CCSL@work:
How to explicitly defines MoCCs within a model

AOSTE sophia
I3S/UNS/INRIA

Synchron 2010

CCSL@work:
the RT-Simex project

(or “a mean to check an implementation against its specification”)

AOSTE sophia
I3S/UNS/INRIA

Frederic Thomas, Étienne Juliot
Obeo

Jean-Philippe Babau
UBO

and more....

Synchron 2010

Logical Time @ work:
How to explicitly defines MoCCs within a model

AOSTE sophia
I3S/UNS/INRIA

Synchron 2010

Logical Time...

- focuses on causal relations between events
- Is independent of the abstraction level
- Is multi-clock (polychronous)
- Provides a partial order between events

After 23 starts of the computer, a disk check is done

The computation duration is 156 processor ticks

Logical Time...

- focuses on causal relations between events
- Is independent of the abstraction level
- Is multi-clock (polychronous)
- Provides a partial order between events

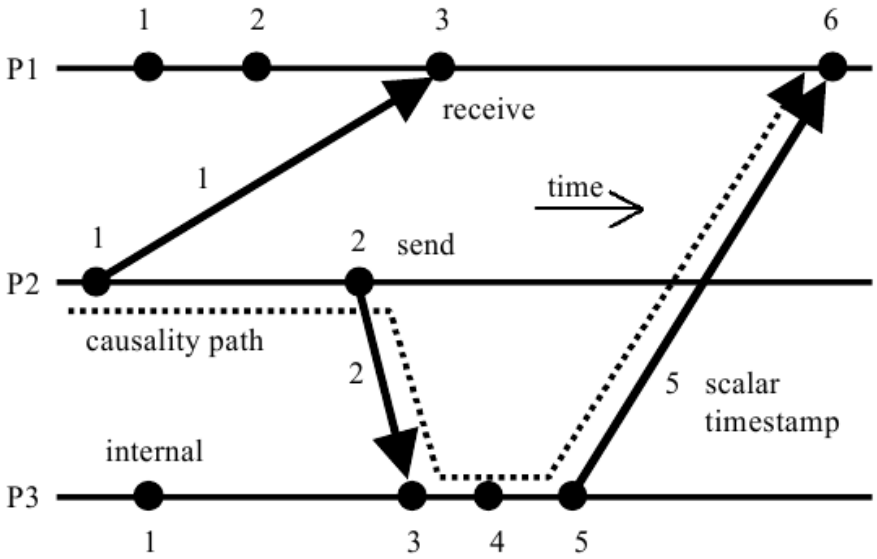
After 23 starts of the computer, a disk check is done

The computation duration is 156 processor ticks

In modern laptop, tick is logical since the processor speed depend on the battery level

Logical Time...

- focuses on causal relations between events
- Is independent of the abstraction level
- Is multi-clock (polychronous)
- Provides a partial order between events



LOGICAL TIME
Friedemann Mattern
Darmstadt University of Technology

Physical Time...

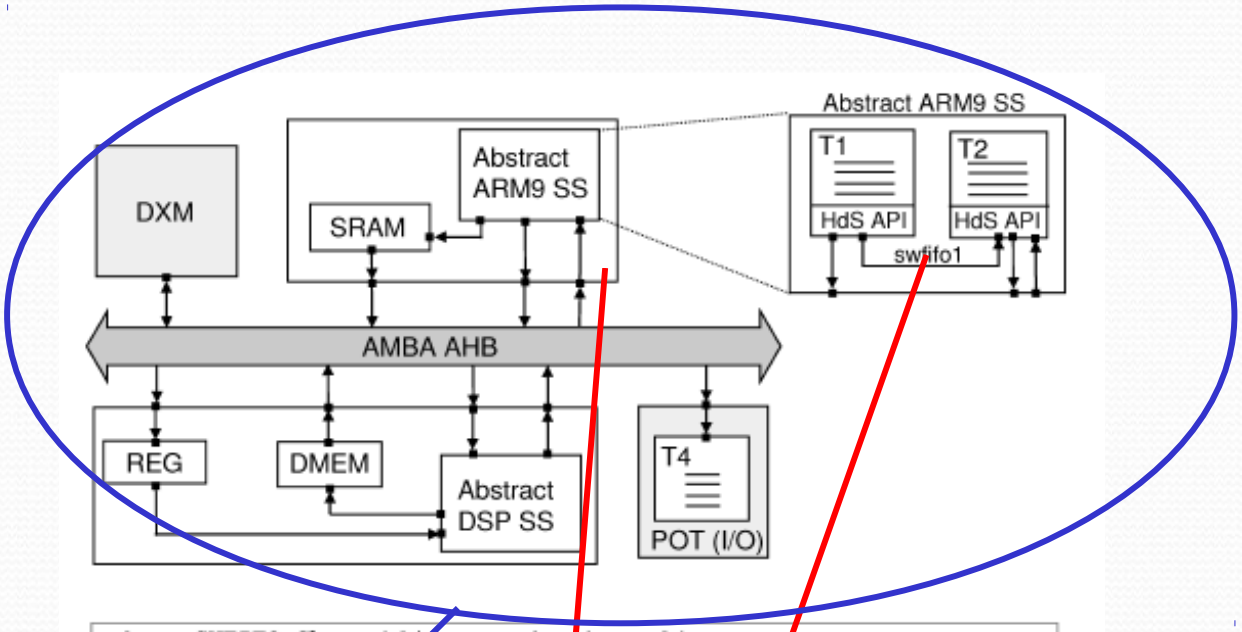
- Can be seen as a special case of logical time

After 5 secondes, it stops...

$\cong$

After 5 events of the "second" clock, it stops

Modeling: an example




```
class SWFIFO_Ch: public sc_prim_channel{
  list<int> buffer;
  void recv_data(void *dst, int size){
       for (int i=0; i<size; i++)
            dst[i]=buffer.pop;
```

```
SC_MODULE(TOP){
 ARM_SS *vARM;
 DXM *vDXM;
 BUS *vBUS;
 ...
 SC_CTOR(TOP){
```

```
SC_MODULE(ARM_SS){
 Task1 *vTask1;
 SWFIFO_Ch swfifo1;

 SC_CTOR(ARM_SS){
 ...
```

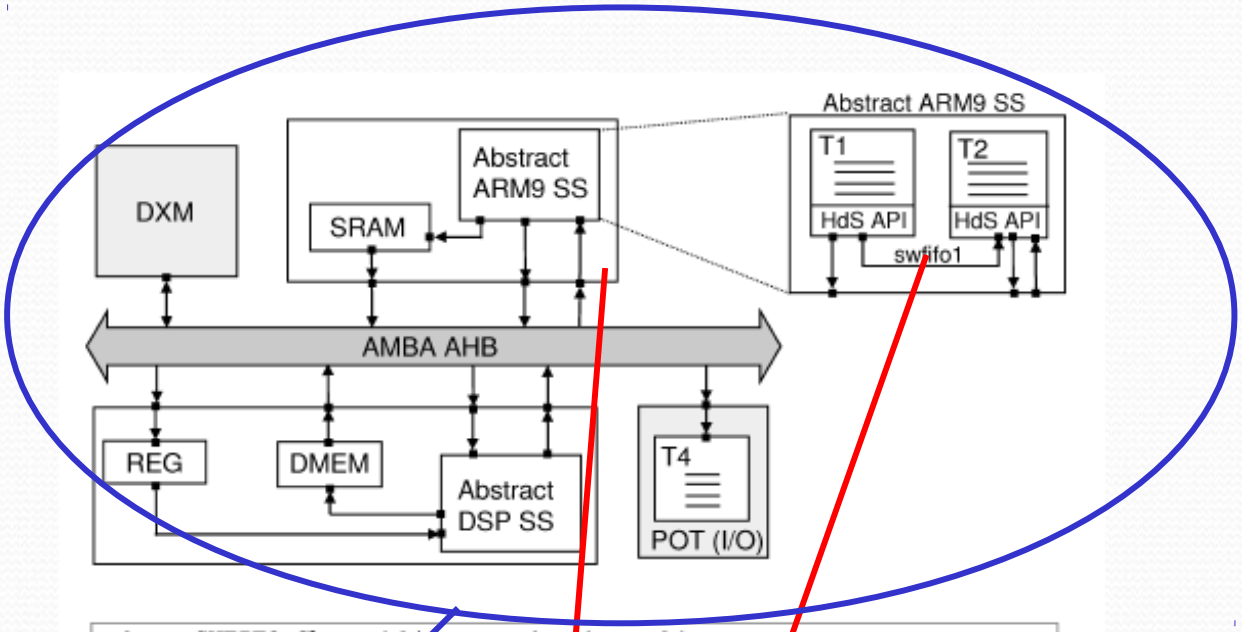
```
SC_MODULE(Task1){
 SWFIFO_Port In1;
 ...
 void Task1_beh();//behavior
 SC_CTOR(Task1){
  SC_CTHREAD(Task1_beh,clk);
 ...
```

Fig. 8. Virtual architecture Diopsis model.

Structure

Behavior

Modeling: an example




```
class SWFIFO_Ch: public sc_prim_channel{
  list<int> buffer;
  void recv_data(void *dst, int size){
       for (int i=0; i<size; i++)
            dst[i]=buffer.pop;
```

```
SC_MODULE(TOP){
 ARM_SS *vARM;
 DXM *vDXM;
 BUS *vBUS;
 ...
 SC_CTOR(TOP){
```

```
SC_MODULE(ARM_SS){
 Task1 *vTask1;
 SWFIFO_Ch swfifo1;

 SC_CTOR(ARM_SS){
 ...
```

```
SC_MODULE(Task1){
 SWFIFO_Port In1;
 ...
 void Task1_beh();//behavior
 SC_CTOR(Task1){
  SC_CTHREAD(Task1_beh,clk);
 ...
```

Fig. 8. Virtual architecture Diopsis model.

Structure

Elaboration phase (SystemC)

Behavior

Simulation

Traditional modelling approach

- Structure
  - Black-box + Interfaces *(Ports, Data Types)*
- Behavioral abstraction
  - Messages + *possibly period and performance requirements, …*

- *What we find missing:*
  - Detailed definition of timing and synchronization properties
  - Communication protocol requirements

- *This missing information is often deported elsewhere*

Time & Semantics

- "physical" time
  - Extra functional
  - Single time (total order)
  - Timing constraints to be satisfied at execution
  - Simulation semantics possibly different from synthesis
  - UML, SystemC

HDLs

Time & Semantics

- Logical functional time
  - Functional: sequence of reaction steps
  - Multiple times (local / global)
  - Synchronization primitives → constraints between local activation times
  - Synthesis / Compilation
  - Process networks (SDF), synchronous reactive formalisms, statecharts
- "physical" time
  - Extra functional
  - Single time (total order)
  - Timing constraints to be satisfied at execution
  - Simulation semantics possibly different from synthesis
  - UML, SystemC

HDLs

Time as semantics

- Logical functional time
  - Functional: sequence of reaction steps
  - Multiple times (local / global)
  - Synchronization primitives → constraints between local activation times
  - Synthesis / Compilation
  - Process networks (SDF), synchronous reactive formalisms, statecharts

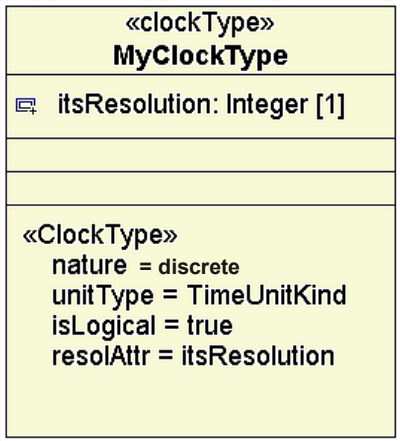
HDLs

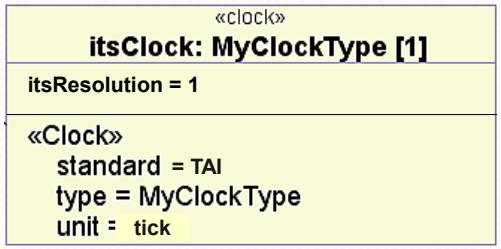
- "physical" time
  - Extra functional
  - Single time (total order)
  - Timing constraints to be ecution
  - Simulation semantics possibly different from synthesis
  - UML, SystemC

Our choice

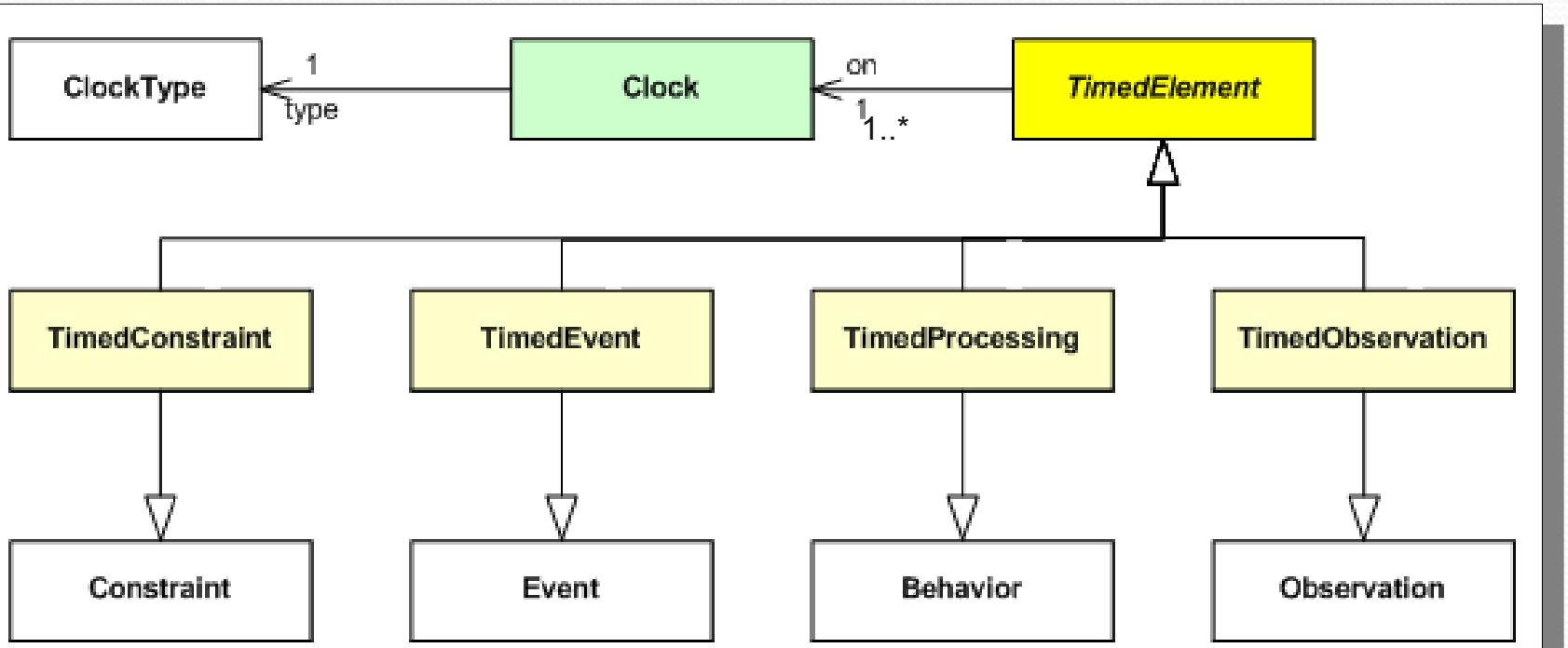
Logical Time (in MARTE TIME MODEL)

- The main concept is the **Clock**.
  - It is a way to specify a, possibly infinite, ordered set of instants
  - It can be logical or chronometric, discrete or dense
  - Its type is a ClockType



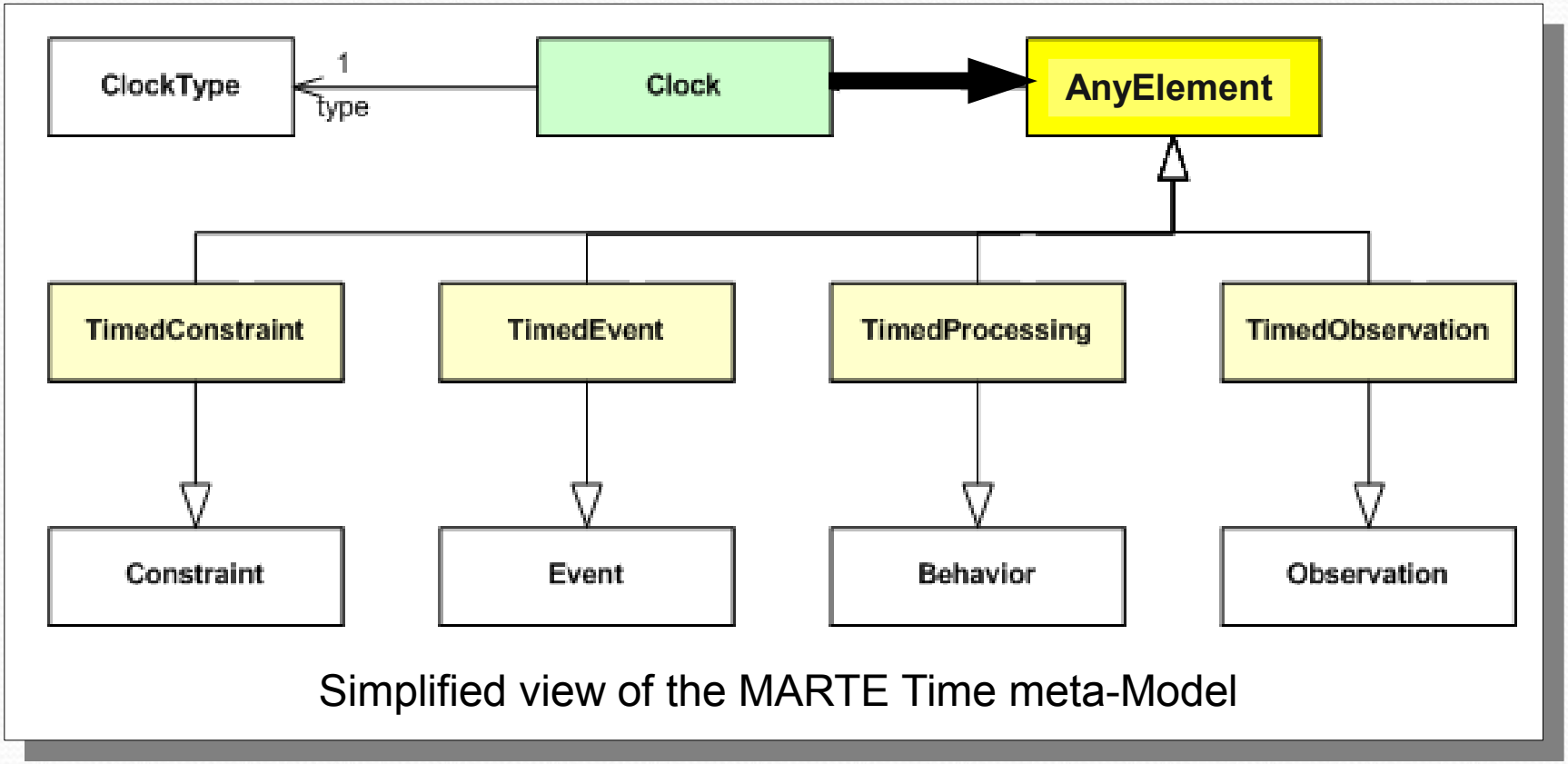


MARTE TIME MODEL



Simplified view of the MARTE Time meta-Model

An extension....

