Introduction to Model-Driven Software Development

Juan de Lara, Elena Gomez, Esther Guerra
{Juan.deLara, MariaElena.Gomez, Esther.Guerra}@uam.es

Escuela Politécnica Superior
Universidad Autónoma de Madrid

Masters: I²TIC and formal methods (shared)

Index

- **Introduction.**

- Concepts and Terminology.
- Technology.
- Bibliography and suggested readings.

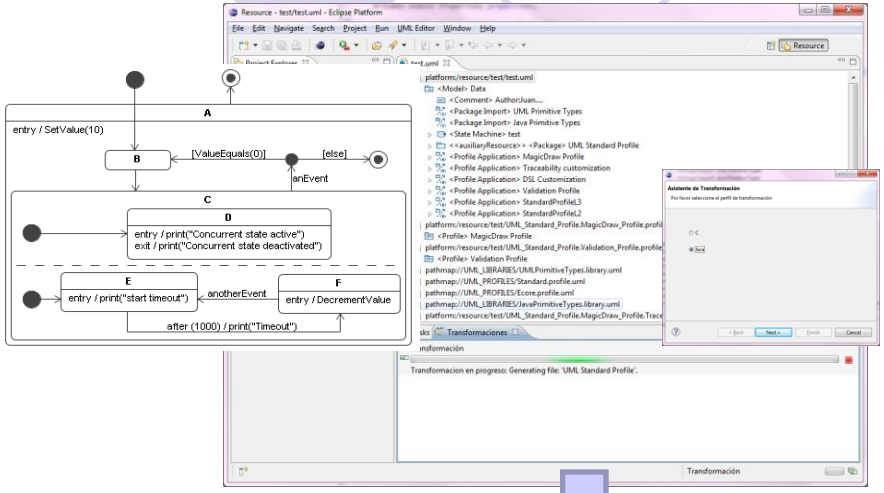
Introduction

- Develop software with higher levels of quality, and productivity.
- Increase the level of abstraction: models.
- Less "*accidental*" details, notations closer to the problem.
- Models are not just documentation: they are used to generate code for part or all the final application.
- Specific, well understood domains.

Introduction

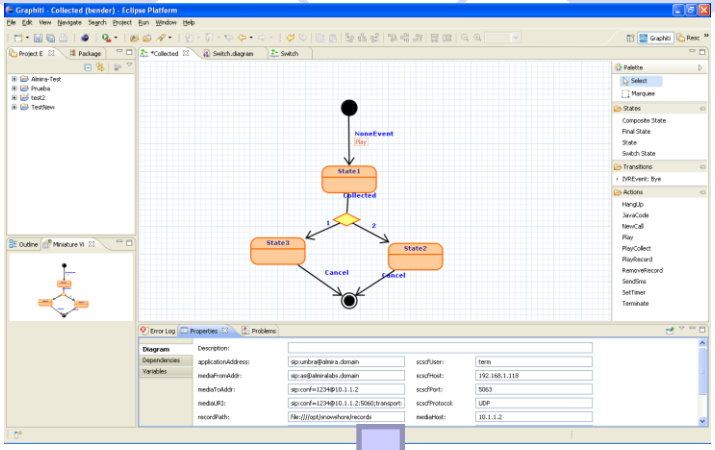
- Product lines, application families.
- Avoid coding the same solutions every time.
- Concrete application area.
- Reuse: Domain Specific Language +
  Architecture +
  Code generation (or configuration).

Examples



Code Generator

- Generation of code from state machines for railway signalling software.



Code Generator

- Modelling, validation and code generation for telephony systems.

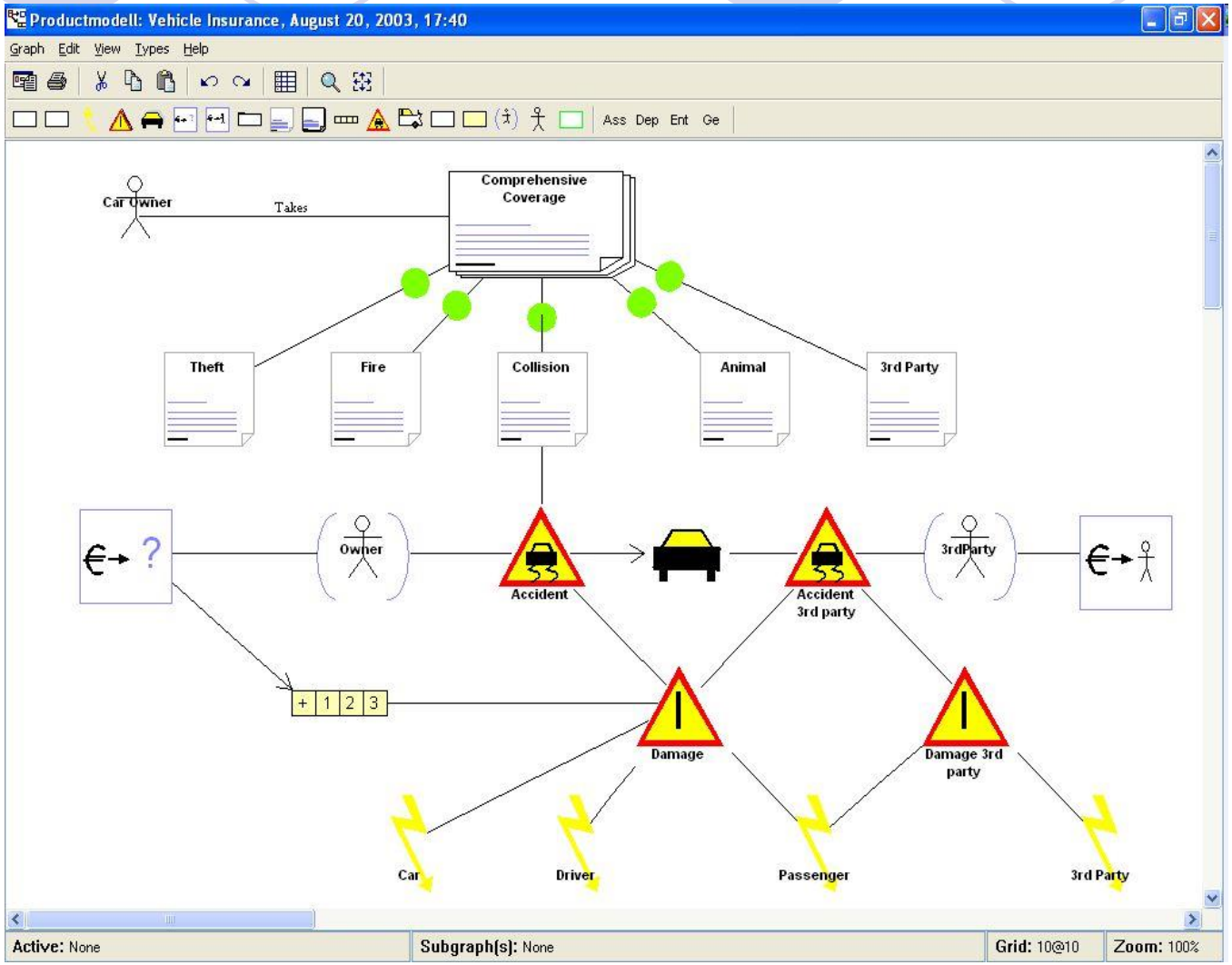
Index

- Introduction.
- **Concepts and Terminology.**
  - MDSD and DSLs.
  - Transformations.
  - Product Lines.
  - MDA.
  - Language-Oriented Programming.
  - Low-code Development.
- Technology.
- Bibliography and suggested readings.

MDSD and DSLs

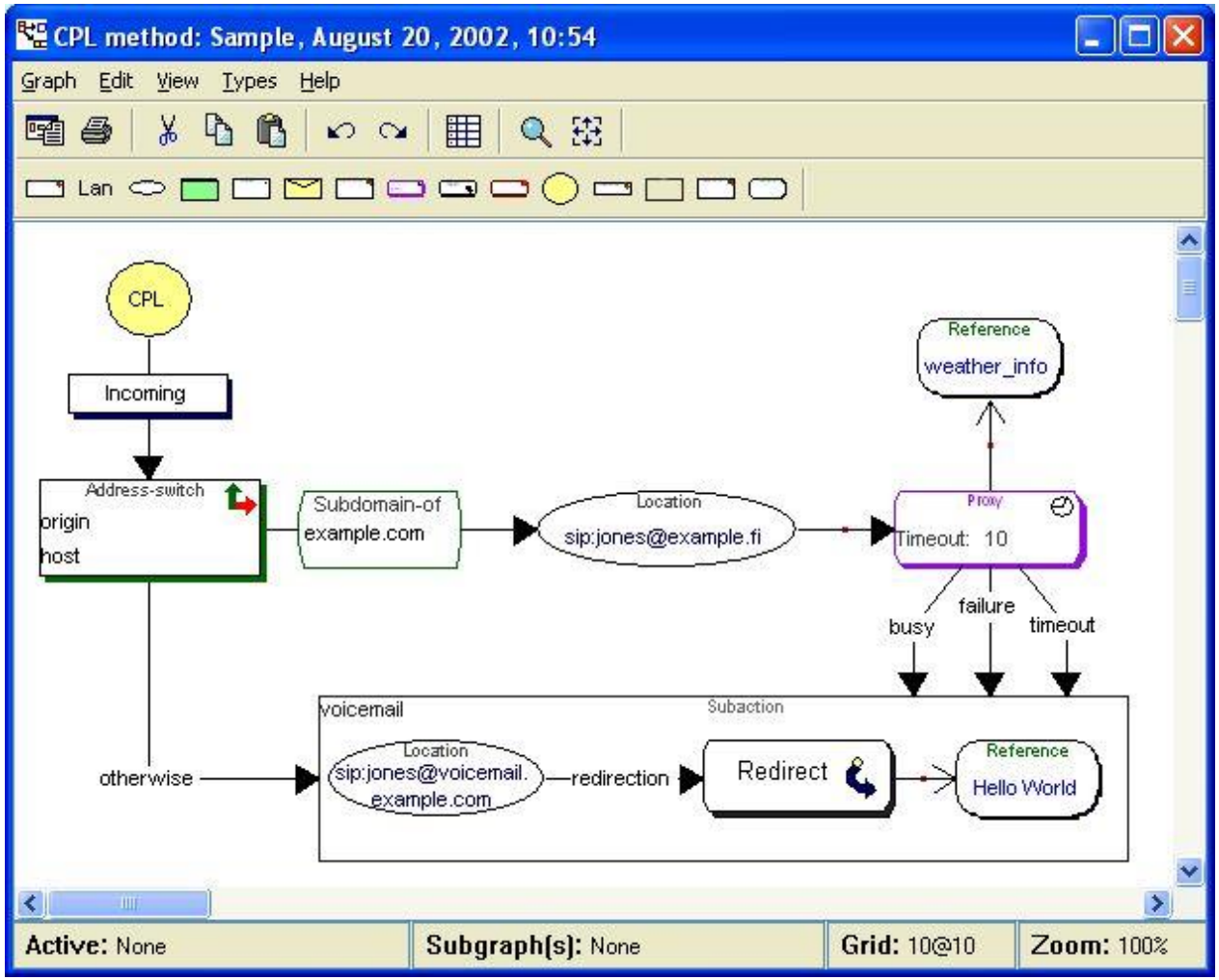
- Model Driven Software Development (MDSD) is based on the concept of Domain-Specific Language (DSL).
- The DSL offers the concepts of an application domain with which we are going to work.
- It allows identifying high-level primitives, gathering knowledge of domain experts.
- For restricted application domains it is possibe to generate 100% of the application code.

