Model Transformations and Code Generation

Ecole IN2P3 Temps Réel

Chokri.Mraidha@cea.fr

Agenda

- **Introduction on Model Transformations and Code Generation ~ 30 mins**
- **Practical Work ~ 90 mins**
- **Results Discussion ~ 15 mins**

Model Transformations and Code Generation

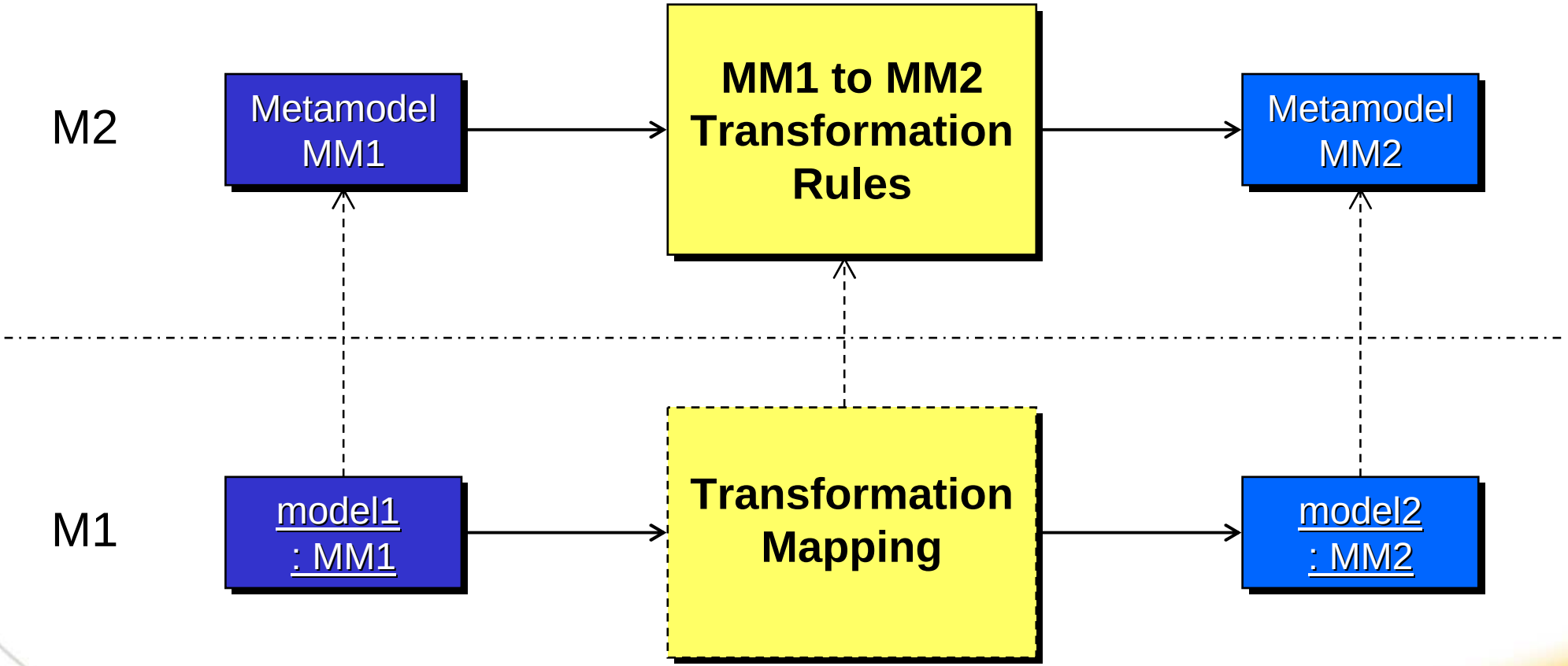
Model Transformations: Purpose

- **Generating a new model from a source model—according to formally defined rules—to:**
  - Extract an interesting subset of the source model (Query)
    - Example: Find all classes that have multiple parents
  - Generate a new model, based on a different metamodel, that is "equivalent" to the source model (Transformation)
    - Example: Create a queueing network model of a UML model to facilitate performance analysis
    - Example: UML to Java transformation for code generation
    - Definition of "equivalence" depends on the purpose
  - Represent the source model from a particular viewpoint (View)
    - In effect, just a special case of Transformation

Courtesy of Bran Selic – Malina Software Corp.

