

Leveraging Executable Language Engineering for Domain-Specific Transformation Languages (Position Paper) EXE 2016, Saint-Malo, France

Erwan Bousse¹ Manuel Wimmer¹ Wieland Schwinger²
Elisabeth Kapsammer²

¹TU Wien, Austria

²JKU Linz, Austria

October 3, 2016

Observations

- **Domain Specific Transformation Language (DSTL)** = model transformation language tailored for specific tasks (eg. strings renaming, code generation)
- DSTLs more and more common:
 - Two papers on DSTLs at *ICMT'16* in a dedicated “*Model Transformation Languages*” session
 - This year *TTC'16* use case: data-flow based DSTL
 - **Increasing need for methods to develop DSTLs**
- Progress in executable Domain-Specific Modeling language (xDSML) engineering:
 - Generic *syntactic* services (eg. editors using Xtext or Sirius)
 - Generic *runtime* services (eg. debugger using GEMOC studio)
 - **Easier and easier to obtain a tool-supported xDSML**

Observations

- **Domain Specific Transformation Language (DSTL)** = model transformation language tailored for specific tasks (eg. strings renaming, code generation)
- DSTLs more and more common:
 - Two papers on DSTLs at *ICMT'16* in a dedicated “*Model Transformation Languages*” session
 - This year *TTC'16* use case: data-flow based DSTL
 - **Increasing need for methods to develop DSTLs**
- Progress in executable Domain-Specific Modeling language (xDSML) engineering:
 - Generic *syntactic* services (eg. editors using Xtext or Sirius)
 - Generic *runtime* services (eg. debugger using GEMOC studio)
 - **Easier and easier to obtain a tool-supported xDSML**

Observations

- **Domain Specific Transformation Language (DSTL)** = model transformation language tailored for specific tasks (eg. strings renaming, code generation)
- DSTLs more and more common:
 - Two papers on DSTLs at *ICMT'16* in a dedicated “*Model Transformation Languages*” session
 - This year *TTC'16* use case: data-flow based DSTL
 - **Increasing need for methods to develop DSTLs**
- Progress in executable Domain-Specific Modeling language (xDSML) engineering:
 - Generic *syntactic* services (eg. editors using Xtext or Sirius)
 - Generic *runtime* services (eg. debugger using GEMOC studio)
 - **Easier and easier to obtain a tool-supported xDSML**

Questions

Is it possible to apply techniques from xDSML engineering to define DSTLs?

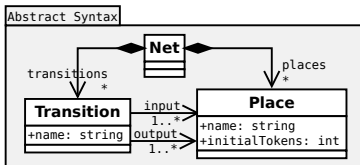
How are xDSMLs and DSTLs related?

Questions

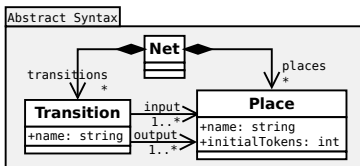
Is it possible to apply techniques from xDSML engineering to define DSTLs?

How are xDSMLs and DSTLs related?

Example of Petri nets xDSML and model

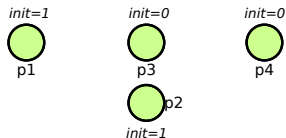
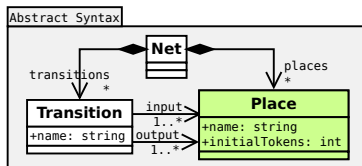


Example of Petri nets xDSML and model



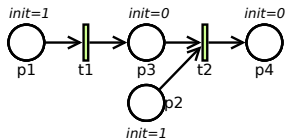
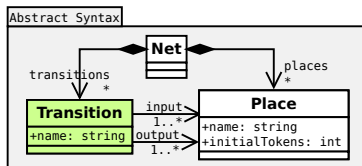
Petri net model

Example of Petri nets xDSML and model



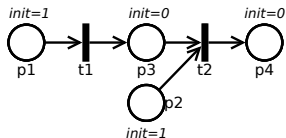
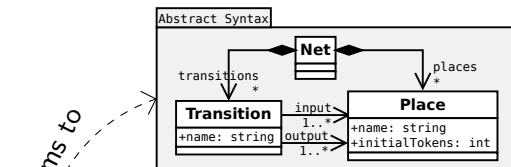
Petri net model

