Logical Time at Work:

Capturing Data Dependencies and Platform Constraints

Calin Glitia Julien DeAntoni Frédéric Mallet

INRIA Sophia Antipolis Méditerranée

Université Nice Sophia Antipolis

Synchron 2010

Seventeenth Seminar on Synchronous languages, modeling and programming

Fréjus, France, November 29th - December 3rd, 2010

Semantics as *constrained logical time*

(Modeling) languages are defined by:

- syntax
- (behavioral) semantics

Semantics should be explicit

Clock Constraint Specification Language (CCSL)

- Define the semantics of syntactical models
- Makes the syntactical models executable

Outline

Semantics of data-flow formalisms: Synchronous Data-Flow (SDF), Multidimensional-SDF

- The description of all the possible system execution schedules
- Functional description: internal constraints = data dependencies
- Time refinement – enforce the constraints
- Environment/execution platform constraints

Outline

Semantics of data-flow formalisms: Synchronous Data-Flow (SDF), Multidimensional-SDF

- The description of all the possible system execution schedules
- Functional description: internal constraints = data dependencies
- Time refinement – enforce the constraints
- Environment/execution platform constraints

1. Define explicit semantics for SDF models
2. Translate data-dependencies to execution dependencies
3. Multi-dimensional semantics (MDSDF)
4. External constraints

Clock Constraint Specification Language

Modeling and Analysis of Real-Time and Embedded systems (MARTE)

- Companion of the **Time** Package
- Chronological relations between events
- **Clocks** = possibly infinite and possibly dense totally ordered sets of instants

CCSL relations:

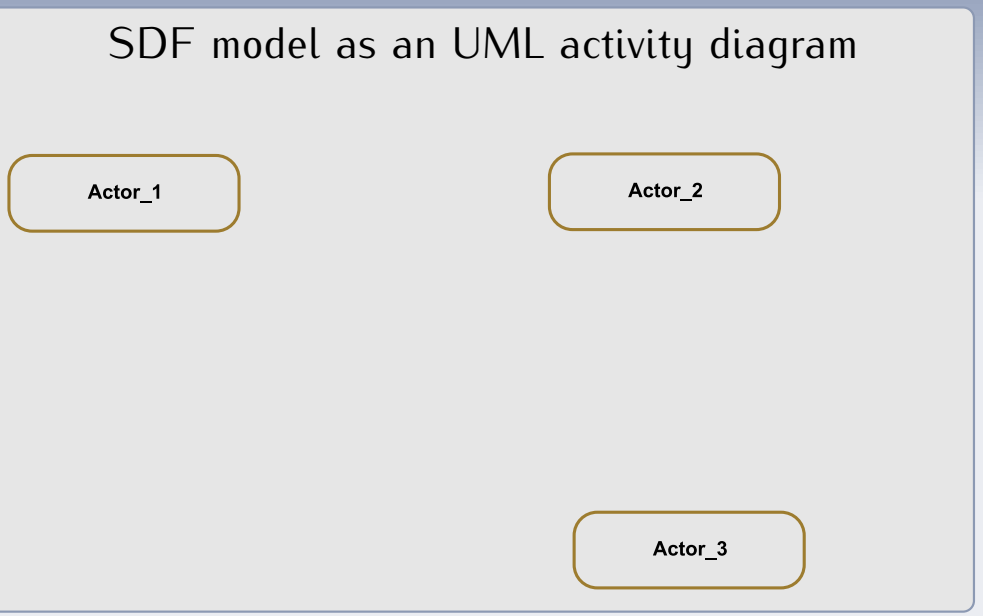
- **precedence** ($\prec$)
- **coincidence** ($\equiv$)
- **exclusion** (#)

CCSL clock expressions:

- **filteredBy** ($\blacktriangledown$) – by a binary periodic word
- **delay** ($) – by an integer value

Data-flow models

SDF model as an UML activity diagram

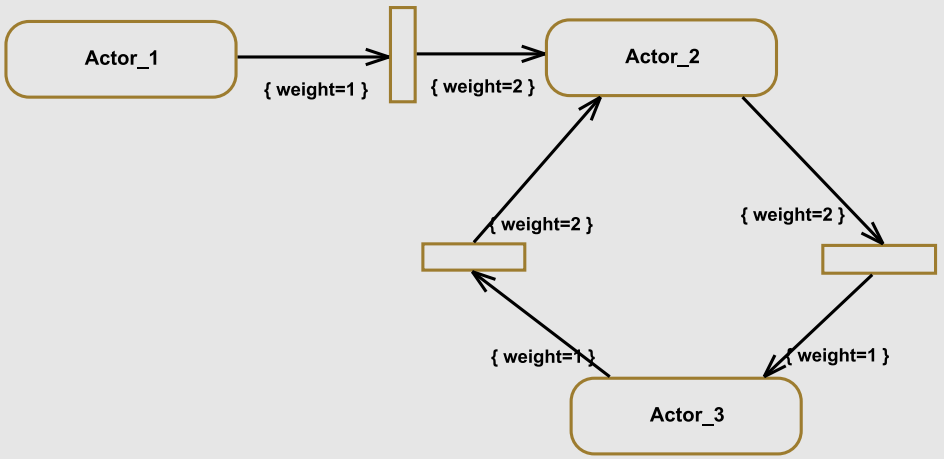


Syntax:

- computational elements
  - **actors**

Data-flow models

SDF model as an UML activity diagram

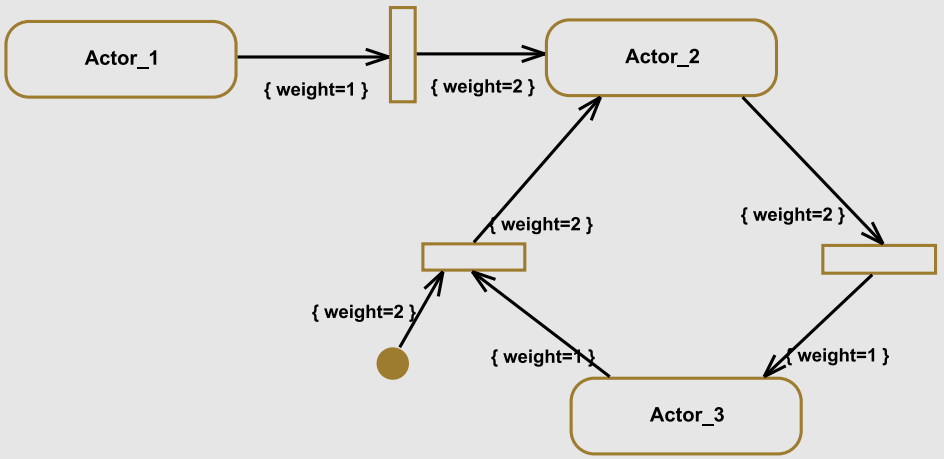


Syntax:

- computational elements – **actors**
- FIFO channels – **arcs**

Data-flow models

SDF model as an UML activity diagram

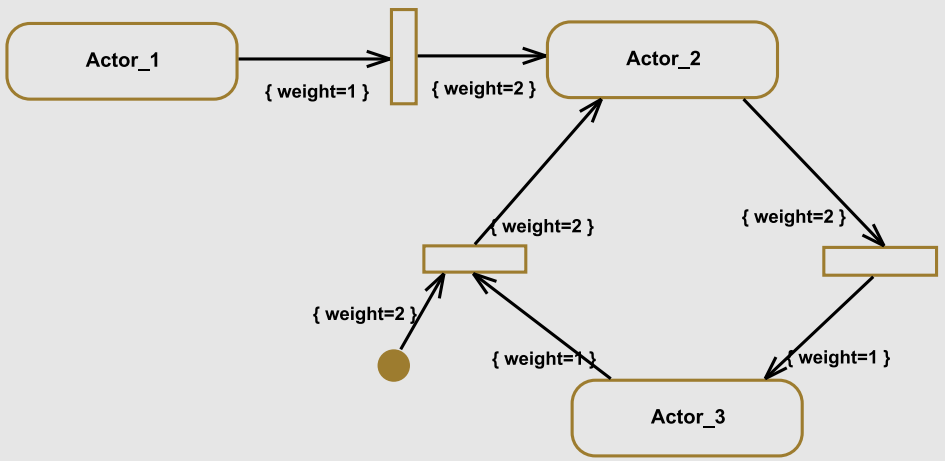


Syntax:

- computational elements – **actors**
- FIFO channels – **arcs**
- initial values – **delays**

Data-flow models

SDF model as an UML activity diagram



Syntax:

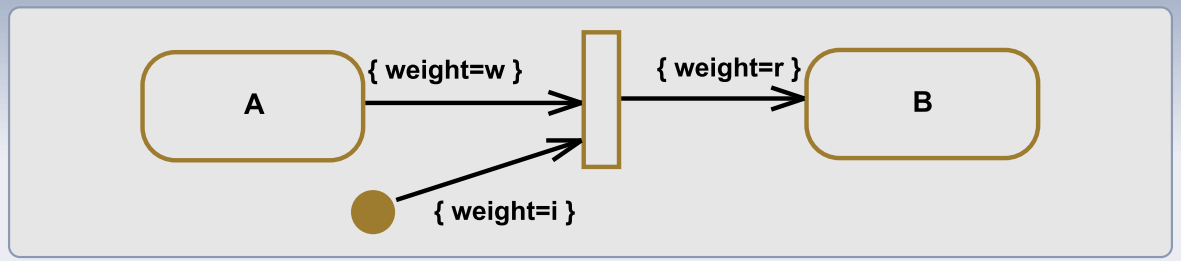
- computational elements – **actors**
- FIFO channels – **arcs**
- initial values – **delays**

Synchronous data-flow semantics:

- **fixed amount** of data elements produces/consumed at each firing
- local **producer/consumer rules** defined by each arc

SDF: data dependency semantics

General representation of a SDF arc

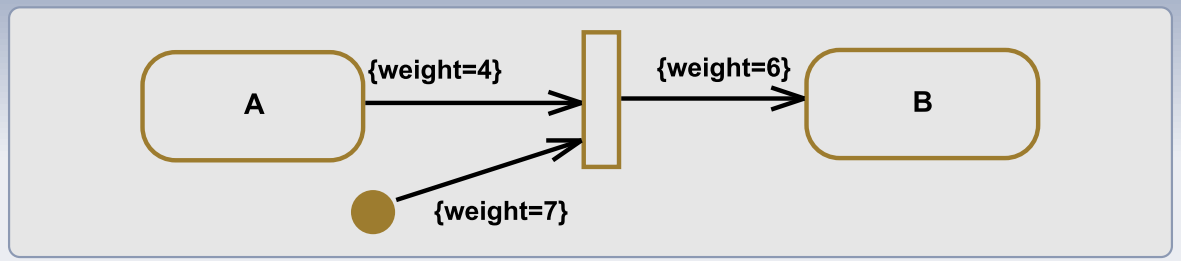


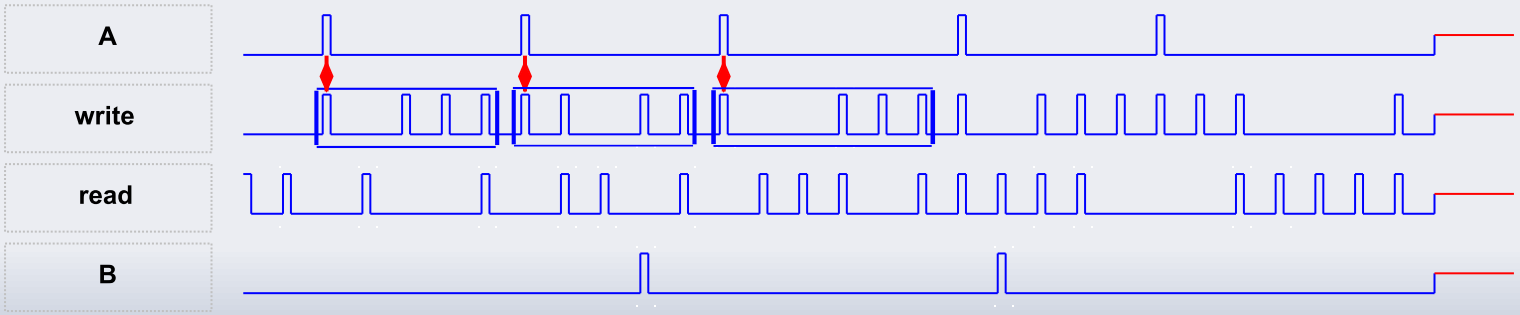
Relevant events in the system:

- Actor **firings**
- Element-wise **write** and **read** events on arcs

SDF: data dependency semantics

General representation of a SDF arc

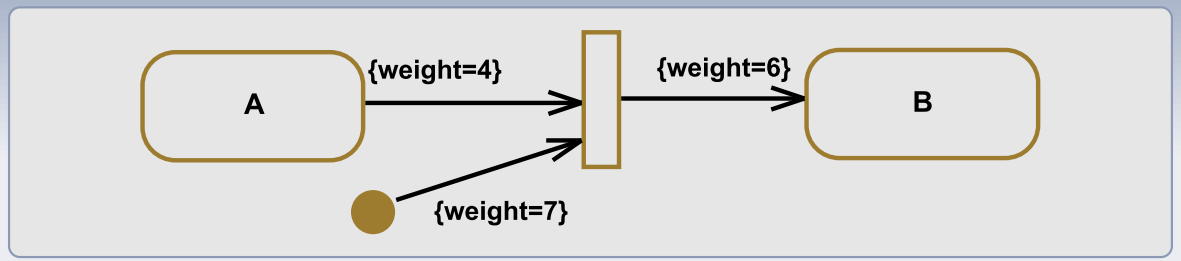


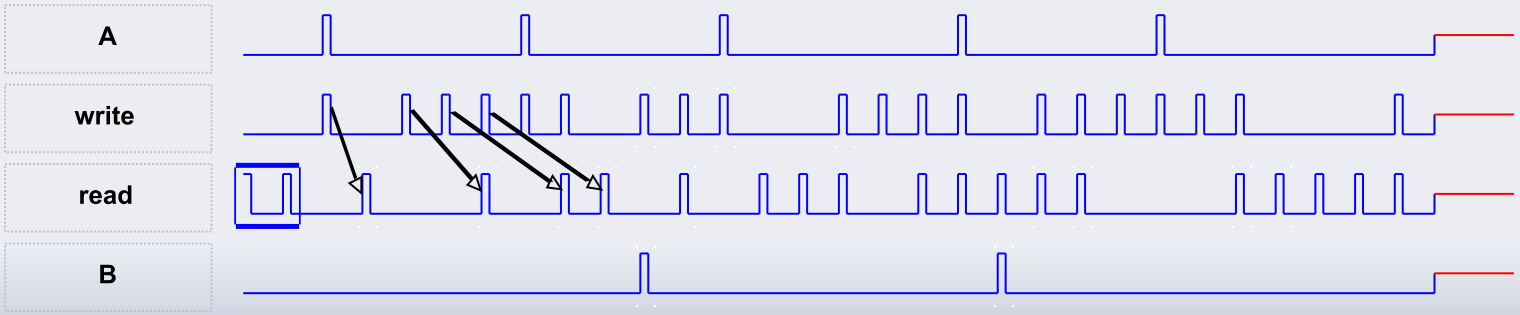


- Each actor firing is followed by $write_{weight}$ write events on each outgoing arc

