A Framework for Rapid Development of Model Compilers

Paul Boocock

The Jamda Project

Introduction

- Jamda is:
  - A solid basis for model compiler development
  - A basis for a library of reusable modules
  - An ongoing open source project
- The name:
  - Java Model Driven Architecture

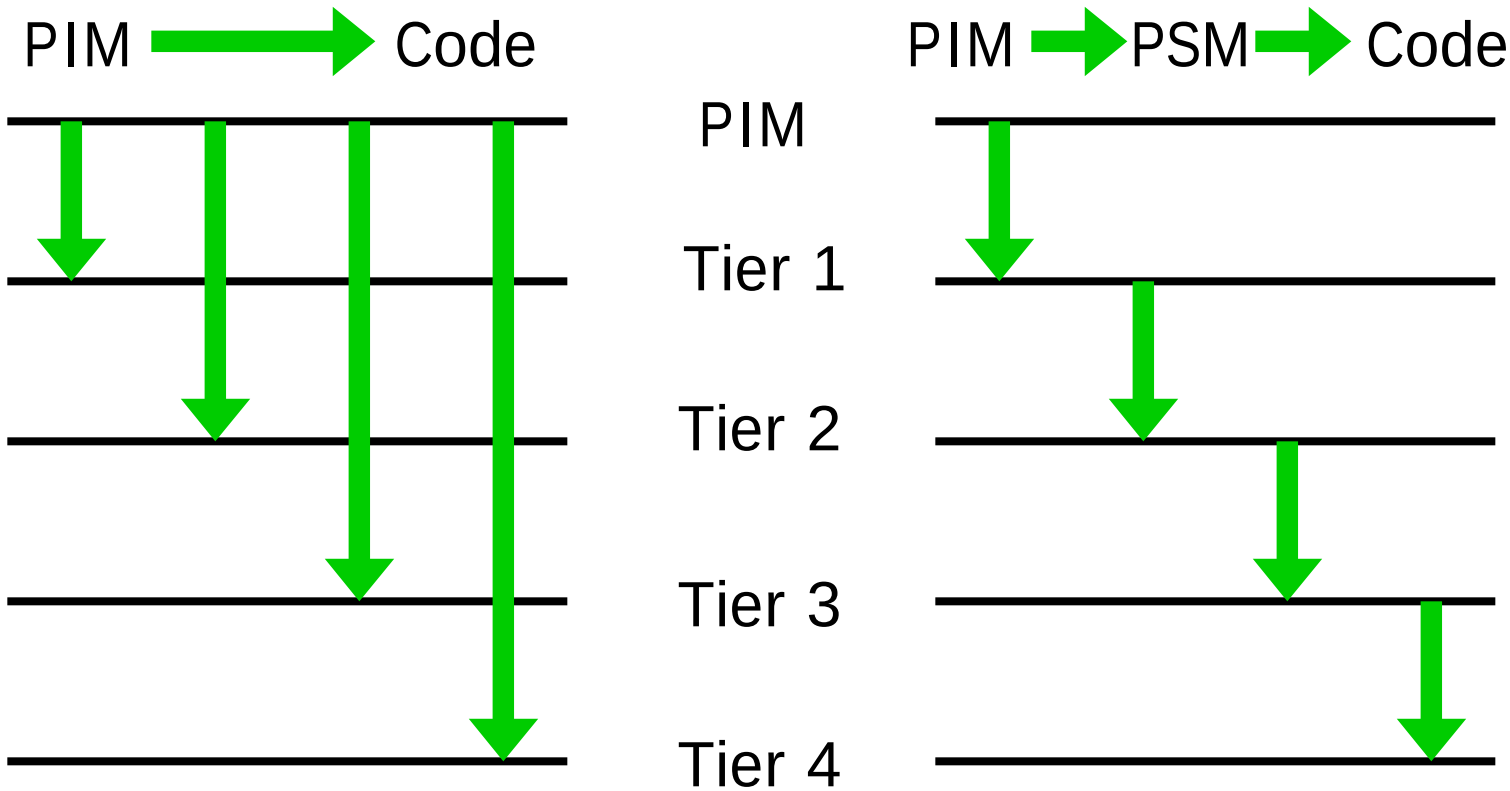
Contents of this presentation

- Problems that Jamda aims to solve
- Approach and principles
- What it provides
- How it is used
- Examples
- Future directions

