System Structure Modeling Language (S2ML)

Models of Structures, Structures of Models

Michel Batteux (IRT SystemX, France)
Tatiana Prosvirnova (IRT Saint-Exupery, France)
Antoine Rauy (IPK NTNU, Norway & Chaire Blériot-France, France)

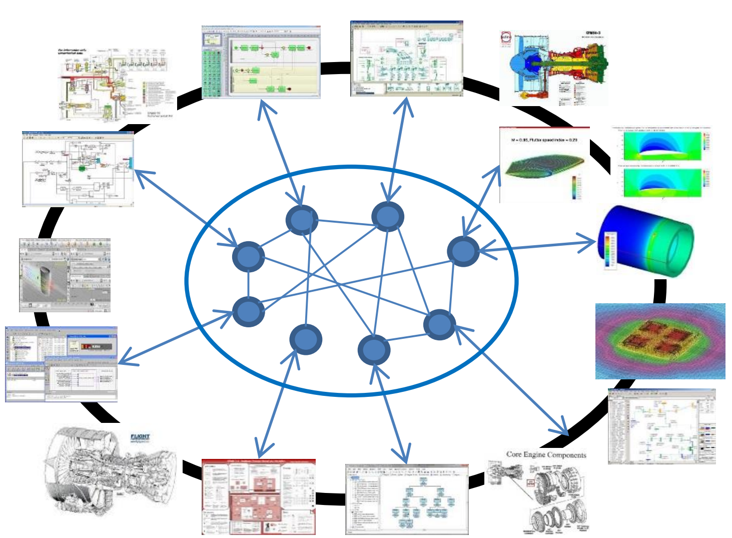
Agenda

- Rational
- Description
- Full Example
- Grammar
- Flattening
- XML Encoding
- Issues

Today's Major Challenge: Integration

Today, a major challenge of industry is to **integrate the different system engineering disciplines** (such as system architecture, control, multi-physics simulation, automatic code generation, safety and performances analyses…).

In all system engineering disciplines, there is a growing interest for the so-called **Model-Based approach** (as opposed to Document-centric approach).



The integration of system engineering disciplines goes through the integration of the artefacts, i.e. the models, they produce.

Modeling Languages

System engineering modeling formalisms and languages are made of two parts:

- An underlying **mathematical model**, that is capturing some aspect of the **behavior** of the system, e.g. differential equations for Modelica and Matlab-Simulink, Data-Flow equations for Lustre, Guarded Transition Systems for AltaRica…
- A **structuring paradigm** that makes it possible to build models by assembling parts (usually copied from libraries of reusable components) into **hierarchical descriptions**. E.g. Modelica is an object-oriented formalism. This structural part is usually flattened/compiled before the actual treatments take place.

Modeling Language = Mathematical Model + Structuring Paradigm

Model Synchronization

It would be of a great interest that the various modeling formalisms share their structuring paradigm. It would make much easier:

- Learning/training,
- Model transformation,
- Storage in collaborative data bases (PLM),
- Co-simulation (to some extent),

