# Orccad, a Model Driven Architecture and Environment for Robot Control

May 18th & 19th 2010, Douai

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

INRIA

centre de recherche GRENOBLE - RHÔNE-ALPES

CAR' 2010

Soraya Arias
Florine Boudin
Roger Pissard-Gibollet
Daniel Simon

## Orccad : status and motivations

**Model:**

- Control design oriented approach for robotics
- Mixed feedback and discrete events

**Tools:**

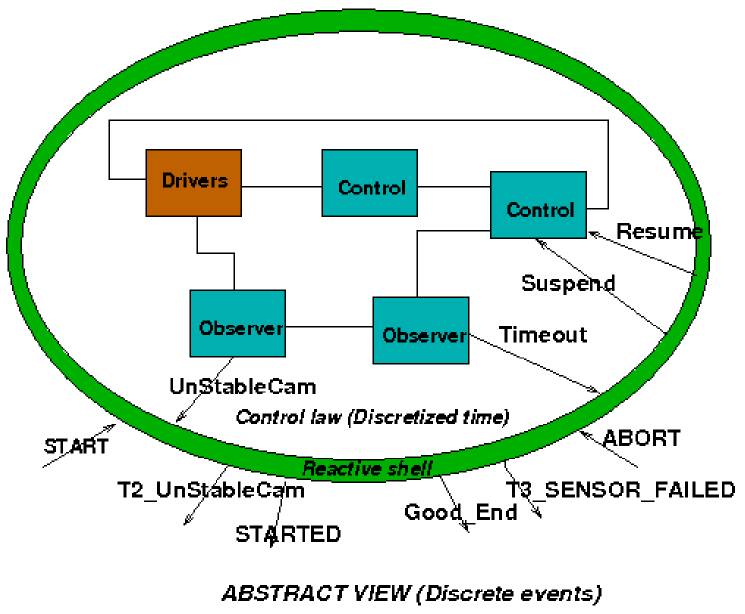
- Design & simulation/validation
- Real-time workshop

**V4 modeling and software development:**

- Aging version, based on proprietary tools
- Sound model & design approach
- Model Driven Architecture based on Eclipse Modeling Tools
- Open Source software

## The Orccad model

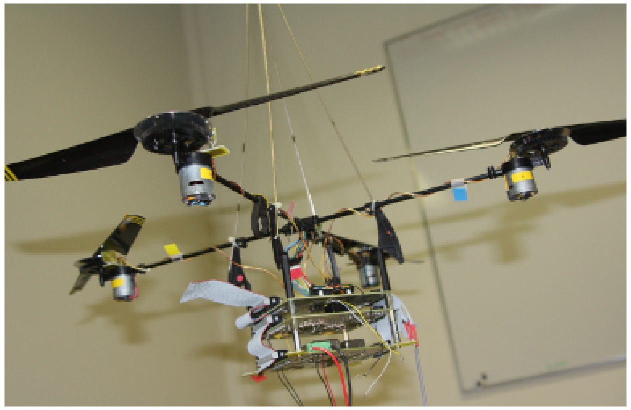
**Top-down requirements capture**
**Bottom-up design**

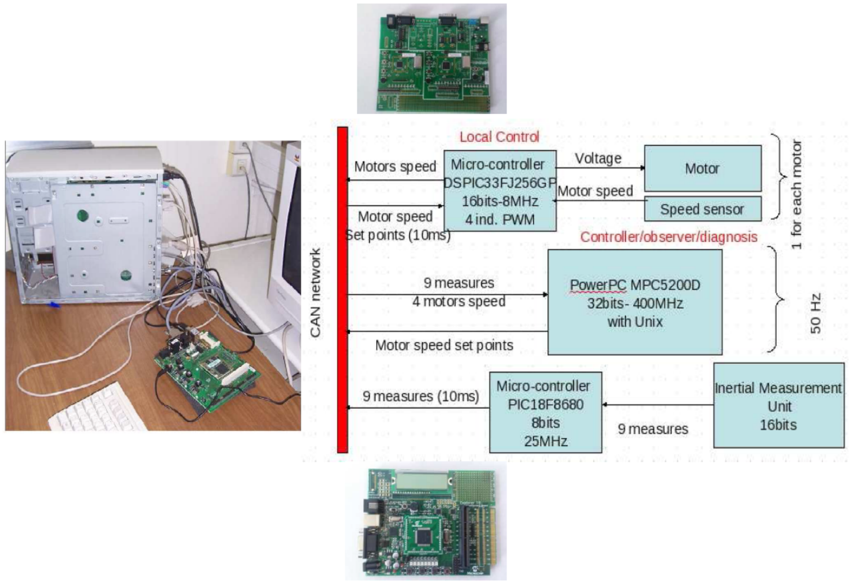


**RobotTasks**
- Feedback Control
- Cyclic real-time data flow
- Event-based view

**RobotProcedures**
- Discrete Events Control
- Incremental design
- Exception processing
- Mission definition

## Quadrotor networked control & diagnosis




Local Control
Motors speed
Micro-controller DSPIC33FJ256GP 16bits-8MHz 4 ind. PWM
Voltage
Motor
Motor speed
Speed sensor
Motor speed Set points (10ms)
1 for each motor
Controller/observer/diagnosis
CAN network
9 measures 4 motors speed
PowerPC MPC5200D 32bits- 400MHz with Unix
Motor speed set points
50 Hz
9 measures (10ms)
Micro-controller PIC18F8680 8bits 25MHz
9 measures
Inertial Measurement Unit 16bits

**Networked system**
- CAN bus
- Distributed diagnosis
- Fault tolerant control

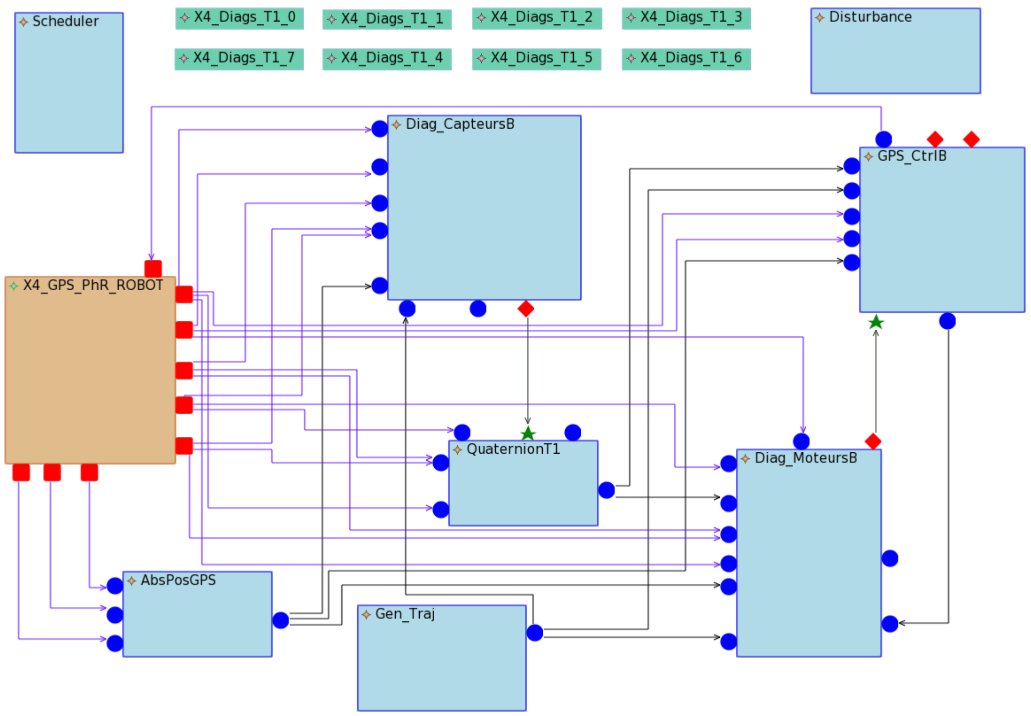
**Flexible scheduling**
- Varying sampling
- (m,k)-firm
- Dynamic priorities

