

# SysML: Systems Modeling Language

Jean-Michel Bruel  
[bruel@iut-blagnac.fr](mailto:bruel@iut-blagnac.fr)



February 16, 2012

# Contents

<b>1</b>	<b>Foreword</b>	<b>4</b>
1.1	Who is the expected audience? . . . . .	4
1.2	History . . . . .	4
1.3	How to read this document? . . . . .	4
1.4	Usage and other legal considerations . . . . .	5
1.5	Organization . . . . .	5
<b>2</b>	<b>Introduction</b>	<b>7</b>
2.1	Overview . . . . .	7
<b>3</b>	<b>System Engineering</b>	<b>9</b>
3.1	Modeling systems . . . . .	9
<b>4</b>	<b>Introduction to SysML</b>	<b>10</b>
4.1	Overview . . . . .	10
4.2	Basic Principles . . . . .	10
4.3	Organisation . . . . .	11
4.4	Requirements . . . . .	14
4.5	Structure . . . . .	18
4.6	Dynamic aspects . . . . .	27
4.7	Cross-cutting . . . . .	35
4.8	Methods considerations . . . . .	35
<b>5</b>	<b>Case studies</b>	<b>36</b>
5.1	HSUV example . . . . .	36
5.2	Radio Clock . . . . .	36
5.3	PaceMaker . . . . .	36
<b>A</b>	<b>Definitions</b>	<b>37</b>
A.1	Glossary . . . . .	37
<b>B</b>	<b>Additional materials</b>	<b>39</b>
B.1	History . . . . .	39
B.2	From UML to SysML . . . . .	39
B.3	Challenges and open questions around SysML . . . . .	39
B.4	Frequently Asked Questions . . . . .	40
B.5	Quizz (ref. 2012-02-06) . . . . .	43
B.6	Exercices . . . . .	47

<i>CONTENTS</i>	2
<b>C Indexes</b>	<b>48</b>
Index des personnalités . . . . .	49
Index des entreprises, sociétés ou groupes . . . . .	50
Index des langages, méthodes et outils . . . . .	51
Index général . . . . .	52
<b>References</b>	<b>54</b>

# List of Figures

4.1	Packages organisation . . . . .	11
4.2	Another Packages organisation . . . . .	12
4.3	Example of Delegation . . . . .	13
4.4	Packages Dependencies . . . . .	14
4.5	Pacemaker Packages organisation . . . . .	15
4.6	Radioclock Packages organisation . . . . .	15
4.7	HSUV Requirements Diagram . . . . .	17
4.8	Detailed HSUV Requirements Package . . . . .	17
4.9	Value Types . . . . .	20
4.10	Pacemaker Bock Definition Diagram . . . . .	21
4.11	Radioclock Block Definition Diagram . . . . .	22
4.12	Using standard types (Papyrus) . . . . .	23
4.13	Using standard types (HSUV) . . . . .	23
4.14	Block Definition Diagram (Distiller v1) . . . . .	24
4.15	Block Definition Diagram (Distiller v2) . . . . .	24
4.16	Internal Block Definition Diagram (Distiller v1) . . . . .	25
4.17	Internal Block Definition Diagram (Distiller v2) . . . . .	25
4.18	Constraints . . . . .	26
4.19	Parametric Diagrams . . . . .	27
4.20	A Pacemaker Use Case Diagram . . . . .	29
4.21	A Pacemaker Sequence Diagram . . . . .	29
4.22	A RadioClock Sequence Diagram . . . . .	30
4.23	A Distiller Sequence Diagram . . . . .	31
4.24	A Pacemaker State Machine Diagram . . . . .	33
B.1	Example of the usage of {strict} [5] . . . . .	41

# Chapter 1

## Foreword

### Contents

1.1	Who is the expected audience? . . . . .	4
1.2	History . . . . .	4
1.3	How to read this document? . . . . .	4
1.4	Usage and other legal considerations . . . . .	5
1.5	Organization . . . . .	5

### 1.1 Who is the expected audience?

My students, my colleagues, and myself!

### 1.2 History

This document is a compilation of notes and discussions. The reference section (see p. 54 where you will find the main books I have used). I would like to thank some of my friends who have helped me in this compilation: my colleagues who also teach, Nicolas Belloir (UPPA), Laurent Nonne (IUT DE BLAGNAC); my colleagues from [SysML-France](#), Pascal Roques from PRFC, Agusti Canals from C-S, Loïc Féjoz from RTAW, the people from UNIVERSIDAD AUTÓNOMA DE GUADALAJARA, and more especially Karina Aguilar for inviting me to teach [SysML](#) there.

### 1.3 How to read this document?

#### Electronic version and printed version

This document is intended to be read in its electronic version (PDF\* format). It allows navigation, references, click on URLs, etc. For example, acronyms such as [DRY\\*](#) appear with a star (\*) that allows a direct link to its definition in the PDF\* document (section A.1, p. 37).

If you are reading a printed version of it, links are useless and that is your punishment for killing trees!

### Typo Conventions

Thanks to the L<sup>A</sup>T<sub>E</sub>X editing system I have used several personal typographic conventions:

- Particular focus on people names (e.g., Jean-Michel Bruel), enterprises or projects (e.g., [OBJECTEERING](#)), etc.

- Definitions can be numbered to be referenced in the text and are grouped in some part of the document (cf. section [A.1](#), p. 37).
- References are used (cf. p. 54).
- All the *flottings* (figures, définitions, etc.) are listed in the table of content (cf. 1, p. 2).

## 1.4 Usage and other legal considerations

This document has been realised with [L<sup>A</sup>T<sub>E</sub>X](#), using the [Texmaker](#) editor, on a [Mac OS X 10.5.3](#) environment. I have used numerous packages and I thank their authors.

This document can be freely used as long as you keep mention of my name and of [SysML-France](#), and as long as you send me feedbacks and comments.

## 1.5 Organization

### 1.5.1 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!)

### 1.5.2 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)

### 1.5.3 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, [L<sup>A</sup>T<sub>E</sub>X](#), ... to be discussed)

### 1.5.4 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 ;-)

### 1.5.5 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

# Chapter 2

## Introduction

### Contents

---

<b>2.1 Overview</b> . . . . .	<b>7</b>
-------------------------------	----------

---

## 2.1 Overview

### 2.1.1 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

### 2.1.2 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/developer/>

# Chapter 3

## System Engineering

### Contents

---

<b>3.1 Modeling systems</b> . . . . .	<b>9</b>
---------------------------------------	----------

---

### 3.1 Modeling systems

#### 3.1.1 Difference with SE

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

#### 3.1.2 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

#### 3.1.3 Systems of Systems

A system:

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

# Chapter 4

## Introduction to SysML

### Contents

---

4.1	Overview . . . . .	10
4.2	Basic Principles . . . . .	10
4.3	Organisation . . . . .	11
4.4	Requirements . . . . .	14
4.5	Structure . . . . .	18
4.6	Dynamic aspects . . . . .	27
4.7	Cross-cutting . . . . .	35
4.8	Methods considerations . . . . .	35

