



QVT: language, tools and usages

Orange Labs

Friday, January 18, 2008

Mariano Belaunde, Sébastien Poivre , Grégoire Dupé

Research & Development



unrestricted



Content

- QVT standardization status and perspectives
- QVT language highlights (focus on QVT/Operational)
- Existing tools for Q/V/T
- Usages of QVT
 - Model to model transformation
 - Data transformation
 - Model validation
 - Model to code generation: The SmartGen approach
- Application Domains
- Conclusions & perspectives
 - Strengths and weak-points



QVT Standardization status and perspectives

Orange Labs

Friday, January 18, 2008
Sebastien Poivre, Research & Development



unrestricted



QVT Standardization status

- Pre-adoption in October 2005
 - A pure declarative formalism: QVT Relation
 - A hybrid formalism: QVT Operational
- Finalization between October 2005 and June 2007
 - ~ 150 issues carried by the FTF
- Official adoption of QVT 1.0 in October 2007
 - Document ptc/07-07-07
- RTF to provide QVT 1.1 by June 2008

QVT Standardization perspectives (2)

- Foresee improvements of the spec
 - Formalizing the link between the textual notation and the metamodel by means of a transformation
 - Formalizing the execution semantics by means of a transformation to a simple OO language.
 - Support of CMOF 2.0 specificities
 - From OCL:
 - User-defined parameterized types/methods (cf KerMeta)
 - Syntax revision??? (equality versus assignment)
- Other possible actions
 - Adaptation for pure XML transformations
 - Usage of QVT for domain-specific MOF mappings



QVT Language Highlights

M2M Principles

QVT formalisms

QVT Operational features

Orange Labs

Friday, January 18, 2008

Sebastien Poivre, Research & Development



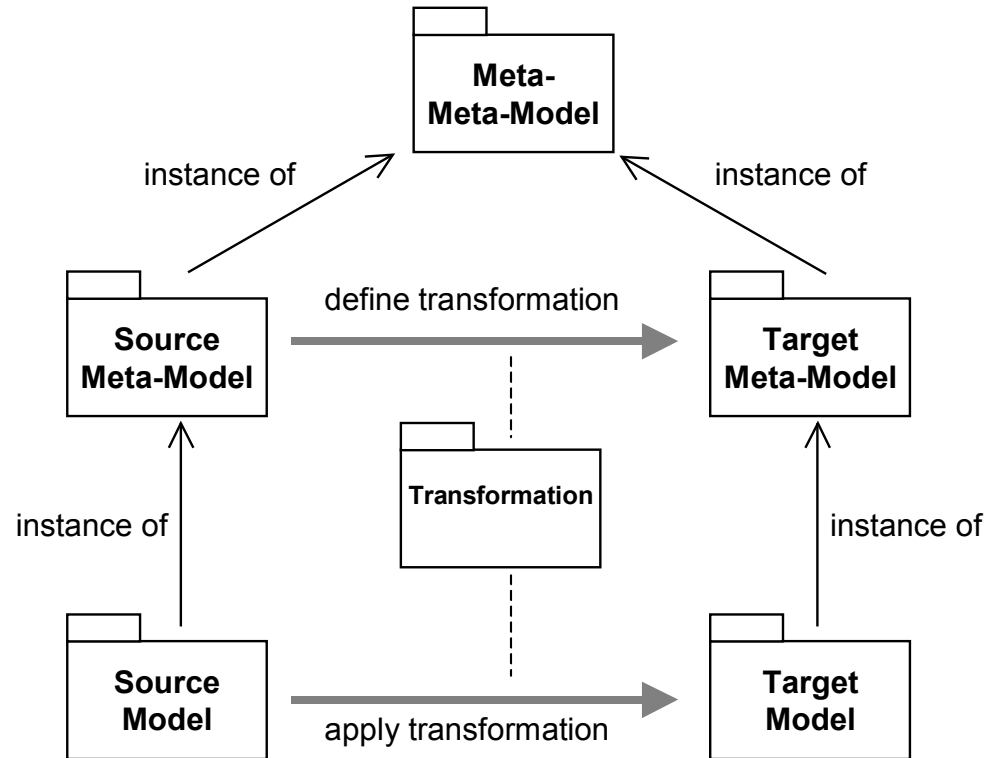
unrestricted



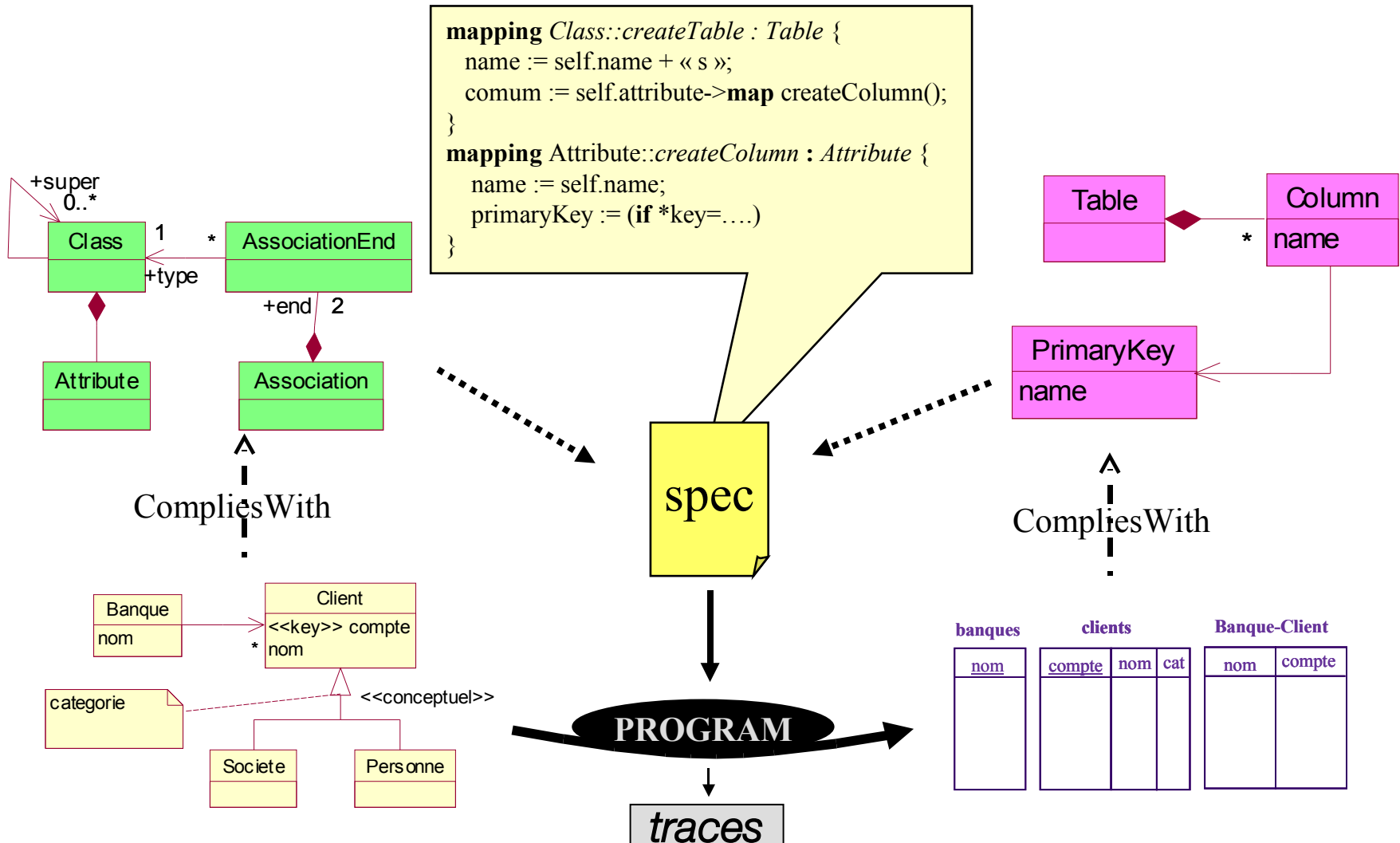
Model Transformation in MDA

- Model transformation is a key element in model-driven approach
 - Vertical transformation: from high-level to low-level (spec to implementations, PIM to PSM)
 - Horizontal transformations (formalism changes)
- Multiple approaches to develop transformations
 - Coded with APIs
 - Declarative relationships (mappings) + code
 - Model oriented languages: J, KerMeta, ...
 - Dedicated languages: Viatra, ATL, QVT/Rel and QVT/Op