DSL Examples



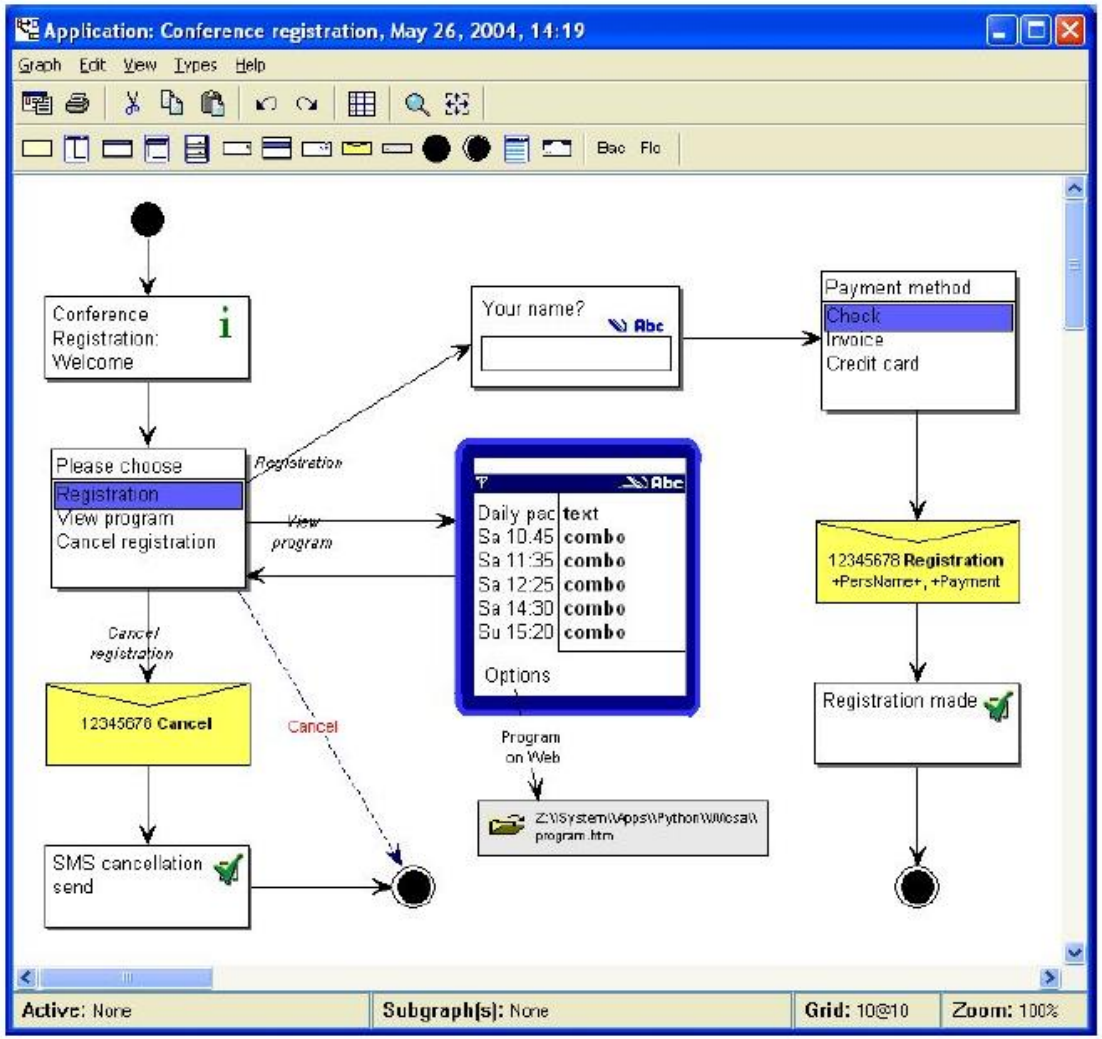
Insurance / J2EE. MetaEdit+ Tool.

DSL Examples



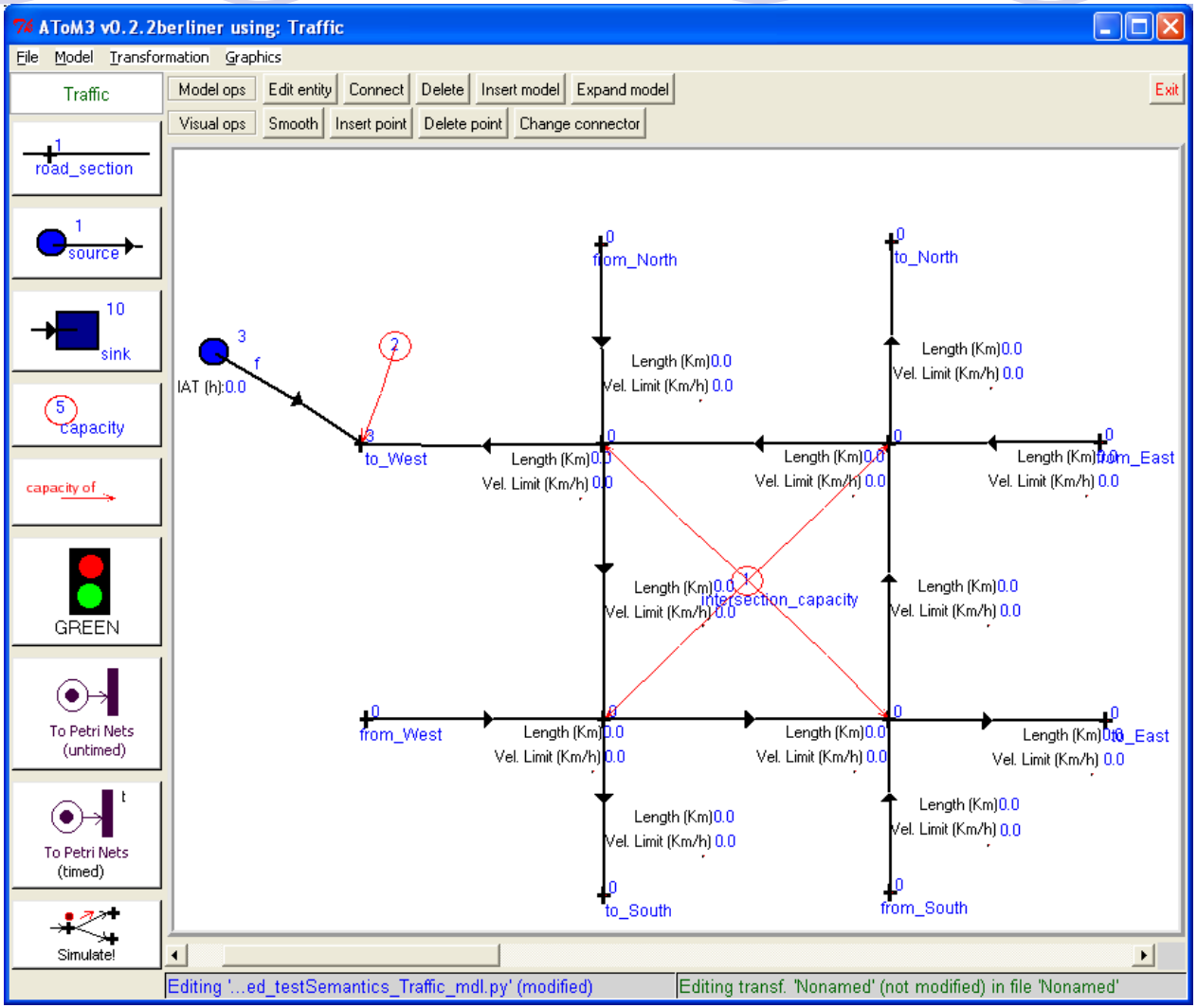
Internet telephony / CPL. MetaEdit+ tool.

DSL Examples



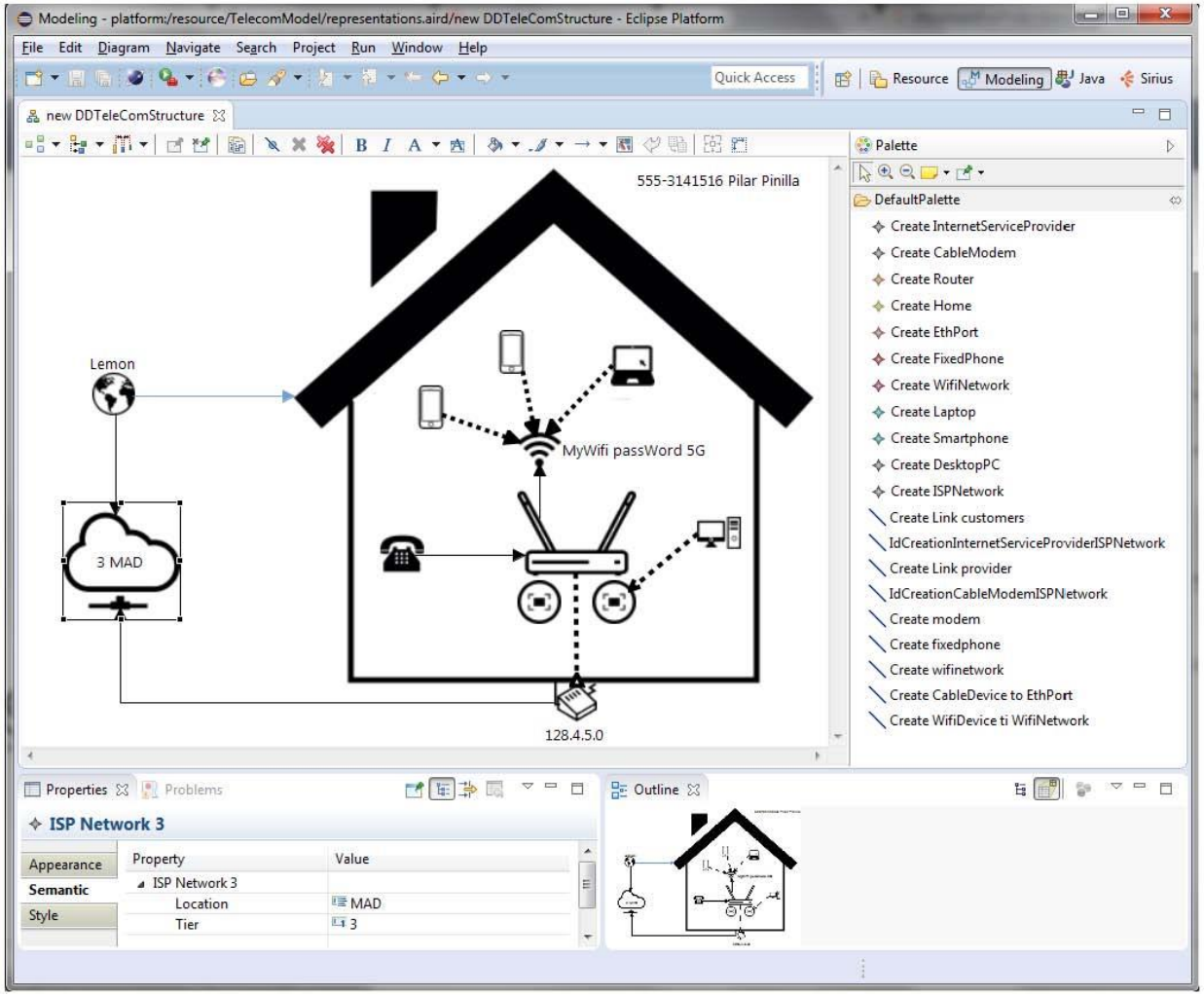
SmartPhones/Python. MetaEdit+ Tool.

DSL Examples



Road network / Petri nets. AToM^3 tool.

DSL Examples



Home networking / EMF-Splitter / Sirius

No need to be graphical…

Java – Test/test.analysis – Eclipse SDK

test.analysis

```
UseCase WithdrawMoney
    handles BankAccount in modification
            Customer in consultation
    references RM1
    Postcondition : 'Customer has withdrawn money'
    Primary Scenario :
        Step S_1 : user -> 'Introduces his card into ATM system'
        Step S_2 : system -> 'Asks PIN code'
        Step S_3 : user -> 'Enters PIN code'
        Step S_4 : system -> 'Checks PIN code'
        Step S_5 : system -> 'Asks operation'
        Step S_6 : user -> 'Selects withdrawal'
        Step S_7 : system -> 'Asks amount to withdraw'
        Step S_8 : user -> 'Selects an amount'
        Step S_9 : system -> 'Checks that bank account has sufficiant money'
        Step S_10 : system -> 'Gives money'
        Step S_11 : system -> 'Gives back the bank card'

    Alternative Scenario 'Wrong PIN code'
            Diverge from S_4 Converge to S_2 :
        Step S_4a_1 : system -> 'Alerts that PIN code is incorrect'

    Alternative Scenario 'Insufficiant balance'
            Diverge from S_9 Converge to S_7 :
        Step S_9b_1 : system -> 'Alerts that account balance is insufficiant'

    Exception Scenario 'Cancel operation'
            Diverge from S_7 :
        Step S_7Xa_1 : user -> 'Cancels operation'

UseCase ConsultBalance
    handles Customer in consultation
            BankAccount in consultation
    Primary Scenario :
        Step S_1 : user -> 'Introduces his card into ATM system'
        Step S_2 : system -> 'Asks PIN code'
        Step S_3 : user -> 'Enters PIN code'
        Step S_4 : system -> 'Checks PIN code'
        Step S_5 : system -> 'Asks operation'
        Step S_6 : user -> 'Selects consult balance'
        Step S_7 : system -> 'Shows balance'
```

Writable | Insert

No need to be graphical…

Java - Test/test.analysis - Eclipse SDK

test.analysis

```
UseCase WithdrawMoney
    handles BankAccou
            Customer
    references RM1
    Postcondition : '
    Primary Scenario
        Step S_1 : us
        Step S_2 : sy
        Step S_3 : us
        Step S_4 : sy
        Step S_5 : sy
        Step S_6 : us
        Step S_7 : sy
        Step S_8 : us
        Step S_9 : sy
        Step S_10 : s
        Step S_11 : s

    Alternative Scena
            Diverge f
        Step S_4a_1 :

    Alternative Scena
            Diverge f
        Step S_9b_1 :

    Exception Scenari
            Diverge f
        Step S_7Xa_1

UseCase ConsultBalanc
    handles Customer
            BankAccou
    Primary Scenario
        Step S_1 : us
        Step S_2 : sy
        Step S_3 : us
        Step S_4 : sy
        Step S_5 : sy
        Step S_6 : use
        Step S_7 : system -> 'Shows balance'
```

Java - my-home/src/Home.rules - Eclipse SDK

Package Ex

- my-home
  - src
    - Home.rules
  - JRE System Library [
  - src-gen

Home.rules

```
Device Window can be OPEN, SHUT
Device Heating can be ON, OFF

Rule 'Close Window, when heating turned on'
  when Heating.ON
  then Window.SHUT

Rule 'Switch off heating, when windows gets opened'
  when Window.OPEN
  then Heating.OFF
```

