



Elements of System Dynamics

Antoine B. Rauzy

PART 2. THE Σ^{TM} LANGUAGE

LECTURE 4. ARCHITECTURE OF MODELS

LECTURE 4. ARCHITECTURE OF MODELS

Key notions:

- Virtual variables
- Object- and prototype-oriented paradigm
- Systems/variables/connections
- Prototypes/cloning, classes/instances
- Composition, aggregation, inheritance

Thesis

Models of complex systems need to be **structured**. Their structure reflects, at least to some extent, the **architecture** of the systems they represent. Hence the thesis:

$$\text{Behaviors} + \text{Structures} = \text{Models}^*$$

Meaning and practical consequences:

- Any modeling language is the combination of a **mathematical framework** to describe the behavior of the system under study and a **structuring paradigm** to organize the model.
- The choice of the **appropriate mathematical framework** for a model depends on which aspect of the system is under study
- **Structuring paradigms** are to a very large extent **independent** of the chosen mathematical framework. They can be studied on their own.

(*) In reference to Wirth's seminal book "Algorithms + Data Structures = Programs"

Agenda

Section 1. Case Study

Section 2. Structure versus Behavior

Section 3. Virtual Variables

Section 4. Cloning

Section 5. Classes and Instances

Section 6. Prototype/Cloning versus Class/Instance

Section 7 Inheritance

Section 8. Back to the Case Study

Section 9. Aggregation

Section 10. Wrap-up

LECTURE 4. ARCHITECTURE OF MODELS

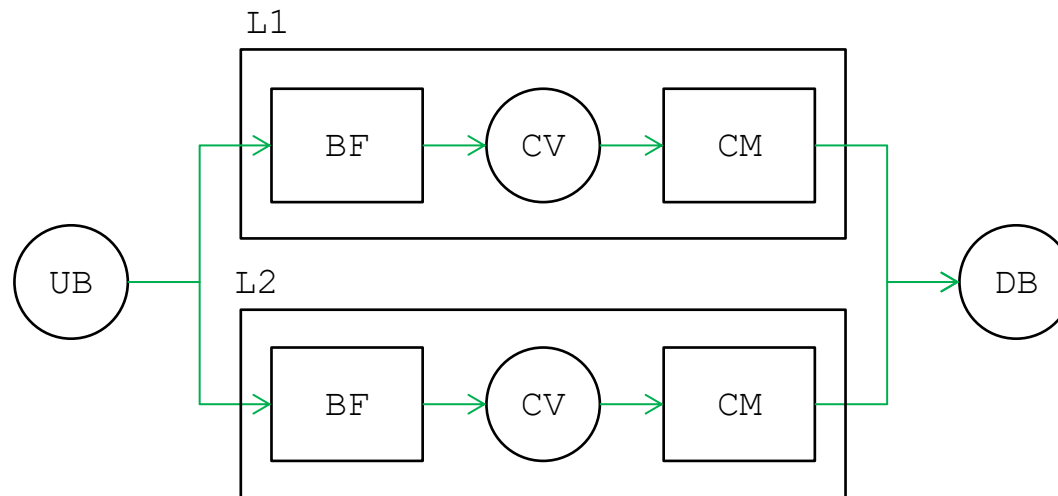
SECTION 1. CASE STUDY

Tablet Packaging System

A pharmaceutical plant packages oral solid tablets into PVC/Aluminum blister packs. To meet high demand and provide redundancy, the plant operates two identical blister-packaging lines L1 and L2 in parallel. Each line consists of:

- A blister forming and tablet loading machine BF (10 tablets per blister sheet).
- A small accumulation conveyor of partially sealed blister sheets CV.
- A cartoning machine CM that inserts blister sheets into folding cartons (2 blister sheets per carton).

Both lines load tablets from an upstream buffer UB and upload cartons into a downstream buffer DB.