Example of Petri nets xDSML and model



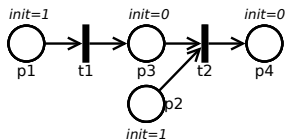
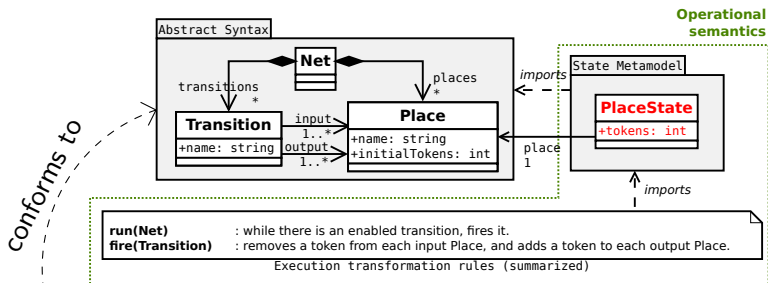
Petri net model

Example of Petri nets xDSML and model



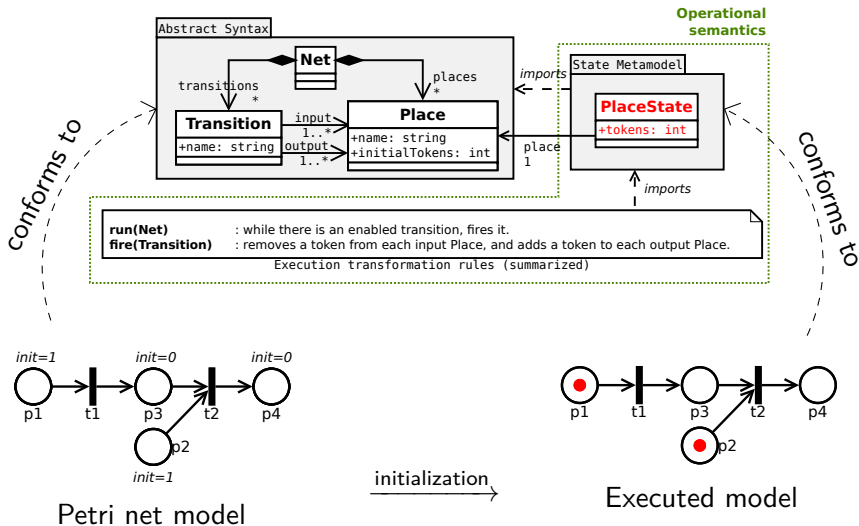
Petri net model

Example of Petri nets xDSML and model

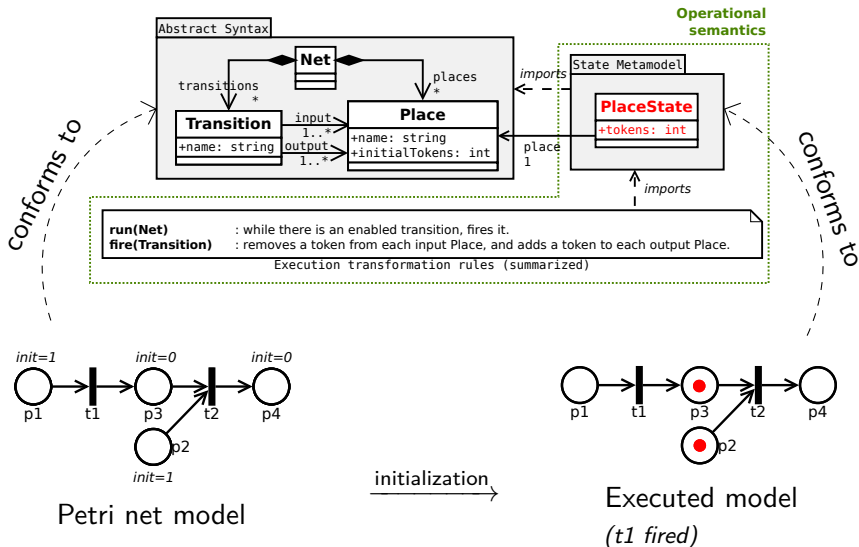


Petri net model

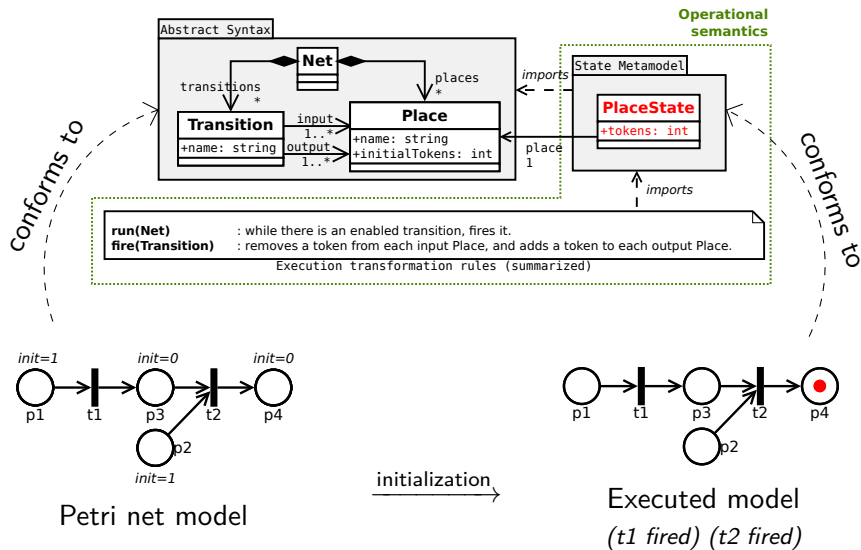
Example of Petri nets xDSML and model



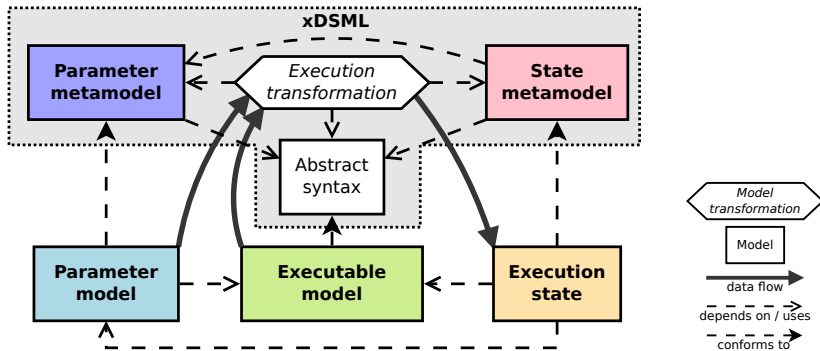
