Logical Time at Work:

Capturing Data Dependencies and Platform Constraints

Calin Glitia Julien DeAntoni Frédéric Mallet

INRIA Sophia Antipolis Méditerranée

Université Nice Sophia Antipolis

Forum on specification & Design Languages, Sept 14th-16th, 2010, Southampton, UK

Overview

Give formal semantics to syntactical models

Overview

Give formal semantics to syntactical models

(Modeling) languages are defined by:

- **syntax**: the form of a valid program/model
- (behavioral) **semantics**: how it should be interpreted

Modeling languages:

- semantics defines apart/informal/hard coded

Overview

Give formal semantics to syntactical models

(Modeling) languages are defined by:

- **syntax**: the form of a valid program/model
- (behavioral) **semantics**: how it should be interpreted

Modeling languages:

- semantics defines apart/informal/hard coded

Semantics should be explicit

...more precise

Unified Modeling Language (UML)

- Extended and specialized by **UML profiles**

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

…more precise

Unified Modeling Language (UML)

- Extended and specialized by **UML profiles**
- Captures just the syntactical aspects
- No formal description of how it should be interpreted

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

...more precise

Unified Modeling Language (UML)

- Extended and specialized by **UML profiles**
- Captures just the syntactical aspects
- No formal description of how it should be interpreted

Define the semantics of syntactical models

- MARTE Time Model & Clock Constraint Specification Language

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

. . . more precise

Unified Modeling Language (UML)

- Extended and specialized by **UML profiles**
- Captures just the syntactical aspects
- No formal description of how it should be interpreted

Define the semantics of syntactical models

- MARTE Time Model & Clock Constraint Specification Language

Integrated Development Environment

- Papyrus UML + MARTE profile
- TimeSquare[1] for simulation/execution/analysis

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

Outline

Define the semantics of synchronous data-flow formalisms

- **Syntax**: UML Activity Diagram
- **Semantics**: constraint logical time
    - relevant events as **logical clocks**
    - translate language rules into **clock relations**

Outline

Define the semantics of synchronous data-flow formalisms

- **Syntax**: UML Activity Diagram
- **Semantics**: constraint logical time
  - relevant events as **logical clocks**
  - translate language rules into **clock relations**

1. Encode data-dependencies of SDF models
2. Translate data-dependencies to execution dependencies
3. Multi-dimensional semantics (MDSDF)
4. Multidimensional order: environment constraints
5. External constraints

Clock Constraint Specification Language

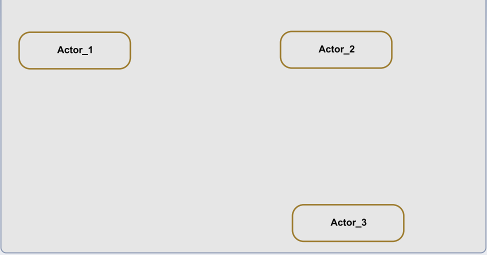
Modeling and Analysis of Real-Time and Embedded systems (MARTE)

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

CCSL relations:	CCSL clock expressions:
• **precedence** ($\prec$) • **coincidence** ($\equiv$) • **exclusion** ($\#$)	• **filteredBy** ($\blacktriangledown$) – by a binary periodic word • **delay** ($\$$) – by an integer value

Data-flow models

SDF model as an UML activity diagram

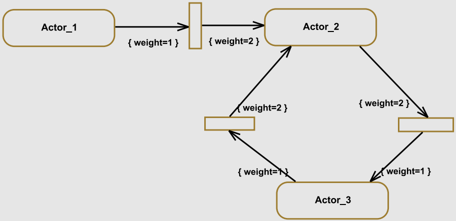


Syntax:

- computational elements
  - **actors**

Data-flow models

SDF model as an UML activity diagram

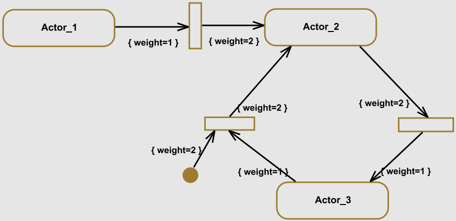


Syntax:

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

Data-flow models

SDF model as an UML activity diagram

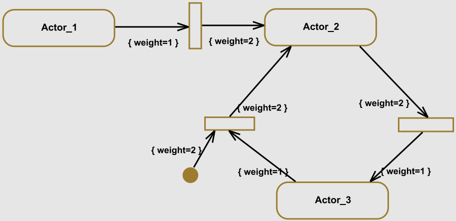


Syntax:

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

Data-flow models

SDF model as an UML activity diagram



Syntax:

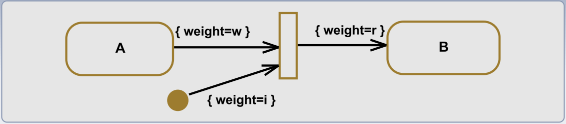
- computational elements – **actors**
- FIFO channels – **arcs**
- initial values – **delays**