Model Compiler problems

- Most projects cannot use a pre-defined architecture
- Unclear what variable parameters are needed
- Unfamiliar, less powerful definition languages
- Reuse difficult
- Cost
- Direct PIM to code transformations

The Transformation Gap



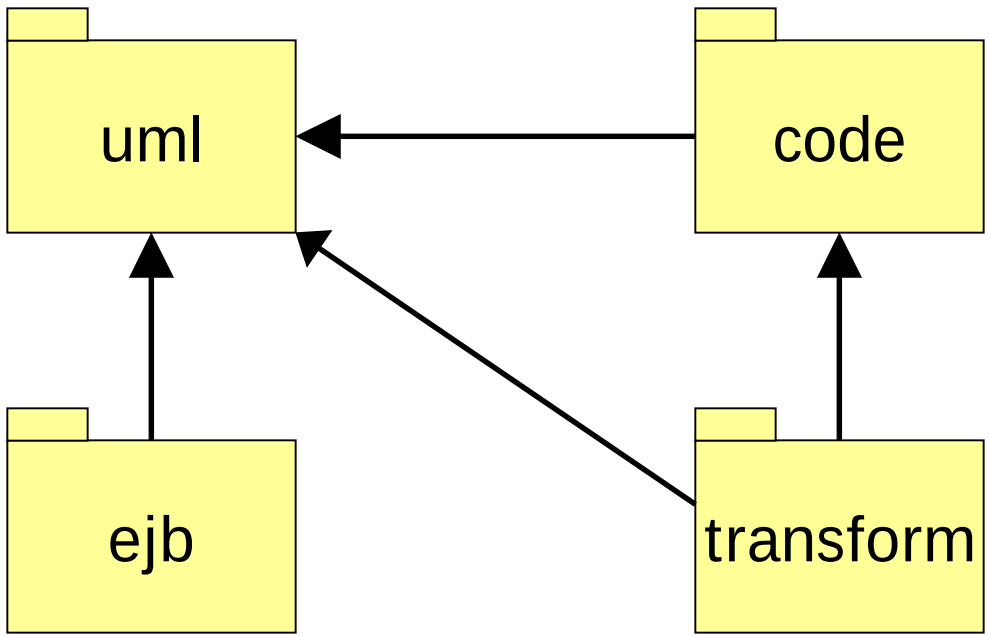
The Jamda Approach

- Foundation and powerful tools
- Program transformations in Java
- Enhance the metamodel elements
- Manipulate metamodel elements as long as possible
- Combine cumulative transformations
- Provide a plugin framework

