Contents

- The problem of requirements modelling
- Limits of the use cases
- Problem frames
- Problem frames with UML
- Dealing with time: extending OCL
- Possible evolutions of the proposed method
- Conclusions

Problem frames with UML

- How to apply UML to model the Environment, problem domain, and shared phenomena.
- Requirements (i.e., the desirable properties of the whole system) are expressed by means of state diagrams and OCL.
- Correctness arguments are also addressed.
- In order to illustrate the technique, a few problems frames are presented and modelled by means of UML.
- Integration of PFs and scenario-based modelling is also illustrated

Motivations

- Problem frames (PFs) drive developers to understand and describe the problem to be solved, which is crucial for a successful development process.
  - PFs can dramatically improve the early lifecycle phases in software projects.
- PFs are far less popular than other less rigorous approaches.
  - The notation has some limitations that make it not very appealing
  - “Impedance mismatch” with design languages, namely UML
- UML is popular, but not very good for requirements modeling
  - low precision and formality

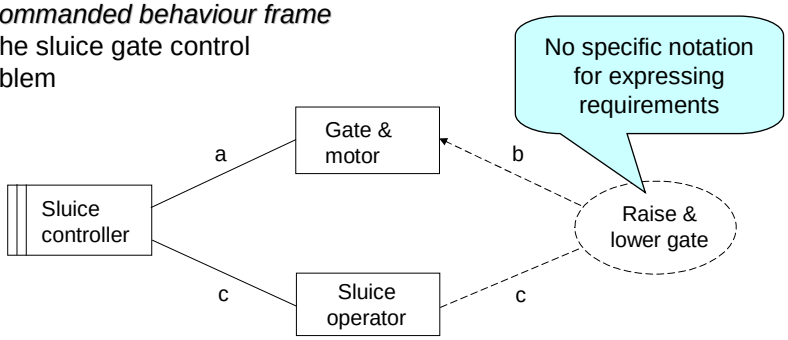
The basic idea

➔ Combine the problem frames approach and UML

- How: make the PF approach seamlessly applicable in the context of the familiar UML language.
  - PFs get a popular, easy to use, expressive notation
  - UML gets a sound, effective, precise method for requirements specification
- Using UML as the notation underlying PFs and as a design language smoothes the transition from the requirement elicitation and modeling phase to the design phase.
  - Moreover, it makes easier to represent traceability relations, since requirements and elements of the solution are represented in a homogeneous way.

A Problem Frame

- A *commanded behaviour frame* for he sluice gate control problem



a: SC!{Clockw, Anti, On, Off}
GM!{Top, Bottom}

b: GM!{Open, Shut, Rising, Falling}
c: SO!{Raise, Lower, Stop}

Readability could be better

A development process using problem frames

Analyst

(Problem frames)

- Impedance mismatch problem

Modeler

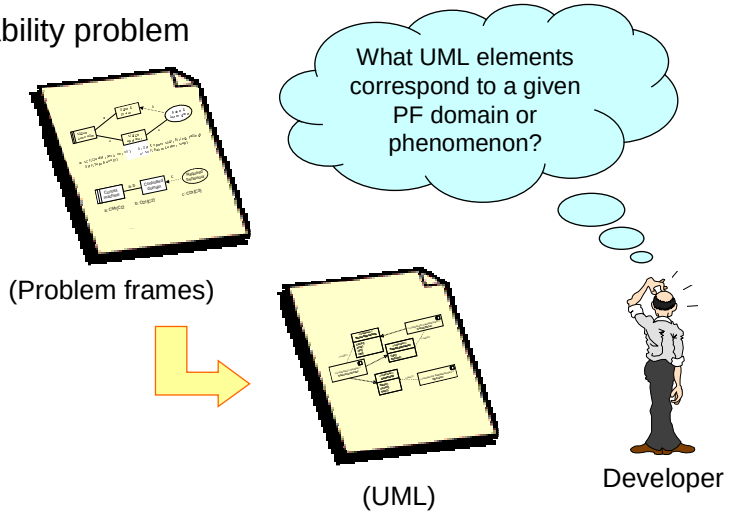
(UML)

Programmer

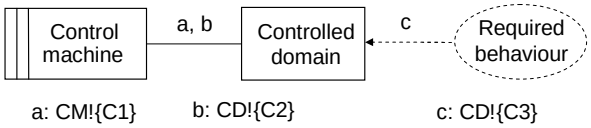
If the programmer does not understand a piece of the specifications, it may found difficult to read the requirements, since he does not understand problem frames (they are not very popular among programmers…)

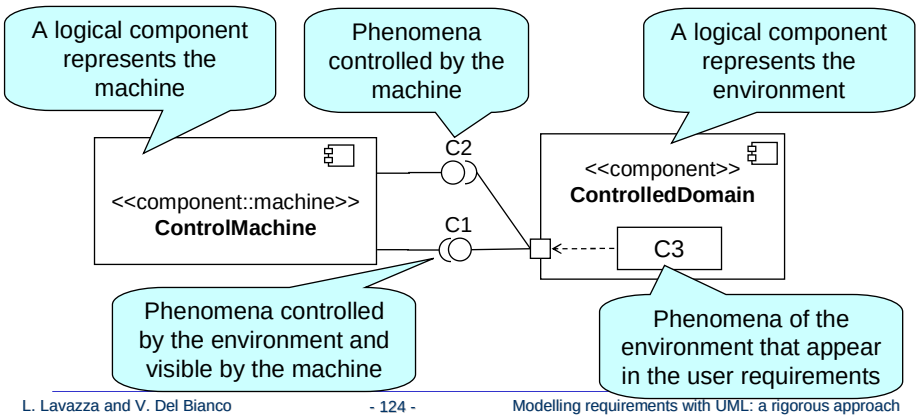
A development process using problem frames

- Traceability problem

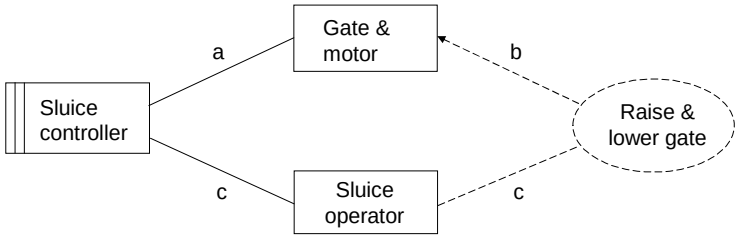


Problem Frames with UML





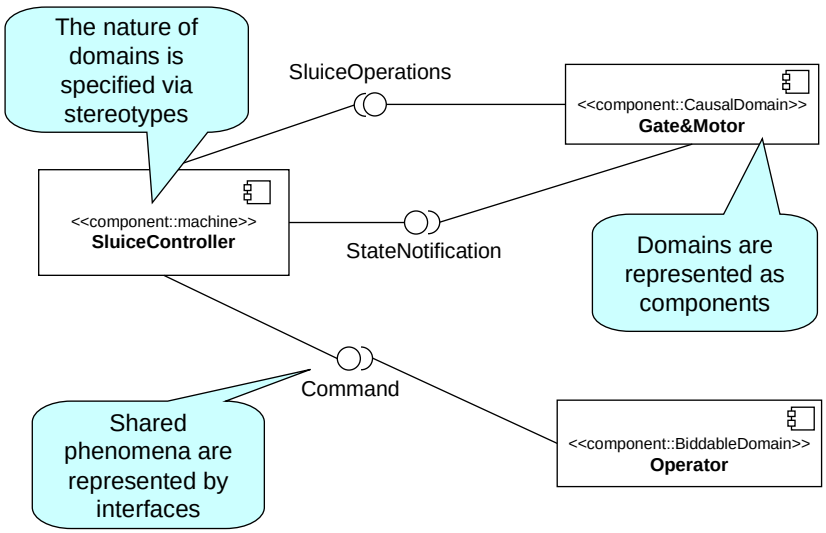
The sluice gate control frame



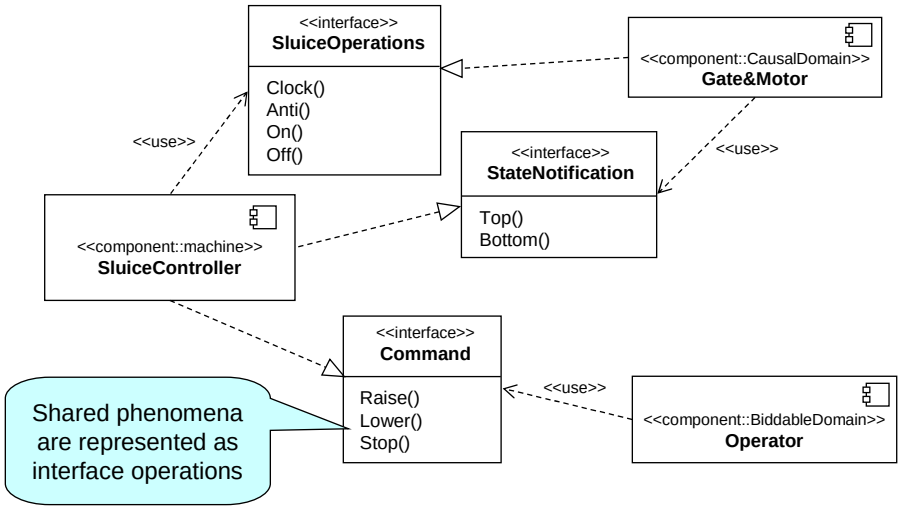
a: SC!{Clockw, Anti, On, Off}
GM!{Top, Bottom}