**Hardware-in-the-loop**
- Linux simulation
- PPC embedded

**V4 Runtime update**

**(SafeNecs ANR)**

## Drone control block-diagram



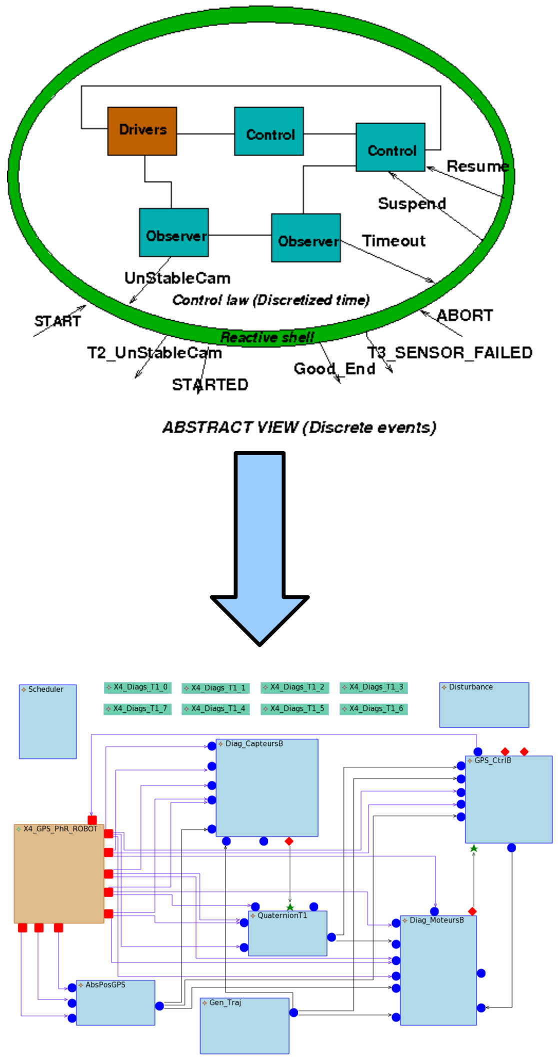
**Networked system**
- CAN bus
- Distributed diagnosis
- Fault tolerant control

**Flexible scheduling**
- Varying sampling
- (m,k)-firm
- Dynamic priorities

**Hardware-in-the-loop**
- Linux simulation
- PPC embedded

**V4 Runtime update**

# Orccad components: RobotTask



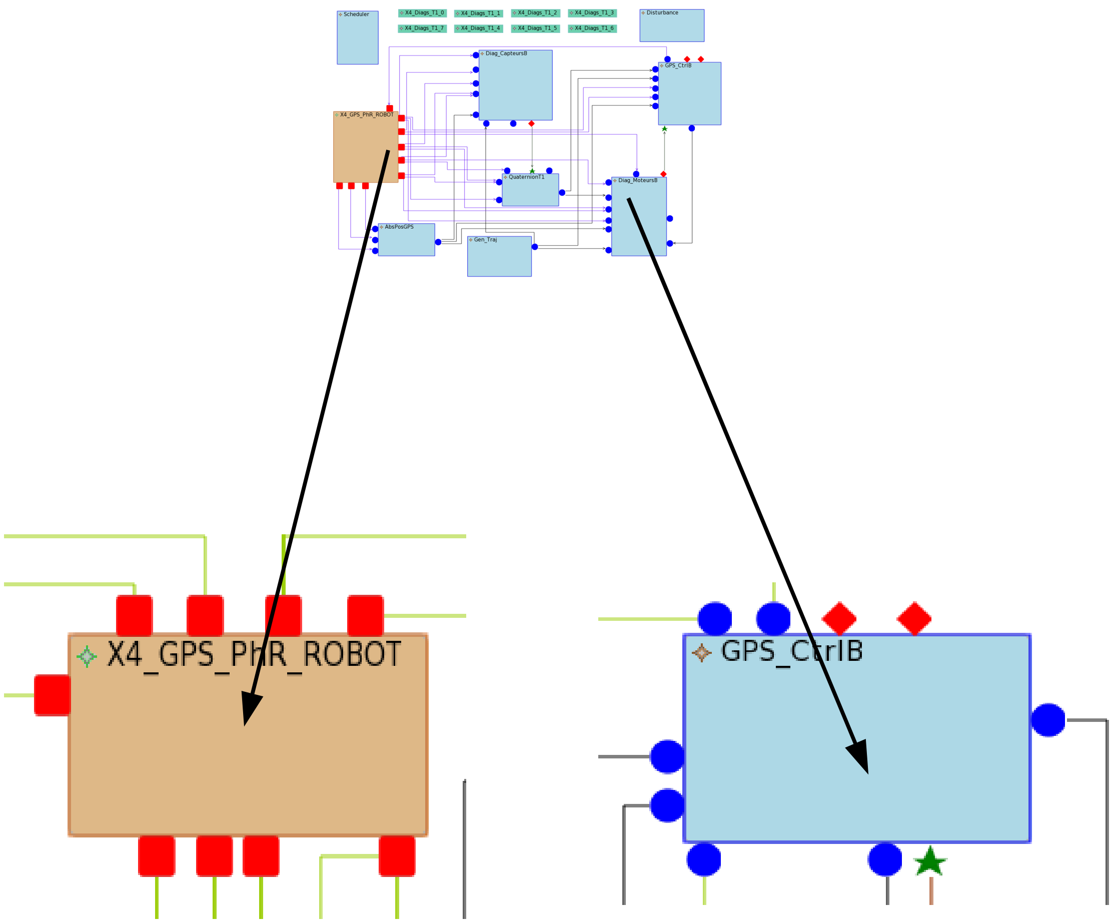
**Feedback control action**

- Control algorithm definition
- Modular design
- Functional parameters
- Timing parameters

**Event based behaviour**

- Precondition
- Synchronization
- Exception
  - Weak T1
  - Strong T2
  - Fatal T3
- Postcondition

## Orccad components: Modules



Implement functions

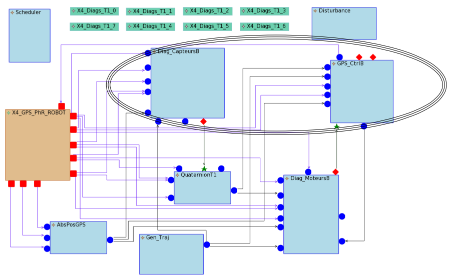
Algorithmic
Phy_Resource (drivers)

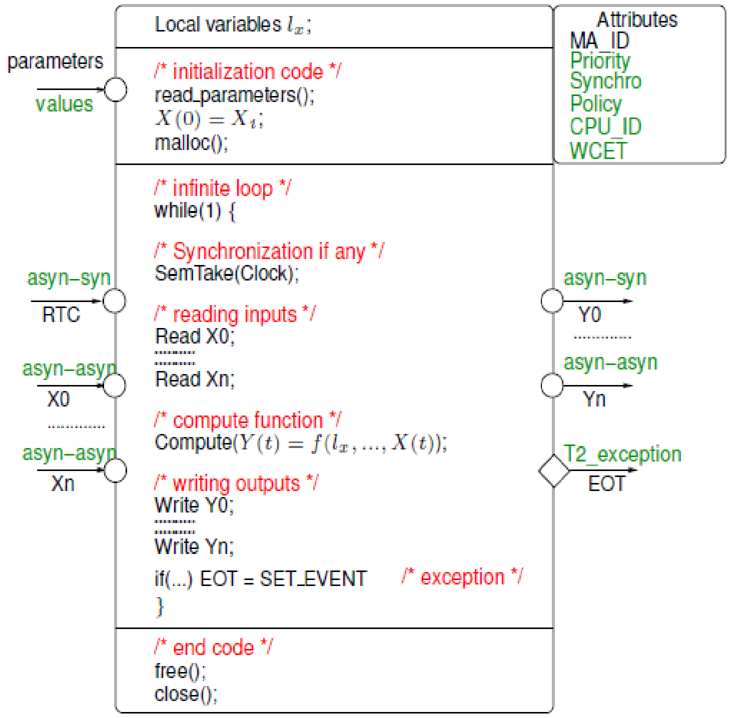
Typed Input/Output ports
• Data
• Drivers
• Parameters
• Events

User defined C code

```
init(inputs)
forever{
compute(inputs)
            }
end()
```

## Orccad components: Temporal Constraint





```
Local variables $l_x$;

/* initialization code */
read_parameters();
$X(0) = X_i$;
malloc();

/* infinite loop */
while(1) {

/* Synchronization if any */
SemTake(Clock);

/* reading inputs */
Read X0;
..........
..........
Read Xn;

/* compute function */
Compute($Y(t) = f(l_x, ..., X(t))$);

/* writing outputs */
Write Y0;
..........
..........
Write Yn;

if(...) EOT = SET_EVENT      /* exception */
}

/* end code */
free();
close();
```

