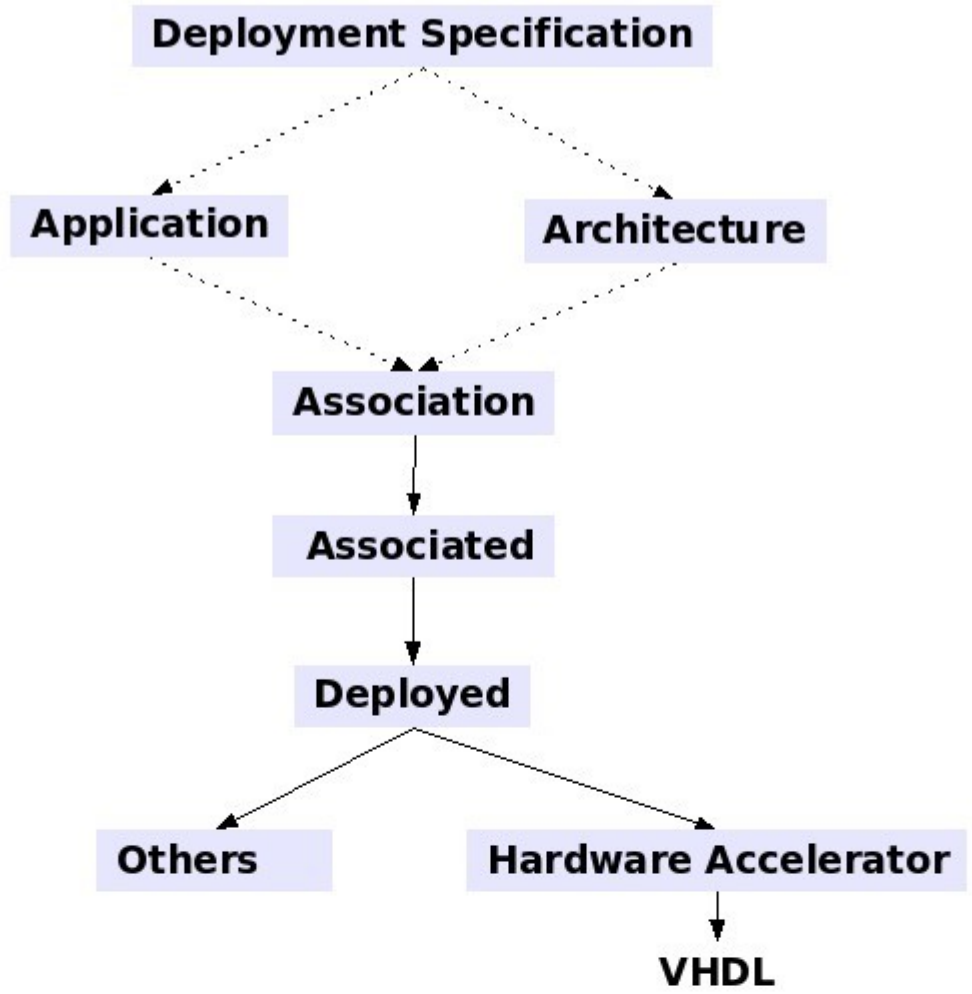