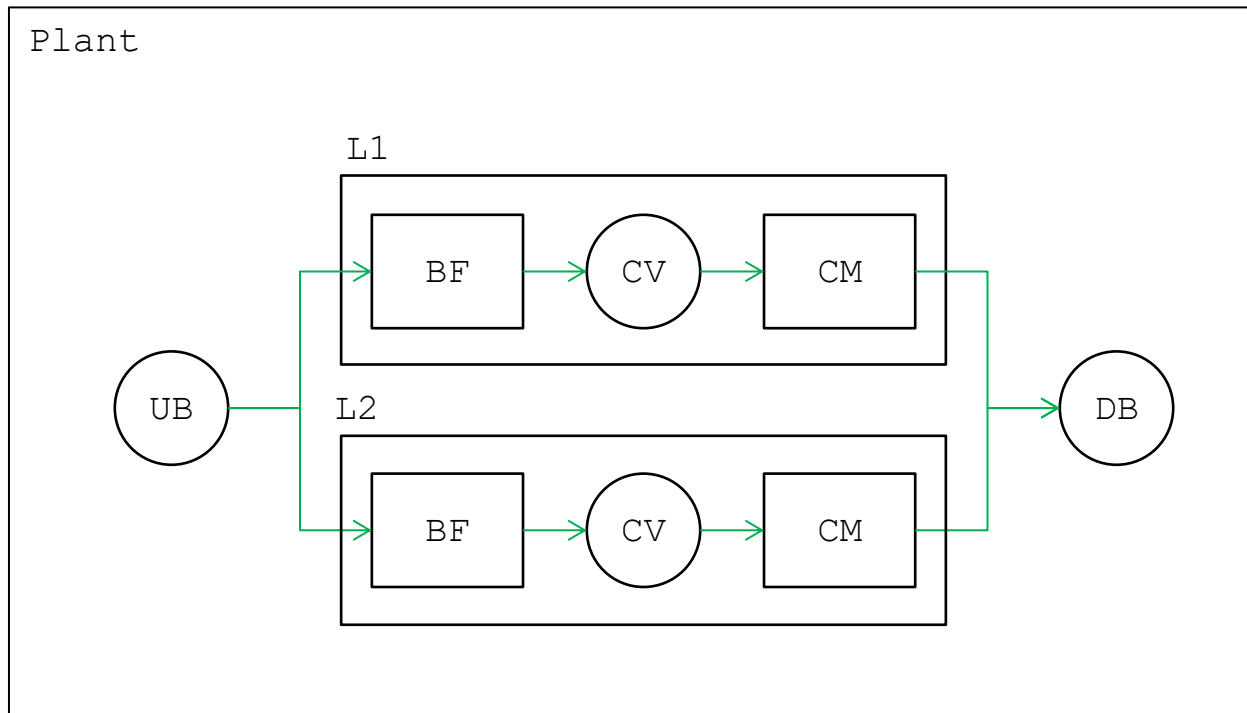
Data

Arrival of tablets to the upstream buffer	300 tablets every 29-31 seconds
Blister thermoforming	10 blisters every 20 seconds
Conveyor capacity	50 blister sheets
Cartoning	1 carton every 3.5-4 seconds
Departure of cartons from the downstream buffer	20 cartons every 40 seconds