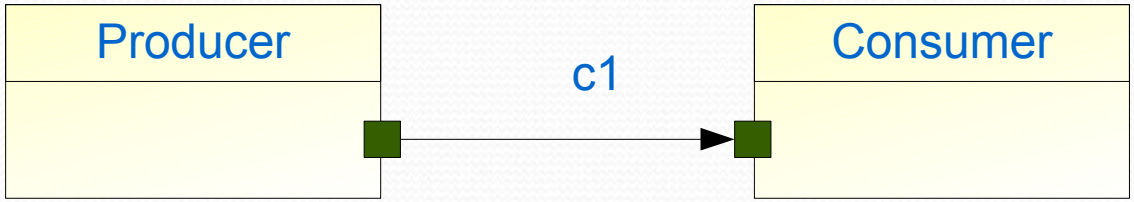


Simplified view of the MARTE Time meta-Model

MARTE TIME MODEL

- Sketchy example of its use

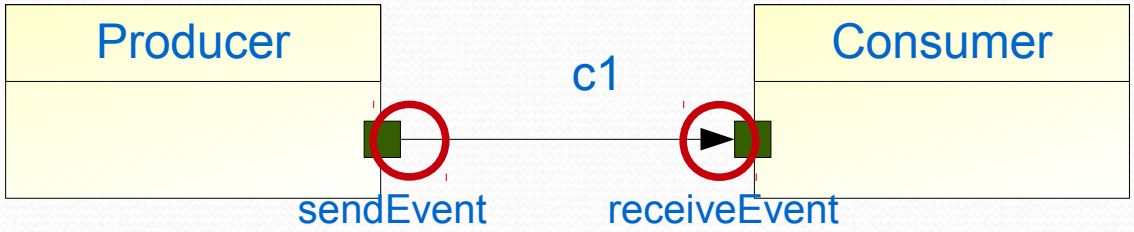
User model



Logical time in models

- Sketchy example of its use

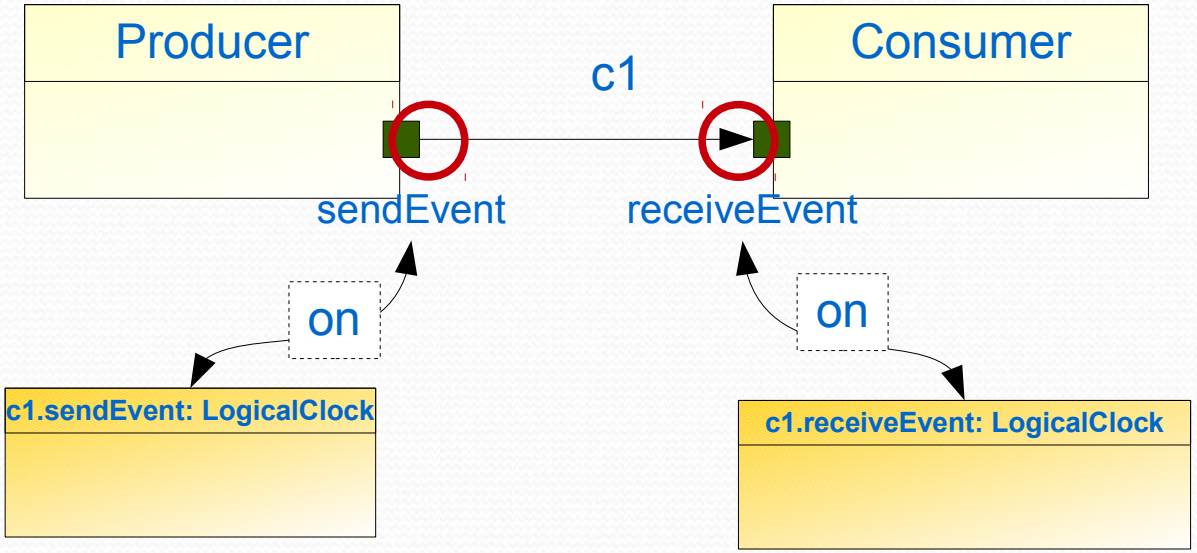
User model



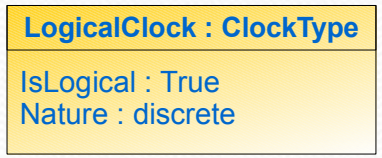
Logical time in models

- Sketchy example of its use

User model



MARTE model



The ordered set of `sendEvent` is bijective with the ordered set of instants of `c1.sendEvent`

CCSL

- **C**lock **C**onstraint **S**pecification **L**anguage
  - Firstly introduced in the MARTE TIME profile
  - Declarative model-based language integrated with Eclipse
  - Formal semantics (both denotational and operational)
  - Tooled (TimeSquare)

→ Explicitly represents / specifies relations between clocks

CCSL (**C**lock **C**onstraint **S**pecification **L**anguage)

- Relations: dependencies between clocks
  - Coincidence → **=**
  - Exclusion → **#**
  - Precedence → **<**
  - Alternance → **~**

- Expressions: a mean to create new clocks from others
  - Delay → **delayedFor** *X* **on** *aClock*
  - Filtering → *aClock* **filteredBy** *aBinaryWord*
  - Union → *aClock* **union** *anotherClock*
  - Intersection → *aClock* **inter** *anotherClock*
  - Periodicity → **periodicOn** aClock **period** X **offset** Y
  - …

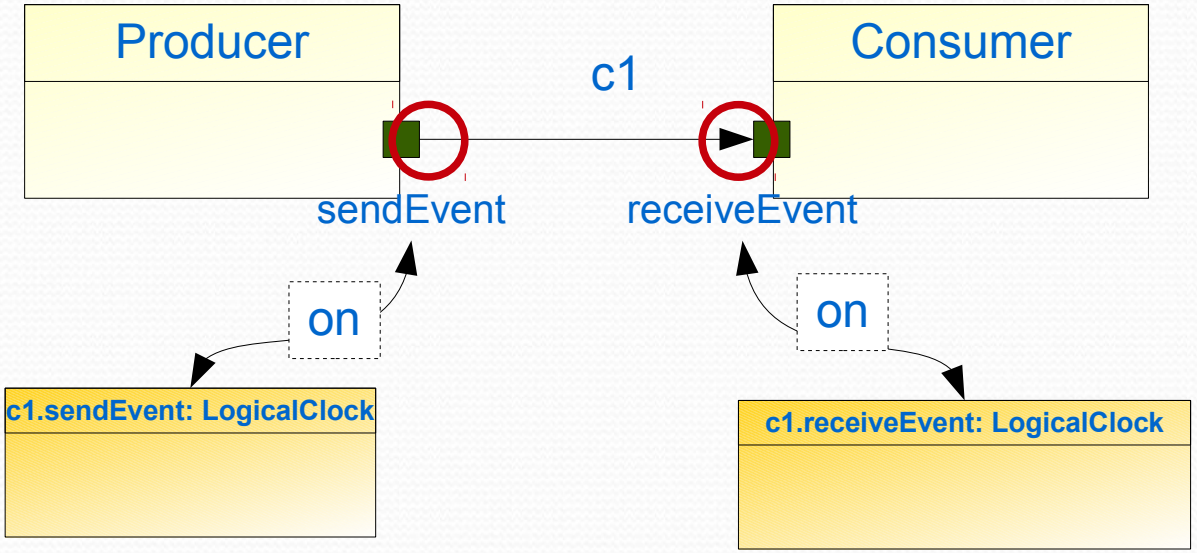
CCSL (**C**lock **C**onstraint **S**pecification **L**anguage)

- Relations: dependencies between clocks
  - Coincidence → **=**
  - Exclusion → **#**
  - Precedence → **<**
  - Alternance → **~**
- Expressions: a mean to create new clocks from others
  - Delay → **delayedFor** *X* **on** *aClock*
  - Filtering → *aClock* **filteredBy** *aBinaryWord*
  - Union → *aClock* **union** *anotherClock*
  - Intersection → *aClock* **inter** *anotherClock*
  - Periodicity → **periodicOn** aClock **period** X **offset** Y
- Libraries: user-defined relations and expressions for a specific domain

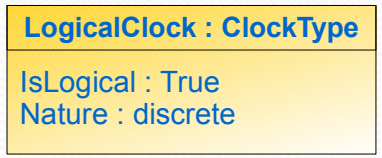
CCSL

- Sketchy example of its use

User model



MARTE model

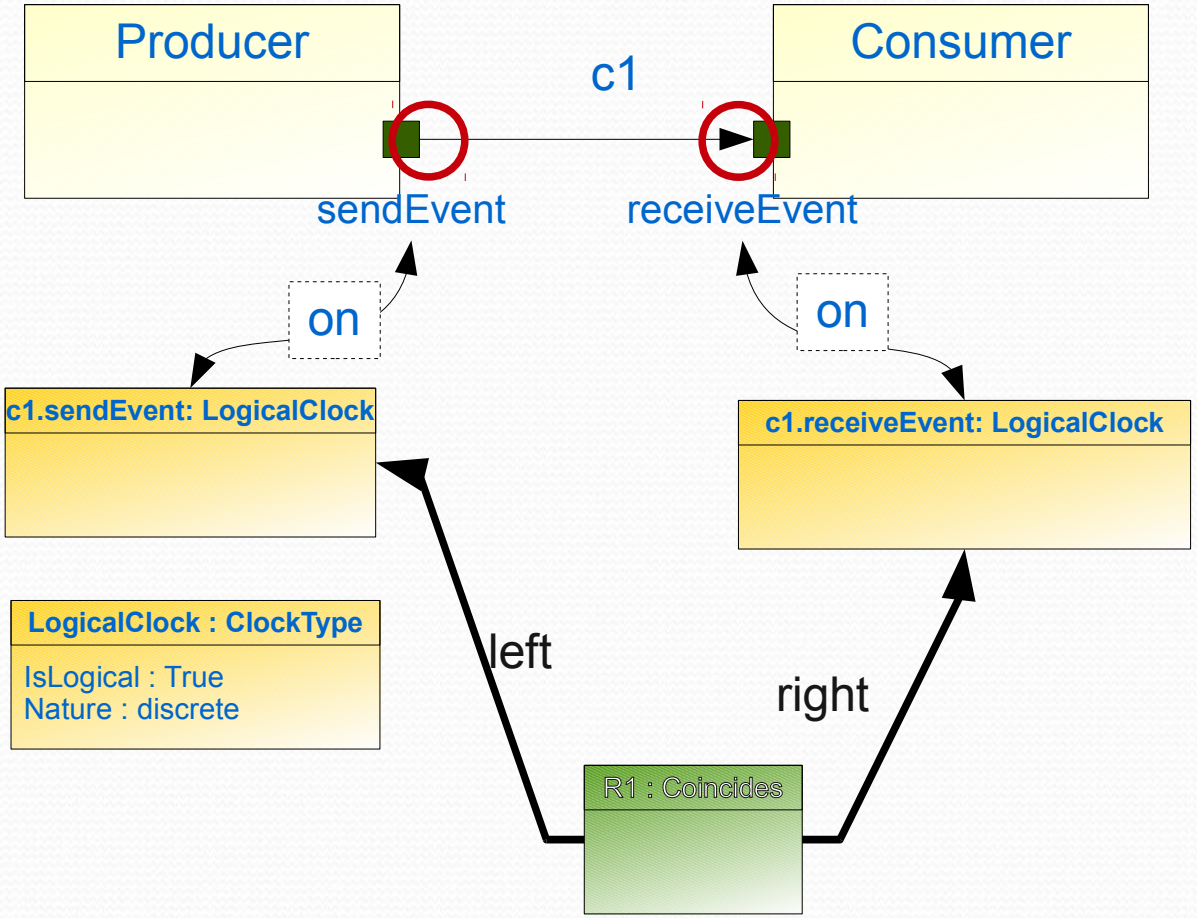


The ordered set of `sendEvent` is bijective with the ordered set of instants of `c1.sendEvent`