---

### 4.1 Overview

#### 4.1.1 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](http://www.sysml-france.org) association: <http://www.sysml-france.org>

### 4.2 Basic Principles

#### 4.2.1 Global approach

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

	Requirements	Structure	Behavior	Crosscutting
Organisation				
Analysis				
Design				
Implementation				

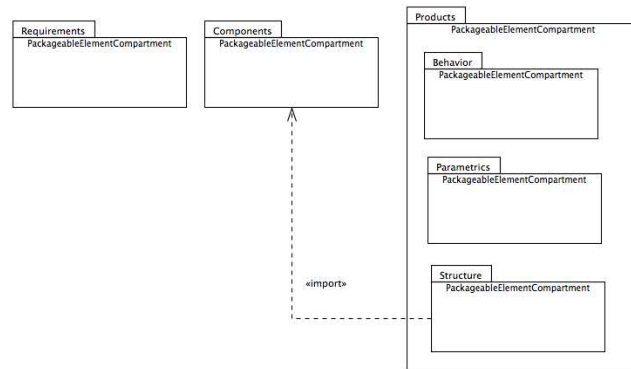


Figure 4.1: Packages organisation

## 4.3 Organisation

### 4.3.1 About this section

	Requirements	Structure	Behavior	Crosscutting
<b>Organisation</b>				
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

### 4.3.2 The Package diagram

(cf. figure 4.1 and figure 4.2)

### 4.3.3 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

### 4.3.4 Possible organisation

- By system hierarchy (system, subsystems, components, ...)
- By process life cycle (reqs, analysis, ...)

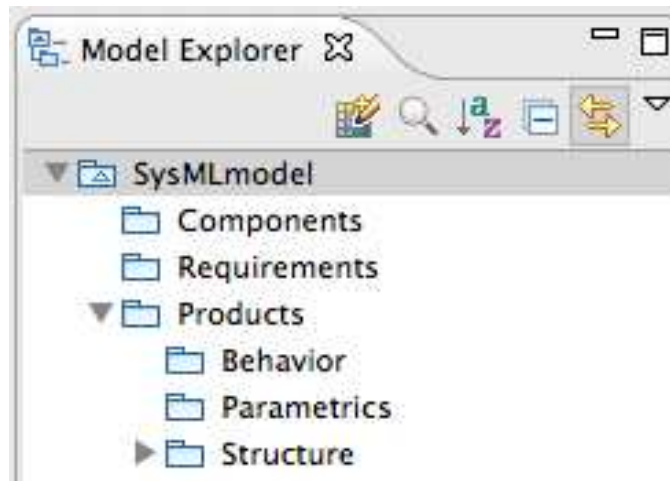


Figure 4.2: Another Packages organisation

- By teams (architects, IPT\*, ...)
- By type of models (Req, behavior, ...)
- By combination of the preceeding

#### 4.3.5 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)

#### 4.3.6 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)

- ----->

**Refine:** • the client is a refinement (more details) of the supplier

- ----->

#### 4.3.7 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)

- ----->

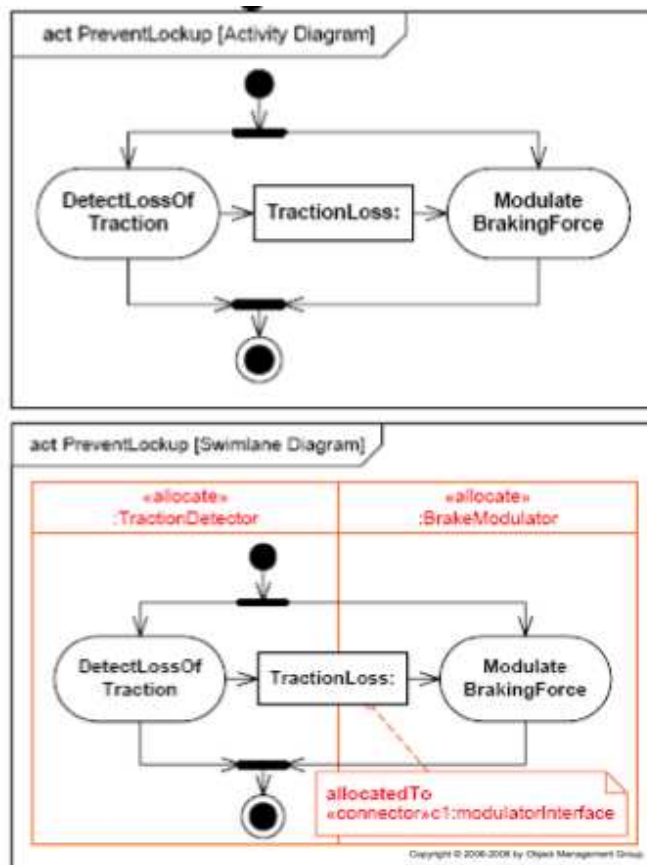


Figure 4.3: Example of Delegation

### 4.3.8 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

### 4.3.9 Delegation: example with swimlanes

(cf. figure 4.3)

### 4.3.10 Dependencies: full example

Here is a full example (taken from [4, p. 90], (cf. figure 4.4))

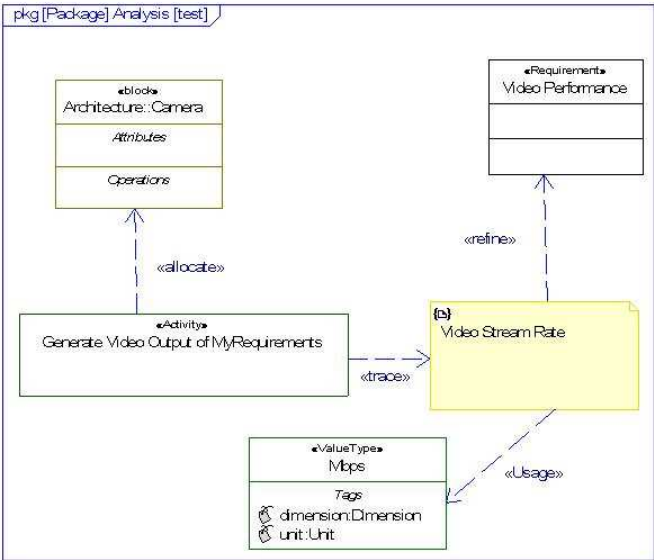


Figure 4.4: Packages Dependencies

4.3.11 Example 1: PaceMaker

Here is a an example of organization [2] (cf. figure 4.5):

4.3.12 Example 2: RadioClock

Here is a an example of organization from [6] (cf. figure 4.6):

4.4 Requirements

4.4.1 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

4.4.2 Requirements organization

Here are some examples:

- taken from [2] (see also section 4.3):
  - General needs (links with use cases)
  - Technical needs (links with design elements)

