

SysML: Systems Modeling Language

Jean-Michel Bruel
bruel@iut-blagnac.fr

SysML-France
sysmlfrance@gmail.com

February 16, 2012

Organization

Who am I?

- Professor at U. of Toulouse
- Co-founder of the [SysML-France](#) association
- Member of the [Software and System Modeling](#) journal editorial board
- Co-lead of the Ambient Systems Research group at [IRIT](#)
- Head of the Computer Science Department of my institute
- Married, one child
- Not a native English speaker (I know, neither do you!)

Who am I?

- Professor at U. of Toulouse
- Co-founder of the SysML-France association
- Member of the Software and System Modeling journal editorial board
- Co-lead of the Ambient Systems Research group at IRIT
- Head of the Computer Science Department of my institute
- Married, one child
- Not a native English speaker (I know, neither do you!)

Who am I?

- Professor at U. of Toulouse
- Co-founder of the [SysML-France](#) association
- Member of the [Software and System Modeling](#) journal editorial board
- Co-lead of the Ambient Systems Research group at [IRIT](#)
- Head of the Computer Science Department of my institute
- Married, one child
- Not a native English speaker (I know, neither do you!)

Who am I?

- Professor at U. of Toulouse
- Co-founder of the [SysML-France](#) association
- Member of the [Software and System Modeling](#) journal editorial board
- Co-lead of the Ambient Systems Research group at [IRIT](#)
- Head of the Computer Science Department of my institute
- Married, one child
- Not a native English speaker (I know, neither do you!)

Who am I?

- Professor at U. of Toulouse
- Co-founder of the [SysML-France](#) association
- Member of the [Software and System Modeling](#) journal editorial board
- Co-lead of the Ambient Systems Research group at [IRIT](#)
- Head of the Computer Science Department of my institute
- Married, one child
- Not a native English speaker (I know, neither do you!)

About this course

I have already taught this course in various occasions:

- Master Internet, U. of Pau, France (Introduction, with my colleague Nicolas Belloir)
- Research Master SAID, U. of Toulouse 3, France (3h Introduction)
- Professional Master ICE, U. of Toulouse 2, France (3h with my colleague Pierre de Saqui Sannes)
- M.Sc. Gotteborg, Sweden (Introduction by my colleague Nicolas Belloir)
- Universidad Autónoma de Guadalajara, Mexico (40h course for employees of CONTINENTAL)

Evaluation

- Team work (2 people)
- Topic of your choice (possibly from other courses)
- I provide template for report redaction (to be discussed)
- Using modern development tools (SVN, Git, asciidoc, \LaTeX , ... to be discussed)

Overview

About the course itself:

- 40 hours
- 2 (big!) weeks duration
- evaluation: final exam and project evaluation
- teaching in English
- first time for me in UAG: be patient ;-)

Overview (ctd.)

- Goals:
- Mastering the SysML notation (main diagrams)
 - Know all the possibilities of modelization offered by SysML
 - Practice with existing open source (e.g., Papyrus)

- Organization:
- Slides (**PDF*** version)
 - Course support (**PDF*** version advised because of the dynamic links)

- Pre-requisite:
- Modelisation
 - ECLIPSE environment

- Program:
- Overview
 - Statical aspects
 - Dynamical aspects
 - A complete case study: the Distiller Controller
 - Project: the Pace Maker



Conclusion

Introduction

Overview

Organisation

- 40h
- 11 sessions + 1 revision + 1 project/final exam
- sessions:
 - 1 Introduction and overview (3h – Feb, 4th, UAG)
 - 2 Tools and first practical work (3h – Feb, 4th, UAG)
 - 3 Quizz, End of introduction, Case studies (3h – Feb, 7th, UAG)
 - 4 Organization and Requirements (3h – Feb, 8th, Continental)
 - 5 Block and Internal Block Definition Diagrams (3h – Feb, 9th)
 - 6 Dynamic aspects: uc, seq, stm (6h – Feb, 11th)
 - 7 Dynamic aspects: act (3h – Feb, 13th)
 - 8 (3h – Feb, 14th) Documentation
 - 9 (3h – Feb, 15th) Animation / Simulation
 - 10 (2h – Feb, 16th) Q/A, Revisions, Course evaluation
 - 11 (6h – Feb, 18th) Project Defence & Exam

Interesting links

- A wiki with lots of examples:

<http://www.omgwiki.org/MBSE/doku.php>

- The **OMG*** web site: <http://www.omgsysml.org/>

- The specification itself:

<http://www.omg.org/spec/SysML/1.2/PDF>

- <http://www.sysml-france.org>

- <http://www.eclipse.org/modeling/mdt/papyrus/>

- <http://www.artisansw.com/>

- <http://www.papyrusuml.org/>



<http://www-01.ibm.com/software/rational/products/rhapsody/>



System Engineering

Modeling systems

Difference with SE

- Not Software Engineering ...
- ... Before Software Engineering!
 - Historically
 - In the development process

Difference with SE

- Not Software Engineering ...
- ... Before Software Engineering!
 - Historically
 - In the development process

A Complex System

- Set of human and material elements composed of various technologies
 - Computer, Hydraulic, Electronic,?
- Integrated to provide services to its environment corresponding to the system finality
- Interacting between themselves and the environment

Note

A **complex system** is very different from a simple software system

A Complex System

- Set of human and material elements composed of various technologies
 - Computer, Hydraulic, Electronic,?
- Integrated to provide services to its environment corresponding to the system finality
- Interacting between themselves and the environment

Note

A **complex system** is very different from a simple software system

A Complex System

- Set of human and material elements composed of various technologies
 - Computer, Hydraulic, Electronic,?
- Integrated to provide services to its environment corresponding to the system finality
- Interacting between themselves and the environment

Note

A **complex system** is very different from a simple software system

A Complex System

- Set of human and material elements composed of various technologies
 - Computer, Hydraulic, Electronic,?
- Integrated to provide services to its environment corresponding to the system finality
- Interacting between themselves and the environment

Note

A **complex system** is very different from a simple software system

Systems of Systems

A system:

- Should manage interactions between parts
- Support expected behavior
- Handle unexpected ones

Systems of Systems

A system:

- Should manage interactions between parts
- Support expected behavior
- Handle unexpected ones

Systems of Systems

A system:

- Should manage interactions between parts
- Support expected behavior
- Handle unexpected ones

Introduction to SysML

Overview

Interesting links

- The official doc: <http://www.omg.org/spec/SysML/1.2/>
- A wiki with lots of examples:
<http://www.omgwiki.org/MBSE/doku.php>
- The **OMG*** web site: <http://www.omgsysml.org/>
- The specification itself:
<http://www.omg.org/spec/SysML/1.2/PDF>
- The French SysML association: <http://www.sysml-france.org>