Real time threads

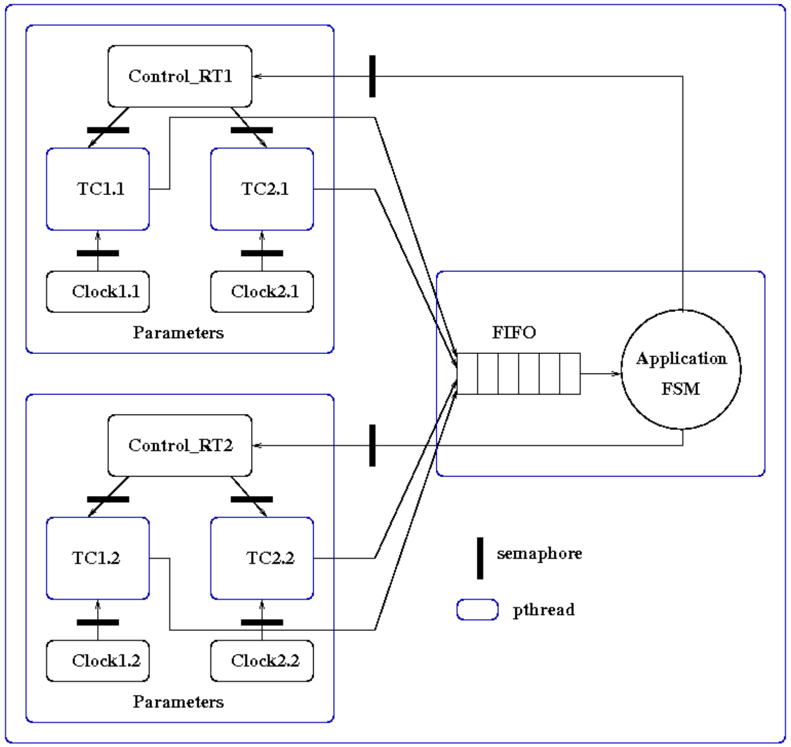
- Task ID
- Modules ID
- Priority
- Synchronization
  - Clock
  - Output port
  - Extern event
- Overrun policy
  - Skip, Soft, Hard
  - User's defined
- WCET
- CPU ID

## Orccad components: RobotProcedure

- **Composition of control actions**
- **Incremental design**
- **From exception processing to mission definition**
- **Currently written in Esterel**

See next talk!

## Runtime



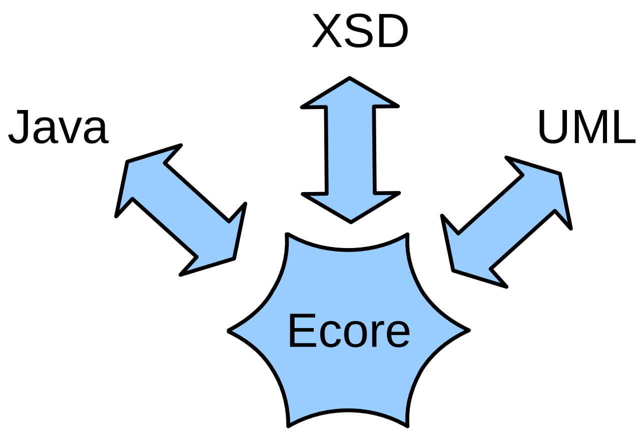
**Code generation**
- **C++ classes**
- **Virtual system calls**

**Compilation**
- **Binding to real calls**
- **Link with specific runtime library**
  - **Linux/Posix**
  - **Xenomai/Native**
  - **...**

| | Orccad | Linux/Posix | Xenomai/Native |
|---|---|---|---|
| launch a real-time task | orcSpawn | pthread_create() | rt_task_spawn() |
| timer | orcTimer_t | timer_t | RT_ALARM |
| message queue | orcsgQ_t | mqd_t | RT_QUEUE |
| semaphore | orcSem_t | sem_t | RT_SEM |

## MDA in Orccad

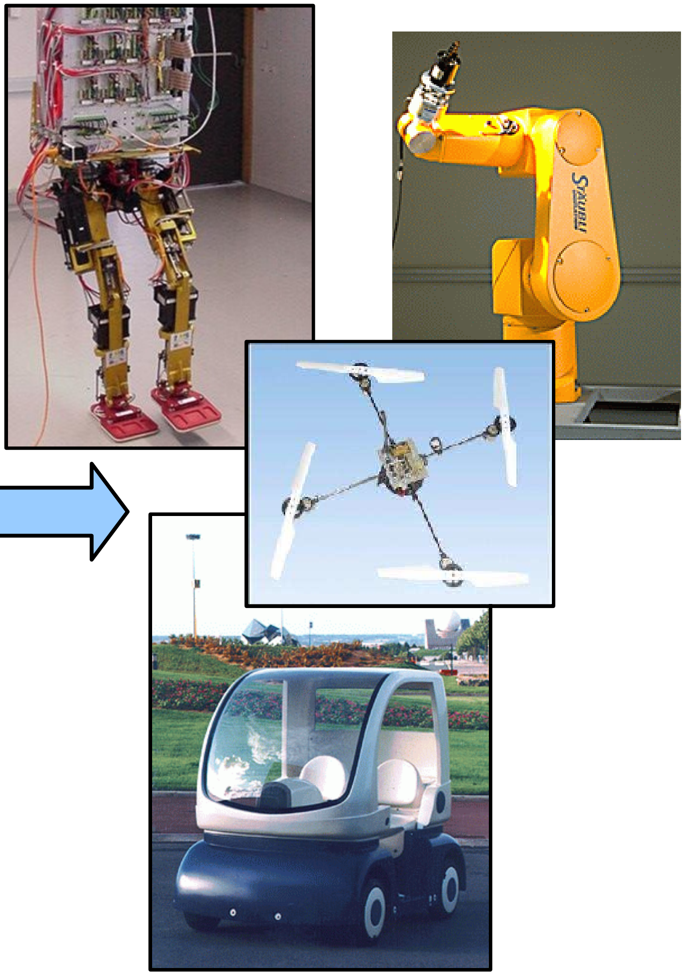
- Eclipse Modeling Project based on the idea of a Model (MetaModel)
- EMP offers different tools for different goals : EMF, GMF, Xpand...
- Principe of plug-in in the Eclipse Environment