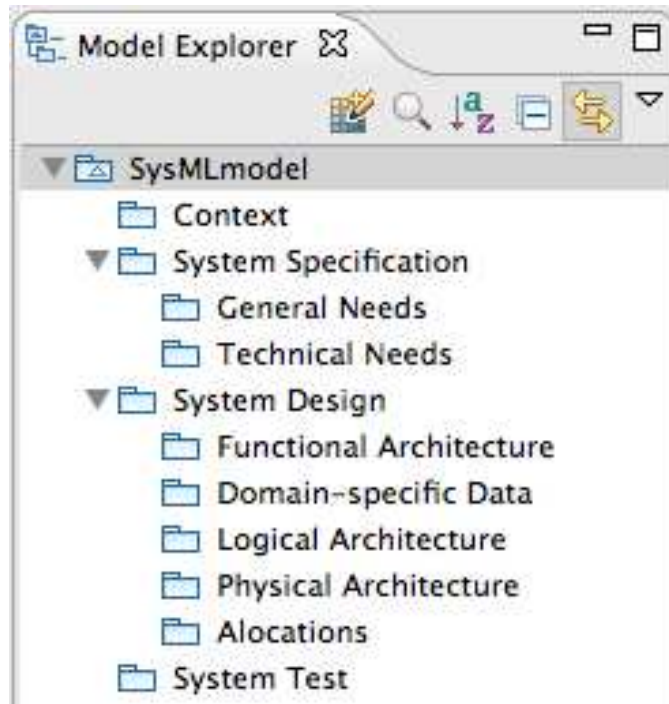


Figure 4.5: Pacemaker Packages organisation

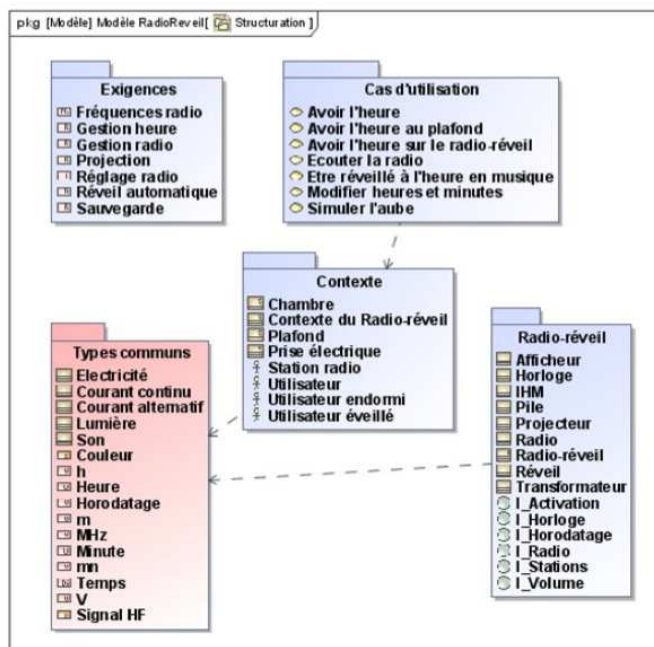


Figure 4.6: Radioclock Packages organisation



- taken from [1]:
  - Main requirements (linked with use cases)
  - Typical groups of requirements:
    - \* Marketing reqs
    - \* Functional reqs
    - \* Environmental reqs
    - \* Business reqs
    - \* ...

### 4.4.3 Requirements properties

*Requirements* can have several properties [6]:

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

### 4.4.4 Requirements links

**Containment:** for decomposition ( $\oplus$ –)

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

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

### 4.4.5 Traceability considerations

Once requirements have been organized they need to be linked to at least use cases (using  $\ll\text{refine}\gg$  for example) and to structural elements (using  $\ll\text{satisfy}\gg$  for example).

**Note**

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

### 4.4.6 Example 1: HSUV

Here is an example of requirement diagram taken from <http://www.uml-sysml.org/sysml> (cf. figure 4.7):

### 4.4.7 Example 2: HSUV

Here is another example of requirement diagram taken from <http://www.uml-sysml.org/sysml> (cf. figure 4.8):

### 4.4.8 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	$\ll\text{satisfy}\gg$ $\ll\text{verify}\gg$			

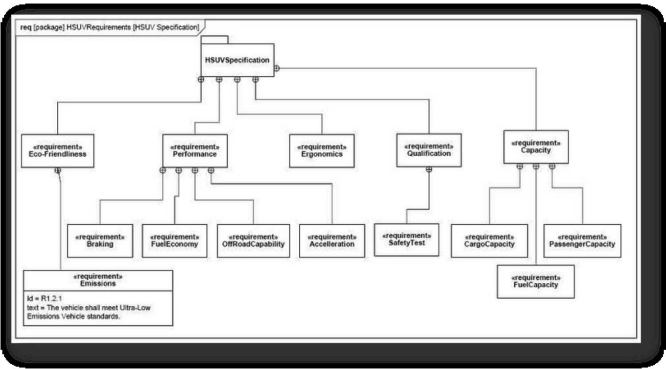


Figure 4.7: HSUV Requirements Diagram

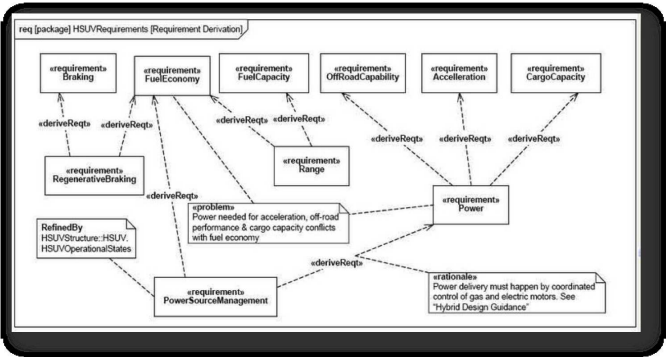


Figure 4.8: Detailed HSUV Requirements Package

## 4.5 Structure

### 4.5.1 About this section

	Requirements	<b>Structure</b>	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

### 4.5.2 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.

### 4.5.3 Block Definition Diagrams

A *bdd* can represent:

- a package
- a block
- a constraint block

### 4.5.4 Blocks Properties

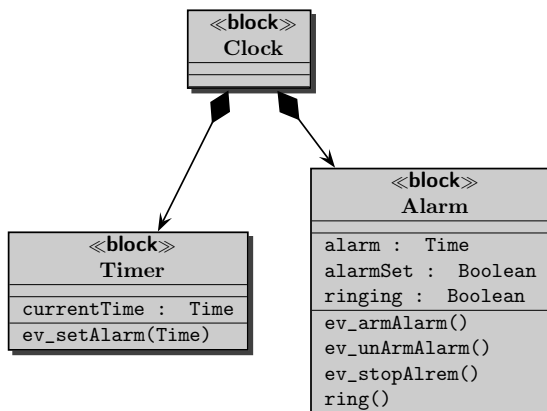
The main properties of a block are:

**parts:** the elements that compose the block

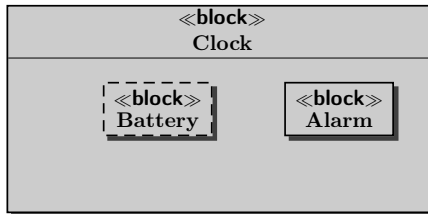
**values:** its (quantifiable) characteristics

**references:** the elements the block has access to

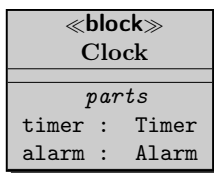
### 4.5.5 Parts: composition view



### 4.5.6 Parts: internal view



### 4.5.7 Parts: bloc details view



### 4.5.8 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

### 4.5.9 Values and Value Types

Quantitative *values* are so commonly used that there is a special kind of block to type them: *Value Types* (cf. figure 4.9).