Basic Principles

Global approach

In order to help you, we will use the following table through this course:

	Requirements	Structure	Behavior	Crosscutting
Organisation				
Analysis				
Design				
Implementation				

Organisation

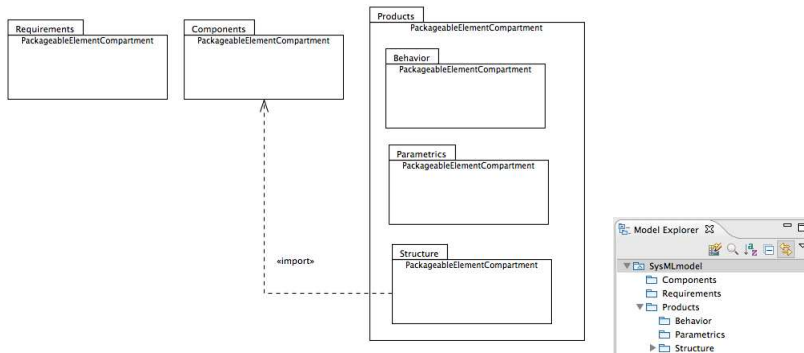
About this section

	Requirements	Structure	Behavior	Crosscutting
Organisation				
Analysis				
Design				
Implementation				

In the Organisation phase we are going to deal with **organisation** (e.g., using Packages). We will see the following:

- The Package Diagram
- Different types of packages
- Possible organisations
- Namespaces
- Dependencies

The Package diagram



Types of packages

The most significant types of packages used to organize models in SysML are [4]:

models: a top-level package in a nested package hierarchy

packages: a container for other model elements

model librairies: a package intended to be reused (imported) by others

views: a special kind of package to illustrate viewpoints

Possible organisation

- By system hierarchy (system, subsystems, components, ...)
- By process life cycle (reqs, analysis, ...)
- By teams (architects, IPT^{*}, ...)
- By type of models (Req, behavior, ...)
- By combination of the preceeding

Package as Namespaces

A package is a **namespace** for all named elements within it.

Usefull

You can ask your favorite tool to show **Qualified names**, that is name of the model element prefixed by its packages (e.g.,
Structure::Products::Clock)

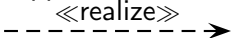
Dependencies between Packageable Elements

Several **dependencies** can occur between elements in and between packages:

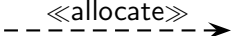
- Dependency:**
- just the dashed arrow, no specific identification of the dependency
- ----->
- Use:**
- the client use the supplier (as a Type of example)
- <<use>>
• ----->
- Refine:**
- the client is a refinement (more details) of the supplier
- <<refine>>
• ----->

Dependencies between Packageable Elements (ctd.)

Realization:

- the client is a realization (implementation) of the supplier
- 

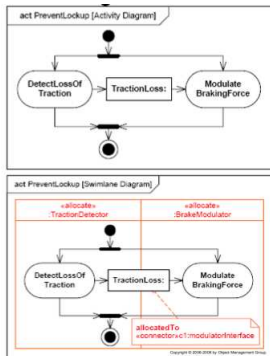
Allocation:

- the client (e.g., an activity or a requirement) is allocated on the supplier (a block most of the time)
- 

Delegation

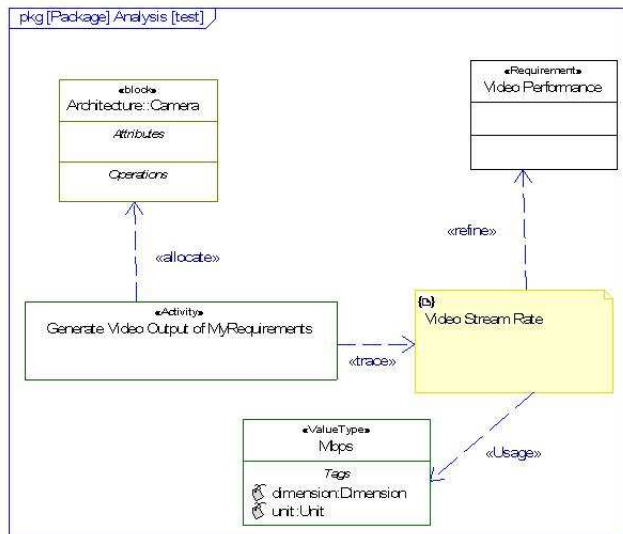
- General relationship between two elements of the model
- Different kinds of allocation:
 - Functionality – Component
 - Logical component – Physical component
 - Software – hardware
 - ...
- Usable in a lot of different diagrams
- Usable under graphical or tabular representation

Delegation: example with swimlanes



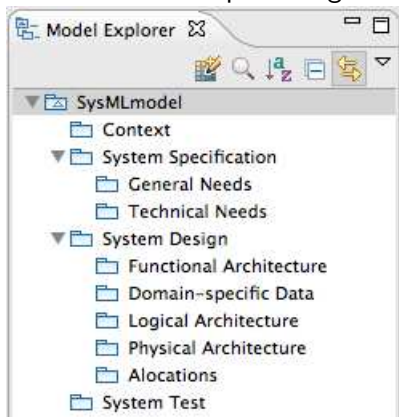
Dependencies: full example

Here is a full example (taken from [4, p. 90])



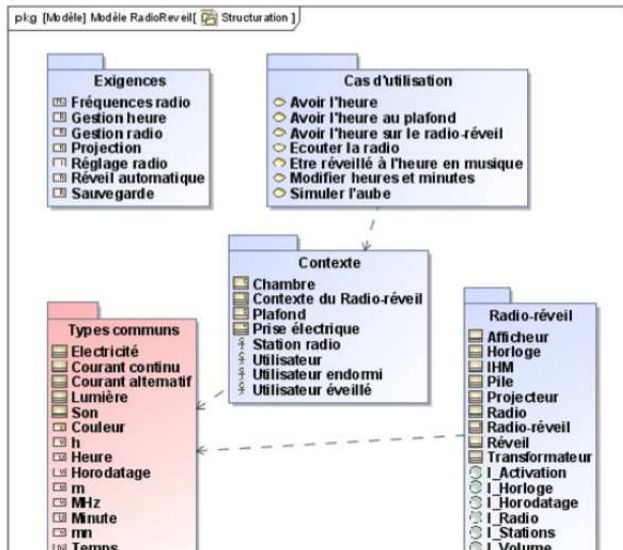
Example 1: PaceMaker

Here is a an example of organization [2] :



Example 2: RadioClock

Here is a an example of organization from [6] :



Requirements

About this section

	Requirements	Structure	Behavior	Crosscutting
Organisation				
Analysis				
Design				
Implementation				

In the Requirements phase we are dealing with the requirements of the system. We will see the following:

- Requirements organization
- Requirements properties
- Requirements links
- Traceability considerations

Requirements organization

Here are some examples:

- taken from [2] (see also section 22):
 - General needs (links with use cases)
 - Technical needs (links with design elements)
- taken from [1]:
 - Main requirements (linked with use cases)
 - Typical groups of requirements:
 - Marketing reqs
 - Functional reqs
 - Environmental reqs
 - Business reqs
 - ...