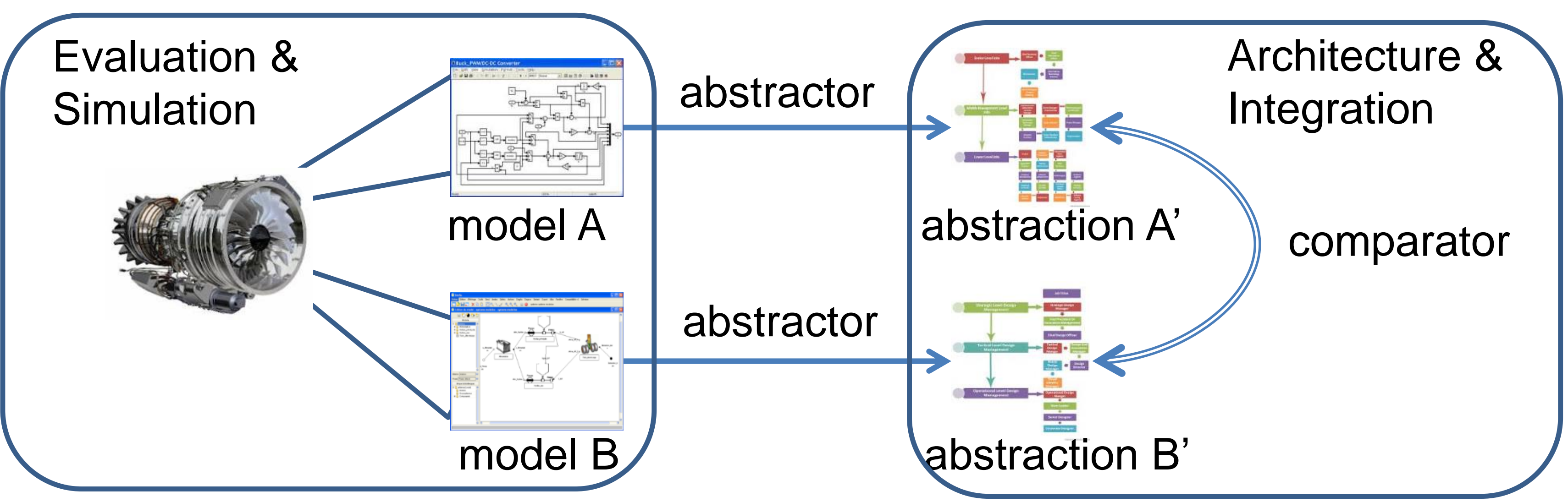
and even more importantly:

- **Model synchronization**, i.e. eventually the (preferably automated) means by which one can warranty that heterogeneous models, possibly designed by different teams with different concerns,
  - are describing the same system, and
  - are coherent.

Abstraction and Comparison

The **synchronization** of possibly heterogeneous models requires to **abstract** these models into a common framework and then to **compare** their abstractions.

Synchronisation = abstraction + comparaison



Although the choice of **abstractors** and **comparators** depends on the system and the level of maturity of the project, what should be synchronized is essentially the structural part of models.

S2ML Promise: 1) Models of Structures

S2ML aims at providing a **necessary and sufficient language** to describe the **functional and/or physical structures of systems**.

system

(representation of the) S2ML model

Describing the structure of a system is a **modeling process** that aims at architecting the system, i.e. eventually at improving the comprehension / specification of that system.

S2ML Promise: 2) Structure of Models

S2ML aims at providing a **structuring paradigm** of system engineering modeling languages.

Language A

Language B

Language C

S2ML: structuring paradigm

Mathematical framework A

Mathematical framework B

Mathematical framework C

Structuring helps to design, to debug, to share, to maintain and to synchronize models.

Why not SysML?

SysML is a graphical notation, derived from UML, to address system modeling. It provides two types of diagrams to represent structures: Definition Block Diagrams and Internal Block Diagrams(1). It could thus be a candidate formalism for our purpose. However,

- A model, which is a **mathematical object**, should not be confused with its **graphical representations**.
- Even though graphical representations are excellent supports for the **communication** amongst stakeholders, they are able to represent only **partially** the models, except for formalisms with very low (or very ambiguous) expressiveness.
- Moreover, there may be **several graphical representations** of the same concept, each more or less convenient in a given context.

(1) Parametric Diagrams and Package Diagrams cannot be used directly to represent structures, although they are considered also as structural.

Why not SysML?

In a word:

- Graphical representations are a very good communication mean. Therefore, we shall use SysML graphics and vocabulary as much as possible.
- However:

Concepts should come first

S2ML aims at proposing a minimal yet sufficient set of concepts to represent structures of systems and to structure models.

Agenda

- Rational
- Description
- Full Example
- Grammar
- Flattening
- XML Encoding
- Issues

Basic Components

S2ML is made of the following basic components.

Component	Representation	Role
Attributes	(name = value)	Attributes are used to associate information to ports, connections and blocks.
Ports		Ports are basic objects of models, e.g. variables, events, equations, transitions…
Connections		Connections are used to describe relations existing between ports.
Blocks		Blocks are containers. They can contain ports, connections and other blocks.

