FormalML

Formal Modeling Language

Authors

|  |  |
| --- | --- |
| **Name** | **Affiliation** |
| Mathilde Arnaud |  |
| Boutheina Bannour |  |
| Arnault Lapitre |  |
| Erwan Mahe |  |
| Xavier Zeitoun |  |

Reviewers

|  |  |
| --- | --- |
| **Name** | **Affiliation** |
| Arnaud Cuccuru |  |
| Christophe Gaston |  |
| Sébastien Gérard |  |

**Content**

[​](#_v7mkp7su07p7)

[1.​ Overview](#_5ky7s0op0t4o)

[2.​ Modeling Elements](#_iecbx9djfn9w)

[2.1.​ FormalBlock](#_8jdok9rlbh8f)

[2.2.​ FormalPart](#_jhk2y646xyim)

[2.4.​ FormalEnvironment](#_3j0btyyx3pu4)

[2.5.​ FlowPort](#_ihweqchgidpt)

[2.7.​ FormalContext](#_lw0tscn9g2k6)

[2.8.​ FormalModel](#_2q3dzvow5y6g)

[2.9.​ FlowVariable](#_k4fjowysiim0)

[2.10.​ ReceiveAnyEvent](#_z2rds414cqgc)

[2.11.​ ReceiveEvent](#_38r4h3iuo0pc)

[2.12.​ ReceiveSignalEvent](#_e0vu9t3pcp3l)

[2.13.​ TimedTransition](#_b60umbdfepyw)

[2.14.​ Buffer](#_o229pzte937y)

[2.16.​ BufferKind](#_10gt34pd19oz)

[2.15.​ Clock](#_9mu2rz5q8enf)

[2.17.​ SchedulingExpression](#_d40jie6jtwat)

[3.​ Examples](#_d0kc5iyvfawb)

[3.1.​ Water System](#_oimczfryx6zu)

[3.2.​ Simplified PhoneX](#_x4ekrnrjh5zk)

[​](#_jqcoydf3izx4)

# **​**

# 

# **1.​ Overview**

The Formal Modeling Language (FormalML for short) is a pivot language which can be used to capture a wide range of formal models. It comes with a variety of primitives which allow encoding all classical communication and execution semantics. FormalML is a UML profile [Unified Modeling Language].

# **2.​ Modeling Elements**

## **2.1.​ FormalBlock**

**Description**

A Formal Block is a communicating hierarchical entity which can be used to define systems in a modular way. A Formal Block may be atomic or reuse other Formal Blocks in its definition to form a system composition. Formal Block is associated with a scheduling model which specifies how the reused blocks executions is scheduled in the context of that Formal Block. An atomic Formal Block may be associated with a State Machine to specify its behavior in an operational manner. A Formal Block may own attributes: typed ports which are used to communicate with its environment, typed variables which are used in local computations, clocks which are used to constrain the duration of computations/communications, and buffers which are used to hold temporarily the data in transit between the reused Formal Blocks. The semantics of an atomic Formal Block is the semantics of its associated State Machine. In case when the Formal Block reuses other Formal Blocks, its semantics is obtained from the semantics of the reused blocks according the scheduling model.

**Extensions**

Class

**Generalizations**

None

**Attributes**

* schedulingExpression: [Expression](#_2bywzndtodbv)[0..1]

The specification of a complex scheduling model of the reused Formal Blocks

* routingExpression: [Expression](#_2bywzndtodbv)[0..1]

The specification of complex routing of data exchanged between reused blocks

**Semantics**

**Constraints**

1. The schedulingExpression of a Formal Block must be specified when the Formal Block contains reused blocks in its definition

**Notation**

A SchedulingExpression is defined using the following:

<scheduling-expression> ::=‘{‘ <scheduling-operator> ( <scheduling-operand> )\* ‘}’

<scheduling-operand> ::= <formal-part-name> | <behavior-name>

<scheduling-operator> ::= <inteleaving-operator> | <sequence-operator> | <parallel-operator> | <and-parallel-operator> | <or-parallel-operator>

<inteleaving-operator> ::= ‘|i|’

<sequence-operator> ::= ‘|;;|’

<parallel-operator> ::= ‘|,|’

<and-parallel-operator> ::= ‘|and|’

<or-parallel-operator> ::= ‘|or|’

where

* <formal-part-name> is ...
* <behavior-name> is ...

and

* <inteleaving-operator> is ...
* <sequence-operator> is ...
* <parallel-operator> is ...
* <and-parallel-operator> is ...
* <or-parallel-operator> is ...

