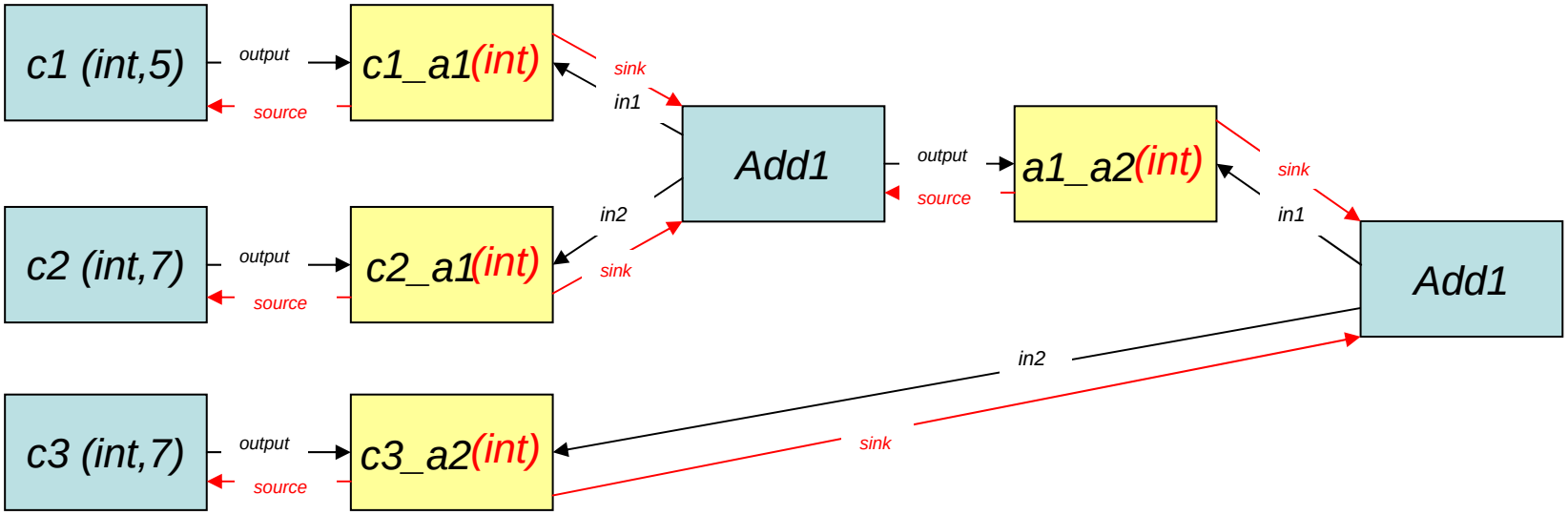
Aktueller Stand - Codegenerator



Durch Modell-zu-Modelltransformation hinzugefügte Information

M2M-Transformation

- Mittels Modell-zu-Modell-Transformationen können Modelle ineinander überführt werden.
- Ziel:
  - Explizites Aufführen von Informationen, die bereits implizit enhalten sind
  - Zusammenführen von verschiedenen Modellen
- M2M im Beispiel:
  - Berechnung der Datentypen für jeden Kanal
  - Zuweisung der Endpunkte jedes Kanals
- openArchitectureWare bietet die Sprache EXTEND zur Definition der M2M-Transformation
  - EXTEND unterstützt Polymorphismus und Aufrufe von Java-Funktionen
  - Auf Attribute und Referenzen mit Kardinalität 1 kann über entsprechende set- und get-Operationen zugegriffen werden

```
Void addActors2Channel(Addition a):
    a.output.setSource(a)->
    a.in1.setSink(a)->
    a.in2.setSink(a);
```

  - Attribute und Referenzen mit Kardinalität >1 werden als Listen behandelt, oAW sucht automatisch die entsprechende Funktion bei Listen als Parameter (entweder Aufruf der Funktion mit Liste als Parameter oder Aufruf für jedes einzelne Element der Liste)
  - Für eine einfache Behandlung der Liste stehen diverse Operationen zur Verfügung (select,typeSelect,remove…)

```
Void transformModel(Application app):
    app.actors.addActors2Channel()->
    app.actors.typeSelect(Constant).propagateType();
```

Codegenerierung

- Die Codegenerierung erfolgt als Textersetzung ähnlich Makroprozessoranweisungen
- oAW bietet mit EXPAND eine entspreche Makrosprache
  - EXTEND ist eine iterative Sprache mit wenigen und einfachen Konzepten (siehe nächste Folie)
  - Polymorphismus wird zur einfachen Codegenerierung unterstützt
  - Genereller Ablauf:
    - **Prinzip**: Alles zwischen den Escapezeichen wird interpretiert, der Rest kopiert
    - Tags werden durch die Escapezeichen « und » (Tastatur **Ctrl+<** und **Ctrl+>**) gekennzeichnet