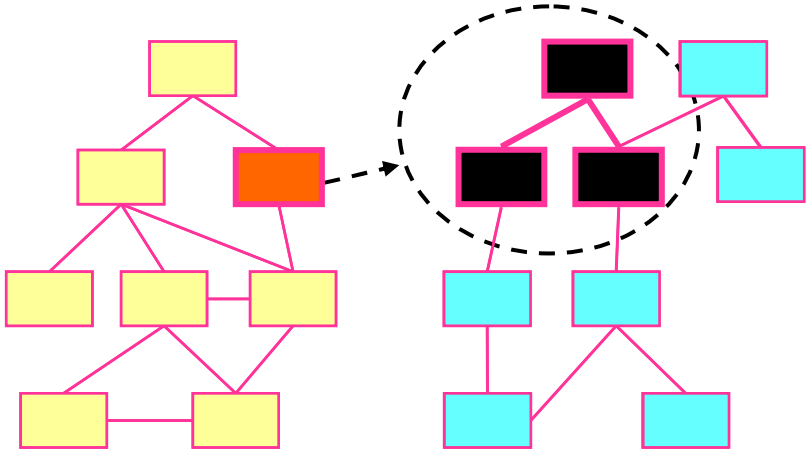
A Basic Representation of Model Transformation

- **Source to target mapping based on pre-defined transformation rules**
  - Conceptually similar to source code compilation

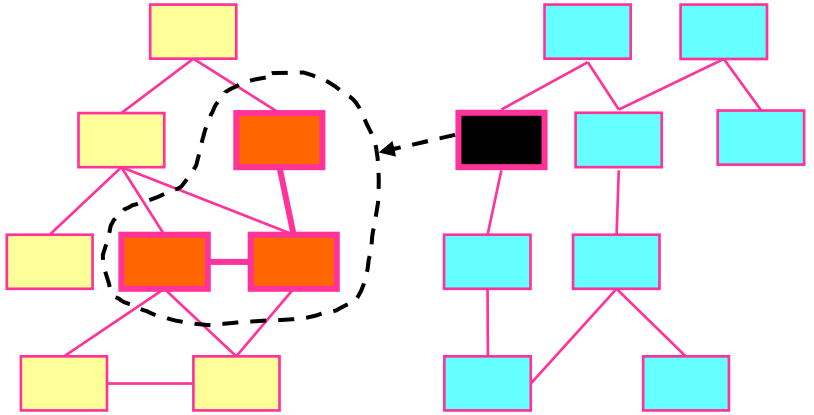


Courtesy of Bran Selic – Malina Software Corp.

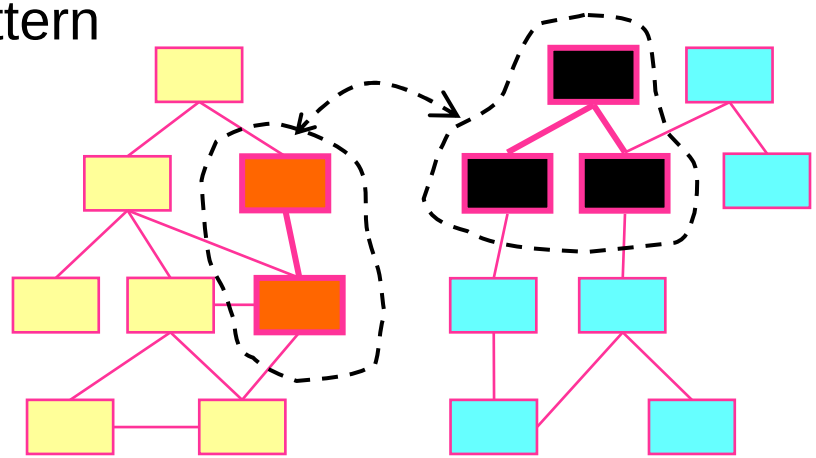
Model Transformations: Styles



(1) Source element to target pattern



(2) Source pattern to target element



(3) Source pattern to target pattern

Courtesy of Bran Selic – Malina Software Corp.