SDF: data dependency semantics

General representation of a SDF arc

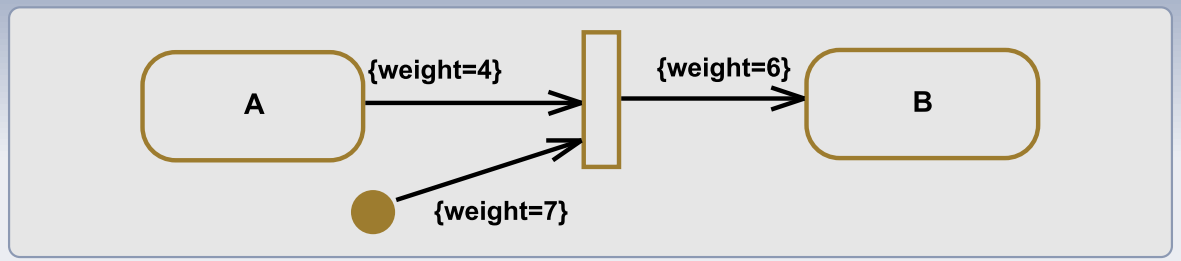


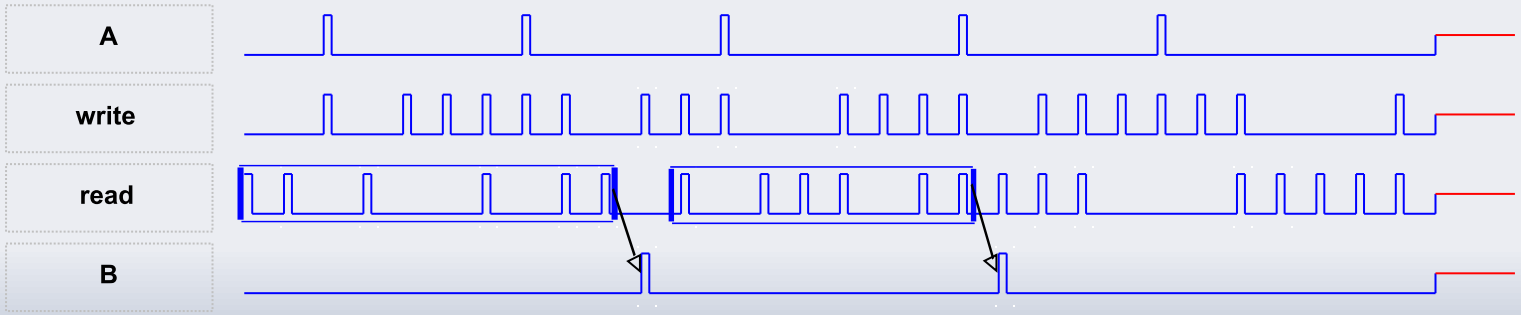


- On each arc, read events (delayed by the initial value) must follow write events

SDF: data dependency semantics

General representation of a SDF arc

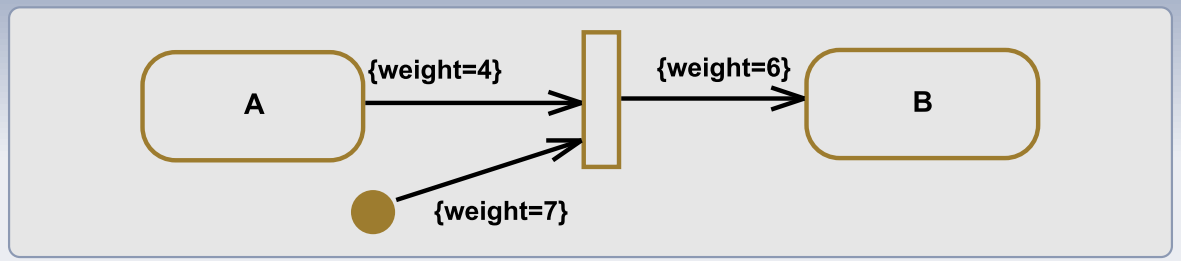




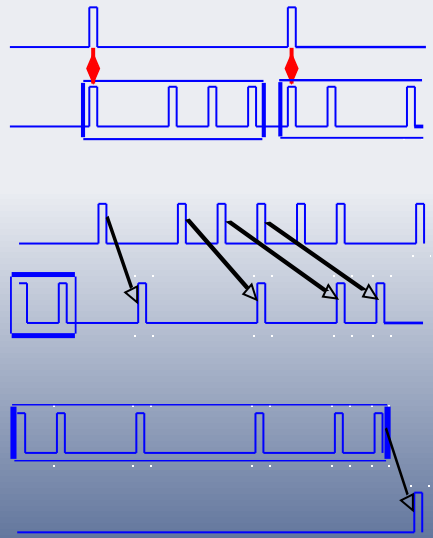
- $read_{weight}$ read events on each of its ingoing arc precede an actor firing

SDF: data dependency semantics

General representation of a SDF arc

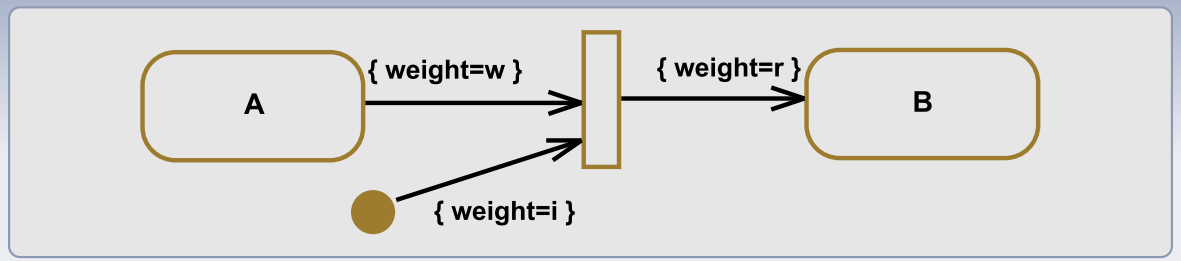


1. *actor* $\boxed{=}$ $\left(write \blacktriangledown \left(1.0^{weight-1}\right)^{\omega}\right)$
2. *write* $\boxed{\prec}$ $(read\ \$\ delay)$
3. $\left(read \blacktriangledown \left(0^{weight-1}.1\right)^{\omega}\right)$ $\boxed{\prec}$ *actor*