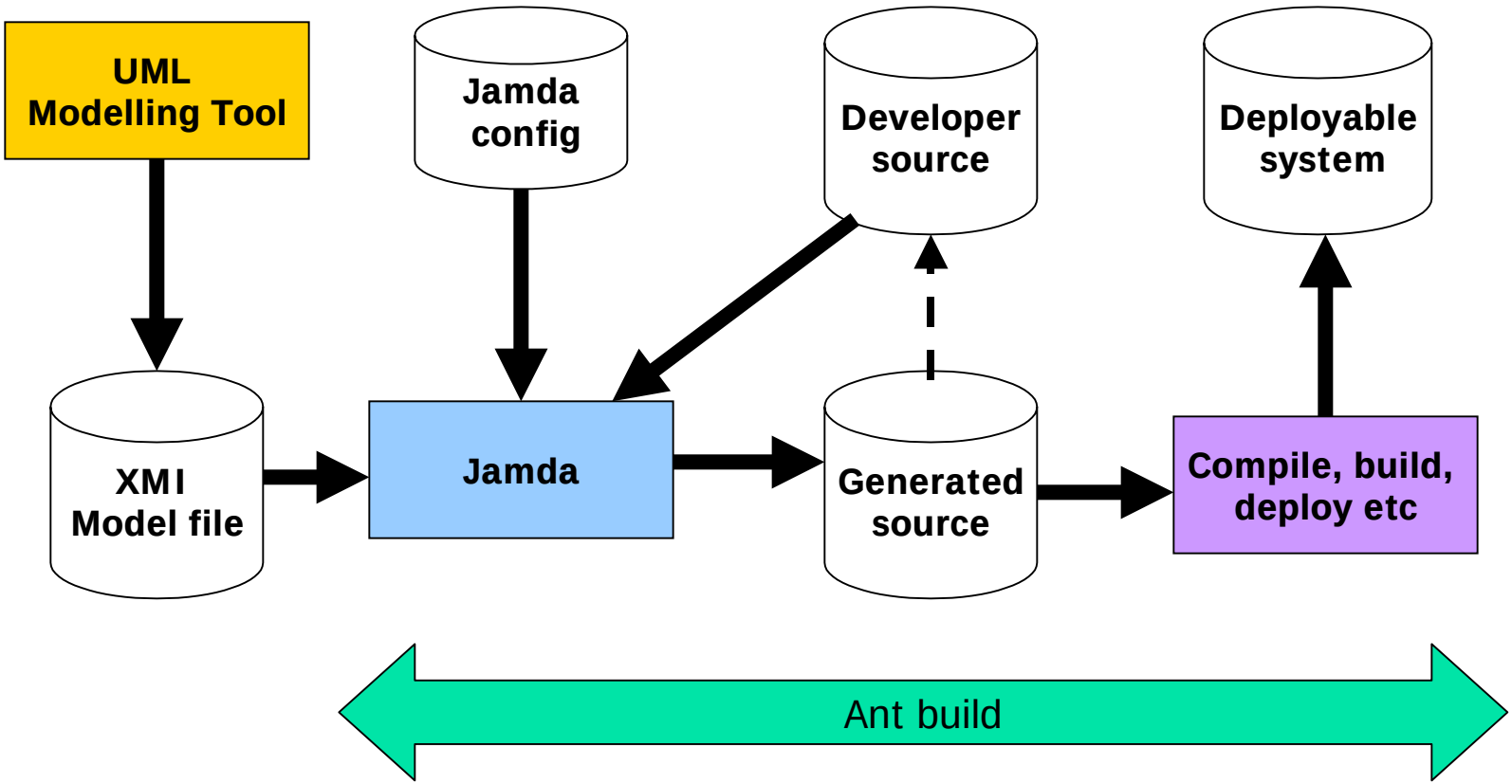
What Jamda provides

- Friendlier, richer and simpler metamodel API
- User-defined metamodel classes
- User-defined metamodel properties
- Configurable Transformer mechanism
- Method code extraction and reinsertion