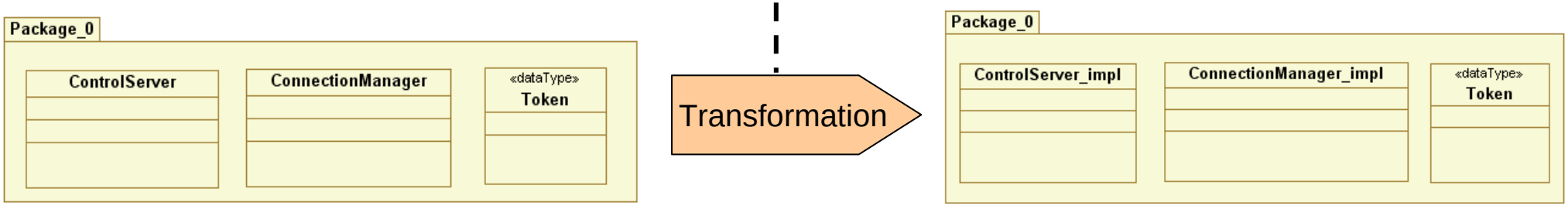
M2M Transformations

- **Examples**
  - Model refinement
  - Model abstraction
  - View generation
  - Design pattern application

- **Some languages**
  - XSLT
  - MOF 2 Queries / Views / Transformations (QVT)
    - ATL (QVT like) (interpretation)
    - Smart-QVT (generation)
  - Java

M2M transformation with Java

```
Iterator<PackageableElement> iter = selectedPackage.getPackagedElements().iterator();
while(iter.hasNext()) {
    PackageableElement currentElement = iter.next();
    if(currentElement instanceof Class) {
        currentElement.setName(currentElement.getName()+"_impl");
    }
}
```



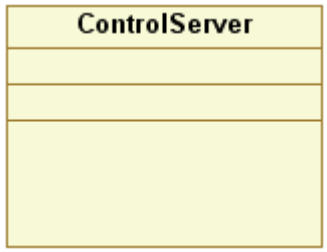
M2T Transformations

- **Examples**
  - Code generation
  - Documentation generation
  - Input format of specialized tools (e.g. analysis tools)

- **Some languages**
  - MOF to Text (standard)
    - No existing full implementation
  - Template-based implementations (Eclipse M2T project)
    - Acceleo
    - Xpand
    - JET
  - Java

M2T transformation using Acceleo

```
<%
metamodel http://www.eclipse.org/uml2/2.0.0/UML
%>

<%script type="uml.Class" name="GenerateClass" file="<%fullFilePath%>"%>

<%visibility%> class <%name%> {

        public <%name%> ( ){ };

}
```



