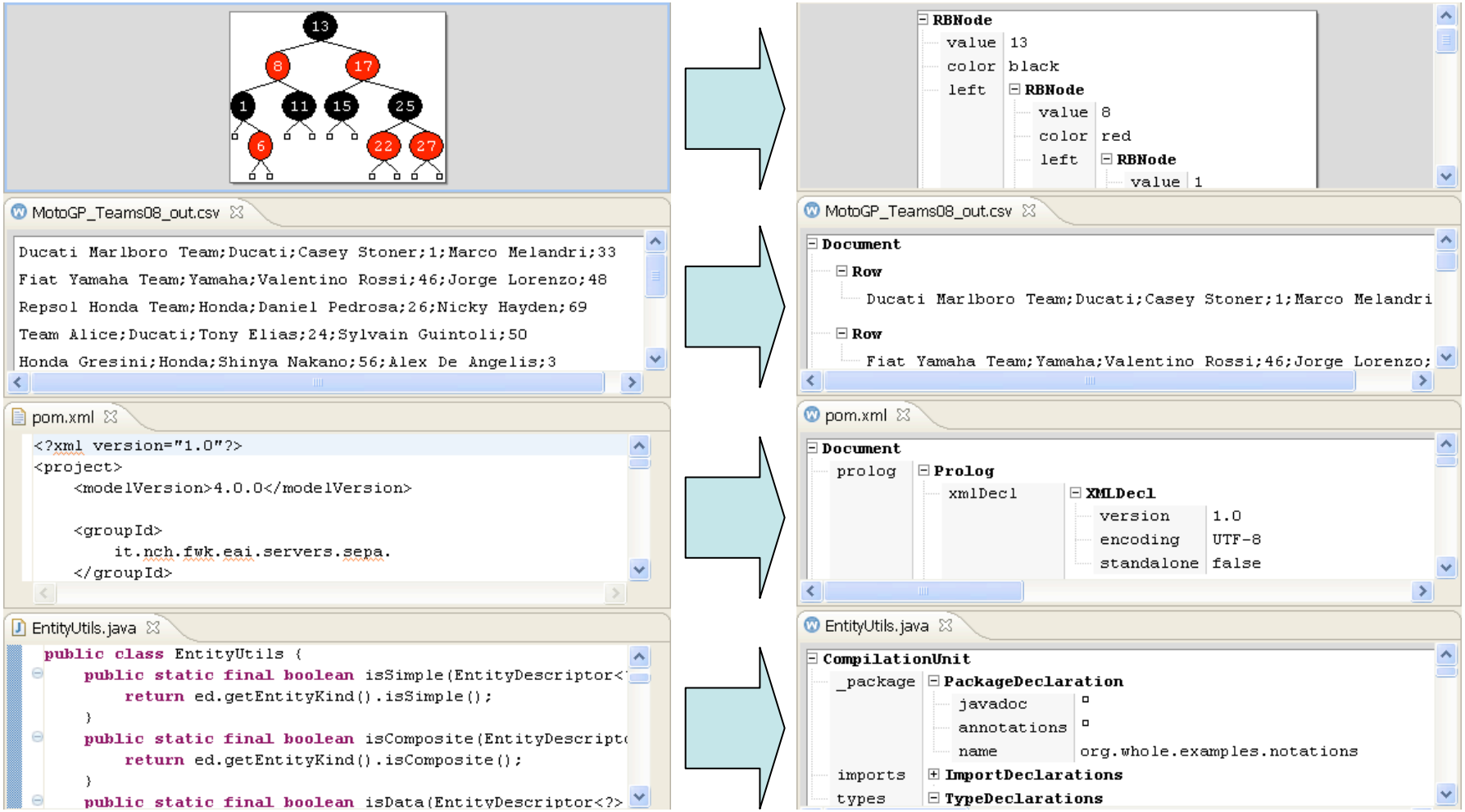


Examples of the **Outline** generic notation

- **Teams**
  - **Team**
    - name: Ducati Marlboro Team
    - manufacturer: Ducati
    - rider1: **Rider**
      - name: Casey Stoner
      - number: 1
    - rider2: **Rider**
      - name: Marco Melandri
      - number: 33
  - **Team**
    - name: Fiat Yamaha Team
    - manufacturer: Yamaha
    - rider1: **Rider**
      - name: Valentino Rossi
      - number: 46
    - rider2: **Rider**
      - name: Jorge Lorenzo
      - number: 48