- OFF - Heating.OFF
- ON - Heating.ON
- OPEN - Window.OPEN
- SHUT - Window.SHUT

State **OFF**

Press 'F2' for focus

Outline

- Home
  - Window
  - Heating
  - Close Window, when
  - Switch off heating, w

Problems | Javadoc | Declaration | Search | Console

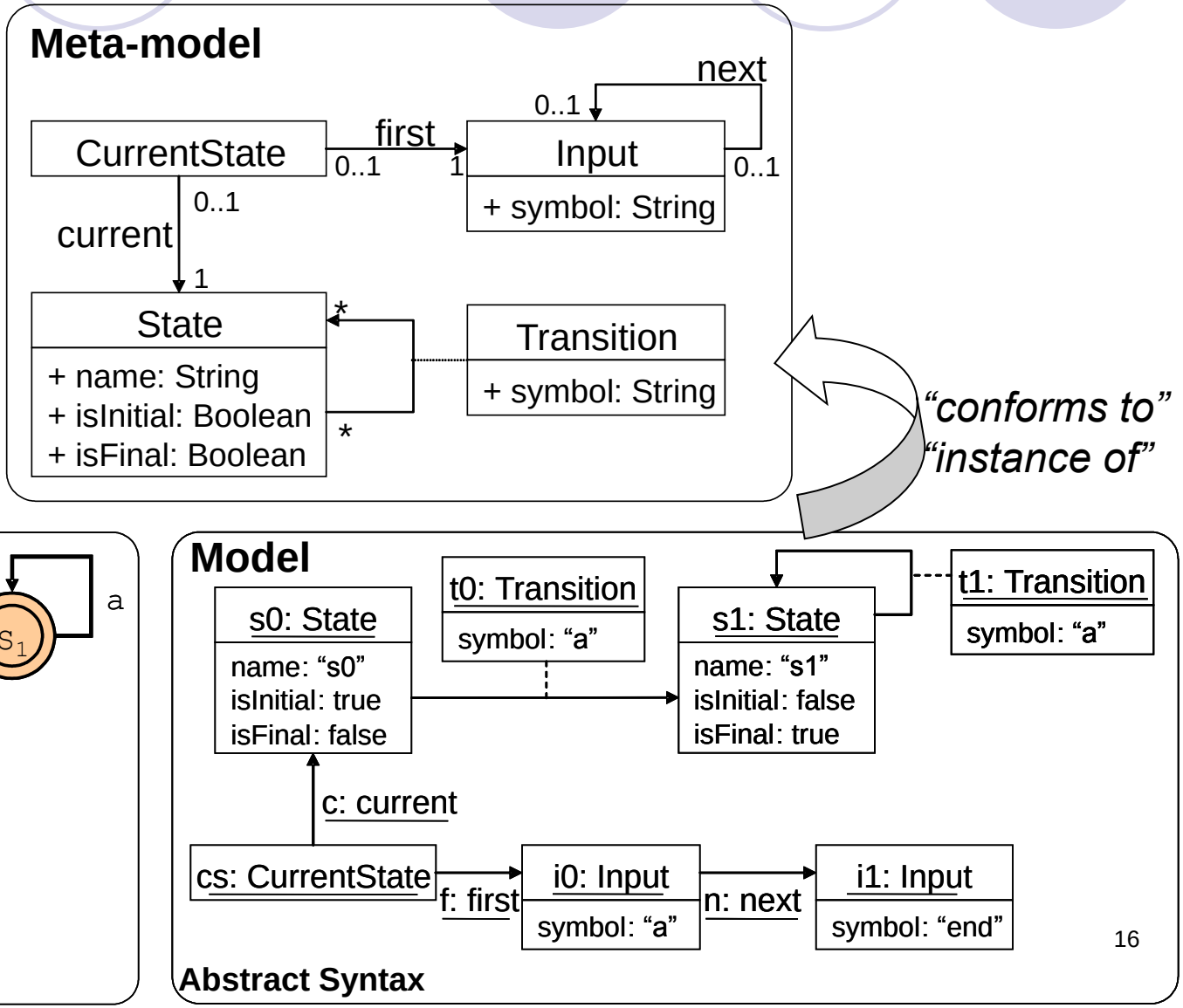
No consoles to display at this time.

Writable | Insert | 10 : 8

How are DSLs defined?

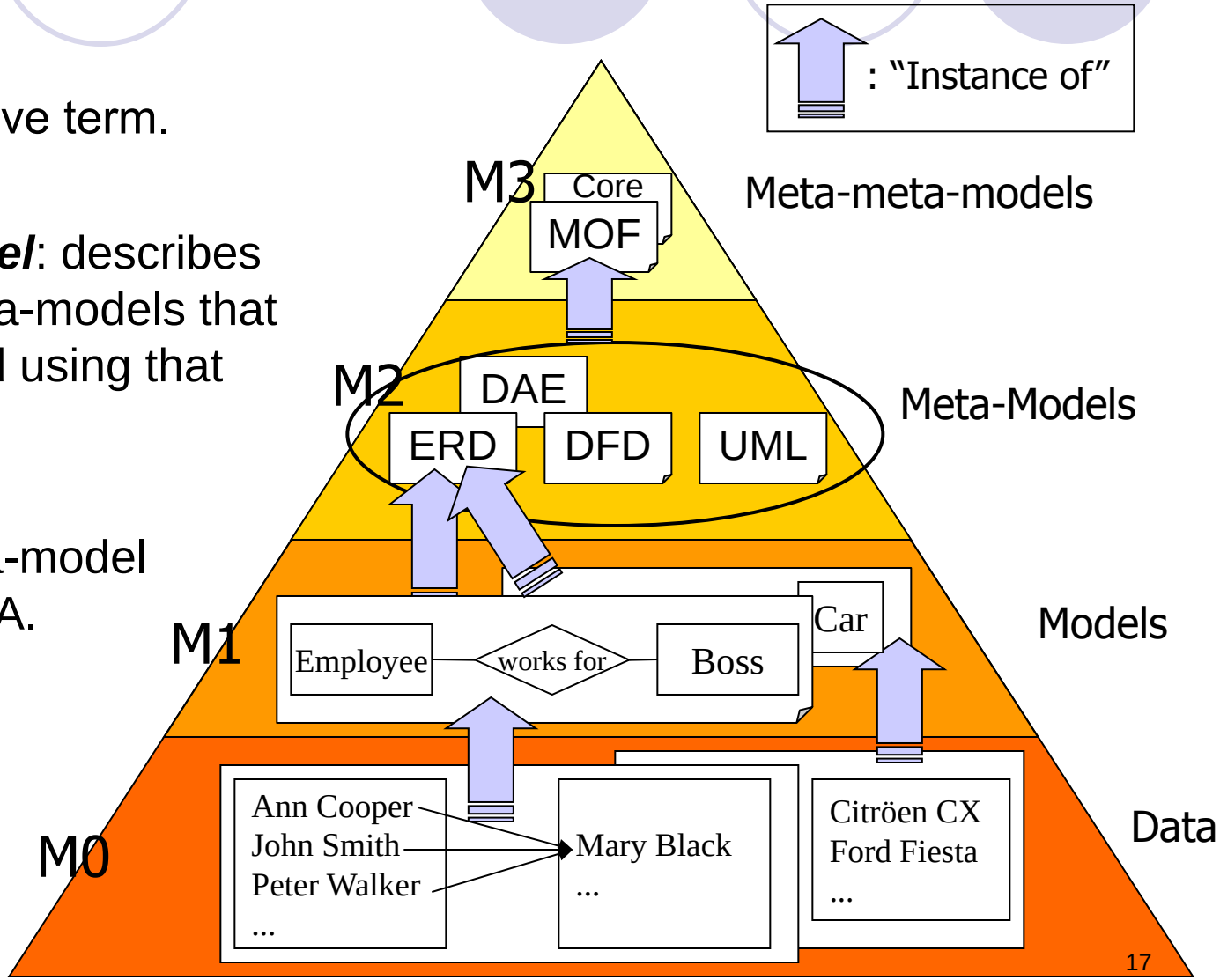
- ***Meta-model***: model that describes the primitives of the language:
  - Describes a set of models considered valid.
  - Defines the language ***abstract syntax***.
- The ***concrete syntax*** includes information about how to visualize/represent the concepts of the abstract syntax.

How are DSLs defined?



How are DSLs defined?

- "meta-" is a relative term.
- ***Meta-meta-model***: describes the set of all meta-models that can be described using that language.
- **MOF**: meta-meta-model proposed by MDA.



Index

- Introduction.
- Concepts and Terminology.
  - MDSD and DSLs.
  - **Transformations.**
  - Product Lines.
  - MDA.
  - Language-Oriented Programming.
  - Low-code Development.
- Technology.
- Classification, Comparison and Conclusions.
- Bibliography and suggested readings.

Transformations

- Model Manipulations.
- Model-to-Model (exogenous):
  - Different source and target meta-models.

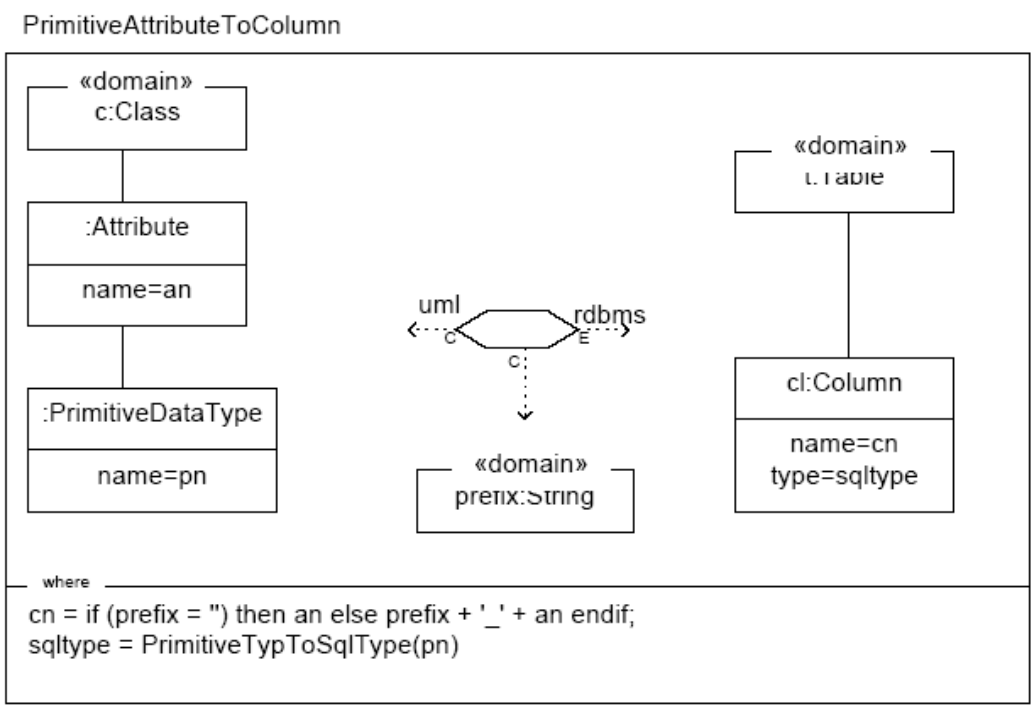
    <u>examples</u>: from class diagrams to relational schema, from “traffic” to petri nets, etc.
- “In-place” (endogenous):
  - Same meta-model.

    <u>examples</u>: redesign (refactoring), simulation, animation.
- Model-to-platform:
  - Code generation.

Transformations

- Types of transformation languages:
  - Imperative/Declarative/Hybrid.
  - Textual/Visual.
  - Formal/semi-formal semantics

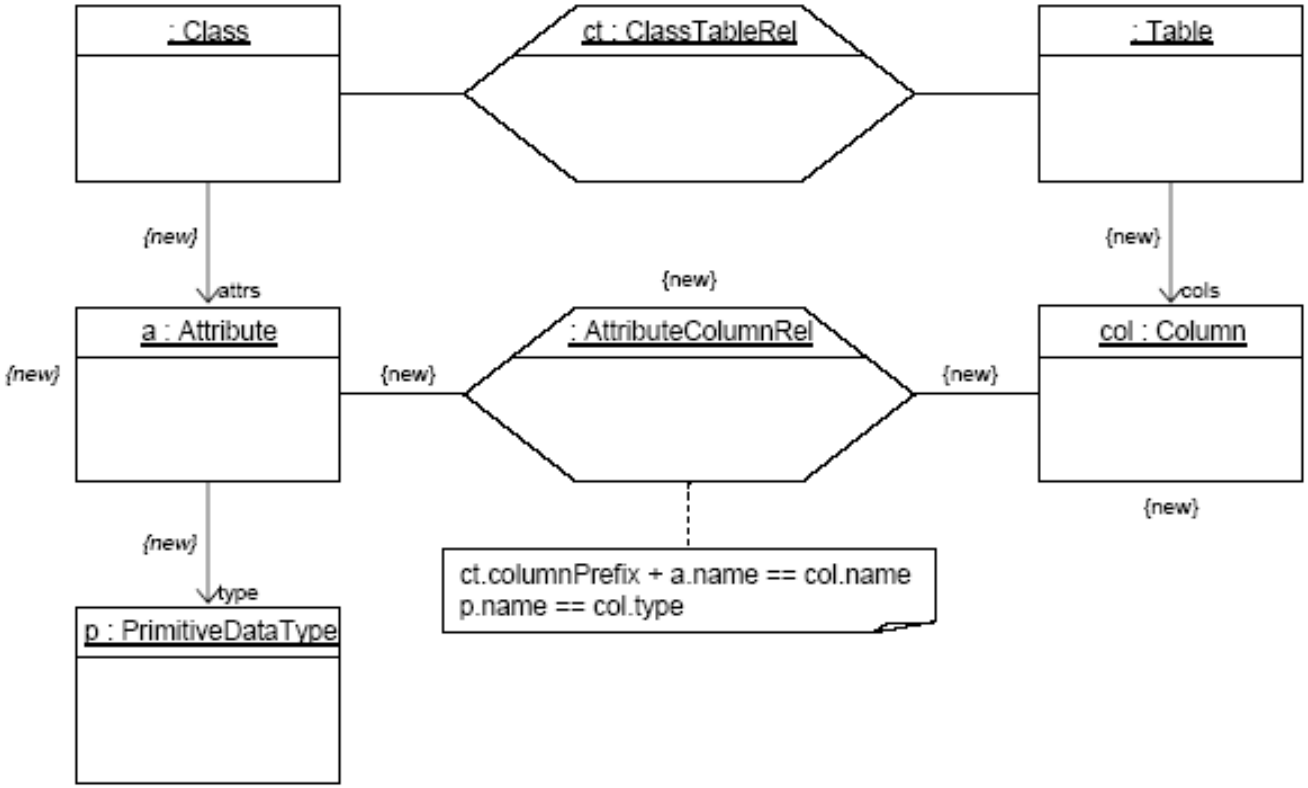
Transformations. QVT.




```
relation PrimitiveAttributeToColumn
{
   an, pn, cn, sqltype: String;

   checkonly domain uml c:Class {attribute=a:Attribute {name=an,
                                   type=p:PrimitiveDataType {name=pn}}};
   enforce domain rdbms t:Table {column=cl:Column {name=cn,
                                   type=sqltype}};
   primitive domain prefix:String;
   where {
      cn = if (prefix = '') then an else prefix+'_'+an endif;
      sqltype = PrimitiveTypeToSqlType(pn);
   }
}
```