SDF: execution dependency semantics

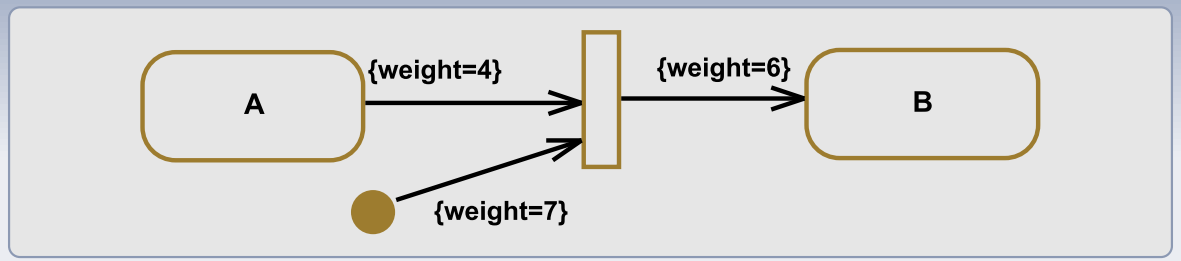
General representation of a SDF arc



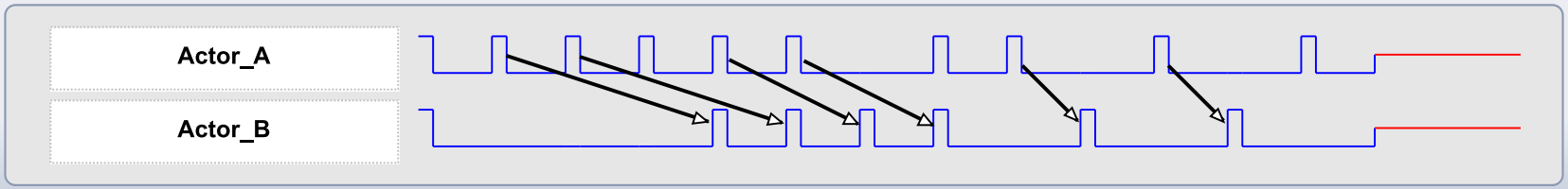
Direct precedence (**computed**) between the two clocks

SDF: execution dependency semantics

Example of SDF arc



Direct precedence (**computed**) between the two clocks



$$(Clock_{master} \blacktriangledown (011)^{\omega}) \prec (Clock_{slave} \$ 1)$$

Direct precedence computation algorithm

Iterative algorithm to compute the parameters of the general **execution precedence** relation:

$$(Clock_{master} \blacktriangledown M) \boxed{\prec} ((Clock_{slave} \$ d) \blacktriangledown S)$$

Direct precedence computation algorithm

Iterative algorithm to compute the parameters of the general **execution precedence** relation:

$$(Clock_{master} \blacktriangledown M) \boxed{\prec} ((Clock_{slave} \$ d) \blacktriangledown S)$$

$$d = \lfloor delay_{\text{weight}} / read_{\text{weight}} \rfloor$$

$$initial = delay_{\text{weight}} \bmod read_{\text{weight}}$$

Direct precedence computation algorithm

Iterative algorithm to compute the parameters of the general **execution precedence** relation:

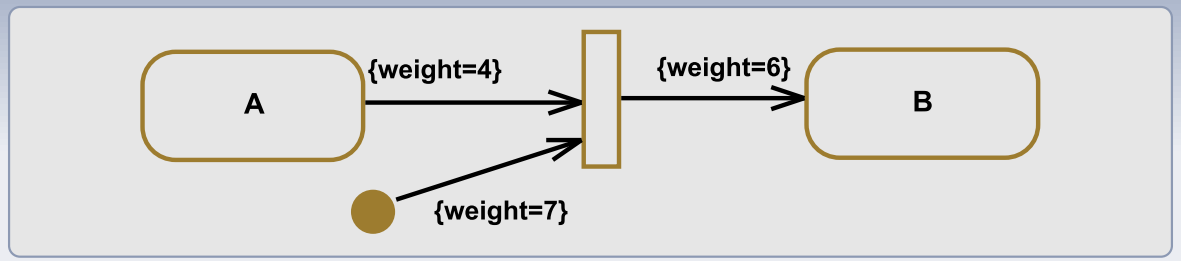
$$(Clock_{master} \blacktriangledown M) \boxed{\prec} ((Clock_{slave} \$ d) \blacktriangledown S)$$

$$d = \lfloor delay_{\text{weight}} / read_{\text{weight}} \rfloor \quad = \lfloor 7/6 \rfloor = 1$$

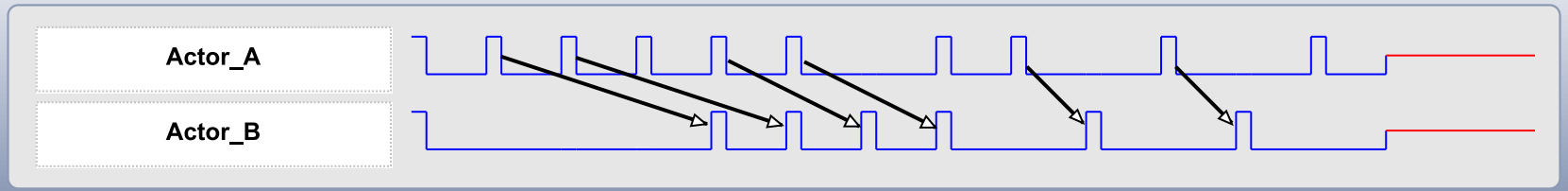
$$initial = delay_{\text{weight}} \bmod read_{\text{weight}} \quad = 7 \bmod 6 = 1$$

	initial		+4	+4	-6+4	-6+4
tokens	1		5	9	7	5
			< 6	> 6	> 6	done
binary		(	0	1	1	)