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

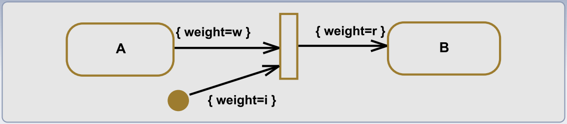


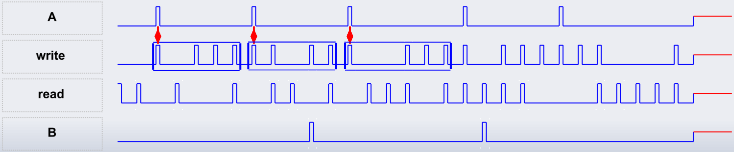
Relevant events in the system:

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

SDF: data dependency semantics

General representation of a SDF arc

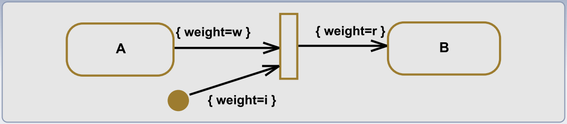


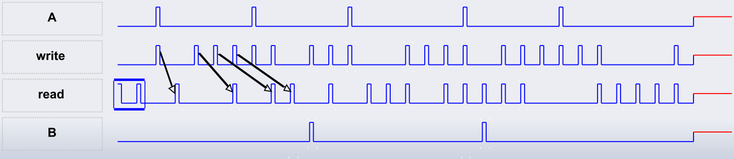


- Each actor firing is followed by $write_{weight}$ write events on each outgoing arc

SDF: data dependency semantics

General representation of a SDF arc

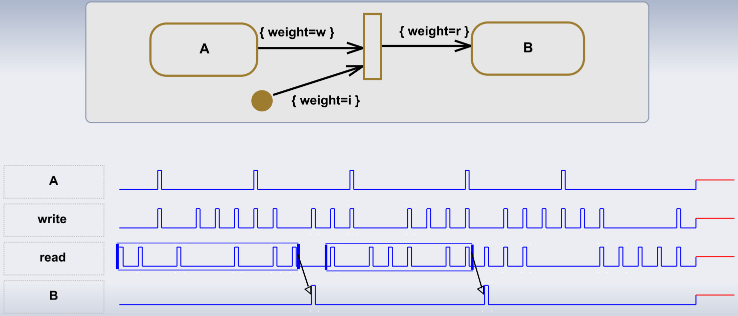




- On each arc, read events (delayed by the initial value) must follow write events

SDF: data dependency semantics

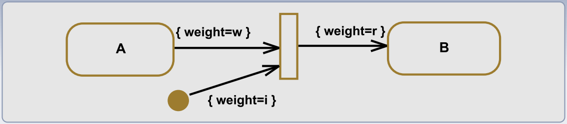
General representation of a SDF arc



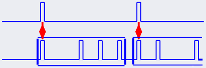
- *read*$_{weight}$ read events on each of its ingoing arc precede an actor firing

SDF: data dependency semantics

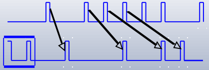
General representation of a SDF arc



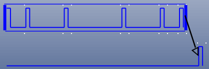
1. $producer \ \boxed{=} \ (write \ \blacktriangledown \ (1.0^{w-1})^{\omega})$



2. $write \ \boxed{\prec} \ (read \ \$ \ i)$

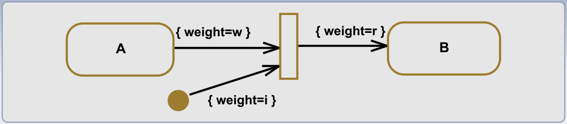


3. $(read \ \blacktriangledown \ (0^{r-1}.1)^{\omega}) \ \boxed{\prec} \ consumer$



SDF: execution dependency semantics

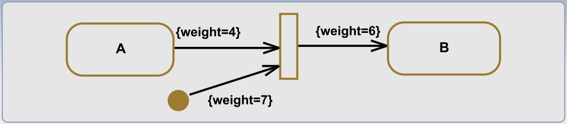
General representation of a SDF arc



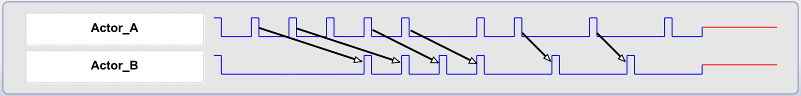
Direct precedence (**computed**) between the two clocks

SDF: execution dependency semantics

Example of SDF arc



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



$$(Clock_{producer} \blacktriangledown (011)^{\omega}) \boxed{\prec} (Clock_{consumer} \, \$ \, 1)$$

Direct precedence computation algorithm

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

$$(Clock_{producer} \blacktriangledown P) \boxed{\prec} ((Clock_{consumer} \$ \, indep) \blacktriangledown C)$$