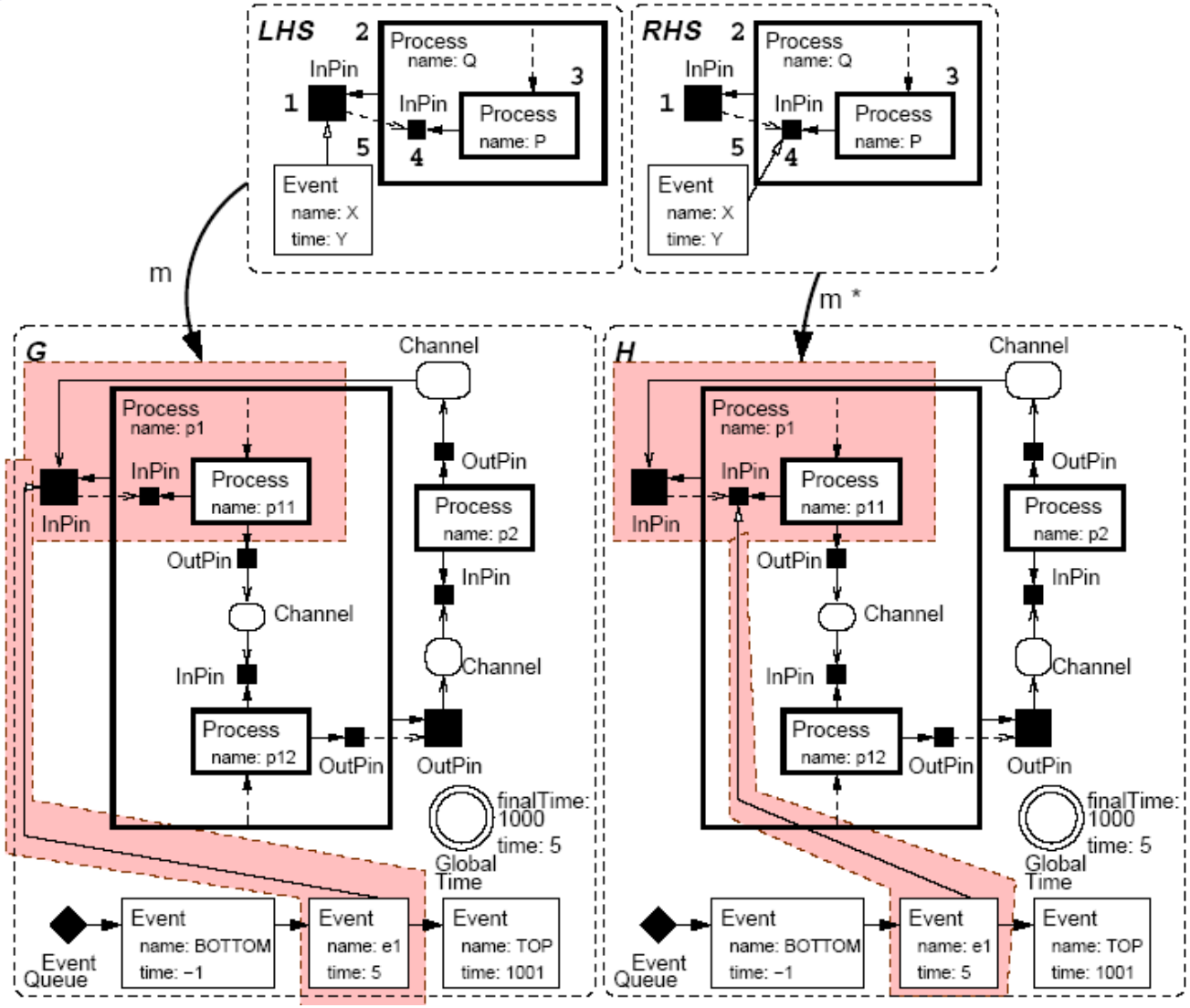
Transformations.
Triple Graph Grammars.



- Synchronized creation of elements in source and target models.
- Algorithms to derive "*operational rules*": transformations in both directions, incremental, bidirectional.

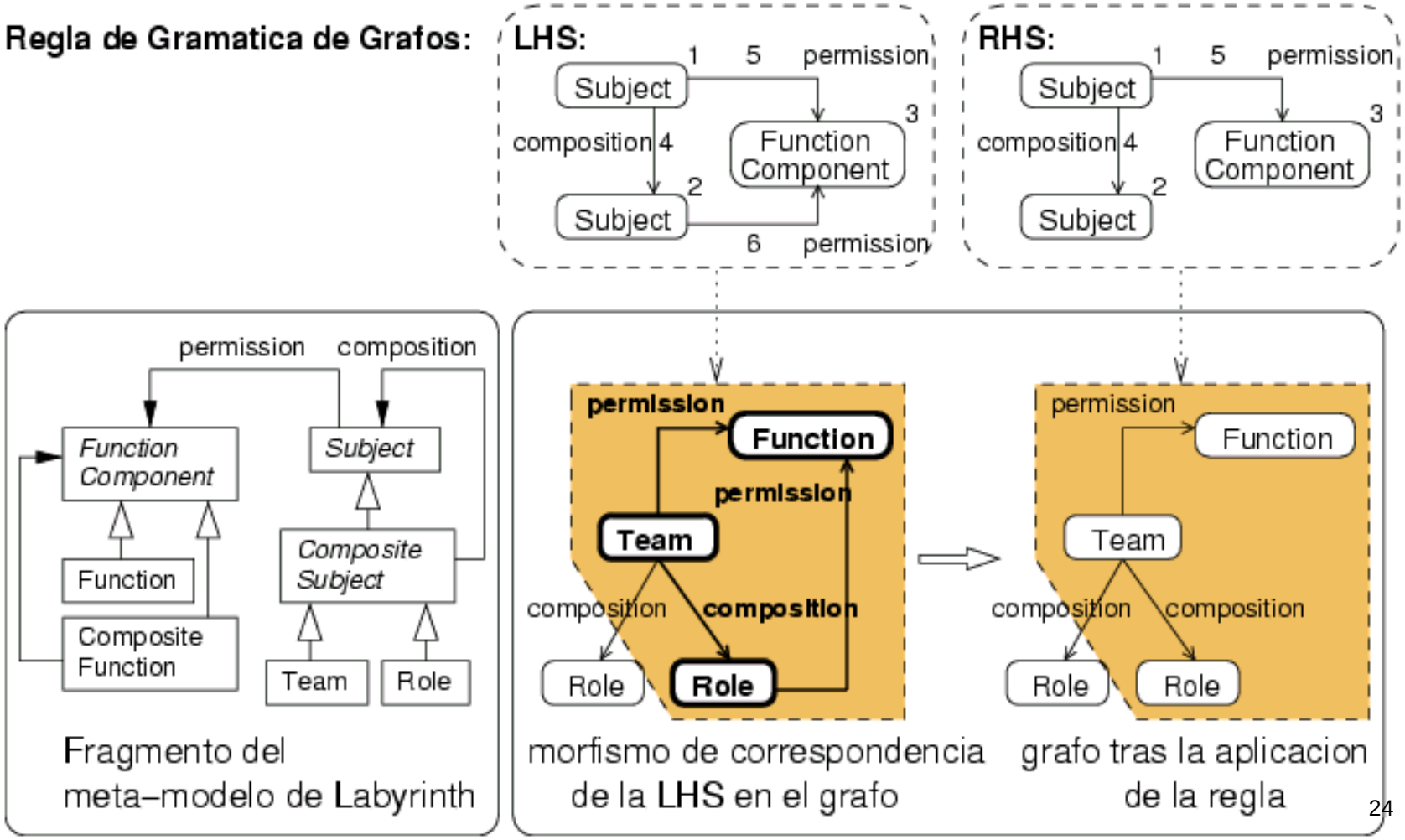
Transformations.

Graph Grammars: Simulation.



Transformations.

Graph Grammars: Refactoring.



Transformations. ATL.

- ATLAS Transformation Language.
- Textual language.
- Declarative and imperative constructs.

```
rule PersistentClass2Table{
    from
        c : SimpleClass!Class (
            c.is_persistent and c.parent.oclIsUndefined()
        )
    to
        t : SimpleRDBMS!Table (
            name <- c.name
        )
}
```

Index

- Introduction.
- Concepts and Terminology.
  - MDSD and DSLs.
  - Transformations.
  - **Product Lines.**
  - MDA.
  - Language-Oriented Programming.
  - Low-code Development.
- Technology.
- Bibliography and suggested readings.

Product Lines

"Set of complementary products that **share** features specific to some **particular domain** and that are developed from a set of artefacts in a prescribed way."

Software Engineering Institute

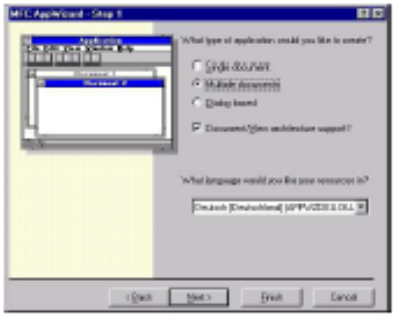
Carnegie Mellon

http://www.sei.cmu.edu/productlines/

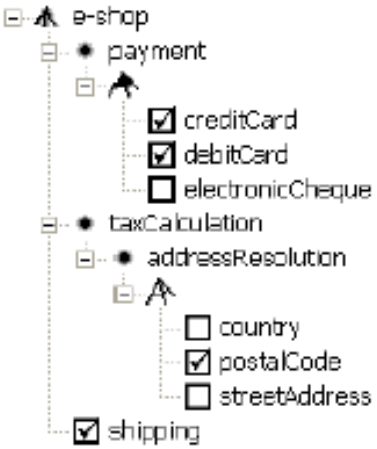
Product Lines

Variability

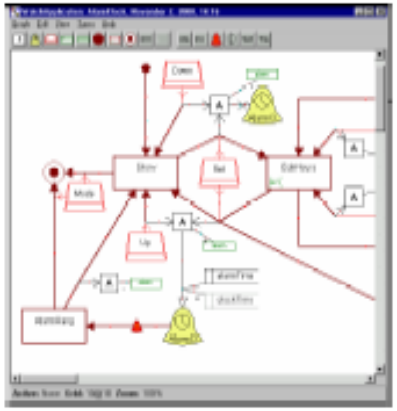
Routine configuration ⟷ *Creative construction*