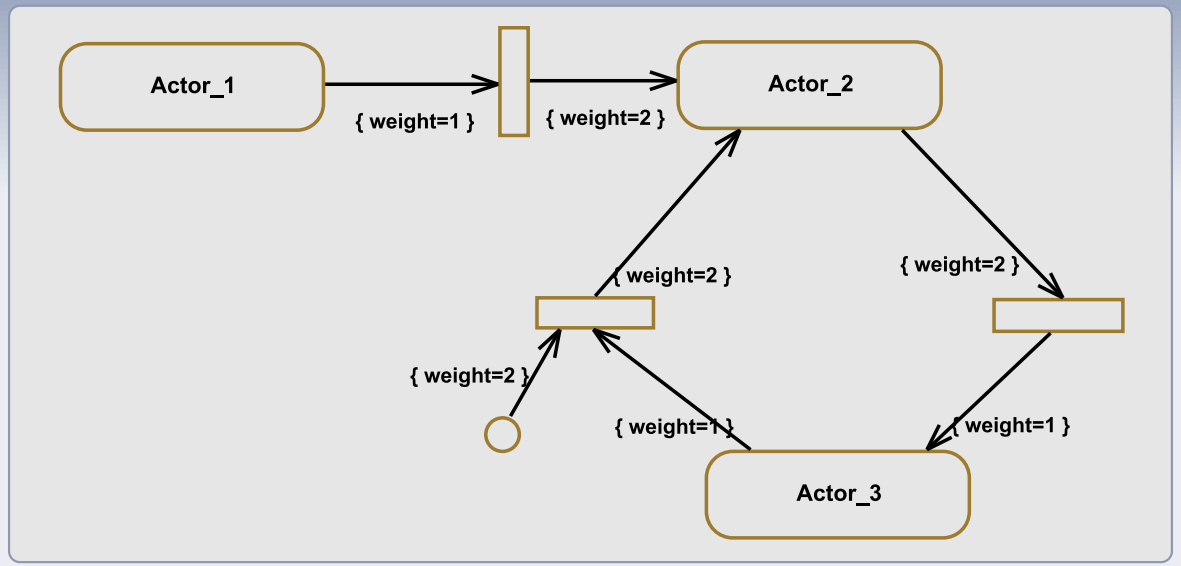
Local rules – global functionality

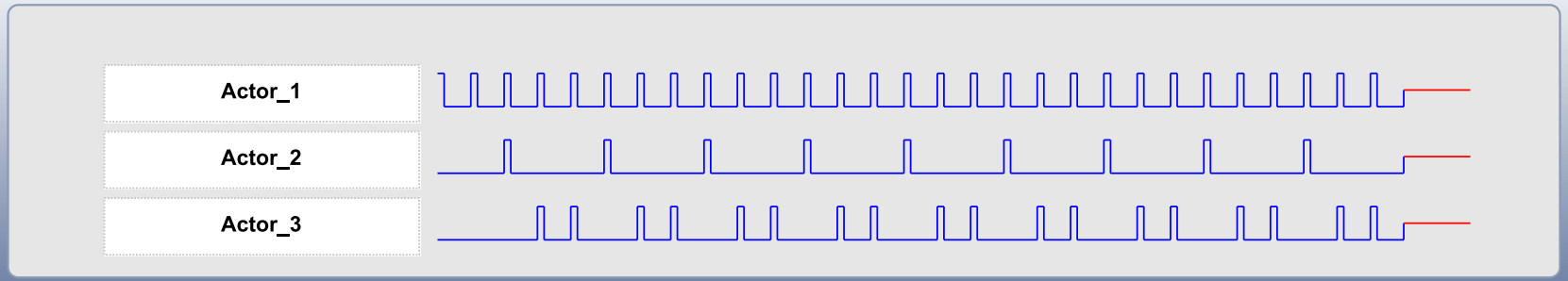


$$(Actor_A \blacktriangledown (011)^{\omega}) \prec (Actor_B \$ 1)$$

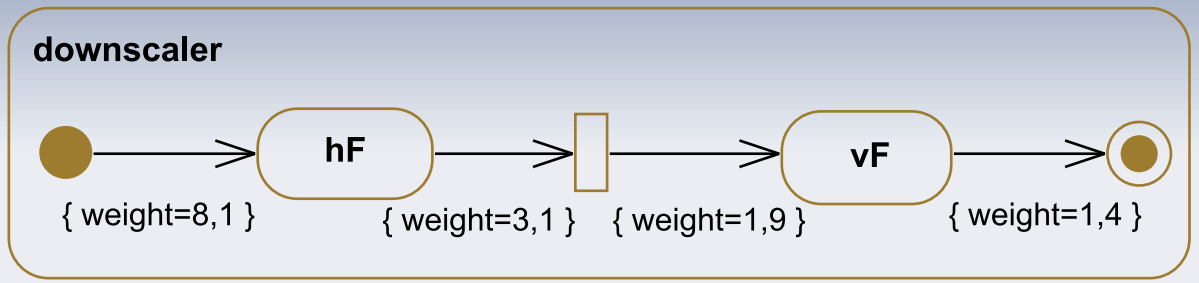


Local rules – global functionality





Encoding MDSDF in CCSL

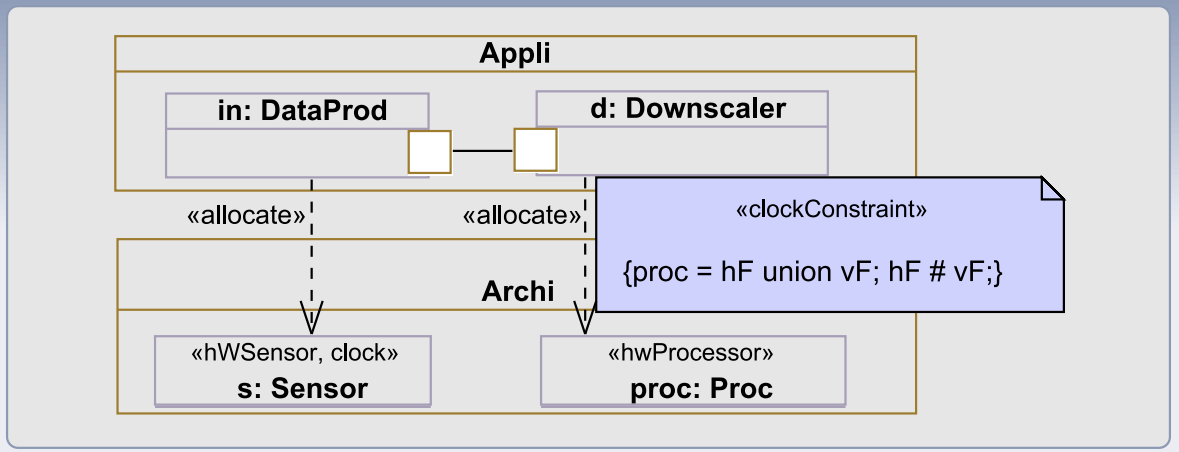


- straightforward multi-D extension of 1-D SDF
- quasi-independent relations producer/consumer by dimension

$in_1 \boxed{\prec} (hF_1 \blacktriangledown (1.0^2)^\omega)$
$in_2 \boxed{\prec} hF_2$

$hF_1 \boxed{\prec} (vF_1 \blacktriangledown (1.0^2)^\omega)$
$(hF_2 \blacktriangledown (0^8.1)^\omega) \boxed{\prec} vF_2$

External constraints



- multidimensional order – environment constraints

$$in_1 = (s \blacktriangledown 1.(0)^{\omega}) \qquad in_2 = s$$
$$hF_1 = (hF \blacktriangledown 1^3.(0)^{\omega}) \qquad hF_2 = (hF \blacktriangledown (0^2.1)^{\omega})$$
$$vF_1 = (vF \blacktriangledown 1^9.(0)^{\omega}) \qquad vF_2 = (vF \blacktriangledown (1.0^8)^{\omega})$$