b: GM!{Open, Shut, Rising, Falling}
c: SO!{Raise, Lower, Stop}

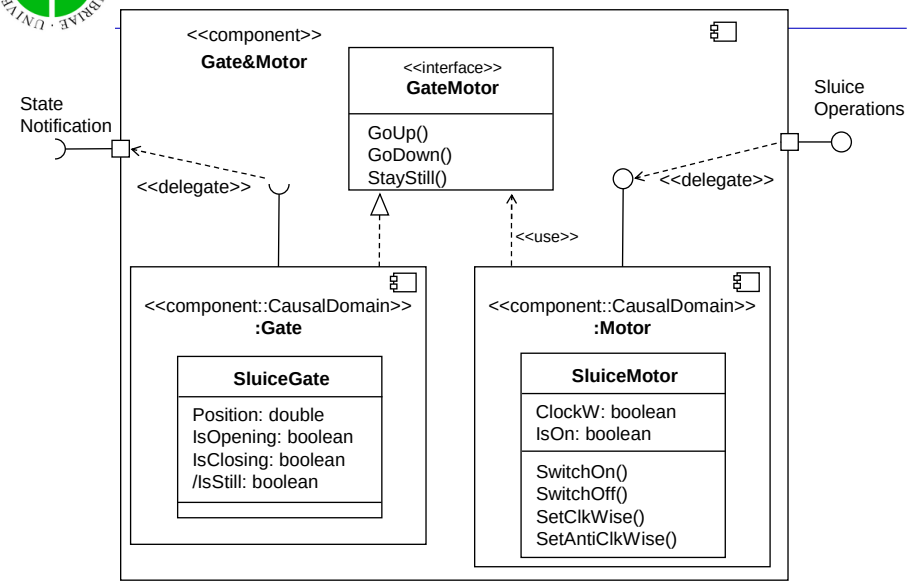
The sluice gate control frame with UML (1)



The sluice gate control frame with UML (2)



The sluice gate control: specifying details



The sluice gate control: specifying behaviour

- The behaviour of the domain can be specified in several ways.
  - In English: the gate is lowering (i.e., closing) iff the motor is on and working anticlockwise and the position of the gate is <1
  - By means of a statechart (as seen before)
  - By means of some logic language. For this purpose, UML provides the OCL (Object Constraint Language)

```
context Gate&Motor inv:
      SluiceGate.IsClosing = (SluiceMotor.IsOn and
      not SluiceMotor.ClockW and SluiceGate.position<1)
```


The sluice gate control: specifying behaviour

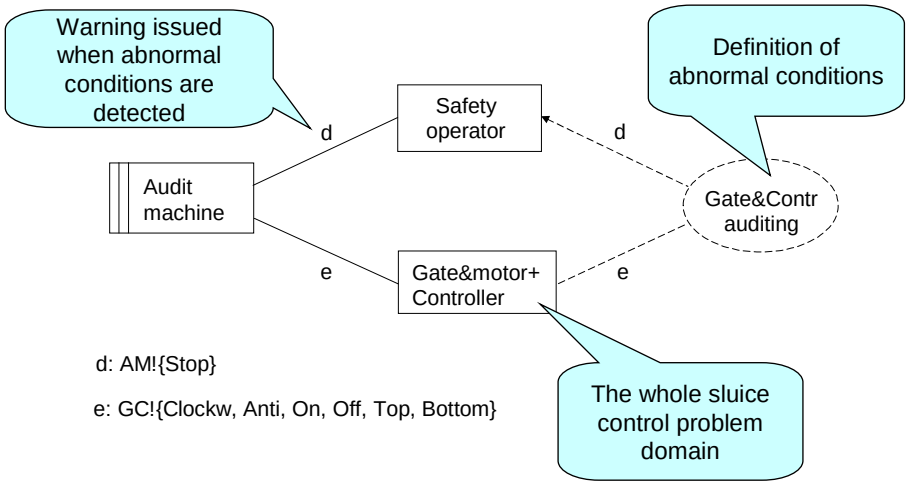
- Unfortunately, OCL is suitable for every property we may wish to express.
- Example:
  - the motor is on iff an On command arrived, and since then no Off command arrived
- This example requires that we can refer to
  - the current time,
  - the time the On command was issued,
  - The interval between these two times.
- OCL simply cannot deal with all these times.
  - We shall see how to overcome this problem.

The reliability concern

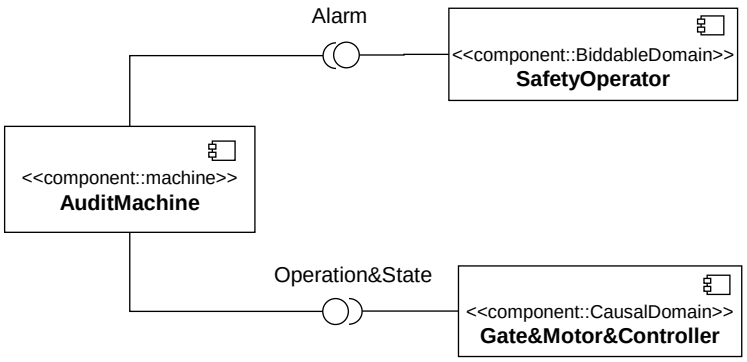
- We specified that the motor is on iff an On command arrived, and since then no Off command arrived: this is true only if we do not consider failures.
- ➔ Reliability concern
  - It is a subproblem
- Subproblems are best identified as *projections* of the original problems
  - subsets of elements and phenomena are relevant to the subproblem.
- Reliability is one of the most common subproblems, since often it is necessary to describe the main problem, and then to take into account reliability issues.

Sluice gate controller: auditing reliability

- An information display frame



Sluice gate auditing with UML



Specifying abnormal conditions

- The abnormal condition corresponding to the failure to complete an operation within the expected time can be specified as follows:

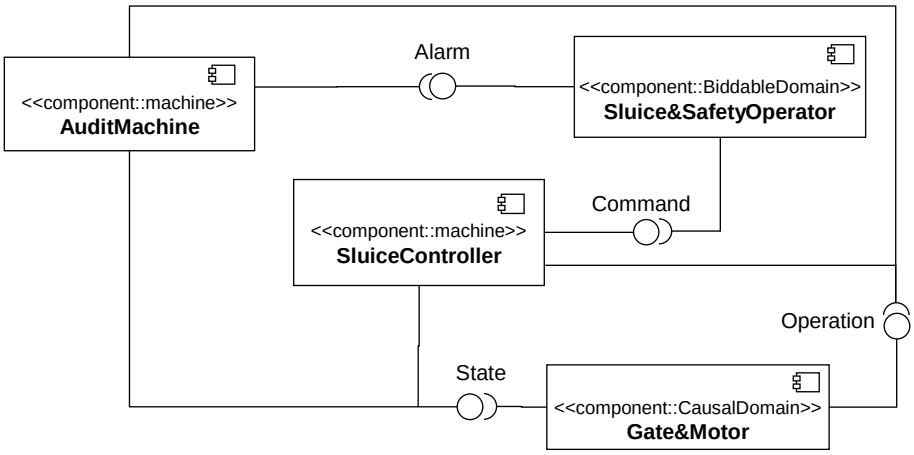
  If the condition for starting lowering the gate (i.e., the motor received the `AnticlockWise` and `On` commands) was verified `D` time unites ago, and `D` is big enough to allow the completion of the operation

  and no counter-order was received

  and the gate sensor did non notify the completion of the operation (`Bottom` signal)

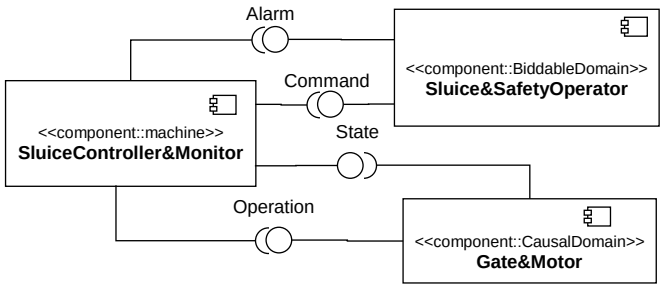
  then a `StopWarning` will be issued within one time unit.