LECTURE 4. ARCHITECTURE OF MODELS

SECTION 2. STRUCTURE VERSUS BEHAVIOR

System of Interest

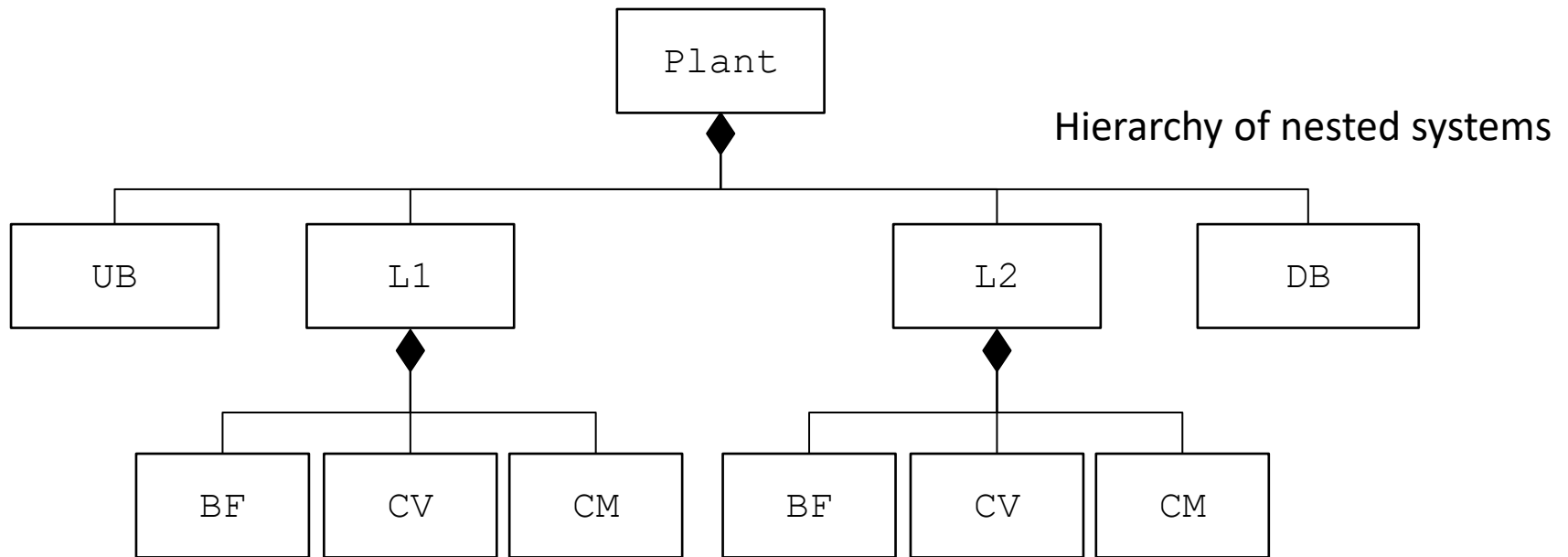


Composition

A system is a **container** for other objects, including and systems. Each system is a **prototype**: it has a unique occurrence in the model.

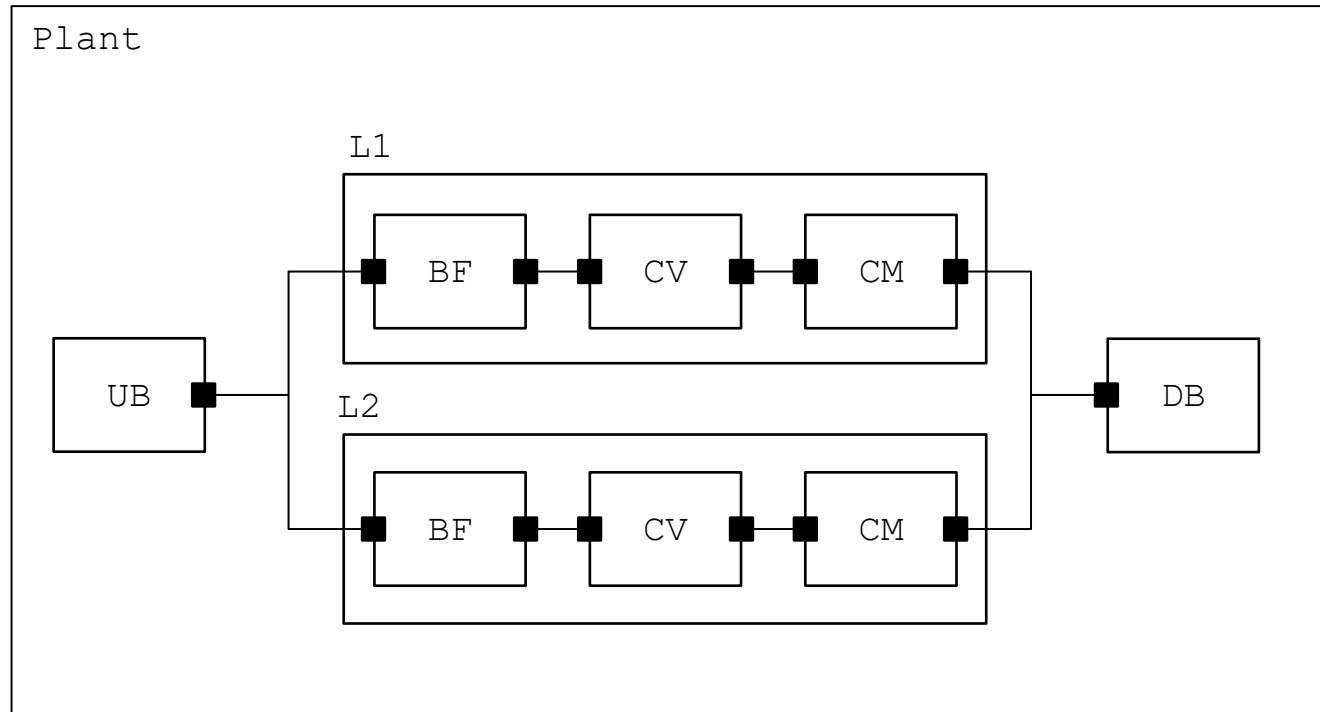
The system “Plant” **composes** the systems “UB”, “L1”...

The system “L1” **is part of** the system “BF”, “CV”.



