DML: a Markup Language for DEVS Models

Luc Touraille, Mamadou K. Traoré, David R.C. Hill

LIMOS
UMR 6158 CNRS

Université Blaise Pascal

UNIVERSITE D'AUVERGNE

Cargese Interdisciplinary Seminar 2009

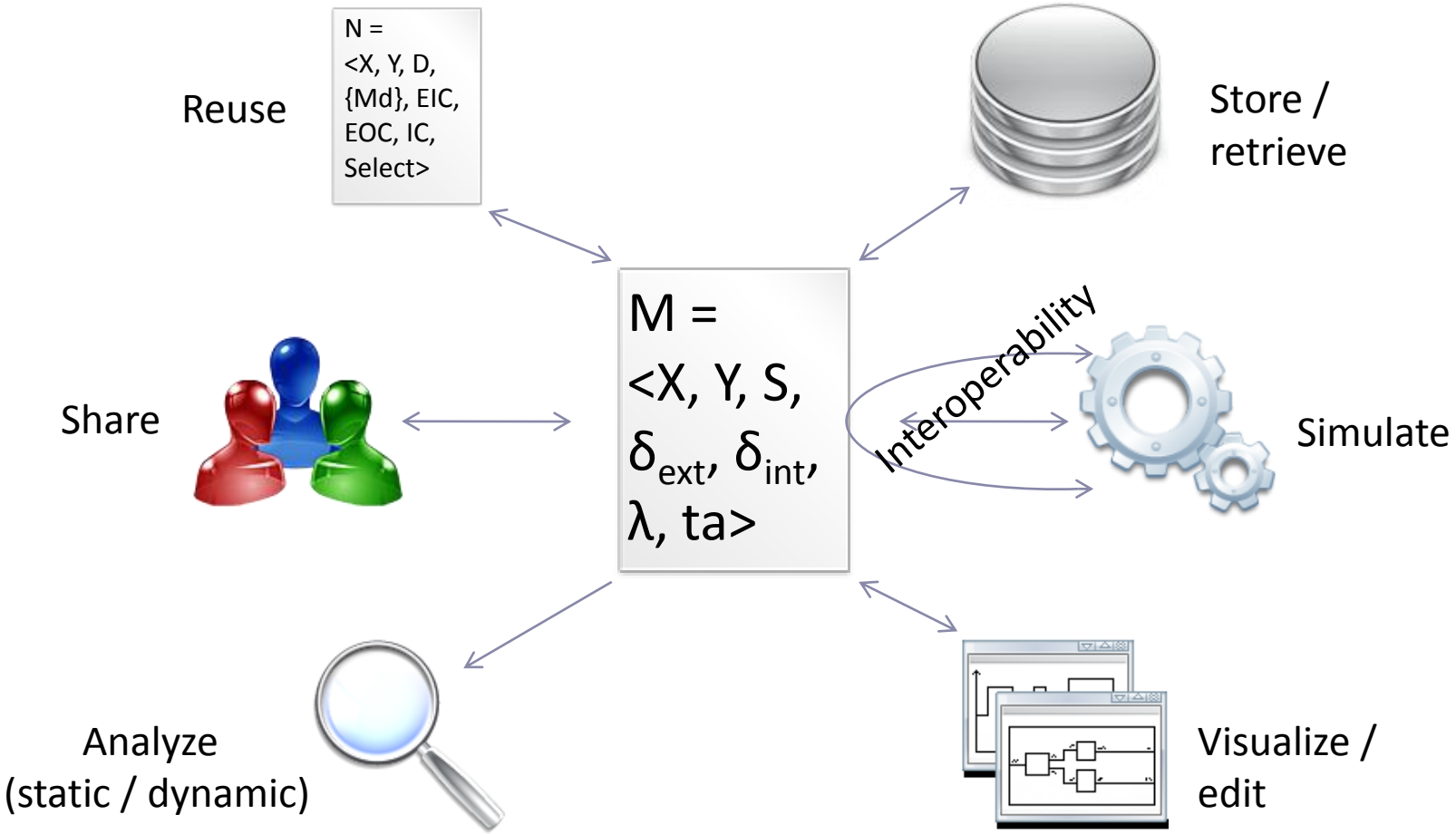
Introduction

- Huge efforts in the DEVS community w.r.t. interoperability
- Need for a unique and complete representation of several facets of DEVS models
- Choice of XML for its numerous advantages