Merging the subproblem diagrams

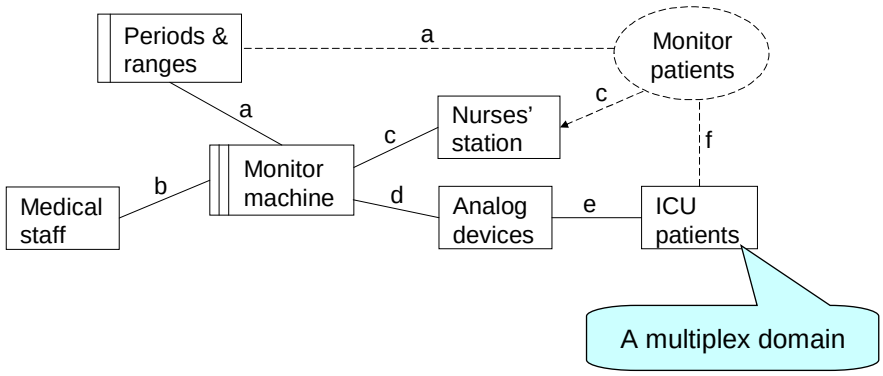


Final diagram

- After merging the two machines:



Patient monitoring: partial problem diagram



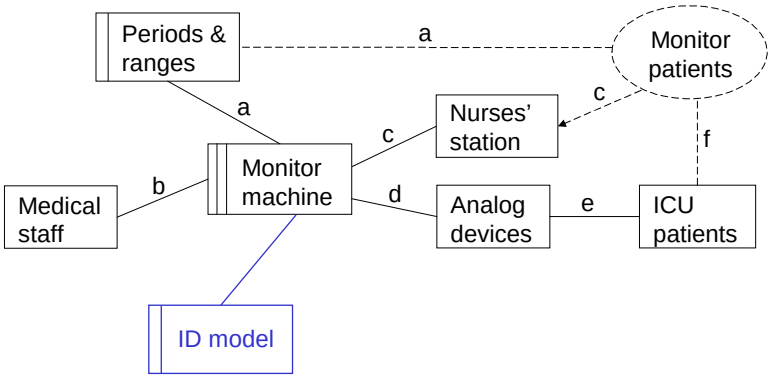
a: Period, Range, PatientName
b: EnterPeriod, EnterRange, EnterPatientName
c: Notify
d: RegisterValue
e: FactorEvidence
f: VitalFactor, Patient

The identity concern

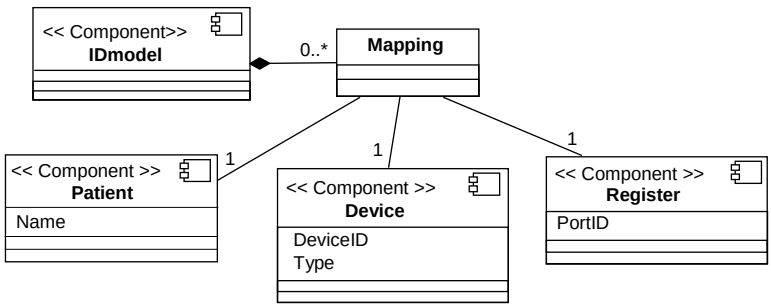
- The machine must monitor every patient according to the periods and ranges specified for him/her by the medical staff
- The medical staff identify the patient by name when they enter the period and ranges (b)
- The machine has access to periods and ranges associated with patients' names (a)
- Each patient is connected to a set of analog devices (e)
- The machine gets values that are referred to the devices (d)
  - more precisely, the device and the machine share the value of a register, accessed at a machine port or storage address.
- Somehow, each patient must be associated with the right set of devices and with the right name

The solution of the ID concern

- We can provide an Identities model to be used by the machine.
- The Identities model contains triples <Patient, Device, Register

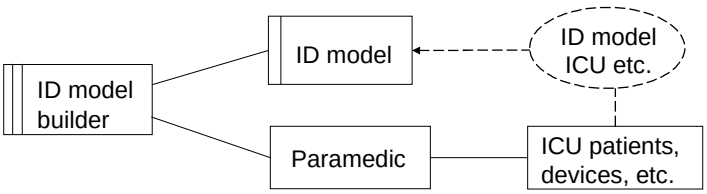


Identities model class diagram

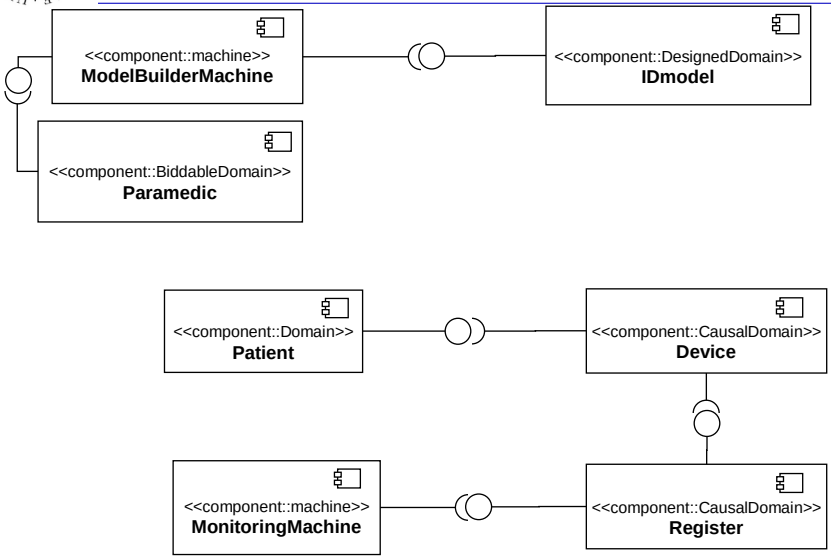


Identities model creation workpieces PF

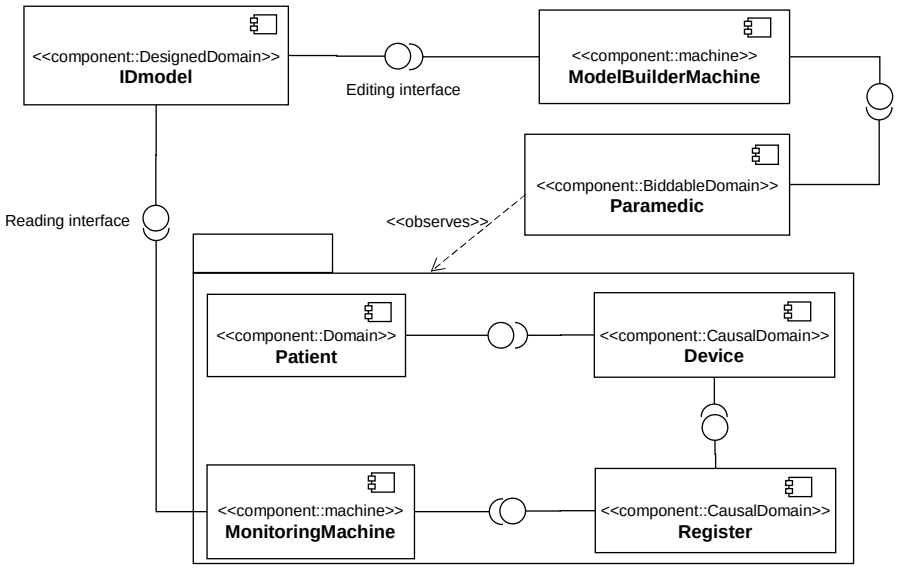
- The creation and maintenance of the identities model domain has to be performed by someone who knows the identity of patients and how they are connected to the system, usually, a paramedic
- The creation and maintenance of the identities model is another a subproblem
  - A workpieces PF



Identities model: integrated view



Identities model: integrated view



Contents

- The problem of requirements modelling
- Limits of the use cases
- Problem frames
- Problem frames with UML
- Dealing with time: extending OCL
- Possible evolutions of the proposed method
- Conclusions

Dealing with time: extending OCL

- UML is limited with respect to the possibility of specifying temporal aspects. By means of OCL it is not possible to reference different time instants in a single OCL formula. Only invariant properties can be formalized, which at most include references to attribute values before or after method execution.
- An extension of OCL is presented to overcome these limitations. Examples are given.

OCL: limitations

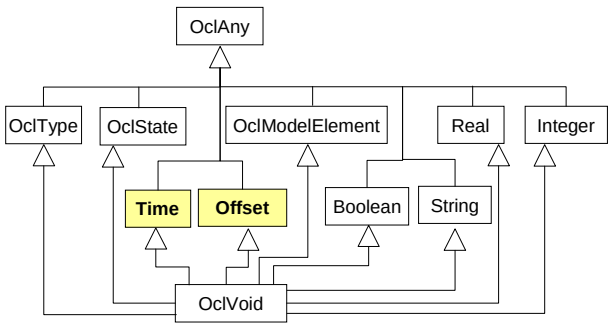
- OCL can be used to state behavioral properties of a system and its parts.
- OCL *cannot* explicitly predicate about the temporal properties of a system
  - Neither in its current form nor in OCL 2.0
  - Only **`inv`** (the Always construct of temporal logic) is available
- When dealing with time-dependent systems, OCL needs to be extended to fully specify temporal aspects.
  - For instance, we need to specify the time distance between events

The OTL language

- OTL is a temporal logic extension to OCL.
  - Based on one fundamental temporal operator
  - It provides the typical basic temporal operators of temporal logics, i.e., Always, Sometimes, Until, etc.
  - It allows to reason about time in a quantitative fashion.
  - Totally integrated with the other UML notations.

OTL extends OCL 2.0

- OTL extends the OCL 2.0 standard library by adding two new classes, Time and Offset.



Our extensions do not require any change in the metamodel of OCL 2.0.
Types Time and Offset are simply new types that are added to the OCL standard library as specializations of OclAny.

New operators

- Methods of class Time
  - `Boolean eval(OclExpression p)`
    - The meaning of t.eval(p) is that **p** is evaluated at time **t**.
    - Consistently with the OCL notation we can write **`p@t`** instead of **`t.eval(p)`**
  - `Time operator +(Offset d)`
  - `Boolean operator ≤(Time t)`
  - `Offset dist(Time t)`
  - `Set(Time) futrInterval(Offset d)`
  - `Set(Time) pastInterval(Offset d)`
  - Methods of class Offset
    - `Offset operator +(Offset d)`
    - `Offset operator -(Offset d)`

Which time?

- `Time` and `Offset` may be discrete or dense, depending on the application at hand.
- The adoption of a possibly dense time has implications on the semantics of the OTL language
  - OCL assumes that quantified variables range only over finite sets and defines the meaning of quantification in terms of finite iterations.
  - In the OTL language the semantics of quantification over time is defined in the same way as in more conventional mathematical logics that include arithmetic.