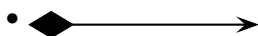
### 4.5.10 References

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

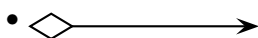
### 4.5.11 Blocks Associations

The main relationship between blocks are:

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



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



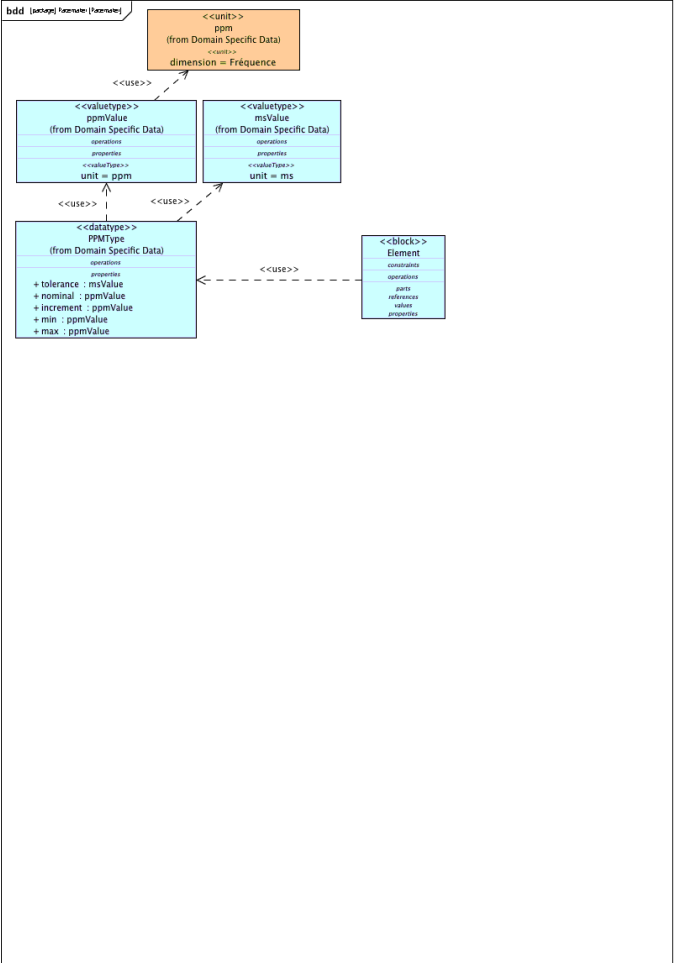


Figure 4.9: Value Types

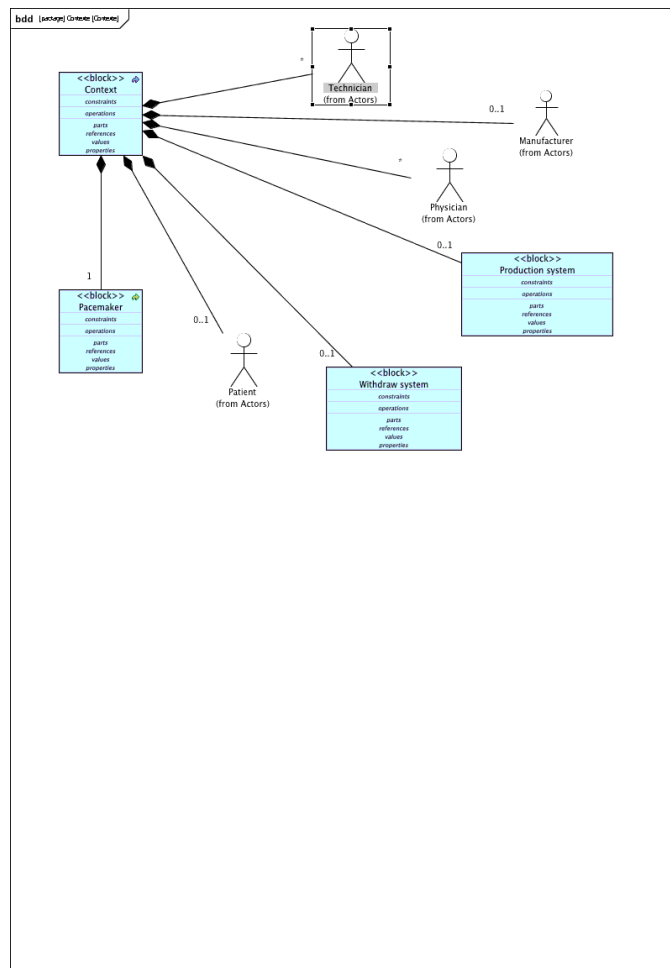


Figure 4.10: Pacemaker Block Definition Diagram

**association:** just link between blocks

**generalisation:** • the client is a special kind of provider



#### 4.5.12 Example 1: PaceMaker

Here is an example of context bdd [2] (cf. figure 4.10):

#### 4.5.13 Example 2: RadioClock

Here is an example of block with parts, references and value types (cf. figure 4.11):

#### 4.5.14 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)

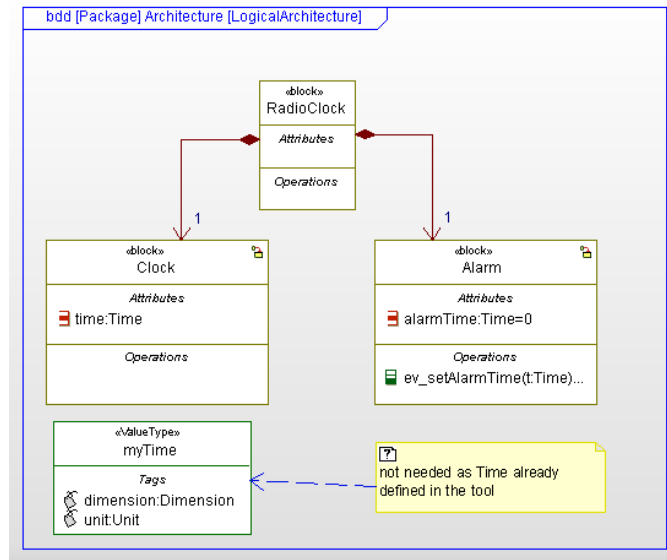


Figure 4.11: Radioclock Block Definition Diagram

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

#### Note

A good modeling tool provides International System of Units definitions.