Example

system

tank

vessel

level
sensor 1

reactor

controller

level
sensor 2

valve 1

valve 2

Blocks as Prototypes & Composition

A block is a **container** for ports, connections and other blocks. Each block is a **prototype**: it has a unique occurrence in the model.
The block "system" **composes** the blocks "tank", "valve 1"… The block "reactor" **is part of** the block "vessel".

system

tank | valve 1 | valve 2 | controller | vessel

sensor 1 | sensor 2 | reactor

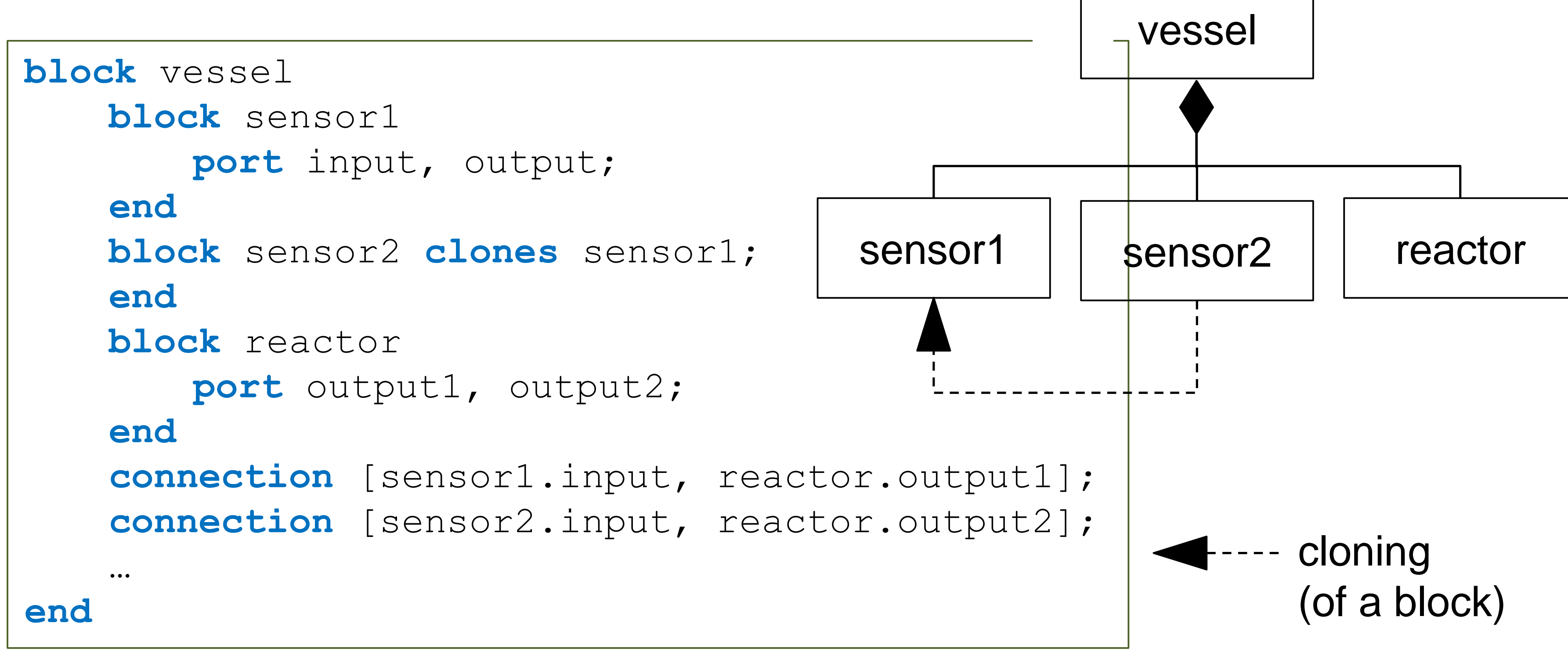
Hierarchy of nested blocks

composition