## Orccad by Developer & User

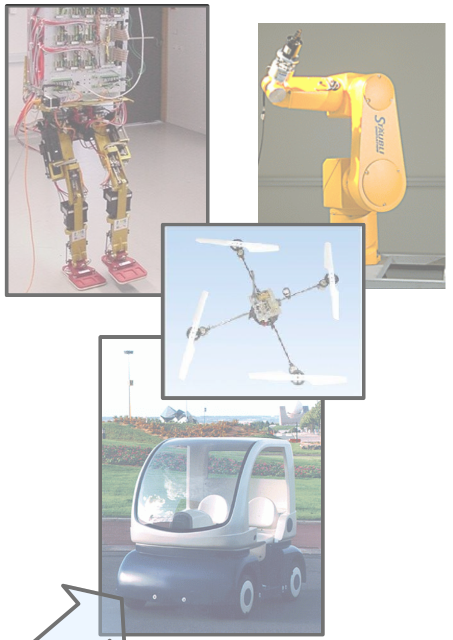
### As a Developer

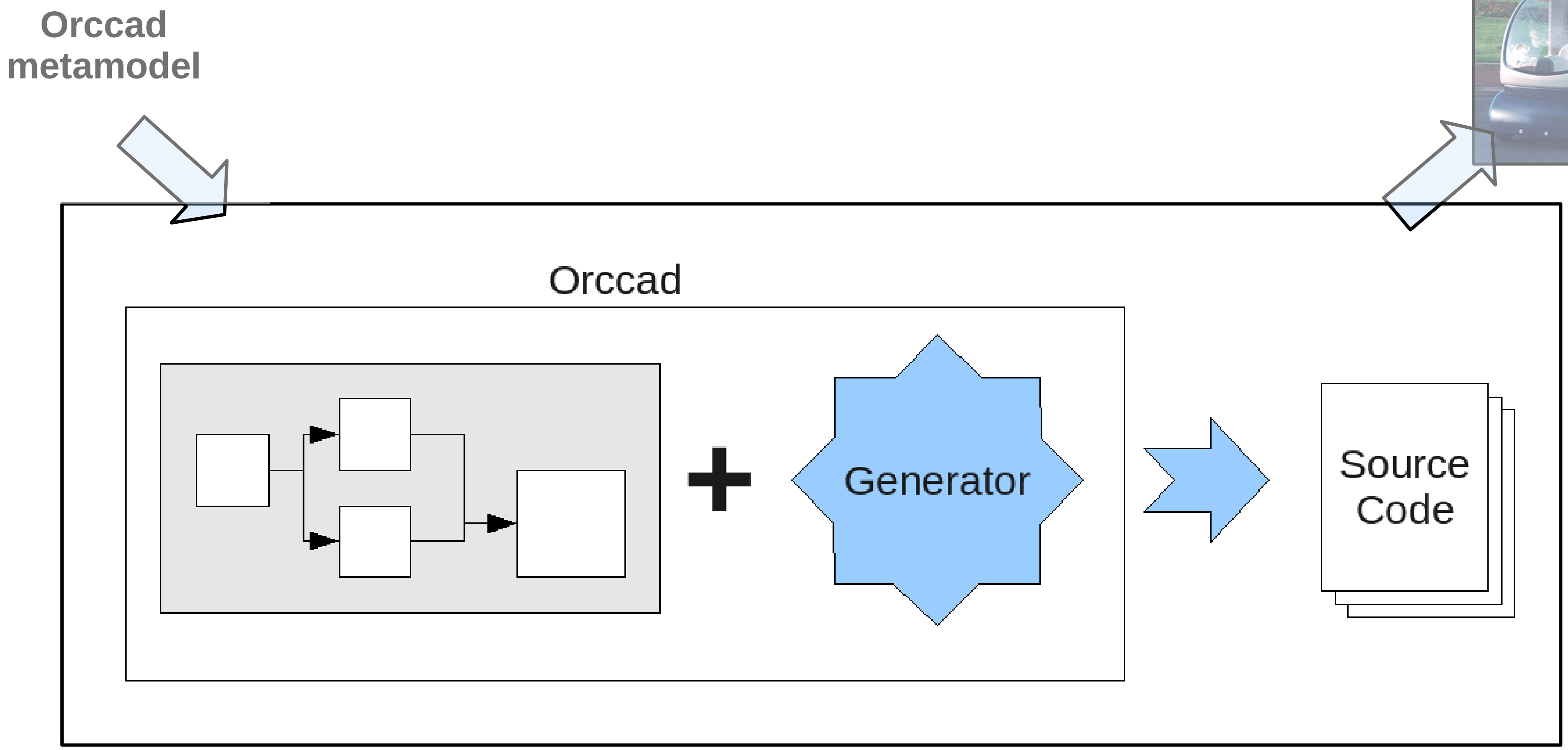
**Orccad MetaModel** → Software →



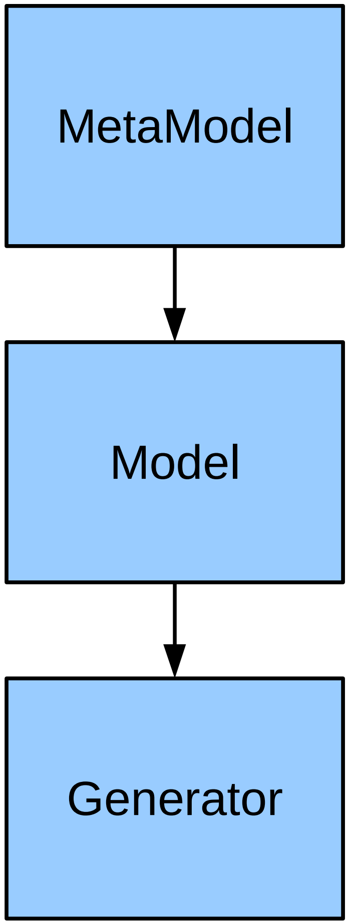
## Orccad by Developer & User

### As a User





## MDA : How it works



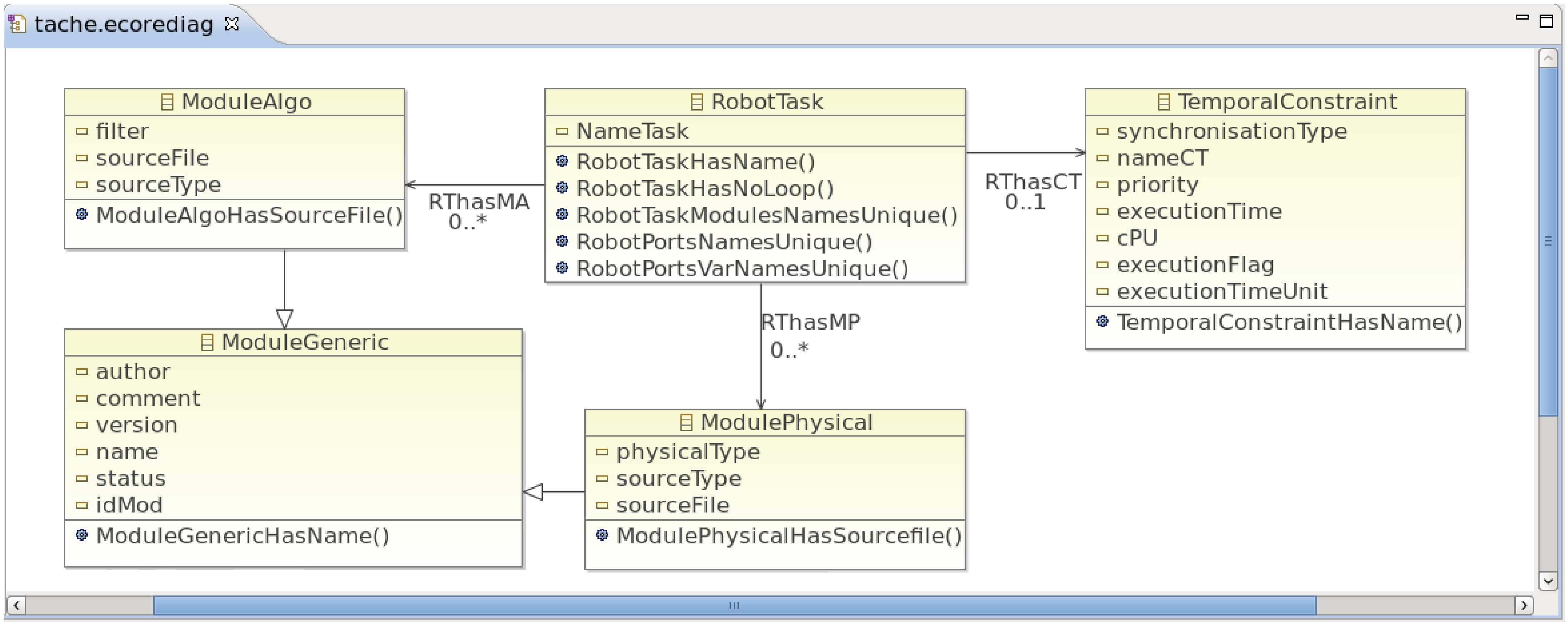
The Metamodel defines how a model is made.
Made by the developer.

The Model is realized by the user.
It matches to the meta-model and its constraints.

It generates the source code from the model, using templates defined by the developer .

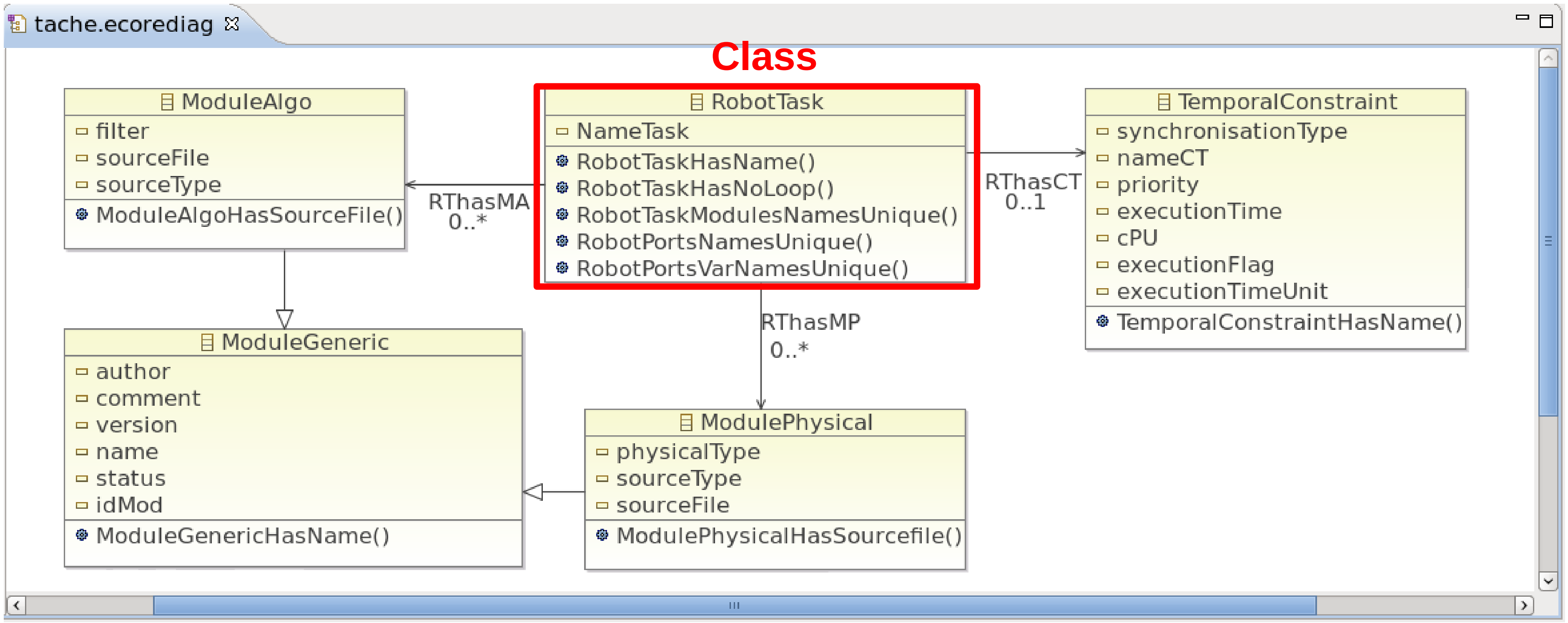
# MetaModel - an example

The graphical view is close to an UML model.