Model Transformations in MOF context



Model Transformation in MOF context



QVT 1.0 Standard

- Two flavors
 - Pure declarative: QVT relational. See [QVT/Rel sample \(1\)](#).
 - Hybrid approach: QVT operational. See [QVT/Op Sample \(1\)](#).
 - + QVT Core as a semantic definition of QVT/Rel
- Implementations available and progressing a lot in terms of tool maturity!

QVT Operational Summary

- Strengths

- Transformation designers concentrate on the logic of the transformation
- Rule structure and compacity of code facilitates maintenance
- Object oriented fledged with extensibility
- Imperative constructs available for complex treatments (while, ordered instructions, loops, intermediate properties, hash-tables). Ordinary programming paradigm required.

- Difficulties

- Requires knowledge of metamodeling principles (the metamodel is the API).
- New syntax to learn and assumes OCL knowledge
- Lack of documentation

QVT Operational Usages

- For Model to Model Transformation
 - Capable of expression complex and large transformation
 - Targeting users with "ordinary" programming skills
- For Data Transformation
 - Requires metamodeling-based representation
 - Specific serializers/deserializes for large scale databases: ex SmartQVT with CDO
- For Model to Code generation
 - Use of blackboxes and string templates
 - Use of intermediate template model : SmartGen tool
- For Text to model
 - Partial treatment through model representation of a BNF.
Ex QvtAST to QVT in SmartQVT
- For Model validation
 - Exploiting hybrid QVT syntax versus pure OCL



Qvt Tools

Focus in SmartQVT and SmartGen

Orange Labs

Friday, January 18, 2008

Sebastien Poivre, Research & Development



unrestricted



QVT available Tools

- QVT Relation
 - ModelMorph, (TCS)
 - MedinaQVT (IKV++)
 - C++ based implementation on top of Medina meta-modelling tool
 - Thales for the relational part
- QVT Operational
 - SmartQVT (France Telecom),
 - Together/QVT now in Eclipse project (Borland)
 - QVT/ATLAS virtual machine (INRIA Nantes)



SmartQvt

QVT parser, compiler and launcher

Orange Labs

Friday, January 18, 2008

Sebastien Poivre, Research & Development



unrestricted



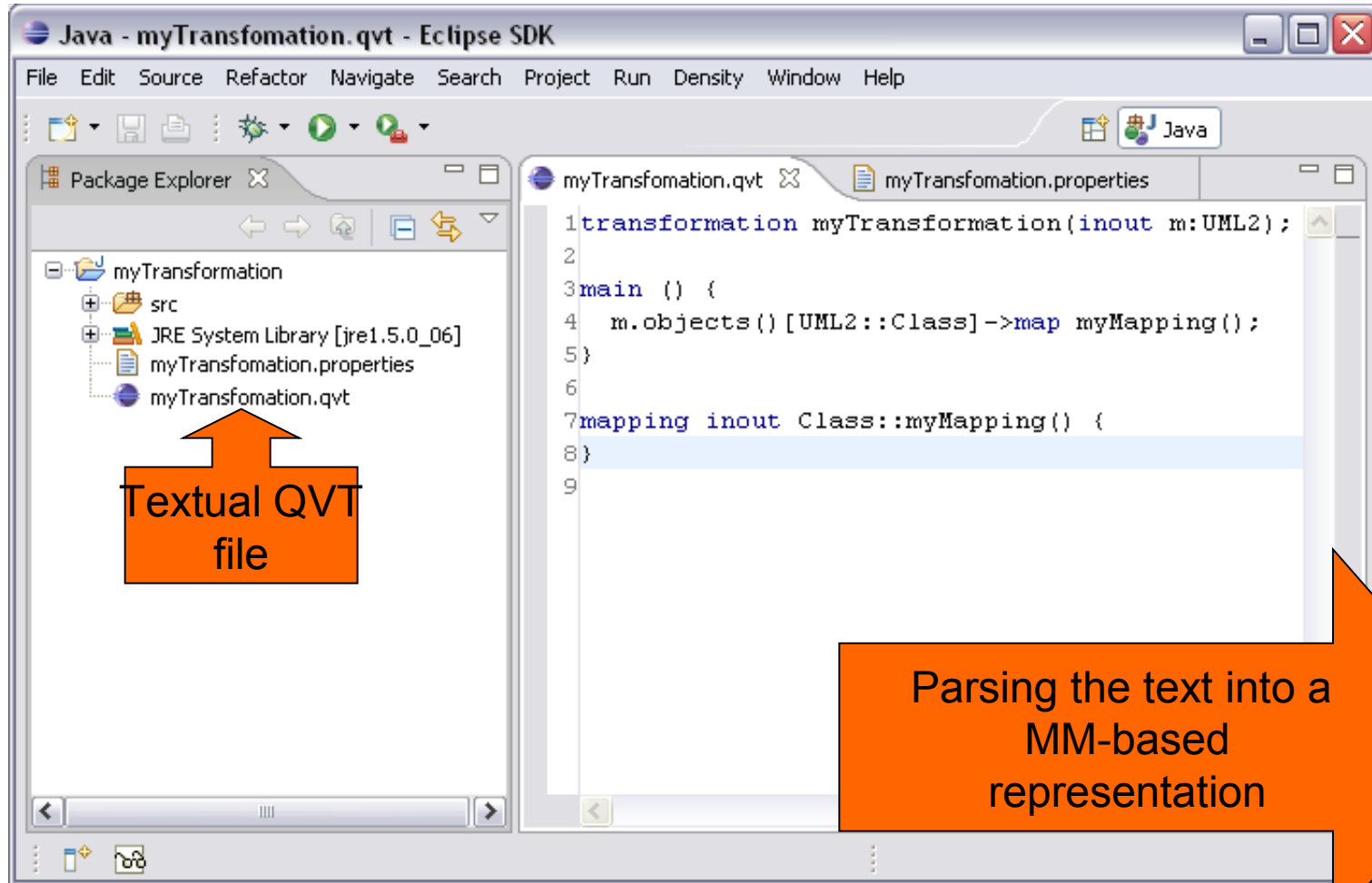


- SmartQVT
 - It is a tool developed by France Télécom R&D (independent to Eclipse).
 - is directly based on ECLIPSE/EMF framework.
 - implements operational mappings in QVT (QVT/Operational).
 - Generates Java executable code realizing the transformations.
 - Is under EPL license.