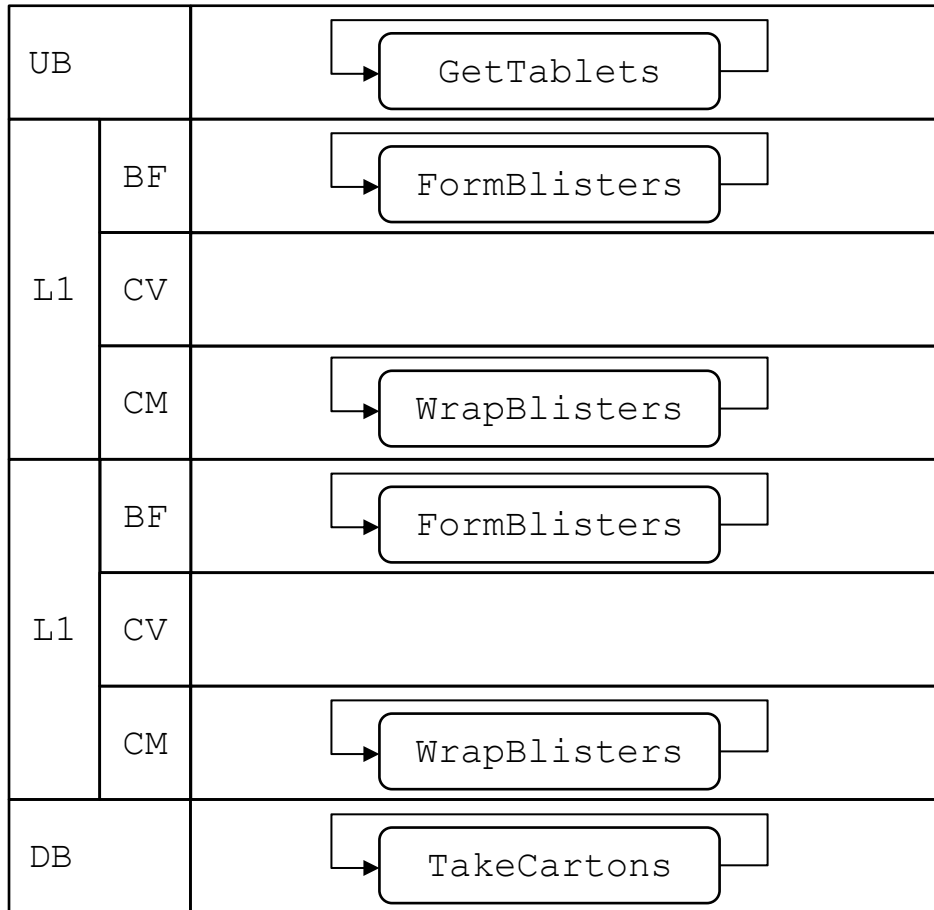
◆ — Graphical representation of the composition in UML/SysML

Block Diagram Decomposition



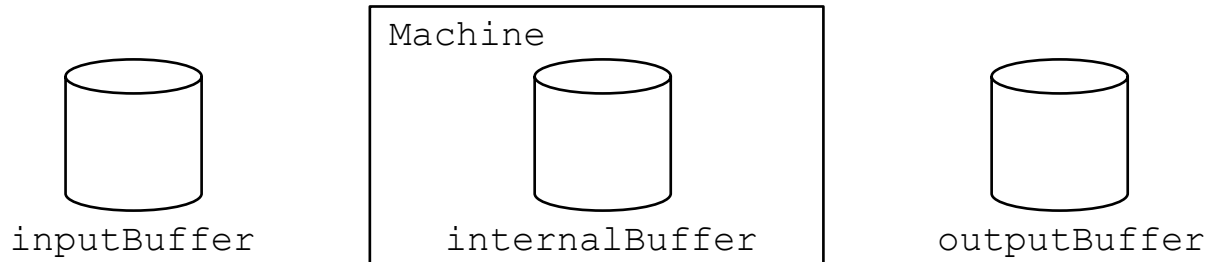
On this example, a block diagram representation provides a much better insight on the structure of the system as it makes it possible to describe **connections** between components.