Requirements properties

Requirements can have several properties [6]:

- priority (high, low, ...)
- source (stakeholder, law, technical, ...)
- risk (high, low, ...)
- status (proposed, approved, ...)
- verification method (analysis, tests, ...)

Requirements links

Containment: for decomposition ($\oplus -$)

Refinement: for adding precision ($\ll \text{refine} \gg$)

Derivation: for different abstraction level ($\ll \text{deriveReq} \gg$)

Traceability considerations

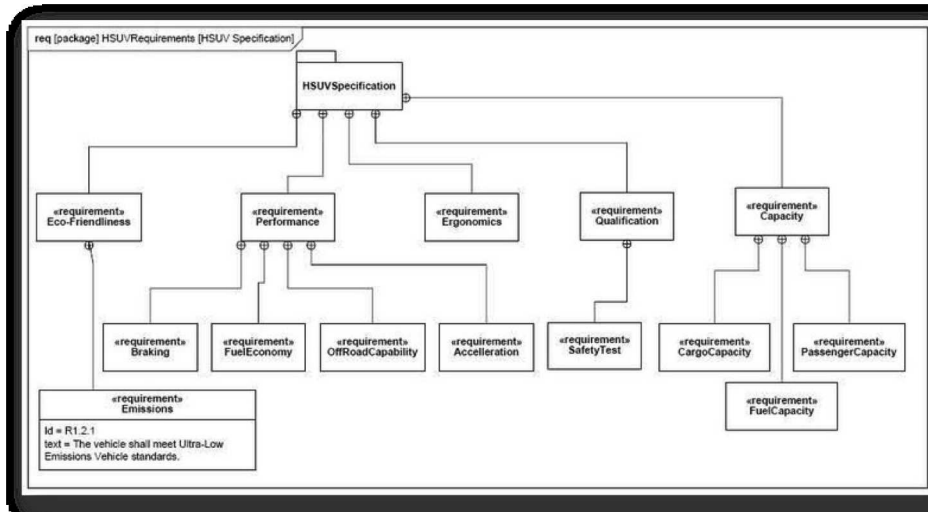
Once requirements have been organized they need to be linked to at least use cases (using «refine» for example) and to structural elements (using «satisfy» for example).

Note

Each **requirement** should be linked to at least one **use case** (and vice-versa!).

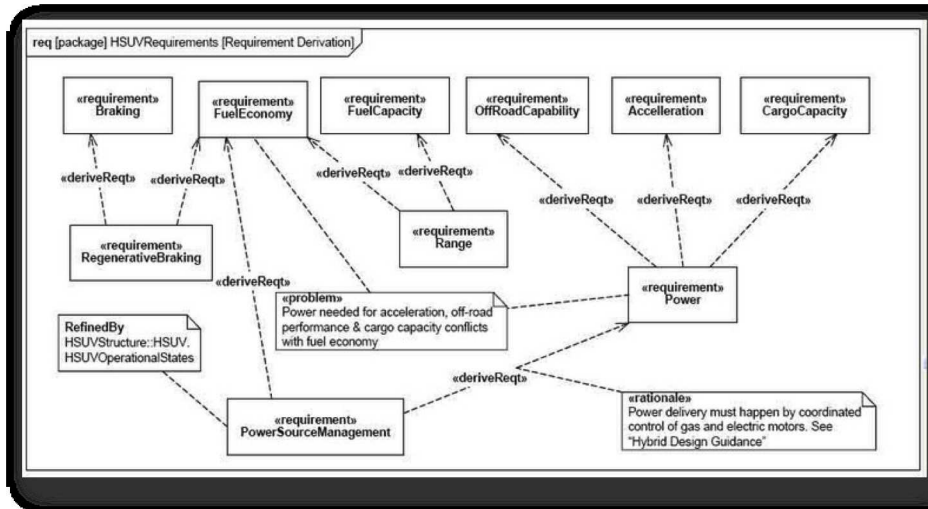
Example 1: HSUV

Here is an example of requirement diagram taken from
<http://www.uml-sysml.org/sysml> :



Example 2: HSUV

Here is another example of requirement diagram taken from <http://www.uml-sysml.org/sysml> :



To summarize

	Requirements	Structure	Behavior	Crosscutting
Organisation	\oplus -, $\ll\text{deriveRqt}\gg$			
Analysis	$\ll\text{satisfy}\gg$ between reqs and UC			
Design	$\ll\text{allocate}\gg$			
Implementation	$\ll\text{satisfy}\gg$ $\ll\text{verify}\gg$			

Structure

About this section

	Requirements	Structure	Behavior	Crosscutting
Organisation				
Analysis				
Design				
Implementation				

In the Structure phase we are dealing with the structural aspects of the system. We will see the following:

- Structure organization
- Block Definition Diagrams
- Internal Block Diagrams
- Parametric Diagrams

Structure organization

The **package** concept feeds the need for the organization of the structure of the system.

Most of the time a **context** block definition diagram provide an overall organization definition.

Block Definition Diagrams

A **bdd** can represent:

- a package
- a block
- a constraint block

Blocks Properties

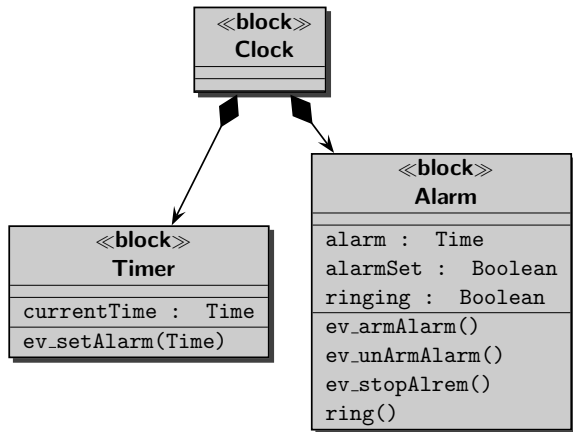
The main properties of a block are:

parts: the elements that compose the block

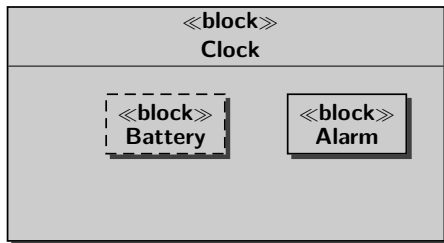
values: its (quantifiable) characteristics

references: the elements the block has access to

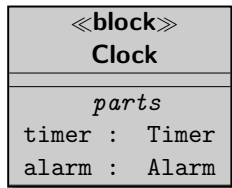
Parts: composition view



Parts: internal view



Parts: bloc details view



Properties details

Properties can have:

multiplicity: the number of parts for example (0 meaning optional for example)