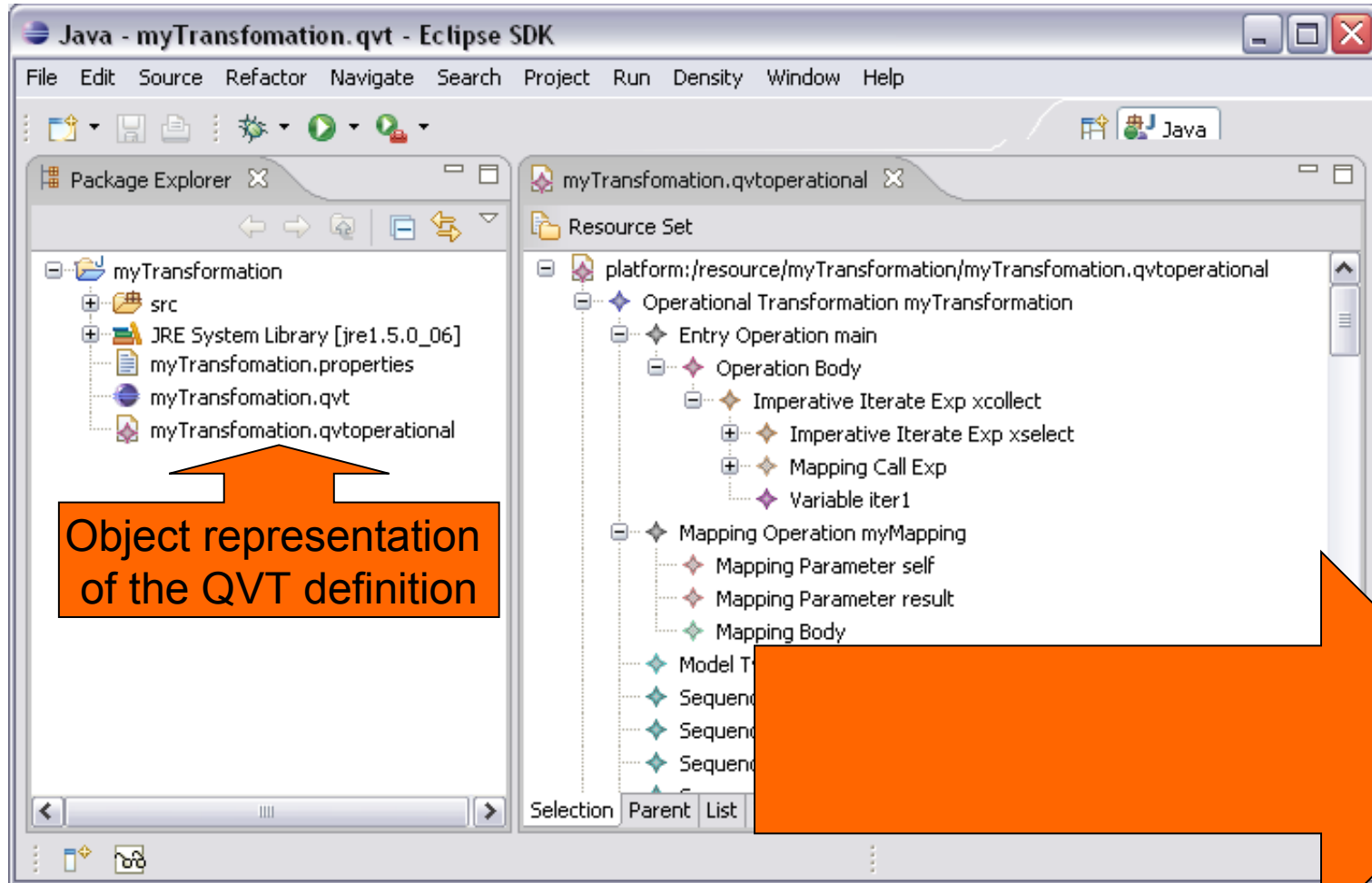
- SmartQVT:
 - Initial release in December 2006,
 - 4 active contributors
 - Frequent releases ~every three months
 - Reaching acceptable maturity – ambition "industrial" quality at the end of 2008.

<http://smartqvt.elibel.tm.fr/>

SmartQVT: How does it works ?



SmartQVT: How does it works ?



Object representation
of the QVT definition

SmartQVT: How does it works ?



Java - myTransformation.qvt - Eclipse SDK

File Edit Source Refactor Navigate Search Project Run Density Window Help

Package Explorer

- modelLib
- myTransformation
 - src
 - mytransformation
 - MyTransformation.java
 - MyTransformationAction.java
 - MyTransformationPlugin.java
 - JRE System Library [jre1.5.0_06]
 - Plug-in Dependencies
 - META-INF
 - tmp
 - myTransformation.properties
 - myTransformation.qvt
 - myTransformation.qvtoperational
 - myTransformation.launch
 - plugin.xml

Overview

General Information

This section describes general information about this plug-in.

ID:

Name:

Provider:

Platform filter:

Activator: Browse...