Activities



All activities are performed in parallel

Key Metaphor

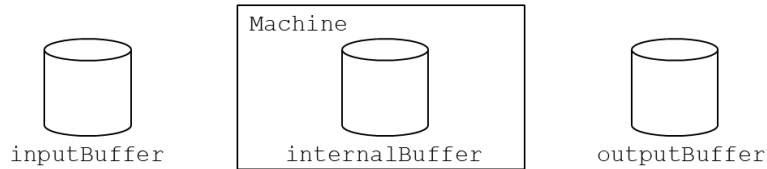


```
activity Machine.Produce
  trigger:
    return state==STANDBY and inputBuffer>0;
  start: {
    state = WORKING;
    inputBuffer -= 1;
    internalBuffer += 1;
  }
  completion: {
    state = STANDBY;
    internalBuffer -= 1;
    outputBuffer += 1;
  }
  duration:
    return 1;
end
```

CHAPTER 5. ARCHITECTURE OF MODELS

SECTION 3. VIRTUAL VARIABLES

Paths



Problem: inputBuffer and outputBuffer are outside Machine

```
activity Machine.Produce
  trigger:
    return state==STANDBY and main.inputBuffer>0;
  start: {
    state = WORKING;
    main.inputBuffer -= 1;
    internalBuffer += 1;
  }
  completion: {
    state = STANDBY;
    internalBuffer -= 1;
    main.outputBuffer += 1;
  }
  duration:
    return 1;
end
```

We can use **paths** to solve the problem.

However, we would like to think about the machine independently of how it is connected to its environment (here inputBuffer and outputBuffer)

Virtual Variables (1)

Virtual variables are a way to solve the previous problem.

```
system Machine
  State state(init = STANDBY);
  int internalBuffer(init = 0);
  virtual int inputBuffer, outputBuffer(init = 0);
  activity Produce
    trigger:
      return state==STANDBY and inputBuffer>0;
    start: {
      state = WORKING;
      inputBuffer -= 1;
      internalBuffer += 1;
    }
    completion: {
      state = STANDBY;
      internalBuffer -= 1;
      outputBuffer += 1;
    }
    duration:
      return 1;
  end
end
```

Virtual Variables (2)

Virtual variables can be **actualized**, i.e. replaced by other, concrete, variables.

```
system Station
  int buffer1(init = 3);
  int buffer2(init = 0);
  system Machine
    ...
  end
  actualizes Machine.inputBuffer as buffer1;
  actualizes Machine.outputBuffer as buffer2;
end
```

In the model as assessed, the variables `inputBuffer` and `outputBuffer` of `Machine` are respectively replaced by the variables `buffer1` and `buffer2`. The virtual variables mechanism is impossible to achieve using ports (and vice-versa).



Virtual variables are a form of **aggregation**. A system aggregates a variable (or any other component) if it **uses** it without composing it.

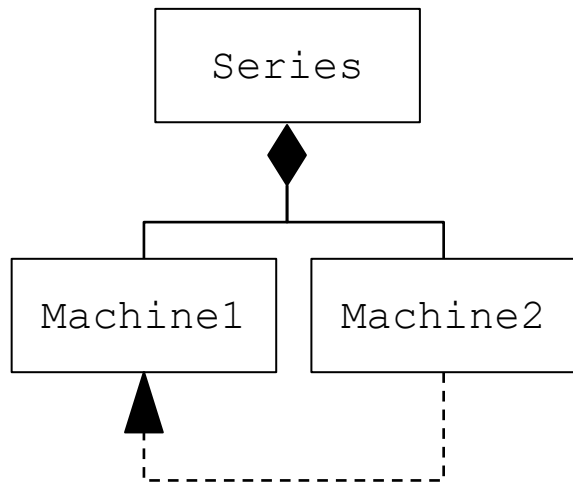
CHAPTER 5. ARCHITECTURE OF MODELS

SECTION 4. CLONING

Definition

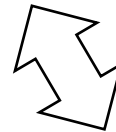
A system may contain similar components, e.g. the two lines our example. Duplicating the code for these subsystems is both tedious and error prone. It makes difficult the maintenance of the model.

Cloning is a way to reflect in the model the presence of similar subsystems.



```
system Series
  system Machine1
  ...
end
clones Machine1 as Machine2;
end
```

Model as
designed



```
system Series
  system Machine1
  ...
end
system Machine2
  ...
end
end
```

Model as
assessed

Model as Script

```
CloningDirective ::= 'clones' Path 'as' Path ';' 
```

The cloning directive duplicates the code of the cloned system as it is when the directive is executed.

```
system Series
  system Machine1
    ...
    activity Produce
    ...
  end
end
clones Machine1 as Machine2;
end
```

The activity Operate must be declared before the Machine1 is cloned.