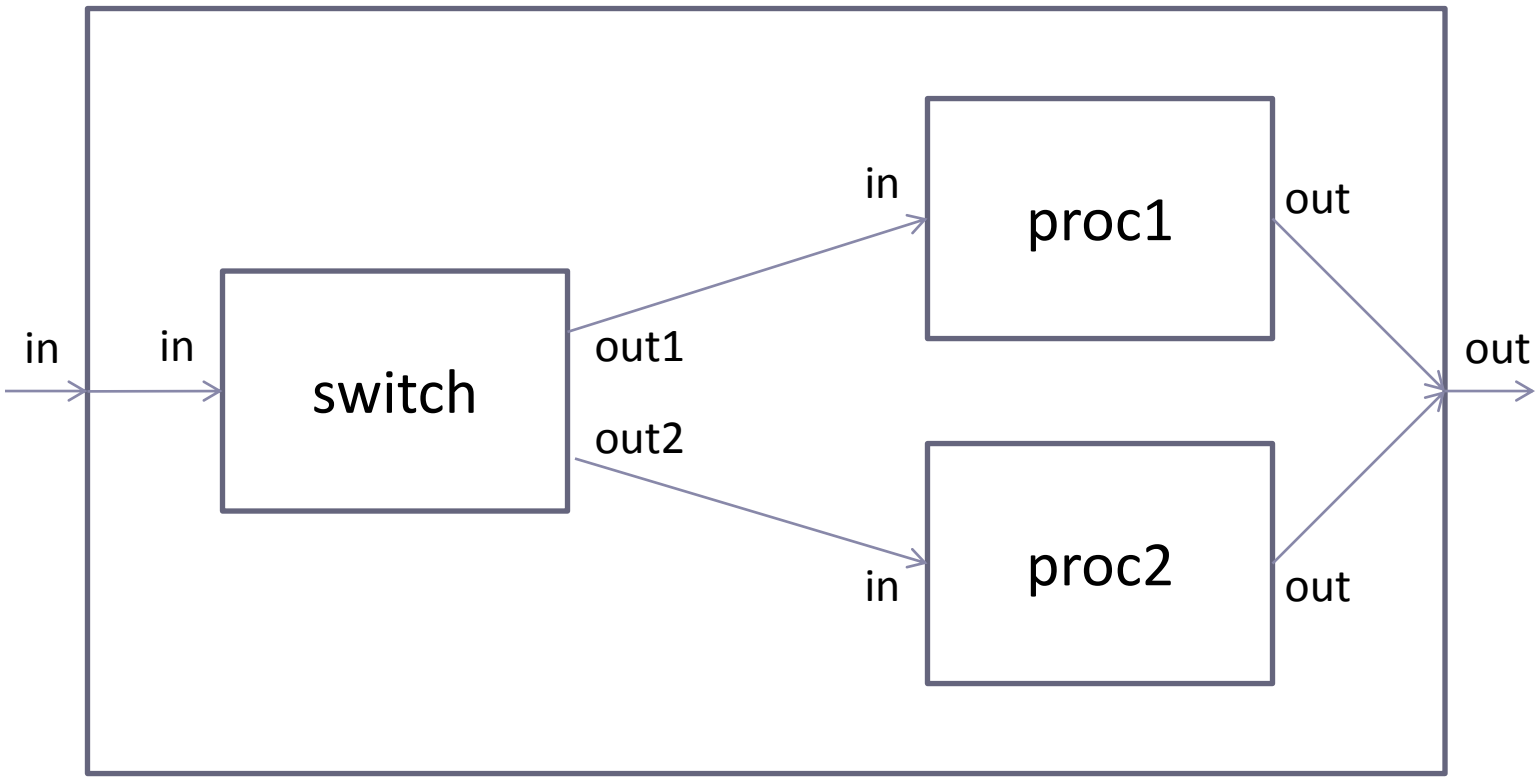
Outline

Outline

What do we do with a model?