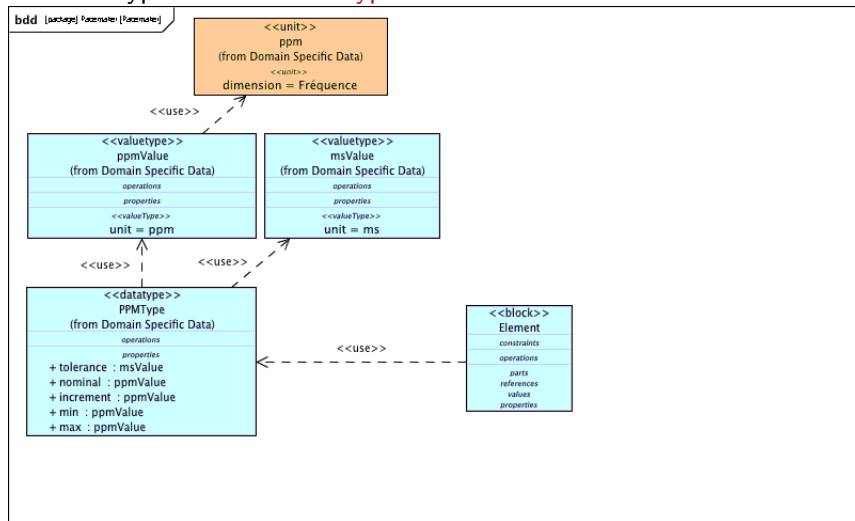
type: for values

name: also called role name for parts

initial value: the default value of an attribute

Values and Value Types

Quantitative **values** are so commonly used that there is a special kind of block to type them: **Value Types**.



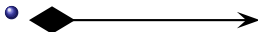
References

Reference properties indicate access to external blocks (removable parts in some sense).

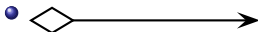
Blocks Associations

The main relationship between blocks are:

composition: • the elements that compose the block (parts)

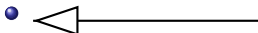


aggregation: • the elements the block has access to (references)



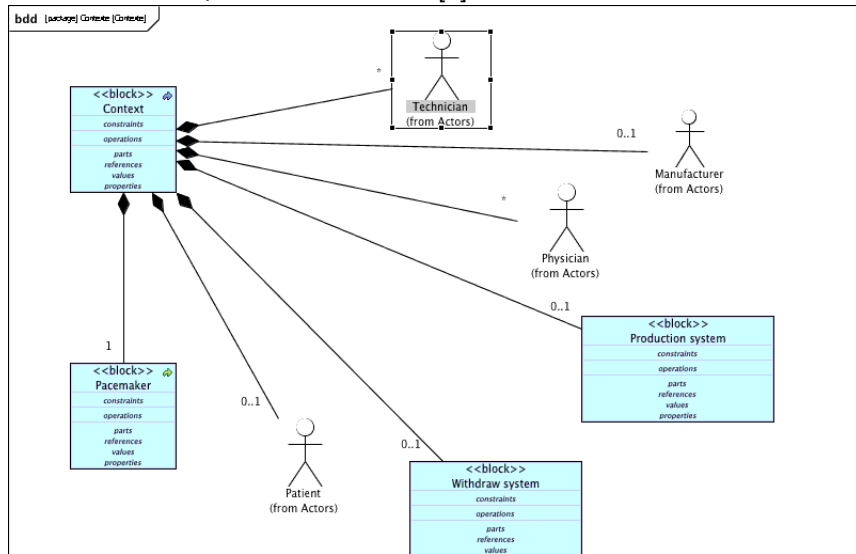
association: just link between blocks

generalisation: • the client is a special kind of provider



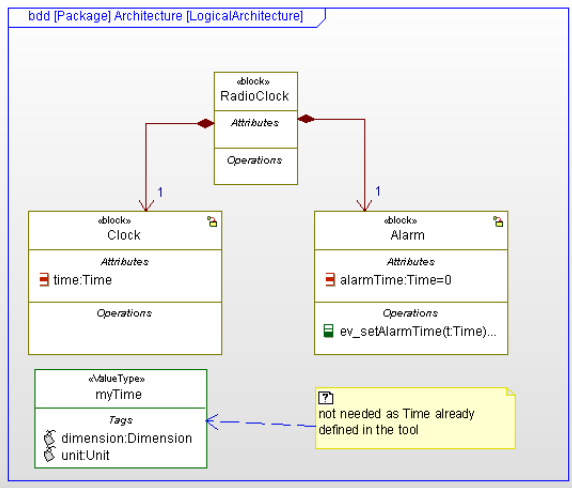
Example 1: PaceMaker

Here is an example of context bdd [2] :



Example 2: RadioClock

Here is an example of block with parts, references and value types :



Value Types

A value type is defined by a name and can have additional properties:

dimension: the element that is measured (e.g., Mass, Time, Length)

unit: the unit used for measurement (e.g., kilogram, pounds, meters)

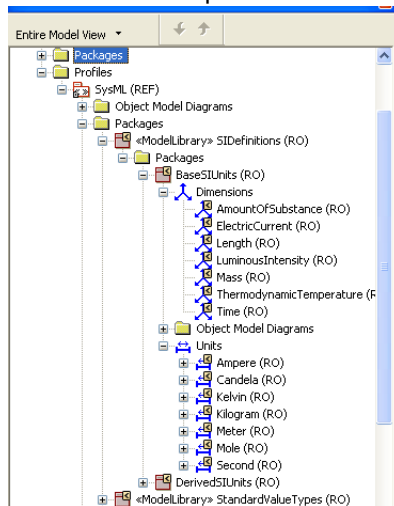
probability distribution: (see [5, Appendix C])

Note

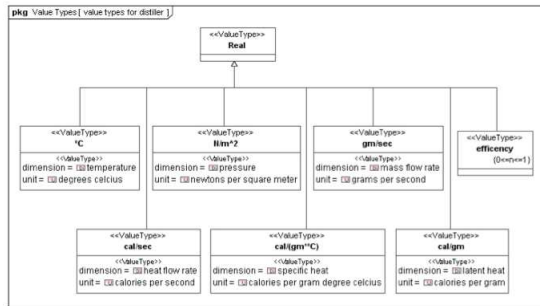
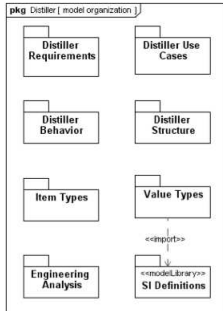
A good modeling tool provides International System of Units definitions.