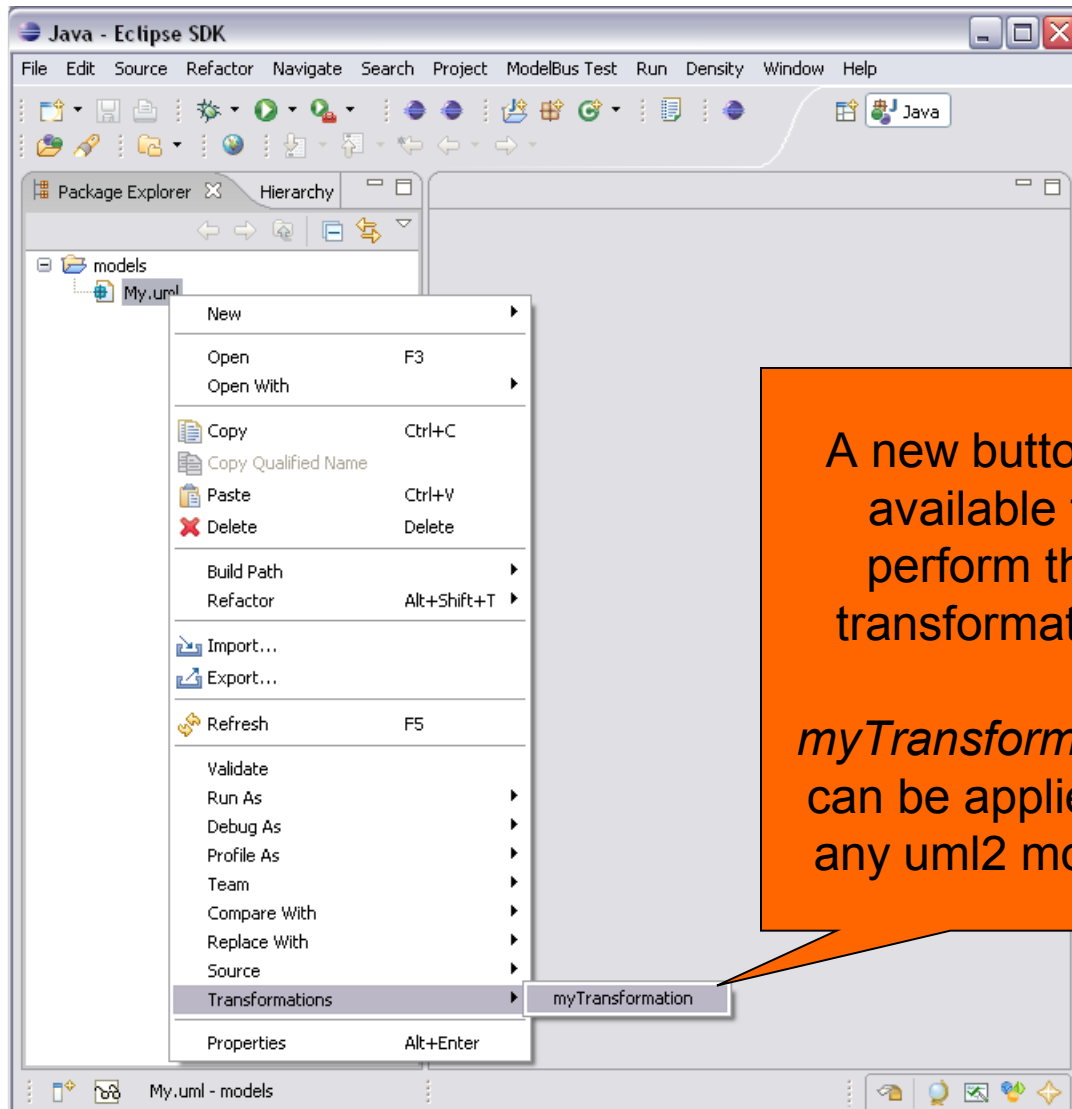
Activate the plug-in

Plugin plugin.xml

Java file

Installed *myTransformation* plug-in

SmartQVT: How does it works ?



A new button is available to perform the transformation

myTransformation can be applied to any uml2 model.



SmartGen

QVT based code generator

Orange Labs

Friday, January 18, 2008

Sebastien Poivre, Research & Development



unrestricted



contents

section 1	presentation
section 2	principle
section 3	meta-model
section 4	sample model and templates
section 5	serializer

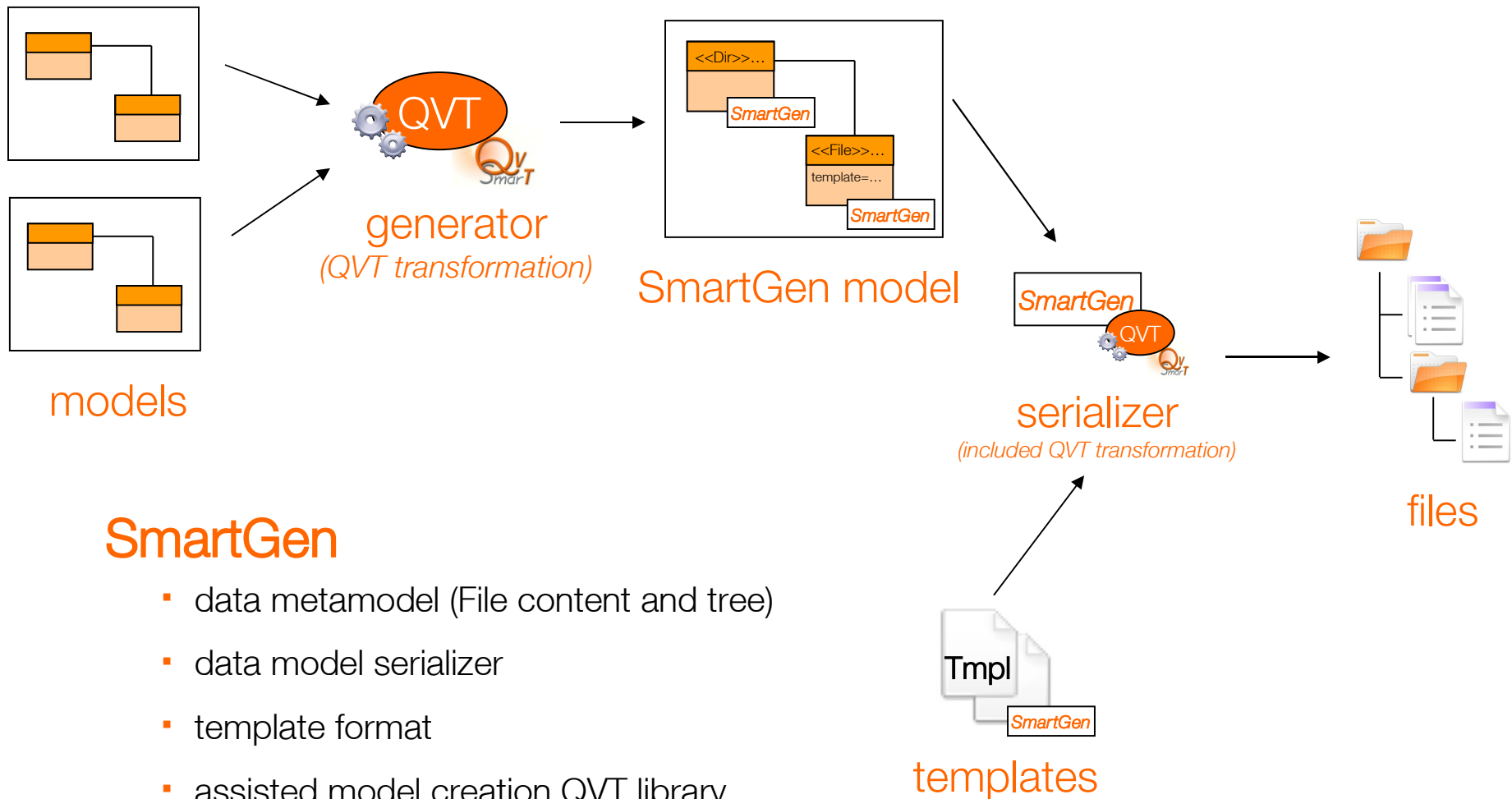
SmartGen presentation

- **code generation tool**
 - produces text **files** and their **directories** from models
 - **model** driven tool
 - based on SmartQVT
 - **same language (QVT)** for model to model transformation and code generation
 - code generation using **templates**
 - SmartGen templates: texts containing slots, filled by model data
 - brings flexibility to the generation process.
 - allows to design separately generation and target templates