Direct precedence computation algorithm

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

$$(\mathit{Clock}_{\mathit{producer}} \blacktriangledown P) \boxed{\prec} ((\mathit{Clock}_{\mathit{consumer}} \; \$ \; \mathit{indep}) \blacktriangledown C)$$

$$\mathit{indep} = \lfloor \mathit{initial}_{\text{weight}} / \mathit{read}_{\text{weight}} \rfloor$$
$$\mathit{initial} = \mathit{initial}_{\text{weight}} \bmod \mathit{read}_{\text{weight}}$$

Direct precedence computation algorithm

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

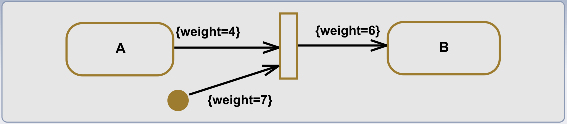
$$(Clock_{producer} \blacktriangledown P) \boxed{\prec} ((Clock_{consumer} \; \$ \; indep) \blacktriangledown C)$$

$$indep = \lfloor initial_{\text{weight}} / read_{\text{weight}} \rfloor \quad = \lfloor 7/6 \rfloor = 1$$

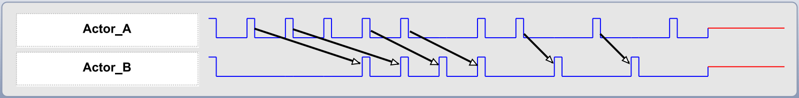
$$initial = initial_{\text{weight}} \bmod read_{\text{weight}} \quad = 7 \bmod 6 = 1$$

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

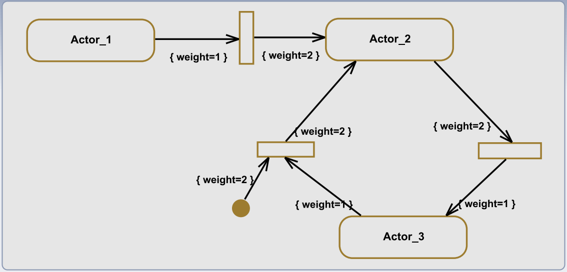
From local rules to global functionality

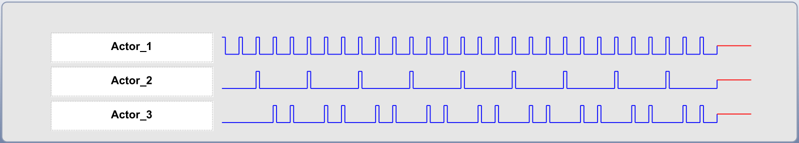


$$(Actor_A \blacktriangledown (011)^{\omega}) \boxed{\prec} (Actor_B \; \$ \; 1)$$

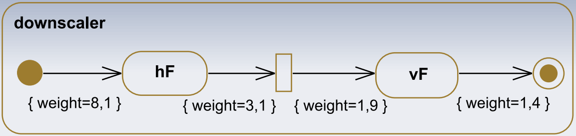


From local rules to global functionality





Encoding MDSDF in CCSL

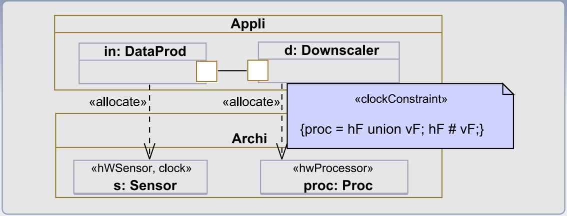


- straightforward multi-D extension of 1-D SDF
- quasi-independent relations producer/consumer by dimension

$$in_1 \prec \left(hF_1 \blacktriangledown \left(1.0^2\right)^\omega\right) \qquad hF_1 \prec \left(vF_1 \blacktriangledown \left(1.0^2\right)^\omega\right)$$

$$in_2 \prec hF_2 \qquad \left(hF_2 \blacktriangledown \left(0^8.1\right)^\omega\right) \prec vF_2$$

External constraints



- multidimensional order – environment constraints

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

- execution platform constraints

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

To resume ...

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

Generally, the behaviour of a system can be seen as:

- a set of **operations** applied to an **initial state**
- into a certain **order** (execution dependencies)

To resume ...

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

Generally, the behaviour of a system can be seen as:

- a set of **operations** applied to an **initial state**
- into a certain **order** (execution dependencies)

Logical Time refinement:

- Functional semantics (**internal** constraints)
- **External** constraints (environment or execution platform)
- **Buffer** capacities
- **Physical time** – durations: execution, communication, ...

Conclusion

We used **constraint logical time** to:

- Define explicit semantics of synchronous data-flow models
- Capture data-dependencies
- Express computed execution dependencies
- Integrate external constraints

Papyrus UML, MARTE profile and TimeSquare:

- OMG standard
- Time simulation/analysis
- Detect inconsistencies (deadlocks)
- Compute periodic schedule