- execution platform constraints

$$proc = hF + vF \qquad hF \# vF$$

System refinement

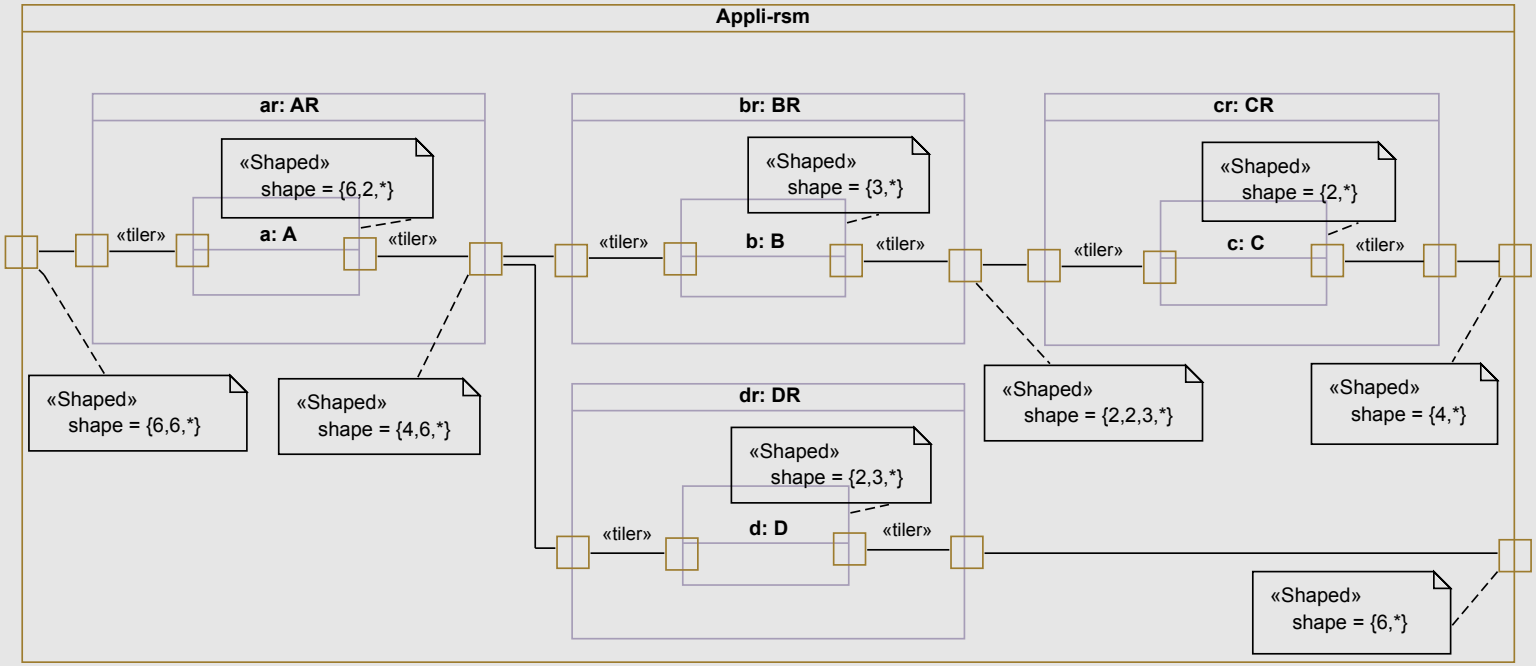
Formal specification encoding the entire set of schedules corresponding to a correct execution

Enforce the constraints:

- internal (data dependencies)
- external (environment or execution platform)
- buffer capacities - reverse constraints
- physical time – durations: execution, communication, ...