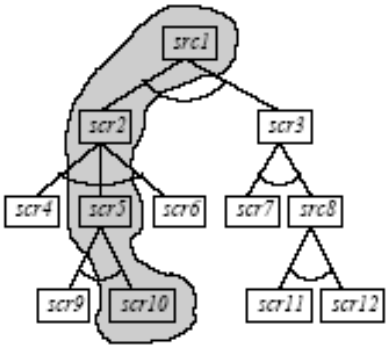
Wizard



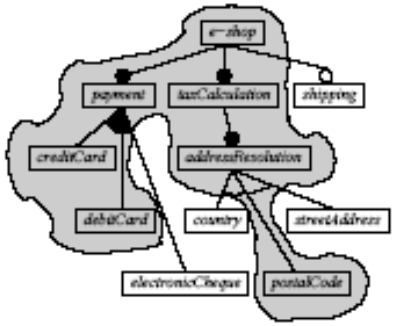
Feature-based configuration



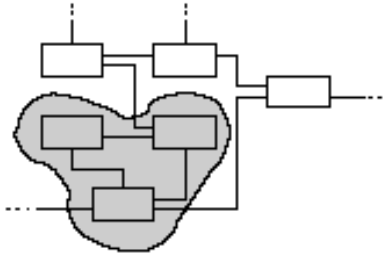
Graph-like language



Path through decision tree



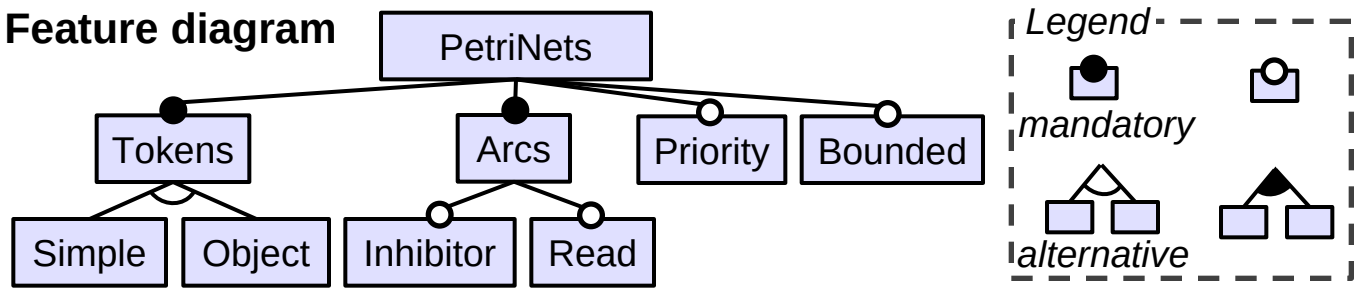
Subtree of feature model



Subgraph of (infinite) graph

(from Czarnecki)

Feature models



- Exponential number of configurations
- Map configurations to products
  - Annotative approaches (#ifdefs)
  - Compositional approaches (assemble artefacts out of components)

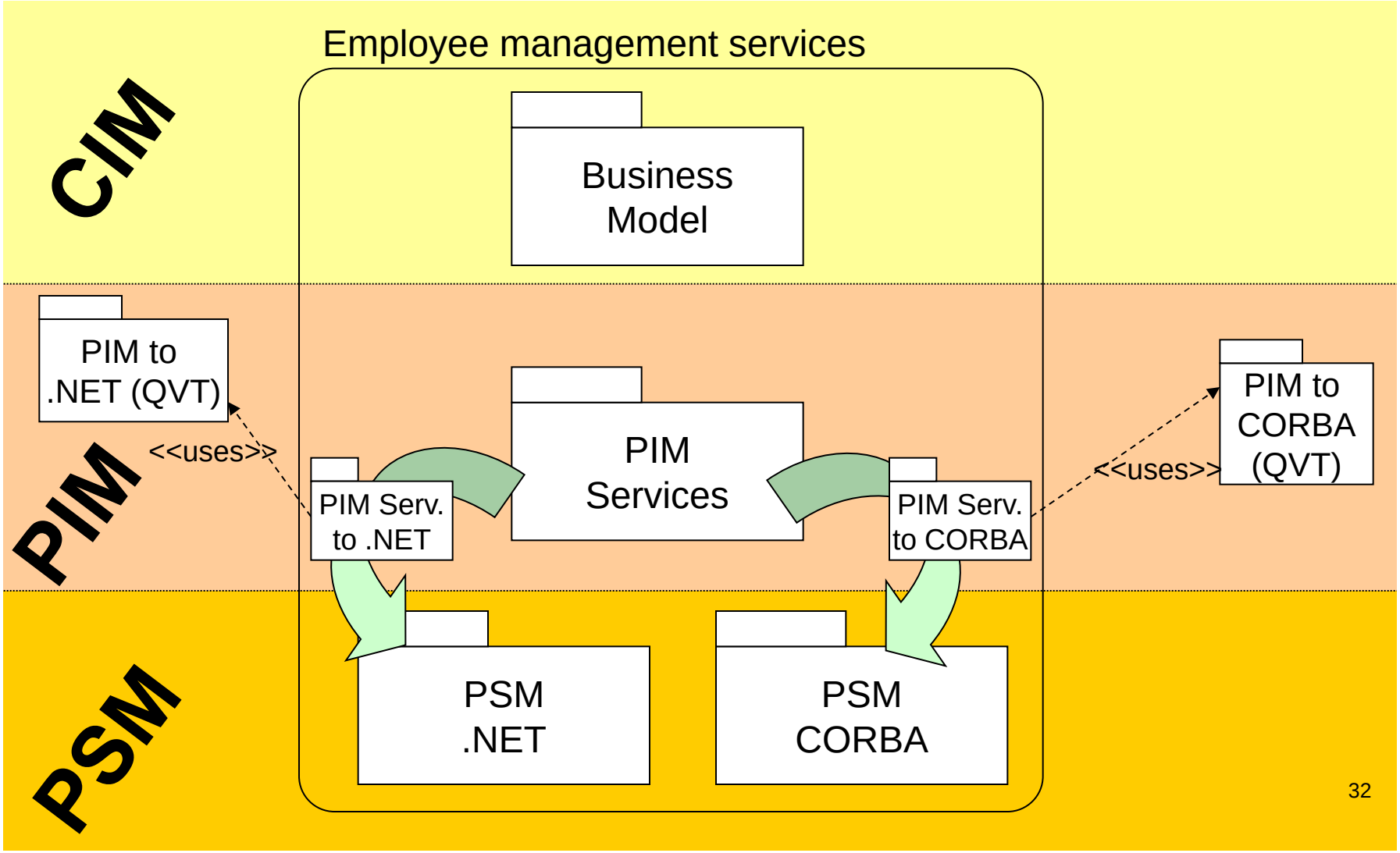
Index

- Introduction.
- Concepts and Terminology.
  - MDSD and DSLs.
  - Transformations.
  - Product Lines.
  - **MDA.**
  - Language-Oriented Programming.
  - Low-code Development.
- Technology.
- Bibliography and suggested readings.

Model-Driven Architecture (MDA)

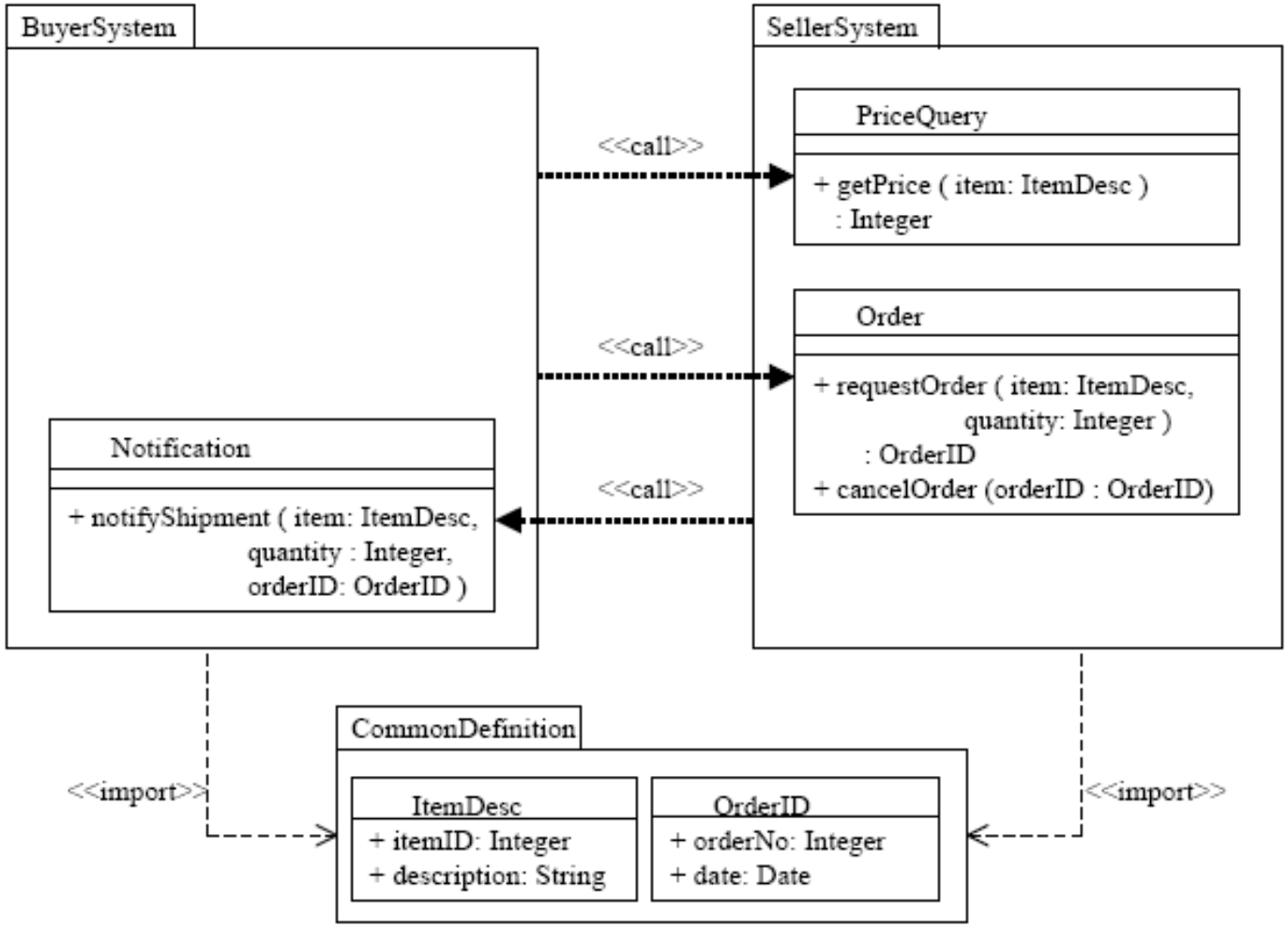
- MDSD incarnation using a set of OMG standards (http://www.omg.org).
- UML (modelling)+OCL (constraints)+MOF (meta-modelling)+QVT (queries/views/transformations). XMI for model persistency.
- Models at different abstraction levels:
  - CIM: Computation Independent Model.
  - PIM: Platform Independent Model.
  - PSM: Platform Specific Model.
  - PDM: Platform Description Model.

MDA


Employee management services
CIM
Business Model
PIM
PIM to .NET (QVT)
<<uses>>
PIM Serv. to .NET
PIM Services
PIM Serv. to CORBA
<<uses>>
PIM to CORBA (QVT)
PSM
PSM .NET
PSM CORBA