Cloning

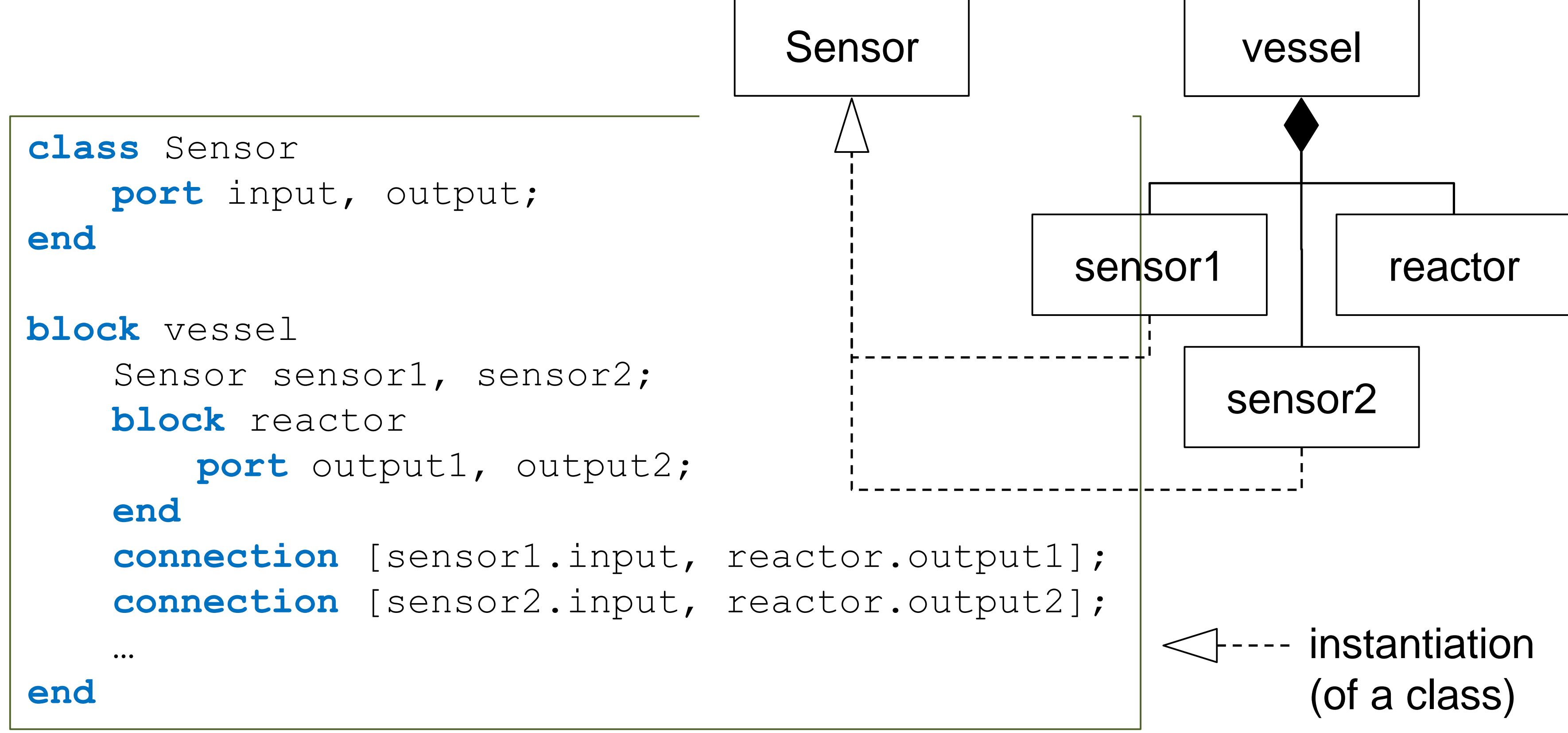
A system may contain similar components, e.g. the sensors or the valves of our example. The corresponding copy then contains several copies of the same block.

A first way to avoid duplicating the description of a block consists in **cloning** an already existing block.



Classes and Instances

A second way to avoid duplicating the description of a block consists in declaring a model of the duplicated block in a separate modeling entity, so-called a **class**, and then in **instantiating** this class.