Example of Petri nets xDSML and model



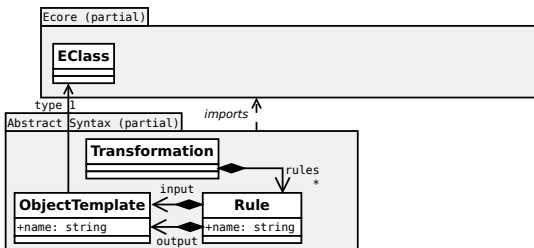
Example of Petri nets xDSML and model



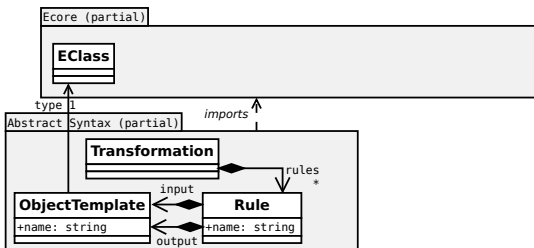
Generalizing xDSMLs



Example of MiniTL DSTL and model

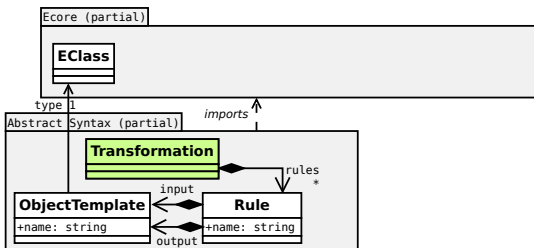


Example of MiniTL DSTL and model



MiniTL model

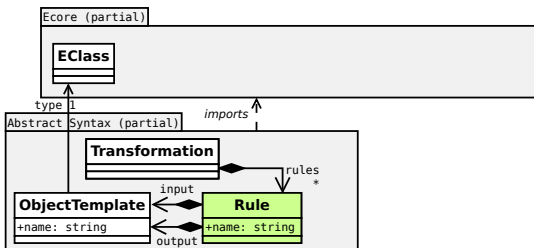
Example of MiniTL DSTL and model



```
transformation simpleAtoB {  
  
}
```