MDA

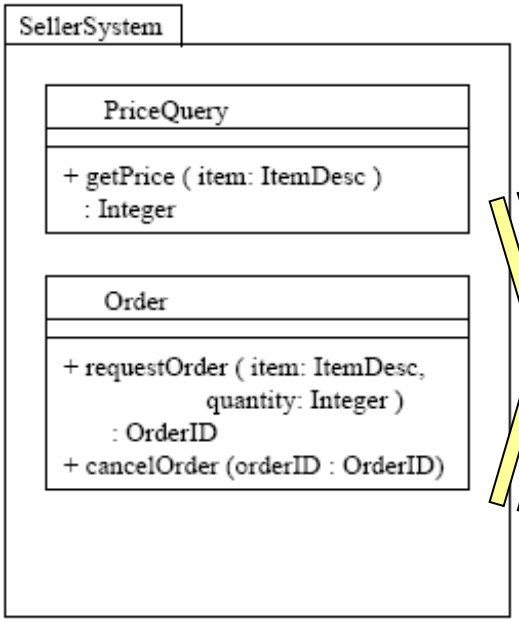
PIM example

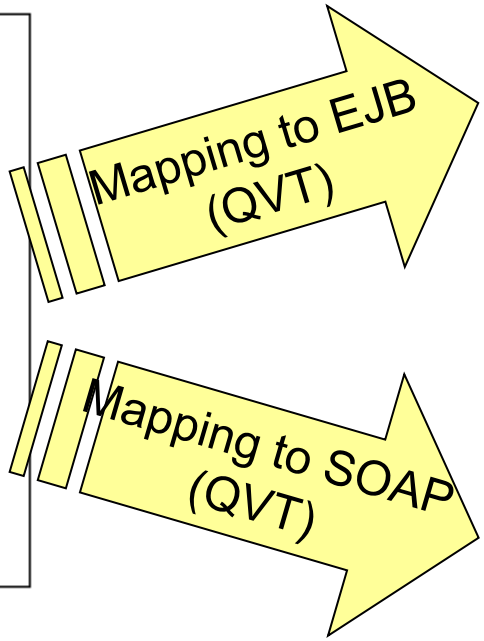


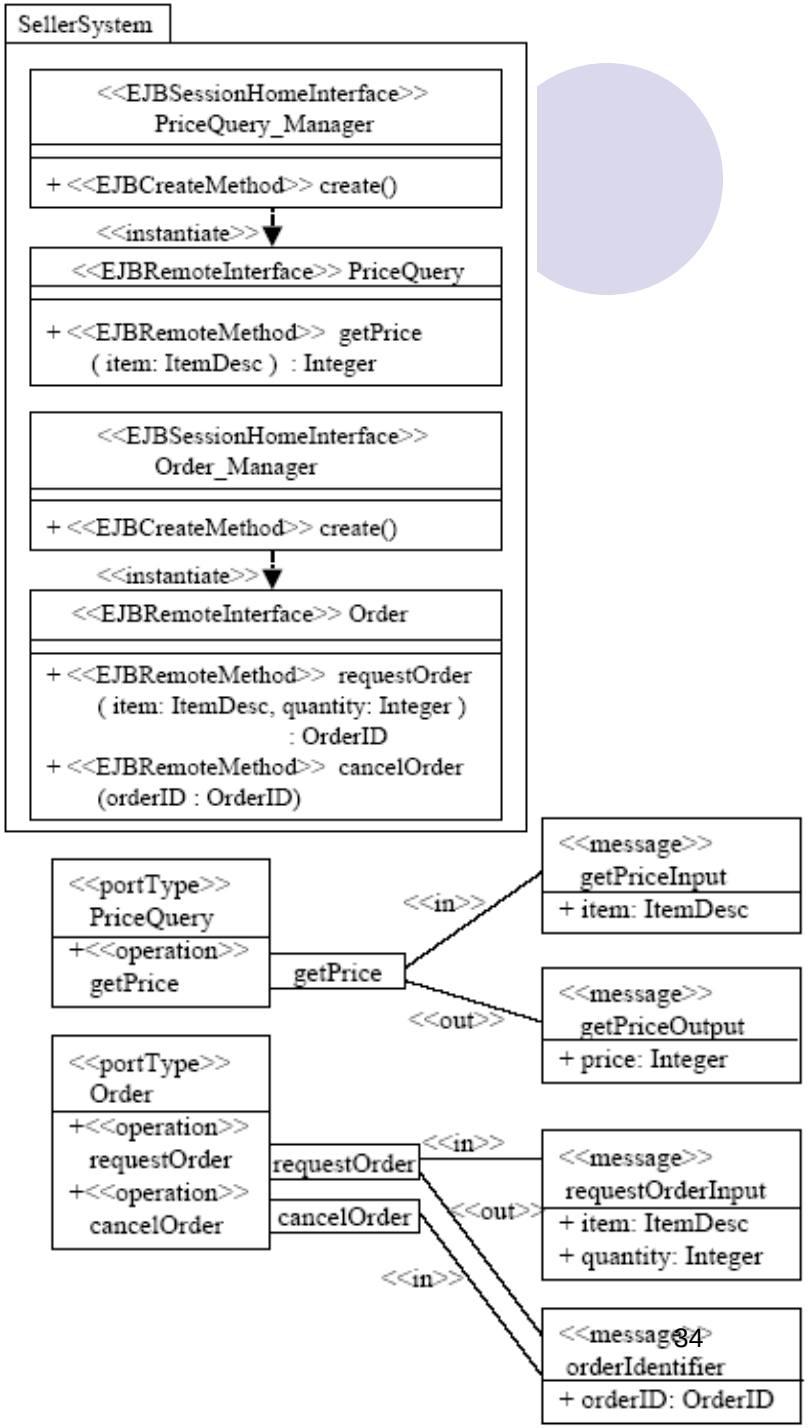
(from Makoto Oya, Hitachi)

MDA

from PIM to PSM(EJB, SOAP)



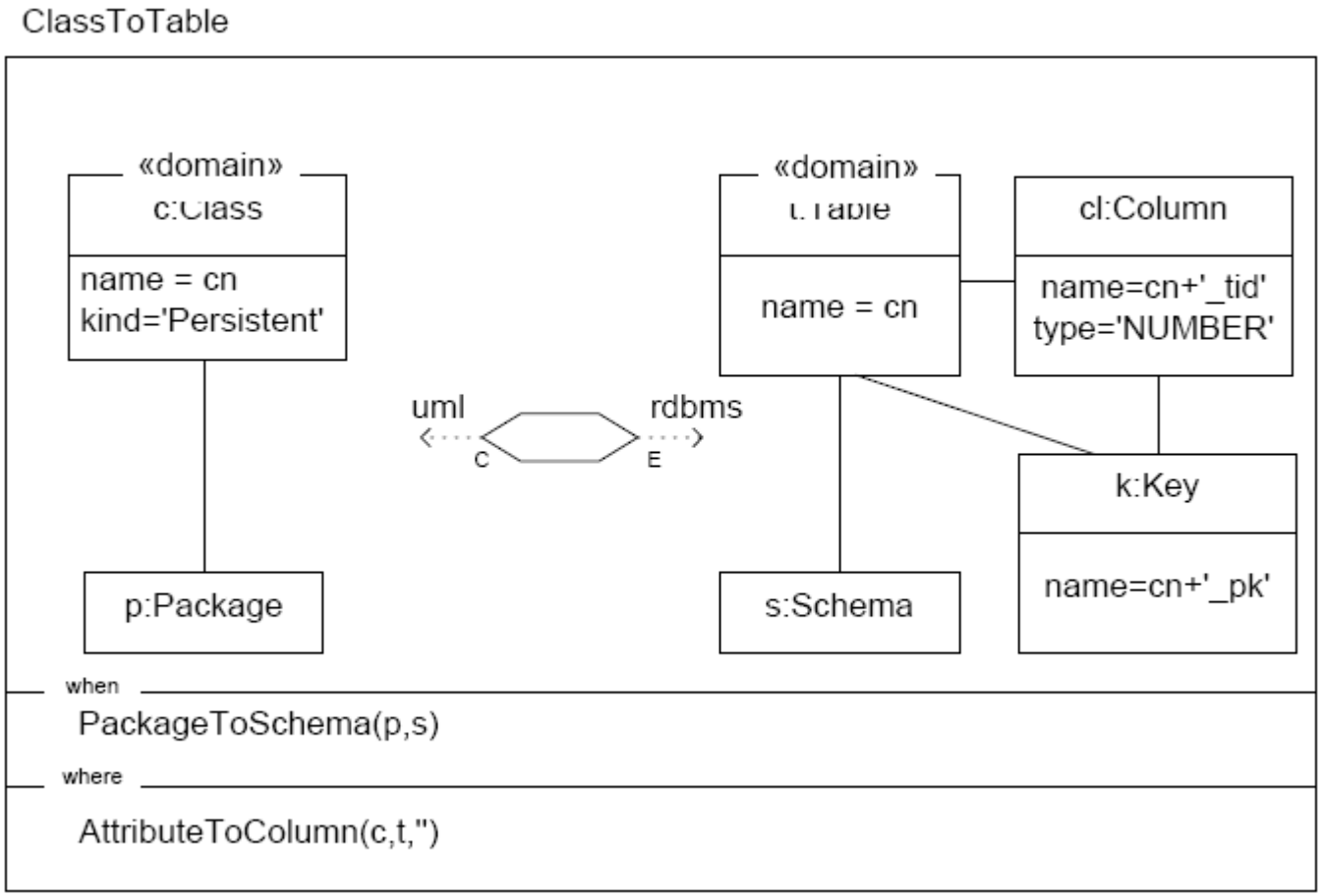




(from Makoto Oya, Hitachi)

MDA: QVT

Example: from UML to Relational Schema



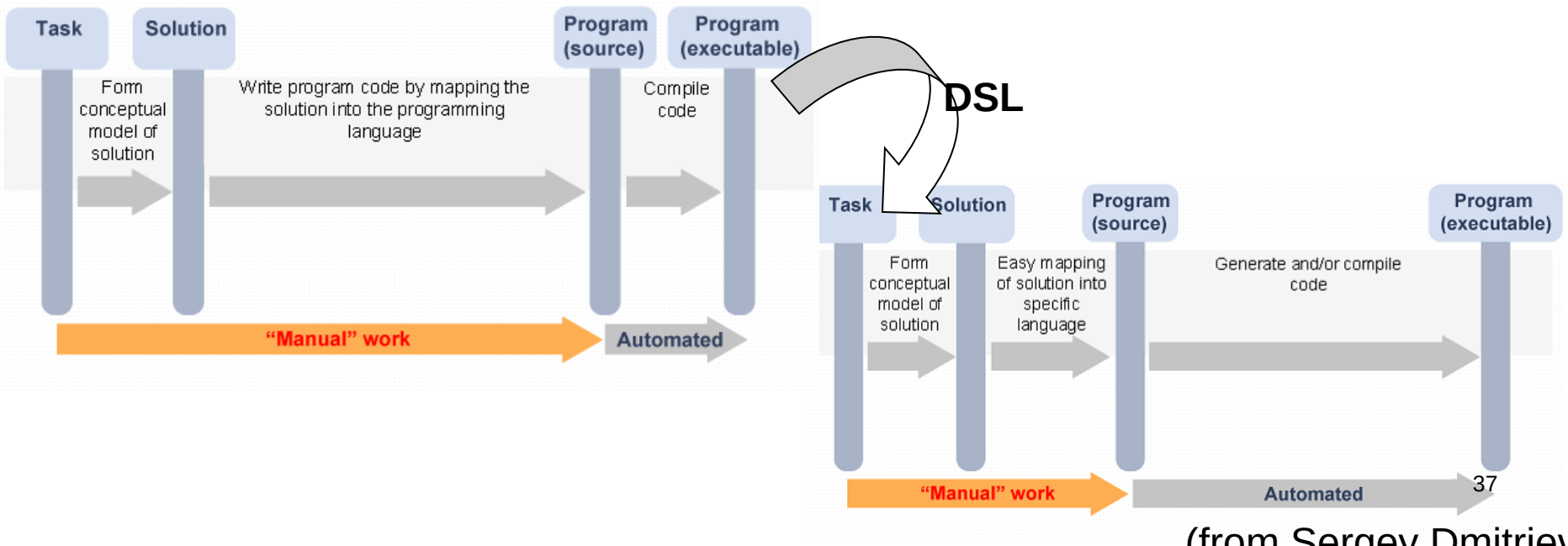
Index

- Introduction.
- Concepts and Terminology.
  - MDSD and DSLs.
  - Transformations.
  - Product Lines.
  - MDA.
  - **Language-Oriented Programming.**
  - Low-code Development.
- Technology.
- Bibliography and suggested readings.

Language-Oriented Programming

- Configure the programming languages, adapting them to the task to be done.

- "*make the computer think as a programmer, not the other way round*"



(from Sergey Dmitriev)

Language-Oriented Programming

- Problems with general-purpose languages:
  - "*Gap*" in the implementation of ideas.
  - Understandability and maintainability of code.
  - Complexity of specific libraries.
- Much of the complexity is due to "*accidental*" details, not being essential to the problem.
- Allow the programmers to create their own languages.
- Editors working with abstract syntax (not only text), and permit a configuration of the concrete syntax.

Language-Oriented Programming

```
Event: Internet on button: 1  Greeting:  Did you know that our internet is fas
    On button: 1  -->  Discont
    On button: 2  -->  Data limit
    On button: *  -->  Return to main menu

Event: Discont on button: 1  Greeting:  Welcome in section of discounts!
    On button: 1  -->  Summer discount
    On button: 2  -->  Hidden discounts
    On button: *  -->  Step back

Event: Summer discount on button: 1  Greeting:  Hello!
    other
Event: Hidden discounts on button: 2
    get info
Event: Step back on button: *

Event: Da
    dire
Event: Re
    back

Event: Payment on button: 2  Greeting:  Since now we offer you easiest way of p
    On button: 1  -->  Billing
    On button: 2  -->  Recharging
    On button: 3  -->  Payments
    On button: *  -->  Step back
```

```
BACK      (Action in jetbrains.mps.samples.VoiceMenu)
GENERAL   (Action in jetbrains.mps.samples.VoiceMenu)
GET_INFO  (Action in jetbrains.mps.samples.VoiceMenu)
MENU      (Command in jetbrains.mps.samples.VoiceMenu)
OTHER     (Action in jetbrains.mps.samples.VoiceMenu)
```

Context Actions

Actions/Events
- <empty>
- BACK
- GENERAL
- GET_INFO
- MENU
- OTHER

Actions/Payments
- BILLING_RETENTION
- CABLE_BILLING_DEPT
- CABLE_T/S_BILLING_DEPT
- INTERNET_BILLING_DEPT
- INTERNET_T/S_DEPT
- PAY_BILL
- PHONE_BILLING_DEPT
- PHONE_T/S_BILLING_DEPT

Actions/Sales
- SALES_CZECH
- SALES_ENGLISH
- SALES_GERMAN

VCS Addons: This project uses Git. For better integration with MPS, it is recom... (3 minutes ago) Git: master :OFF 199 of 1981M

MPS (Meta Programming System) – JetBrains

Index

- Introduction.
- Concepts and Terminology.
  - MDSD and DSLs.
  - Transformations.
  - Product Lines.
  - MDA.
  - Language-Oriented Programming.
  - **Low-code Development.**

- Technology.
- Bibliography and suggested readings.