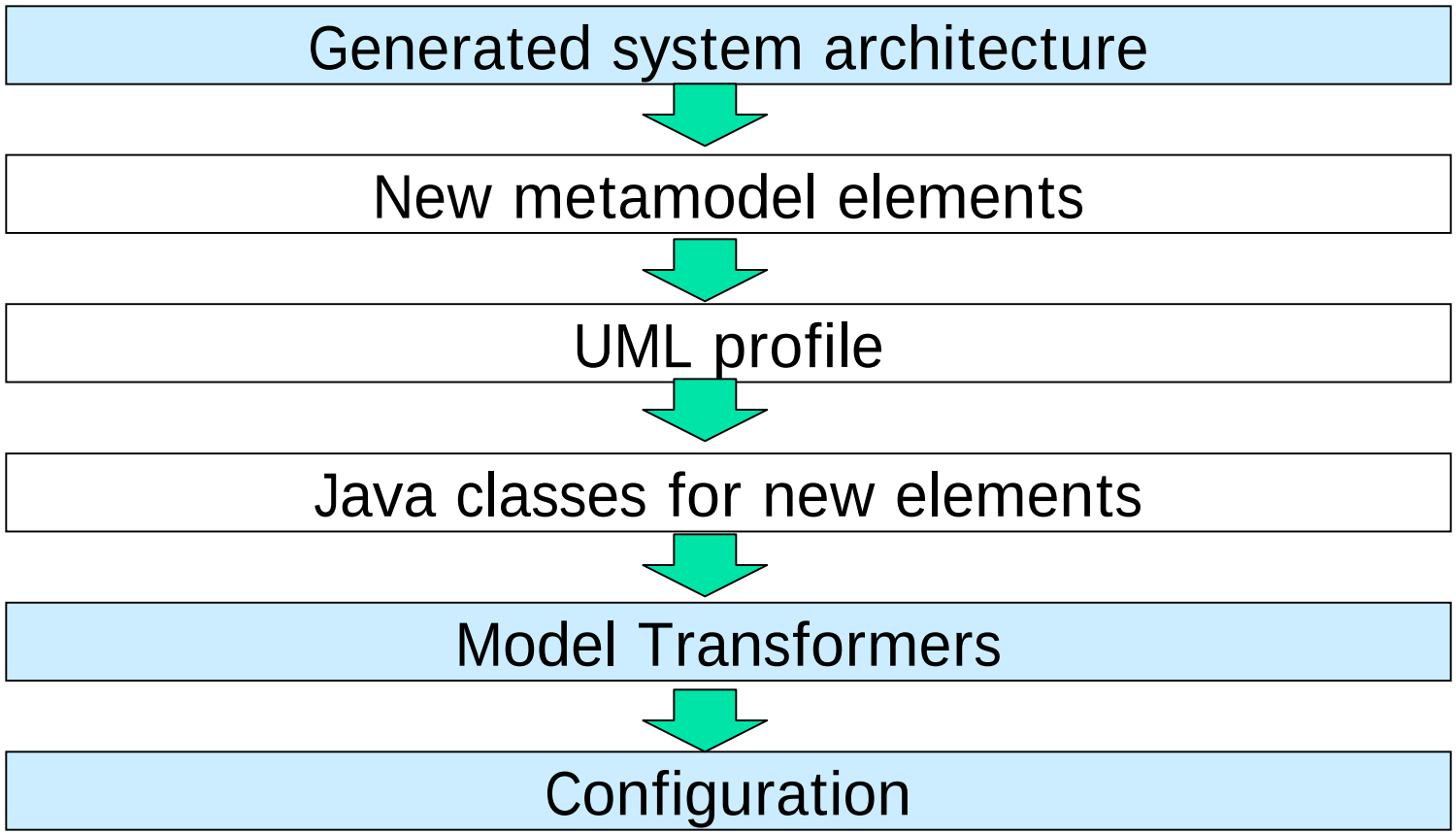
Jamda Packages



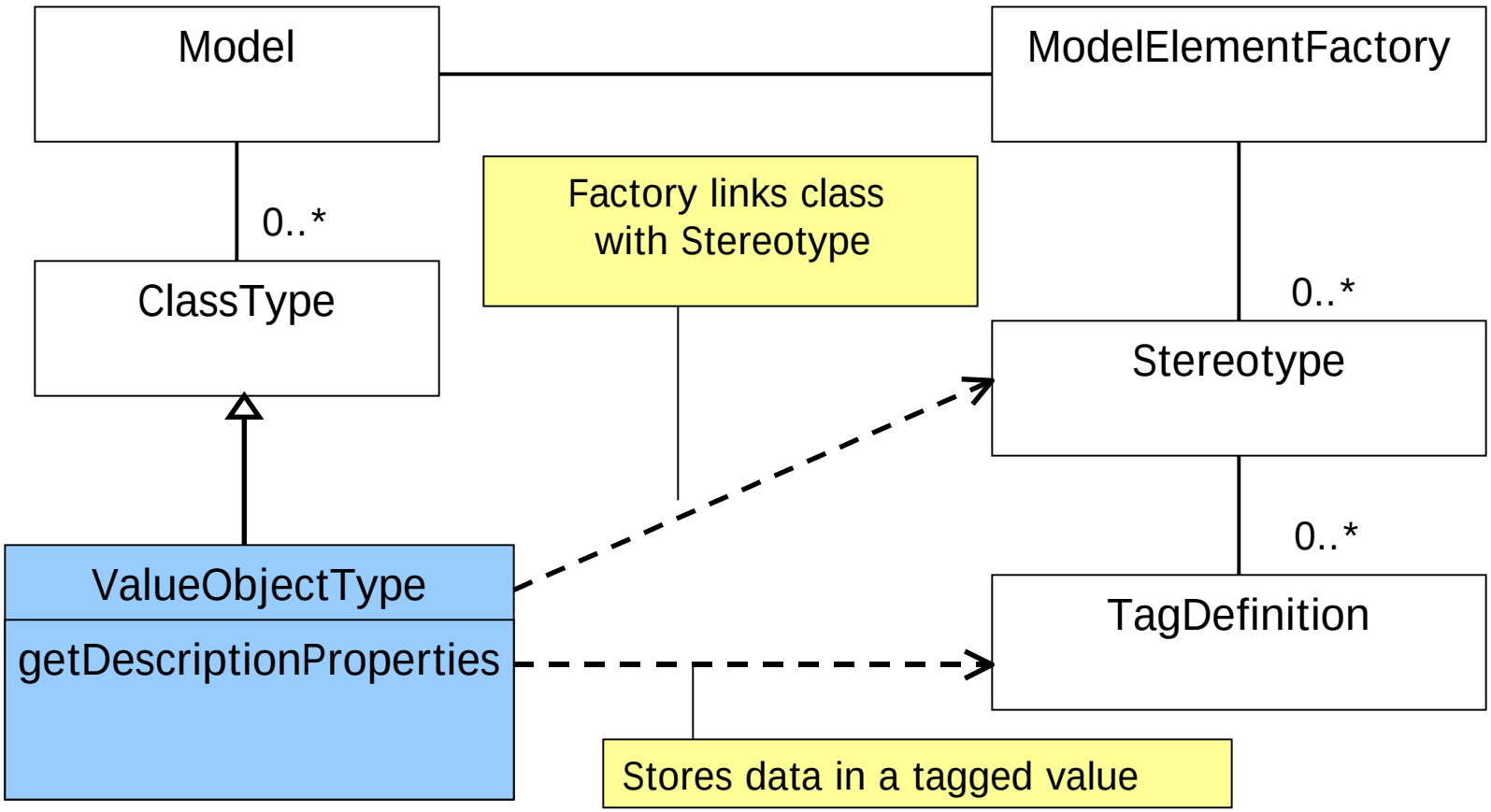
Using Jamda in development



Creating a Model Compiler



User-defined metamodel elements



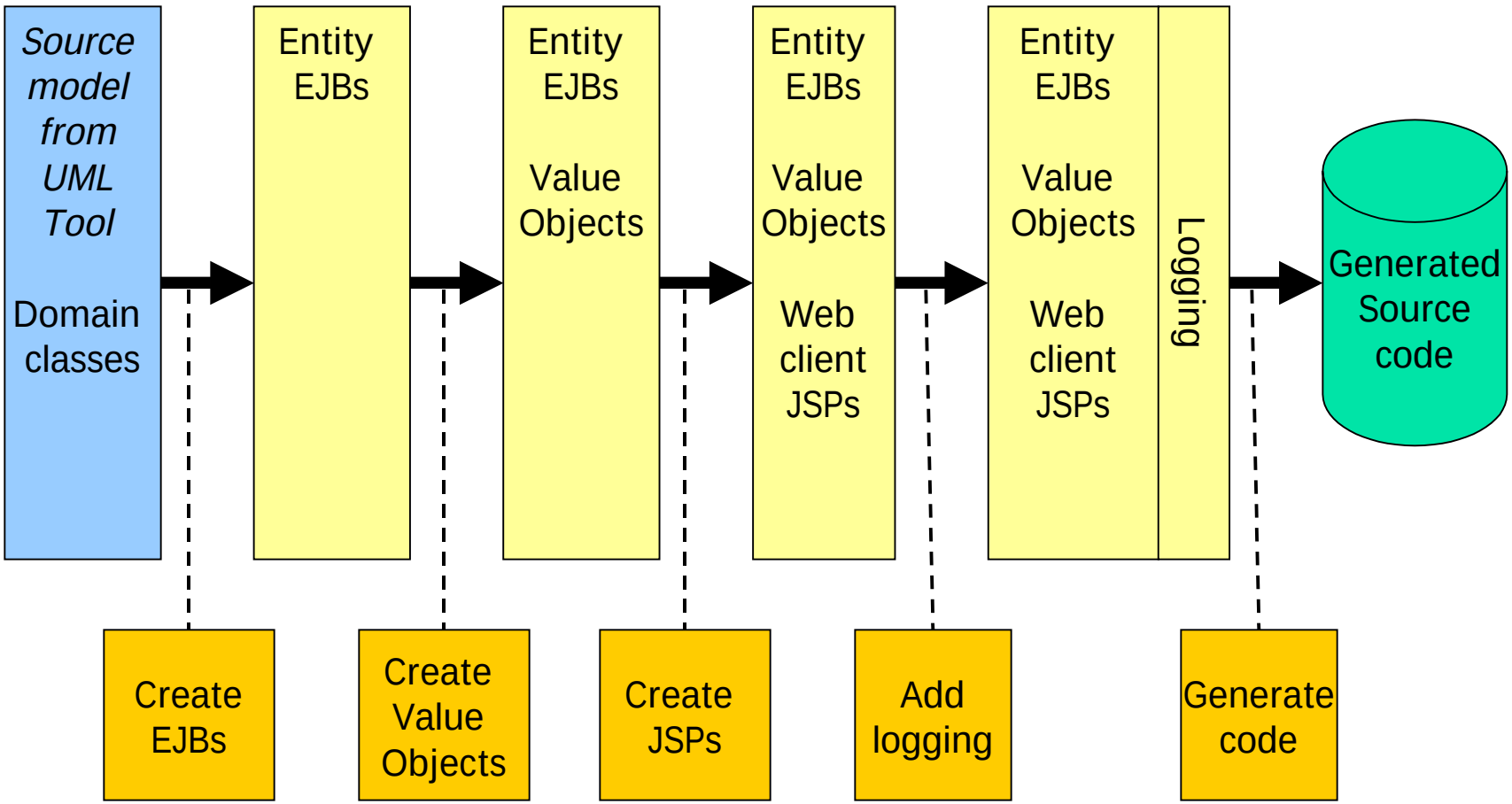
Hello World Transformer

```
public class HelloWorld extends ModelTransformer {

void transform( Model model ) {
  ModelPackage pkg
      = new ModelPackage( model, "hello");
  ClassType cls = new ClassType( pkg, "Hello" );
  Operation op = new Operation( cls, "sayHello" );
  new Parameter( op, type(UmlTypes.STRING), "whoTo");
  op.setStatic( true );
  op.setProcedure( Language.JAVA,
      "System.out.println( \"Hello \" + whoTo );" );
  op.setComment( "Why are we doing this?" );
}
}
```