CCSL

- Sketchy example of its use

User model

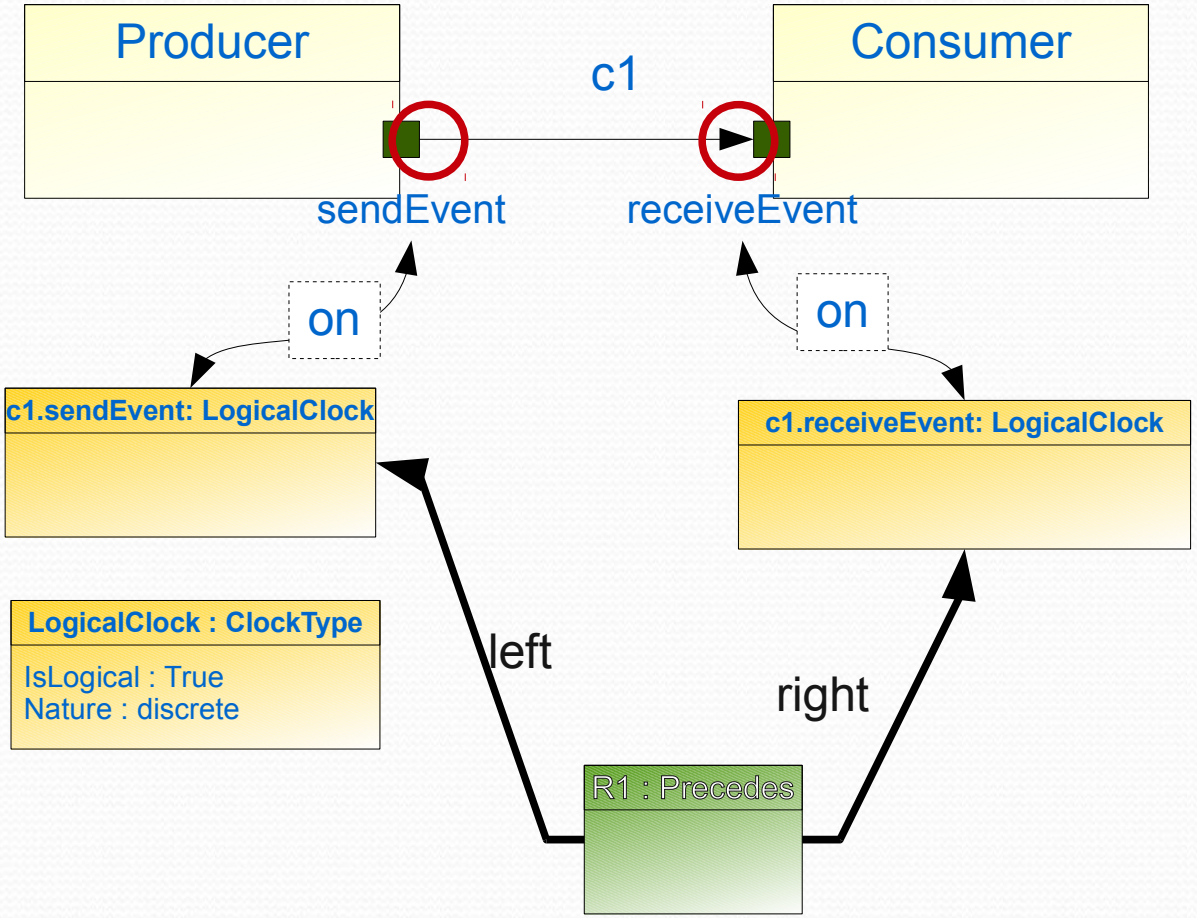


MARTE model

CCSL

- Sketchy example of its use

User model



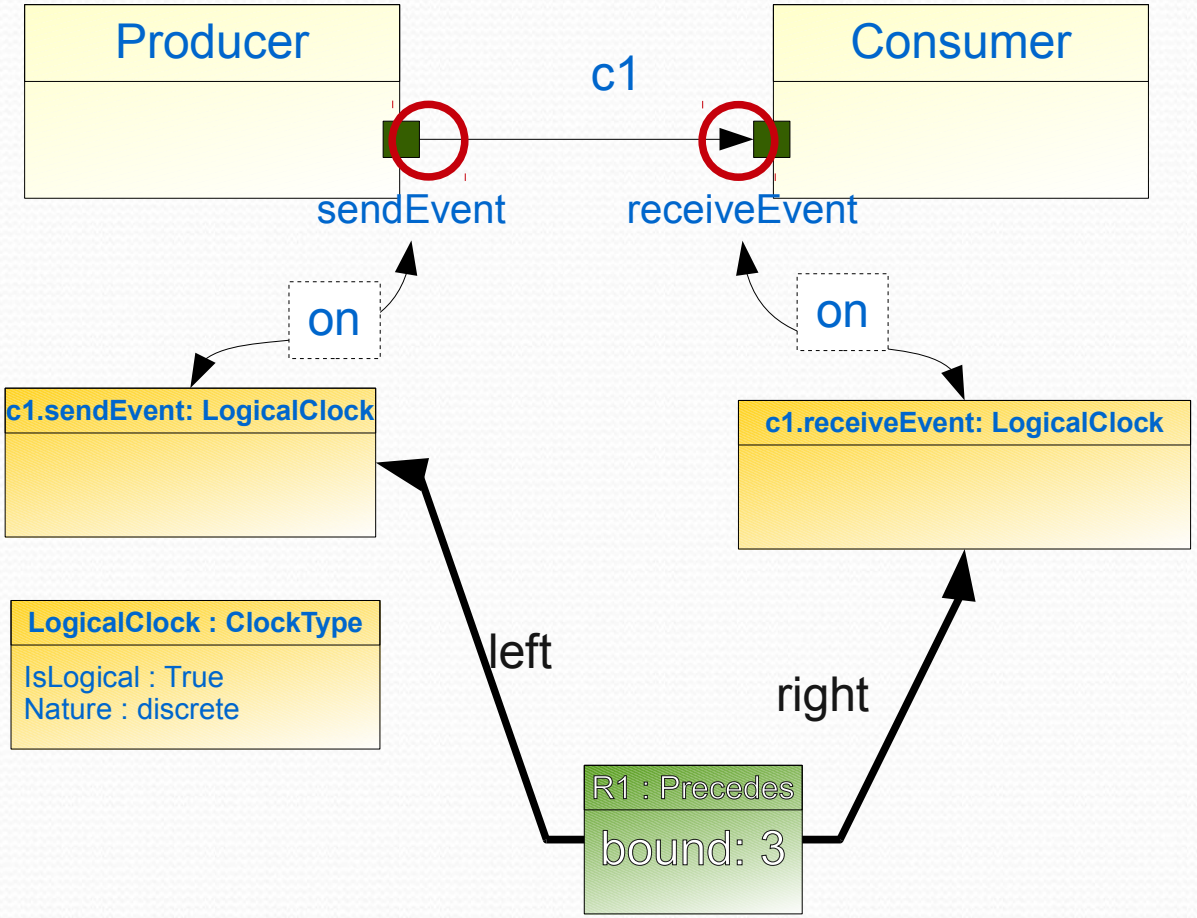
MARTE model

CCSL

- Sketchy example of its use

User model

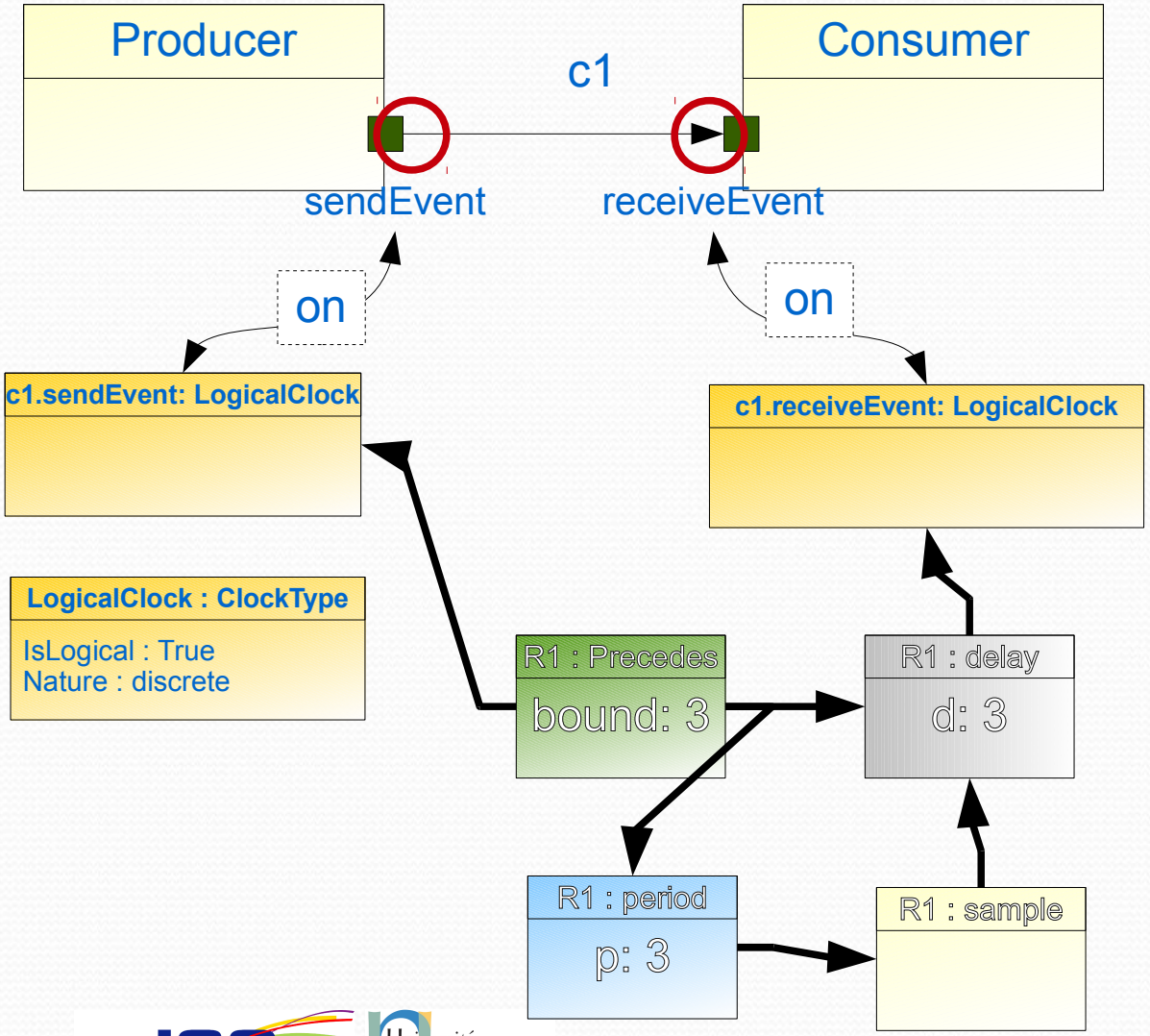
MARTE model



CCSL

- Sketchy example of its use

User model

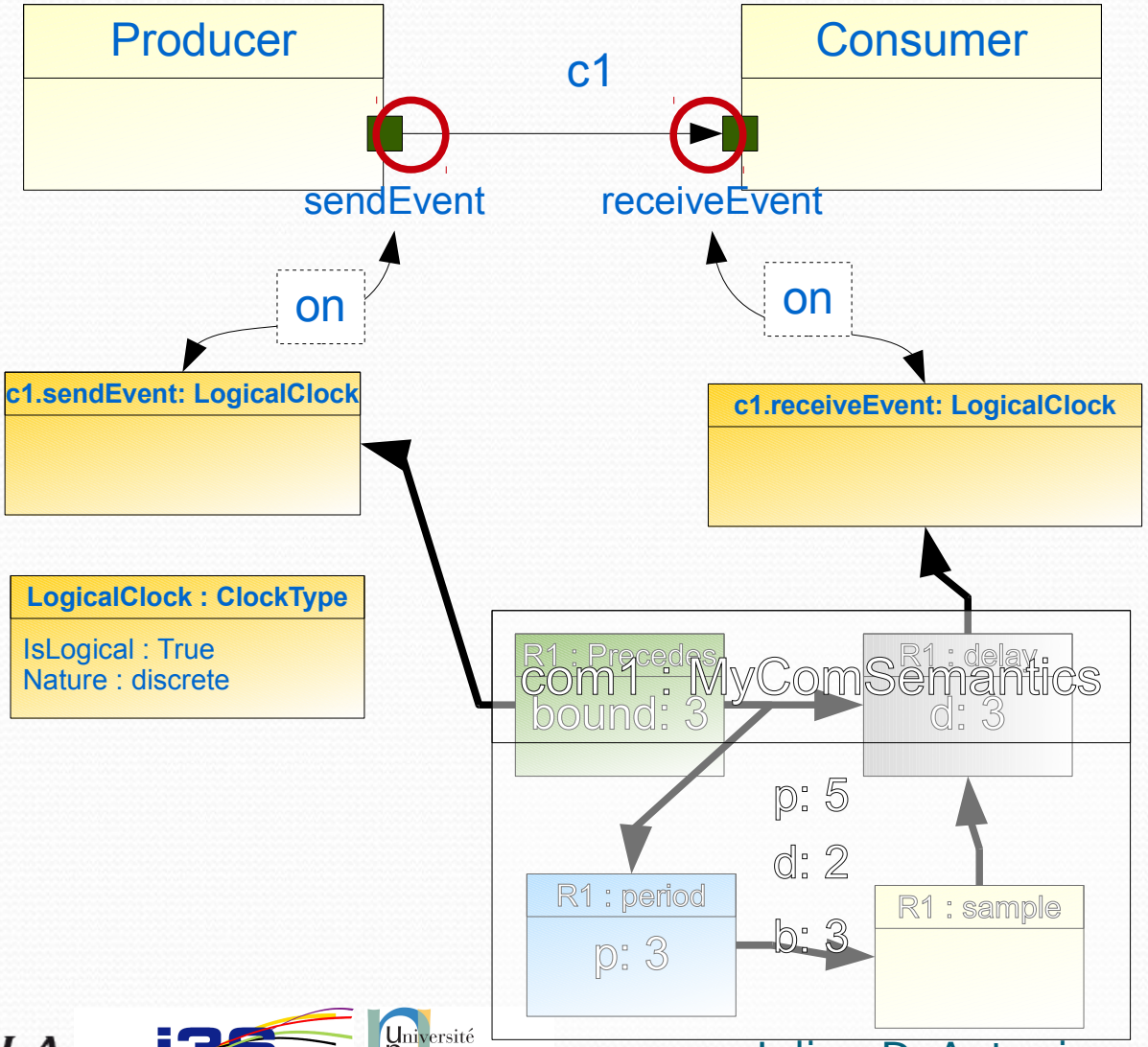


MARTE model

CCSL

- Sketchy example of its use

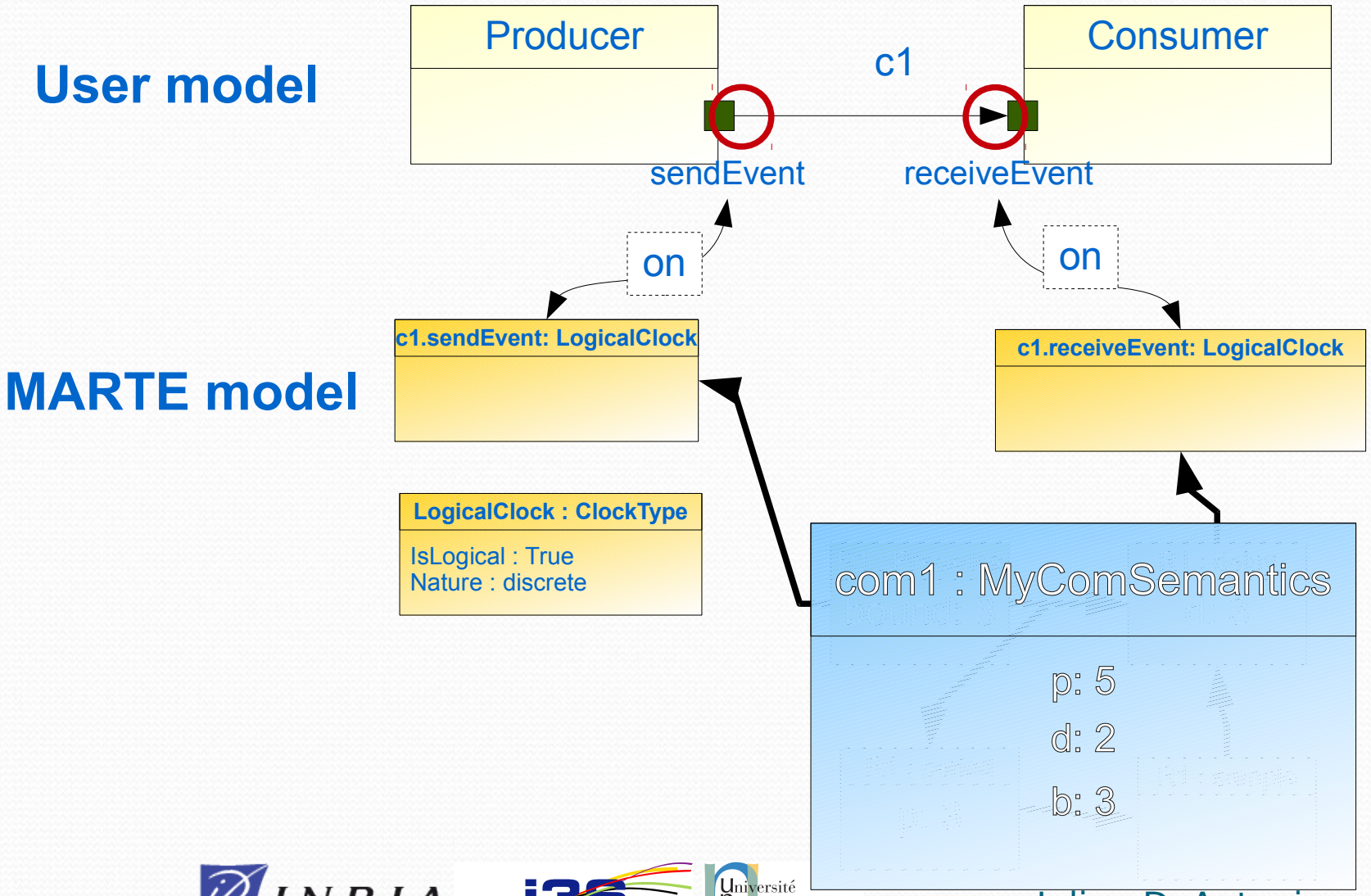
User model



MARTE model

CCSL

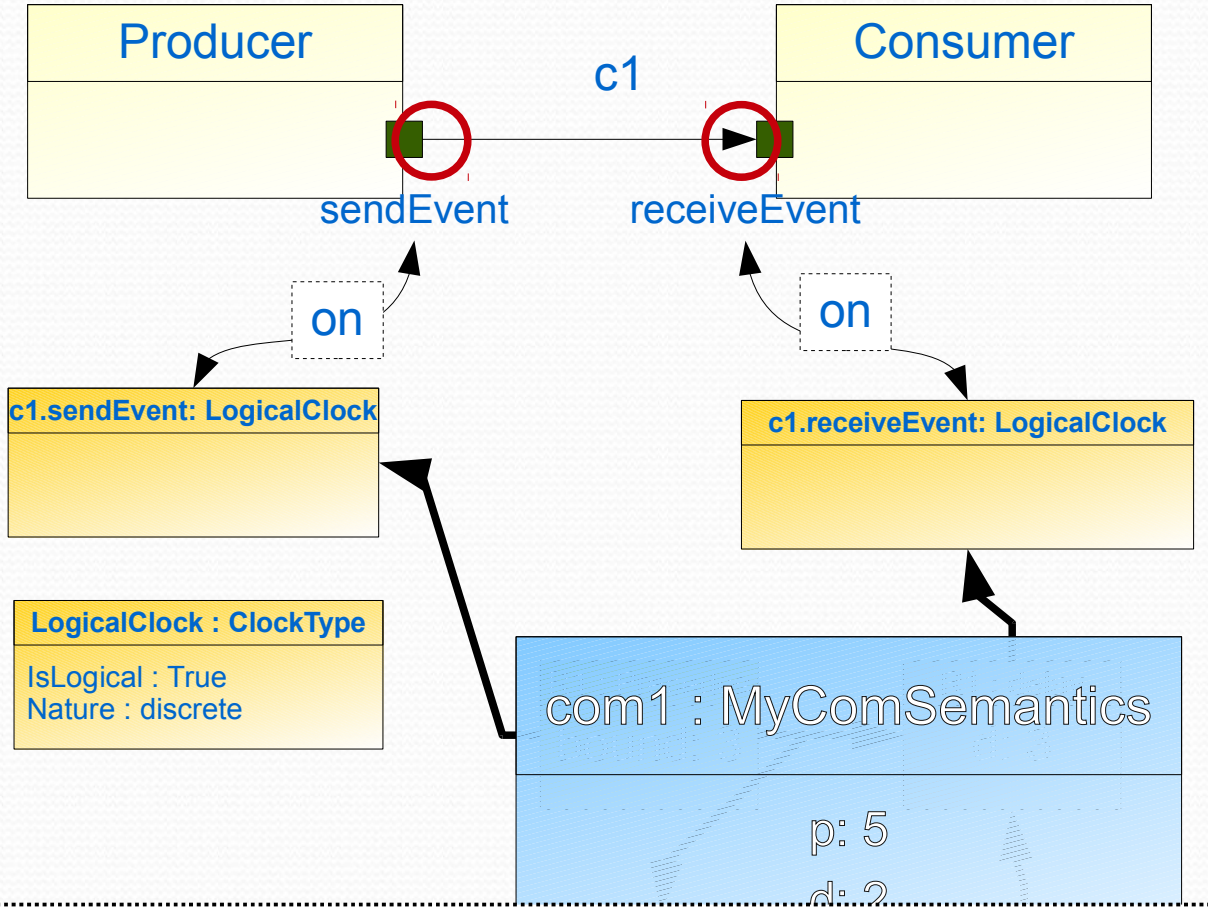
- Sketchy example of its use



CCSL

- Sketchy example of its use

User model



MARTE model

Means to define formally **timed** Models of Computations and Communications (MoCCs)

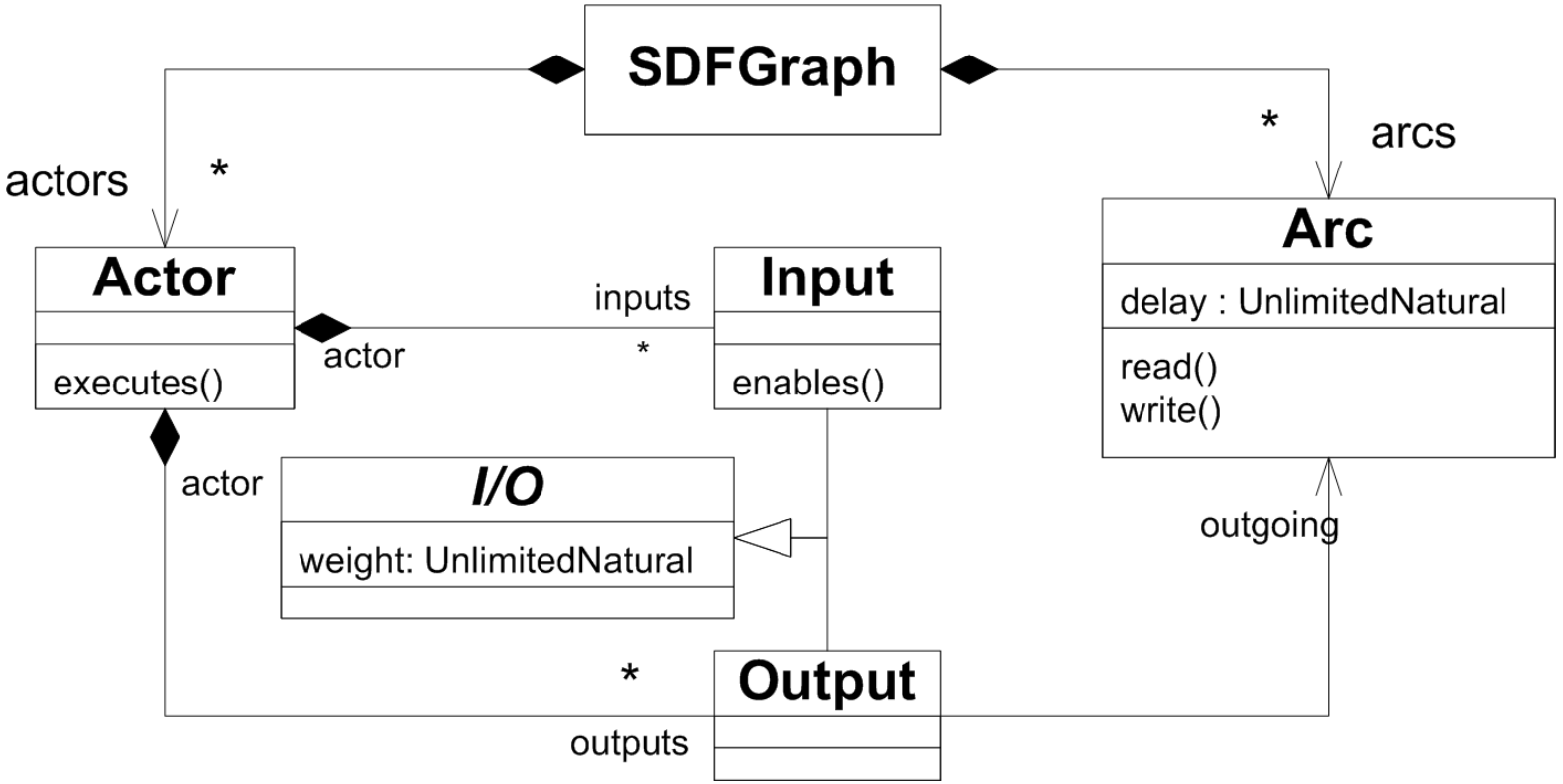
Why CCSL?

- Means to define formally **timed** Models of Computations and Communications (MoCCs)
- Notation to describe semantic relations between timed behaviors (illustrated below)
- Specification of sophisticated synchronizations
- Polychronous system modeling
- Akin to *Tagged Systems* (Lee & Sangiovanni-Vincentelli)

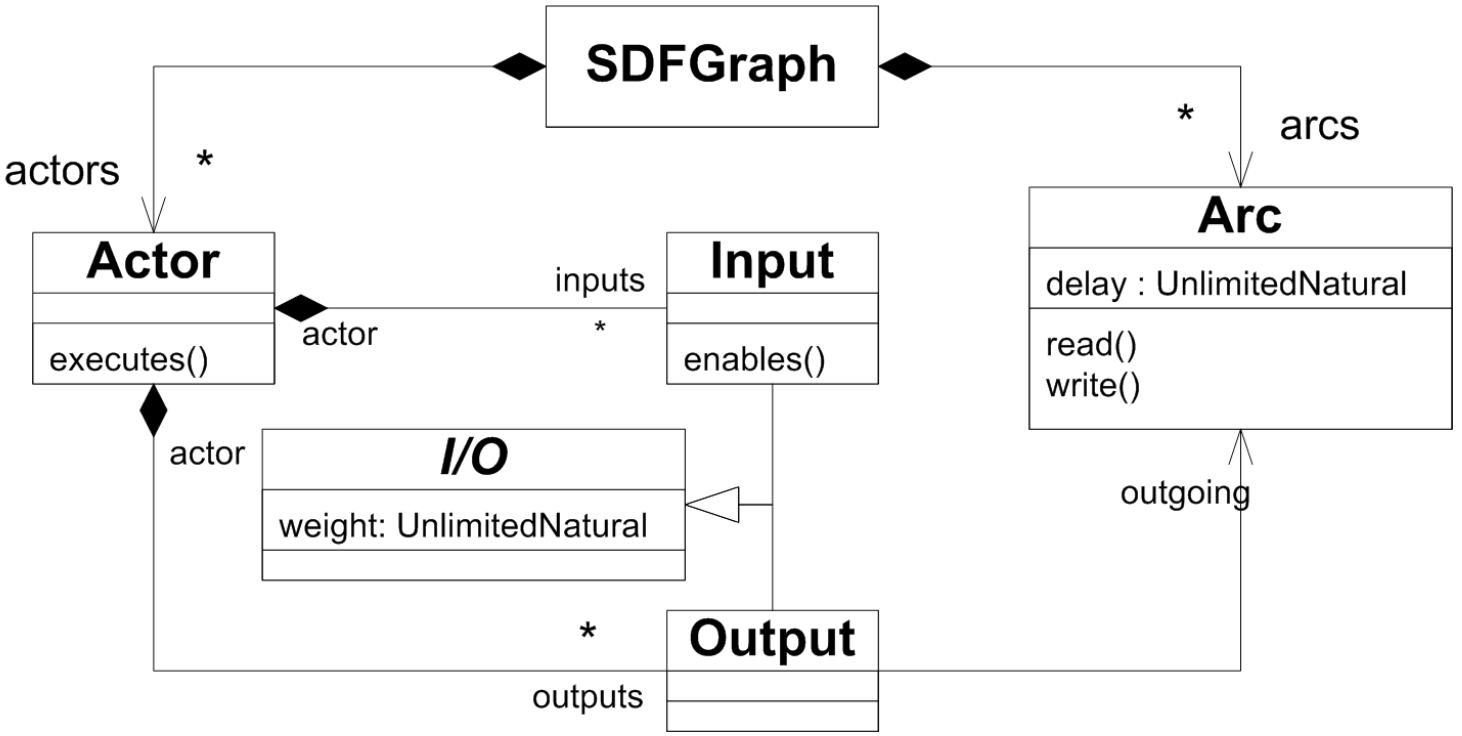
Synchronous DataFlow

- Nodes are called actors
- Arcs have a delay
- Input/Output have a weight (Number of data samples consumed/produced)