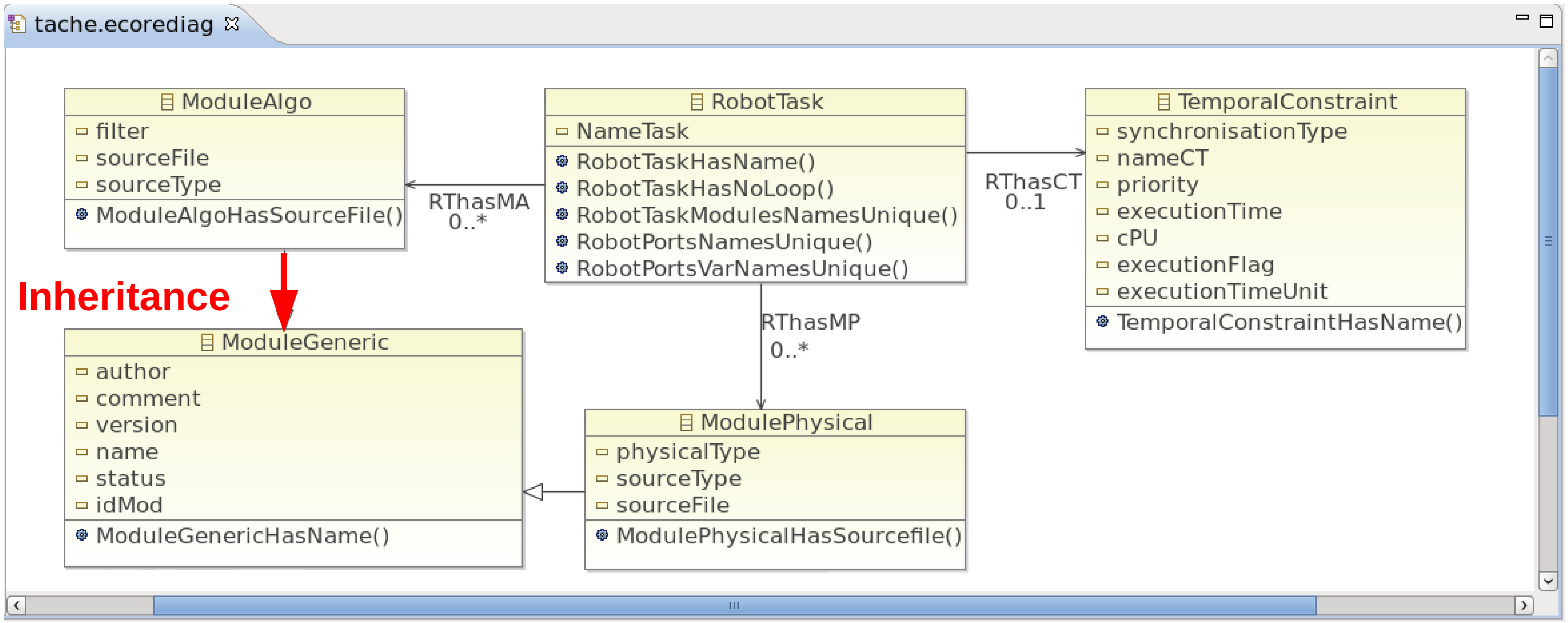
# MetaModel -  an example

Code is generated in Java, we find Java properties in the Ecore model.

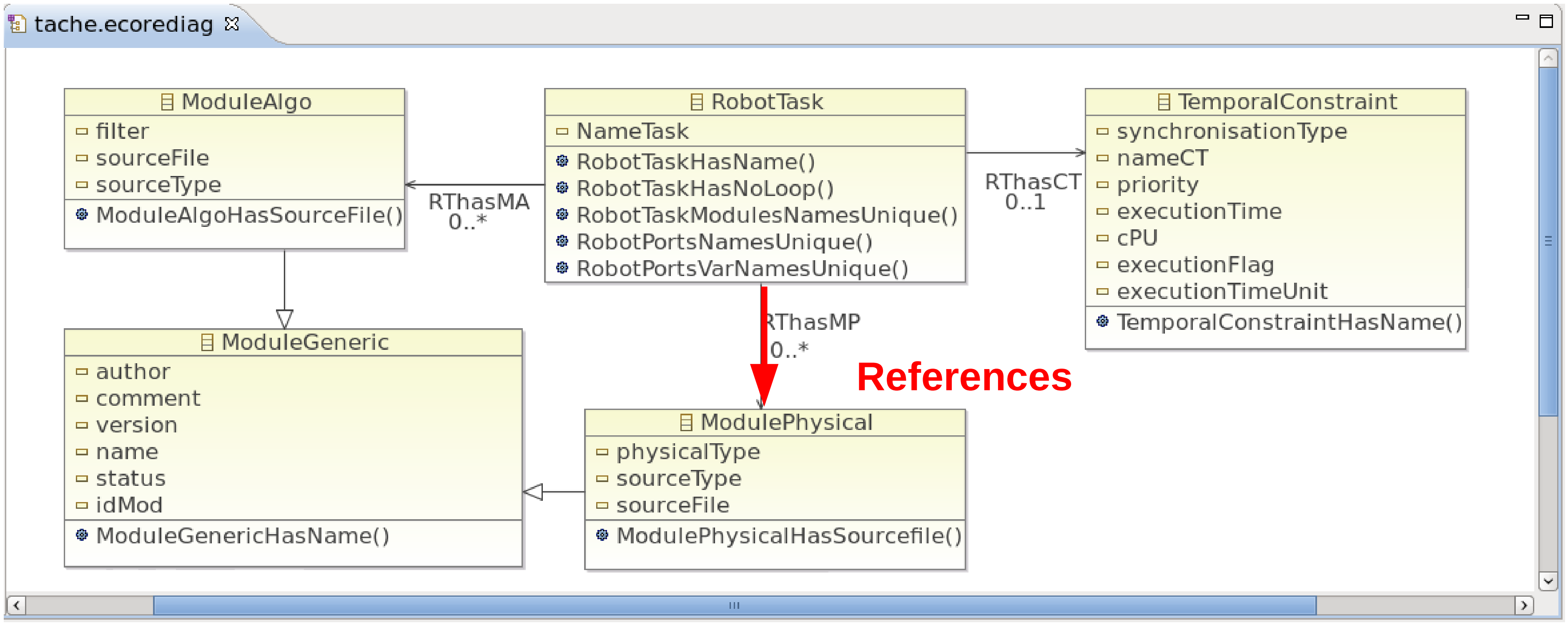


# MetaModel - an example

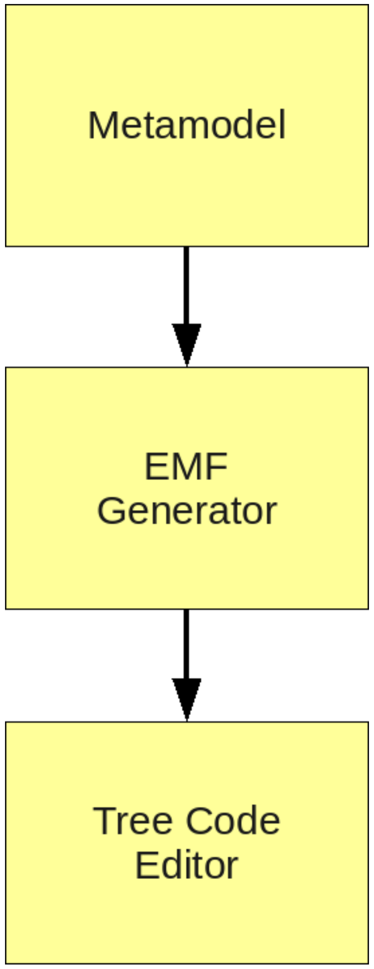
Code generated in Java, we find Java properties in the Ecore model.