Defining temporal operators

- Based on method **`eval`**, all other temporal operators can be defined. E.g.,

```
context C
 inv: Lasts(p, d)
```

specifies that p holds at all times between **`now`** and **`now+d`**. It is a shorthand for

```
context C
 inv: let I:Set(Time)=now.futrInterval(d) in
      I->forall(t: Time| t.eval(p))
```

  - **`now`** denotes the time with reference to which the (sub)formula is interpreted.

More temporal operators

Operator	Meaning
`Futr(p,d)`	p is true d time units in the future
`SomF(p)`	p is true sometimes in the future
`AlwF(p)`	p is always true in the future
`WithinF(p,d)`	p is true within d time units in the future
`Until(p,q)`	p holds until q occurs

- Operators referring to the past (**`Past(p,d)`**, **`SomP(p)`**, **`AlwP(p)`**, **`WithinP(p,d)`**, and **`Since(p,q)`**) are similarly defined.

Formal definition of temporal operators

operator	formal definition
`Futr(p,d)`	`p@(now + d)`
`SomF(p)`	`let I: Set(Time) = now.futrInterval(inf) in I->exists(t: Time| p@t)`
`AlwF(p)`	`let I: Set(Time) = now.futrInterval(inf) in I->forall(t: Time| p@t)`
`WithinF(p,d)`	`let I: Set(Time) = now.futrInterval(d) in I->exists(t: Time| p@t)`
`Until(p,q)`	`let I: Set(Time) = now.futrInterval(inf) in I- >exists(t:Time| q@t and Lasts(p,t-now))`

Sluice gate controller: properies

- Here we show how the properties of the sluice gate controller can be expressed by means of OTL

The sluice gate control: specifying behaviour

- The behaviour of the domain can now be specified by means of OTL statements like the following.
- Rule: the motor is on iff an On command arrived, and since then no Off command arrived

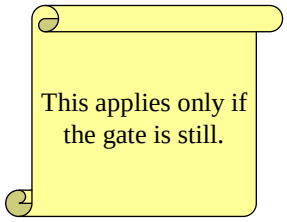
```
context Gate&Motor inv:
 SluiceMotor.IsOn = Since(not SluiceOperations^Off,
                          SluiceOperations^On)
```

- where `Since(p,q)` states that q occurred in the past, and since then p is true.
- Note: this is the specification of the behaviour of the domain. According to the requirements, the motor will be on only when
  - The direction is clockwise and the position is not fully open, or
  - The direction is anticlockwise and the position is not fully closed

The sluice gate control: specifying requirements

- The `Raise` command from the operator is simply transformed by the machine in the pair of commands <`ClockWise`, `On`> which are sent to the `Motor`, unless the `Gate` is already `Open`.

```
context Operator inv:
 (Command^Raise and not SluiceGate.position<0.05) =
 (SluiceOperations^Clock and SluiceOperations^On)
```



The sluice gate control: specifying requirements

- The `Raise` command from the operator is simply transformed by the machine in the pair of commands <`ClockWise`, `On`>, when the command is viable and sensible.

```
context Operator inv:
 ((Command^Raise and SluiceGate.IsStill and
   not SluiceGate.position<0.05) implies
   (SluiceOperations^Clock and SluiceOperations^On)) and

  ((Command^Raise and SluiceGate.IsClosing and
   not SluiceGate.position<0.05) implies
   (SluiceOperations^Off and
    Futr(SluiceOperations^Clock and SluiceOperations^On, D))

context Operator inv:
 SluiceOperations^On = (Command^Raise or Command^Lower)

context Operator inv:
 (Command^Raise and SluiceGate.IsOpening) implies Nop
```

- D is the time taken by the motor and gate to stop

Realistic specifications

- The specifications we saw are sort of “idealized” one: for istance they assume that commands can be issued simultaneously.
- When this is not the case, we must be able to specify that events occurring “close enough” can be considered practically simultaneous.

The sluice gate control: specifying behaviour

- The motor starts in the clockwise direction if...
  - `SwithOn` and `SetClkWise` arrived (in any order!) in a small ST time interval.
  - No `SwithOff` or `SetAntiClkWise` arrived in the same interval

```
context Gate&Motor inv:
 (SluiceMotor^SwitchOn and
  WithinP(SluiceMotor^SetClkWise, ST) or
  SluiceMotor^SetClkWise and
  WithinP(SluiceMotor^SwitchOn, ST))
 and not WithinP(SluiceMotor^SetAntiClkWise, ST)
 and not WithinP(SluiceMotor^SwitchOff, ST))
 implies WithinF(SluiceMotor.IsOn and SluiceMotor.ClockW,MD)
```

where `MD` is the time taken by the motor to react to commands

The sluice gate control: specifying abnormal conditions

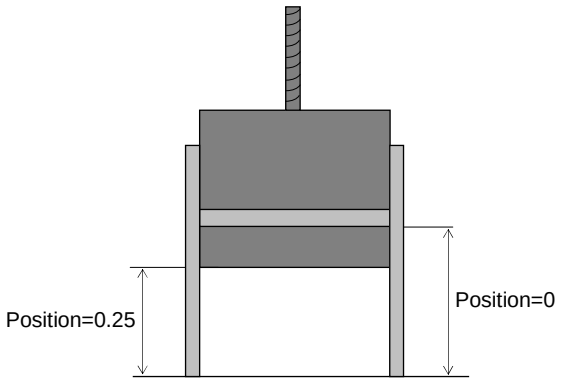
- If the condition for starting lowering the gate (i.e., the motor received the `<AnticlockWise,On>` commands) was verified `D` time unites ago, and `D` is big enough to allow the completion of the operation (`CT` being the expected completion time and `MD` the motor reaction time), and no counter-order was received, and the gate sensor did non notify the completion of the operation (`Bottom` signal) then a `StopWarning` will be issued within one time unit.

```
context AuditMachine inv:
  (Operation^On and Operation^Anticlockwise)@now-D
  and not WithinP((Operation^Clockwise or Operation^Off),D)
  and D >= CT+MD and not WithinP(State^Bottom, D)
 implies WithinF(Alarm^StopWarning, 1)
```

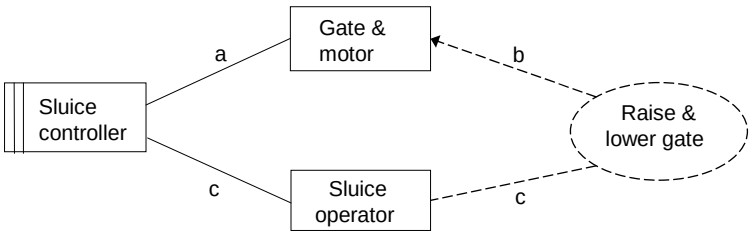
Contents

- The problem of requirements modelling
- Limits of the use cases
- Problem frames
- Problem frames with UML
- Dealing with time: extending OCL
- Problem frames with UML
  - Enhancing Problem Frames with Scenarios and Histories
- Possible evolutions of the proposed method
- Conclusions

The sluice gate



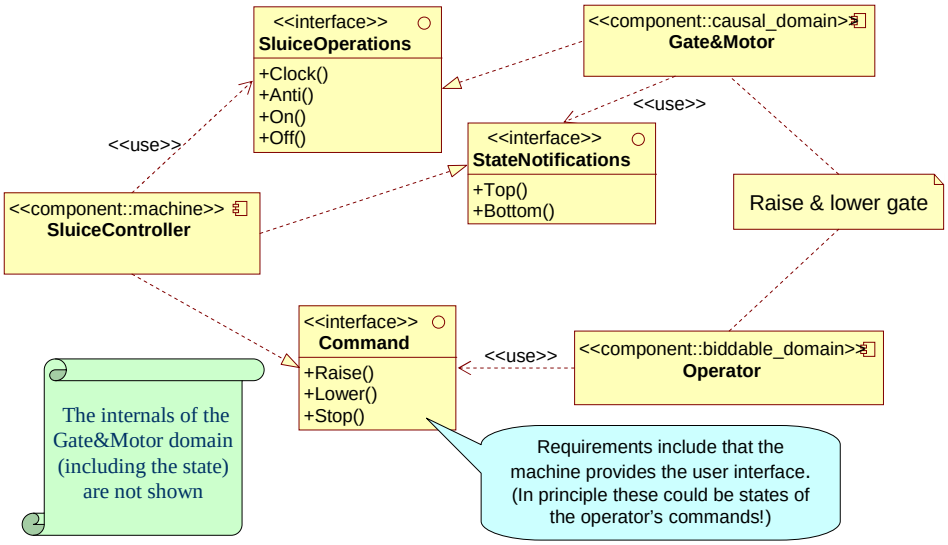
The sluice gate commanded behaviour frame