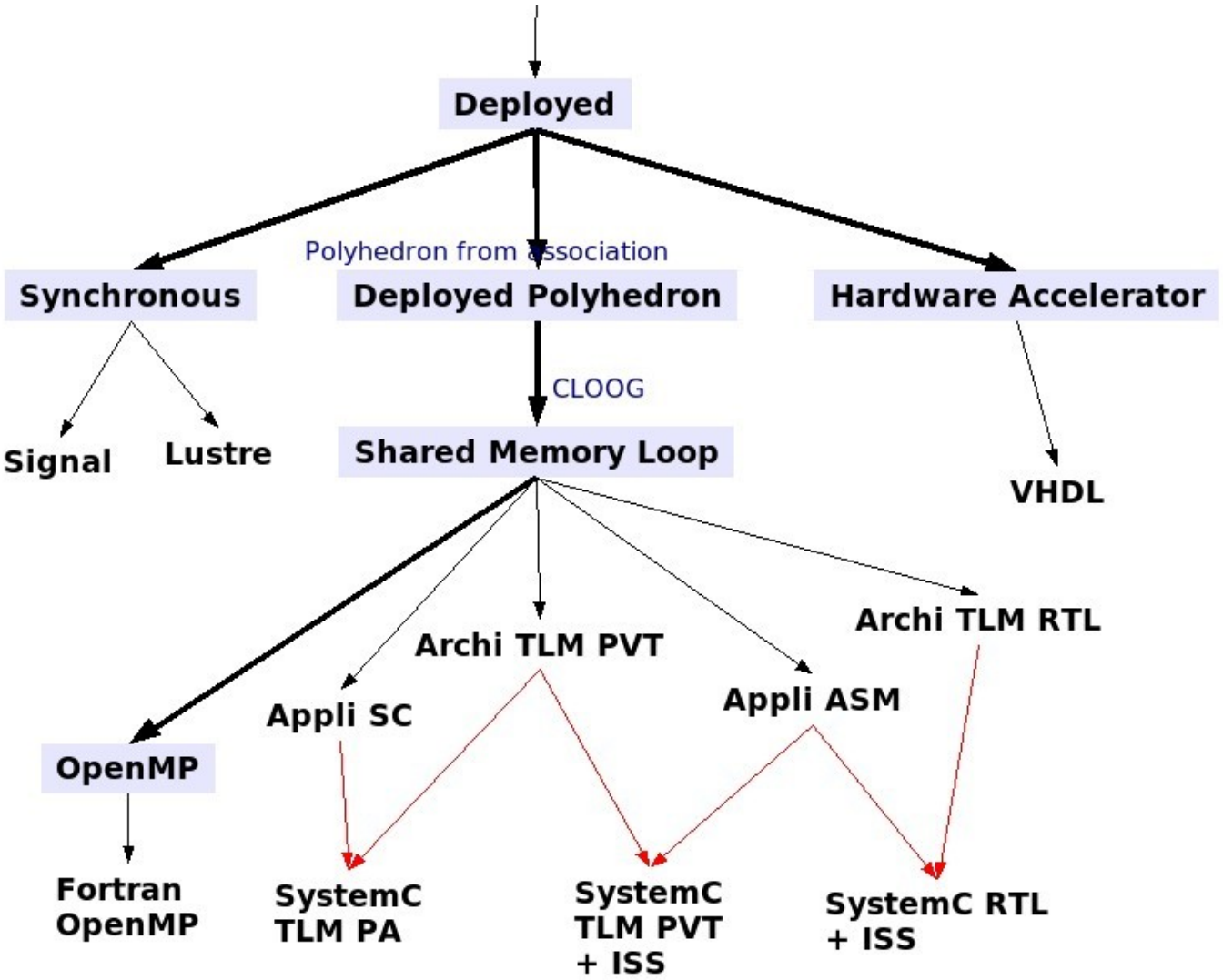
# Model Transformations in Gaspard2