- **structured code generation approach to complete SmartQVT capabilities**

- available on SmartQVT site: <http://smartqvt.elibel.tm.fr>

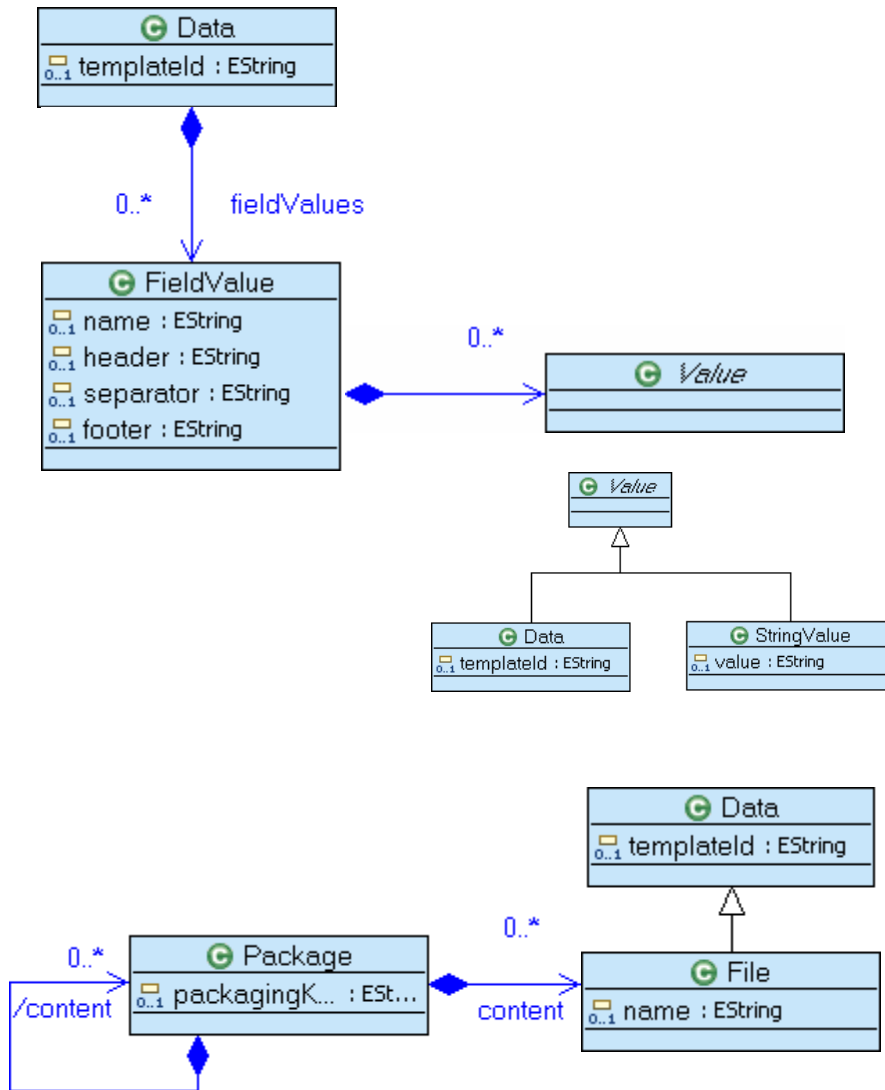
SmartGen principle



SmartGen

- data metamodel (File content and tree)
- data model serializer
- template format
- assisted model creation QVT library
- examples wizard

SmartGen meta-model



- **data**: file, file part ...
 - text part, with content described by a template
- **field value**: name of a place in data:
 - corresponds to a named "hole" in the data template, which can be filled by one or several values
- **value**: either a string, or a file part (data)
- **file**: files are simply data (with a storage name)
 - associated to a template
 - filled by string value and sub-data
- **package**: files are contained in packages

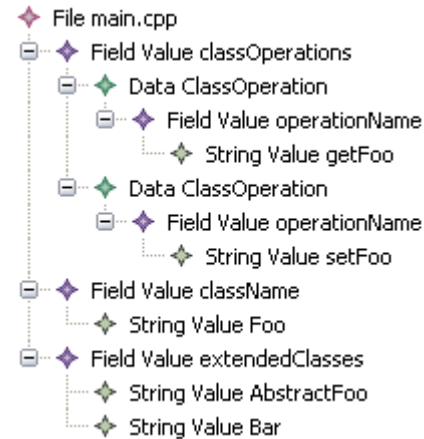
SmartGen sample model and templates

CClass template

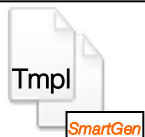
```
<%includes%>
class <%className%>[% extends <%extendedClasses%> %]{
public:
    <%classOperations%>
}
```

CClassOperation template

```
[%<%returnType%> %]<%operationName%>() {
}
```



SmartGen model



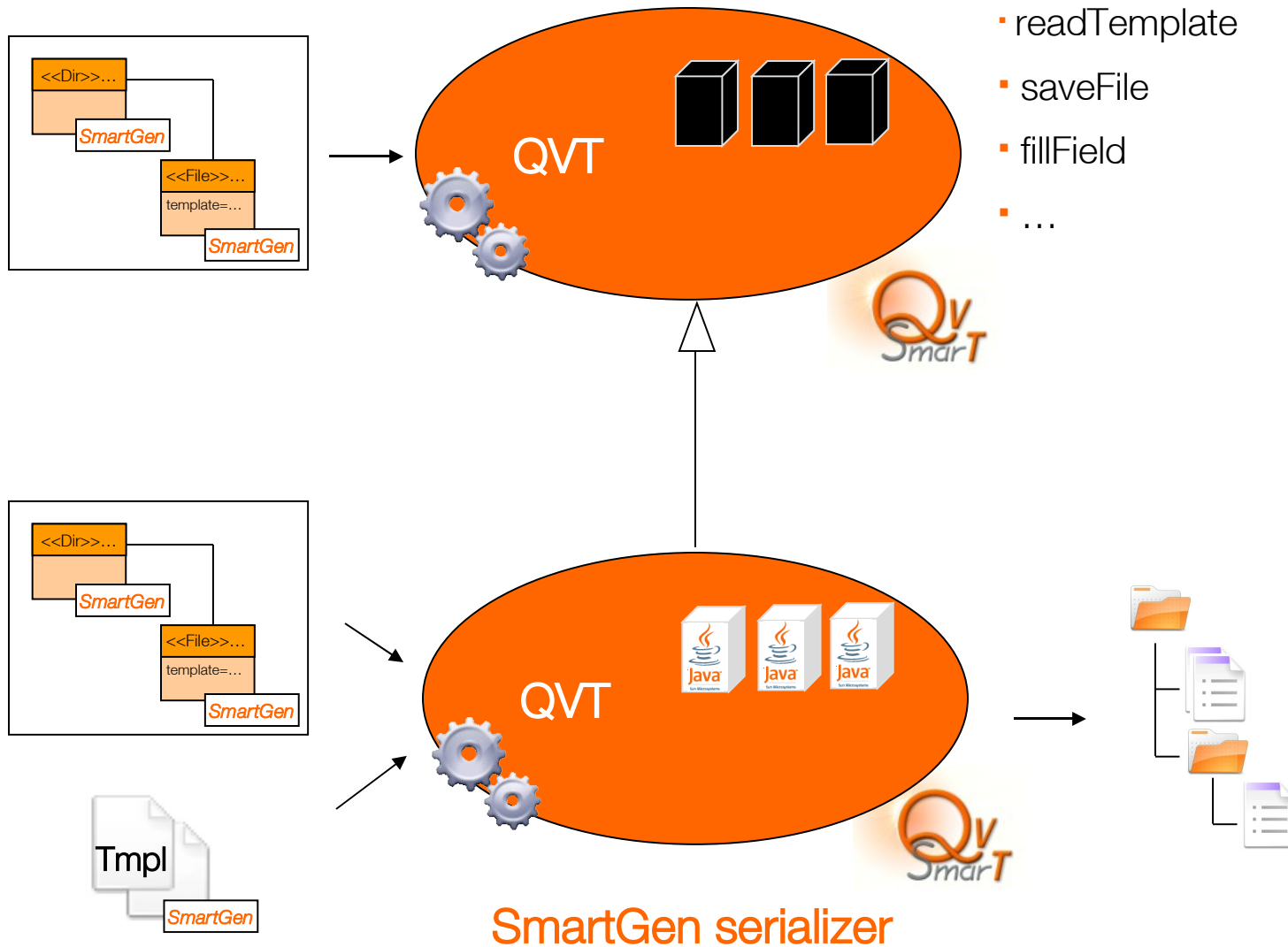
main.cpp

```
class Foo extends AbstractFoo,Bar{
public:
    getFoo() {
    }
    setFoo() {
    }
}
```

SmartGen serializer

QVT Blackboxes

- readTemplate
- saveFile
- fillField
- ...