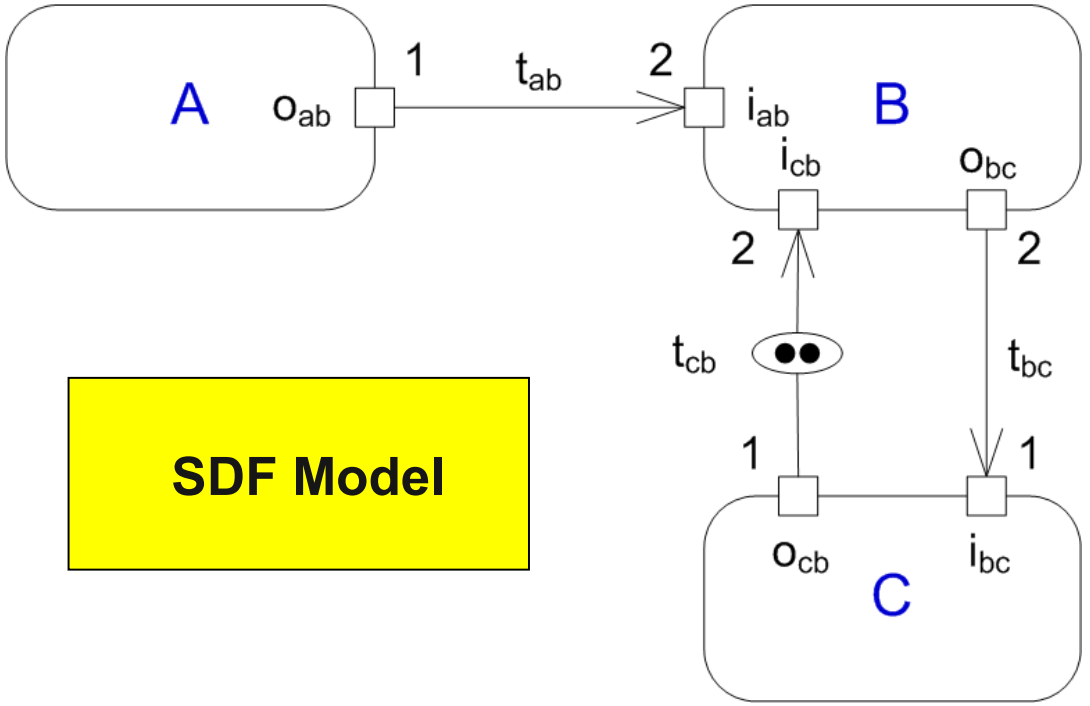
SDF Meta-model



SDF Metamodel



SDF Model



Synchronous DataFlow

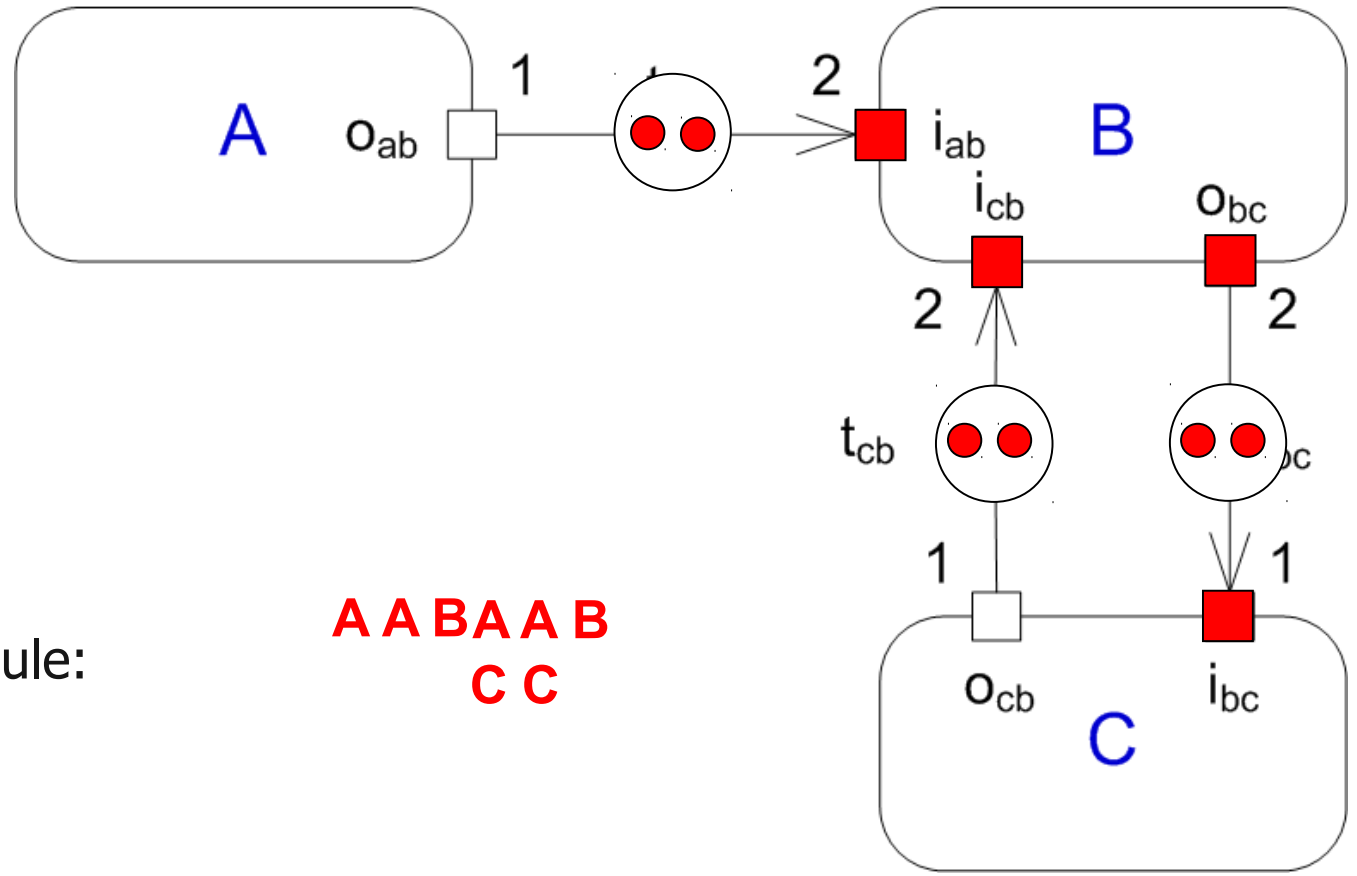
SDF firing rules:

- *Actor* enabling = each incoming arc carries at least *weight* tokens
- *Actor* execution = atomic consumption/production of tokens by an enabled actor
  - i.e., consume *weight* tokens on each incoming arcs and produce *weight* tokens on each outgoing arc
- *Delay* is an initial token load on an arc.

How can CCSL express this semantics?

SDF Example

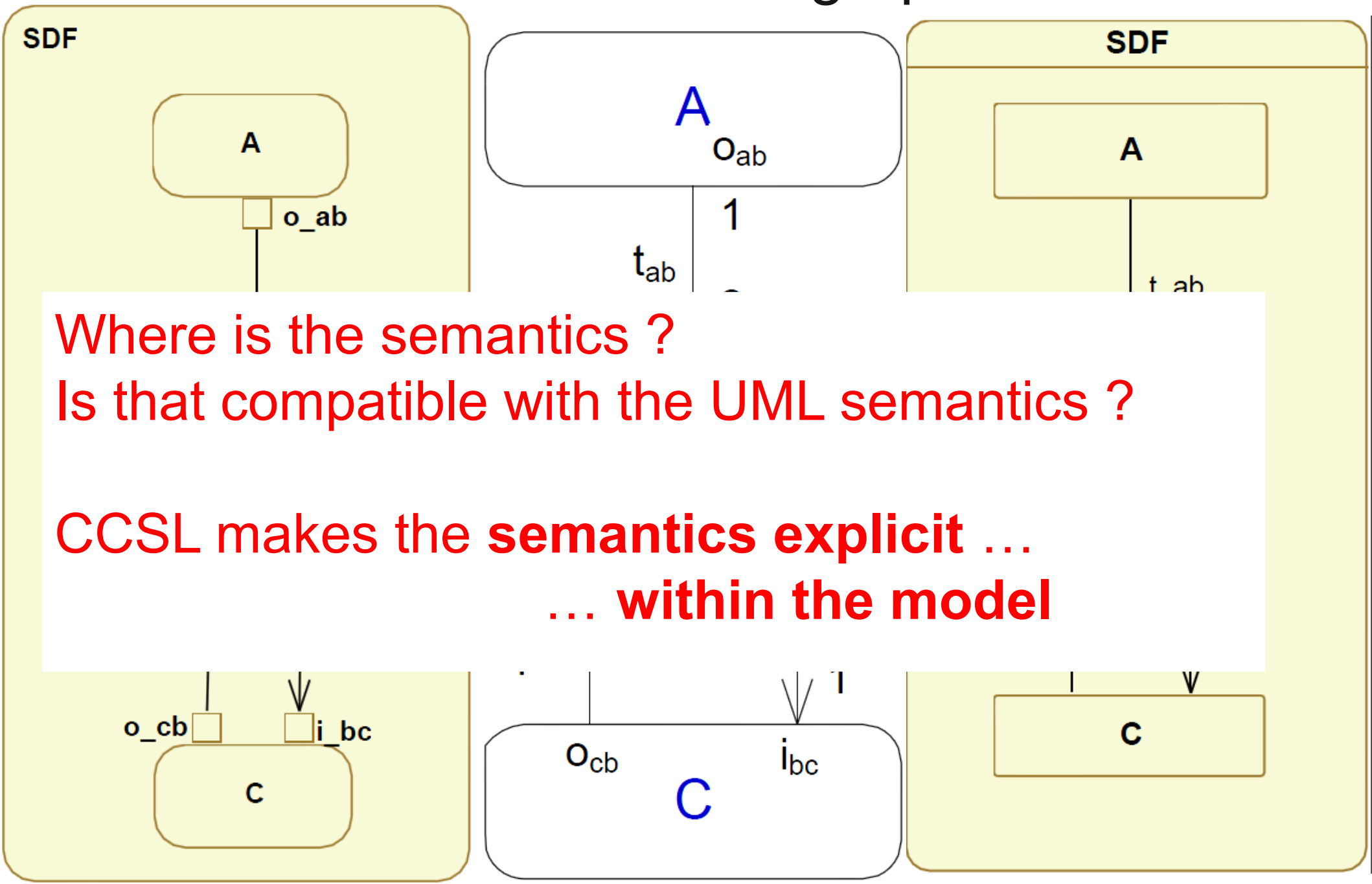
Evolutions of the model



Static schedule:

A A B A A B
C C

How to model SDF graphs in UML ?



Where is the semantics ?

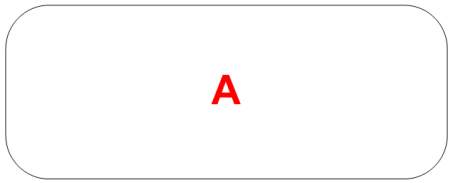
Is that compatible with the UML semantics ?

CCSL makes the **semantics explicit** …

… **within the model**

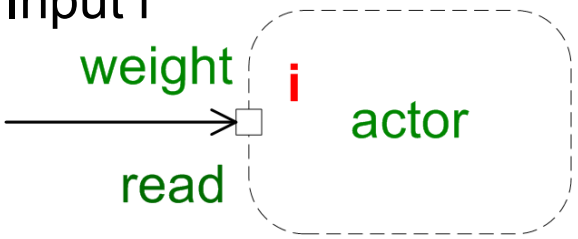
SDF semantics with CCSL (1/2)

- SDF
  - Actor A

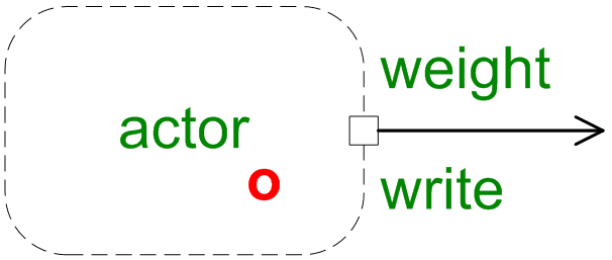


  - Token T

  - Input i



  - Output o



- CCSL

```
Clock A;

Clock write, read;
```

def $token$(clock $write$, clock $read$, int $delay$) $\triangleq$

$write \prec (read$ delayedFor $delay)$

def $input$(clock $actor$, clock $read$, int $weight$) $\triangleq$

$(read$ by $weight) \prec actor$

def $output$(clock $actor$, clock $write$, int $weight$) $\triangleq$

$actor = \left(write \text{ filteredBy } \left(1.0^{weight-1}\right)^{\omega}\right)$

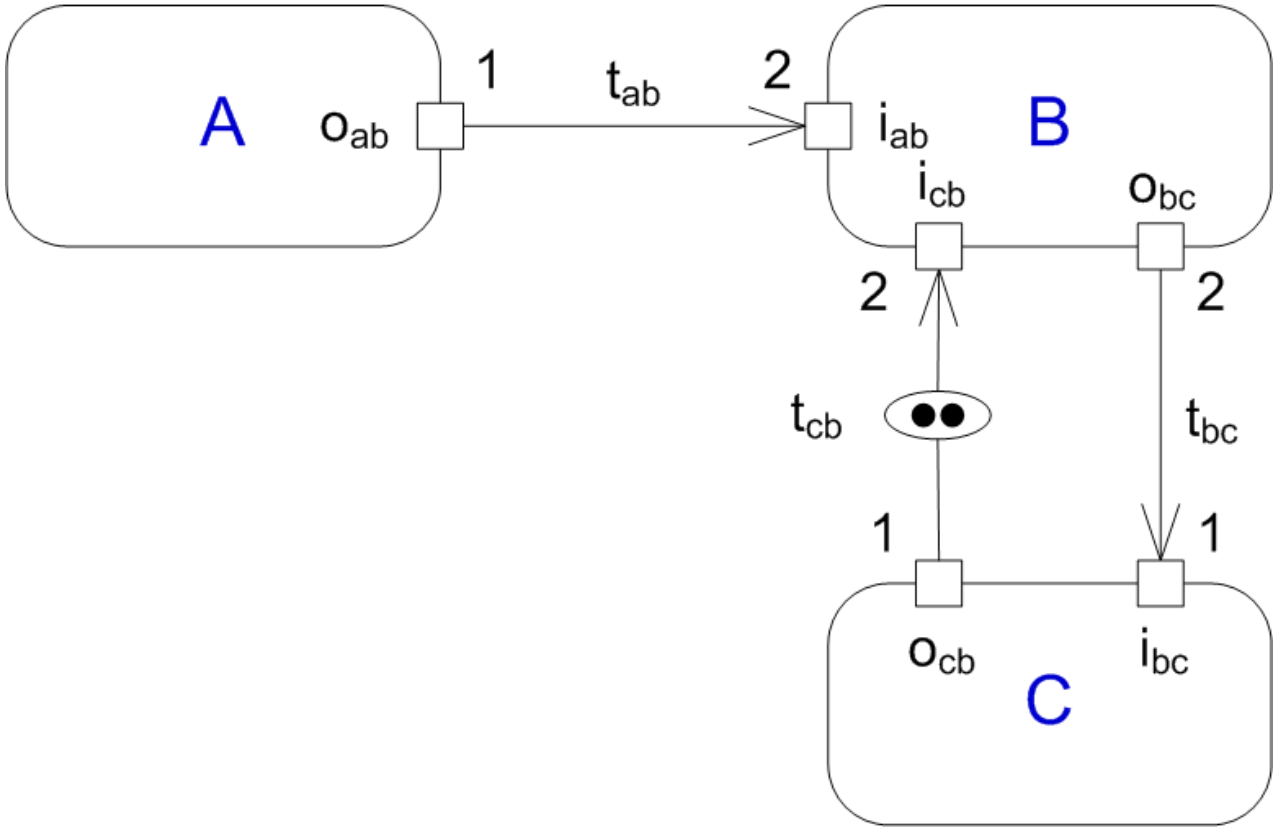
SDF semantics with CCSL (2/2)

- SDF

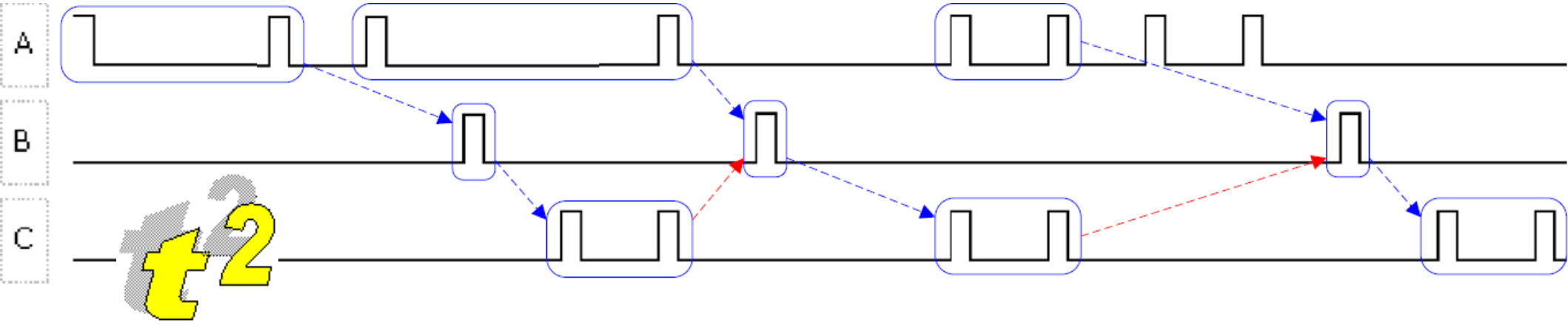
- CCSL

def arc(int $delay$, clock $source$, int out, clock $target$, int in) $\triangleq$

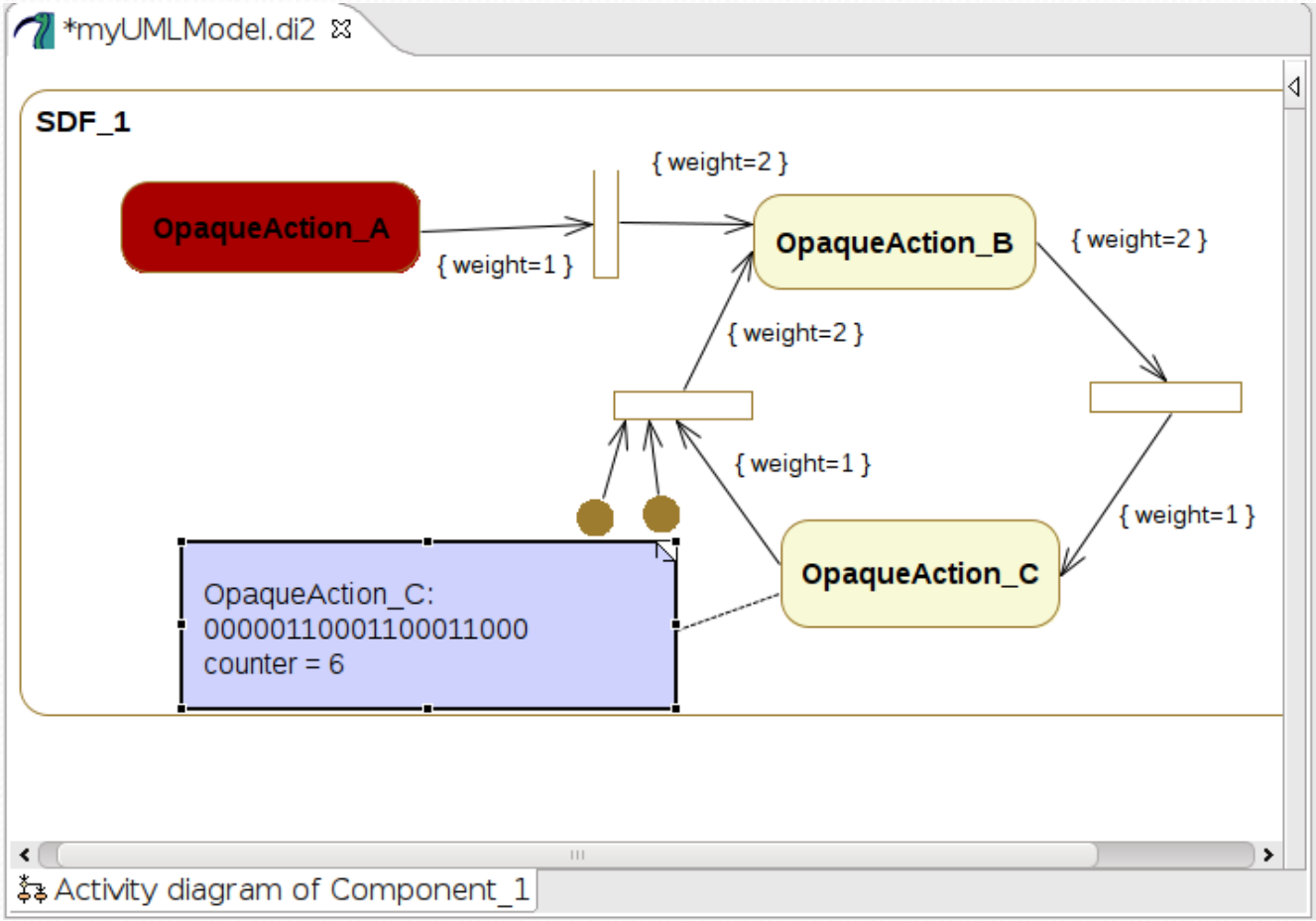
$output(source, write, out); token(write, read, delay); input(target, read, in)$