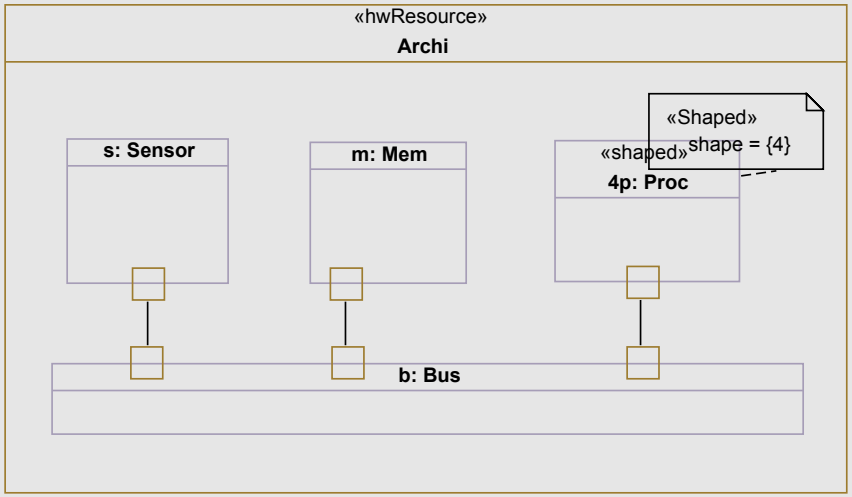
Repetitive Structure Modeling

Application – data dependencies



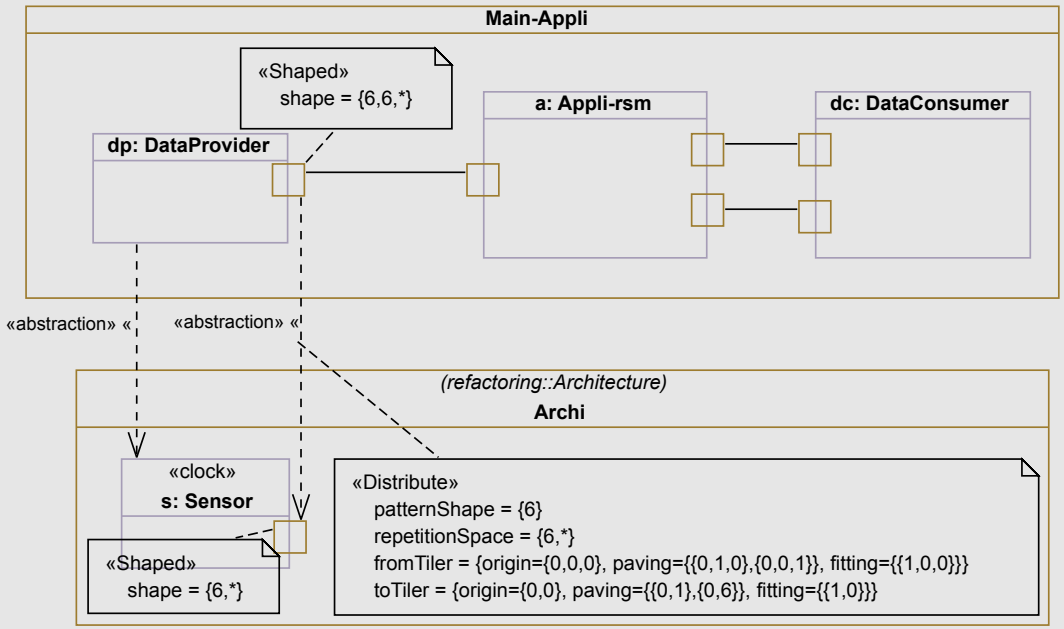
Repetitive Structure Modeling

Architecture – execution platform



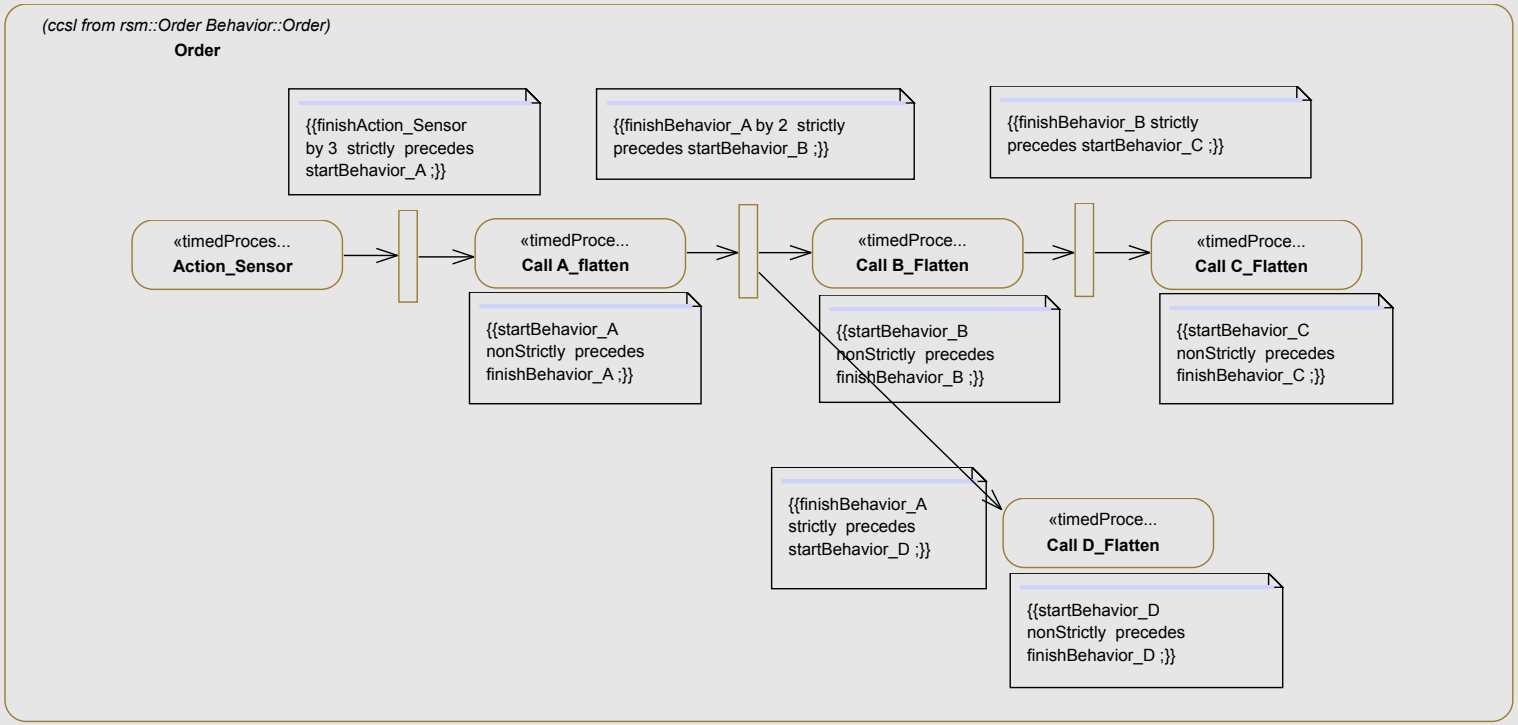
Repetitive Structure Modeling

Multi-D structures – projection in time and space



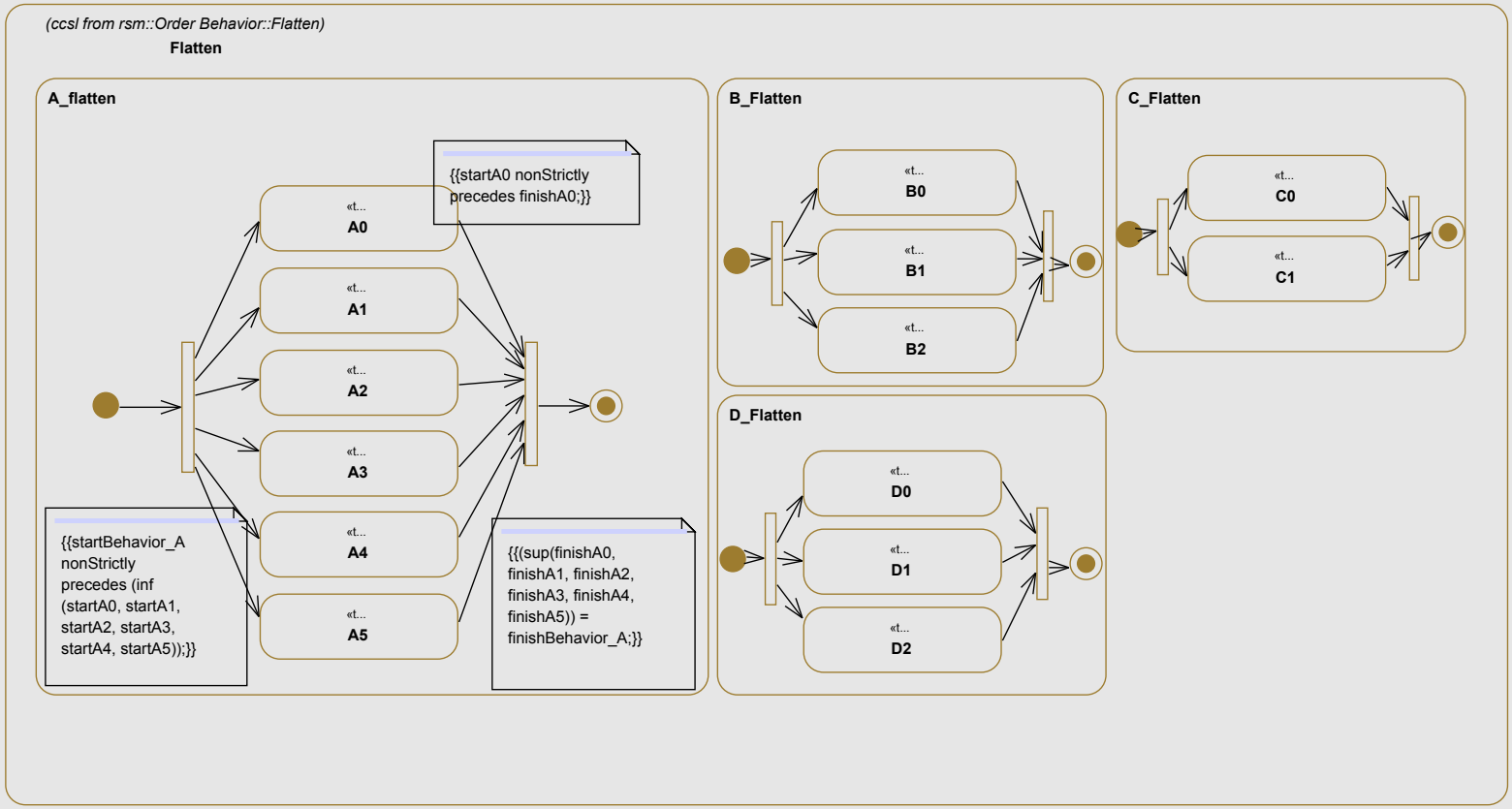
Internal constraints

Execution dependencies : precedence



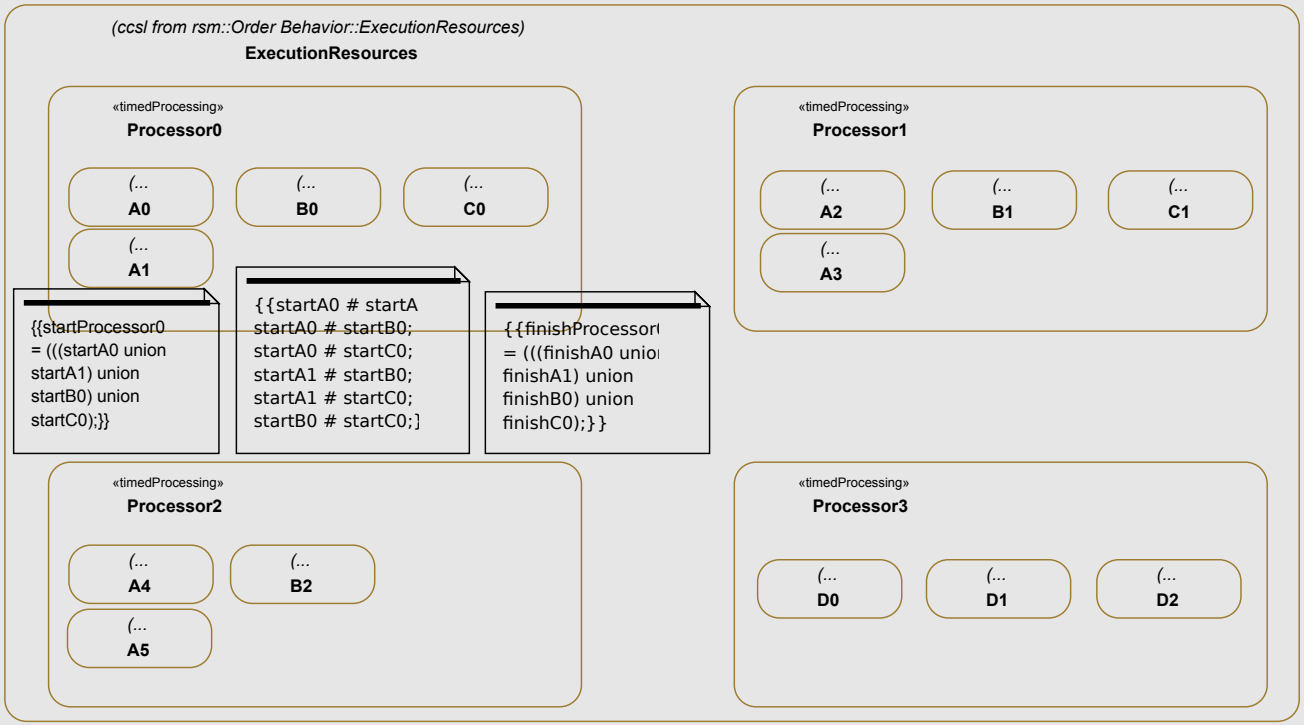
Internal constraints

Task parallelism – unbounded resources



External constraints

Execution resources



Time Model & CCSL (MARTE)