SmartGen serializer



QVT Application Cases at France Telecom

Orange Labs

Friday, January 18, 2008
Sebastien Poivre, Research & Development

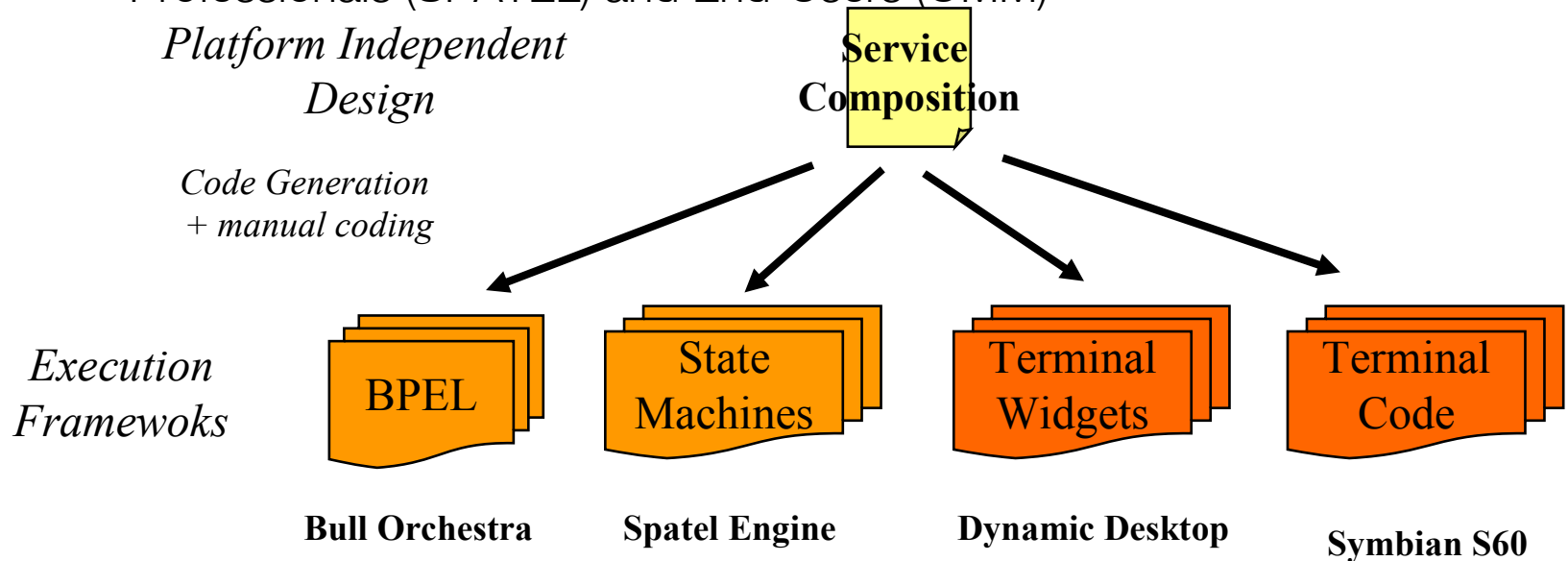


unrestricted



QVT Application domains at France Telecom

- For Application migration
 - Migrating obsolete client/server NSDK to J2EE
- For Service Development
 - Professionals (SPATEL) and End-Users (OMM)



Conclusions and perspectives

- Need to reduce the gap between XML standards and object modeling standards
 - This will make M2M and QVT technology much more attractive in industry context
- MDD and QVT Tools in particular need to reach much more maturity to take benefit of specialized features (agility, maintainability, portability, etc, etc).

thank you



Illustrations

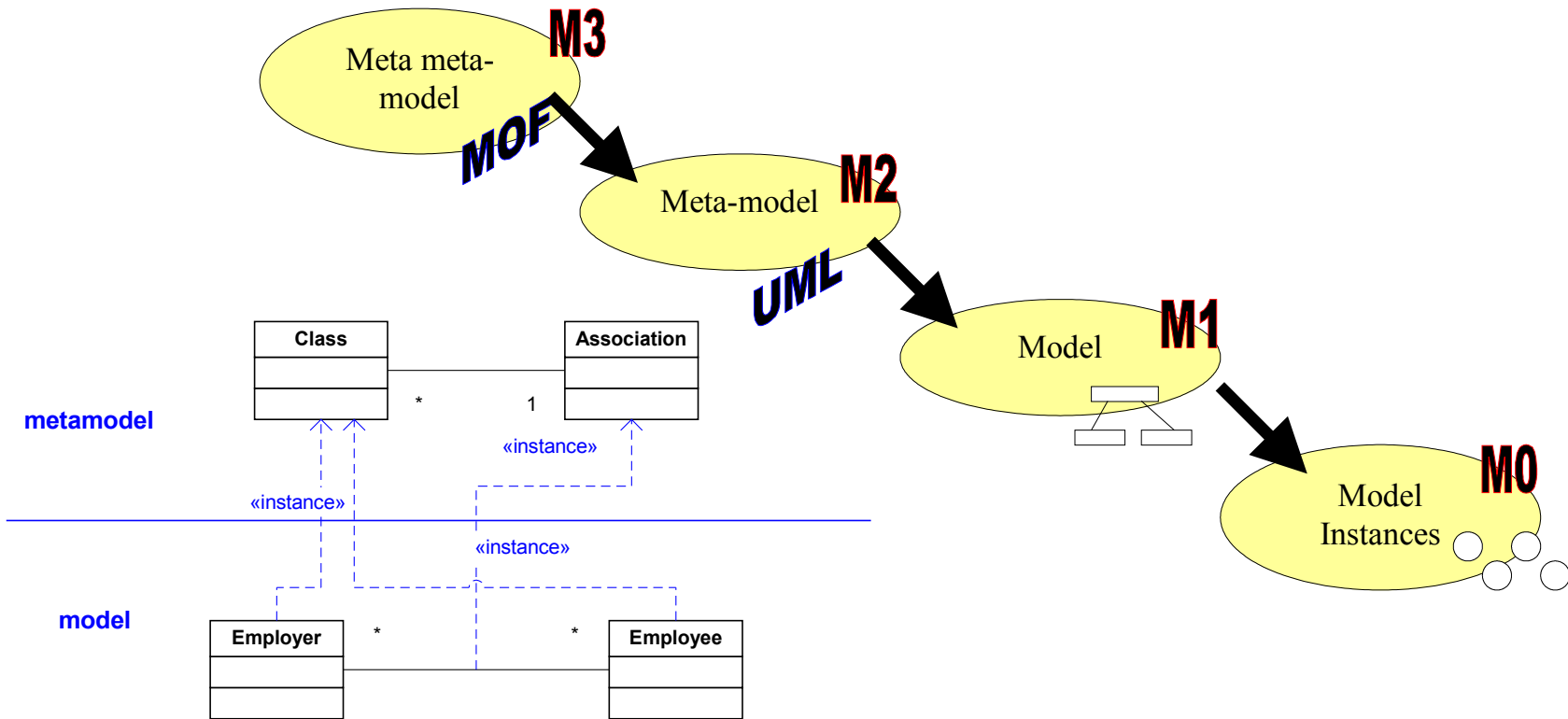


Orange, the Orange mark and any other Orange product or service names referred to in this material are trade marks of Orange Personal Communications Services Limited. © Orange Personal Communications Services Limited.