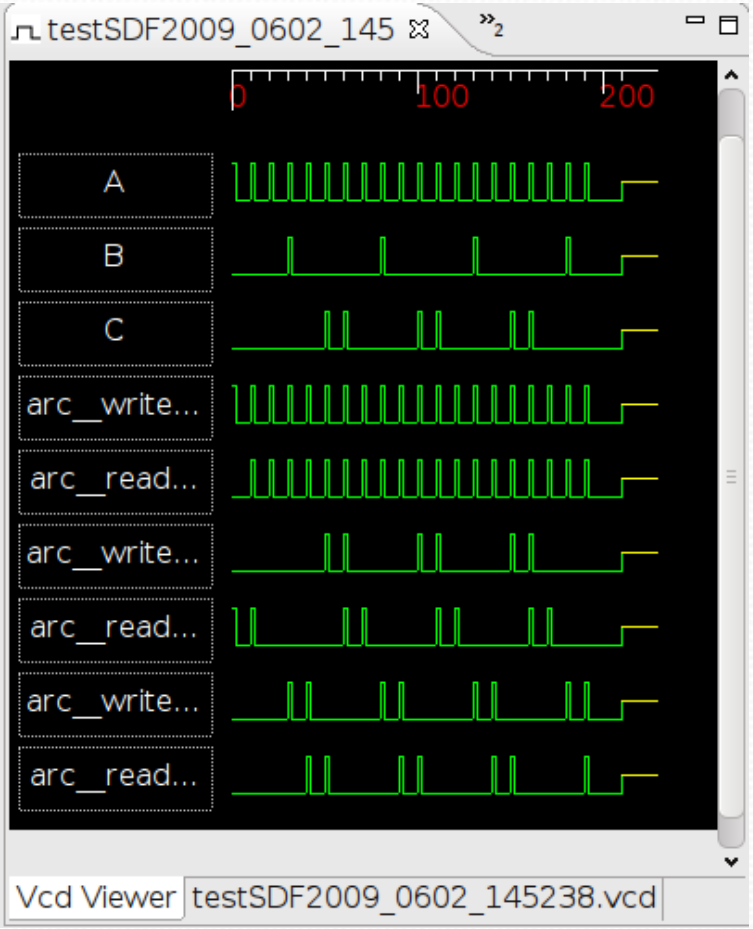


$$\mathcal{S} = arc(0, A, 1, B, 2) \mid arc(0, B, 2, C, 1) \mid arc(2, C, 1, B, 2)$$



Simulate and animate the UML/MARTE model **in TimeSquare** (http://www-sop.inria.fr/aoste/dev/time_square/)

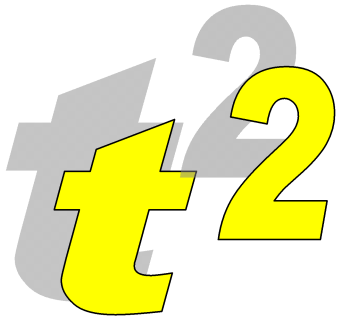




Simulate and animate the UML/MARTE model **in TimeSquare** (http://www-sop.inria.fr/aoste/dev/time_square/)

AOSTE's Tools

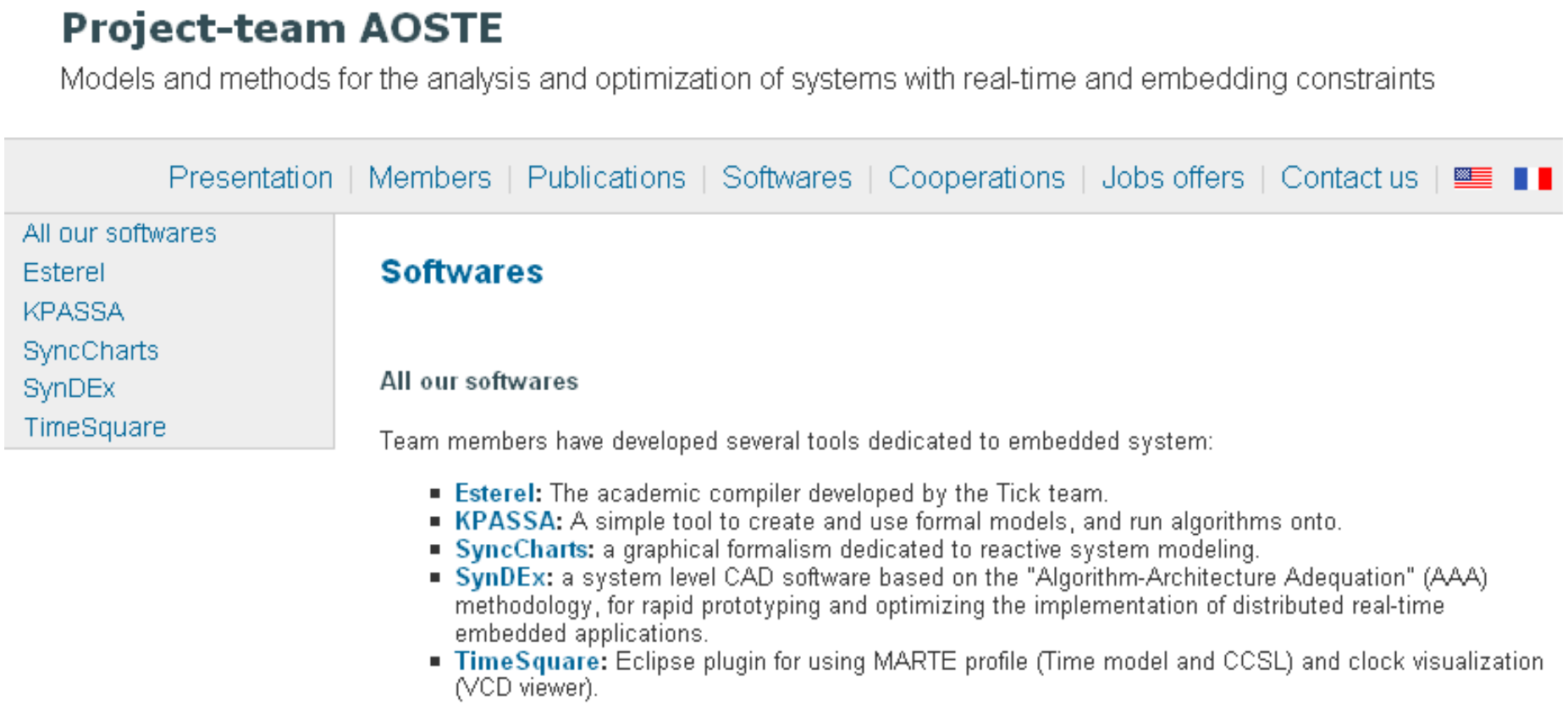
- TimeSquare
  - Software environment dedicated to the
    - Specification of CCSL constraints
    - Resolution of CCSL constraints
    - Simulation and generation of trace model
    - Animation of UML models
    - Exploration of augmented timing diagrams



- K-Passa
  - Computation of static schedules for specific MoCCs
    - Marked Graphs, Synchronous DataFlow, Latency-Insensitive Designs, K-periodical Routed Graphs
  - Analysis (deadlock freeness, safety)
  - Optimization (latency, throughput, interconnect buffer size)
  - Code generation (stand-alone simulator, co-simulation)

Tool download

- http://www-sop.inria.fr/aoste/



Project-team AOSTE

Models and methods for the analysis and optimization of systems with real-time and embedding constraints

Presentation | Members | Publications | Softwares | Cooperations | Jobs offers | Contact us

All our softwares
Esterel
KPASSA
SyncCharts
SynDEx
TimeSquare

Softwares

All our softwares

Team members have developed several tools dedicated to embedded system:

- **Esterel:** The academic compiler developed by the Tick team.
- **KPASSA:** A simple tool to create and use formal models, and run algorithms onto.
- **SyncCharts:** a graphical formalism dedicated to reactive system modeling.
- **SynDEx:** a system level CAD software based on the "Algorithm-Architecture Adequation" (AAA) methodology, for rapid prototyping and optimizing the implementation of distributed real-time embedded applications.
- **TimeSquare:** Eclipse plugin for using MARTE profile (Time model and CCSL) and clock visualization (VCD viewer).

CCSL @ work: the RT-Simex project

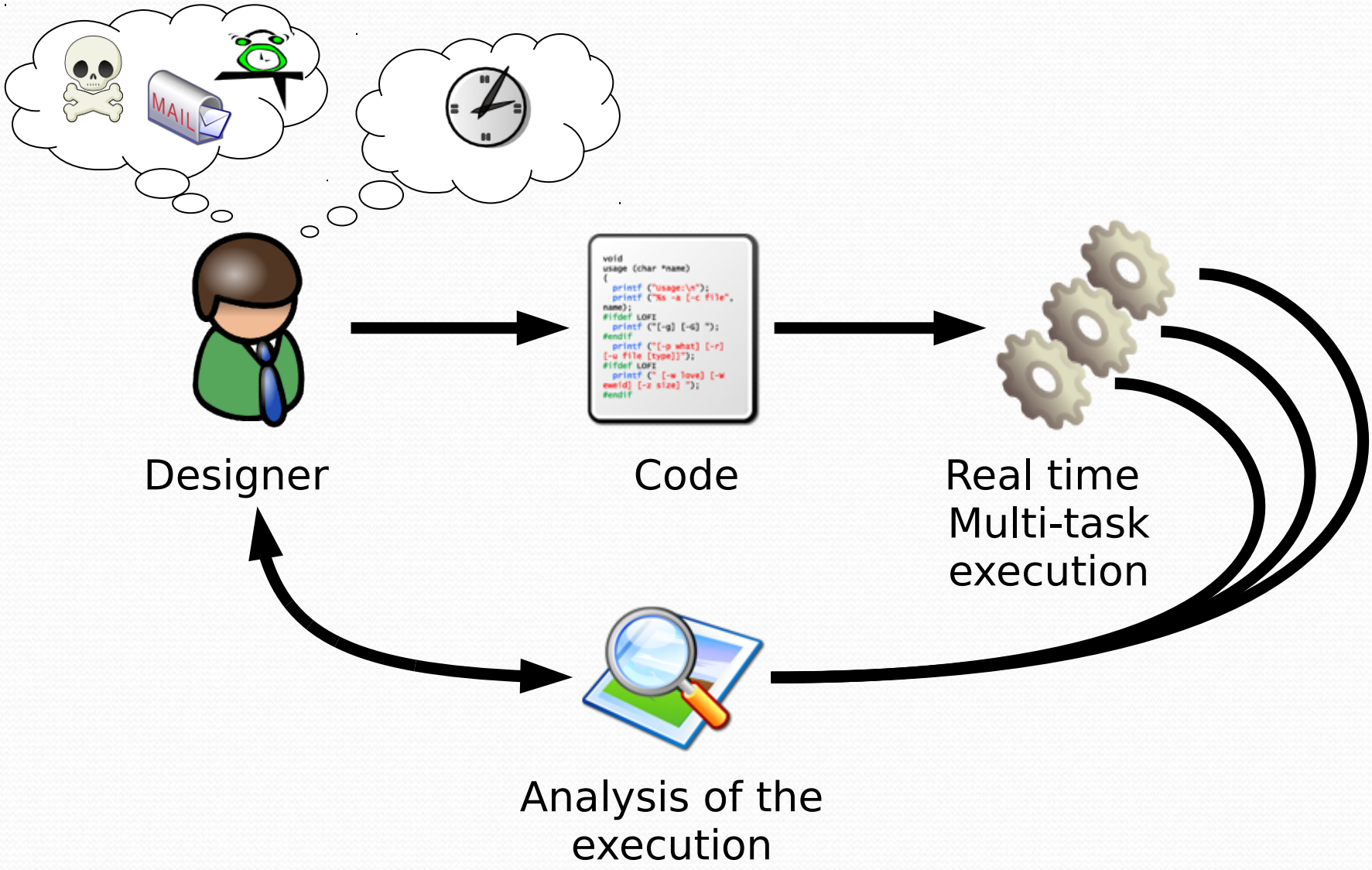
AOSTE sophia
I3S/UNS/INRIA

Frédéric Thomas, Étienne Juliot
Obeo

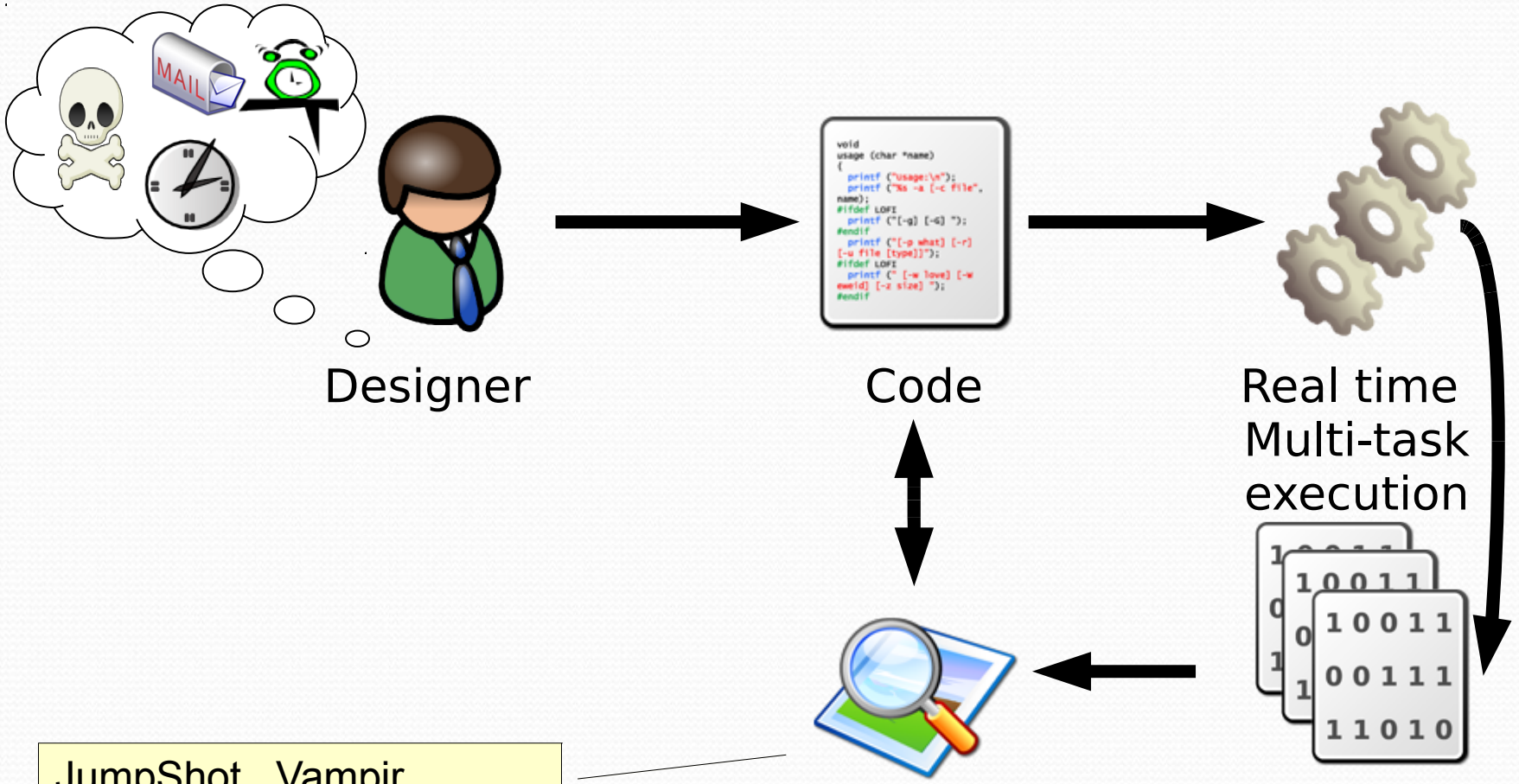
Jean-Philippe Babau
UBO
and more...

Retro-ingénierie de Traces d'analyse de SIMulation et d'EXécution de systèmes temps-réel

Context : help the designer



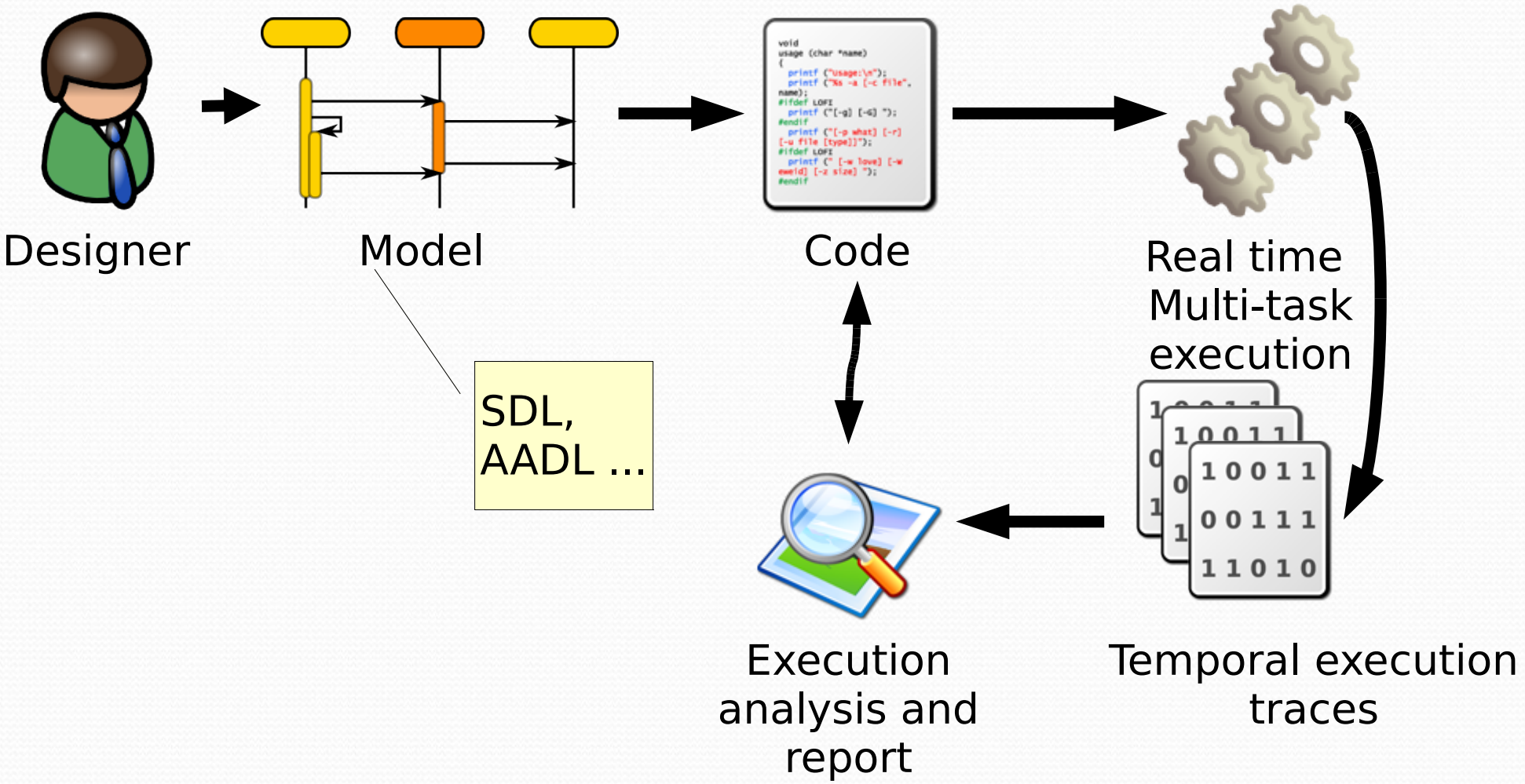
Context: execution traces



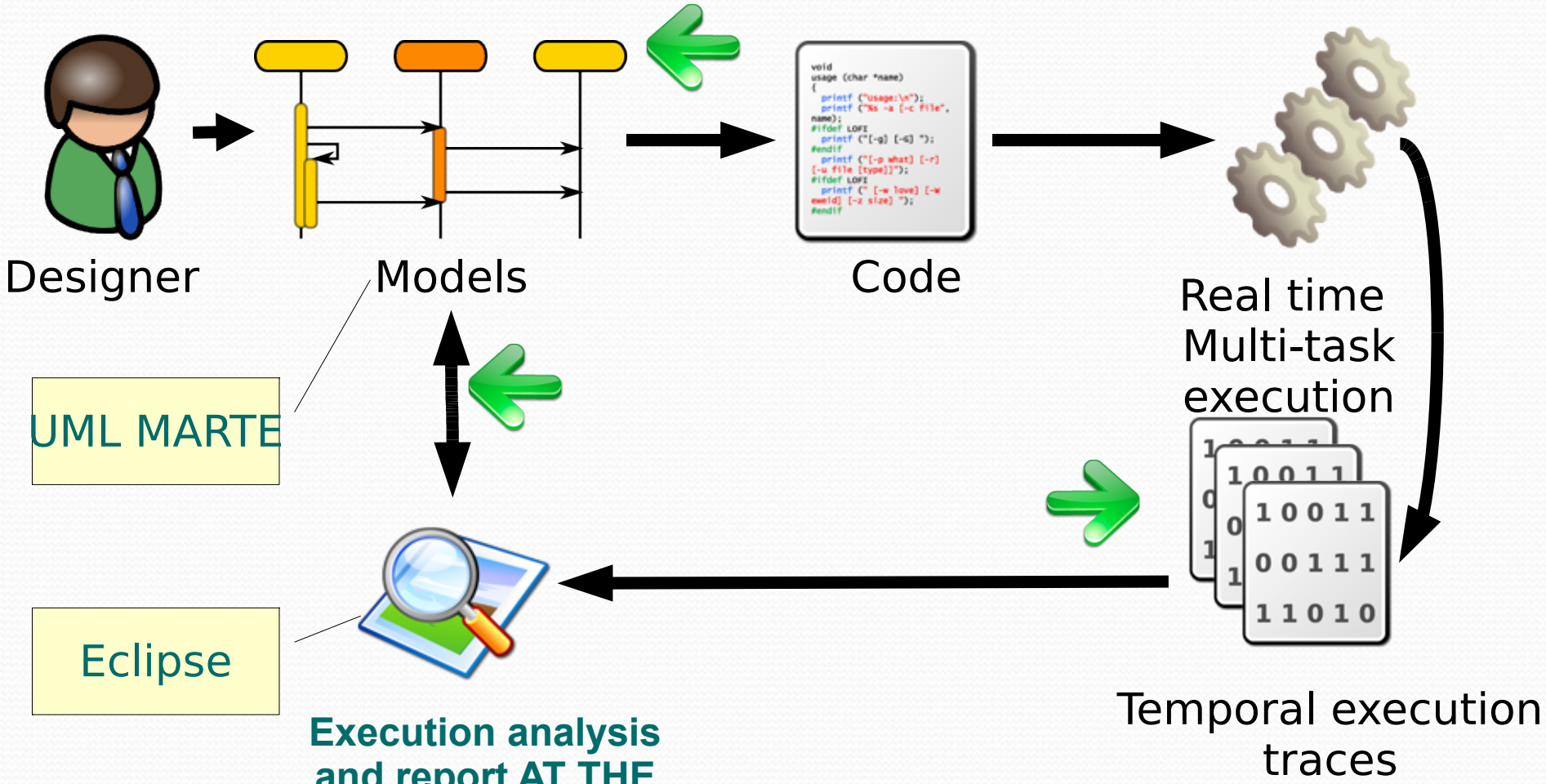
JumpShot , Vampir,
Linux Tool Eclipse Project