#### 4.5.15 Example 1: SI in Papyrus

Here is an example of use of SI values (cf. figure 4.12 and figure 4.13):

#### 4.5.16 Example 2: HSUV

#### 4.5.17 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

#### 4.5.18 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).

#### 4.5.19 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)

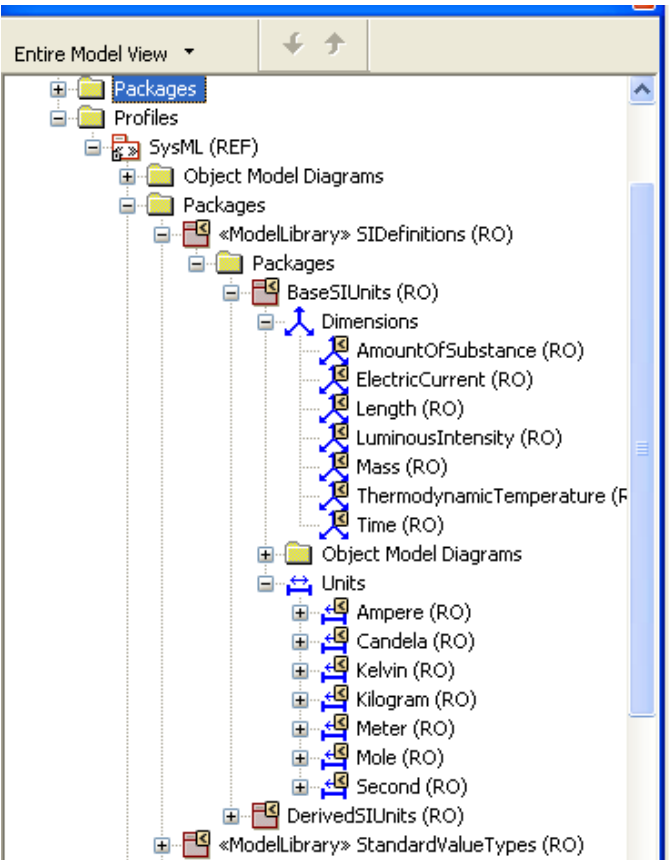


Figure 4.12: Using standard types (Papyrus)



Figure 4.13: Using standard types (HSUV)



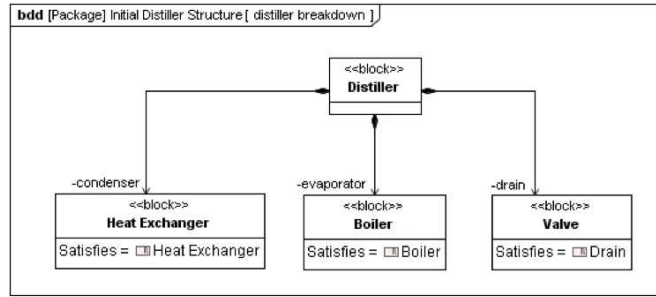


Figure 4.14: Block Definition Diagram (Distiller v1)

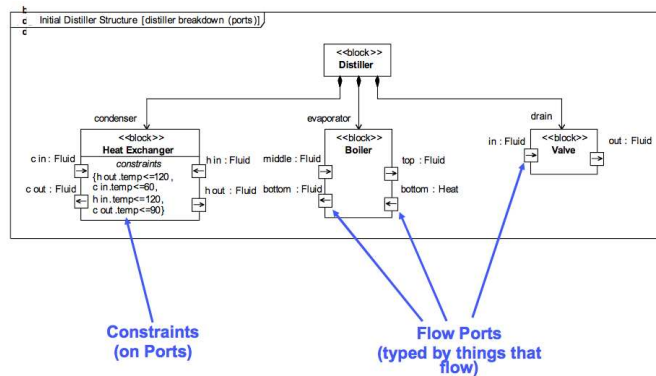



Figure 4.15: Block Definition Diagram (Distiller v2)

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

#### 4.5.20 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.

#### 4.5.21 Example 1: Distiller (bdd v1)

(cf. figure 4.14)

#### 4.5.22 Example 1: Distiller (bdd v2)

(cf. figure 4.15)

#### 4.5.23 Example 1: Distiller (ibd v1)

(cf. figure 4.16)

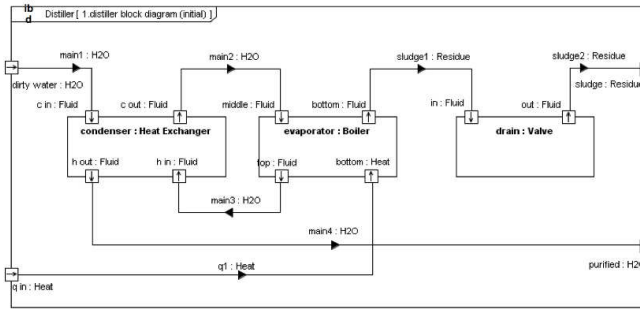


Figure 4.16: Internal Block Definition Diagram (Distiller v1)

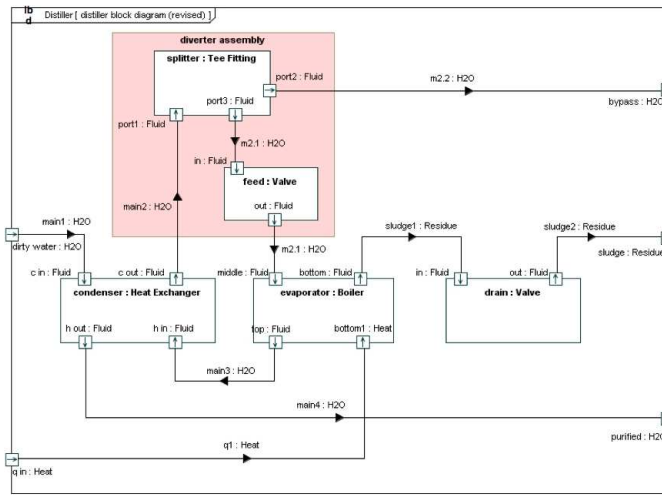


Figure 4.17: Internal Block Definition Diagram (Distiller v2)

#### 4.5.24 Example 1: Distiller (ibd v2)

(cf. figure 4.17)

#### 4.5.25 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

#### 4.5.26 Constraints

A constraint is a special block with:

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

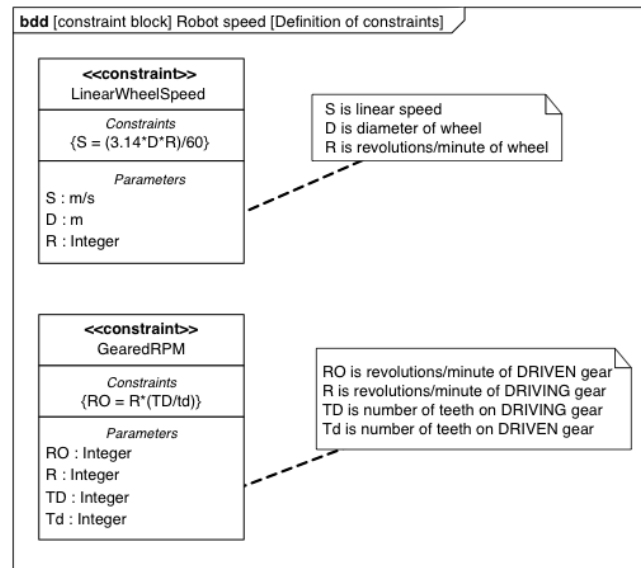
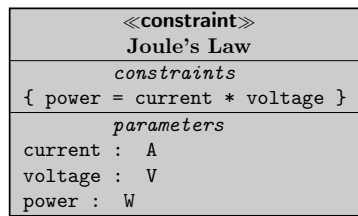


Figure 4.18: Constraints



### 4.5.27 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.

### 4.5.28 Value binding

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

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

### 4.5.29 Example: Constraints

Here is an example taken from [3] (cf. figure 4.18):

### 4.5.30 Example: Parametrics

Also from [3] (cf. figure 4.19):

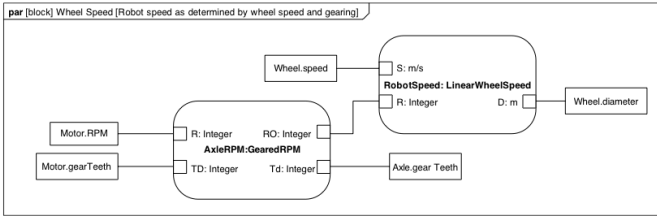


Figure 4.19: Parametric Diagrams

4.5.31 To summarize

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

4.6 Dynamic aspects

4.6.1 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:


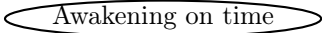
- Use Case Diagrams
- Sequence Diagrams
- State Machines
- Activity Diagrams

4.6.2 Use Case Diagrams

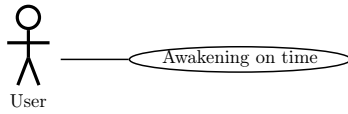
Use cases represent the way a system is used. We can have Use Case diagrams at different level of abstraction:

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

4.6.3 Use Case Diagrams: basic concepts

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

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

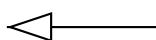


#### 4.6.4 Relation between Use Cases

A use case is fully described by some properties:

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

#### 4.6.5 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

#### 4.6.6 Example 1: PaceMaker uc

Here is a PaceMaker example taken from [2] (cf. figure 4.20):

#### 4.6.7 Example 2: PaceMaker seq

Here is the corresponding sequence diagram (nominal scenario) [2] (cf. figure 4.21):

#### 4.6.8 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

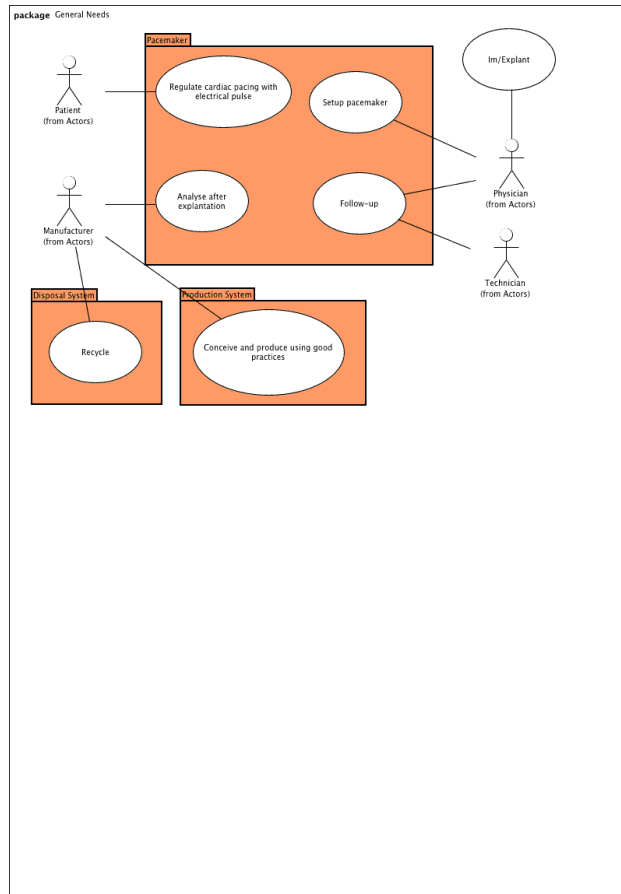


Figure 4.20: A Pacemaker Use Case Diagram

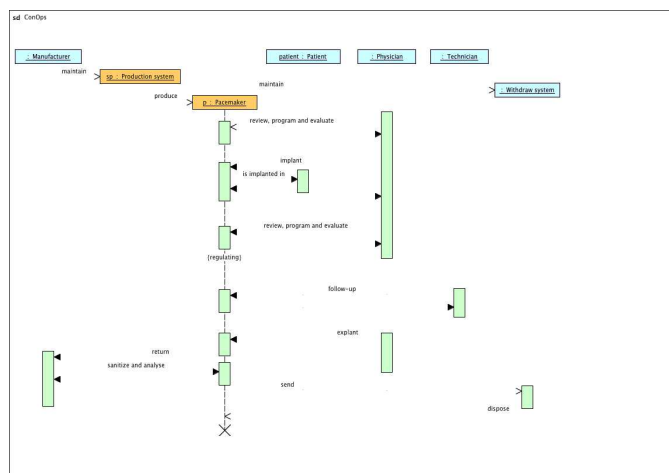


Figure 4.21: A Pacemaker Sequence Diagram

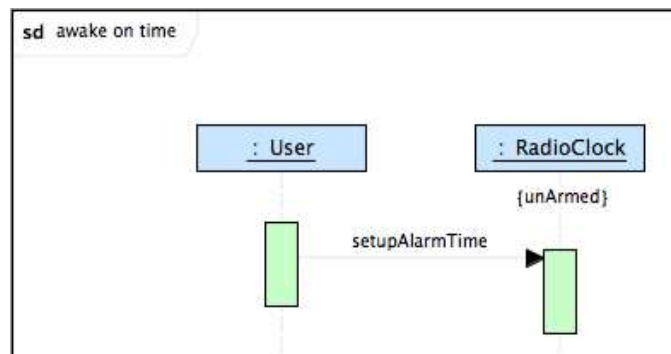


Figure 4.22: A RadioClock Sequence Diagram

#### 4.6.9 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.

#### 4.6.10 Combined fragments

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

- par: for parralel 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).