Generation

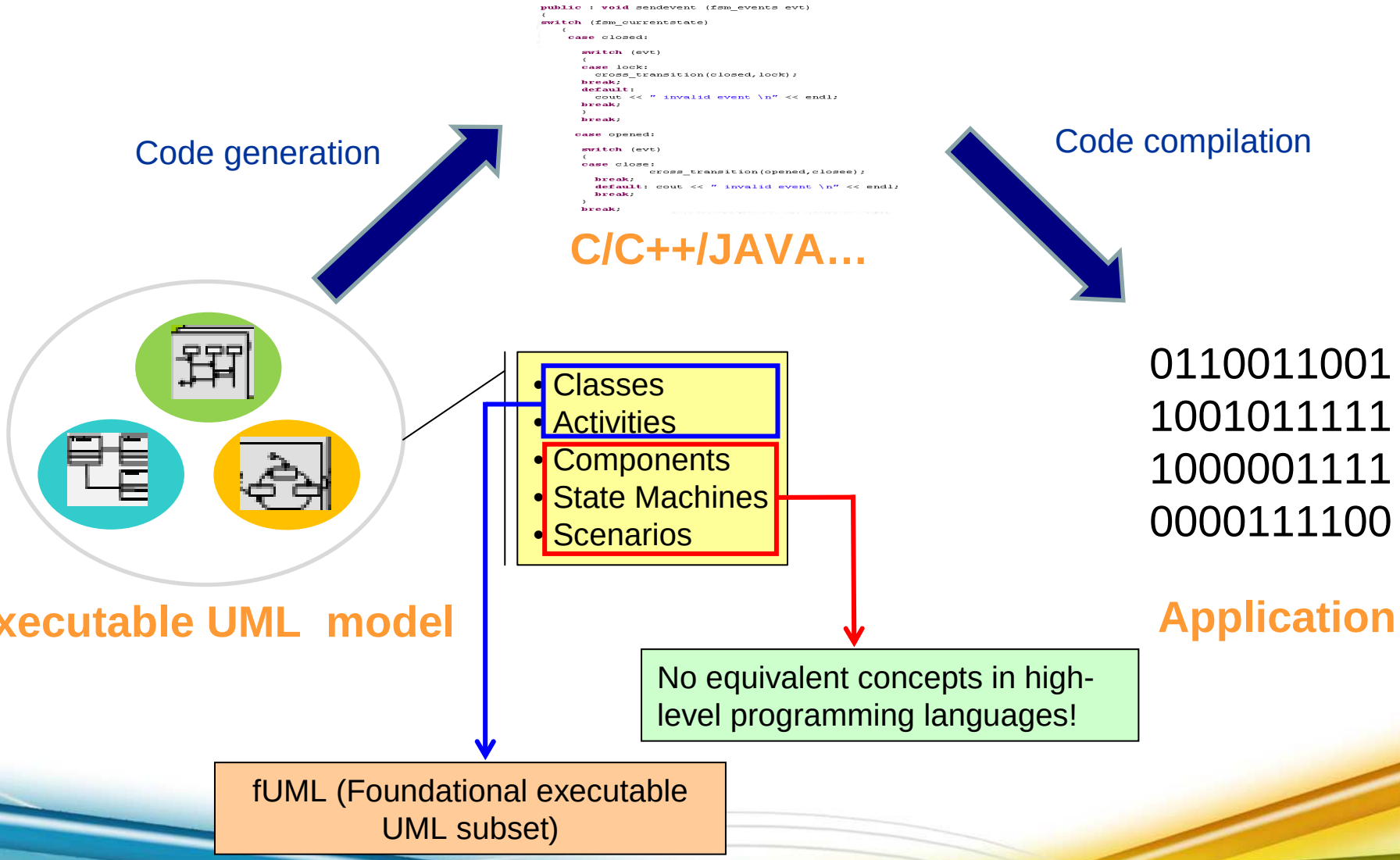
```
public class ControlServer {

        public ControlServer () { };

}
```

Summary

- **Model transformation is a key element of model-based development**
  - Similar to compiling but has a larger scope
  - Convert models to equivalent models for several purposes
    - Model-based analysis
    - Application synthesis (code generation)
    - Documentation generation
    - Model refactoring
    - Model refinement
    - …
- **Several languages and styles**
  - M2M, M2T
  - Declarative or imperative transformation languages
- **A lot of work remains to deal with**
  - Scalability issues
  - Performance issues
  - Optimization issues
  - Traceability issues