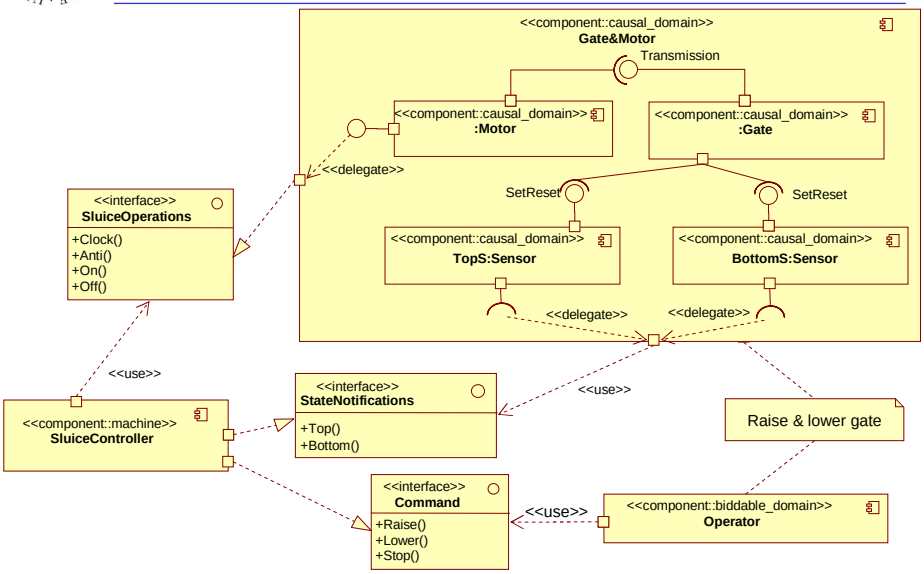
a: SC!{Clockw, Anti, On, Off}
GM!{Top, Bottom}

b: GM!{Open, Shut, Rising, Falling}
c: SO!{Raise, Lower, Stop}

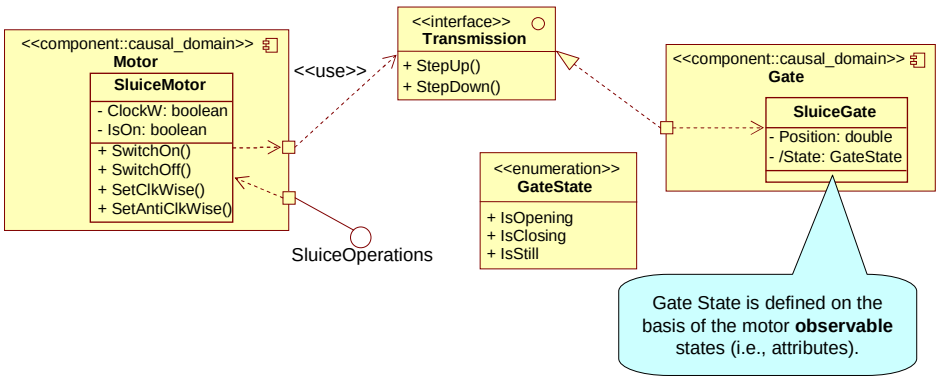
The sluice gate commanded behaviour frame in UML



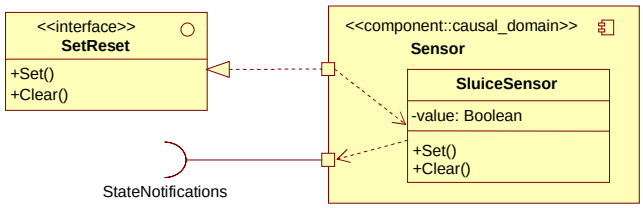
The sluice gate commanded behaviour frame in UML



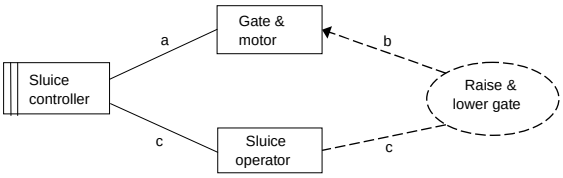
Internal organization of the Motor and Gate



Details of the sensor component



Scenario 1



a: SC!{Clockw, Anti, On, Off}
GM!{Top, Bottom}

b: GM!{Open, Shut, Rising, Falling}

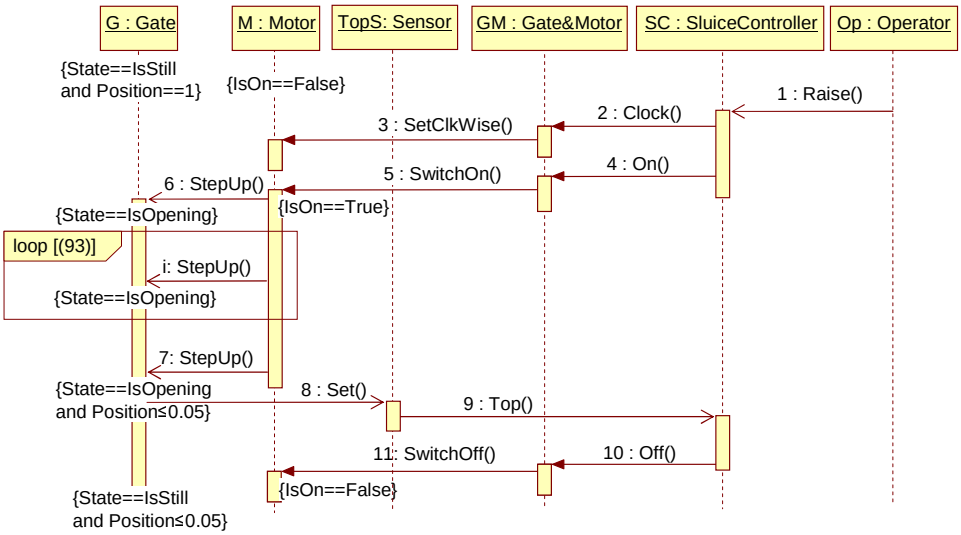
c: SO!{Raise, Lower, Stop}

1. Gate is shut.
2. The operator issues the command to raise (open) the gate.
3. The control machine activates the motor.
4. After a while the gate is open.
5. The control machine stops the motor.
6. The gate is still in the open position.

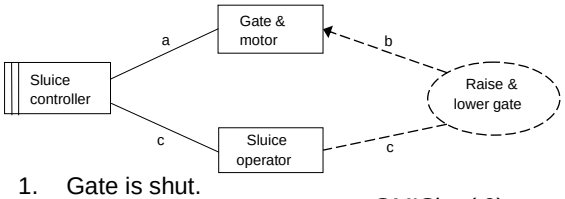
GM!Shut(t0)
SO!Raise(t1)
SC!Clockw(t2)
SC!On(t3)
GM!Rising(t4)
GM!Top(t5)
SC!Off(t6)
GM!Open(t7)

t2-t1 = SC_reaction_time
0< t3-t2 < d
t4-t3 = GM_reaction_time
t5-t4 = Sluice_height/Speed+
GateSensor_reaction_time
t6-t5 = SC_reaction_time
t7-t6= GM_reaction_time

The UML sequence diagram representing scenario 1



Scenario2 requires the introduction of a new state



a: SC!{Clockw, Anti, On, Off}
GM!{Top, Bottom}

b: GM!{Open, Shut, Rising, Falling}

c: SO!{Raise, Lower, Stop}

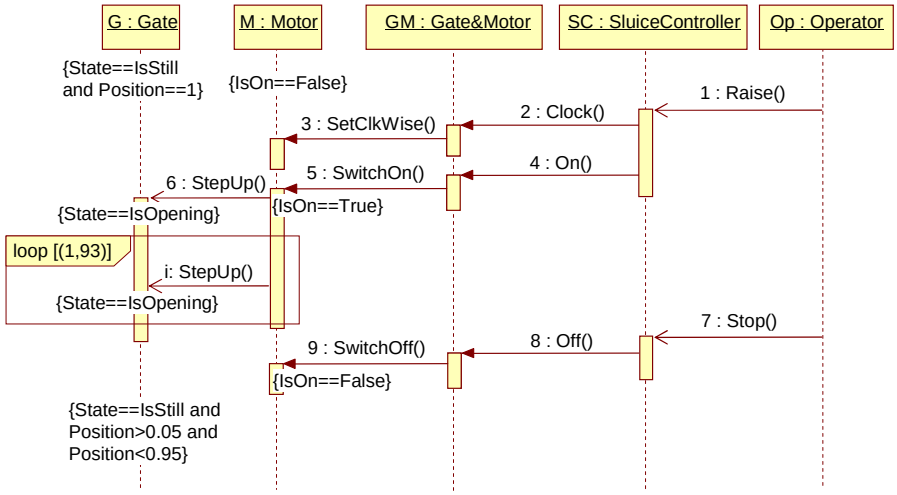
1. Gate is shut.
2. The operator issues the command to raise (open) the gate.
3. The control machine activates the motor.
4. After a while –before the gate is completely open– the operator issues a stop command.
5. The control machine stops the motor.
6. The gate is still in an intermediate position.