Example 1: SI in Papyrus

Here is an example of use of SI values:



Example 2: HSUV



Internal Block Diagrams

An **ibd** shows the relations between parts of a block.

The important notions in a ibd are:

Items (that flow): matter or information. It can be Blocks, Value Types or Signals. Can be associated to properties.

Flow port: the entry/exit point through which the item flow in or out the block

Connectors: the link between two (or more) ports


Flow Items


- They can be Blocks (complex data), Value Types (like water) or Signals (information control).
- Can be associated to a property (of the enclosing block).

Flow Ports

They are interaction point for flow items.

There are different kind of ports:

Atomic Flow Ports:  only one type of input/output (e.g., electricity plug)

Nonatomic Flow ports:  several flow items. Need a full description of the flow properties (e.g., USB port).

Connectors

There are different kind of connectors (link between ports):

Delegation connector: between a block port and a port of one of its part

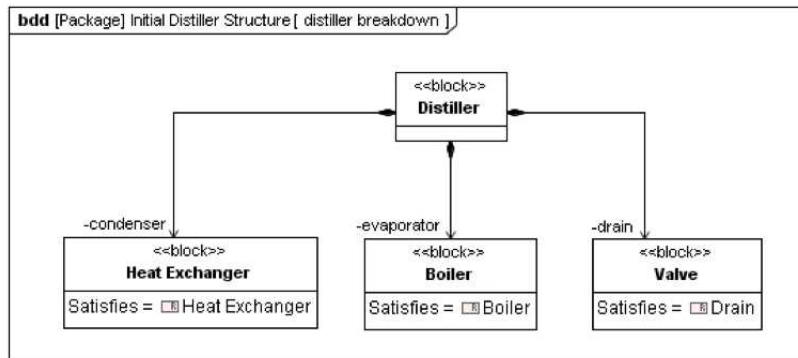
Assembly connector: between parts' ports

A connector can be typed by an association (e.g., “wireless”, “ADSL”)

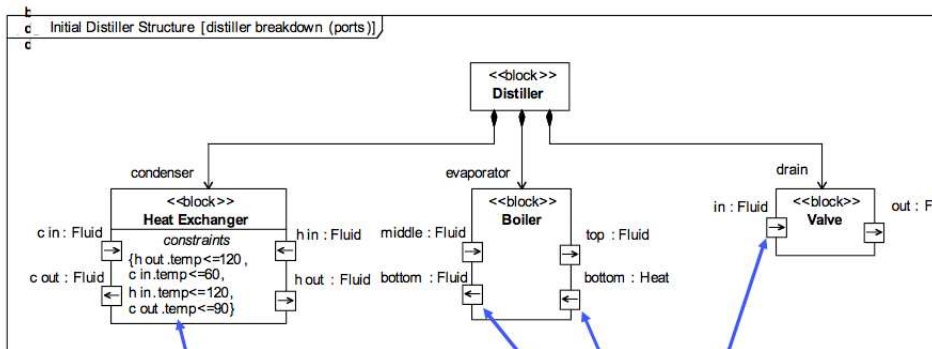
Note

Delegation is not mandatory. An external port that has no delegation is called a **behavior port**. The delegation mechanism is described by the behavioral models of the block.

Example 1: Distiller (bdd v1)



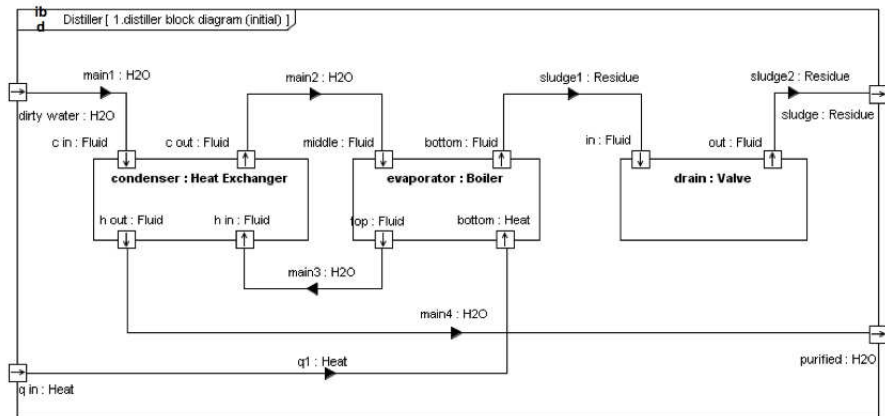
Example 1: Distiller (bdd v2)



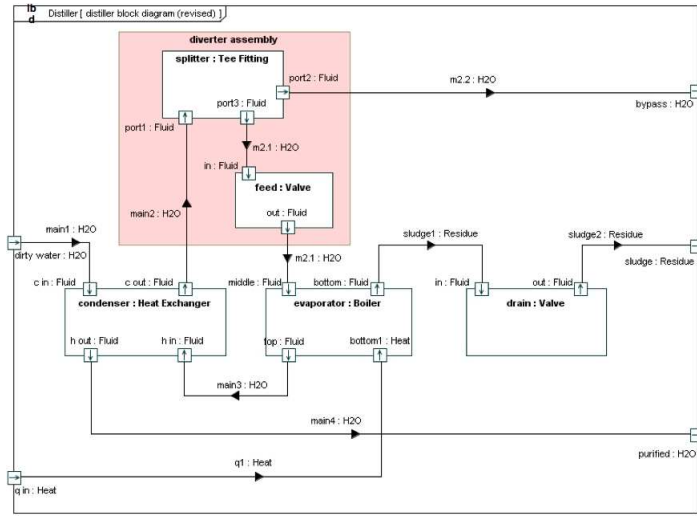
Constraints
(on Ports)

Flow Ports
(typed by things that
flow)

Example 1: Distiller (ibd v1)



Example 1: Distiller (ibd v2)



Parametric diagrams concepts

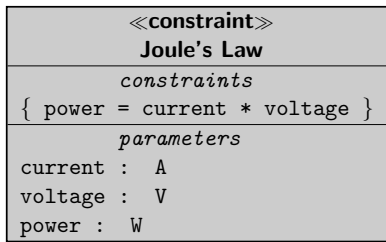
This diagram is a nice way to express **constraints** to the system. The main concepts are:

- Constraints (a kind of block)
- Parametric diagram (a kind of ibd)
- Value binding

Constraints

A constraint is a special block with:

- the stereotype `«constraint»` instead of block.
- parameters
- some relations linking those parameters



Parametric diagram

It inherits from the Internal Block Definition diagram with some differences:

- The constraints are shown with boxes with round corner
- The “ports” are in fact parameters of the constraint

Note

As there is no ambiguity between parts and constraints, the stereotype is optional.