Network switch [Zeigler]



Processor with buffer [Zeigler]

$DEVS_{proc\_time} = \langle X_M, Y_M, S, \delta_{ext}, \delta_{int}, \delta_{con}, \lambda, ta \rangle$

$X_M = \{ (\text{"in"}, V) \}$ $\quad Y_M = \{ (\text{"out"}, V) \}$

$S = \{ \text{"idle"}, \text{"busy"} \} \times \mathbb{R}^+ \times V^*$

$\delta_{ext}((phase, \sigma, buf, e, ((\text{"in"}, x1), ..., (\text{"in"}, xn)))$

$= (\text{"busy"}, proc\_time, (x1, ..., xn))$ *if phase = "idle"*

$= (\text{"busy"}, \sigma - e, buf.(x1, ..., xn))$ *otherwise*

$\delta_{int}((\text{"busy"}, \sigma, (x_i, ..., x_n))) = (\text{"idle"}, +\infty, ())$ *if i = n*

$= (\text{"busy"}, proc\_time, (x_{i+1}, ..., x_n))$ *otherwise*

$\delta_{con}(s, ta(s), x) = \delta_{ext}(\delta_{int} (s), 0, x)$

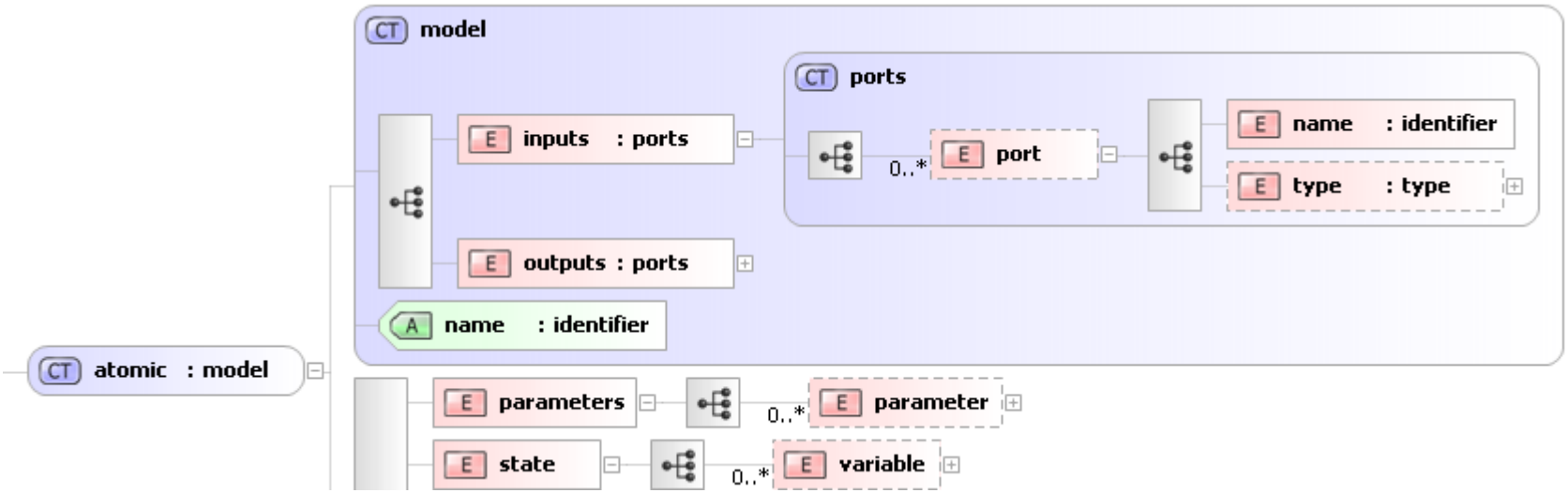
$\lambda(\text{"busy"}, \sigma , (x_i, ..., x_n)) = x_i$

$ta(phase, \sigma, buf) = \sigma$

Outline

Model structure

- Must allow the representation of state, ports, couplings, parameters



- Challenge: representation of types

Types

- Several type categories:
  - basic types (intrisic): integer, real, boolean, ...
    - built-in into the schema
  - user-defined types (custom): record, enumeration
    - language neutral representation (XML Schema)
  - library types (language-dependent): collections, RNG, matrices, ...
    - abstract type bound to several language specific types

Processor state variables

```
<atomic name="processorWithBuffer" xsi:type="classicAtomic">
  <!-- Model state variables -->
  <state>
    <variable>
      <name>sigma</name>
      <type>double</type>
    </variable>
    <variable>
      <name>phase</name>
      <type kind="custom">ProcessorPhase</type>
    </variable>
    <variable>
      <name>queue</name>
      <type kind="languageDependent">Queue</type>
    </variable>
  </state>
```