EXPAND Sprachkonstrukte I:

- DEFINE: Definition einer Generierungsfunktion

```
«DEFINE generateGetFunction FOR Const-»
 ...
«ENDDEFINE»
```

- EXPAND: Aufruf einer Generierungsfunktion

```
«EXPAND generateActorCode FOR a
```

- FILE: Erzeugung einer Datei

```
«FILE "actor_code.c"-»

 ...

«ENDFILE»
```

- FOREACH: Generierung von Code für jedes Objekt einer Liste

```
«FOREACH actors AS a»«EXPAND generateActorVariables FOR a-»«ENDFOREACH-»
«FOREACH actors AS a»«EXPAND generateActorCode FOR a»
```

```
«FOREACH max.channelList AS ch ITERATOR c-»
```

EXPAND Sprachkonstrukte II

- IF / ELSE zur Verzweigung innerhalb der Codegenerierung:

```
«IF type==Type::INT»
        ...
«ELSE»
        ...
«ENDIF»
```

- Auf Attribute kann direkt über den Namen zugegriffen werden:

```
int channel_«source.name»_«sink.name»;
```

- REM: Kommentare

```
«REM»
Abstract class MinMaxActor
«ENDREM»
```

- ERROR: Abbruch der Codegenerierung mit Fehlernachricht:

```
«ERROR "Type of constant "+name+" currently not supported in code generation"»
```