Anne Etien
Post Doc
etien@lifl.fr


# Context: Gaspard 2



# Context: Gaspard 2



# Problematic

- Two issues:
  - Communicate and exchange about transformations
  - Execute the transformations

Two different but complementary solutions

# Outline


1. Presentation of TrML Profile
2. Presentation of MoMoTE
3. Conclusion and Perspectives



# Presentation of TrML Profile

*Transformation Modelling Language*


# Lack of graphical notation


- Engine like ATL
  - But no graphical notation
- Procedural graphical flow chart like MOLA
  - But difficulty to apprehend
- QVT standart
  - But not immediatly useable

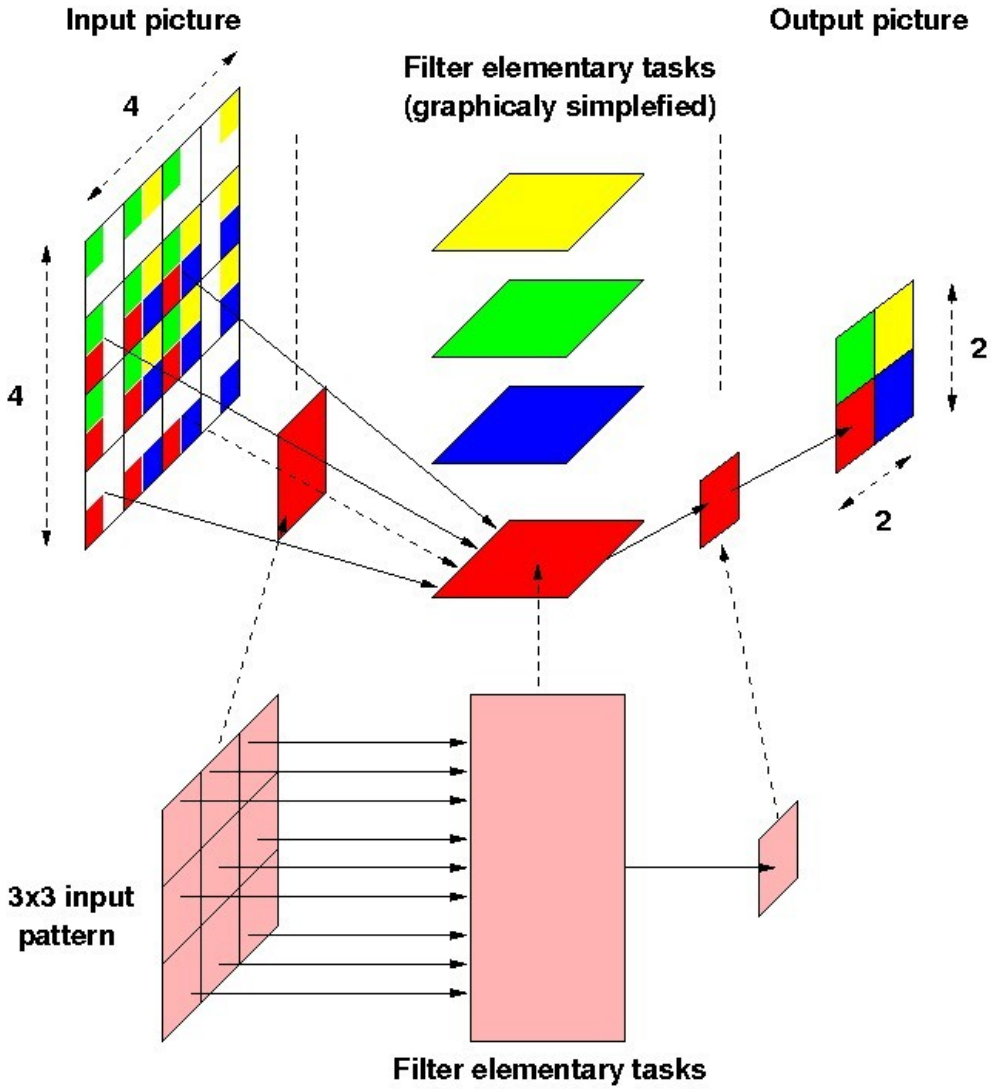

# TrML profile

- Extension of the UML class diagram
- Adapted for transformation description
- QVT compliant
  - Rule and pattern based
- Adapted from ModTransf
- Easily useable
- Possibility to use the UML2 modeler (MagicDraw, Rose, Objecteering…)

