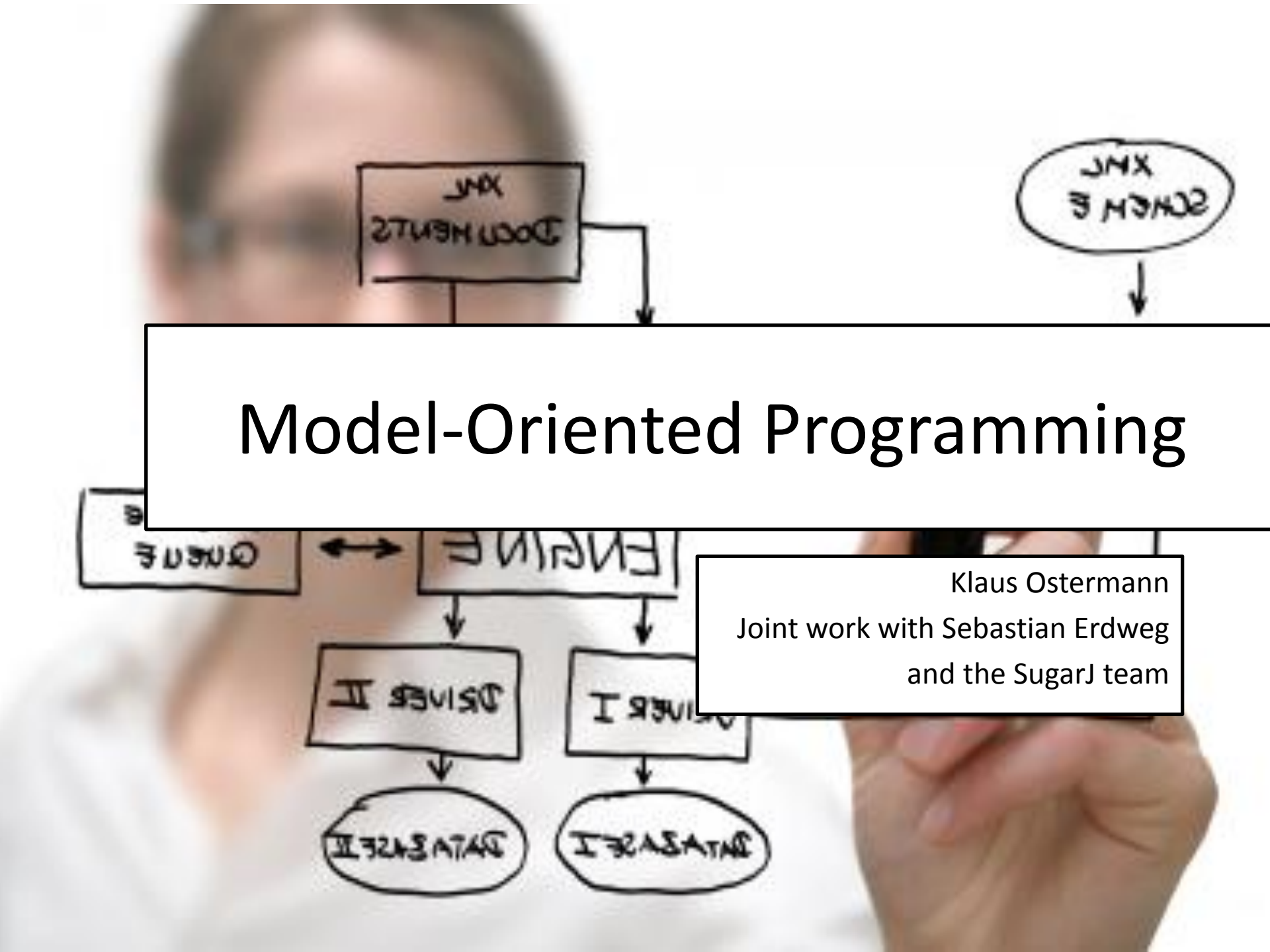
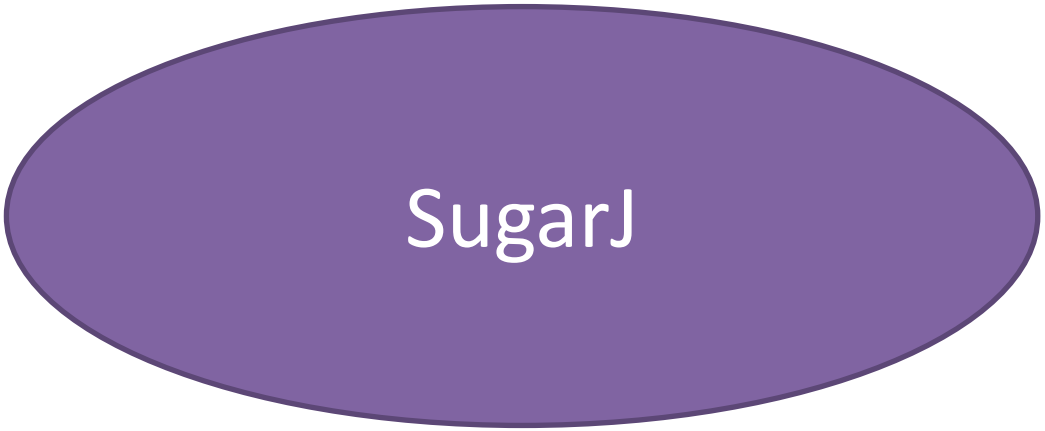