Series1



CHAPTER 5. ARCHITECTURE OF MODELS

SECTION 5. CLASSES AND INSTANCES

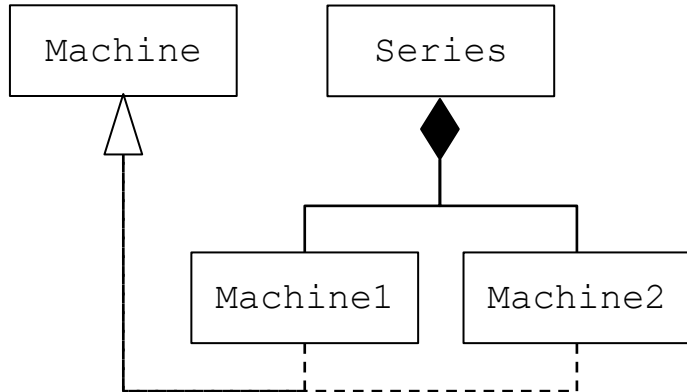
Classes

Classes define on-the-shelf, reusable modeling components.

```
class Machine
  State state(init = STANDBY);
  int internalBuffer(init = 0);
  virtual int inputBuffer, outputBuffer(init = 0);
  activity Produce
    trigger:
      return state==STANDBY and inputBuffer>0;
    start: {
      state = WORKING;
      inputBuffer -= 1;
      internalBuffer += 1;
    }
    completion: {
      state = STANDBY;
      internalBuffer -= 1;
      outputBuffer += 1;
    }
    duration:
      return 1;
  end
end
```

Instances

Once declared, a class can be **instantiated** as many times as required in a model.



◁----- Instantiation (of a class)

```
class Machine
...
end
```

```
system Series
  Machine Machine1;
  Machine Machine2;
end
```

Instances of the class
Machine

Model as
designed



```
system Series
  system Machine1
  ...
end
  system Machine2
  ...
end
end
```

Model as
assessed

Model as Script

```
SystemDeclaration ::=  
    'system' Path SystemBody 'end'  
  
ClassDeclaration ::=  
    'class' Identifier SystemBody 'end'  
  
InstanceDeclaration ::=  
    Identifier Path ';' |  
    Identifier Path SystemBody 'end'
```

As for the cloning directive, instantiating a class duplicates the code of the class as it is when the instantiation directive is executed.
Consequently, the class should be fully described before it is instantiated.

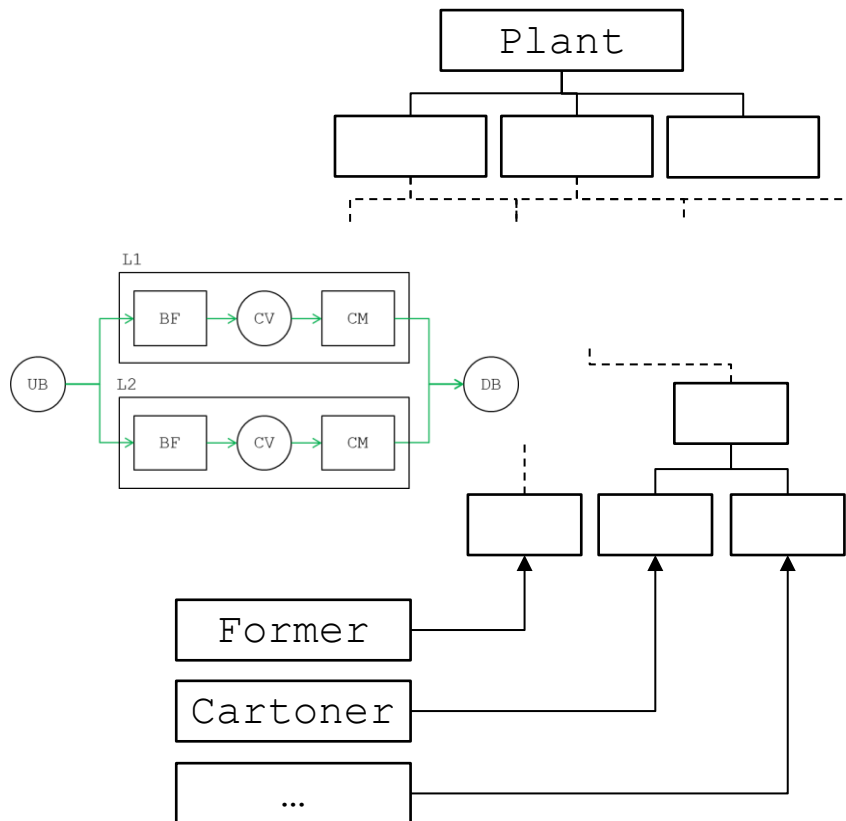
Series2



CHAPTER 5. ARCHITECTURE OF MODELS

SECTION 6. PROTOTYPE/CLONING VERSUS CLASS/INSTANCE

Prototypes versus Classes



- Top-down model design
- System level
- Reuse of modeling **patterns**
- Prototype-Oriented