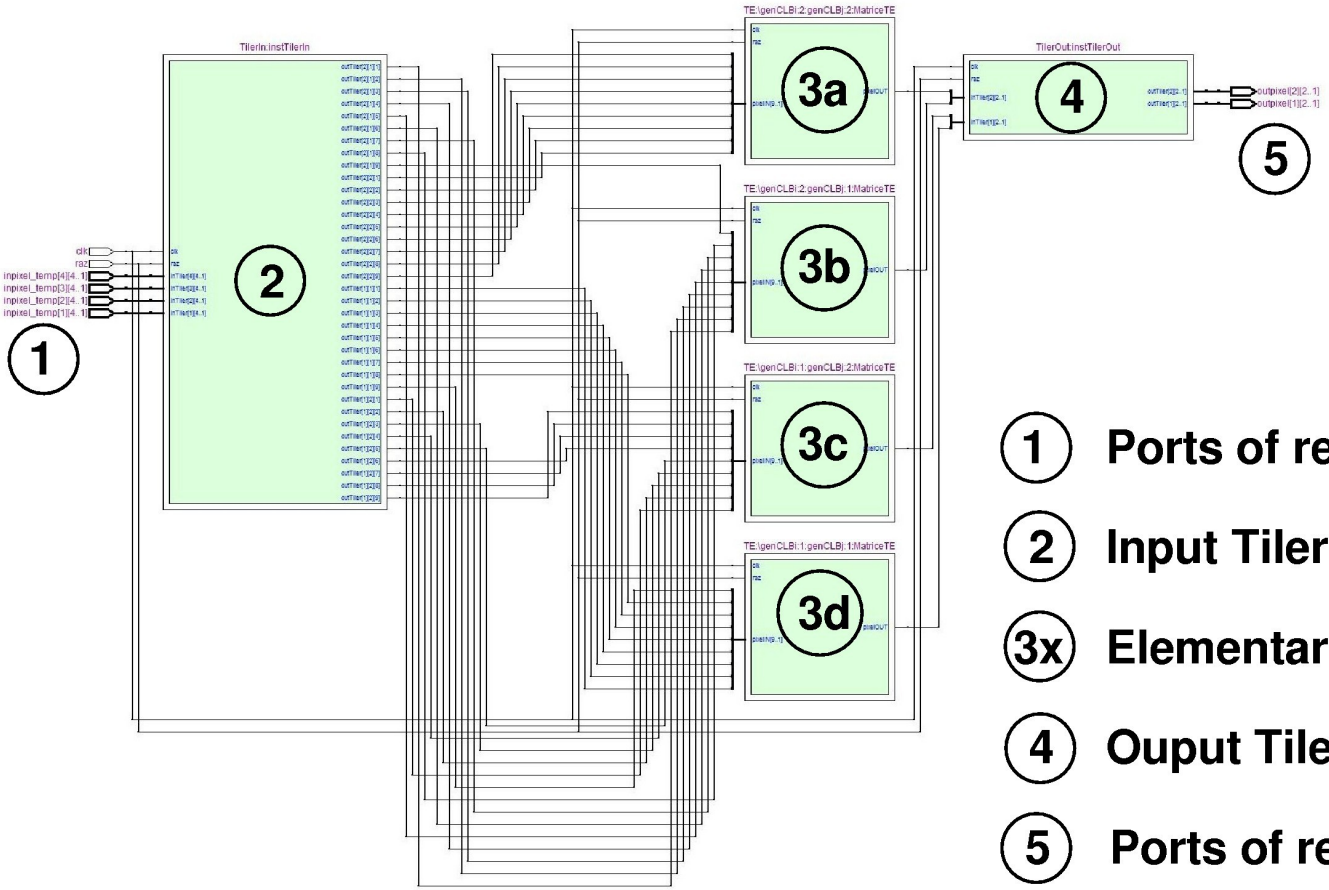
# TrML characteristics


- Multi models as sources or targets
- Transformation decomposed into rules
- Model decomposed into patterns
  - A pattern is a set of model elements necessary to the rule description
- Bind names associated to elements
- Actions to specify the target patterns generation
- Guards to restrict the applicability of the rule


# The image filter example



# The image filter example (2)




1. **Ports of repetitive**
2. **Input Tiler**
3. **Elementary Task instance**
4. **Ouput Tiler**
5. **Ports of repetitive**