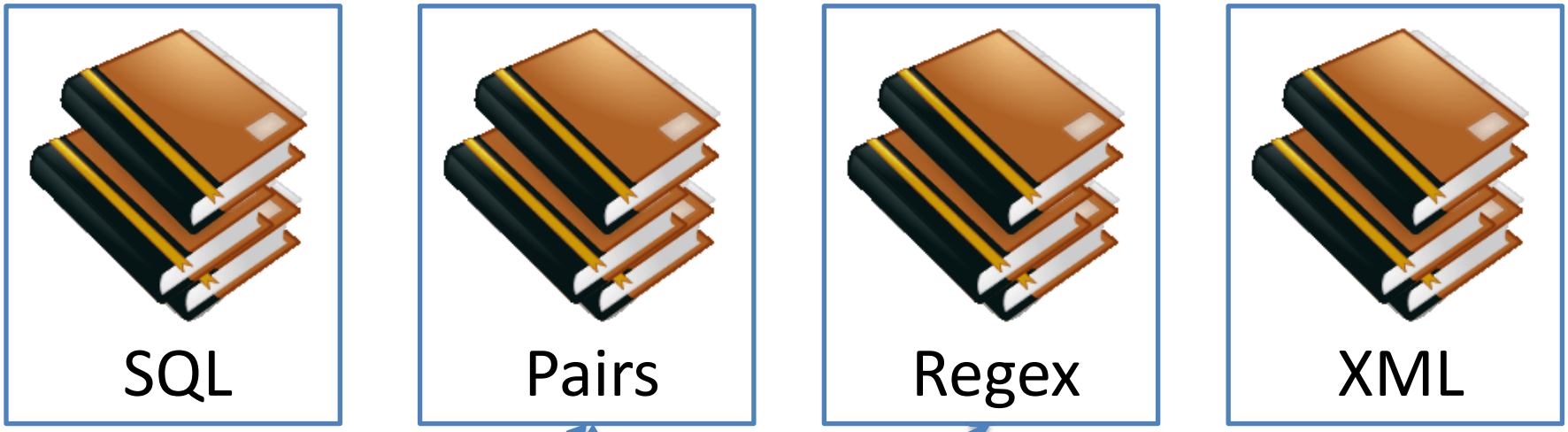
Model-Oriented Programming

Klaus Ostermann
Joint work with Sebastian Erdweg
and the SugarJ team

Goals

- domain-specific syntax
- language composition
- extensible language-definition mechanism






```
import Pair
import Reg

public clas
  private (St
    ("/Users/
}
```

```
import Pairs;

public class Test {
  private (String, Integer) p = ("12", 34);
}
```

Example: XML serialization

in Java using SAX

```
public void appendBook(ContentHandler ch) {
  String title = "Sweetness and Power";
  ch.startDocument();
  AttributesImpl bookAttrs = new AttributesImpl();
  bookAttrs.addAttribute("", "title", "title", "CDATA", title);
  ch.startElement("", "book", "book", bookAttrs);
  AttributesImpl authorAttrs = new AttributesImpl();
  authorAttrs.addAttribute("", "name", "name", "CDATA", "Sidney W. Mintz”);
  ch.startElement("", "author", "author", authorAttrs);
  ch.endElement("", "author", "author");
  ch.startElement("", "editions", "editions", new AttributesImpl());
  AttributesImpl edition1Attrs = new AttributesImpl();
  edition1Attrs.addAttribute("", "year", "year", "CDATA", "1985");
  edition1Attrs.addAttribute("", "publisher", "publisher", "CDATA", "Viking");
  ch.startElement("", "edition", "edition", edition1Attrs);
  ch.endElement("", "edition", "edition");
  ch.endElement("", "editions", "editions");
  ch.endElement("", "book", "book");
  ch.endDocument();
}
```