Logical Time:

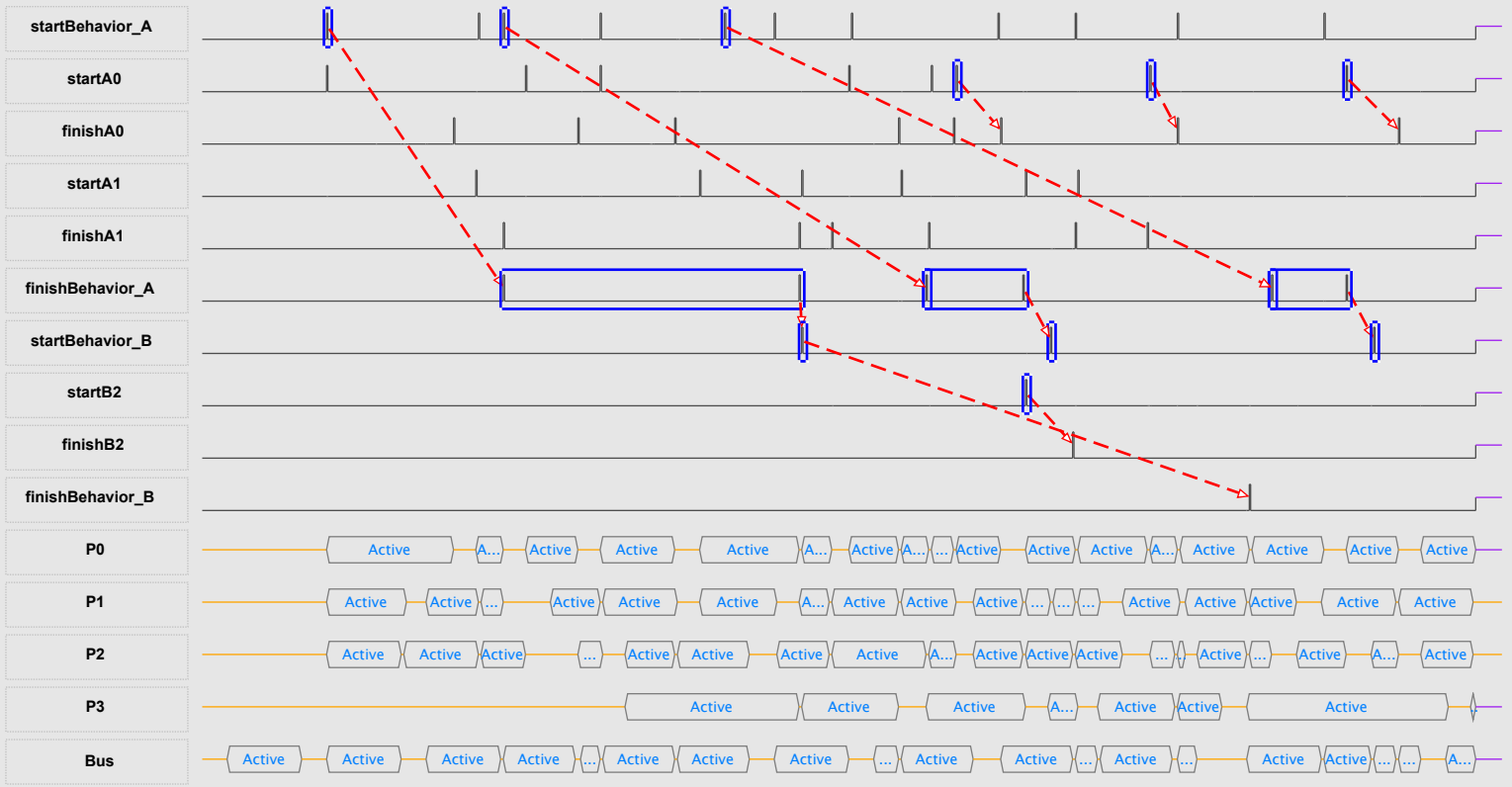
- Dependencies (data, execution)
- Parallelism
- Concurrence (resources: execution, communication, storage)

Projection into **Physical Time**

- Physical clocks
- Durations
- Delays
- ...

Schedule constraints

TimeSquare simulation



Conclusion

Results:

- Define explicit semantics
- For analysis, static scheduling, state space exploration
- Simulation in TimeSquare[1]

More complex languages:

- Array-OL: no predefined order
- Uniform loop nests: complex dependence analysis

Heterogeneous systems

[1] http://www-sop.inria.fr/aoste/dev/time_square