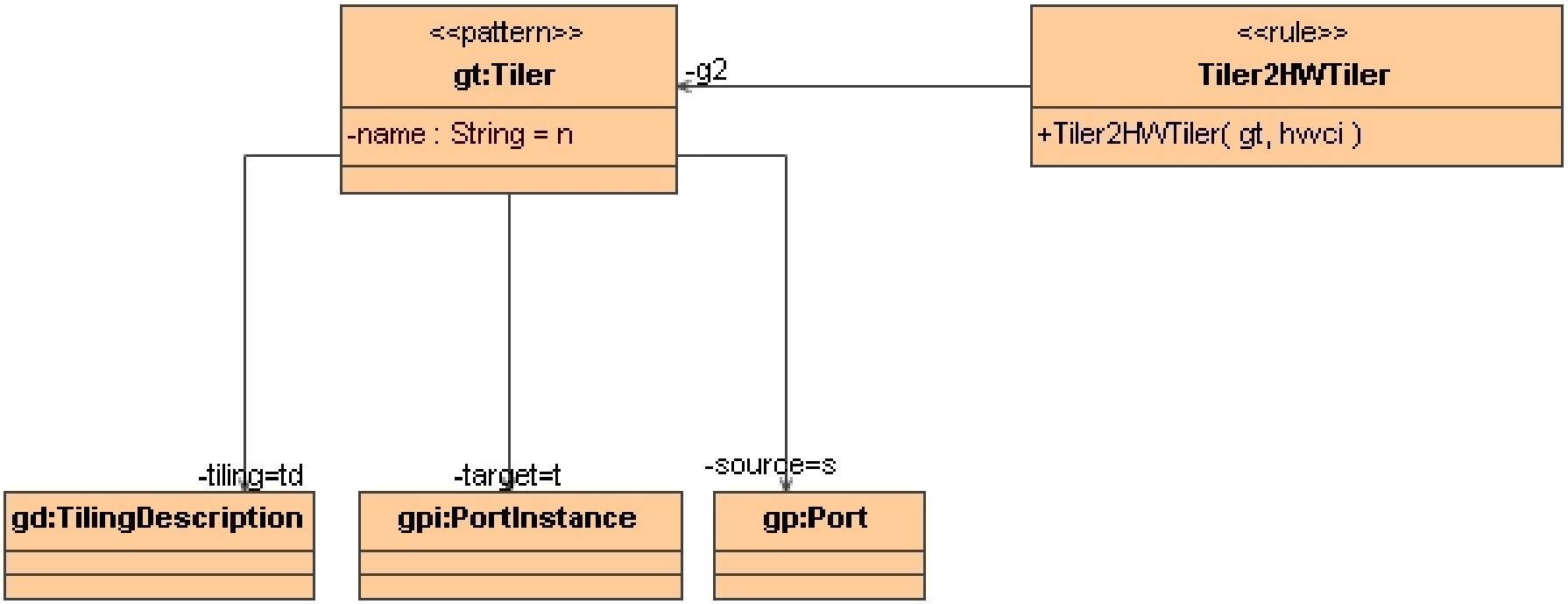
# Rules Decomposition

- Instinctively: complete description of the Repetitive Component in the same rule
- But some issues:
  - Complexity of the rule (illisibility)
  - Lack of reusability
  - Limits of the graphical notation
- Rules Decomposition
  - From large grain to finer grain
  - Top-down approach induced by the profile

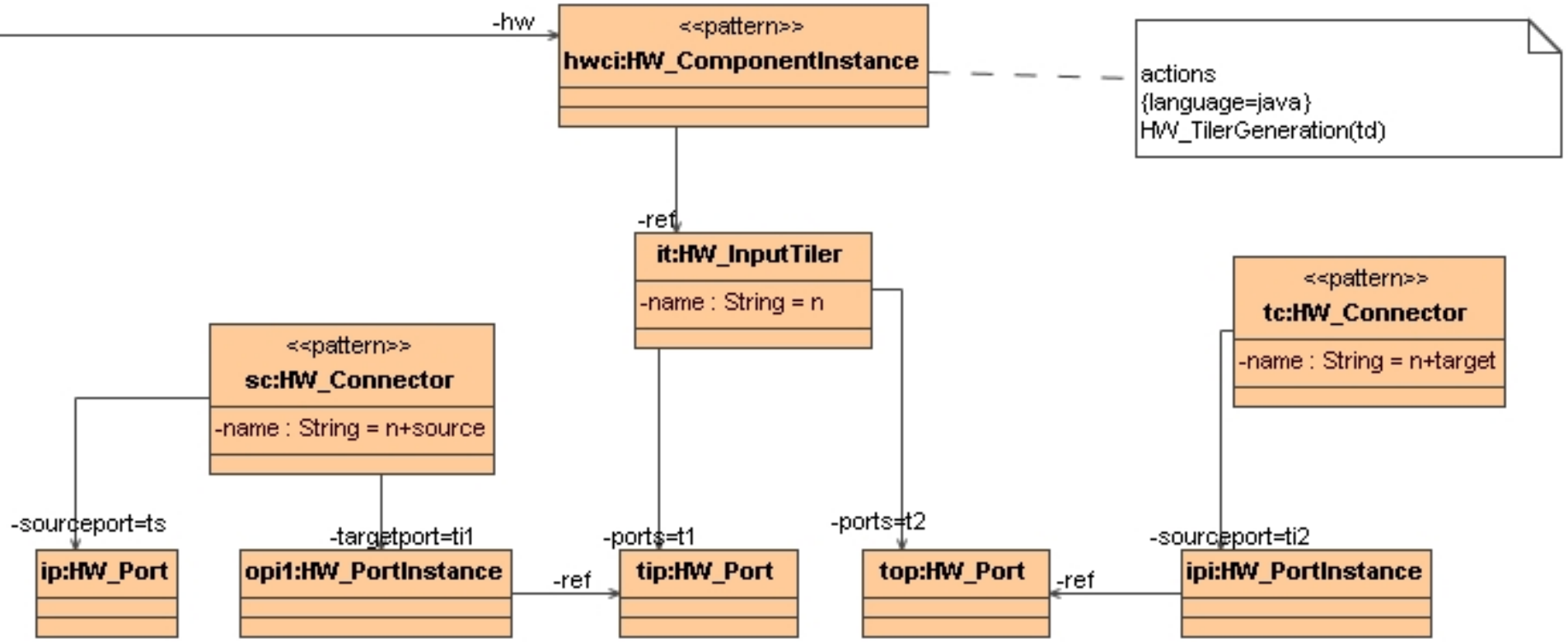
# The Tiler2HWTiler rule

- Context: transformation of a Repetitive Component
- Concept of rule
- Signature of the rule to specify the application context
- Specialisation of the origin models
- Description of the patterns