GM!Shut(t0)
SO!Raise(t1)
SC!Clockw(t2)
SC!On(t3)
GM!Rising(t4)
SO!Stop(t5)
SC!Off(t6)
GM!Still(t7)

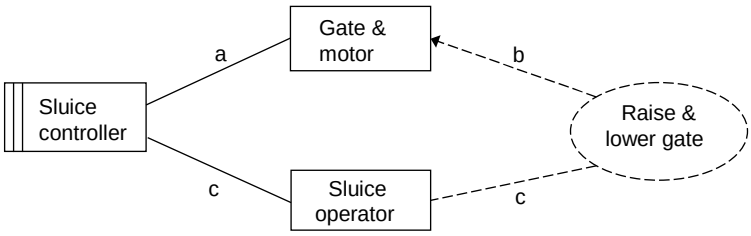
t2-t1 = SC_reaction_time
0< t3-t2 < d
t4-t3 = GM_reaction_time
t5-t4 < Sluice_height/Speed
t6-t5 = SC_reaction_time
t7-t6= GM_reaction_time

Needed to specify the effect of the Off command

The UML sequence diagram representing scenario 2



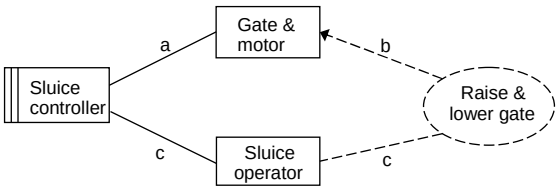
The sluice gate PF updated with Still state



a: SC!{Clockw, Anti, On, Off}
GM!{Top, Bottom}

b: GM!{Open, Shut, Rising, Falling, **Still**}
c: SO!{Raise, Lower, Stop}

Scenario 3 (Prohibited scenario)



a: SC!{Clockw, Anti, On, Off}
GM!{Top, Bottom}

b: GM!{Open, Shut, Rising, Falling, Still}

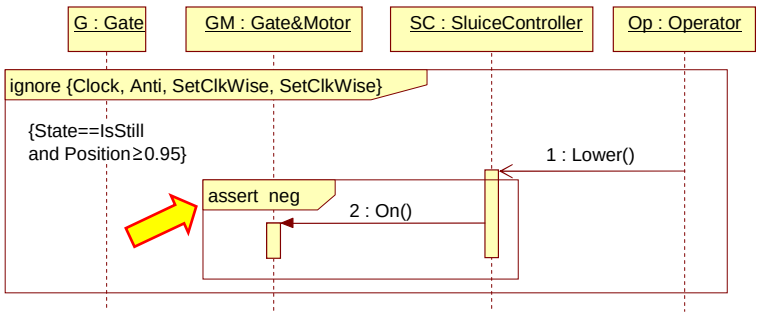
c: SO!{Raise, Lower, Stop}

1. Gate is shut.
2. The operator issues the command to lower (shut) the gate.
3. The control machine –which "knows" the state of the gate– does nothing.
4. The gate is still in the shut position.

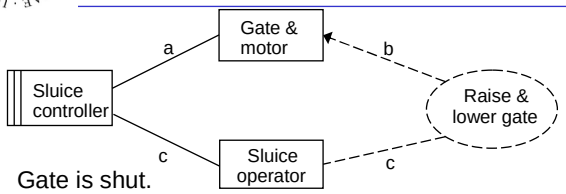
Prohibited scenario

GM!Shut(t0)
SO!Lower(t1)
SC!Clockw(t2) or SC!Anti(t2) or nothing
SC!On(t3)

The UML sequence diagram representing scenario 3



Scenario 4



a: SC!{Clockw, Anti, On, Off}
GM!{Top, Bottom}

b: GM!{Open, Shut, Rising, Falling, Still}

c: SO!{Raise, Lower, Stop}

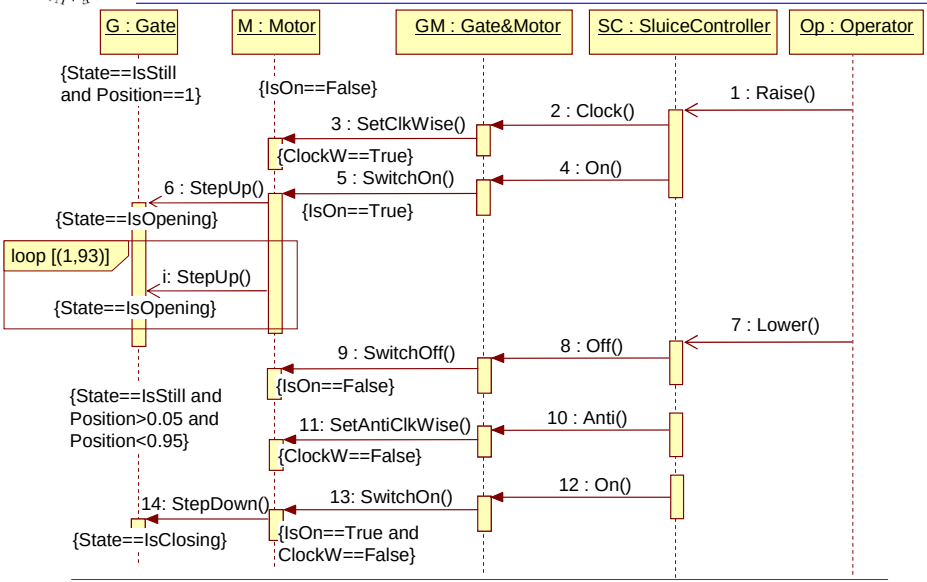
1. Gate is shut.
2. The operator issues the command to raise (open) the gate.
3. The control machine activates the motor.
4. After a while –before the gate is open– the operator issues a lower (close) command.
5. The control machine first stops the motor, then inverts the movement, and finally restarts the motor.

GM!Shut(t0)
SO!Raise(t1)
SC!Clockw(t2)
SC!On(t3)
GM!Rising(t4)
SO!Lower(t5)
SC!Off(t6)
GM!Still(t7)
SC!Anti(t8)
SC!On(t9)
GM!Falling(t10)

It could be t7>t8!

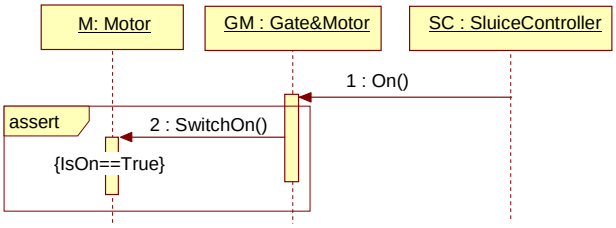
t5-t4 < Sluice_height/Speed
t6-t5 = SC_reaction_time
t7-t6=t10-t9= GM_reaction_time
0< t8-t6 < d
0< t9-t8 < d

The UML sequence diagram representing scenario 4



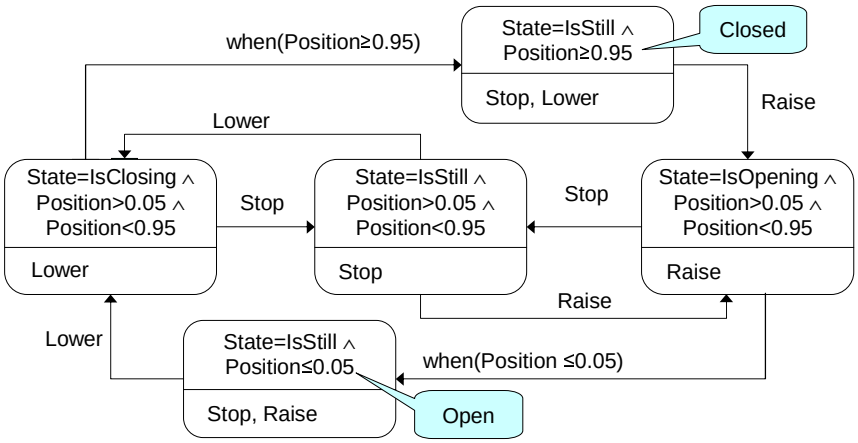
A SD asserting a behavioural rule

- Scenario fragments can be used to state general behavioural rules that the system has to obey in all cases.
- For instance, this sequence diagram states that whenever the **`Gate&Motor`** receives an **`On`** command, it issues a **`SwitchOn`** command to the motor.

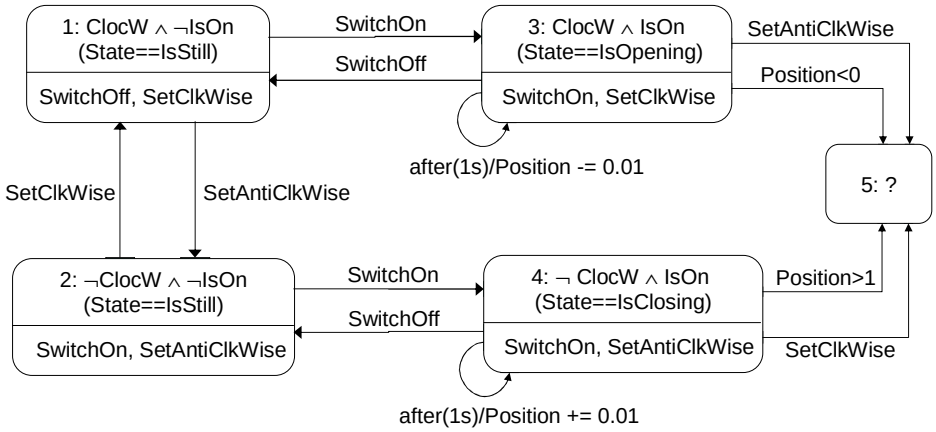


Requirements: reaction to commands and events

- Requirements are expressed as effects on the problem domain caused directly by the user's commands.
  - More straightforward than in MJ's book
  - Note: condition on Position rather than on the sensor state!



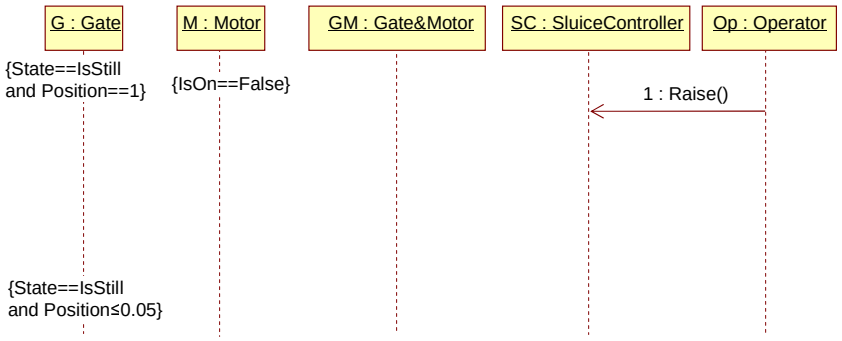
Problem domain behaviour



Scenarios for user requirements modelling

- These scenarios involve only elements from the problem domain (including the operator). In our case, every scenario specifies the desired effects of a user's command (or sequence of commands, or change events, such as reaching the completely open of closed positions) on the problem domain. These scenarios should start with an Operator's command and end with an effect on the controlled domain.