# MetaModel - an example



## EMF – Tree Editor

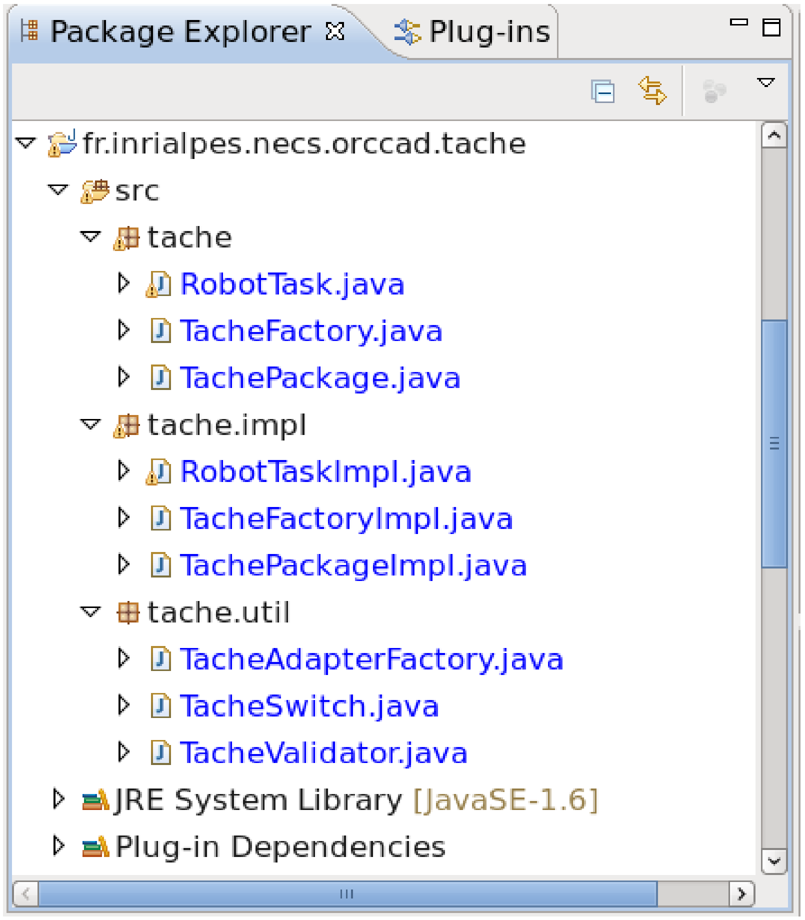


- A plugin developed in the Eclipse project

- From a metamodel, generates a Tree Editor as a plugin
    - For Eclipse
    - RCP plugin

- Really useful to realize beta-version
- Constraints must be defined and filled at this step.

# EMF – Tree Editor

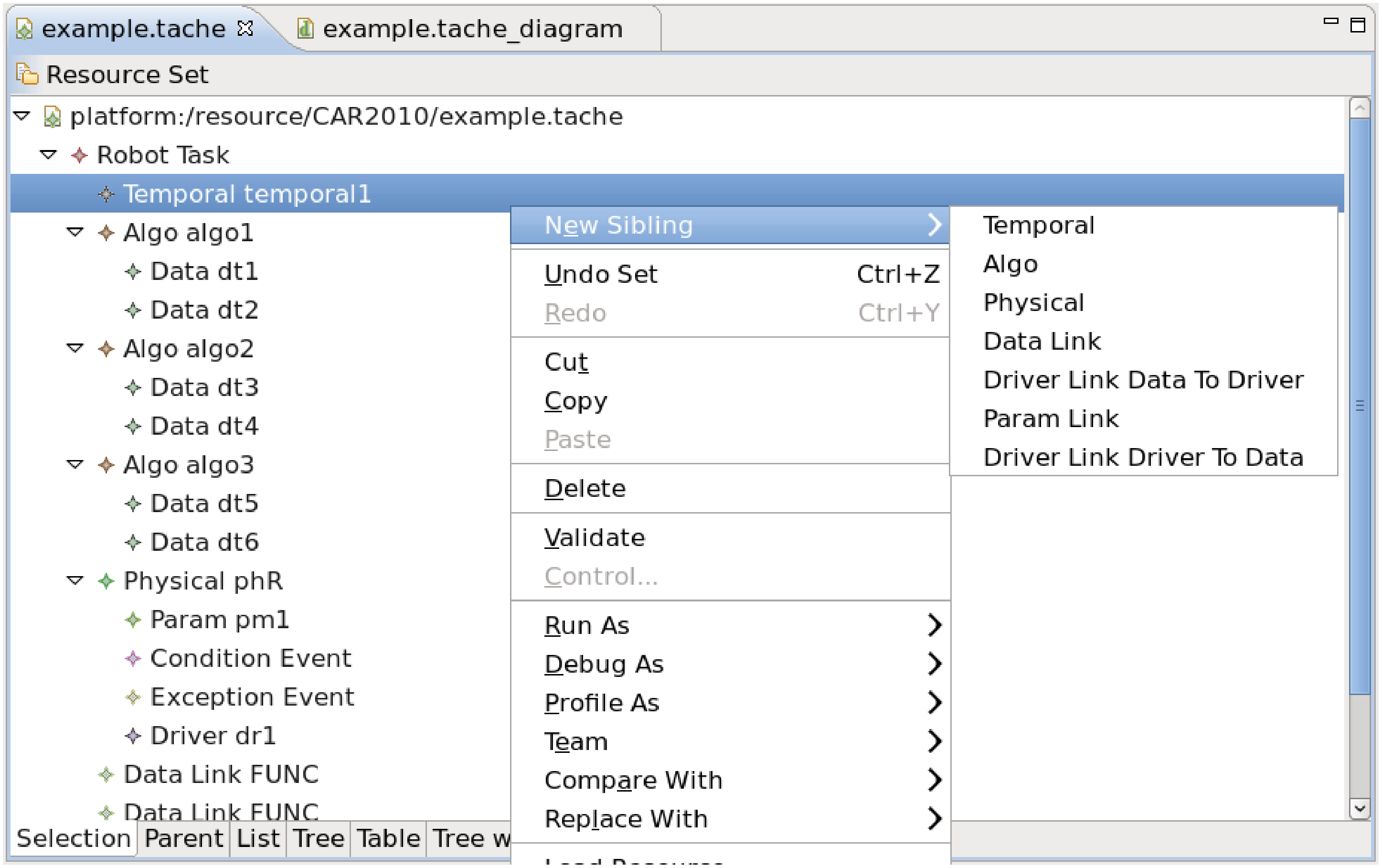


```
fr.inrialpes.necs.orccad.tache
  src
    tache
      RobotTask.java
      TacheFactory.java
      TachePackage.java
    tache.impl
      RobotTaskImpl.java
      TacheFactoryImpl.java
      TachePackageImpl.java
    tache.util
      TacheAdapterFactory.java
      TacheSwitch.java
      TacheValidator.java
  JRE System Library [JavaSE-1.6]
  Plug-in Dependencies
```

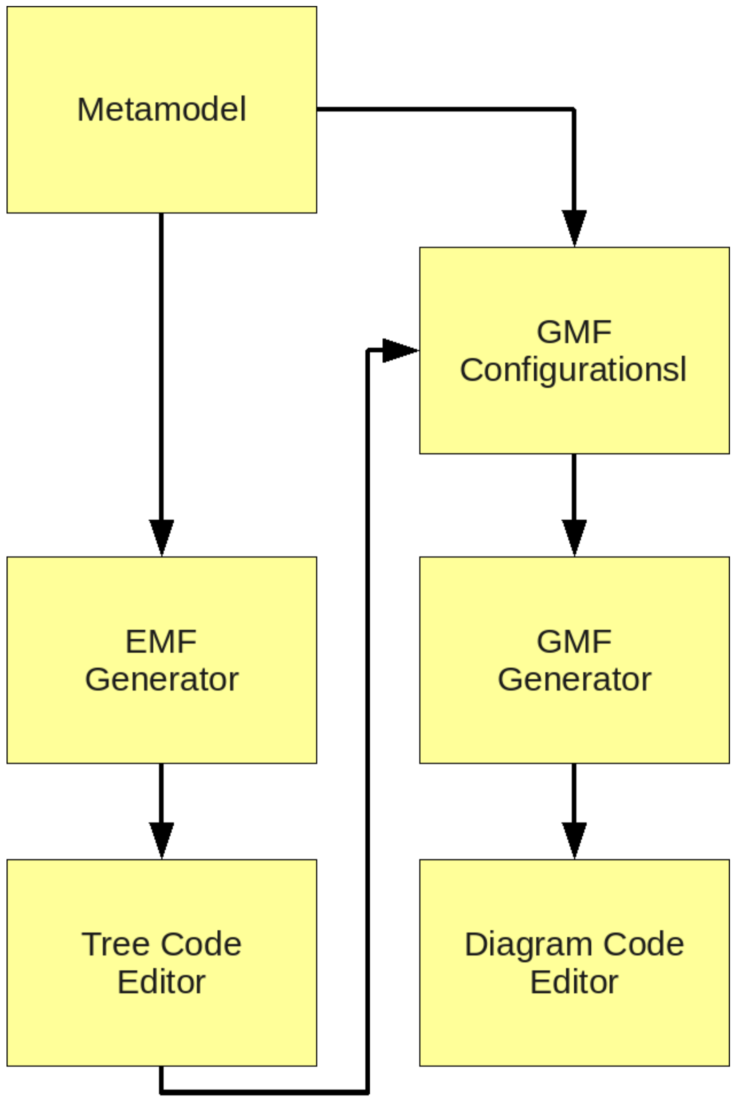
- Generation of Code
  - Creation of a new Project (Plug-in)
  - Packages by functions
  - All the customization on eclipse plug-in are allowed

- Generated code must be modified and/or completed. With keyword, a re-generation of the code is safe.