Practical Work

From UML models to binary application

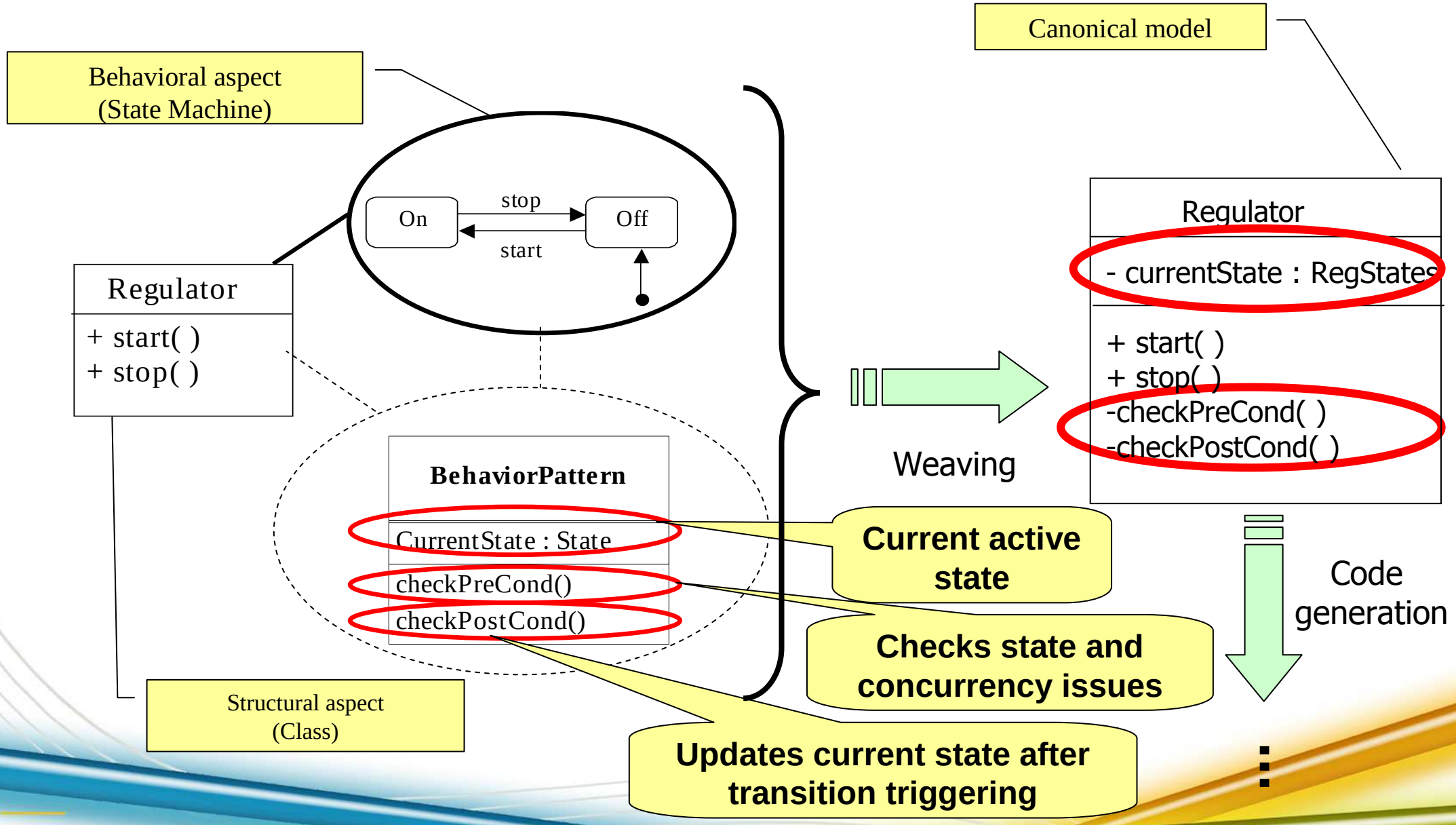


Two alternatives for code generation

- **Implicit mapping: embed the code generation rules in the code generator**
  - Pros:
    - Perform a M2T transformation only
  - Cons:
    - Complex code generator, hard to maintain
    - Cannot easily use a new mapping (pattern)
    - Difficulties to debug the model (indirect mapping between model concept and code concepts)