  - **Team**
    - name: Repsol Honda Team
    - manufacturer: Honda
    - rider1: **Rider**
      - name: Daniel Pedrosa
      - number: 26

- **Model**
  - name: Libraries
  - templateTypes: **TemplateTypes**
  - declarations: **ModelDeclarations**
    - **SimpleEntity**
      - modifiers: **EntityModifiers**
      - name: Library
      - types: **Types**
      - features: **Features**
        - **Feature**
          - modifiers: **FeatureModifiers**
          - type: URI
          - name: uri
        - **Feature**
          - modifiers: **FeatureModifiers**
          - type: Name
          - name: name
        - **Feature**
          - modifiers: **FeatureModifiers**
          - type: Version
          - name: version

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

```
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

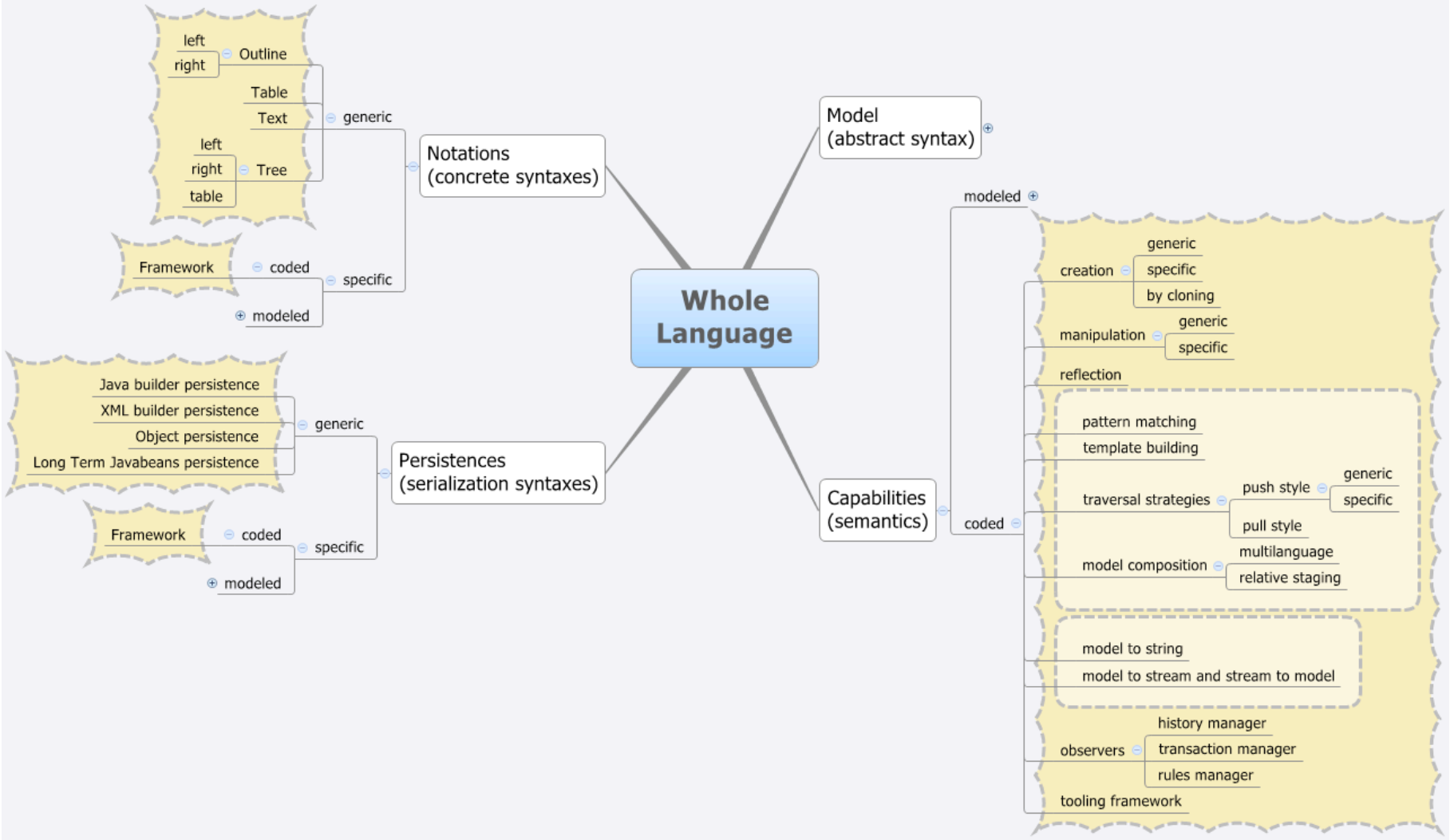
Whole Platform Overview

- 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

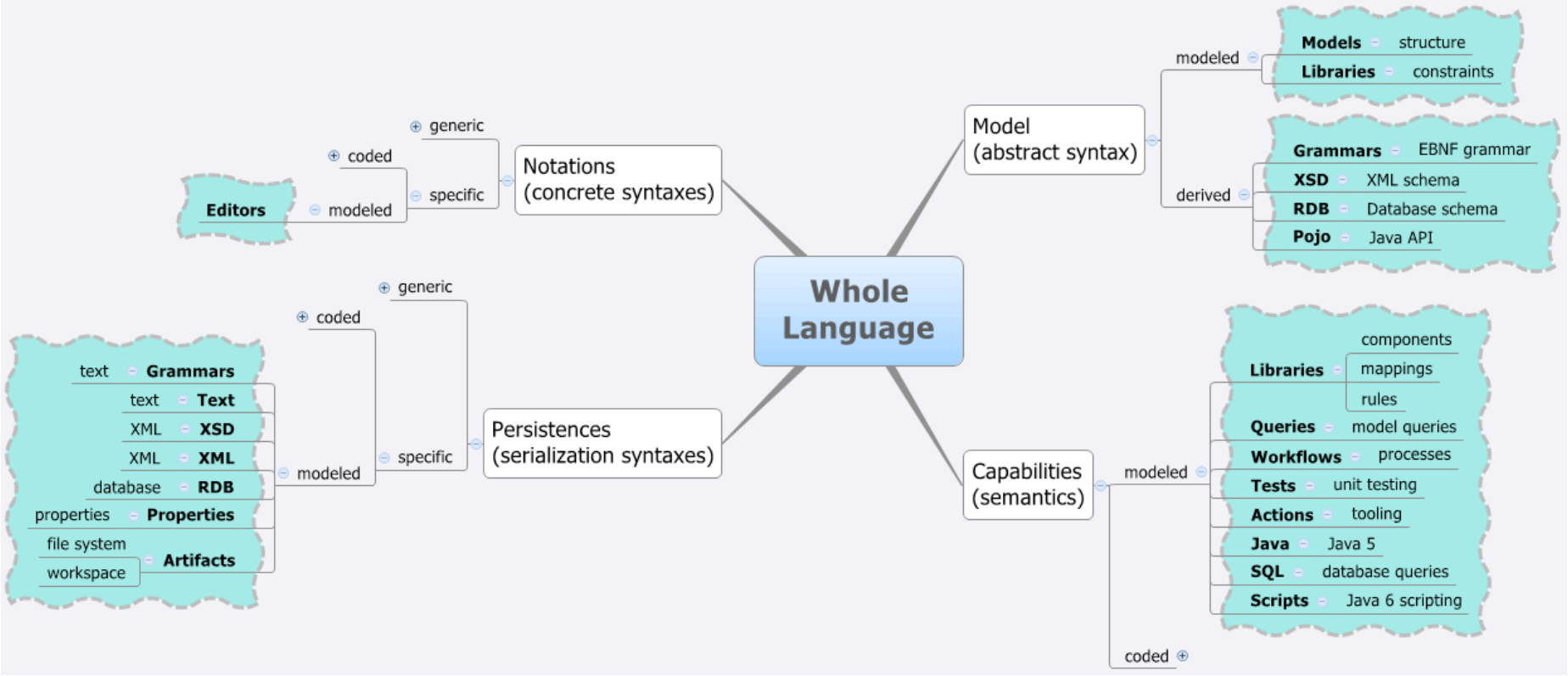
Whole Platform Components

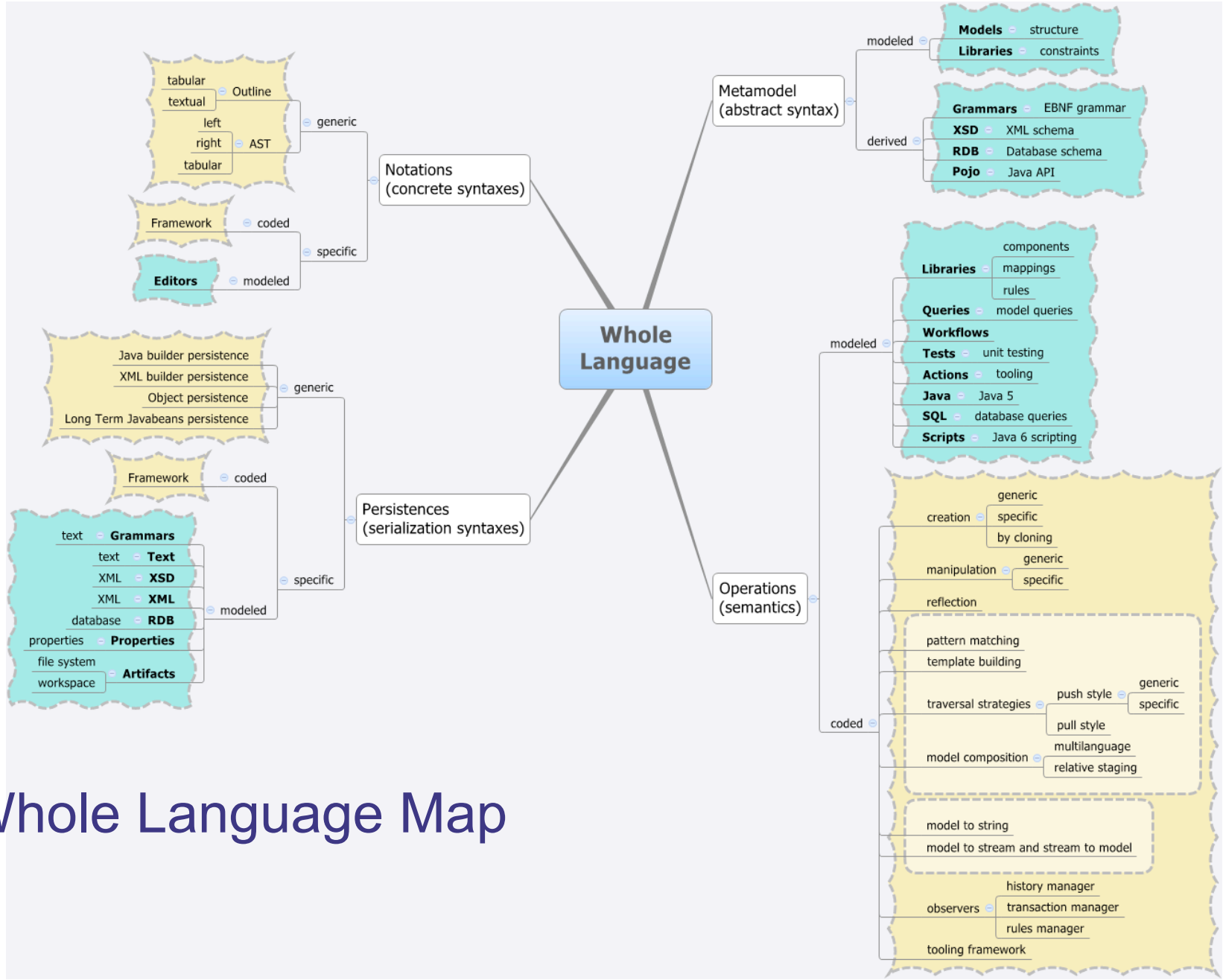
- Whole Frameworks
  - Common infrastructure for working on languages and tooling
  - A Java library supporting Language Oriented Programming
- Whole Languages
  - An extensible set of languages acting as a family
  - Include Modeling, data integration, and transformation languages
  - Also include popular languages and data formats
- Whole Language Workbench
  - A development environment to program at the domain level
  - Based on Eclipse

Whole Frameworks Map



Whole Languages Map





Whole Language Map