- Bottom-up model design
- Component level
- Reuse of modeling **components**
- Object-Oriented



system
architecture

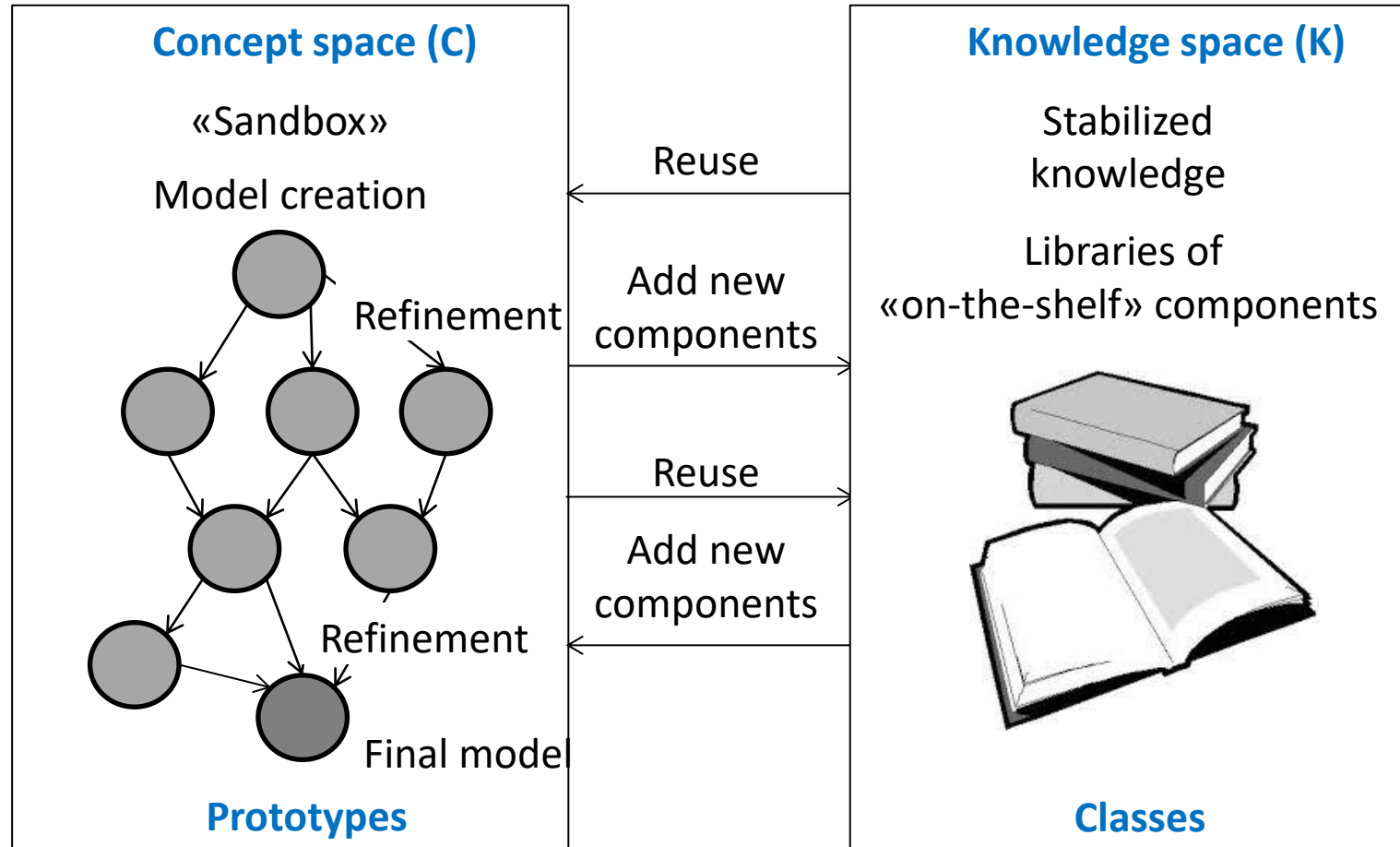


safety



Multiphysics
simulation

Hatchuel C-K Theory of Innovation



CHAPTER 5. ARCHITECTURE OF MODELS

SECTION 7. INHERITANCE

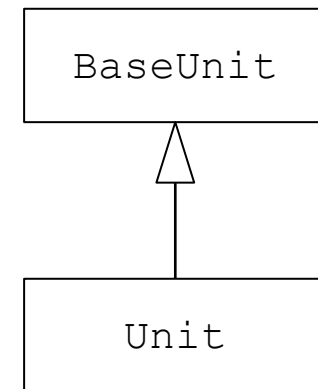
Inheritance

Aside the composition, that defines a “is-part-of” relation, the object-oriented paradigm 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 BaseUnit
  State state(init = WORKING);
  parameter float failureRate = 1.0e-5;
  bool operating(init = false);
end