Low-code development

- Cloud-based, visual approach to software development
- Idea of "*citizen developers*"

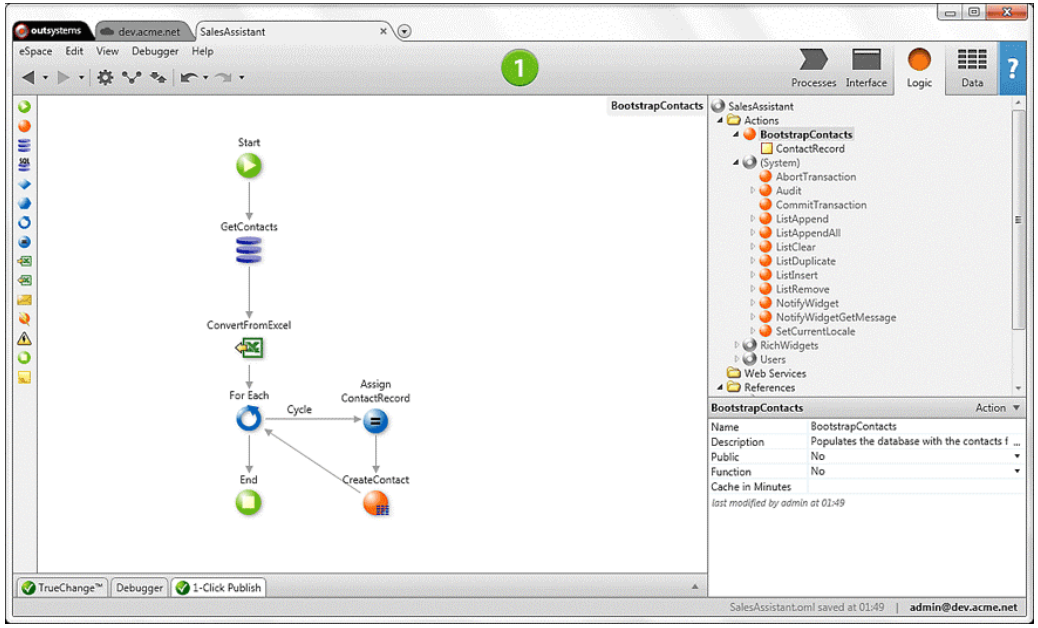
- Low-code and No-code environments

- Is this how programming will be conducted in the future? platforms vs. traditional programming languages

"*By 2024, low-code application development will be responsible for more than 65% of application development activity*." (Gartner)

Low-code development

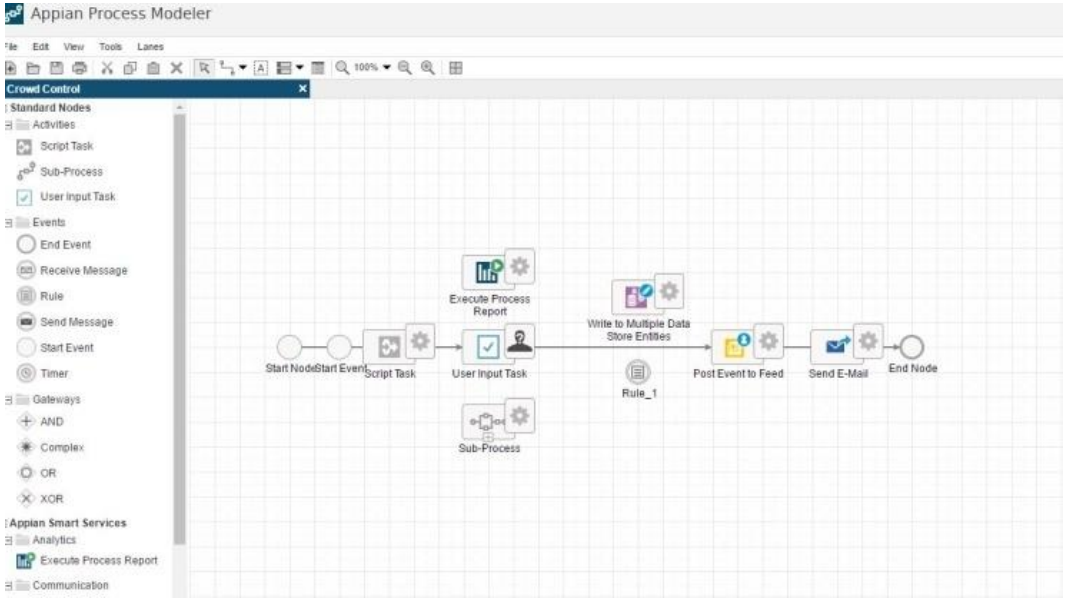
- Many companies offer low-code development environments:
  - Google App Maker
  - Microsoft Business
  - mendix
  - OutSystems
  - Salesforce
  - Appian
  - ...



Outsystems

Low-code development

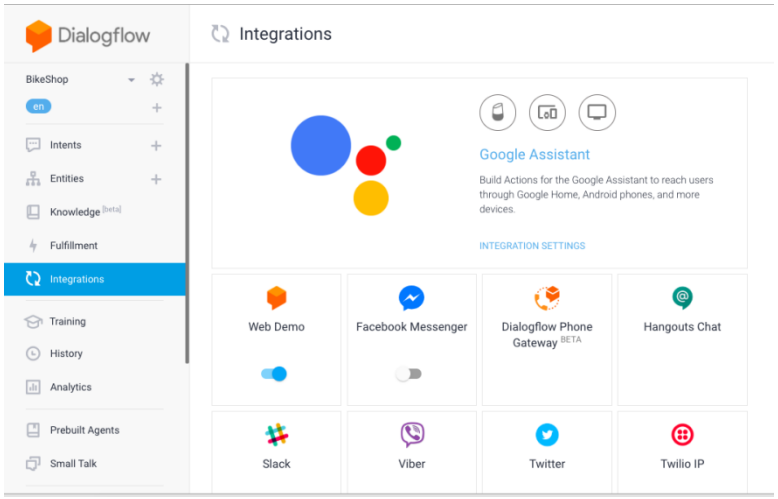
- Many companies offer low-code development environments:
  - Google App Maker
  - Microsoft Business
  - mendix
  - OutSystems
  - Salesforce
  - Appian
  - ...

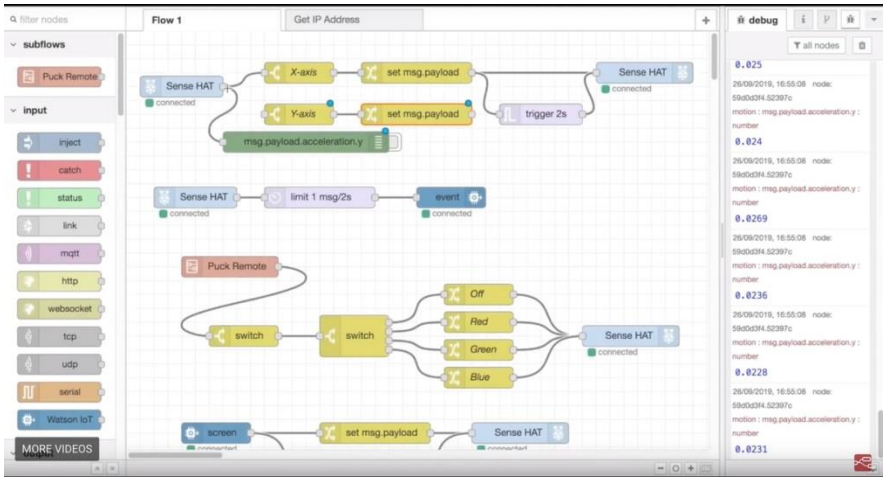


Appian

Low-code development

- Low-code platforms exist also for specialized domains:
  - Chatbot development
    - DialogFlow, Lex, Watson, FlowXO, ...
  - Internet of Things
    - NodeRed, thethings.io
  - Data processing
    - MuleSoft, SnapLogic,...
  - ...





Index

- Introduction.
- Concepts and Terminology.
- **Technology.**
  - **Eclipse Modelling Framework.**
- Bibliography and suggested readings.

Eclipse Modeling Framework

Eclipse: *http://www.eclipse.org/*

Java - MetaDepth/src/meteoric/at3rdx/kernel/Clabject.java - Eclipse Platform

File Edit Source Refactor Navigate Search Project Run Window Help

Packag | Hierarc | JUnit

- src
  - (default package)
  - javax
  - javax.jmi
  - javax.jmi.corbaidltypes
  - javax.jmi.model
  - javax.jmi.primitivetypes
  - javax.jmi.reflect
  - javax.jmi.xmi
  - junit
  - meteoric
  - meteoric.at3rdx
  - meteoric.at3rdx.kernel
    - Clabject.java
    - ClabjectVisitor.java
    - Classifier.java
    - DerivedField.java
    - Edge.java
    - ElementContainer.java
    - Field.java
    - Function.java
    - Instance.java
    - Mapping.java
    - MDPropertyGetter.java
    - MDPropertySetter.java
    - MDTransactionSupport.java
    - Model.java
    - ModelFactory.java
    - Node.java
    - PatternModelFactory.java
    - QualifiedElement.java
    - QualifiedElementInterface.ja
    - Relation.java
    - Type.java
    - VirtualMachine.java
  - meteoric.at3rdx.kernel.behaviou

Clabject.java

```
package meteoric.at3rdx.kernel;

import java.util.Collection;

/**
 * Clabject: The top-most class of the hierarchy
 * @author Juan de Lara
 *
 */
public abstract class Clabject implements Instance, Type, RefObject, RefClass {
    public static final     int UNBOUND = -1;       // -1 to mean infinite.
    protected               int minimum = 0;
    protected               int maximum = UNBOUND;
    protected               String name;            // By now we assume that this also serves for the unique MOF id
    protected               int potency = UNBOUND;  // TODO: Which is the right value for potency?
    // ---------------------------------------------------------
    protected               Type type = null;          // To implement instance interface
    protected               HashMap<QualifiedElement,List<Instance>> instances;  // To implement type interface
    // ---------------------------------------------------------
    protected               List<Constraint> constraints = new ArrayList<Constraint>();
    // ---------------------------------------------------------

    /**
     * Controls whether instances can be extended (strict false) or not. If this is a Model, then extending
     * means adding both new attributes and new types.
     */
    protected boolean strict = false;

    public Clabject(String name) {
        this.name = name;
        this.instances = new HashMap<QualifiedElement,List<Instance>>();
        if (VirtualMachine.instance().debug()) System.out.println("Clabject["+name+"]");
    }

    public Clabject(String name, int potency) throws At3InvalidPotencyException {
        this(name);
        this.setPotency(potency);
    }
```

Writable | Smart Insert | 1 : 1

Eclipse

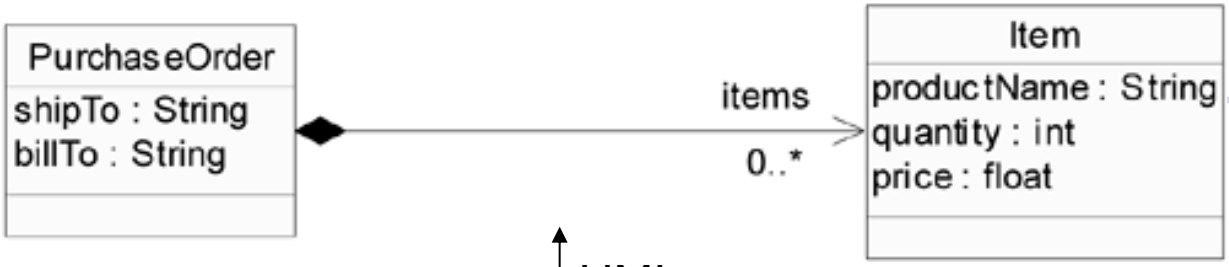
What is Eclipse?

- Not only an environment to develop in Java.
- An open-source, extensible development environment.
- A framework that allows to develop IDEs for other languages.
- An extensible framework for tool integration.
- A large community with more than 400 code projects.

Eclipse Modeling Framework

- A framework to describe (meta)models and generate Java code.
- The generated code has to be completed with extra functionality.
- Integration of modelling and programming.
- Regenerate the model from the code and vice-versa. Techniques for not overwriting the manual code.

Eclipse Modelling Framework



UML

emf
ECLIPSE MODELING FRAMEWORK

Java

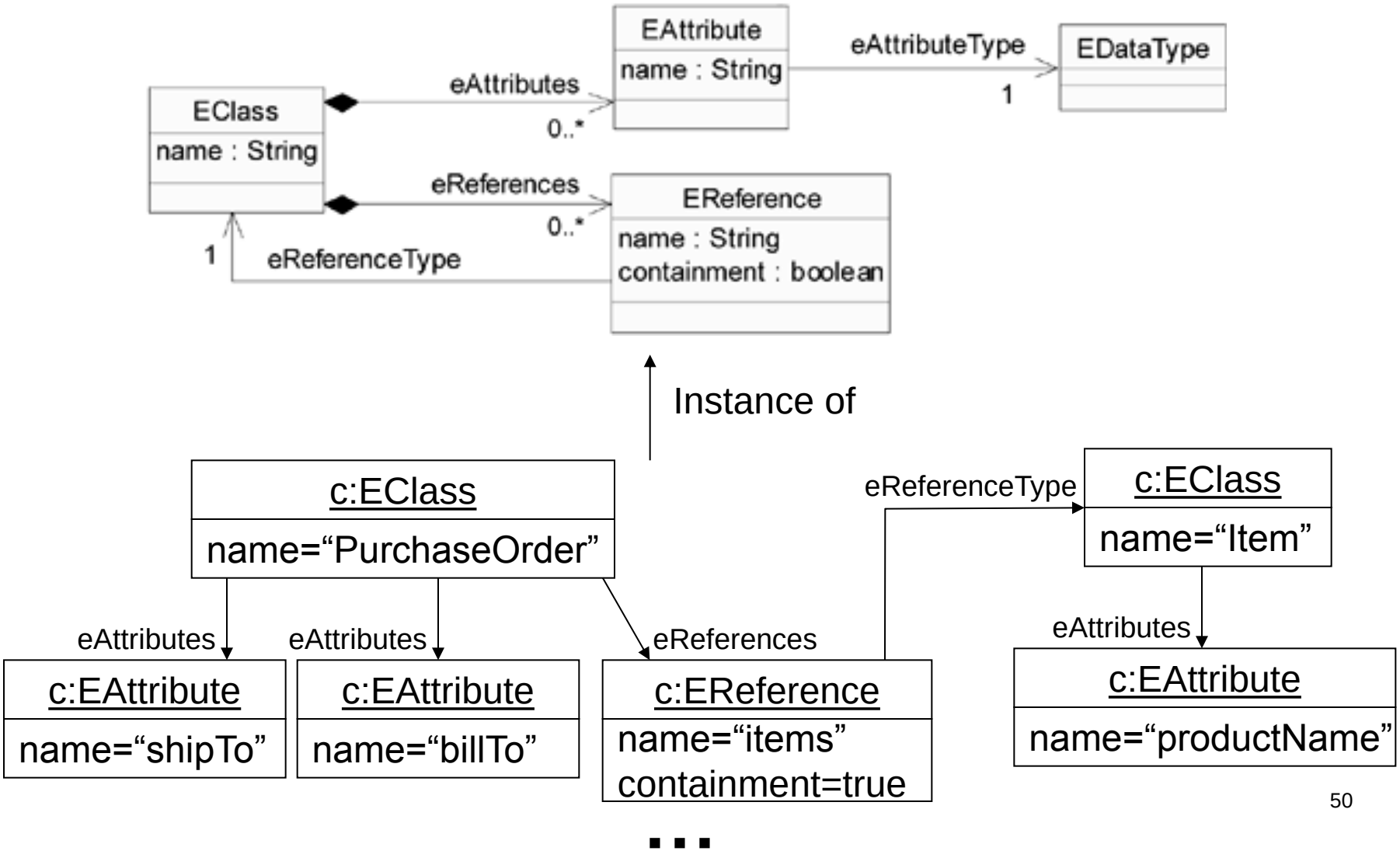
XML

```
public interface PurchaseOrder {
   String getShipTo();
   void setShipTo(String value);
   String getBillTo();
   void setBillTo(String value);
   List getItems(); // List of Item
}
public interface Item {
   String getProductName();
   void setProductName(String value);
   int getQuantity();
   void setQuantity(int value);
   float getPrice();
   void setPrice(float value);
}
```