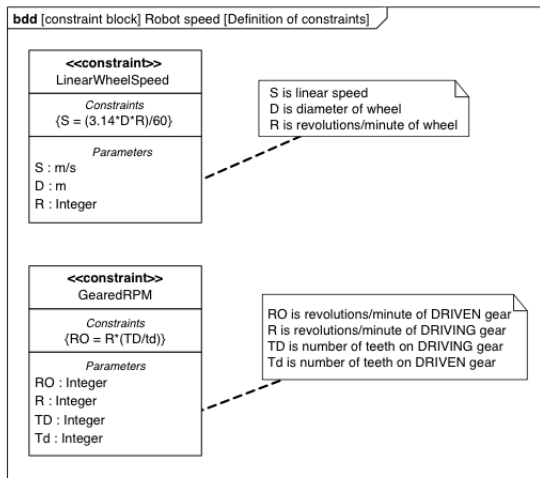
Value binding

After expressing the constraint properties one need to link the parameters to actual values. This is called **value biding**.

For showing specific values, it is necessary to use **Block Configurations** where explicit values are given to parameters.

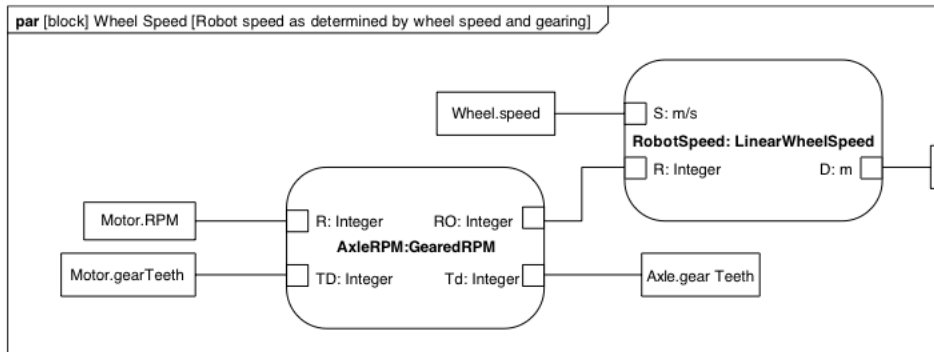
Example: Constraints

Here is an example taken from [3]:



Example: Parametrics

Also from [3]:



To summarize

	Requirements	Structure	Behavior	Crosscutting
Organisation		Packages		
Analysis		Context		
Design		bdd, ibd		
Implementation		par		

Dynamic aspects

About this section

	Requirements	Structure	Behavior	Crosscutting
Organisation				
Analysis				
Design				
Implementation				

In this phase we are going to deal with **behavior**. We will see the following:


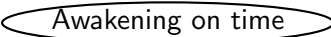
- Use Case Diagrams
- Sequence Diagrams
- State Machines
- Activity Diagrams

Use Case Diagrams

Use cases represent the way a system is used. We can have Use Case diagrams at different level of abstraction:

	Requirements	Structure	Behavior	Crosscutting
Organisation			UC of the context	
Analysis			UC \iff Requirements	
Design			UC \iff Sequence Diagrams	
Implementation				

Use Case Diagrams: basic concepts

- Users ()
User
- Use Cases ()  Awakening on time

Note

Actors have to be seen as **roles**.

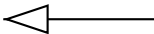


Relation between Use Cases

A use case is fully described by some properties:

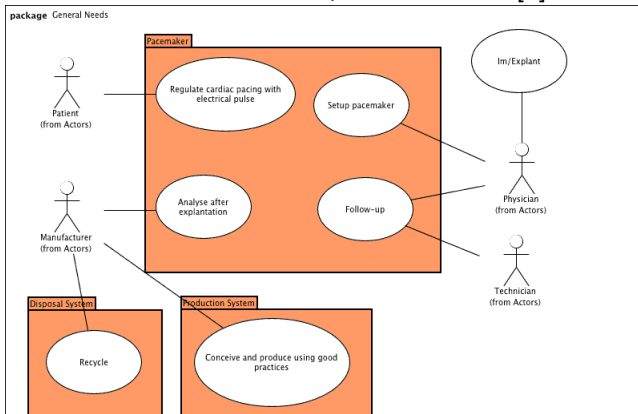
- Name
- Associated requirements
- Preconditions
- Postconditions
- Trigger
- ...

Relation between Use Cases

- «extend» to show an optional part of a use case
- «include» to show a (most of the time reused) part of a use case
-  to show specialisation/generalisation

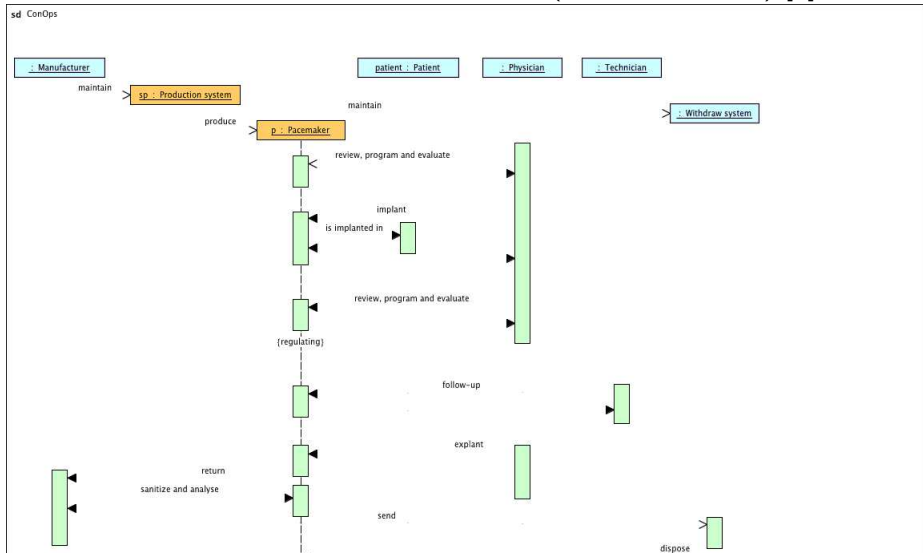
Example 1: PaceMaker uc

Here is a PaceMaker example taken from [2]:



Example 2: PaceMaker seq

Here is the corresponding sequence diagram (nominal scenario) [2]:



Sequence diagrams

Very usefull to describe scenarios. A sequence diagram represents an **interaction**. The elements in interaction are the members of the owning block. The main concepts are:

- lifeline**: a rectangle (the head) and a dashed line descending from its base

- messages**: between lifelines

- execution**: part of the lifeline where the element is actively running

- combined fragments**: for complex interactions

- state invariants**: for conditioning the interactions

Sequence diagrams: messages

Messages can be of different type:

synchronous: where the sender wait an answer →

asynchronous: such as signals →

reply messages: for answers ← - - - - -

lost messages: ———●

found messages: ●————→

create (arrows arrive to the head) and destroy (a cross at the end of the lifeline) messages

Note

The sequencing of messages is only between sending/receiving messages and on each lifeline, not between them.