#### 4.6.11 Example 1: RadioClock

(cf. figure 4.22)

#### 4.6.12 Example 2: Distiller

(cf. figure 4.23)

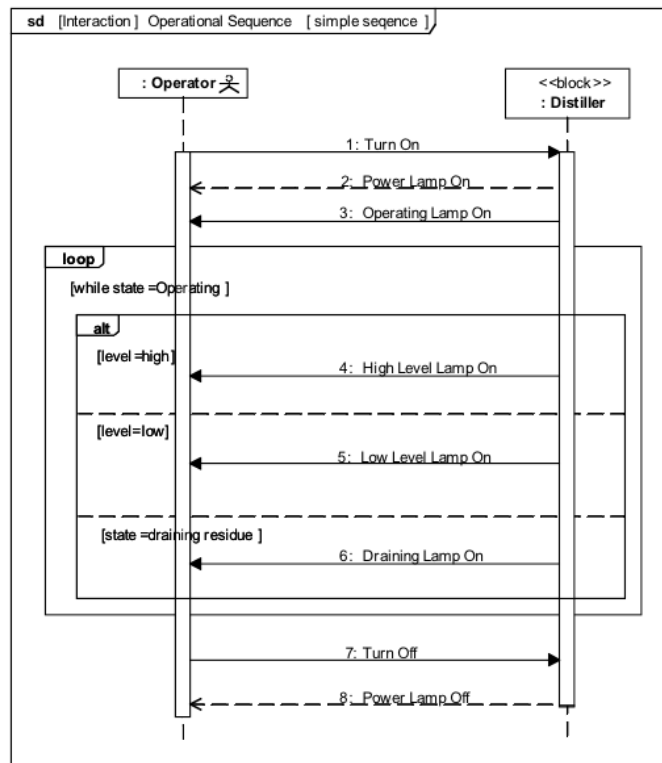


Figure 4.23: A Distiller Sequence Diagram

### 4.6.13 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 `unArmed`
  - final state ●
  - each state can have a entry/exit/do activity
- transitions (with triggers, gard, effect)
- regions
- pseudostates junction ● and choice ◇

### 4.6.14 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

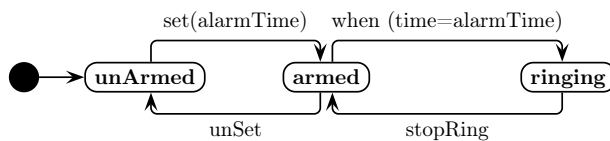
#### 4.6.15 State Hierarchies

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

#### 4.6.16 Example 1: RadioClock



#### 4.6.17 Example 2: PaceMaker

(cf. figure 4.24)

#### 4.6.18 Activity Diagrams: basic concepts

- Activities
- Actions
- Object Flows
- Control Flows

#### 4.6.19 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.



Figure 4.24: A Pacemaker State Machine Diagram

#### 4.6.20 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.

#### 4.6.21 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)

#### 4.6.22 Control Flows

*Control flows* are used to route tokens between actions.

#### 4.6.23 Buffers and data stores

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

#### 4.6.24 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

## 4.7 Cross-cutting

## 4.8 Methods considerations

### 4.8.1 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
  -

# Chapter 5

## Case studies

### Contents

<b>5.1</b>	<b>HSUV example . . . . .</b>	<b>36</b>
<b>5.2</b>	<b>Radio Clock . . . . .</b>	<b>36</b>
<b>5.3</b>	<b>PaceMaker . . . . .</b>	<b>36</b>

### 5.1 HSUV example

#### 5.1.1 About this case study

- Classical [SysML](#) example [5]

### 5.2 Radio Clock

#### 5.2.1 About this case study

- Classical radio clock example [6]
- use of the [Rhapsody](#) tool
- contact: Philippe Leblanc ([philippe.leblanc@fr.ibm.com](mailto: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]).

#### 5.2.2 Requirements

- Being able to be awake at a certain time

### 5.3 PaceMaker

#### 5.3.1 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.

# Appendix A

## Definitions

### A.1 Glossary

Glossaire

#### A.1.1 Sources and conventions

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](#)

#### A.1.2 List of terms

---

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.

---

---

PDF

---

*Portable Document Format*, printing standard.

---

---

RoR

---

*Ruby on Rails*, famous framework.

---

# Appendix B

## 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, ...).

### B.1 History

History

### B.2 From UML to SysML

From UML to SysML

For those who already know [UML](#), here are some *advices* (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)

### B.3 Challenges and open questions around SysML

#### B.3.1 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



### B.3.2 Modeling and Analysis of Real-Time and Embedded Systems (MARTE)

- A [UML](#) profile
- Real-Time Oriented
- Supported by the [OMG](#)\*
- <http://www.aadl.info/>

### B.3.3 Architecture Analysis and Design Language (AADL)

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

### B.3.4 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.

### B.3.5 How do you keep links between requirements and corresponding model elements?

### B.3.6 Which existing method could be adapted for SysML?

We have considered partly this question in section 4.8.

## B.4 Frequently Asked Questions

### B.4.1 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>

### B.4.2 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>.

### B.4.3 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

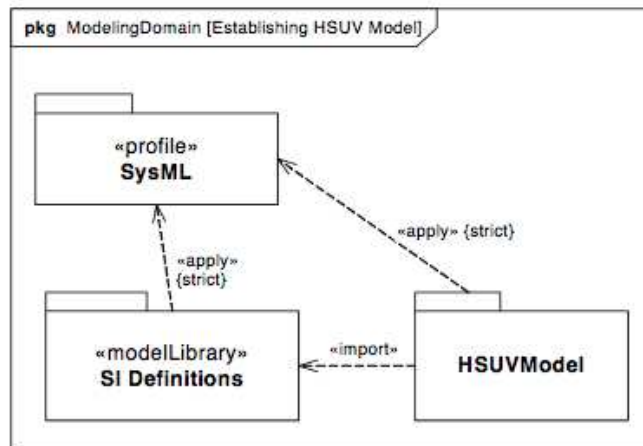


Figure B.1: Example of the usage of {strict} [5]

- 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.

#### B.4.4 How do we setup ranges (limits) of (input) variables (in a par for example)?

In the details or the definition of the Flow Item.

#### B.4.5 What does the {strict} keyword means for profile application?

No other meta-elements than the one in the applied profile (e.g., figure B.1) 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])

#### B.4.6 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.

#### B.4.7 What is the difference between an internal and a self transition?

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

### B.4.8 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.

### B.4.9 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?

## B.5 Quizz (ref. 2012-02-06)

An answer sheet is available for you on [B.5.1](#), the answer are in [B.5.2](#), p. 47. Mutiple answers possible.