OTF, VCD

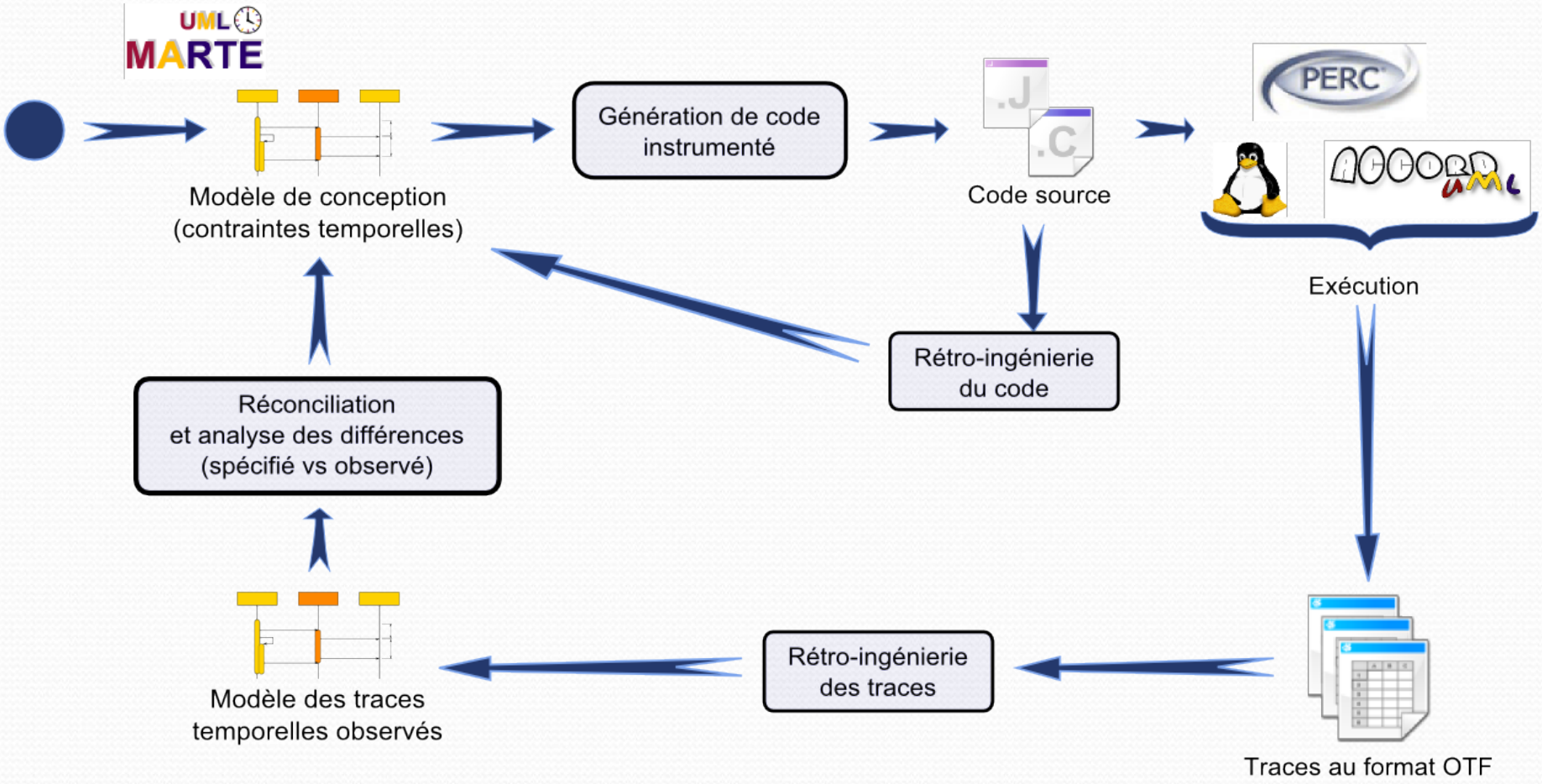
observations



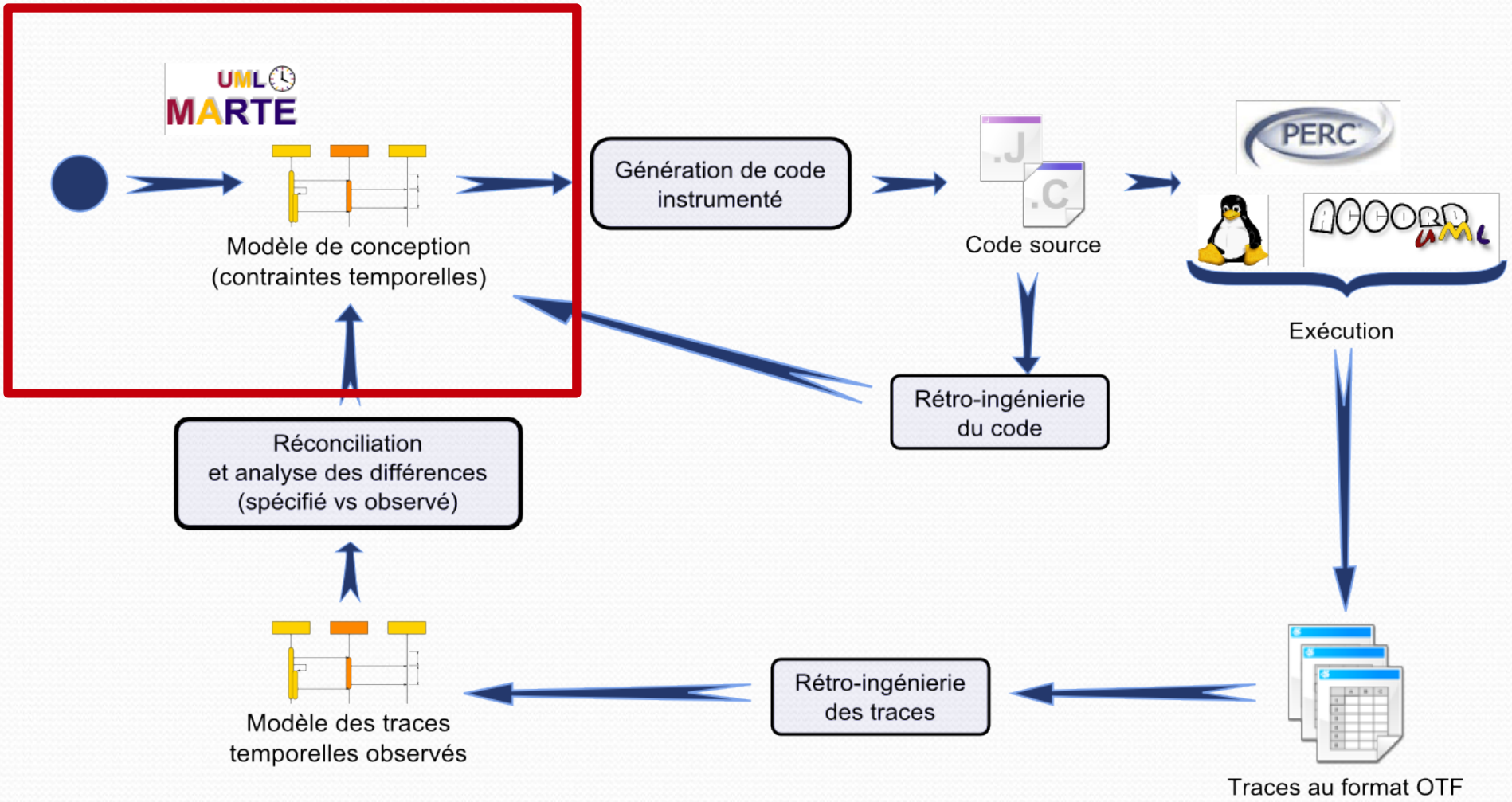
RT-Simex objectives



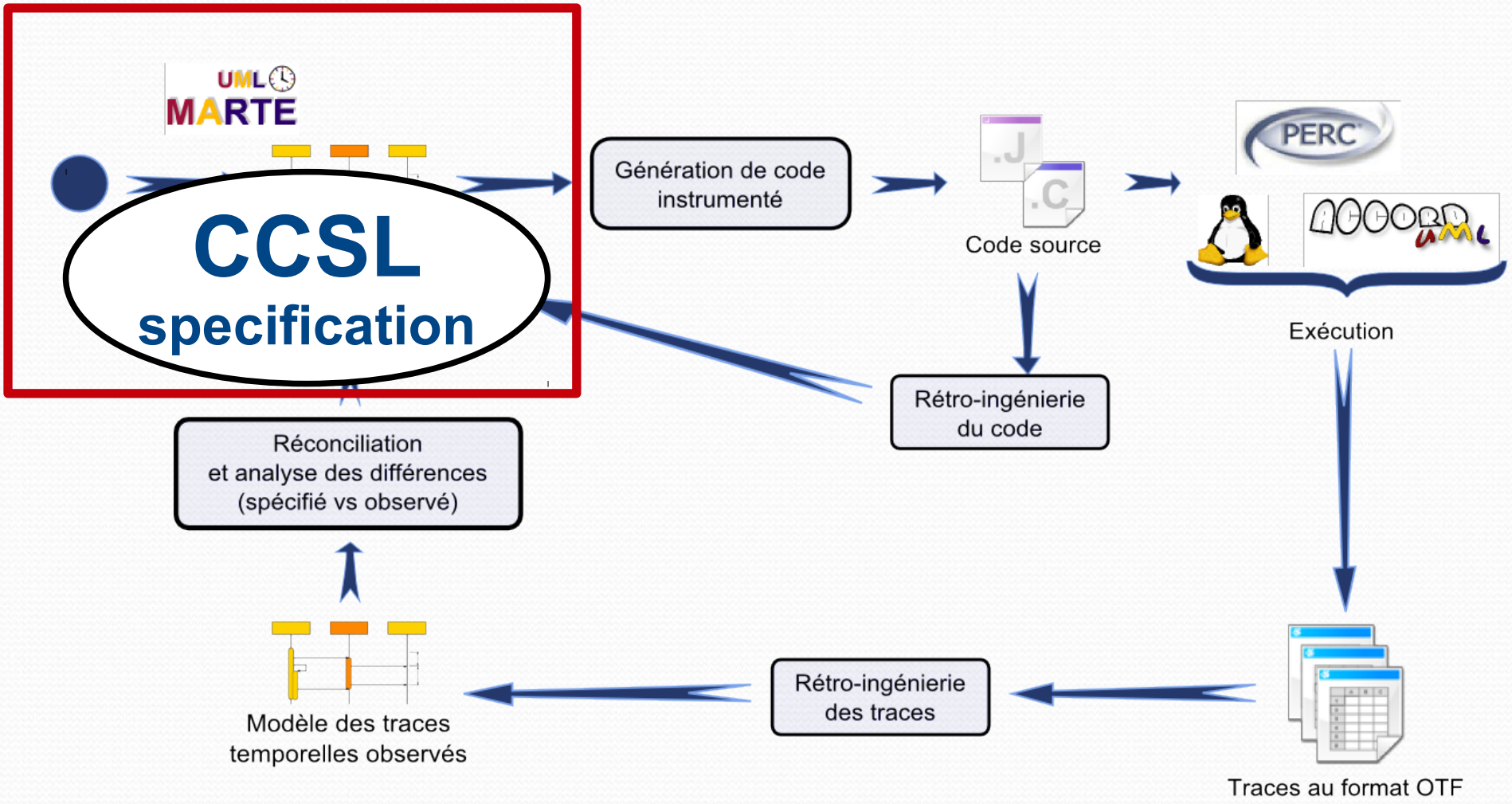
Processus RT-Simex



Processus RT-Simex



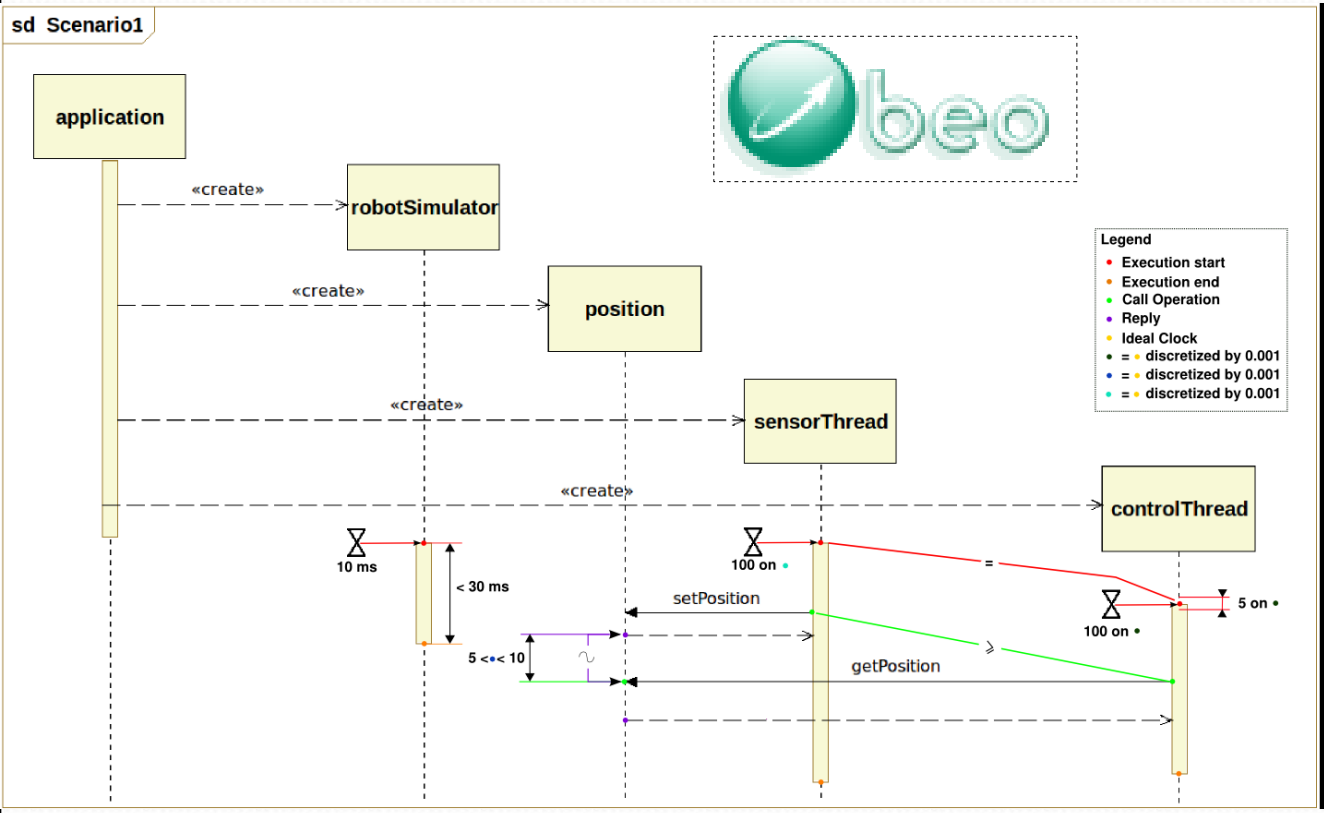
Processus RT-Simex

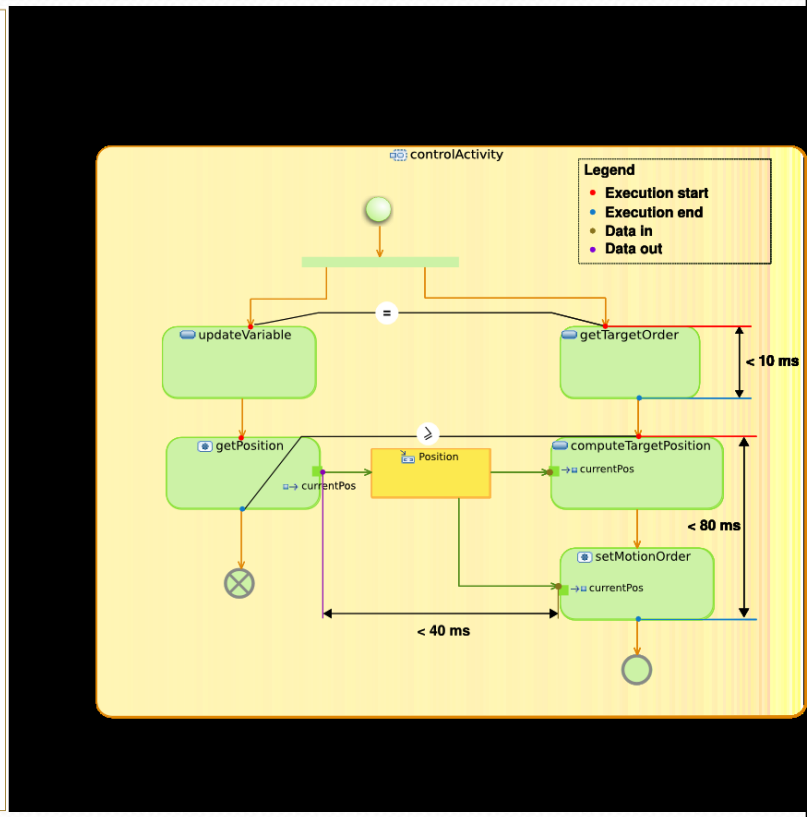


CCSL (**C**lock **C**onstraint **S**pecification **L**anguage)