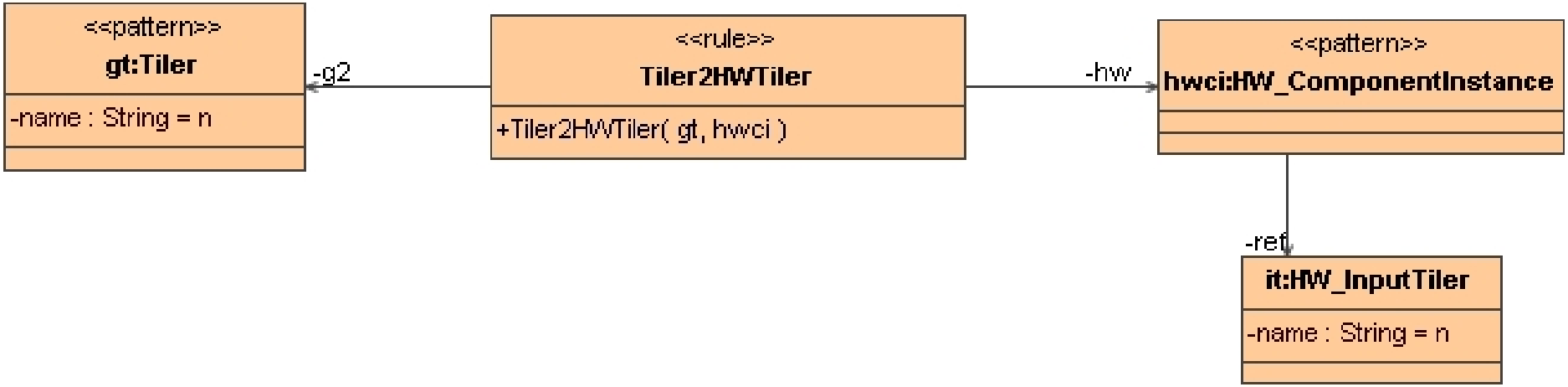
# The source pattern



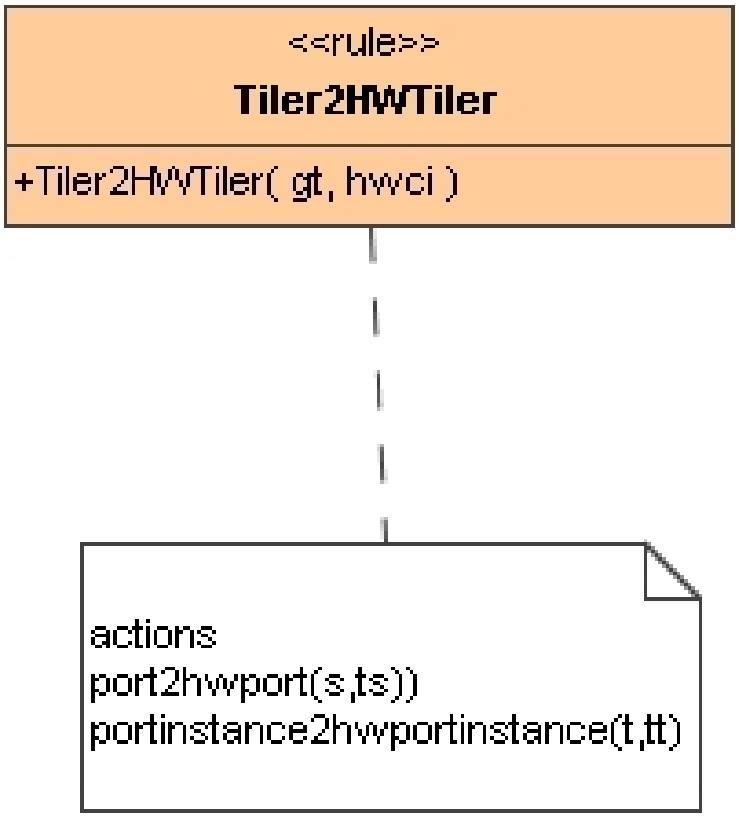
# The target pattern



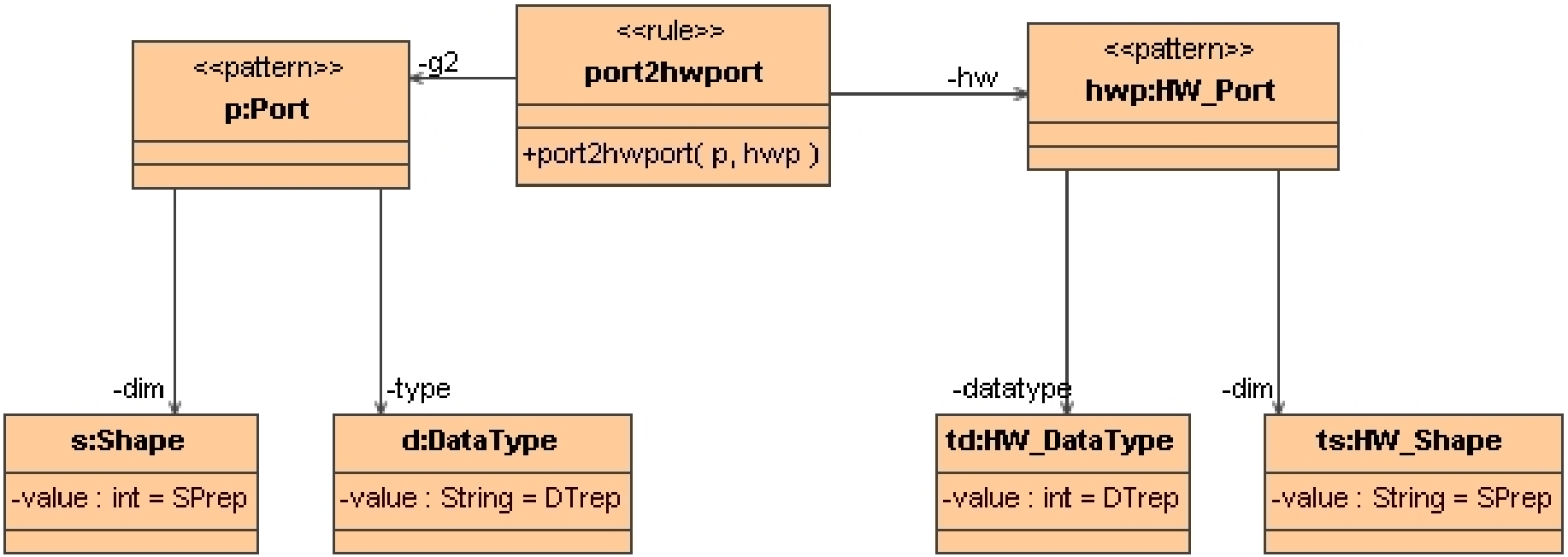
# A usage of Bind name



# The rull call



# The port2HWPortRule



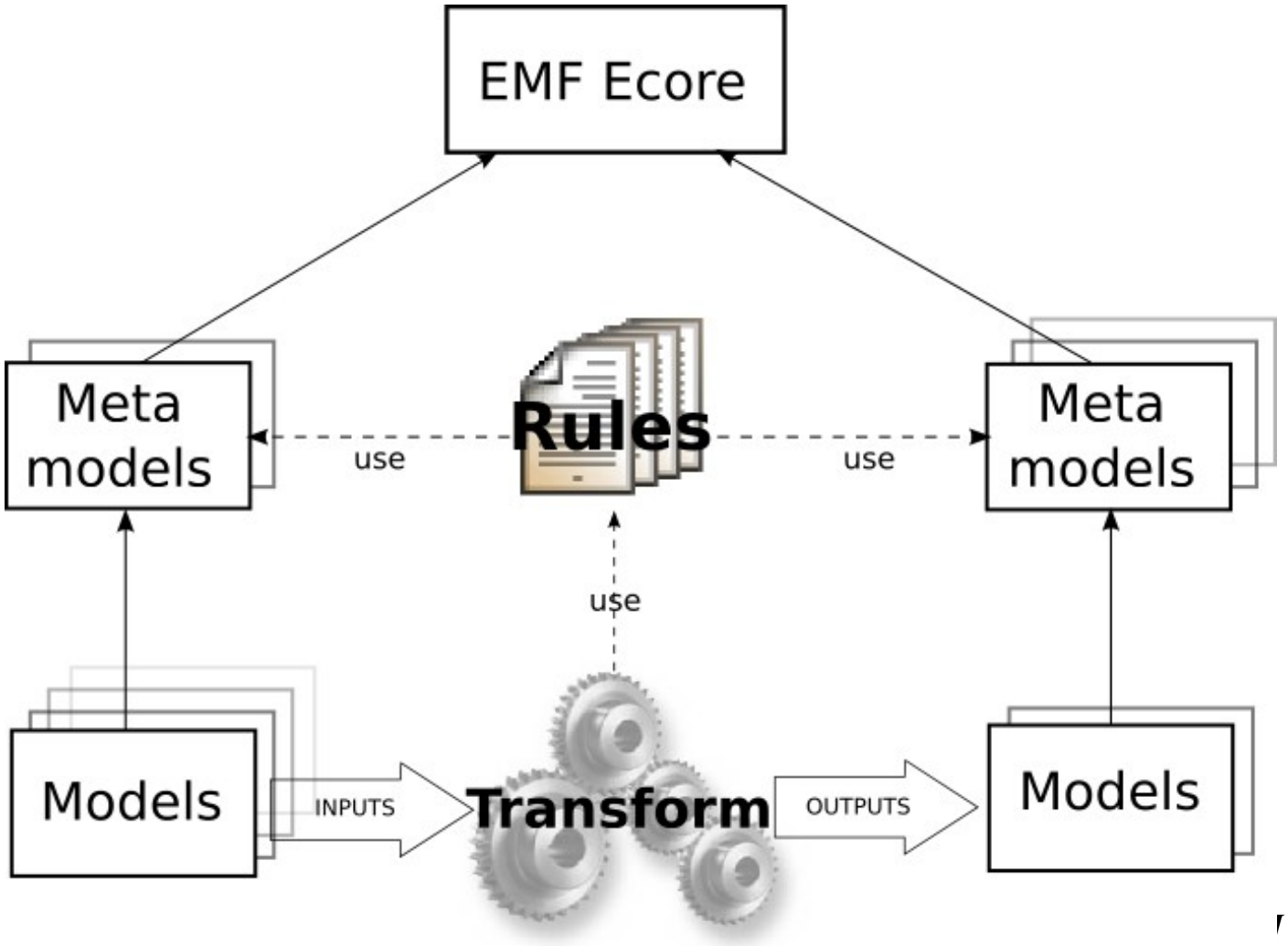

# Presentation of MoMoTE

## *Model to Model Transformation Engine*


# Technology


- Java API
- Eclipse plugin
- Based on:
  - Java 5.0
  - EMF 2.2.0 to manipulate the model
  - EMFT Query 1.0.0 (OMG standard QVT applied to Ecore meta-metamodel) to query the model


# MoMoTE Overview



# MoMoTE Characteristics


- *N* input models to *M* output models
- Domain application of a rule: either an element of the calling rule or the complete model
- Rules can create a new output model root
- Two passes:
  - One pass to create output model elements