# EMF – Tree Editor



# GMF – Graphical Editor

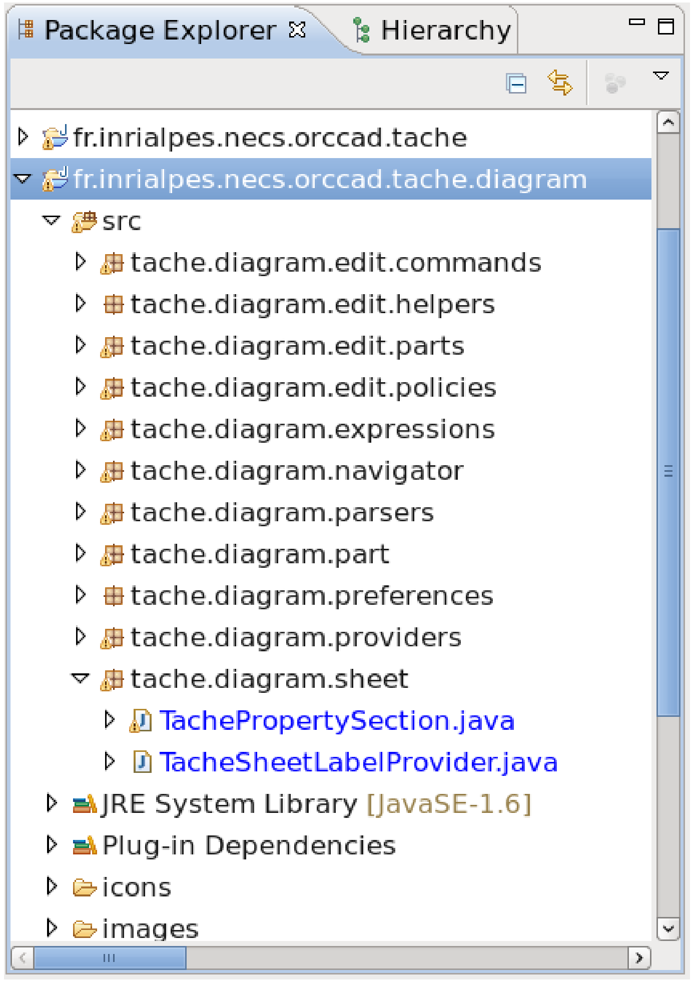


- The Tree Editor must be generated before the generation of the Graphical Editor.

- We specify through files
  - Graphical representations of elements and links
  - Palette tool
  - Mapping, the coherence between view, ecore and palette.

- Then we can generate the Graphical Editor.

## GMF – Graphical Editor



Package Explorer
Hierarchy

- fr.inrialpes.necs.orccad.tache
- fr.inrialpes.necs.orccad.tache.diagram
  - src
    - tache.diagram.edit.commands
    - tache.diagram.edit.helpers
    - tache.diagram.edit.parts
    - tache.diagram.edit.policies
    - tache.diagram.expressions
    - tache.diagram.navigator
    - tache.diagram.parsers
    - tache.diagram.part
    - tache.diagram.preferences
    - tache.diagram.providers
    - tache.diagram.sheet
      - TachePropertySection.java
      - TacheSheetLabelProvider.java
  - JRE System Library [JavaSE-1.6]
  - Plug-in Dependencies
  - icons
  - images

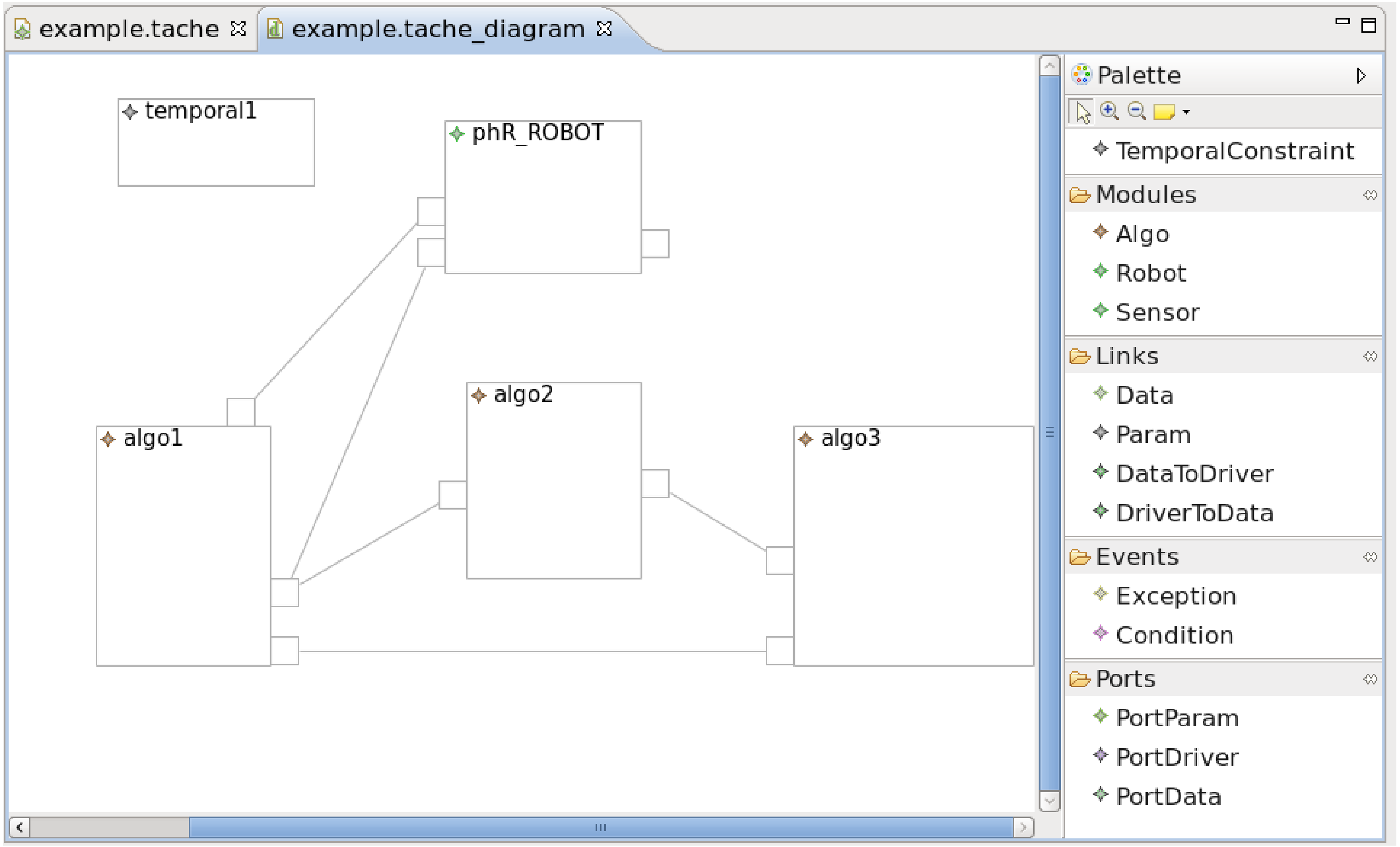
Graphical Interface Code :

- MVC design pattern

Model, Controller and View are independent for a easier maintenance.