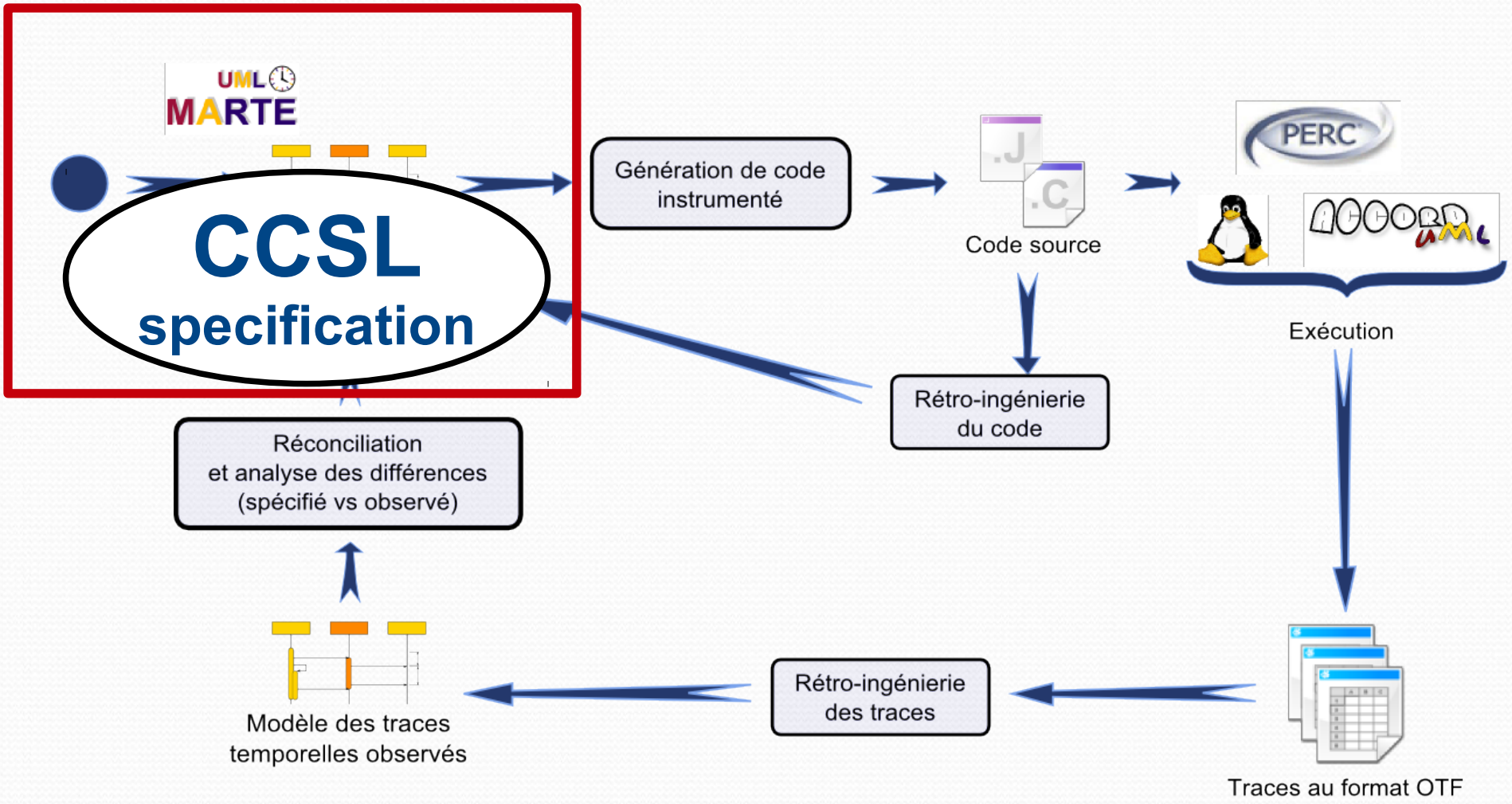
- Relations: dependencies between clocks
  - Coincidence → **=**
  - Exclusion → **#**
  - Precedence → **<**
  - Alternance → **~**
- Expressions: a mean to create new clocks from others
  - Delay → **delayedFor** *X* **on** *aClock*
  - Filtering → *aClock* **filteredBy** *aBinaryWord*
  - Union → *aClock* **union** *anotherClock*
  - Intersection → *aClock* **inter** *anotherClock*
  - Periodicity → **periodicOn** aClock **period** X **offset** Y
- Libraries: user-defined relations and expressions for a specific domain