  - One pass to solve model element references

lifL

INRIA FUTURS

# The port2HWPort Rule

```
public class Port2HWPort extends Rule
{
    public Port2Port()
    {
        setQueryFromContext(true);
    }
    protected EObject create(EObject srcElement)
    {
        return HardwareAcceleratorFactory.eINSTANCE.createPort();
    }
protected EObjectCondition getCondition(EObject srcElementContext)
    {
        return new
            EObjectTypeRelationCondition(Gaspard2Package.eINSTANCE.getPort(
            ), TypeRelation.SUBTYPE_LITERAL));
    }
```

Constructor

Methods to overload

# The port2HWPort Rule

Same name in source pattern and target pattern

```
protected void process(EObject srcElement, EObject tgtElement)
{
    gaspard2.Port srcPort = (gaspard2.Port) srcElement;
    hardwareaccelerator.Port tgtPort = (hardwareaccelerator.Port) tgtElement;
    tgtPort.setName(srcPort.getName());
}
protected void processReferences(EObject srcElement, EObject
    tgtElement)
{
    gaspard2.Port srcPort = (gaspard2.Port) srcElement;
    hardwareaccelerator.Port tgtPort = (hardwareaccelerator.Port) tgtElement;
    tgtPort.setType(getTransformation().getGlobalRefs().get(srcPort.getType())
        );
}
}
```

- Conclusion and Perspectives

# Conclusion


- Presentation of a UML2 profile for model transformation description
- Improvement of exchange and communication about transformations
- TrML is based on a meta-model
- MoMoTE helps to generate and executes model transformations
- Easily available via a Eclipse plugin


# Perspectives


- Some separated future works
- Toward a common future:

**Merge MoMoTE with TrML**