XML in SugarJ

```
import XML;

public void appendBook(ContentHandler ch) {
  String title = "Sweetness and Power";

  ch.<book title="{title}">
       <author name="Sidney W. Mintz" />
       <editions>
         <edition year="1985" publisher="Viking Press" />
         <edition year="1986" publisher="Penguin Books" />
       </editions>
     </book>;
}
```

```
public sugar Pairs {

  context-free syntax
    "(" JavaExpr "," JavaExpr ")" -> JavaExpr

  rules
    pair-desugaring
      |[ (~e1, ~e2) ]|

  desugarings
    pair-desugaring
}
```

SDF

```
import Pairs;

public class Test {
  private (String, Integer) p = ("12", 34);
}
```

Stratego

```
private (String, Integer) p = ("12", 34);
```

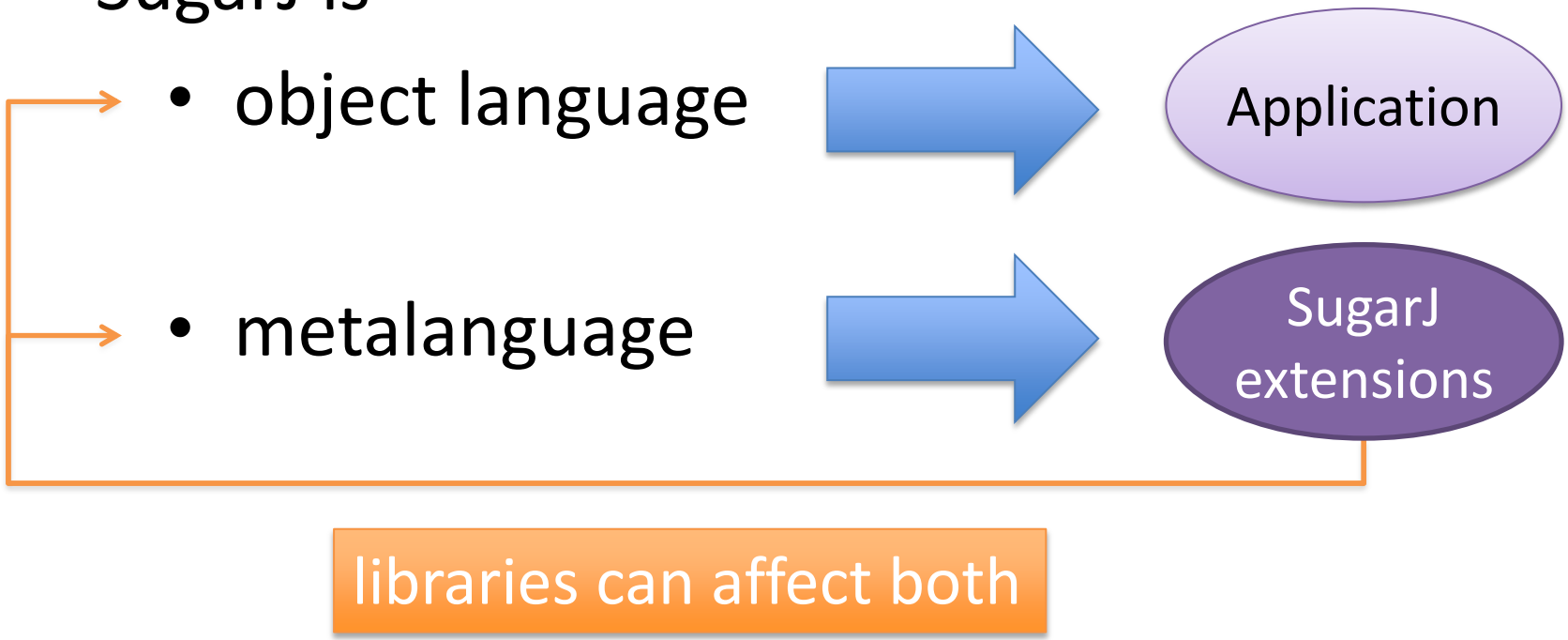
Desugar

```
private Pair<String, Integer> p = new Pair("12", 34);
```

Metalevels and SugarJ

SugarJ is

- object language
- metalanguage



libraries can affect both

Metalanguage extensions

- alternative syntax for grammars
- concrete syntax for transformations
- me

```
rules
  pair-desugaring :
    |[ (~e1, ~e2) ]| -> |[ new Pair(~e1, ~e2) ]|
```

```
prog   :  stat+ ;
stat   :      expr NEWLIN
       |      ID '=' expr
       |      NEWLINE                ;
```

desugar

```
rules
  pair-desugaring :
    PExpr(e1, e2) ->
    NewInstance(
      None(),
      ClassOrInterfaceType(TypeName(Id("Pair")), None()),
      [e1, e2],
      None())
```