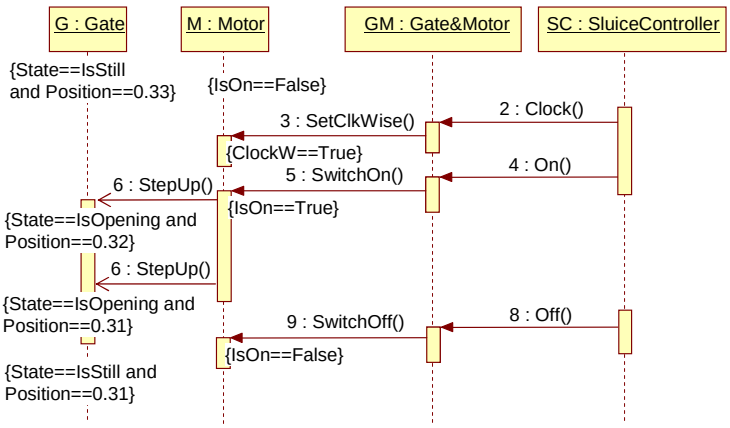
Scenario describing user requirements



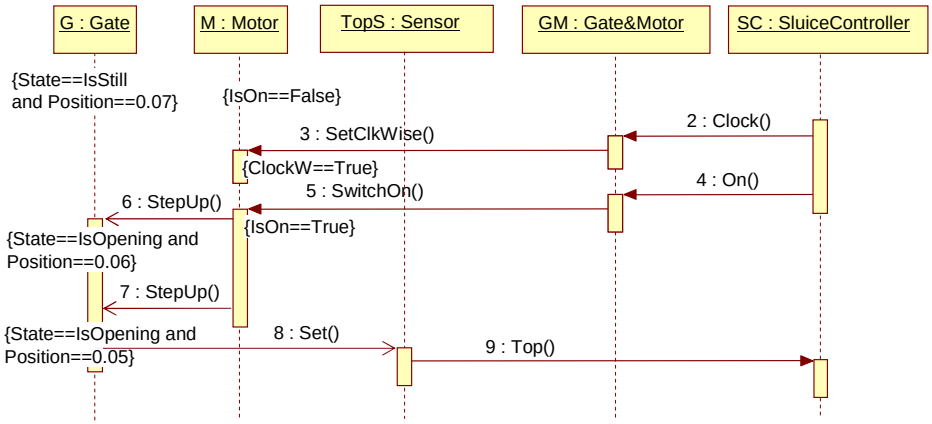
Scenarios for problem domain modelling

- These scenarios involve only elements from the problem domain. In our case, the effects of signals and commands applied directly at the motor are described. These scenarios should start with an Operator's command and end with an effect on the controlled domain.

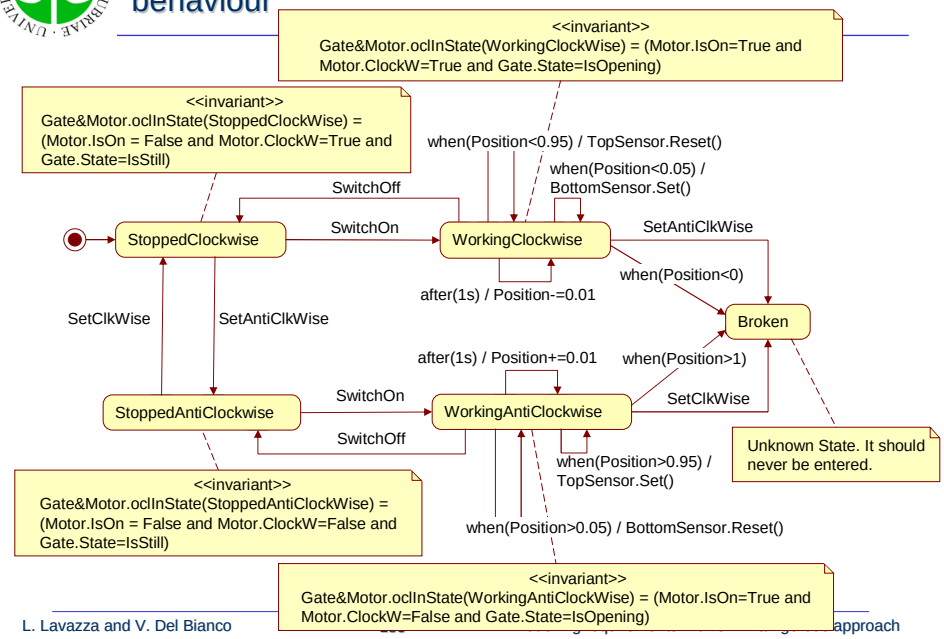
Scenario describing the problem domain (gate rising)



Scenario describing the problem domain (gate rising and sensor reaction)



UML statechart describing the problem domain behaviour



Contents

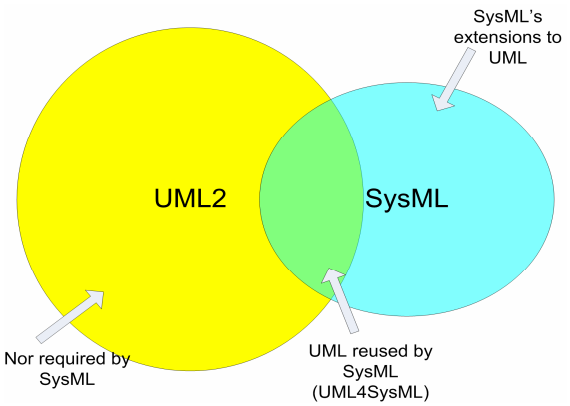
- The problem of requirements modelling
- Limits of the use cases
- Problem frames
- Problem frames with UML
- Dealing with time: extending OCL
- Possible evolutions of the proposed method
- Conclusions

Evolutions of the proposed method

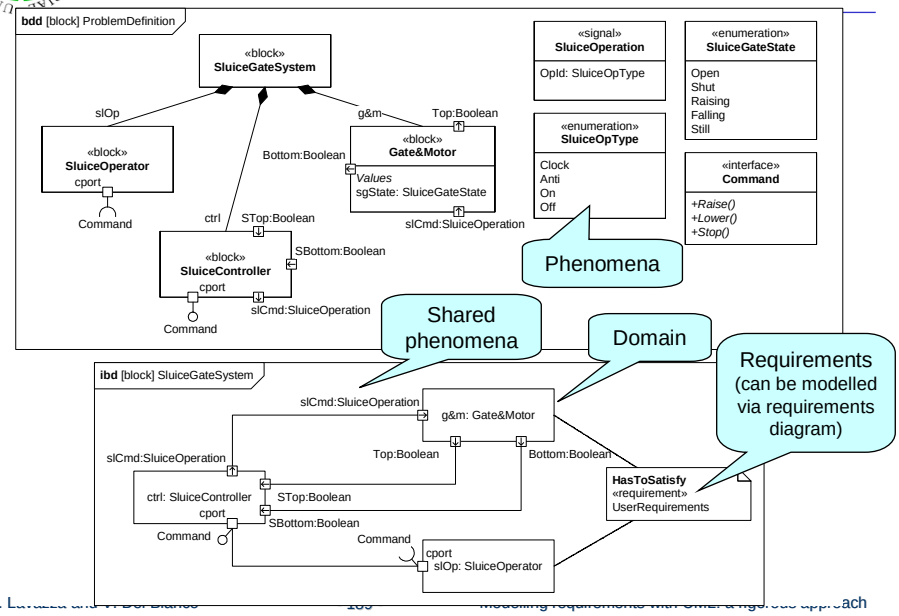
- New modelling language have been proposed to overcome UML weaknesses.
- For instance, SysML is intended to provide a bettrer support for system modelling
  [*OMG System Modeling Language (OMG SysML) Specification*, May 2005. Final Adopted Specification,ptc/06-05-04.]