Prototypes versus Classes

Concept space (C)

«Sandbox»

Model creation

refinement

refinement

Final model

prototypes

Reuse

Add new components

Reuse

Add new components

Knowledge space (K)

Stabilized knowledge

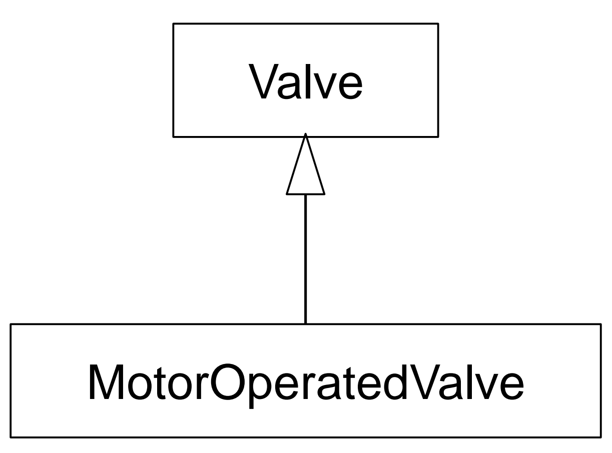
Libraries of «on-the-shelf» components

classes

Inheritance

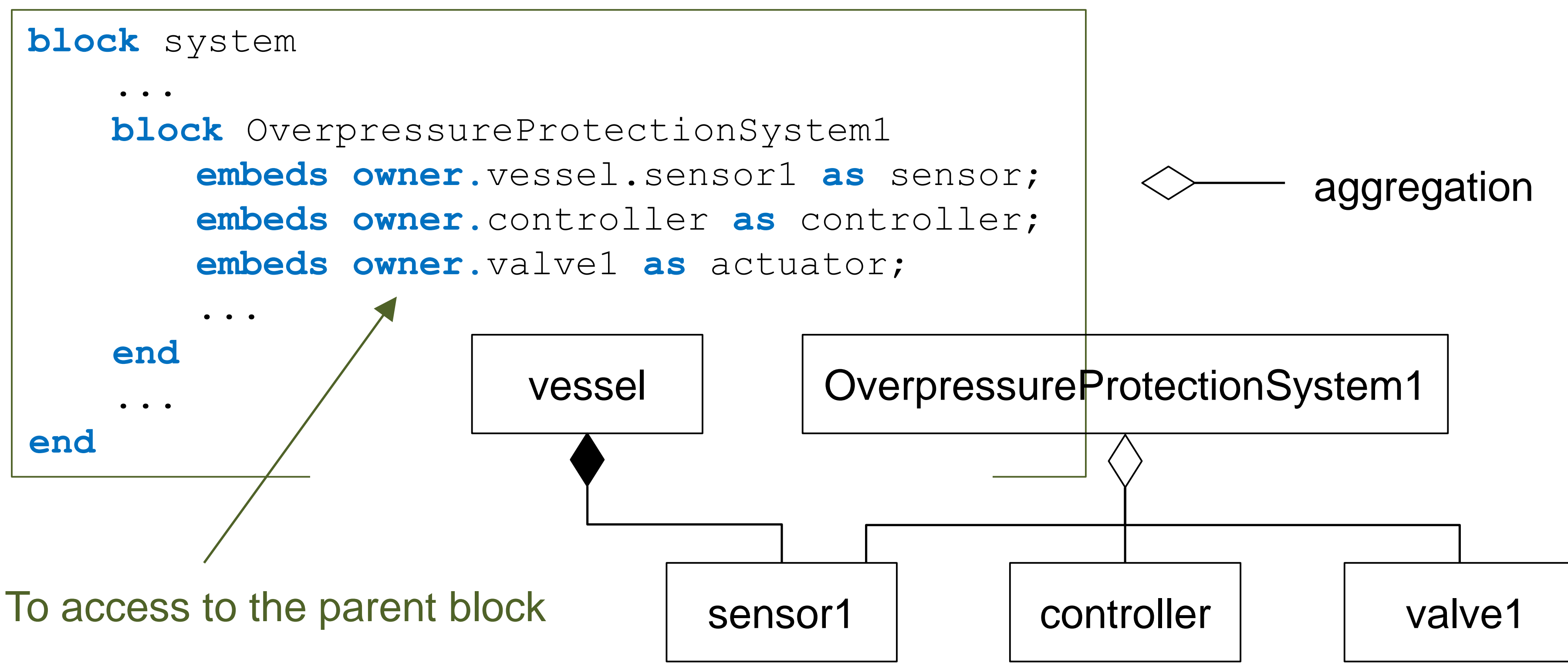
Aside the composition, that defines a "is-part-of" relation, S2ML provides also a **inheritance** mechanism, i.e. a "**is-a**" relation. A class or a block can inherit the content of another class (or another block in the same modeling entity).

```
class Valve
    port input, output;
end

class MotorOperatedValve extends Valve;
    port inputTorque;
end

block system
    ...
    block MyValve extends Valve;
        ...
    end
    ...
end
```