Graphical formal annotation over a UML model **for RT-Simex**…

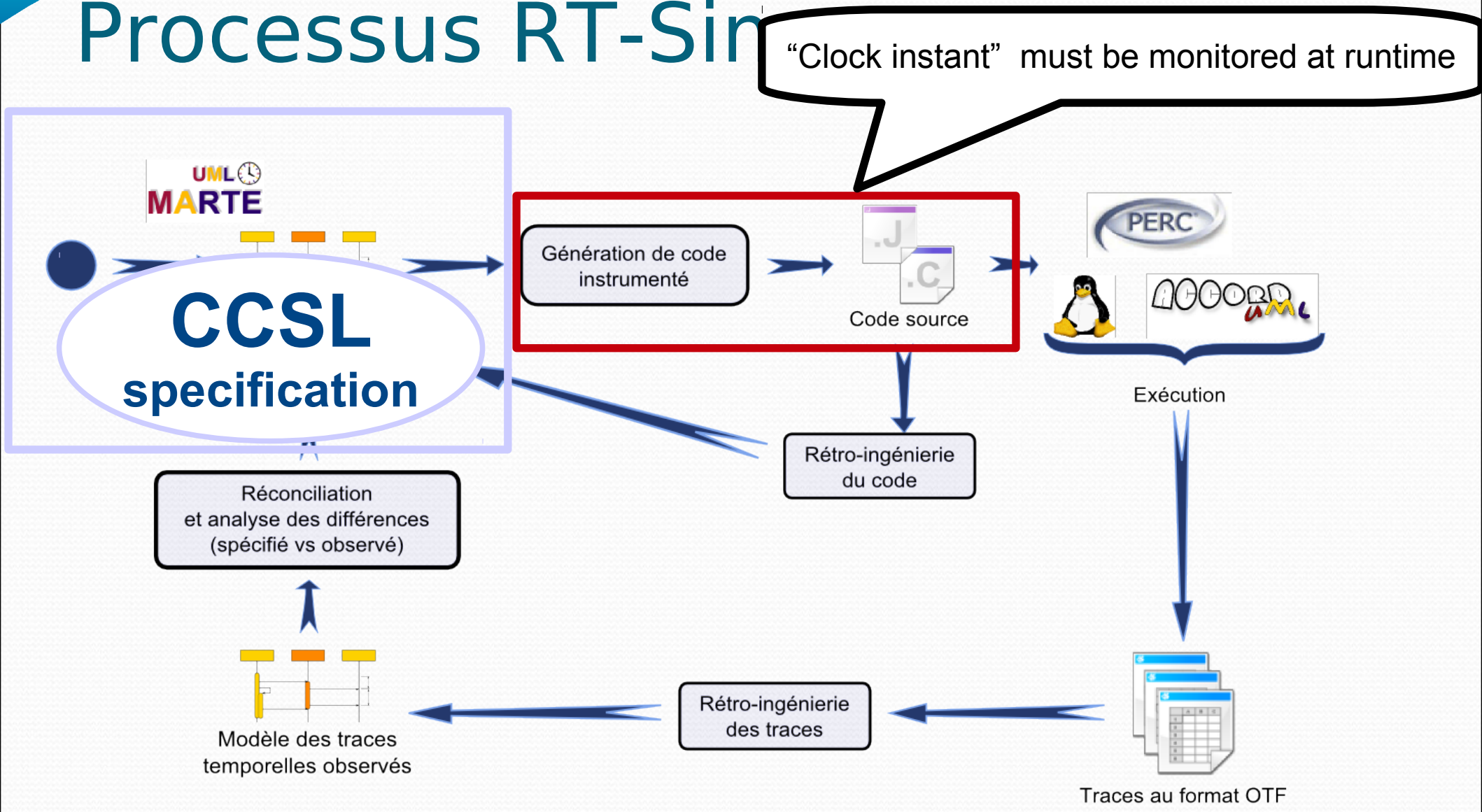




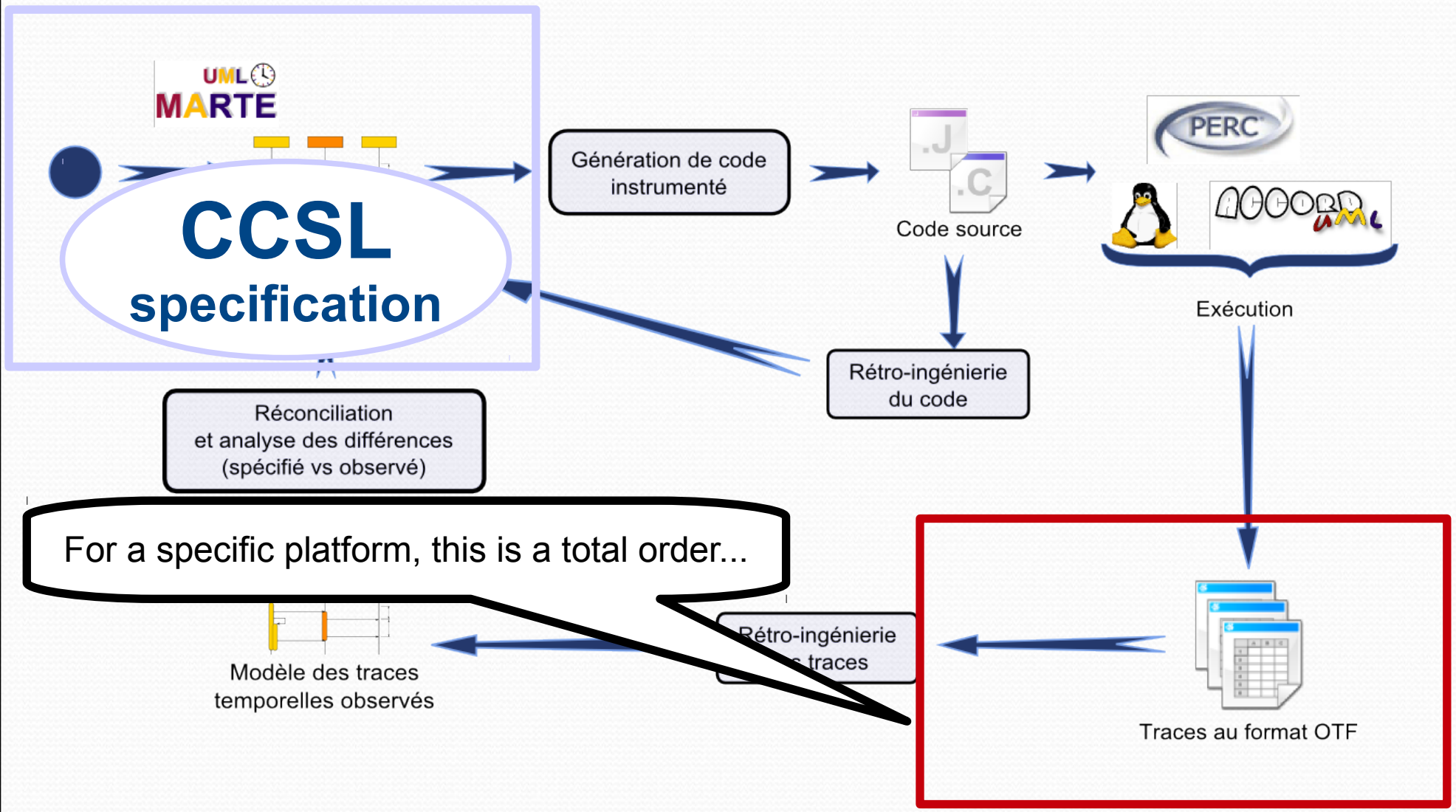
Processus RT-Simex



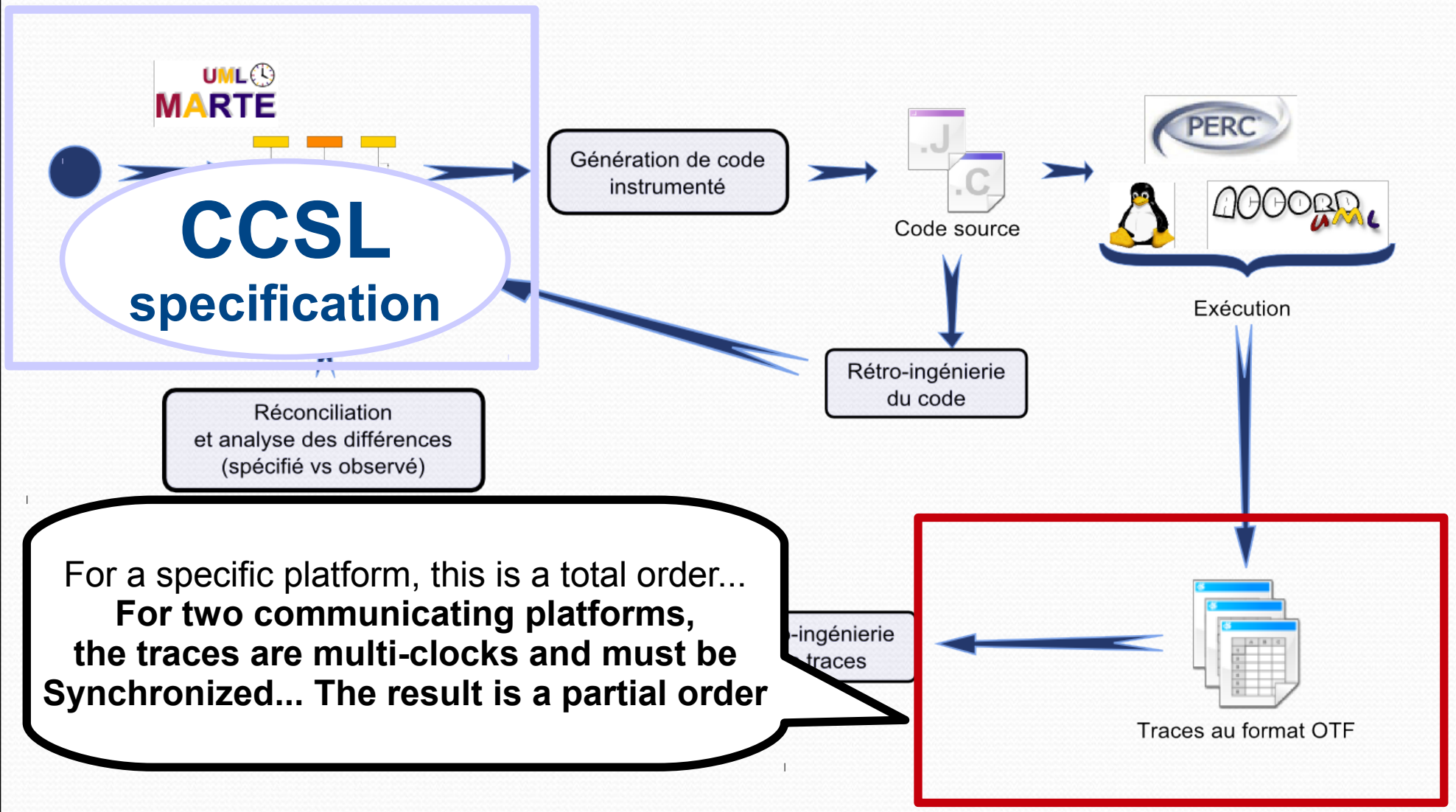
Processus RT-Sim



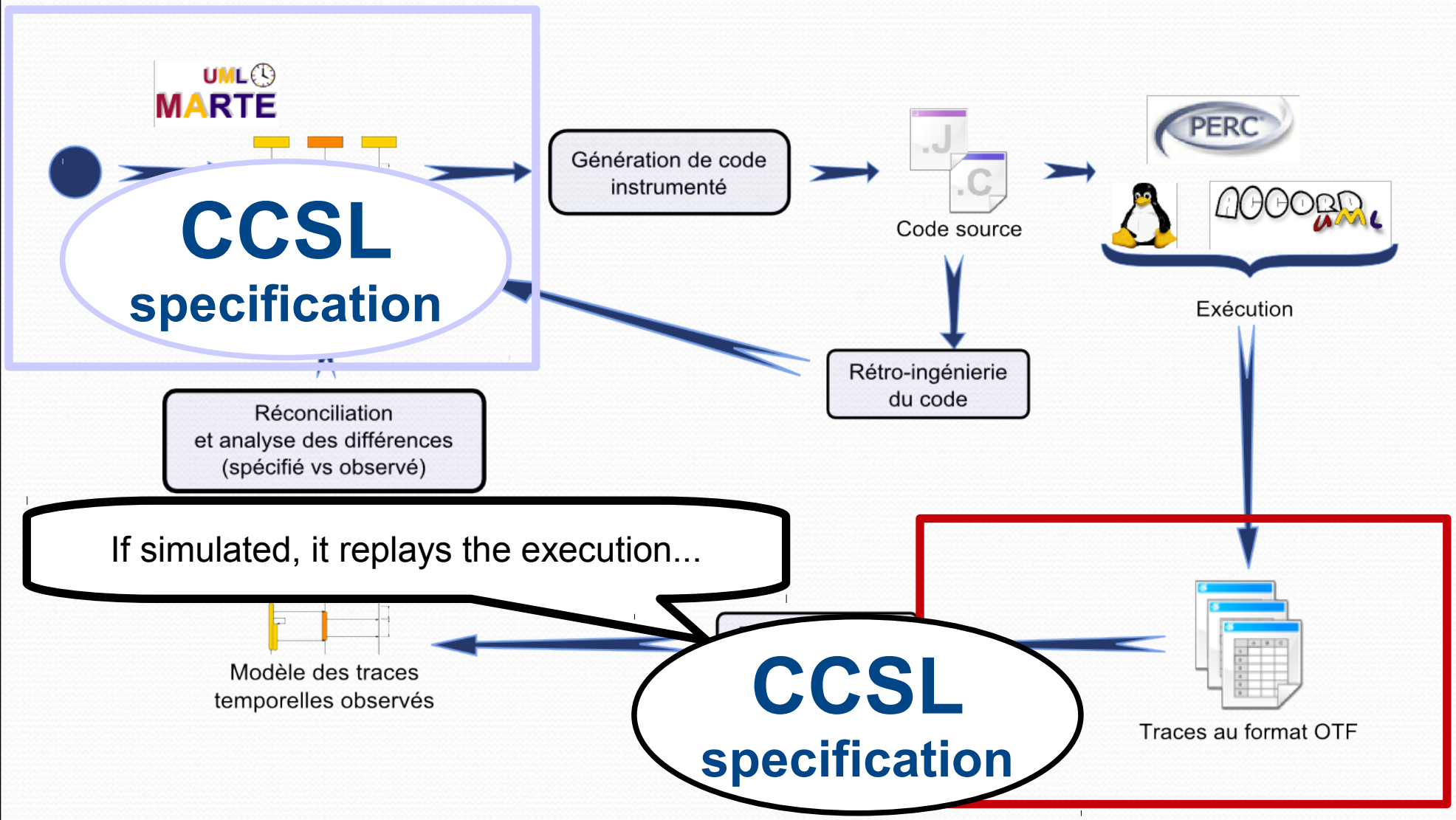
Processus RT-Simex



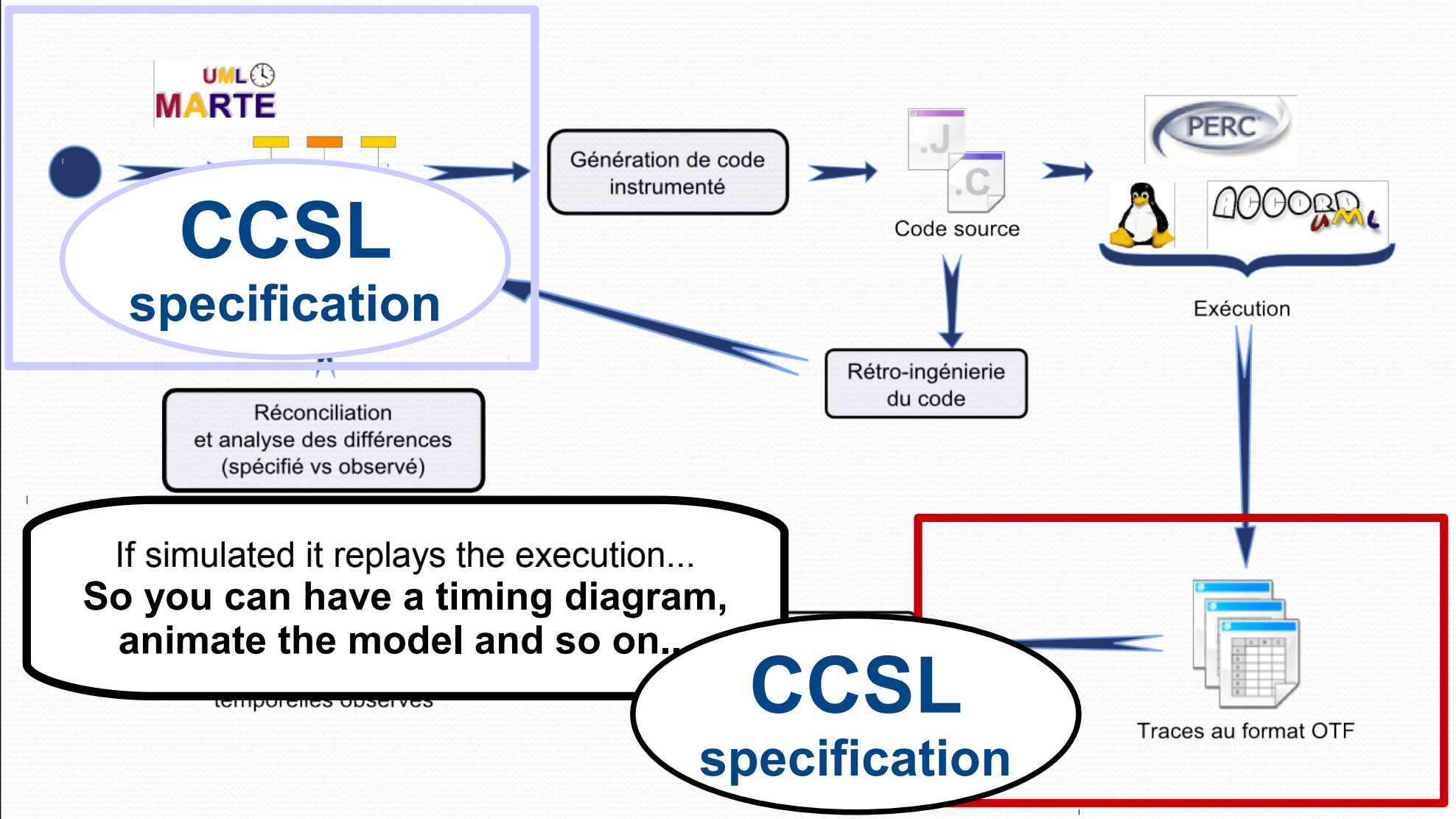
Processus RT-Simex



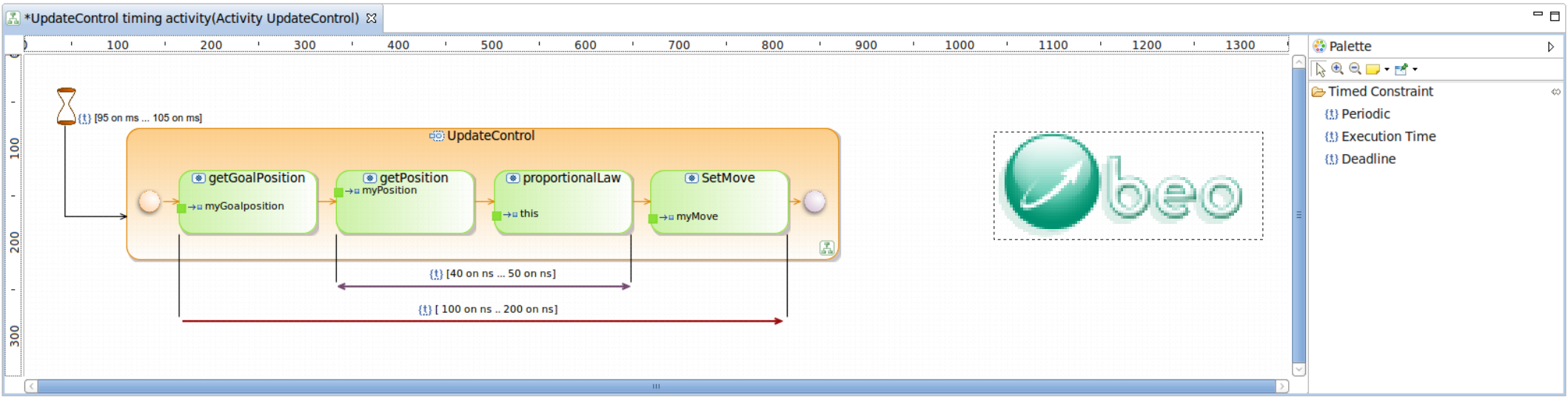
Processus RT-Simex

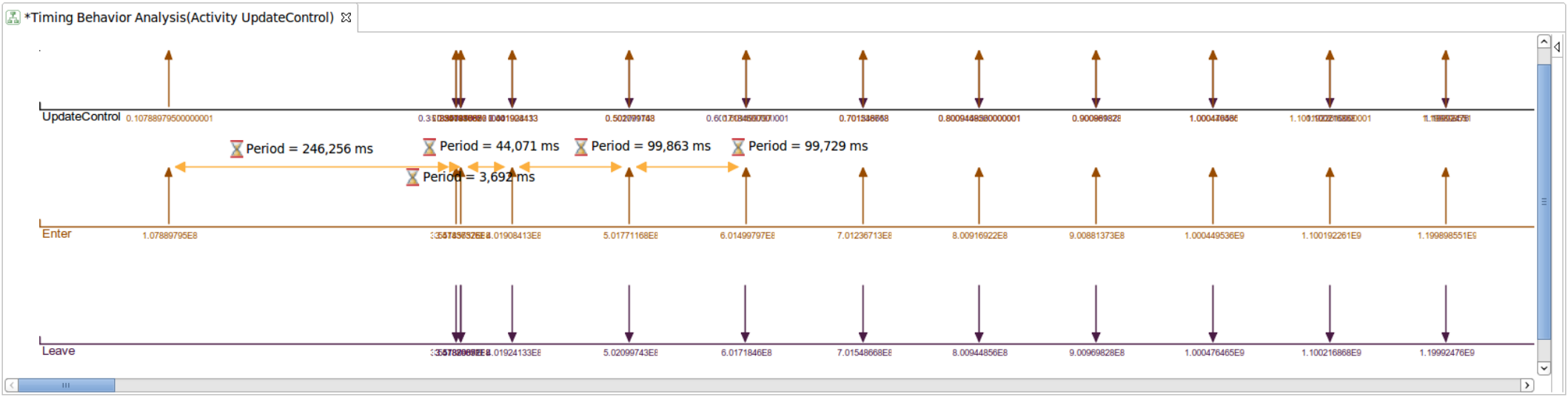


Processus RT-Simex

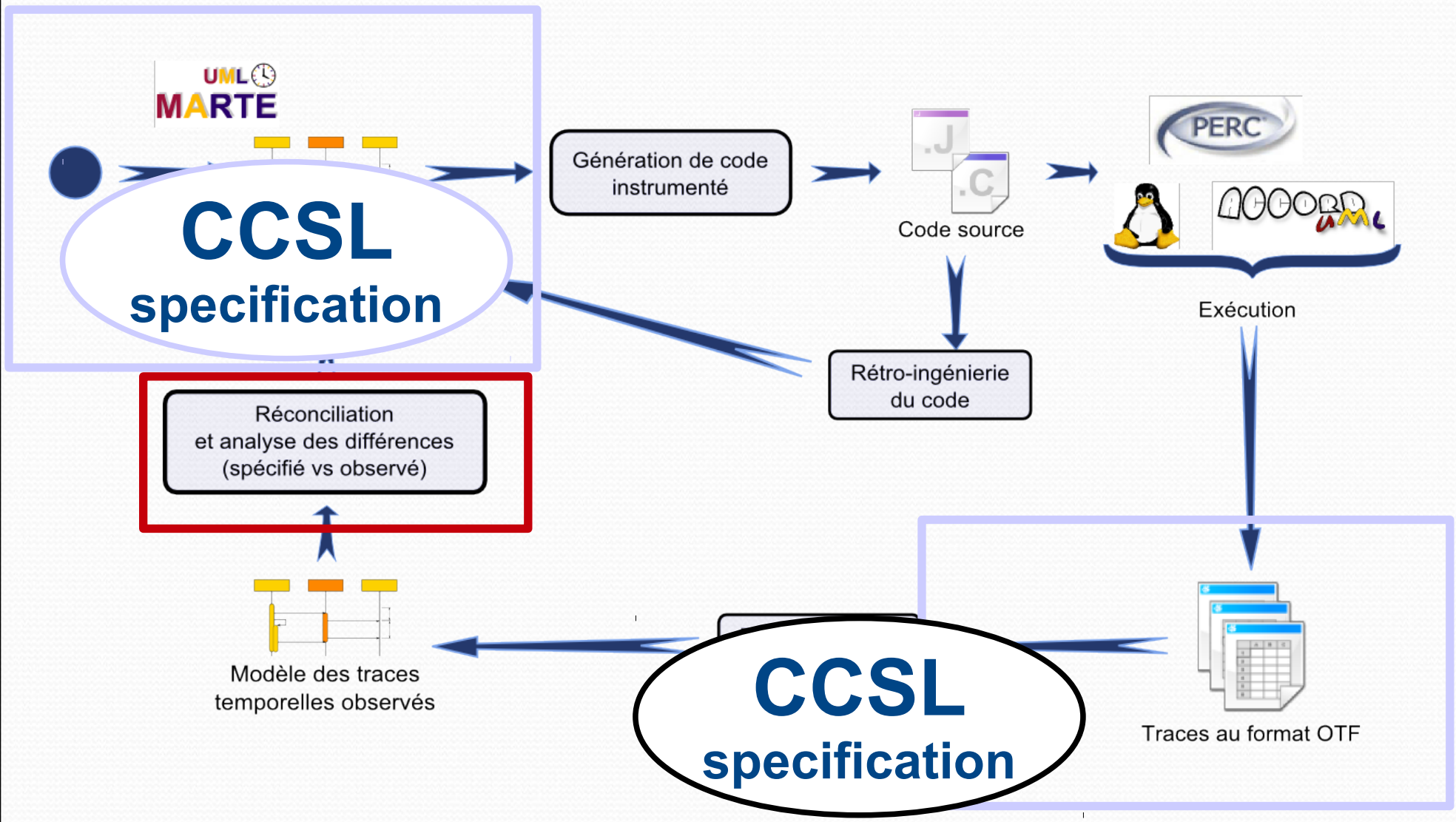


A specification and the actual execution

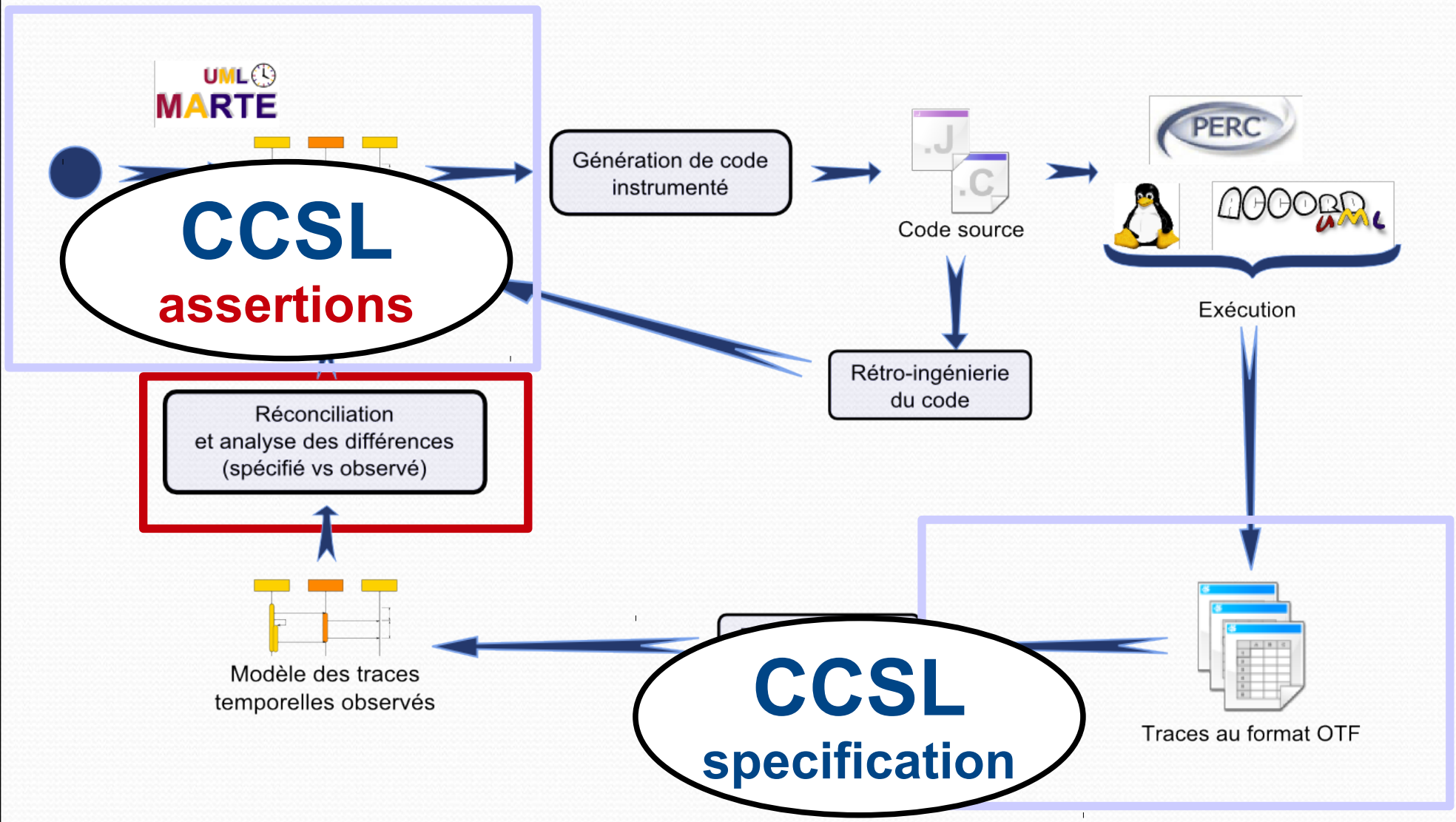




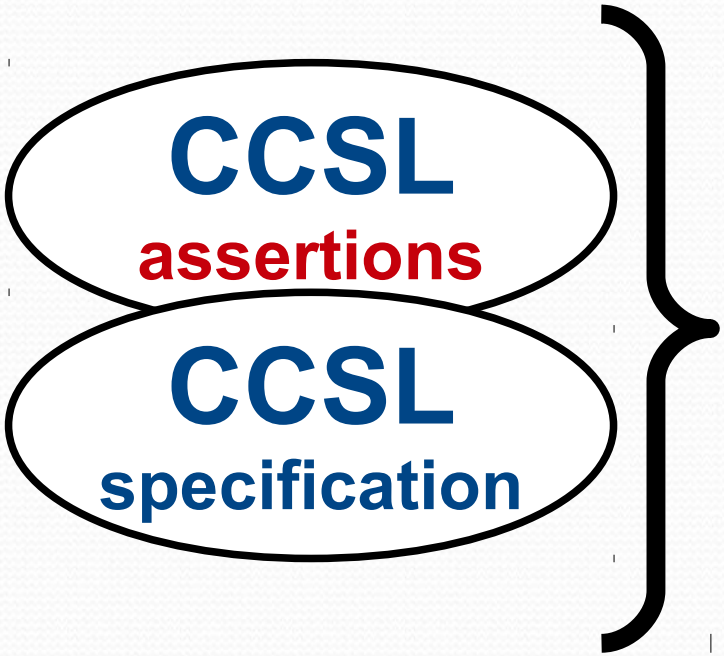
Processus RT-Simex



Processus RT-Simex



Processus RT-Simex



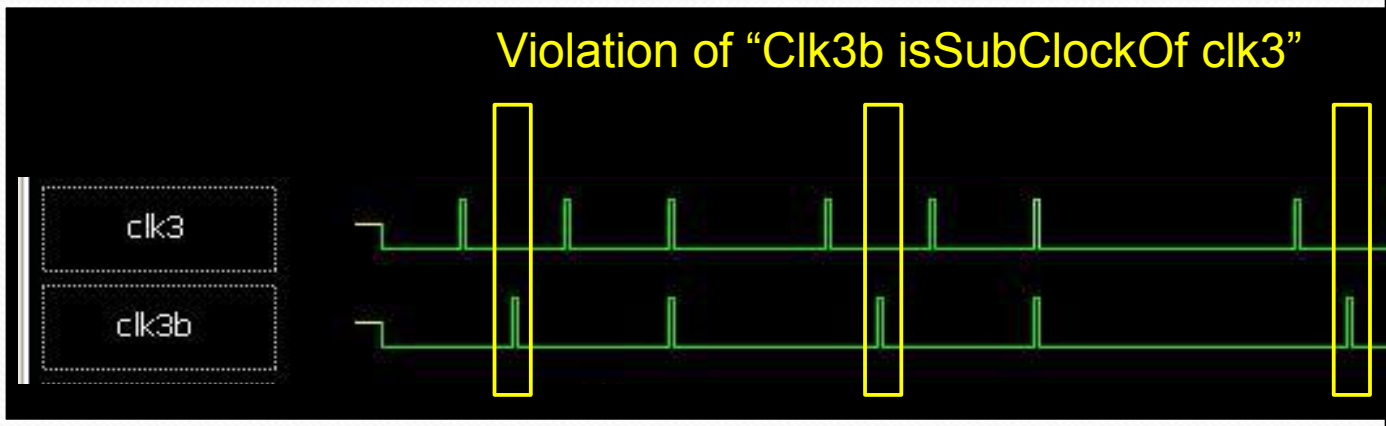
You can simulation both together to see if a violation occurs

Processus RT-Simex

You can simulation both together to see if a violation occurs

CCSL
assertions

CCSL
specification



Conclusion

Two uses of CCSL:

➔A formal language to specify MoCC explicitly within a model,

- MOCC library construction,
- No hidden rules or implementation
- Directly linked to the model

➔A formal language to specify real-time properties AND to check the correctness of a specific implementation

- Possibility to specify construct for a specifict domain,
- Possibility to change the rolle of a CCSL model, from specification, to assertion.

Some future ongoing works

- Heterogenenous systems
  - Clocks and CCSL are a way to explicitly glue various MoCC
- Use of CCSL for Power Domain

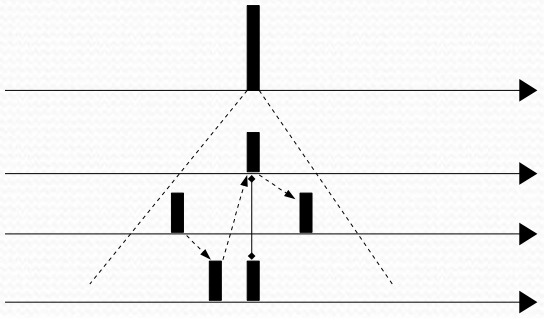
- Add of user defined simulation policy
  - EDF, RMA and all you can imagine (use of back-end)
- Optimization of the TimeSquare solver
  - Optimize BDD depending on clock domains
- Instant refinement

- ...

Some future ongoing works

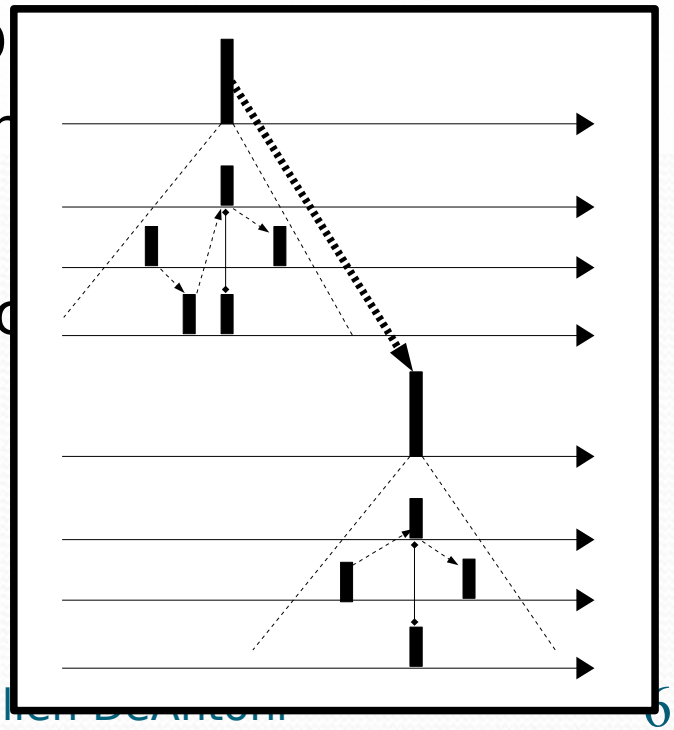
- Heterogenenous systems
  - Clocks and CCSL are a way to explicitly glue various MoCC
- Use of CCSL for Power Domain

- Add of user defined simulation policy
  - EDF, RMA and all you can imagine (use of back-end)
- Optimization of the TimeSquare solver
  - Optimize BDD depending on clock domains
- Instant refinement
- …



Some future ongoing works

- Heterogenenous systems
  - Clocks and CCSL are a way to explicitly glue various MoCC
- Use of CCSL for Power Domain
- Add of user defined simulation p
  - EDF, RMA and all you can imagi
- Optimization of the TimeSquare
  - Optimize BDD depending on cloc
- Instant refinement
- …



Thank you !