Combined fragments

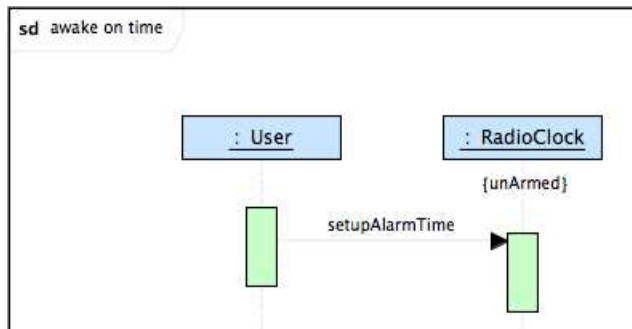
It is a construct used to model complex pattern of interactions. The main ones are:

- par: for parallel behavior
- alt: for alternatives
- opt: for optional parts
- loop: well, for loops!
- ref: for reuse

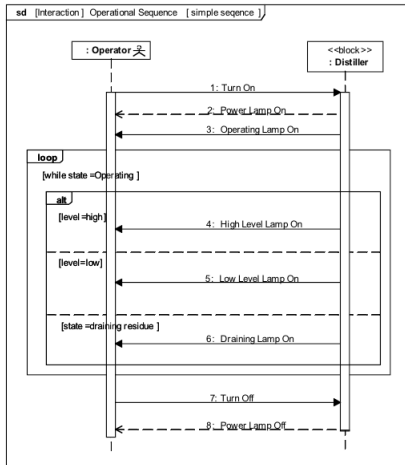
Note

It is also possible to represent timing constraint (see [5] for more details).

Example 1: RadioClock





Example 2: Distiller



State Machines

To model the behavior of a block that reacts differently to the same event according to its current state, then the State Machines are usefull.

The main concepts of this diagram are:

- states
 - initial (pseudo)state ●
 - regular state 
 - final state 
 - each state can have a entry/exit/do activity
- transitions (with triggers, gard, effect)
- regions
- pseudostates junction ● and choice ◇

Transitions

A transition may include one ore more **triggers**, a **guard** and an **effect**.

- triggers:**
- signal events: name of the signal or operation
 - time events: using *after* or *at*
 - change events: using *when*
 - call events: similar to signal but use of () to represent method calls
- guard:** logical expression (must be exclusive)
- effect:** behavior (activity) executed during the transition

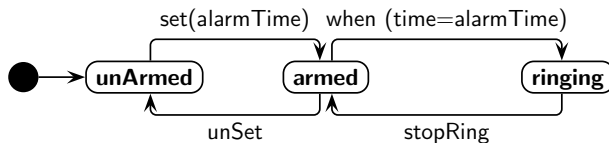
State Hierarchies

When a state has itself a behavior, it is called **composite** (or hierarchical) state.

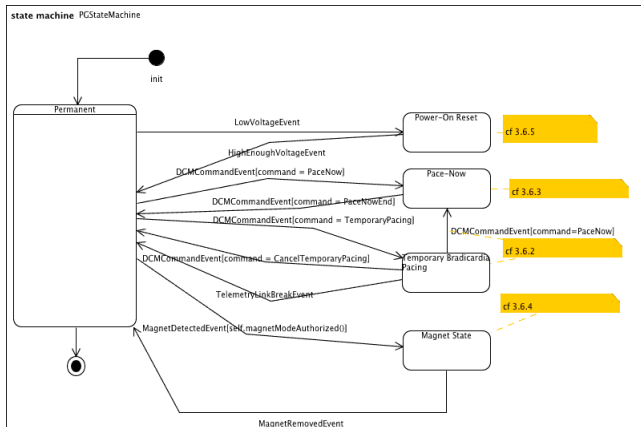
Composite states can have:

- single region: introduction of the **history** pseudostate
- multiple (orthogonal) region: introduction of the **fork/join** pseudostates

Example 1: RadioClock



Example 2: PaceMaker



Activity Diagrams: basic concepts

- Activities
- Actions
- Object Flows
- Control Flows

Activities

An **activity** is represented by one Activity Diagram and is the description of the organisation of a set of actions (mainly transforming data, controlling events, etc.).

Each activity has **parameters** to link the inside with the outside. Each Activity Diagram represents one Activity.

Activities can also be represented in a Block Definition Diagram in order to show the relations between them.

Note

- An activity can itself be an action of another activity using a **call behavior action**.
- A parameter in activities is different from a parameter in an internal block diagram.

Actions

An **action** is the atomic element of an activity. It processes **tokens** placed on **pins**. A pin acts as a token buffer.

The actions are determined by a set of precise rules:

- The action's owning activity is executing
- The number of available tokens at each required input pin is sufficient
- A token is available on each action's control flow

Then the action executes:

- Tokens are placed on each of the action's output pins
- The action terminates only if the required number of tokens are reached or if its owning activity terminates.

Object Flows

Object flows are used to route tokens between actions.

They can be routed by using several mechanisms:

- fork node:** one input flow and several output flows (replication)

- join node:** one output flow and several input flows (synchronisation)

- decision node:** one input and several output (routing according some properties)

- merge node:** one output and several input (no synchronisation required and all tokens routed)

Control Flows

Control flows are used to route tokens between actions.

Buffers and data stores

Buffers and **Data stores** are special kind of object node (like pins and parameters).