Aggregation & Functional Chains

S2ML provides a mechanism for blocks to **use** blocks defined elsewhere in the same modeling entity. The using block **aggregates** the used block. This mechanism is especially useful to describe the so-called **functional chains**.



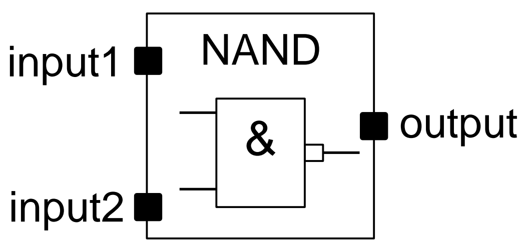
Crossing the Walls

Ports are used to describe all atomic components (on which rely the description of the behavior of the system). So there may be ports linked with no other ports or only with ports within the same block.
Also, ports can be connected with any other block of the same modeling entity (connections can cross the wall).

```
block system
    ...
    connection [valve2.output, vessel.reactor.input];
    ...
end
```

Multiple Connections and Attributes

Connections can involve more than two ports. The type of ports and connections can be described by means of attributes.




```
class CMOSNANDGate
    port input1 (type=voltage);
    port input2 (type=voltage);
    port output (type=voltage);
    connection (type=voltage, operator=NAND) [input1, input2, output];
end
```

Agenda

- Rational
- Description
- Full Example
- Grammar
- Flattening
- XML Encoding
- Issues

Example

system

tank

vessel

level sensor 1

reactor

controller

level sensor 2

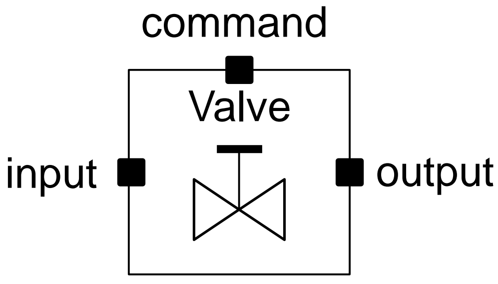
valve 1

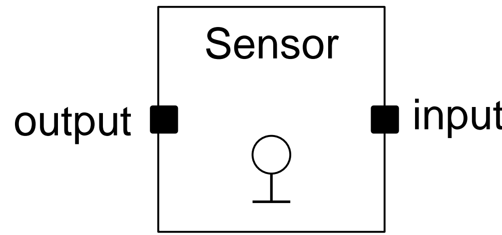
valve 2

Classes for Valves & Sensors

```
class Valve
    port input, output (type=fluid);
    port command (type=signal);
end

class Sensor
    port input, output (type=signal);
end
```