**Question 1 : What's the meaning of the acronym SysML?**

---

- a. SYStems Modeling Language
- b. SYStematic Modeling Language
- c. SYstems of Systems Modeling Language

**Question 2 : When was SysML born?**

---

- a. 2001
- b. 2002
- c. 2003

**Question 3 : What is the current version of SysML?**

---

- a. 1.1
- b. 1.2
- c. 1.3

**Question 4 : What diagram allow for organisation of models?**

---

- a. Requirements Diagram
- b. Package Diagram
- c. Organization Definition Diagram

**Question 5 : SysML is supported by ...**

---

- a. IEEE
- b. OMG
- c. INCOSE

**Question 6 : How many diagrams is there in SysML?**

---

- a. 14
- b. 9
- c. 10

**Question 7 : SysML is ...**

---

- a. a specific language for complex systems
- b. strongly UML-Based
- c. just a UML profile

**Question 8 : SysML is ...**

---

- a. mainly focusing on analysis
- b. sufficient in itself
- c. a method

**Question 9 : A link from an external port of a block to an port of one of its part is called a ...**

---

- a. allocation
- b. delegation
- c. reference

**Question 10 : Namespaces are handled in SysML by ...**

---

- a. Delegation
- b. Packages
- c. Dependencies

**Question 11 :** The name of the model element prefixed by its packages (e.g., Structure::Products::Clock) is called a ...

---

- a. Full Name
- b. Qualified Name
- c. Qualified reference

**Question 12 :** Use and Refine are some kind of ...

---

- a. Allocations
- b. Dependencies
- c. Stereotypes

**Question 13 :** The white diamond indicates in SysML:

---

- a. aggregation
- b. composition
- c. generalisation

**Question 14 :** The two different kinds of Flow ports are:

---

- a. Atomic Flow Ports
- b. Broadcast Flow Ports
- c. Non-atomic Flow Ports

**Question 15 :** The Parametric diagram inherits from:

---

- a. The Internal Block Definition diagram
- b. The Block Definition diagram
- c. The Activity diagram

B.5.1 Quizz answer sheet

	a	b	c
Question n.1			
Question n.2			
Question n.3			
Question n.4			
Question n.5			
Question n.6			
Question n.7			
Question n.8			
Question n.9			
Question n.10			
Question n.11			
Question n.12			
Question n.13			
Question n.14			
Question n.15			

### B.5.2 Quizz answers

	a	b	c
Question n.1	X		
Question n.2	X		
Question n.3		X	
Question n.4		X	
Question n.5		X	X
Question n.6		X	
Question n.7	X	X	
Question n.8	X		
Question n.9		X	
Question n.10		X	
Question n.11			X
Question n.12		X	X
Question n.13	X		
Question n.14	X		X
Question n.15	X		

## B.6 Exercices

### B.6.1 Exercice: Crosswords

Here are some puzzle for your training:

- [Introduction](#) (6 words)
- [Structure and dynamics](#) (14 words)



# Appendix C

## Indexes

### Contents

<b>Index des personnalités . . . . .</b>	<b>49</b>
<b>Index des entreprises, sociétés ou groupes . . . . .</b>	<b>50</b>
<b>Index des langages, méthodes et outils . . . . .</b>	<b>51</b>
<b>Index général . . . . .</b>	<b>52</b>

## Index des personnalités

Aguilar Karina, 4

Belloir Nicolas, 4, 5

Bruel Jean-Michel, 4

Burkhart Roger, 41

Canals Agusti, 4, 36

de Saqui Sannes Pierre, 5

Féjoz Loïc, 4, 36

Leblanc Philippe, 36

Nonne Laurent, 4

Roques Pascal, 4, 36

Steiner Rick, 41

## **Index des entreprises, sociétés ou groupes**

C-S, 4, 36

Continental, 5

IBM, 36

IRIT, 5

IUT de Blagnac, 4

Objecteering, 4

PRFC, 4, 36

RTaW, 4

SysML-France, 4, 5

Universidad Autónoma de Guadalajara, 4

UPPA, 4

## **Index des langages, méthodes et outils**

Eclipse, 6

Papyrus, 6

Rhapsody, 36

SysML, 4, 6, 10, 11, 36, 39, 40

Texmaker, 5

TopCased, 36

UML, 39, 40

## Index général

- AADL, 39
- action, 34
- actions, 32
- activity, 32
- aggregation, 19
- Allocation, 12
- alt, 30
- association, 21
- asynchronous, 30
- bdd, 18
- behavior, 27
- behavior port, 24
- block, 21
- Block Configurations, 26
- Buffers, 34
- call behavior action, 32
- combined fragments, 28
- complex system, 9
- composite, 32
- composition, 19
- Connectors, 22
- connectors, 24
- constraints, 25, 26
- Containment, 16
- context, 18
- Control flows, 34
- CRUD, 37
- Data stores, 34
- dependencies, 12
- Dependency, 12
- Derivation, 16
- dimension, 21
- DRY, 37
- effect, 31, 32
- execution, 28
- Flow port, 22
- Flow Ports, 22
- fork, 32
- generalisation, 21
- guard, 31, 32
- heuristics, 40
- history, 32
- ibd, 22
- interaction, 28
- IPT, 37
- Items, 22
- join, 32
- lifeline, 28
- loop, 30
- MARTE, 39
- messages, 28, 30
- model librairies, 11
- models, 11
- multiplicity, 19
- namespace, 12
- Object flows, 34
- OMG, 37
- opt, 30
- organisation, 11
- package, 18
- Packages, 11
- packages, 11
- par, 30
- parameter, 32
- parameters, 26, 32
- parts, 18, 21, 22
- PDF, 37
- pins, 34
- probability distribution, 22
- pseudostates, 31
- Qualified names, 12
- Realization, 12
- ref, 30
- references, 18, 21
- Refine, 12
- Refinement, 16
- regions, 31
- requirement, 16
- Requirements, 16
- roles, 27
- RoR, 38
- sequence diagram, 28
- state invariants, 28
- states, 31
- synchronous, 30
- tokens, 34

Traceability, 39  
transitions, 31  
triggers, 31

unit, 21  
Use, 12  
use case, 16  
Use cases, 27

value biding, 26  
value type, 21  
Value Types, 19  
value types, 21  
values, 18, 19  
viewpoints, 11  
views, 11

# References

- [1] Martin Baudouin. Construction du modèle sysml de la balance halo de chez terraillon. Technical report, 2012. [16](#)
- [2] Sanford Friedenthal, Alan Moore, and Rick Steiner. *Modélisation et analyse de systèmes embarqués*. Hermès, To be published in 2012. [14](#), [21](#), [28](#), [35](#), [36](#)
- [3] John Holt and Simon Perry. *SysML for Systems Engineering*. Professional Applications of Computing Series 7, 2008. [26](#)
- [4] Fabrice KORDON, Jérôme HUGUES, Agusti CANALS, and Alain DOHET. *A Practical Guide to SysML*. The MK/OMG Press, 2008. [11](#), [13](#)
- [5] OMG. Systems modeling language version 1.2. Technical report, 2010. [3](#), [22](#), [30](#), [36](#), [41](#)
- [6] Pascal Roques. *SysML par l'exemple*. Eyrolles, 2009. [14](#), [16](#), [36](#)