activity BaseUnit.Operate
  ...
end

class Unit
  extends BaseUnit;
  port bool in = false;
  port bool out = state==WORKING and in;
end
```



◀ Inheritance

Model as Script

```
ExtendDirective ::=  
    'extends' Path ';' |  
    'extends' Path SystemBody 'end'
```

As for the cloning directive, the extend directive duplicates the code of the inherited class as it is when the extend directive is executed. Consequently, the base class should be fully described before it is inherited.

CHAPTER 5. ARCHITECTURE OF MODELS

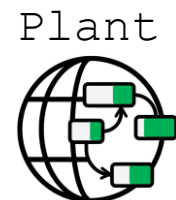
SECTION 8. BACK TO THE CASE STUDY

BlisterFormer

```
class BlisterFormer
  State state(init = STANDBY);
  int internalTablets(init = 0);
  virtual int inputTablets, outputBlister(init = 0);
  parameter int tabletsPerBlister = 10;
  parameter int blisterPerBatch = 20;
  parameter int tabletsPerBatch = tabletsPerBlister * blisterPerBatch;
  parameter float formTime(unit = "s") = 20;
  activity FormBlister
    trigger:
      return state==STANDBY and inputTablets>=tabletsPerBatch;
    start: {
      inputTablets -= tabletsPerBatch;
      internalTablets += tabletsPerBatch;
      state = WORKING;
    }
    completion: {
      internalTablets -= tabletsPerBatch;
      outputBlister += blisterPerBatch;
      state = STANDBY;
    }
  duration:
    return formTime;
end
end
```

Plant

```
system Plant
  InputBuffer IB;
  system L1
    BlisterFormer BF;
    int CV(init = 0);
    Cartoner CM;
    actualizes BF.outputBlisters as CV;
    actualizes CM.inputBlisters as CV;
  end
  clones L1 as L2;
  OutputBuffer OB;
  actualizes L1.BF.inputTablets as IB.tablets;
  actualizes L1.CM.outputCartons as OB.cartons;
  actualizes L2.BF.inputTablets as IB.tablets;
  actualizes L2.CM.outputCartons as OB.cartons;
end
```



Input buffers, BlisterFormer, Cartoner and OutputBuffers are declared as classes for the sake of clarity.

L2 is cloned from L1. Lines could also be declared as classes, but this makes less sense here as they are very specific to this particular model.

CHAPTER 5. ARCHITECTURE OF MODELS

SECTION 9. WRAP-UP

Σ^{TM} Constructs

Σ^{TM} provides constructs to **structure** models. The structure of the model reflects to some extent the structure of the underlying system.

Σ^{TM} combines **prototype-** and **object-oriented** constructs:

- **Prototypes** and **cloning**;
- **Classes** and **instances**;
- **Inheritance**.

Moreover, Σ^{TM} provides constructs to represent **connections** between components (and consequently a certain form of **aggregation**):

- **Ports**;
- **Virtual variables**.

Relations

There are four fundamental operations to structure models:

- **Composition**, which represents a **is-part-of** relation.
- **Instantiation** (of classes) and **cloning** (of prototypes) which represent also a **is-part-of** relation and which copy an existing modeling element.
- **Inheritance**, which represent a **is-a** relation and copies an existing modeling element (usually a class).
- **Aggregation** which represents a **uses** relation.

Recommend Readings

Books or articles about object-oriented programming:

Mauricio Abadi and Luca Cardelli. A Theory of Objects. Springer-Verlag. New-York, USA. ISBN 978-0387947754. 1998.

Bertrand Meyer. Touch of Class: Learning to Program Well With Objects and Contracts. Springer-Verlag. Berlin and Heidelberg, Germany. ISBN 978-3540921448. 2009.

Erich Gamma, Richard Helm, Ralph Johnson and John Vlissides. Design Patterns -- Elements of Reusable Object-Oriented Software. Addison-Wesley. Boston, MA 02116, USA. ISBN 978-0201633610. October 1994.