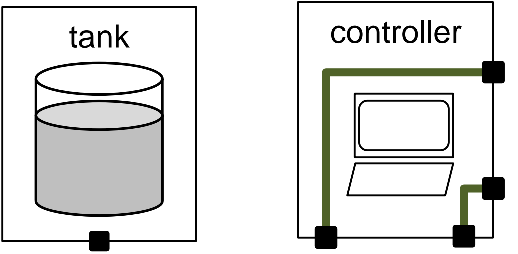
Attribute to characterize ports

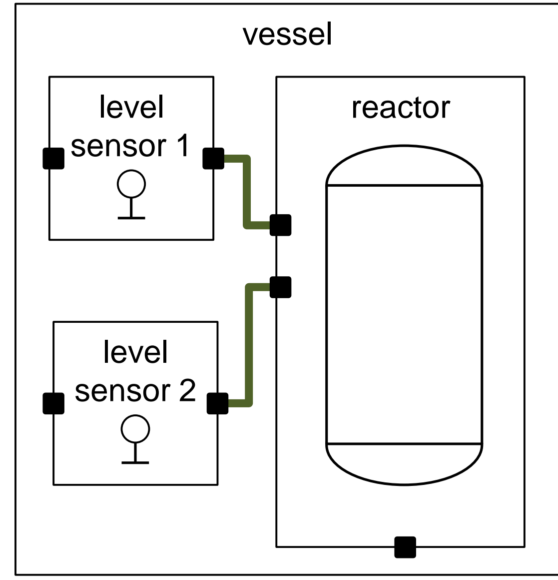




Blocks for other Components

```
block tank
    port output;
end

block controller
    port input1, input2;
    port command1, command2;
    connection [input1, command1];
    connection [input2, command2];
end

block vessel
    Sensor sensor1, sensor2;
    block reactor
        port input;
        port level1, level2;
    end
    connection [sensor1.input, reactor.level1];
    connection [sensor2.input, reactor.level2];
end
```





Blocks for Functional Chains

```
block OverpressureProtectionSystem1
    embeds owner.vessel.sensor1 as sensor;
    embeds owner.controller as controller;
    embeds owner.valve1 as actuator;
    connection (type=signal)[sensor.output,  controller.input1];
    connection (type=signal)[controller.command1, actuator.command];
end

block OverpressureProtectionSystem2
    embeds owner.vessel.sensor2 as sensor;
    embeds owner.controller as controller;
    embeds owner.valve2 as actuator;
    connection (type=signal) [sensor.output,  controller.input2];
    connection (type=signal) [controller.command1, actuator.command];
end
```

Attribute to characterize connections

Block for the system

…with attributes to characterize connections

system
tank
controller
vessel
level sensor 1
level sensor 2
reactor
valve 1
valve 2

```
block system
    block tank ... end
    block controller ... end
    Valve valve1, valve2;
    block vessel ... end
    block OverpressureProtectionSystem1 ... end
    block OverpressureProtectionSystem2 ... end
    connection (type=fluid) [tank.output, valve1.input];
    connection (type=fluid) [valve1.output, valve2.input];
    connection (type=fluid) [valve2.output, vessel.reactor.input];
end
```

Agenda

- Rational
- Description
- Full Example
- Grammar
- Flattening
- XML Encoding
- Issues

Models, Classes & Blocks

```
Model ::=
    ClassDeclaration* BlockDeclaration?

ClassDeclaration ::=
    class Identifier AttributeList?
        MemberDeclaration*
        ConnectionDeclaration*
    end

BlockDeclaration ::=
    block Identifier AttributeList?
        MemberDeclaration*
        ConnectionDeclaration*
    end

MemberDeclaration ::=
        InheritanceClause
    |   PortDeclaration | BlockDeclaration | InstanceDeclaration
    |   EmbedsClause
```

Members & Connections

```
InheritanceClause ::=
        extends Path (, Path)* AttributeList? ;
    |   clones Path (, Path)* AttributeList? ;

PortDeclaration ::=
    port Identifier (, Identifier )* AttributeList? ;

InstanceDeclaration ::=
    Path Identifier (, Identifier )* AttributeList? ;

EmbedsClause ::=
    embeds Path as Identifier ;

ConnectionDeclaration ::=
    connection Identifier? AttributeList? ArgumentList (, ArgumentList )* ;
```

Attributes, Arguments & Identifiers

```
AttributeList ::=
     ( Attribute (, Attribute)* )

Attribute ::=
      Path  = ( Path | Text )

ArgumentList ::=
     [ Path (, Path)* ]

Path ::=
         owner . Path
     |   Identifier (. Path)?

Identifier ::=
     [a-zA-Z][a-zA-Z0-9_]*

Text ::=
     "([^\"] | \\ | \").*"
```

Agenda

- Rational
- Description
- Full Example
- Grammar
- Flattening
- XML Encoding
- Issues

Flattening: Composition

Although the structure of the model plays a very important role into its clarity, it normally plays no role in the behavioral description of the system. Any hierarchical model can be flattened into an equivalent non-hierarchical one. The flattening process depends on the modeling language. It can usually be defined by means of the so-called flattening rules that are applied bottom-up.

For S2ML, flattening rules are indeed purely structural.

