Building Synchronous DataFlow graphs with UML & MARTE/CCSL

F. Mallet, J. DeAntoni, C. André, R. de Simone

Aoste - INRIA/I3S

Université de Nice Sophia Antipolis
Building Synchronous Data Flow graphs with UML and MARTE/CCSL

UML & Formal methods

- Ambiguity and structural properties of basic sequence diagrams
  - Interactions + trace

- Extending statecharts with process algebra operators
  - Untimed StateMachines + CSP

- UML Behavioral consistency checking using instantiable Petri Nets
  - Activities + PN

- Timing analysis and validation with UML
  - StateMachines + Timed automata
Building Synchronous Data Flow graphs with UML and MARTE/CCSL

Compare their constraining power: level of concurrency

Do we want to choose between all these?
- Use UML as a framework for combining all of these semantics
- Apply directors (like in Ptolemy) to choose the suitable semantics
Profiles to give the semantics

- Profile for CSP

- Profile for OMEGA

How to combine these two diagrams?
  - Put them next to each other?
Proposition

- **What**
  - Explicit execution semantics within the model

- **How**
  - Annotate the meta-model
  - Execution semantics defined with MARTE/CCSL

- **UML Profile for MARTE and CCSL**
  - *Modeling and Analysis of Real-Time and Embedded systems*
    - Time model => Timed Causality Semantics to UML models
  - CCSL: MARTE Companion *Modeling Language*
    - Apply to any (EMF) model => UML or not

- **Example**
  - UML (activity/state machine): *Synchronous DataFlow graphs*
  - CCSL Library for SDF
CCSL – Polychronous systems

Clock Model
- Clock $C=\langle I, \prec \rangle$, infinite ordered set of instants
  - Discrete-time clocks: $I$ is discrete and indexed by $\mathbb{N}^*$
Clock Model

- Clock $C = \langle I, \prec \rangle$, infinite ordered set of instants
  - Discrete-time clocks: $I$ is discrete and indexed by $\mathbb{N}^*$
- Instant relations: coincidence, (strict) precedence, exclusion
- Clock relations:
  - infinitely many instant relations according to predefined patterns
    (periodicity, alternation, sampling, …)

![Clock Model Diagram](image)
CCSL – Polychronous systems

- Clock Model (static)
  - Clock $C = \langle I, \prec \rangle$, infinite ordered set of instants
  - Instant/clock relations = constraints

- Time system (dynamic)
  - Clocks = set of boolean variables
  - Constraints = set of boolean equations => SAT problem

Event structures [Winskel]
Occurrence nets [Petri]
TaggedSystems [LSV]
CCSL clock constraint - precedence

Precedence

A precedes B (strict form) written as $A \preceq B$

Semantics

$$\forall k \in \mathbb{N}^* \ A[k] \preceq B[k]$$

$$\beta \models (\chi(A) = \chi(B))$$

$A \preceq B = (\beta \Rightarrow \neg B)$

Logical representation

Simulation

On-demand visualization of precedence relation between instants

Use: causal dependency or asynchronous communication
Building Synchronous Data Flow graphs with UML and MARTE/CCSL

CCSL clock constraint - Synchrony

Synchrony

\[ A = B \quad \text{written as} \quad A \equiv B \]

Semantics

\[ (\forall k \in \mathbb{N}^*) \ A[k] \equiv B[k] \]

Simulation

Use: synchronous evolutions
CCSL clock constraint—filtering

Filtering

\[ B = A \text{ filteredBy } w \text{ written as } B \equiv A \downarrow w \text{ where } w \in B^\omega \text{ (infinite) Binary Word} \]

Semantics

\[ (\forall k \in \mathbb{N}^*) B[k] \equiv A[w \uparrow k] \]

where \( w \uparrow k \) is the index of the \( k^{\text{th}} \) 1 in \( w \)

\[ \beta \equiv (w = 1.v) \]

\[ \equiv A \downarrow w \equiv (\beta \land A) \]

Logical representation

Simulation

Use: a special case of synchrony (on selected instants)
Synchronous Data Flow (SDF)

- Data Flow graphs
  - Directed graphs
  - Nodes = functions/computations
  - Arcs = data path

- Synchronous Data Flow  
  [E.A. Lee, 1987]
  - Static number of data samples consumed/produced by each node
  - Well-suited for multi-rate **DSP algorithms** with continuous stream of data
  - Reduction of **Kahn-Process Networks** to allow **static scheduling** and ease parallelization
  - Equivalent to Computation Graphs  
    [Karp & Miller, 1966]
  - Popular due to Ptolemy developed in Berkeley
Nodes are called actors
Arcs have a delay
Input/Output have a weight
  - Number of data samples consumed/produced
Equivalent to a Marked-Event Graph

- Conflict-free Petri Net
- Static scheduling: A A B A A B C 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
CCSL Library for SDF (1/2)

- **SDF**
  - Actor A
  - Token T
    - `write delay read`
  - Input i
    - `write weight read`
  - Output o
    - `write weight read`

- **CCSL**
  - Clock A;
  - Clock write, read;
    - `def token(clock write, clock read, int delay) △
      write [read delayedFor delay]`
  - `def input(clock actor, clock read, int weight) △
    (read by weight) < actor`
  - `def output(clock actor, clock write, int weight) △
    actor [=] (write filteredBy (1.0^weight – 1)^ω)`
CCSL Library for SDF (2/2)

- SDF

source \[\xrightarrow{\text{delay}}\] target

- CCSL

```plaintext
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)
```
Building Synchronous Data Flow graphs with UML and MARTE/CCSL

Example

\[ S = \text{arc}(0, A, 1, B, 2) \ | \ \text{arc}(0, B, 2, C, 1) \ | \ \text{arc}(2, C, 1, B, 2) \]
Conclusion

- **(UML) Models must come with**
  - A meta-model to describe the structural/composition rules
  - An explicit execution semantics

- **CCSL can be used for describing**
  - Temporal patterns
  - Causal relationships

- **MARTE: attach CCSL specifications to UML models**

- **TimeSquare can**
  - execute CCSL specifications
  - Animate DI2 models in Papyrus
  - [http://www.inria.fr/sophia/aoste/time_square/](http://www.inria.fr/sophia/aoste/time_square/)