Beschreibung des Arbeitsbereiches

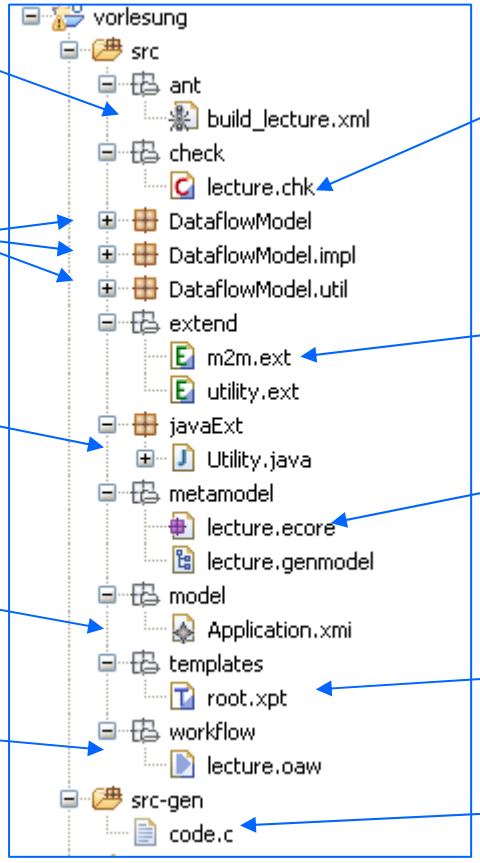
ANT Funktionen zur Automatisierung

Automatisch generierte Java-Klassen

Java-Funktionen für Modell-zu-Modell-Transformation und Codegenerierung

Konkretes Modell

Prozessablaufsbeschreibung



Konsistenzregeln

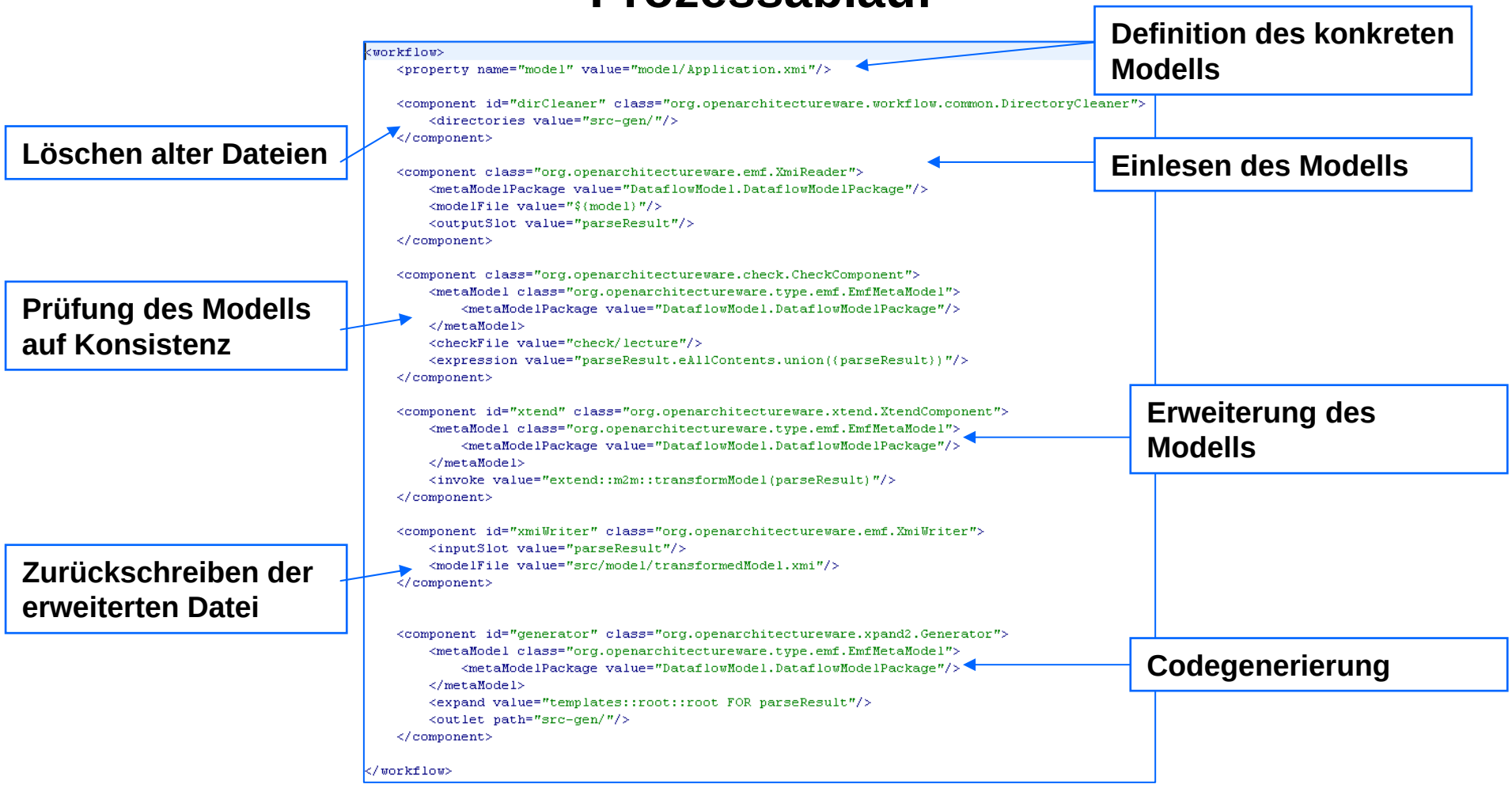
Funktionen zur Modell-zu-Modell-Transformation

Definition des Meta-Modells

Vorlagen zur Codegenerierung

Generierte Dateien

Prozessablauf




```
<workflow>
    <property name="model" value="model/Application.xmi"/>

    <component id="dirCleaner" class="org.openarchitectureware.workflow.common.DirectoryCleaner">
        <directories value="src-gen/"/>
    </component>

    <component class="org.openarchitectureware.emf.XmiReader">
        <metaModelPackage value="DataflowModel.DataflowModelPackage"/>
        <modelFile value="${model}"/>
        <outputSlot value="parseResult"/>
    </component>

    <component class="org.openarchitectureware.check.CheckComponent">
        <metaModel class="org.openarchitectureware.type.emf.EmfMetaModel">
            <metaModelPackage value="DataflowModel.DataflowModelPackage"/>
        </metaModel>
        <checkFile value="check/lecture"/>
        <expression value="parseResult.eAllContents.union({parseResult})"/>
    </component>

    <component id="xtend" class="org.openarchitectureware.xtend.XtendComponent">
        <metaModel class="org.openarchitectureware.type.emf.EmfMetaModel">
            <metaModelPackage value="DataflowModel.DataflowModelPackage"/>
        </metaModel>
        <invoke value="extend::m2m::transformModel(parseResult)"/>
    </component>

    <component id="xmiWriter" class="org.openarchitectureware.emf.XmiWriter">
        <inputSlot value="parseResult"/>
        <modelFile value="src/model/transformedModel.xmi"/>
    </component>


    <component id="generator" class="org.openarchitectureware.xpand2.Generator">
        <metaModel class="org.openarchitectureware.type.emf.EmfMetaModel">
            <metaModelPackage value="DataflowModel.DataflowModelPackage"/>
        </metaModel>
        <expand value="templates::root::root FOR parseResult"/>
        <outlet path="src-gen/"/>
    </component>

</workflow>
```