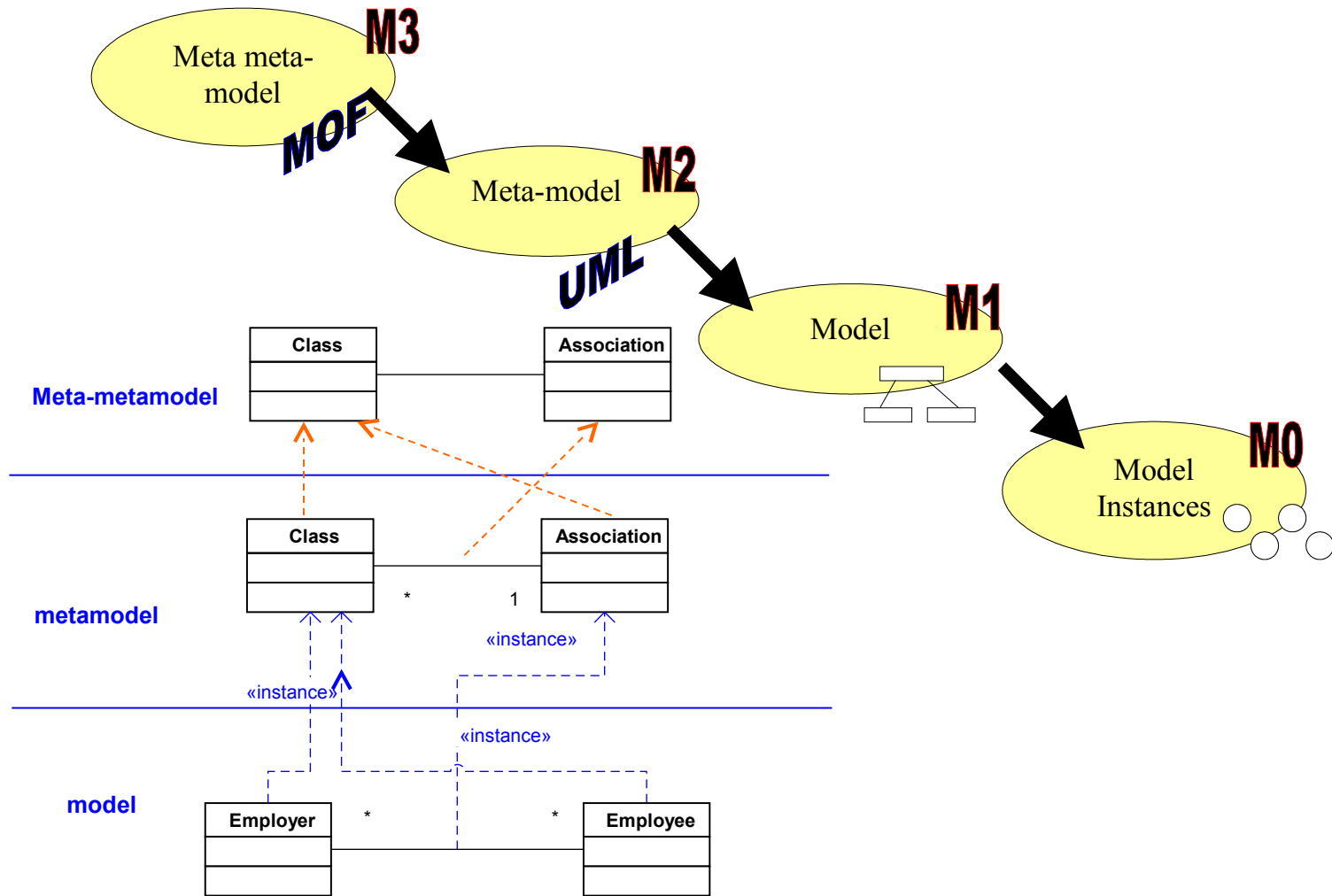
unrestricted. © Orange Personal Communications Services Limited.



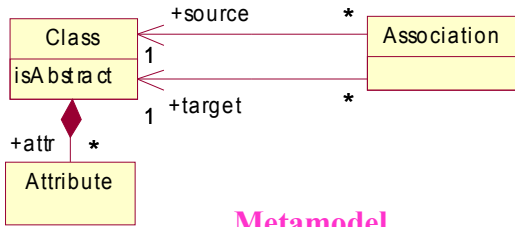
Metamodeling levels (1)



Metamodeling levels (2)



MOF Facilities (1)



Metamodel

```

interface Class {
    Boolean isAbstract();
    setIsAbstract(Boolean);
    addAttr(Attribute);
    removeAttr(Attribute);
    List getAttr();
}

interface Association {
    Class getSource();
    Class getTarget();
    setSource(Class src);
    setTarget(Class tgt);
}
    
```

Generated API

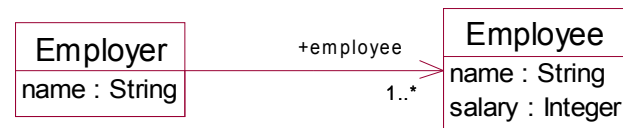
persistence



```

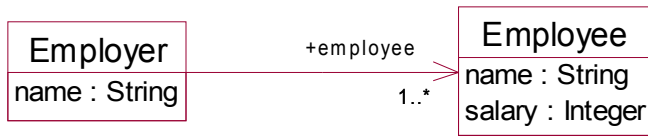
<Class name="Employer" isAbstract="false">
  <attr xmi:type="Attribute" name="name"/>
</Class>
<Class name="Employee" isAbstract="false">
  <attr xmi:type="Attribute" name="name"/>
  <attr xmi:type="Attribute" name="salary"/>
</Class>
<Association source="Employee" dest="Employer"/>
    
```

Instantiation (XMI)



Instantiation (graphical)

MOF Facilities (2)

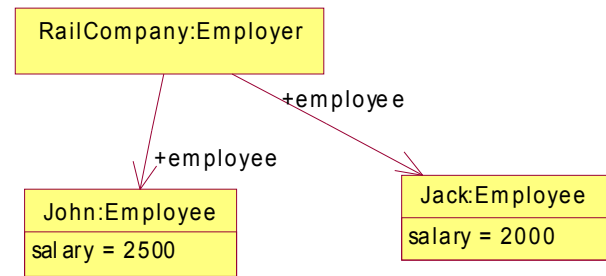


Model

```

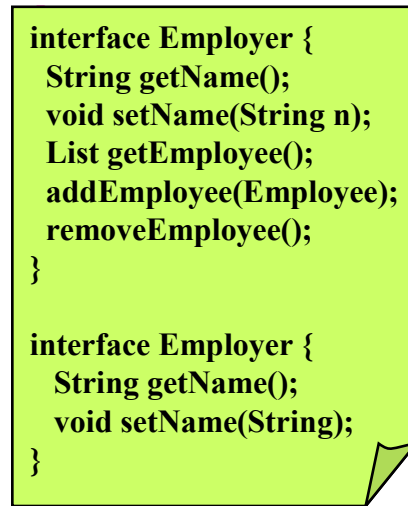
<Employer name="RailCompanu">
  <employee name="John" salary="2500"/>
  <employee name="Jack" salary="2000"/>
</Employer>
    
```

Instantiation (XML)



Instantiation (graphical)