MiniTL model

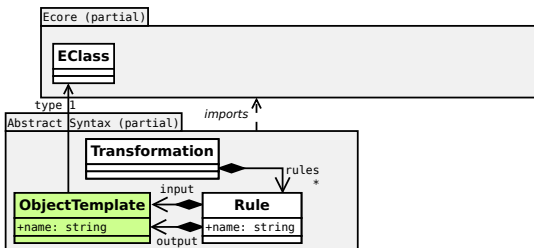
Example of MiniTL DSTL and model



```
transformation simpleAtoB {  
  rule AToB {  
    }  
}
```

MiniTL model

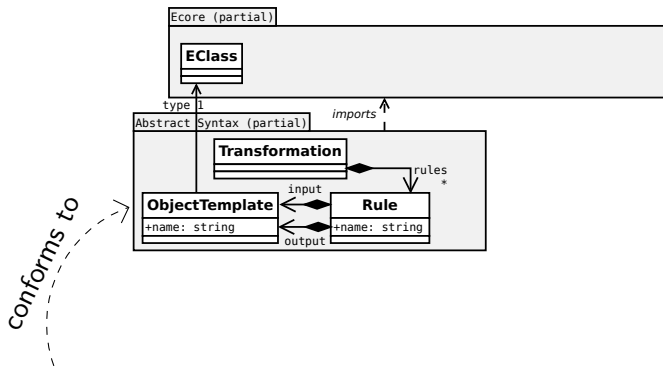
Example of MiniTL DSTL and model



```
transformation simpleAtoB {  
  rule AtoB {  
    from a : metamodelA.A  
    to b : metamodelB.B {  
      y = a.x + "_out";  
    }  
  }  
}
```

MiniTL model

Example of MiniTL DSTL and model



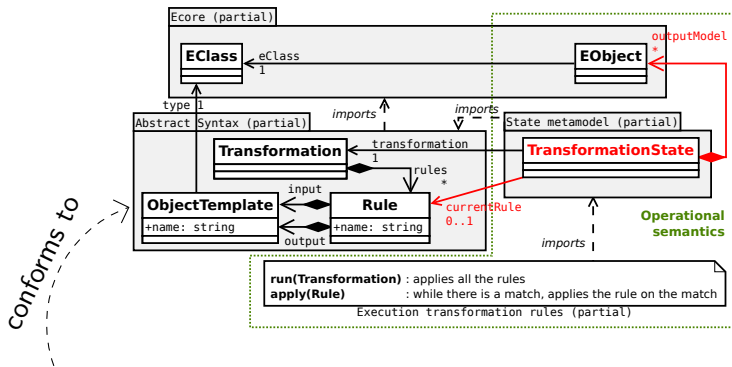
```

transformation simpleAtoB {
  rule AtoB {
    from a : metamodelA.A
    to b : metamodelB.B {
      y = a.x + "_out";
    }
  }
}

```

MiniTL model

Example of MiniTL DSTL and model



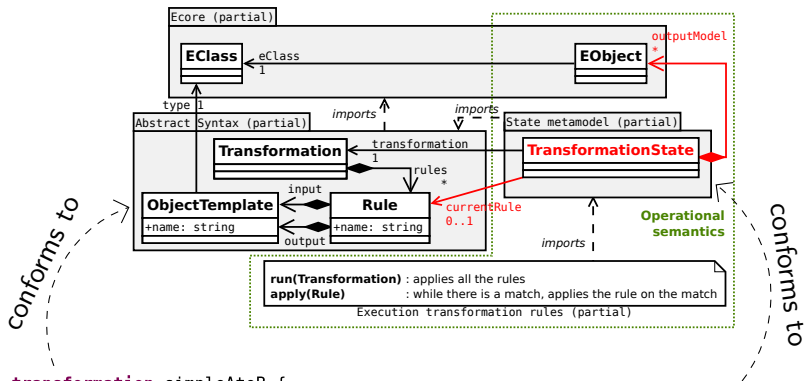
```

transformation simpleAtoB {
  rule AtoB {
    from a : metamodelA.A
    to b : metamodelB.B {
      y = a.x + "_out";
    }
  }
}