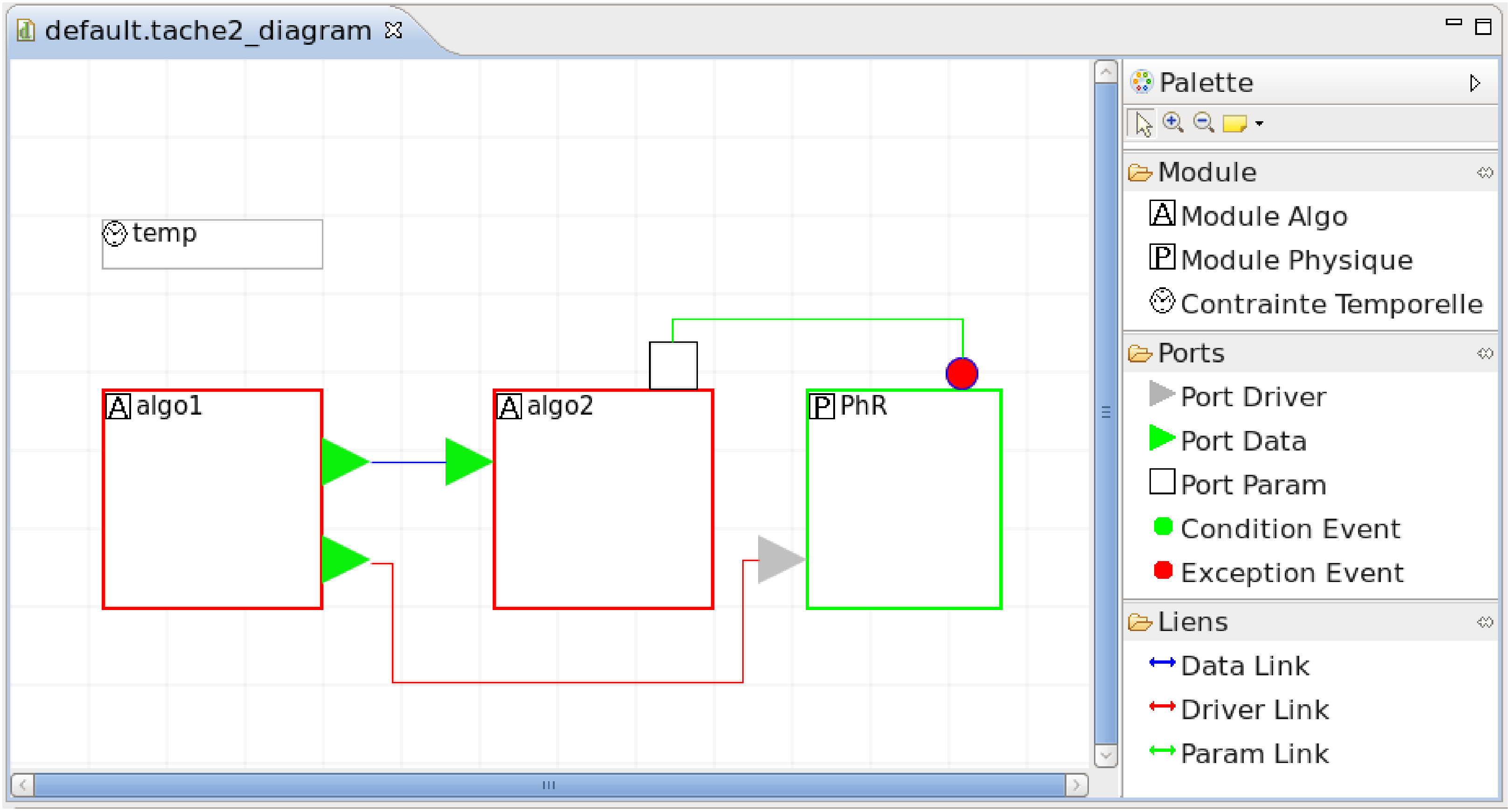
# GMF – Graphical Editor

Result of a quick Graphical Interface uncluttered -> customization !



# GMF – Graphical Editor

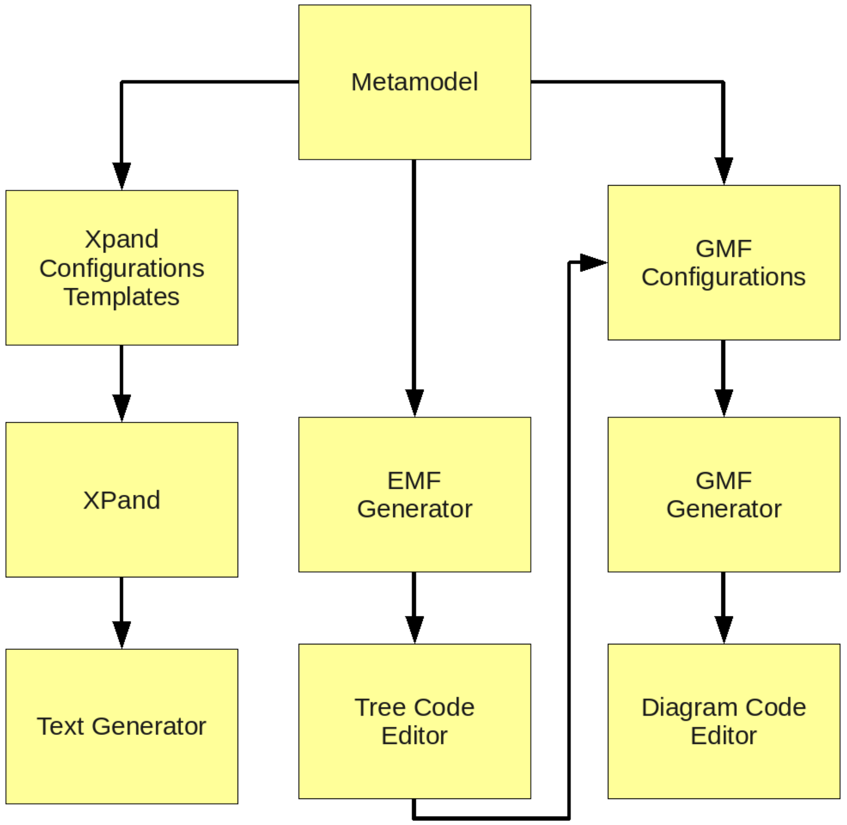
Example of a simple customization



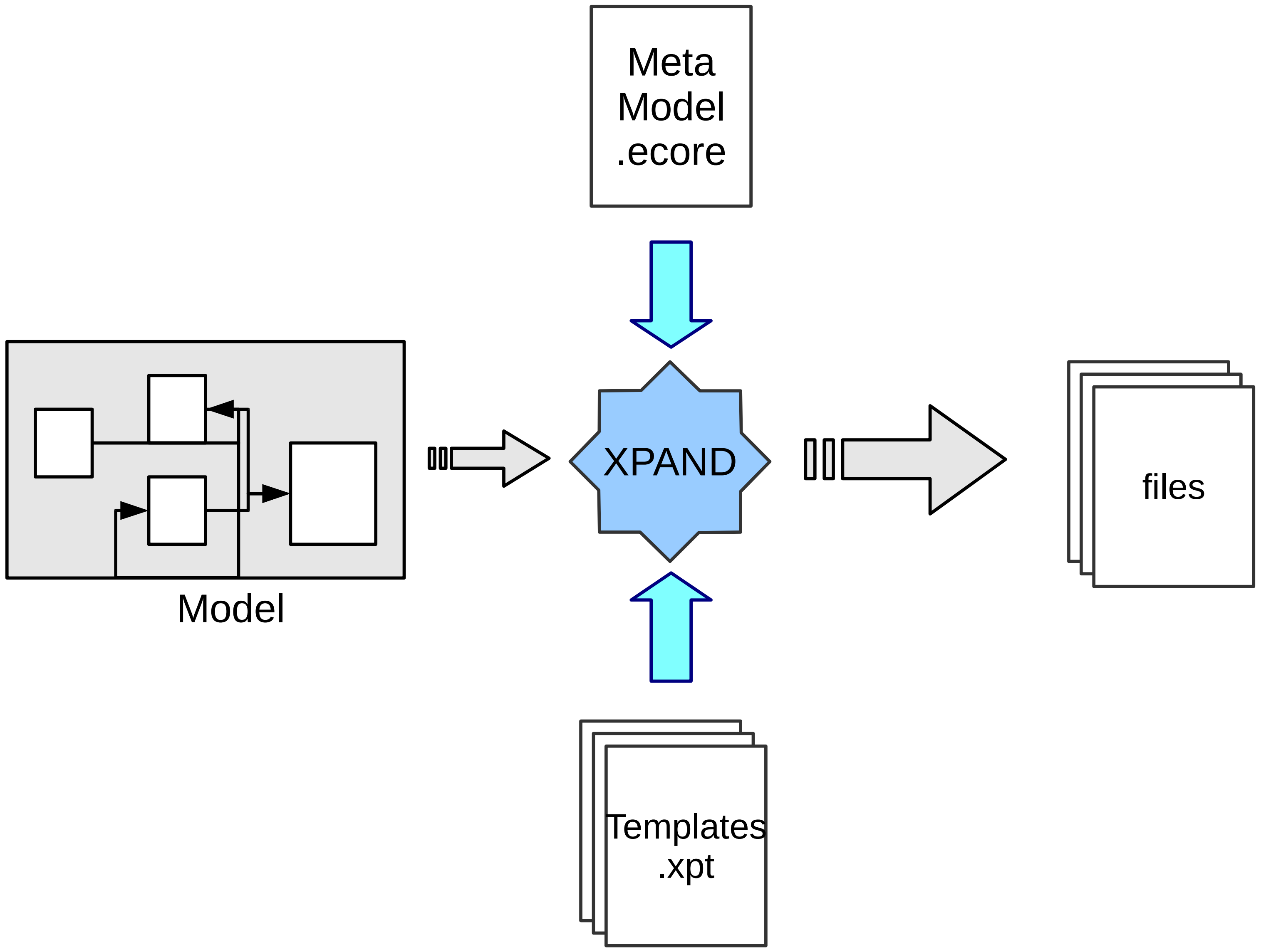
## Xpand – The Code generator

### Why Xpand ?

- Xpand is proposed as a M2T (Model to Text ) technology in the Eclipse Modeling Project
- It fits with the Ecore Metamodel
- Entirely customization for any type of file
- Templates have a simple syntax
- Code generator is independent from the source code



## Xpand – The Code generator



- Source code
- Documentation
- Webpages
- Whatever you want

# Eclipse tooling assessment

- Disadvantages

  ✗ Abandoned tools

  ✗ Choices

  ✗ Technology not easy to master

- Advantages

  ✔ Eclipse Environment

  ✔ Model and Code independence

  ✔ Extensibility/scalability

  ✔ Fast when technology mastered

http://orccad.gforge.inria.fr

Opening soon!

Questions ?