Metalanguage Extensibility: XML Schema

```
<xsd:schema targetNamespace="lib">
  <xsd:element name="book" type="B
  <xsd:complexType name="Book">
    <xsd:choice maxOccurs="unbound
      <xsd:element name="author" t
      <xsd:element name="editions"
    </xsd:choice>
    <xsd:attribute name="title" ty
  </xsd:complexType>
</xsd:schema>
```


```
import BookSchema;

ch.<book title="{title}">
     <author name="Sidney W. Mintz" />
     <editions>
       <edition year="1985" publisher="Viking Press" />
       <edit    year="1986" publisher="Penguin Books" />
     </editions>
   </book>;
```

!

!

MODEL-ORIENTED PROGRAMMING

Macros

```
(module m
  (provides t)
  (define-syntax t …))
```

```
(requires m)
…(t a)…
```

MDD

Model a = …

Transformation t = …

---

generate b as t(a) — Build script

… b … — Code using b

Modular but no “model reuse”

SugarJ is in this camp

Independent models but
not modular, CWA, no separate comp.

Undesirable stratification into meta-levels

Model-Oriented Programming

- Extends SugarJ
- First-class notion of models
- All dependencies explicit through imports
  - Separate compilation
- First-class notion of transformation application through imports
- Can quote/unquote every software artifact

- Combines the strengths of macros and models

Example: A state machine DSL

```
package banking;

import statemachine.Language;

public statemachine ATM {
  initial state Init

  events DoWithdraw, Cancel, PinOK, PinNOK, [...]

  state Init {
    DoWithdraw => Withdraw
    Cancel => Init
  }
  state Withdraw {
    PinOK => GiveMoney
    PinNOK => RevokeCard
    Cancel => Init
  }
  state GiveMoney { MoneyTaken => ReturnCard }
  state ReturnCard { CardTaken => Init }
  state RevokeCard { CardRevoked => Init }
}
```

Using the state machine

```
package banking;

import banking.ATM<statemachine.ToJava>;

public class ATMTest {
  public void test() {
    ATM machine = new ATM();
    machine.step(machine.event_DoWithdraw());
    machine.step(machine.event_PinOK());
    machine.step(machine.event_MoneyTaken());
    machine.step(machine.event_CardTaken());
    assert machine.currentState() == machine.state_Init();
  }
}
```

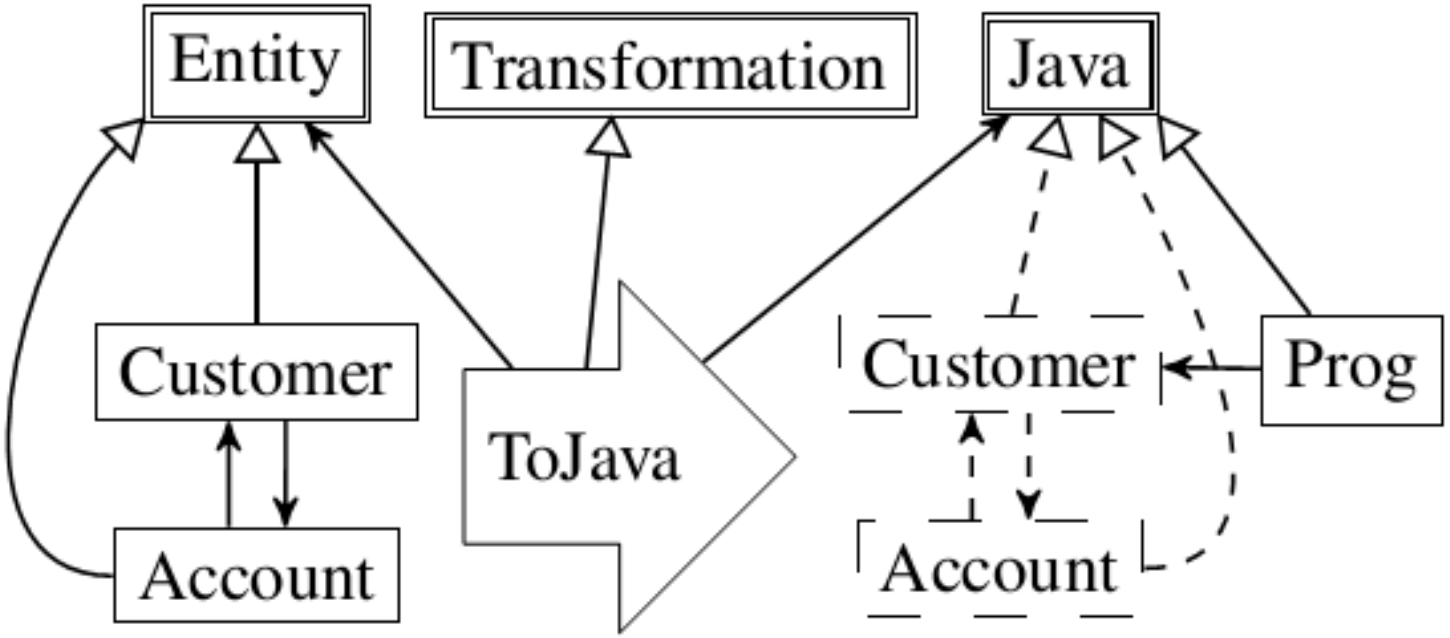
Using the same model twice

```
package banking;

import banking.ATM<statemachine.ToJava>;
import banking.ATM<statemachine.ToJavaIO> as ATM_IO;
import java.io.*;

public class ATMSimu {
  public static void main(String[] args) {
    ATM machine = new ATM();
    [...] // initialize reader
    while ((in = reader.readLine()) != null) {
      ATM.Event e = ATM_IO.parseEvent(machine, in);
      machine.step(e);
      System.out.println(machine.currentState());
    }
  }
}
```

```
package banking.entity;

import entity.Language;
model import banking.entity.Customer;

public entity Account {
  uid :: Integer
  owner :: Customer
  balance :: Integer
  pin :: String
}
```

```
package banking.entity;

import entity.Language;
model import banking.entity.Account;

public entity Customer {
  name :: String
  address :: String
  accounts :: Set<Account>
}
```