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema              xmlns:xsd="http://www.w3.org/..."
                         targetNamespace="http://www..."
                         xmlns:PO="http://www....">
   <xsd:complexType name="PurchaseOrder">
   <xsd:sequence>
      <xsd:element name="shipTo" type="xsd:string"/>
      <xsd:element name="billTo" type="xsd:string"/>
      <xsd:element name="items" type="PO:Item"
minOccurs="0" maxOccurs="unbounded"/>
    </xsd:sequence>
   </xsd:complexType>
   <xsd:complexType name="Item">
      …
   </xsd:complexType>
</xsd:schema>
```

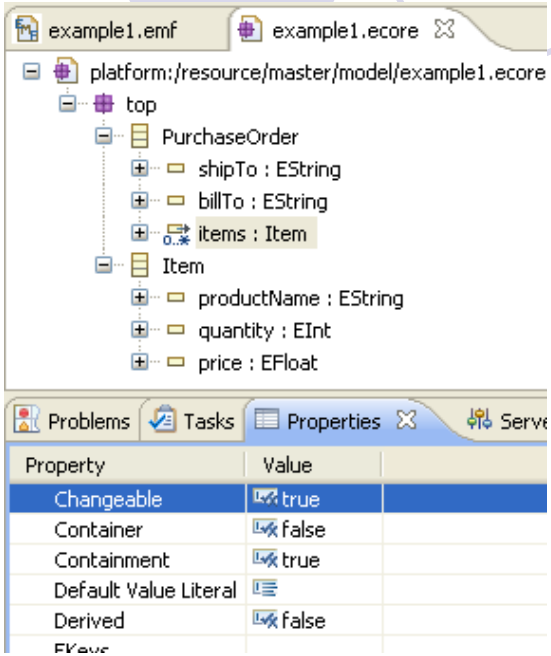
Eclipse Modelling Framework

Ecore.



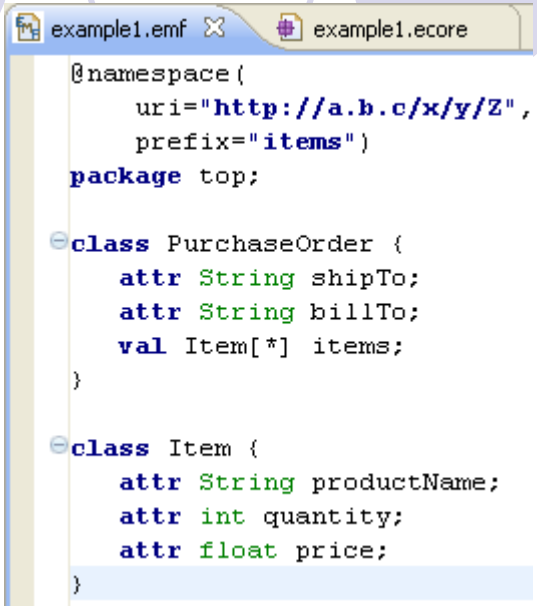
Eclipse Modelling Framework

Editing/persistency of ecore models.



Tree based editor (direct editing).

Import/Export from Rational Rose (.mdl)



Other editors (generation)
E.g.: emfatic

ECore

XMI
Persistency

XMI: Persistency

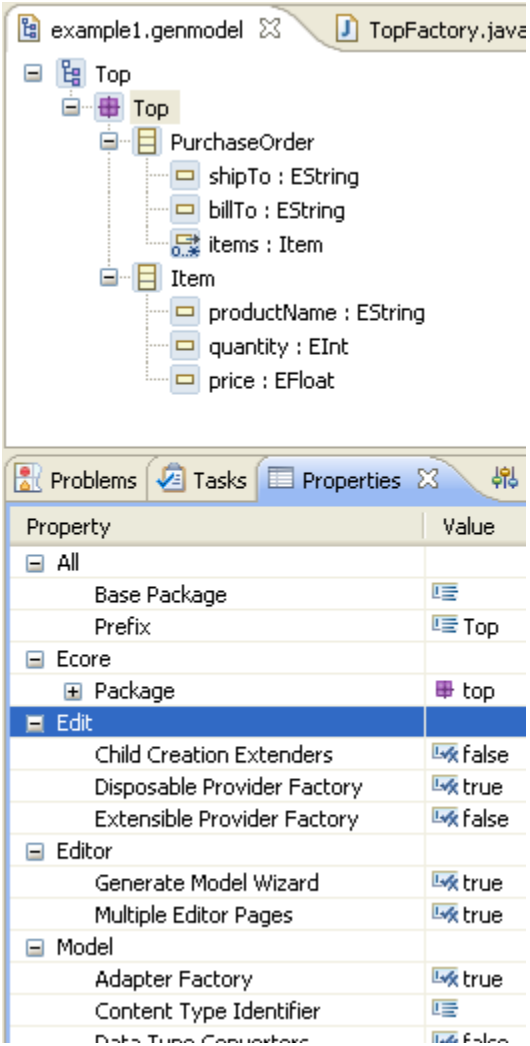
- A standard formal, based on XML, for model persistency.

```
<?xml version="1.0" encoding="ASCII"?>
<ecore:EPackage xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI"
                xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                xmlns:ecore="http://www.eclipse.org/emf/2002/Ecore"
                name="po" nsURI="http:///com/example/po.ecore"
                nsPrefix="com.example.po">
  <eClassifiers xsi:type="ecore:EClass" name="PurchaseOrder">
    <eReferences name="items"
        eType="#//Item" upperBound="-1" containment="true"/>
    <eAttributes name="shipTo"
        eType="ecore:EDataType
               http://www.eclipse.org/emf/2002/Ecore#//EString"/>
    <eAttributes name="billTo"
        eType="ecore:EDataType
               http://www.eclipse.org/emf/2002/Ecore#//EString"/>
  </eClassifiers>
  <eClassifiers xsi:type="ecore:EClass" name="Item">
    <eAttributes name="productName"
        eType="ecore:EDataType
               http://www.eclipse.org/emf/2002/Ecore#//EString"/>
    <eAttributes name="quantity"
        eType="ecore:EDataType
               http://www.eclipse.org/emf/2002/Ecore#//EInt"/>
    <eAttributes name="price"
        eType="ecore:EDataType
               http://www.eclipse.org/emf/2002/Ecore#//EFloat"/>
  </eClassifiers>
</ecore:EPackage>
```

EMF: Code Generation.

Customization of the generated code

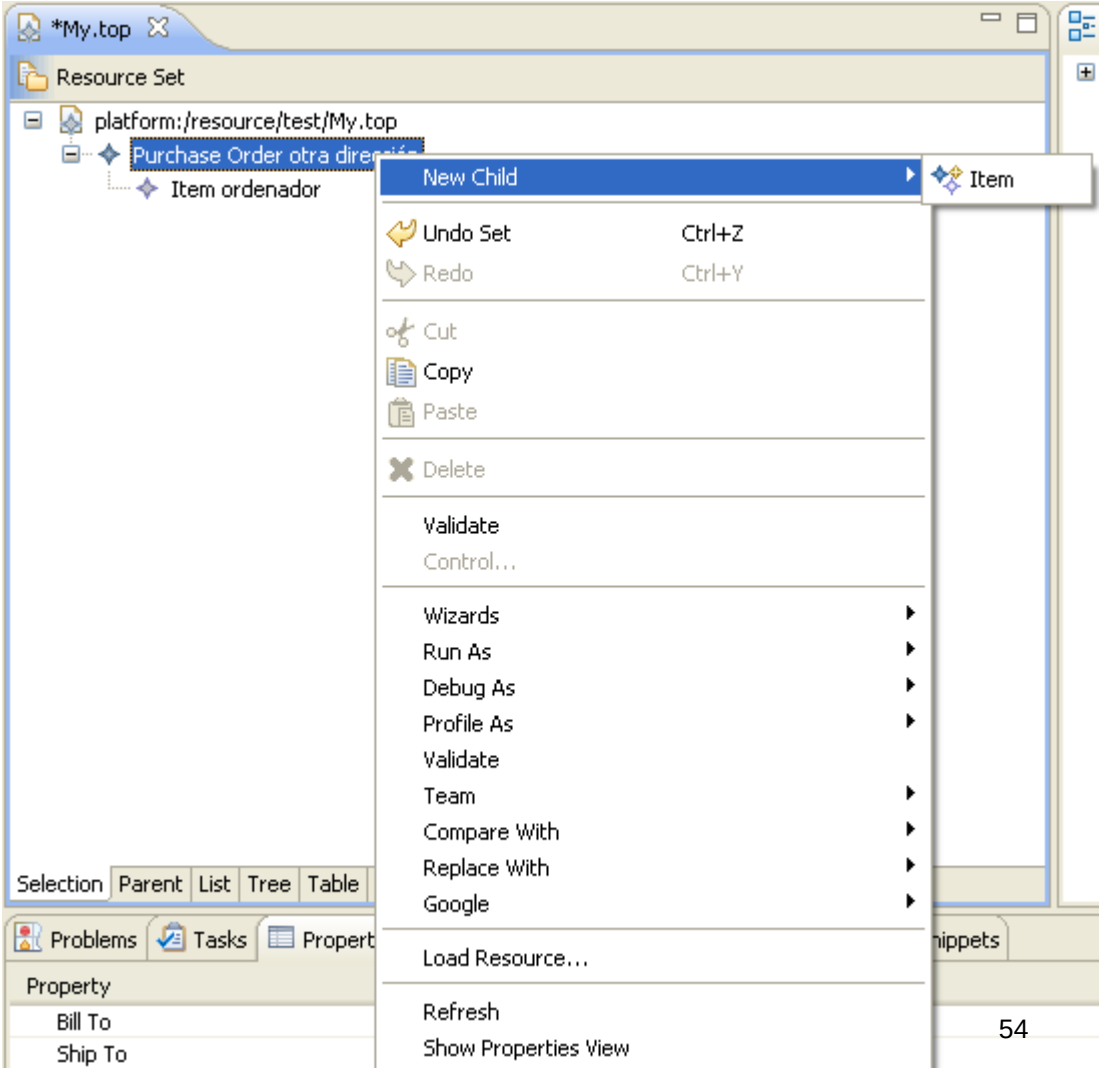
- The generated code can be parameterized by a *.genmodel* model.
- The genmodel model is automatically generated from the ecore model.
- Parameters, like the directory for code generation, or the suffix for implementation classes can be changed.



EMF: Code Generation.

Generated Editor.

- In addition to interfaces and implementation classes, a tree editor is generated to build models.



Bibliography

- MDE:
  - Stahl, Völter, “Model Driven Software Development”. Wiley, 2006.
  - DSLs: http://www.dslbook.org/
  - Brambilla, Cabot, Wimmer. “*Model-Driven Software Engineering in Practice*”, 2nd Edition. Synthesis Lectures on Sof. Eng., Morgan & Claypool. 2017
  - García Molina, y otros. “Desarrollo de Software Dirigido por Modelos: conceptos, métodos y herramientas”. Ed. RA-MA. 2014.
- Product Lines:
  - L. Northrop and P. Clements. 2002. Software Product Lines: Practices and Patterns. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.
- Transformations:
  - Ehrig, Ermel, Golas, Hermann. “*Graph and Model Transformation - General Framework and Applications*”. Monographs in Theoretical Computer Science. An EATCS Series, Springer 2015.
  - Ehrig, Ehrig, Prange, Taentzer. “*Fundamentals of Algebraic Graph Transformation*”. Monographs in Theoretical Computer Science. An EATCS Series, Springer 2006
- Eclipse Modeling Framework:
  - http://www.eclipse.org/modeling/emf/
  - Steinberg, Budinsky, Paternostro, Merks. “*EMF: Eclipse Modeling Framework, 2nd Edition*”. 2008 Addison-Wesley Professional.