```

MiniTL model

Example of MiniTL DSTL and model



```

transformation simpleAtoB {
  rule AToB {
    from a : metamodelA.A
    to b : metamodelB.B {
      y = a.x + "_out";
    }
  }
}

```

MiniTL model

initialization →

```

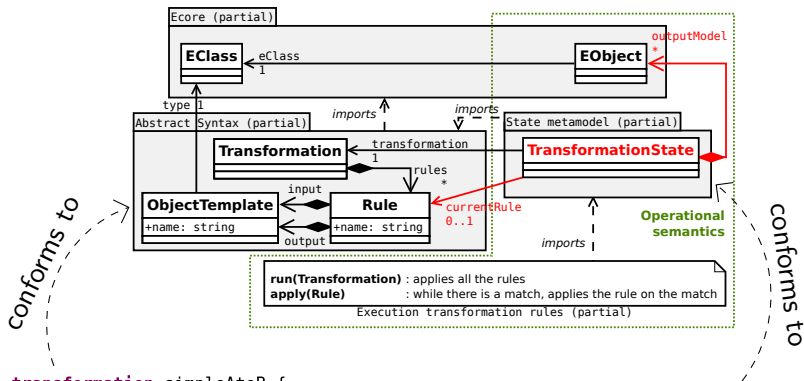
transformation simpleAtoB {
  rule AToB {
    from a : metamodelA.A
    to b : metamodelB.B {
      y = a.x + "_out";
    }
  }
}

```

Executed model



Example of MiniTL DSTL and model



```

transformation simpleAtoB {
  rule AToB {
    from a : metamodelA.A
    to b : metamodelB.B {
      y = a.x + "_out";
    }
  }
}
  
```

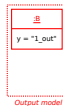
MiniTL model

initialization →

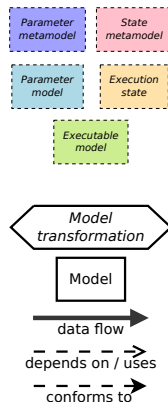
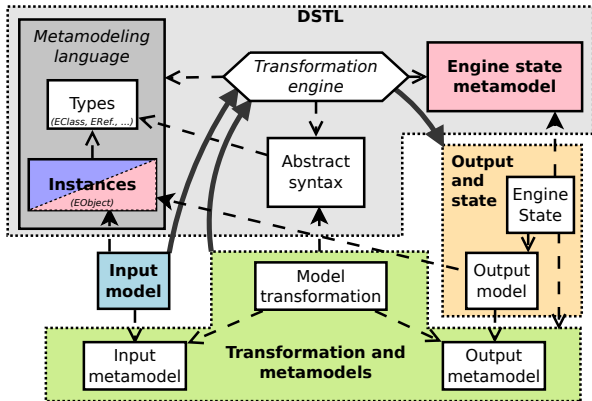
```

transformation simpleAtoB {
  rule AToB {
    from a : metamodelA.A
    to b : metamodelB.B {
      y = a.x + "_out";
    }
  }
}
  
```

Executed model
(AToB app.)



Generalizing DSTLs as specific xDSMLs



Research directions

- Experiment with **generic and generative approaches** for DSTL engineering:
 - Reuse xDSML engineering approaches , *eg.* getting a debugger “for free” for a given DSTL
 - Define/adapt new generic approaches for DSTL engineering
- Evaluate the **implications of DSTL specificities** : *eg.* can we generate a usable/relevant debugger using generic approaches?
- **DSTLs as case studies** for xDSML engineering (cf. TTC’16)

Research directions

- Experiment with **generic and generative approaches** for DSTL engineering:
 - Reuse xDSML engineering approaches , *eg.* getting a debugger “for free” for a given DSTL
 - Define/adapt new generic approaches for DSTL engineering
- Evaluate the **implications of DSTL specificities** : *eg.* can we generate a usable/relevant debugger using generic approaches?
- **DSTLs as case studies** for xDSML engineering (cf. TTC'16)

Research directions

- Experiment with **generic and generative approaches** for DSTL engineering:
 - Reuse xDSML engineering approaches , *eg.* getting a debugger “for free” for a given DSTL
 - Define/adapt new generic approaches for DSTL engineering
- Evaluate the **implications of DSTL specificities** : *eg.* can we generate a usable/relevant debugger using generic approaches?
- **DSTLs as case studies** for xDSML engineering (cf. TTC'16)

Conclusion and future work

- Just an observation: **DSTLs are a sort of xDSMLs**, complex and with interesting characteristics
- Prospects:
 - Use state of the art xDSML engineering for DSTL engineering?
 - Consider DSTLs as nice case studies for model execution?