BACK



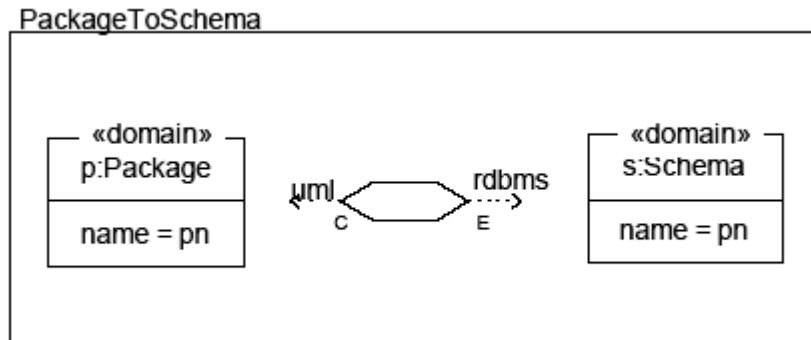
API generated

persistence



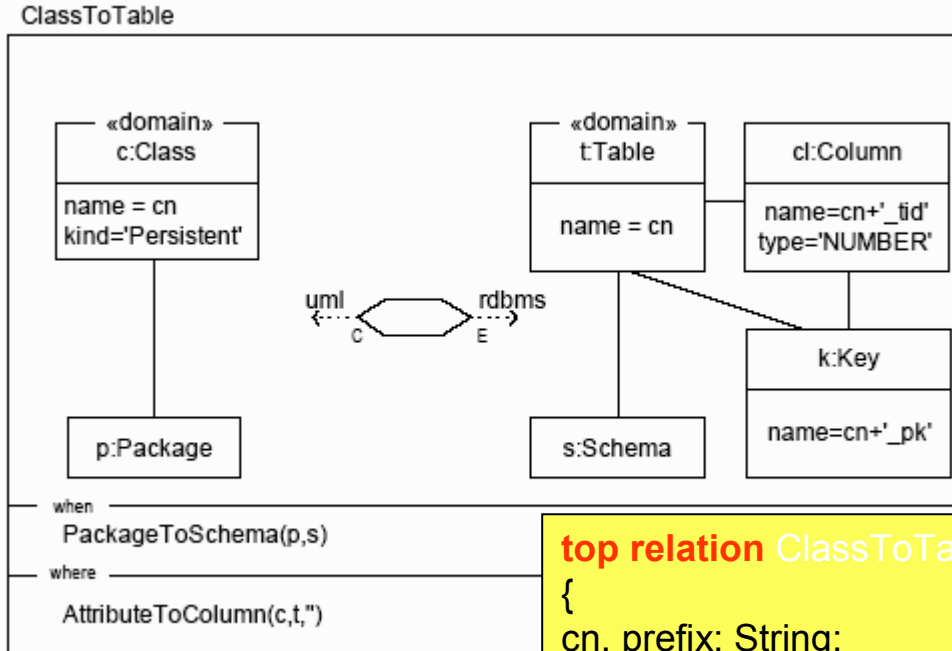
QVT/Rel sample (1)

```
transformation umlToRdbms (uml:SimpleUML, rdbms:SimpleRDBMS);  
key Table (name, schema);  
  
top relation PackageToSchema // map each package to a schema  
{  
  pn: String;  
  checkonly domain uml p:Package {name=pn};  
  enforce domain rdbms s:Schema {name=pn};  
}
```



QVT/Rel sample (2)

BACK



```

top relation ClassToTable // map each persistent class to a table
{
  cn, prefix: String;
  checkonly domain uml c:Class {
    namespace=p:Package {}, kind='Persistent', name=cn};
  enforce domain rdbms t:Table {
    schema=s:Schema {},
    name=cn,
    column=cl:Column {name=cn+'_tid', type='NUMBER'},
    key=k:Key {name=cn+'_pk', column=cl}};
  when { PackageToSchema(p, s); }
  where { prefix = "; AttributeToColumn(c, t, prefix);}
}
    
```

QVT/Op Sample (1)

Signature

Helper Functions

Intermediate data

Entry point

Mapping signature

Inlined Construction

```
transformation Uml2Rdb(in srcModel:UML, out RDBMS);  
query UML::Class.isPersistent() = (self.super.name='PersistentClass');  
  
intermediate UML::Class::leafAttributes : Sequence(LeafAttribute);  
intermediate class LeafAttribute {  
    name:String;kind:String;attr:Attribute;  
};  
main() {  
    srcModel.objects()->map class2table();  
    srcModel.objects()->map asso2table();  
}  
mapping Class::class2table () : Table  
    when {self.isPersistent()}  
{  
    init {self.leafAttributes := self.attribute->attr2LeafAttrs();}  
    name := 't_' + self.name;  
    column := self.leafAttributes->map leafAttr2OrdinaryColumn();  
    key := object Key {  
        name := 'k_' + self.name; column := t.column[kind='primary'];  
    };  
}
```

QVT/Op Sample (2)

```
mapping Association::asso2table() : Table
  when {self.isPersistent()}
  {
    init { result := self.destination.resolveone(Table); }
    foreignKey := self.map asso2ForeignKey();
    column := result.foreignKey.column;
  }
```

BACK

Retrieving
already
created
element

```
mapping Attribute::attr2Column (prefix:String) : Column {
  name := prefix+self.name;
  kind := self.kind;
  type := if self.attr.type.name='int' then 'NUMBER'
         else 'VARCHAR' endif;
}

mapping Attribute::attr2ForeignColumn (prefix:String) : Column
inherits leafAttr2OrdinaryColumn { kind := "foreign";}
```

Rule reuse

Imperative constructs

- **Compute:**

- `compute(x : T) do {`

 }

- **ForEach:**

- `mylist->forEach(i | cond) {`

 }

- **While:**

- `while(cond;acc) {`

 }

Composition of coarse grained transformations

```
transformation Uml2rdf (inout srcmodel:UML,out destmodel:RDF)
  access Uml2annotateduml(UML), Annotateduml2rdf(UML,RDF);
main() {
  var t1 := new Uml2annotateduml(srcmodel);
  if (t1.transform().failed()) then do {
    log("Failed UML transformation",t1.status);
    return;
  };
  var t2 := new Annotateduml2rdf (srcmodel,destmodel);
  if (t2.transform().failed()) then
    log("Failed RDF transformation",t2.status);
}
```

Parallelism in coarse grained transformations

```
transformation Uml2rdf (inout src1:UML,inout src2:UML,out dest1:RDF)
    access Uml2annotateduml(UML), Annotateduml2rdf(UML,RDF);
main() {
    var status1 := new Uml2annotateduml(src1).parallelTransform();
    var status2 := new Uml2annotateduml(src2).parallelTransform();
    status1.wait();
    status2.wait();
    if (status1.failed()) then do {
        log("Failed UML transformation on first model",t1.status);
        return;
    };
    if (status2.failed()) then do {
        log("Failed UML transformation on second model",t1.status);
        return;
    };
    var t := new Annotateduml2rdf (src1,src2,destmodel);
    if (t.transform().failed()) then
        log("Failed RDF transformation",t2.status)
}
```

Mapping Disjunction

```
mapping UML::Feature::convertFeature () : JAVA::Element
    disjuncts convertAttribute, convertOperation,
                convertConstructor() {}
mapping UML::Attribute::convertAttribute : JAVA::Field {
    name := self.name;
}
mapping UML::Operation::convertConstructor : JAVA::Constructor
when {self.name = self.namespace.name;} {
    name := self.name;
}
mapping UML::Operation::convertOperation : JAVA::Constructor
when {self.name <> self.namespace.name;} {
    name := self.name;
}
```

AsTransformation

```
transformation PimToPsm(inout pim:PIM, in transfSpec:UML, out psm:PSM)
access UmlGraphicToQvt(in uml:UML, out qvt:QVT)
access AnnotatedPimToPsm(in pim:PIM, out psm:PSM);
main() {
    transfSpec->objectsOfType(Package)->forEach(umlSpec:UML) {
var qvtSpec : QVT;
var retcode := new UmlGraphicToQvt(umlSpec,qvtSpec).transform();
if (retcode.failed()) {
    log("Generation of the QVT definition has failed",umlSpec); return;};
if (var transf := qvtSpec.asTransformation()) {
    log("Instanciation of the QVT definition has failed",umlSpec); return;};
if ( transf.transform(pimModel,psmModel).failed()) {
    log("failed transformation for package spec:",umlSpec); return;};
    }
    }
}
```