A RoutingExpression is defined using the following syntax:

<routing-table> ::=@route:( <routing-line> ‘;’ )\*

<routing-line> ::= ‘route’ [ <route-id> ] ‘<’ <routing-protocol> > [ ‘[‘ <routed-element-list> | ‘\*’ ‘]’ ] [ <routing-option-list> ]

<route-id> ::= <route-name>

<routing-protocol> ::= ( [ ‘buffered:’ ] <buffer-id>) | <anonymous-buffer> | <synchronous-protocol> | <shared-memory-protocol> | <default-env-protocol>

<buffer-id> ::= <property-buffer-name>

<anonymous-buffer> ::= ( ‘fifo’ | ‘lifo’ | ‘multiset’ ) [ ‘[‘ [ ‘size:’ ] <unlimited-natural> ‘]’ ]

<synchronous-protocol> ::= ‘rdv’ | ‘multi\_rdv’

<shared-memory-protocol> ::= ‘ram’

<default-env-protocol> ::= ‘env’

<routed-element-list> ::= <routed-element> ( ‘,’ <routed-element> )\*

<routed-element> ::= <connector-name>

<routing-option-list> ::= ‘{‘ ( <routing-option> ‘;’ )\* ‘}’

<routing-option> ::= ( ‘input’ | ‘output’ ) [ <connectable-element-name> ] ‘:’ <casting-protocol>

<casting-protocol> ::= ‘unicast’ | ‘multicast’

where

* <property-buffer-name> is a declared property stereotyped by buffer
* <connectable-element-name> could be a directed port defined with a compatible direction

**Examples**

A routing table where all connector shared one anonymous unbounded fifo

@route:

route< fifo > [ \* ]

A routing table with different connector’s specific route

@route:

route< fifo > [ connector1 , connector2 ];

route< rdv > [ connector3 ];

A routing table with route option

@route:

route< fifo > [ connector4 , connector5 ] { output : multicast } // A REVOIR

TO-DO: raccourci pour dire un ensemble de connecteurs sont chacun en rdv

TO-DO: raccourci pour dire un ensemble de connecteurs ont chacun leur fifo, une source (port) commune et les outputs sont en multicast (déposé dans chacun des fifo)

## **2.2.​ FormalPart**

**Description**

A FormalPart represents the role that a reused block is assigned with, as a part of a given Formal Block definition. A FormalPart has a multiplicity which indicates the number of times the reused block will be instantiated to play that role during the FormalBlock lifetime. The lower multiplicity indicates that the number of initial instances, and the upper multiplicity indicates the maximum number of times the reused block will be instantiated including initial instances number.

**Extensions**

Property

**Generalizations**

None

**Attributes**

* Instances: InstanceSpecification[0..\*]

Represents initial instances designated by name

**Semantics**

**Constraints**

1. The initial number of instances must be less or equal to the lower multiplicity of the Formal Part.

**Notation**

**Examples**

## **2.4.​ FormalEnvironment**

## **2.5.​ FlowPort**

## **2.7.​ FormalContext**

## **2.8.​ FormalModel**

## **2.9.​ FlowVariable**

## **2.10.​ ReceiveAnyEvent**

## **2.11.​ ReceiveEvent**

## **2.12.​ ReceiveSignalEvent**

## **2.13.​ TimedTransition**

## **2.14.​ Buffer**

**Description**

A Buffer is a buffer used for to hold communication data before being consumed by Formal Blocks

**Extensions**

Property

**Attributes**

* kind:BufferKind[1]

The specification of the buffer kind as a fifo, lifo, multiset, or shared memory

## **2.16.​ BufferKind**

The BufferKind is an enumeration type which defines literals used to specify the kind of the buffer as a fifo, lifo, multiset, or shared memory.

**Generalizations**

* None

**Literals**

* SHARED\_MEM

The buffer is a one place buffer such that: when is read the stored data is consumed and replaced by a special value which signify that no value is available, when is written the already stored value is replaced by the new one.

* FIFO

The buffer respects the first in first out buffering rules

* LIFO

The buffer respects the last in first out buffering rules

* MULTI\_SET

The buffer is a multi-set

## **2.15.​ Clock**

## **2.17.​ SchedulingExpression**

# **3.​ Examples**

## **3.1.​ Water System**

## **3.2.​ Simplified PhoneX**

## **​**

**Bibliography**

## 

Bibliography

JavaScript, B.M.: Unified Modeling Language™ (UML®), http://www.omg.org/spec/uml/.