<u>Composition Flattening:</u>

```
block A
    block B
        port p, q;
        connection [x, y];
    end
end
```

⇒

```
block A
    port B.p, B.q;
    connection [B.x, B.y];
end
```

This rule applies the same way to class declarations (class A…).
Other flattening rules are derived from that one.

Flattening: Instantiation

Instantiation Flattening:

```
class C
    port p, q;
    connection [x, y];
end

block A
    C B;
end
```

⇨

```
block A
    port B.p, B.q;
    connection [B.x, B.y];
end
```

This rule applies the same way to class declarations (class A…).

Flattening: Inheritance & Cloning

Inheritance Flattening:

```
class C
    port p, q;
    connection [x, y];
end

block A
    extends C;
end
```

⇨

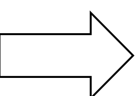
```
block A
    port p, q;
    connection [x, y];
end
```

This rule applies the same way to class declarations (class A…) and to cloning of blocks.

Flattening: Aggregation

<u>Aggregation Flattening:</u>

```
block R
    port x;
end

block A
    embeds owner.R as B;
    port y;
    connection [B.x, y];
end
```



```
block A
    port y;
    connection [R.x, y];
end
```

Agenda

- Rational
- Description
- Full Example
- Grammar
- Flattening
- XML Encoding
- Issues

XML Representation

It is sometimes better to use a **XML based representation** rather than the textual/abstract grammar presented before (to simplify parsing issues).

The XML representation follows Open-PSA principles:

- The declaration of a named element of type “xxx” is introduced by means of a XML tag “declare-xxx”.
- References to a named element of type “xxx” are introduced by means of a XML tag “xxx”.

It would be possible to write a full XSD Schema for S2ML. Here just follows some indications about how to encode S2ML constructs in XML. We give a grammar in Bacckus Naur form (as previously).

Models, Blocks & Classes

```
Model ::=
    <declare-model name="Identifier" >
        Attribute* ClassDeclaration* BlockDeclaration?
    </declare-model>

ClassDeclaration ::=
    <declare-class name="Identifier" >
        Attribute* MemberDeclaration* ConnectionDeclaration*
    </declare-class>

BlockDeclaration ::=
    <declare-block name="Identifier" >
        Attribute* MemberDeclaration* ConnectionDeclaration*
    </declare-block>
```

Members

```
MemberDeclaration ::=
        InheritanceClause
    |   PortDeclaration | BlockDeclaration | InstanceDeclaration
    |   EmbedsClause

InheritancClause ::=
        <extends class="Path" />
    |   <clones block="Path" />

PortDeclaration ::=
    <declare-port name="Identifier" >
        Attribute*
    </declare-port>
```

Members

```
InstanceDeclaration ::=
    <instance class="Path" name="Identifier" >
        Attribute*
    </instance>

EmbedsClause ::=
    <embeds block="Path" name="Identifier" />
    <embeds instance="Path" name="Identifier" />

ConnectionDeclaration ::=
    <connection name="Identifier" >
        Attribute* PortReference+
    </connection>

PortReference ::=
    <port name="Path" />
```

Attributes & Identifiers

```
Attribute ::=
    <attribute name="Path" value="Text" />

Path ::=
        owner Path
    |   Identifier (. Path)?

Identifier ::=
    [a-zA-Z][a-zA-Z0-9_]*

Text ::=
     ([^\"] | \\ | \").*
```

Agenda

- Rational
- Description
- Full Example
- Grammar
- Flattening
- XML Encoding
- Issues

Issues

There remains a number of issues not solved/discussed yet.

- Do we need to introduce **parameters** in addition to ports?

  Parameters are constants that get a default value but that can be modified at instantiation (or cloning) time.

- Do we need to introduce **universal objects**?

  Universal objects would be typically blocks defined aside the main block to store universal constants (such as the gravity).

- Do we need to introduce a **substitution mechanism** and/or **parametric classes**?

  A substitution mechanism would typically replace a sub-block (or class instance) by another one when instantiating a class (or cloning a block).

  Parametric classes would typically contain “holders” i.e. predefined place to be filled with blocks or class instances at instantiation (or cloning).

APPENDIX