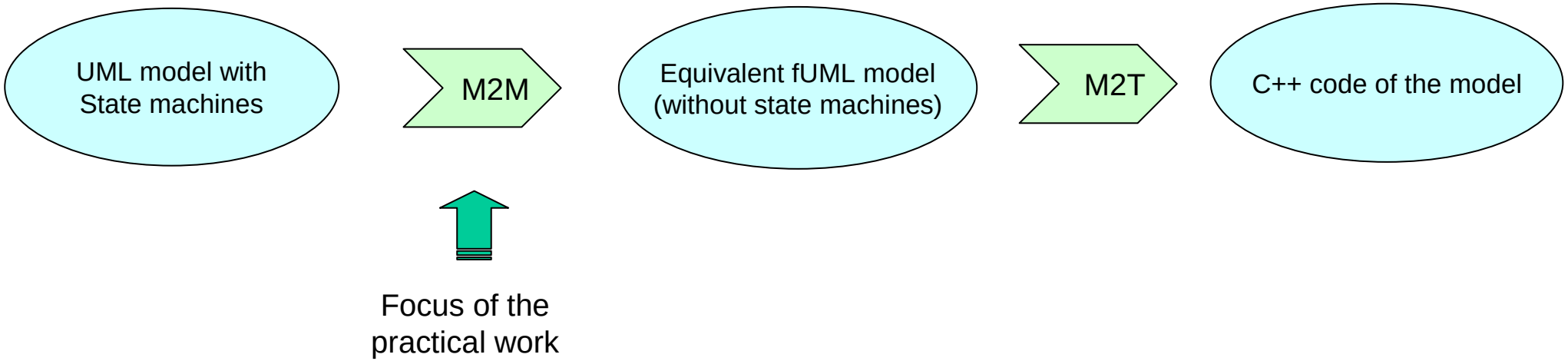
- **Explicit mapping: built a "canonical" model that contains only concepts provided by the targeted high-level programming language (fUML subset)**
  - Pros:
    - Separation of concerns (UML → fUML → code)
    - Simple code generator (one to one mapping, easy to maintain)
    - Facilitates debug of the canonical model
  - Cons:
    - Multiple model transformations (M2M + M2T)

Pattern for state machines code generation

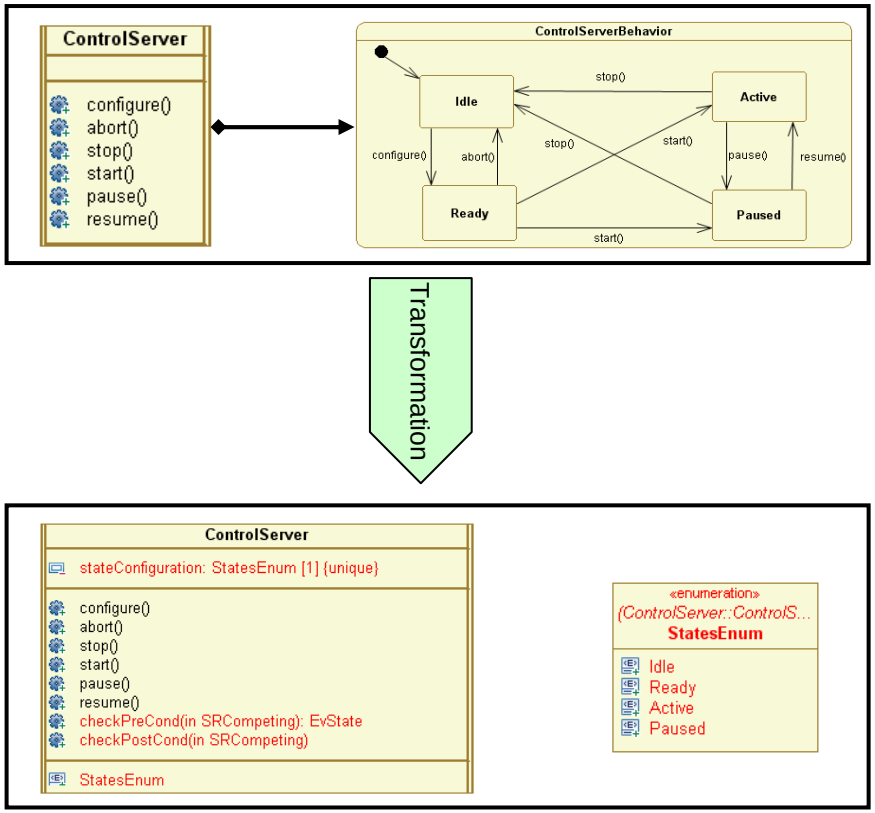


Work description

- **Objective:**
  - Write the M2M transformation in order to generate the canonical model containing concepts of OO programming languages only.
  - Apply a given design pattern for state machines.



Pattern Description



- **Implement the following steps:**
  - Add an Enumeration type named “StatesEnum” containing an enumeration literal for each state of the state machine.
  - Add a property named “stateConfiguration” typed by the previously created Enumeration.
  - Add a checkPreCond() operation.
  - Add a checkPostCond() operation.