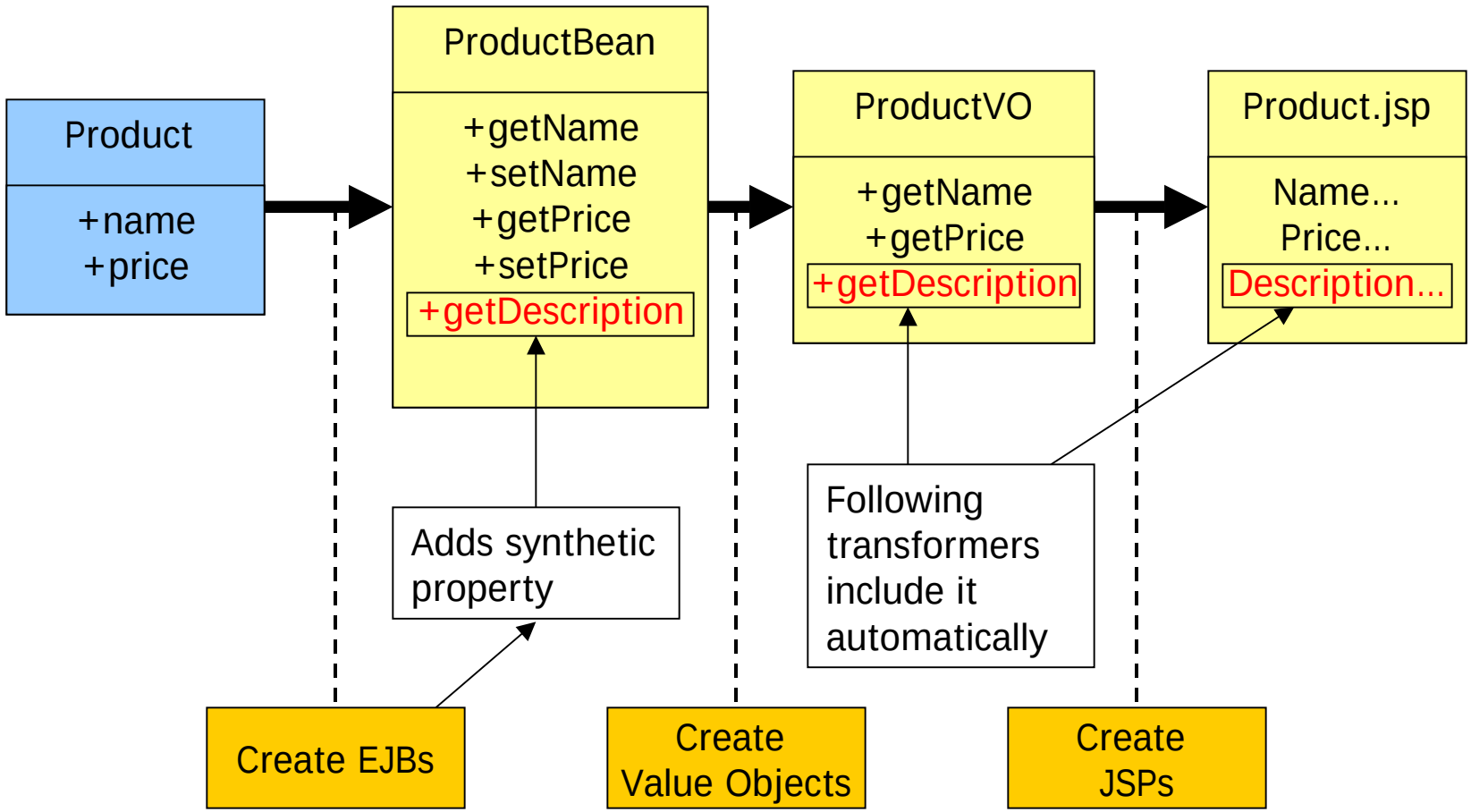
Hello World generated code

```
package hello;

public class Hello {

  /**
   * Why are we doing this?
   */
  public static void sayHello( String whoTo ) {
    System.out.println( "Hello " + whoTo );
  }
}
```

Chaining Transformers together



Changes ripple through



Logging Aspect Element Transformer

```
public class LoggingAdder extends ElementTransformer {

void transform( Element element ) {
  ClassType cls = (ClassType) element;
  List ops = cls.getOperations();
  for ( Iterator i = ops.iterator(); i.hasNext(); ) {
    Operation op = (Operation) i.next();
    String msg = Util.quote( "In " + op.getName() );
    op.setBeforeProcedure(
      "System.out.println( " + msg + " );" );
  }
}
}
```

Future directions

- Continue development based on experience
- Closer integration with modelling tools
- Establish a transformer library
- Action language compiler

Transformer library outlook

- Specific purpose transformers
  - Entity EJBs
  - Session façade EJBs
  - Database definition
  - JSPs for data display
- General purpose transformers
  - Logging
  - Assertions from OCL
  - XML/Java conversion
  - toString, hashCode, equals

The Jamda Project

- Good: flexible, powerful, reusable tools
- Bad: you have to think!
- www.jamda.net
- Apache style licence
- Please use, criticise and contribute!