SysML

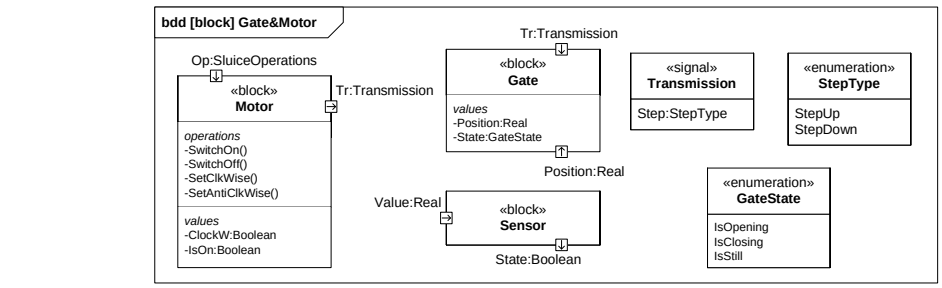
- A general-purpose graphical modeling language
  - for specifying, analyzing, designing, and verifying complex systems



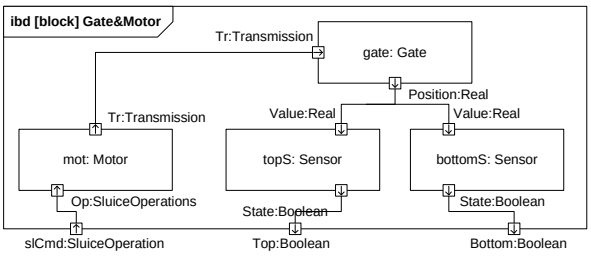
Problem frames and domain modelling with SysML



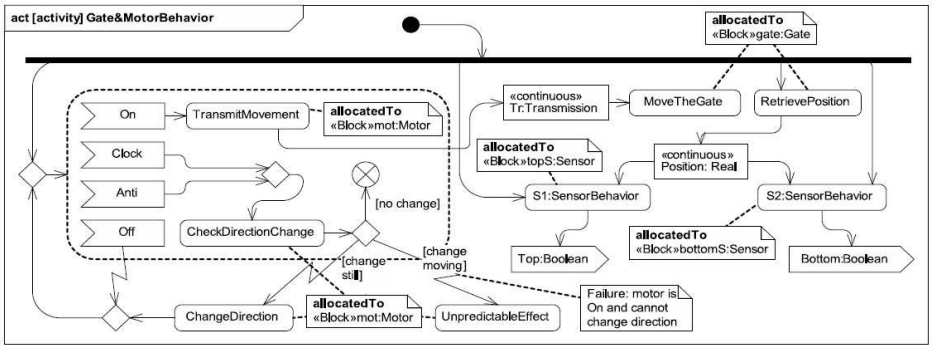
Gate&Motor domains with SysML



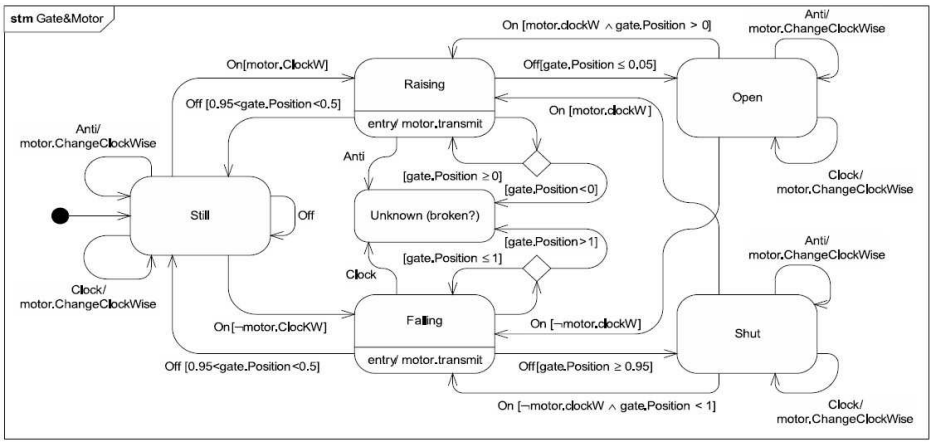
Blocks can be hierarchically decomposed in order to show details



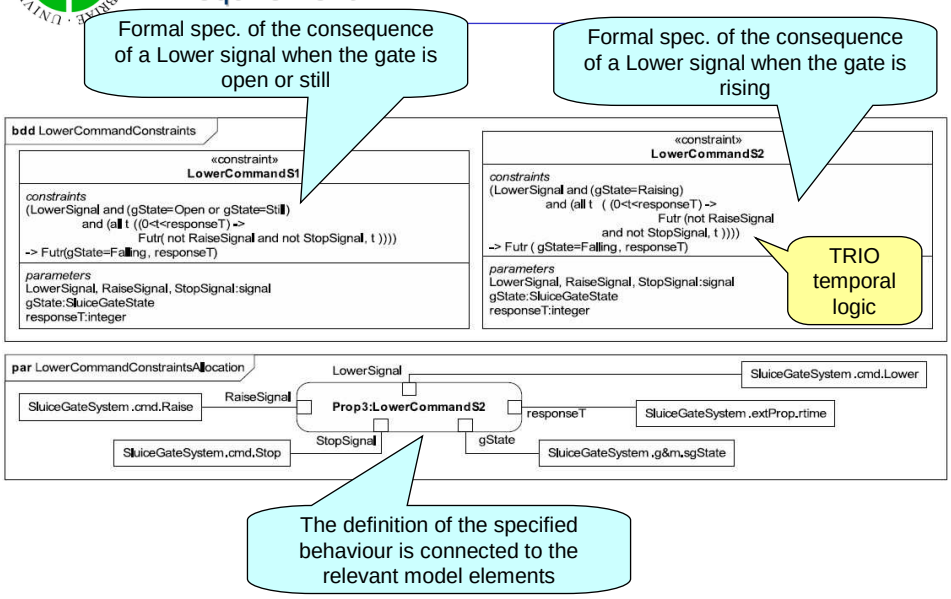
The activities associated with the components of the problem domain



The dynamics of the Gate and Motor



Formal specification of the Lower Command requirement



Contents

- The problem of requirements modelling
- Limits of the use cases
- Problem frames
- Problem frames with UML
- Dealing with time: extending OCL
- Possible evolutions of the proposed method
- Conclusions

Conclusions - summary

- The seminar presented UML-oriented ways of representing problems frames, so that PF-based requirements engineering practices can be effectively integrated into the UML development process.
- We showed that problem frames can be actually described by means of UML diagrams complemented with declarative specifications exploiting the OTL language.

Conclusions - summary

- The UML-based notation seems to be quite expressive, and to enable a natural and readable style.
  - For instance, in the sluice control system it is quite natural to represent separately the motor and the gate, to describe the motor in terms of a class with its own properties (attributes and methods), and to map methods onto interface operations, thus contributing to explain the structure and behavior of the controlled domain.
- The modelling activities carried out with the technique that integrates scenarios and problem frames concerned domain modelling, requirements modelling, sub-problem composition, dealing with the frame concern, and even sketching a simple design schema.
  - Scenarios were used in all these cases, and they proved expressive enough and quite readable.

Conclusions – expected advantages

- Using the problem frames concepts in a UML-based development is expected to provide two major benefits:
  - problem frames usage and representation is made more intuitive, thus making them more appealing to professional software developers;
  - UML-based development should greatly benefit from the rigorous concepts embedded in the problem frames approach.
- This applies to the integration of scenario-based representation as well.
- Although problem frames can be used in any development process, in practice they are not widely used: the popularity of UML is expected to maximise the target audience for the proposed approach. We expect that the usage of problem frames could be enhanced via the integration in a process employing the UML notation.

Conclusions – expected advantages

- The UML-based notation favors traceability.
- With our approach the notation used to describe the problem domain and the requirements is the same used to describe the machine specifications and the design.
- This homogeneity makes it easier to establish/recognize dependency relations, since most relations link elements of the same nature (components, classes, attributes, states, etc.) in requirements, specifications and design.
- Several tools for requirements management can import UML models, thus permitting to establish and maintain traceability relationships.

Conclusions – expected advantages

- Describing the requirements with UML makes it possible to define UML-based techniques that guide the transition from the requirements modeling phase to the design phase.
  - Concerning the transition from problem frames to design, see Choppy, C. and Reggio, G., “A UML-Based Method for the Commanded Behavior Frame”, *1st International Workshop on Advances and Applications of Problem Frames*, co-located with 26th ICSE, Edinburgh, May 2004

Conclusions - applicability conditions

- We found no feature of a problem domain, shared phenomenon, behavior specification, etc. that could not be expressed in the proposed UML-based notation.
  - In principle, the proposed techniques should be applicable in any context.
- When dealing with domains or requirements that have relevant time-related properties, and it is therefore necessary to specify timing properties, one can:
  - Use OLT, which unfortunately is not standard and not supported by tools.
  - Use SysML instead of UML, and exploit any suitable notation embedded in SysML **`<<constraints>>`**

Conclusions – tool support

- The UML-based representation of problem frames is effectively supported by any UML compliant tool.
- Unfortunately there is no tool support for OTL

- As a research activity, we are developing a tool that allows the user to create problem frame diagrams and than convert them into the UML-base representation we saw.
  - As a next step, we are planning to develop a tool that supports full integration of UML and problem frame concepts.

References

- C. Gunter, E. Gunter, M. Jackson, P. Zave, A reference model for requirements and specifications. *IEEE Software* 3(17), 2000
- M. Jackson. *Problem Frames - analysing and structuring software development problems*. Addison-Wesley ACM Press, 2001.
- L. Lavazza and V. Del Bianco. Combining Problem Frames and UML in the description of software requirements. In *Proc. of Int. Conf. on Fundamental Approaches to Software Engineering (FASE06)*, 2006.
- V. Del Bianco and L. Lavazza, “Enhancing Problem Frames with Scenarios and Histories in UML-based software development”, *Expert Systems – The Journal of Knowledge Engineering* – Blackwell publishing, IWAAPF06 special issue, to appear.
- L. Lavazza, S. Morasca, A. Morzenti, “A Dual Language Approach to the Development of Time-Critical Systems with UML” *TACoS (International Workshop on Test and Analysis of Component Based Systems) in conjunction with ETAPS 2004*, Barcelona, March 27 - 28, 2004. Electronic Notes in Theoretical Computer Science 116 (2005), January 19, pag. 227–239.
- Documentation on UML, OCL, and SysML can be found at the OMG site (www.omg.org)