Activity diagrams

The basic modeling process for an activity diagram is:

- 1 Define input and output parameters of the activity
- 2 Define the set of actions
- 3 Place pins on those actions
- 4 Connect them with object flows
- 5 Organize activity with control flows

Cross-cutting

Methods considerations

Example 1: PaceMaker

Here are the steps taken in the PaceMaker case study [2]:

1 System Specification

- Context: Context bdd and Lifecycle sm
- General needs: uc and nominal scenarios (sd)
- Requirements model : req and traceability

2 System Design

- Functional models: ac
- Domain-Specific Data: bdd
- Logical Architecture: bdd, ibd and sm
- Physical Architecture: bdd

3 Traceability and Allocations

- link between tech. needs and use cases

4 Testing Models

-

Case studies

HSUV example

About this case study

- Classical SysML example [5]

Radio Clock

About this case study

- Classical radio clock example [6]
- use of the Rhapsody tool
- contact: Philippe Leblanc (philippe.leblanc@fr.ibm.com), from IBM for the tool and Pascal Roques from PRFC for the case study itself (taken from his book [6]).

Requirements

- Being able to be awake at a certain time

PaceMaker

About this case study

- The main case study fully treated in [2]
- use of the TOPCASED tool
- contact: Agusti Canals, from C-S for the tool and Loïc Féjoz from PRFC for the case study itself.

Definitions

Glossary

Glossaire I

I have grouped in this section the terms and acronyms used in this course. The one that are already defined in the text are simply listed and reference the definition in the text itself.

Here are the sources I have used:

- [Glossaire du Software Engineering Institute](#)
- [IEEE Computer Dictionary Online](#)
- [WIKIPEDIA – The Free Encyclopedia](#)
- [A glossary of terms with multilingual translations](#)

Glossaire II

CRUD

Create, Read, Update, and Delete, main action on database data.

DRY

Don't Repeat Yourself, common rules in engineering (e.g., **RoR***).

IPT

Integrated Product Team classical team in system developments.

OMG

Object Management Group The group leading the development of SysML, UML, etc.

Glossaire III

PDF

Portable Document Format, printing standard.

RoR

Ruby on Rails, famous framework.

Additional materials

I have grouped together in this appendix all the generally boring part of the course (the one that often come first in traditional courses!). I have also included (hopefully) usefull materials for exam preparation (FAQ,

quizz, ...).

History

History I

From UML to SysML

From UML to SysML I

For those who already know UML, here are some **advice**s (only) to quickly jump into SysML:

- Forget about class and objects
- Think like an engineer
- Focuss on the requirements
- Play with SysML tools
- Stay tuned (SysML forums, groups and lists)

Challenges and open questions around SysML

What more than SysML?

In this section we are going to illustrate the ongoing questions around SysML:

- Modeling and Analysis of Real-Time and Embedded Systems (MARTE)
- Architecture Analysis and Design Language (AADL)
- Wide range of levels of abstraction in SysML
- Traceability
- Methodological concerns

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

- A UML profile
- Real-Time Oriented
- Supported by the **OMG***
- <http://www.aadl.info/>

Architecture Analysis and Design Language (AADL)

- A architecture description language
- Verification and Validation using tools
- Extension mechanisms (parser)
- <http://www.aadl.info/>

What granularity level using SysML?

It is hard in SysML to address the question of the level of abstraction that should be considered for a particular design or analysis decision. May be some **heuristics** would help.

How do you keep links between requirements and corresponding model elements?

Which existing method could be adapted for SysML?

We have considered partly this question in section 102.

Frequently Asked Questions

About this FAQ

This FAQ has been constructed by experience, using questions of the students during my courses. I have also added questions often found in discussions and forum. It can be a good starting point for preparing an exam.

You can also check an existing one:

<http://www.sysmlforum.com/FAQ.htm>

What is the current version of SysML and how can I obtain it?

Version 1.2 and here is the specification link:

<http://www.omg.org/cgi-bin/doc?formal/10-06-02>.

What changes were made during the last revision?

Notable changes in Version 1.2 of SysML include:

- Synchronization with changes in UML 2.3
- Conjugate ports metamodel and notation
- Naming of interruptible activity regions
- Inclusion of UML instance specifications
- Inclusion of UML structured activity nodes
- Inclusion of UML multiple item flow notation
- Improvements to Unit and QuantityKind support for value types, and a non-normative model to define systems of units and quantities

The SysML v1.3 Revision Task Force led by Roger Burkhart and Rick Steiner is continuing to work on proposed improvements to SysML based on feedback from the systems modeling community.



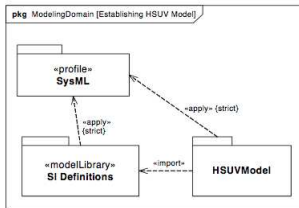
How do we setup ranges (limits) of (input) variables (in a par for example)?

In the details or the definition of the Flow Item.

What does the {strict} keyword means for profile application?

No other meta-elements than the one in the applied profile were used (e.g., you can use a tool supporting the profile with confidence).

“The semantics of UML profiles ensure that when a user model “strictly” applies the SysML profile, only the UML metaclasses referenced by SysML are available to the user of that model. If the profile is not “strictly” applied, then additional UML metaclasses that were not explicitly referenced may also be available.” ([5, p. 8])



What does the visibility (private, public, ...) means for a block?

As any model elements, the **visibility** of a block describes how it can be imported outside its namespace.

Note

It depends on the tool support for visibility controls to use this feature.

What is the difference between an internal and a self transition?

In a self transition the exit and the the entry events are triggered.

Can I attach a History pseudostate to a particular (non composite) state?

No. The History pseudostate (H) is indicated inside a composite state and means that when back in this superstate, the machine goes back to its last active state.

Potential questions for exams

Here is a list of unanswered questions that you should work on:

- Why do systems engineer need yet-another-modeling-language?
- What is the relationship between “open source SysML” and “OMG SysML”?
- What is the roadmap for OMG SysML 2.0?
- Who are the SysML Partners?
- What is the relationship between UML and SysML?
- Can SysML and UML be used together?
- Can SysML be customized?
- Which language is easier to learn, SysML or UML?

Support and references








Support and references (tools)


- [TOPCASED](#)
- [Papyrus](#)
- [Rhapsody](#)
- Enterprise Architect
- Artisan
- Magic Draw
- Visio SysML Template



Bibliographie

-  Martin Baudouin.
Construction du modèle sysml de la balance halo de chez terrailon.
Technical report, 2012.
-  Sanford Friedenthal, Alan Moore, and Rick Steiner.
Modélisation et analyse de systèmes embarqués.
Hermès, To be published in 2012.
-  John Holt and Simon Perry.
SysML for Systems Engineering.
Professional Applications of Computing Series 7, 2008.
-  Fabrice KORDON, Jérôme HUGUES, Agusti CANALS, and Alain DOHET.
A Practical Guide to SysML.
The MK/OMG Press, 2008.

 **OMG.**
Systems modeling language version 1.2.
Technical report, 2010.

 **Pascal Roques.**
SysML par l'exemple.
Eyrolles, 2009.