Decomposing models with model imports



A model transformation applies to the transitive closure of the model imports

```
package entity;

import template.ConcreteSyntax;
import template.Unquote;
import template.Shorthands;
import template.LocalStrategies;
import template.IdentifierComposition;
import template.Foreach;

import entity.Language;
import org.sugarj.languages.Java;
```

```
public template ToJava {
  package $pkg;
  $*{?CompilationUnit(_,<id>,_)}
  public class $classname {

    $for(Property(name, sort) in collect-all(?Property(_,_
      $type = <sort-to-javatype> sort
      $upperName = <ld(first-upper)> name

      private $type $name;
      public $type get#$uppername() {
        return $name;
      }
      public void set#$uppername($type $name) {
        this.$name = $name;
      }
    }
  }
```

Modeling on the meta-level: Assembling features of a template language

Formalization [Work in progress]

Syntax:

$$
\begin{array}{rcl}
n & \in & \text{Identifier} \\
m & ::= & (n, (i, \ldots, i), e) \\
i & ::= & n \mid i \langle i \rangle \mid i! \\
e & ::= & \ldots
\end{array}
$$

Semantic domains:

$$
\begin{array}{rcl}
\mathbb{D} & = & \mathbb{B}ase + \mathbb{T}rans + \mathbb{M}odel \\
\mathbb{B}ase & = & \ldots \\
\mathbb{M}odel & = & (\mathbb{D} \times \cdots \times \mathbb{D}) \times e \\
\mathbb{T}rans & = & \mathbb{M}odel \to \mathbb{D} \\
\mathbb{E}nv & = & \text{Identifier} \to \mathbb{D}
\end{array}
$$

$$
\begin{array}{lcl}
[\![\cdot]\!]_i & : & i \to \mathbb{E}nv \to \mathbb{D} \\
[\![n]\!]_i\ \rho & = & \rho\ n \\
[\![i_1\ \langle i_2 \rangle]\!]_i\ \rho & = & \begin{cases} d_2\ d_1, & \text{if} \quad d_1 = ([\![i_1]\!]_i\ \rho) \in \mathbb{M}odel \text{ and} \\ & & \qquad d_2 = ([\![i_2]\!]_i\ \rho) \in \mathbb{T}rans \\ \bot, & & \text{otherwise} \end{cases} \\
[\![i!]\!]_i\ \rho & = & reify\ ([\![i]\!]_i\ \rho)
\end{array}
$$

Conclusions

- We try to combine the strengths of macros and MDD
  - Macros: Modularity, Referential Integrity
  - MDD: Separate models and transformations
- All dependencies are explicit
- Everything is (or can be) a model
- All concepts applicable across meta-levels