Future work

- **Short term:** Experiment (more) xDSML engineering on some transformation languages, eg. MiniTL
- **Long term:** analyse a DSTL to automatically provide it with a *white-box testing framework* (test model generation, coverage metrics, fault localization, etc.)

Done!

Thank you! 😊

Implementation of MiniTL example:

<https://github.com/tetrabox/minitl>

Contact:

erwan.bousse@tuwien.ac.at

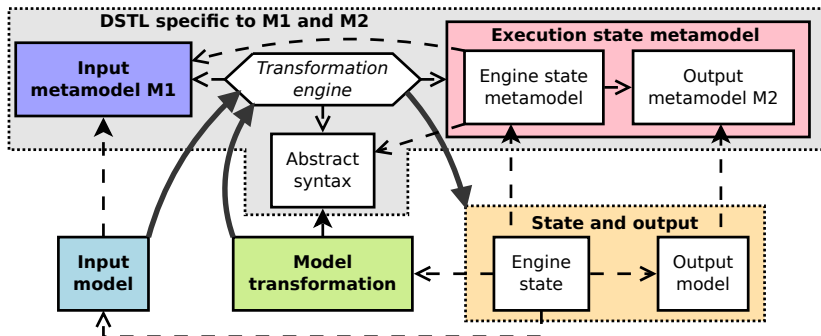
<http://big.tuwien.ac.at/staff/ebousse>

Research project:

TETRA Box: <http://modeltransformation.net/tetrabox/>

we have funding and an open position for a PhD student!

Generalisation of Metamodel-specific DSTLs



Screenshot of MiniTL debugging session

The screenshot displays the Gemoc Studio IDE during a debugging session. The main window is titled "Debug - minitl-models/sample.minitl - Gemoc Studio".

Debug Panel: Shows the execution stack. The current step is "Global context: Transformation". The stack includes:

- run_AtoB [Gemoc Sequential eExecutable Model]
- run_AtoB [Gemoc Sequential eExecutable Model]
- Gemoc debug target
 - Model debugging
 - (ObjectTemplate) simpleAtoB.AToB.b -> construct()
 - (Rule) simpleAtoB.AToB -> apply()
 - (Transformation) simpleAtoB -> execute()

Variables Panel: Lists variables and their values:

Name	Value
currentObject (simpleAtoB.AToB.a :ObjectTemplate)	org.eclipse.emf.ecore.impl.DynamicEObjectImpl@7
currentObject (simpleAtoB.AToB.b :ObjectTemplate)	null
inputModel (simpleAtoB :Transformation)	[org.eclipse.emf.ecore.impl.DynamicEObjectImpl@:
inputModelURI (simpleAtoB :Transformation)	platform:/resource/minitl-models/modelIA.xml
outputFilePath (simpleAtoB :Transformation)	/home/ebousse/dev/runtime-test-minitl-modeling/n
outputModel (simpleAtoB :Transformation)	[org.eclipse.emf.ecore.impl.DynamicEObjectImpl@f

Code Editor: Shows the MiniTL transformation code for `simpleAtoB`:

```

transformation simpleAtoB {
  // Ecore metamodel called "input"
  inputMetamodel metamodelA
  // Ecore metamodel called "output"
  outputMetamodel metamodelB

  // Rule to translate each "A" element into "B"
  rule AtoB {
    // Input template
    from a : metamodelA.A
    // Output template
  }
}

```

Console: Shows the output of the modeling workbench console:

```

Modeling workbench console
About to initialize and run the GEMOC Execution Engine...
Initialization done, starting engine...
Execution finished.
About to initialize and run the GEMOC Execution Engine...
Initialization done, starting engine...

```