Processor state variables

```
<!-- Custom types -->
<type id="ProcessorPhase" xsi:type="O2XML:enum">
  <O2XML:value>idle</O2XML:value>
  <O2XML:value>busy</O2XML:value>
</type>

<type id="Queue" xsi:type="languageDependentType">
  <implementation language="java">
    java.util.LinkedList<String>
  </implementation>
  <implementation language="c++">
    std::queue<std::string>
  </implementation>
</type>
```

Model dynamics

- Description of functions: internal/external transitions, output, confluent, time advance
- Should be
  - as generic and language independent as possible...
  - ...but flexible enough to authorize the use of certain language features or library types
  - transformable into source code (and vice-versa (?))
- Combination of generic and language specific code snippets

Processor internal transition function

```
<deltaInt>
  ...
  <if>
    <test>
      <codeSnippet id="QueueIsEmpty" queue="queue" />
    </test>
    <then>
      <assignment>
        <lhs><var>phase</var></lhs>
        <rhs>
          <enumValue>
            <enum>ProcPhase</enum>
            <value>idle</value>
          </enumValue>
        </rhs>
        ...
```

Processor internal transition function

```
<deltaInt>
  ...
  <if>
    <test>
      <codeSnippet id="QueueIsEmpty" queue="queue" />
    </test>
    <then>
      <assignment>
        <lhs><var>phase</var></lhs>
        <rhs>
          <enumValue>
            <enum>ProcPhase</enum>
            <value>idle</value>
          </enumValue>
        </rhs>
        ...
```

C++

```
if (queue.empty()) {
  phase = ProcPhase::idle;
  ...
```

Java

```
if (queue.isEmpty()) {
  phase = ProcPhase.idle;
  ...
```

Processor internal transition function

```
<codeSnippet id="emptyQueue">
  <parameter name="queue" />
  <implementation language="java">
    queue.isEmpty()
  </implementation>
  <implementation language="c++">
    queue.empty()
  </implementation>
</codeSnippet>
```

- Reduce to a minimum the need for code rewriting

Parameterization and initialization

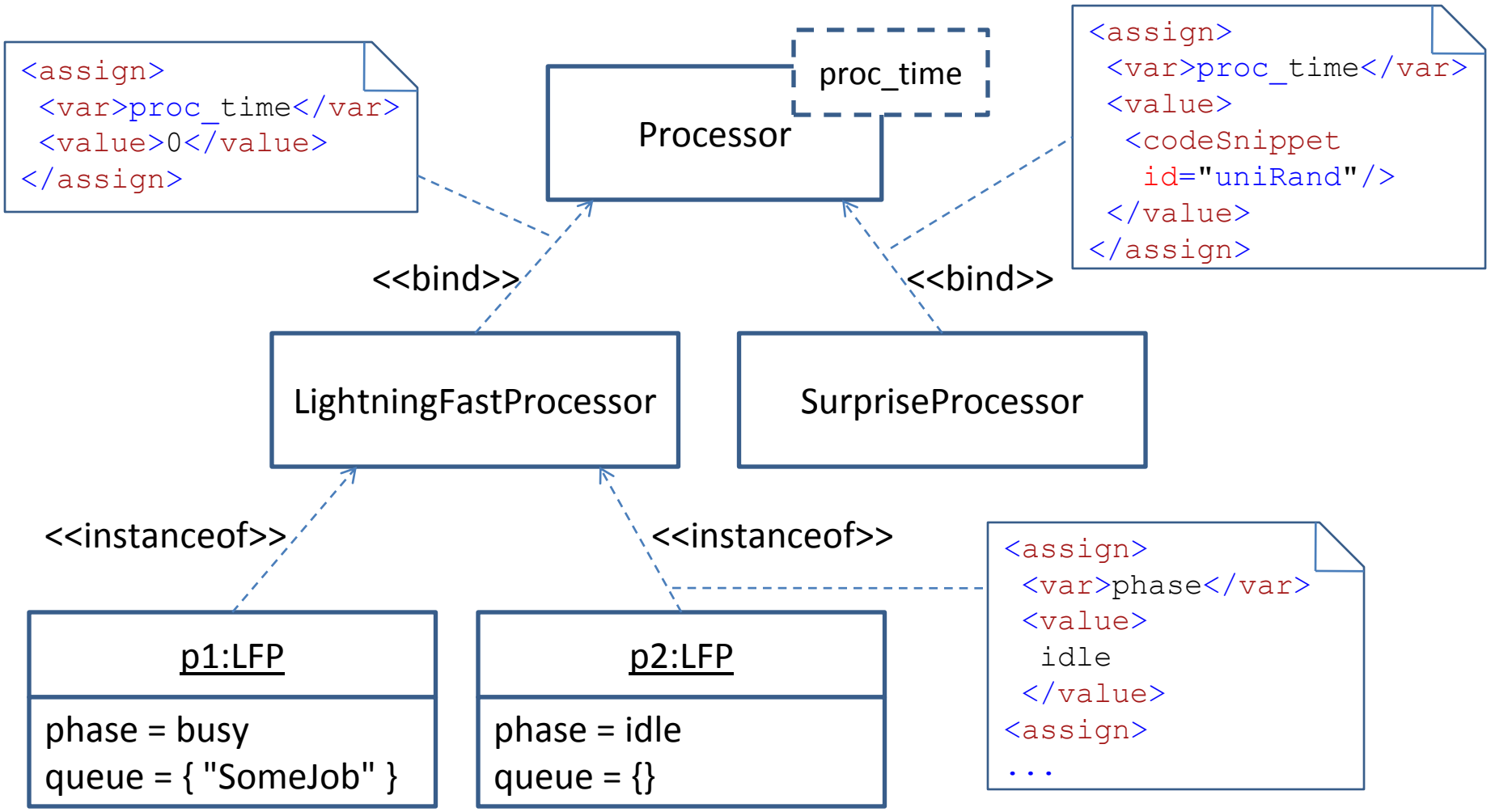
- Models are not simulatable as they are:

$$DEVS_{proc\_time} = \langle X_M, Y_M, S, \delta_{ext}, \delta_{int}, \delta_{con}, \lambda, ta \rangle$$

What value?

Initial state?

Instantiation of a parameterized model



Parameterization and initialization

- Models are not simulatable as they are:

$$DEVS_{proc\_time} = \langle X_M, Y_M, S, \delta_{ext}, \delta_{int}, \delta_{con}, \lambda, ta \rangle$$

What value? Initial state?

- Parameterization and initialization can be complex operations
  - described as functions
  - coupled models recursively instantiate theirs components

Model trace through time

- Unified representation of state, input and output trajectories
  - interoperability between simulators
  - reuse of simulation results across several tools (visualization, analysis)
- Easily achieved through XML serialization

Processor state and output trajectories

```
<state>
  <snapshot time="0">
    <phase>idle</phase>
    <queue />
    <sigma>infinity</sigma>
  </snapshot>
  <snapshot time="3">
    <phase>busy</phase>
    <queue>
      <count>1</count>
      <item>job1</item>
    </queue>
    <sigma>10</sigma>
  </snapshot>
</state>
```

```
<output>
  <ports time="13">
    <out>job1</out>
  </ports>
</output>
```

Model format: position, size,...

- Used in visualization tools
- Separated from the model
- No standard graphical language ⇒ Tool dependent

```
<coupledModel id="switchNetwork">
  <component id="switch">
    <position x="20" y="100" />
    <size height="50" width="50" />
    <color r="0" g="0" b="0" />
  </component>
  <component id="proc1">
    <position x="60" y="50" />
    ...
```

Outline

To be continued

- Development of tools using DML, tackling all the aspects of the use cases described previously
- Integration into a unified framework: SimStudio

Conclusion

- We proposed a markup language for DEVS that:
  - unify previous works
  - find a compromise between genericity and flexibility to represent model dynamics
  - takes into account all facets of model use
- Basis for a discussion towards a consensus

Conclusion

Thank you for your attention!

From source code to DML

- Conversion from source code to DML must deal with simulator specific constructs
  - ports and state variables handling (instance members? stored in a collection?)
  - special functions (holdIn, poke...)
- We need to provide a set of abstract functions, whose implementation depends on the simulator: setStateVar, isValueOnPort, getValueOnPort, etc.

XML representation of language-dependent types

- No problem with built-in or user-defined type...
- ...but with language-dependent types?
  - XML representation must be convertible to all bound types
  - Same problem as web services communication
  - There must be a common XML schema (e.g. 1 schema for both `java.lang.LinkedList<String>` and `std::queue<std::string>`)