Modelling Constraints

– *by Constraining the Models*

Gerrit Renker

RGU Constraints Group

The Robert Gordon University, Aberdeen

Talk Outline

4 Blocks:

- Purpose and Concept
- UML and OCL
- Modelling Example
- Outlook and Conclusion

Part 1 – Purpose and Concept

Project Goals

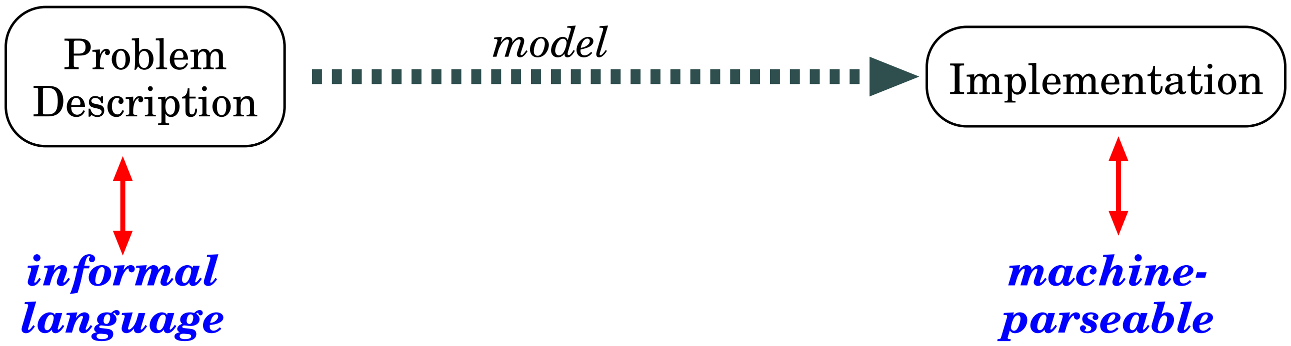
Goals:

(1) modelling support for constraint problems

(2) suitable reuse of public standards (UML/OCL, . . .)

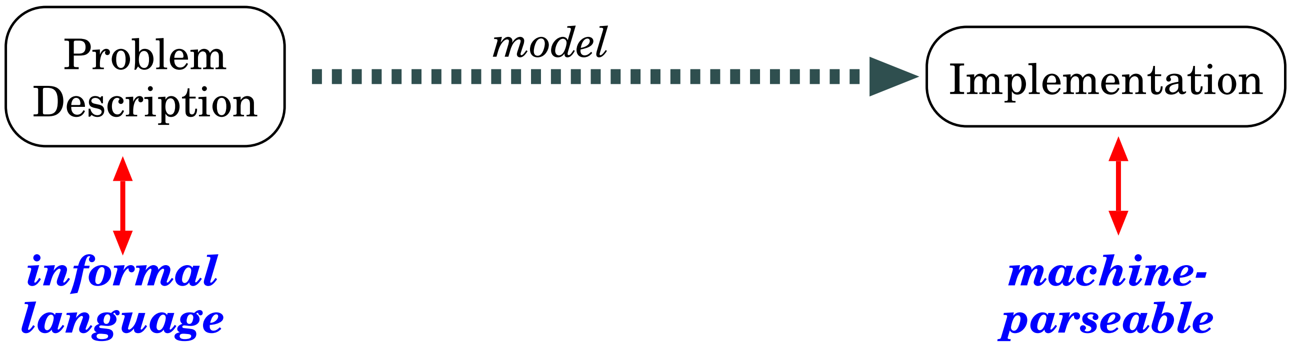
Motivation

- little modelling support for (re-) formulating CSPs
  - most problems still solved by experts
- constraint engineering



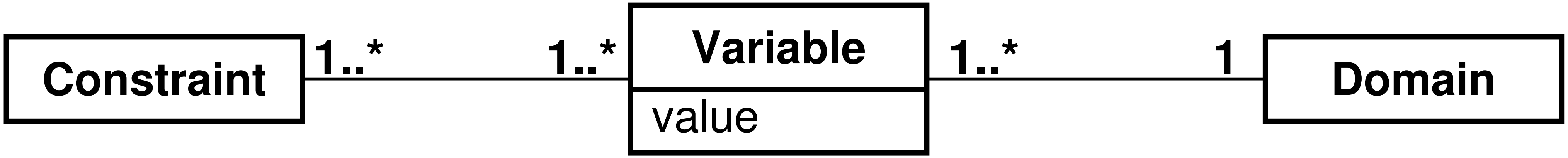
Motivation

- little modelling support for (re-) formulating CSPs
  - most problems still solved by experts
- constraint engineering



- structure information gained in analysis
  - varying levels of *abstraction*
  - different degrees of *precision*

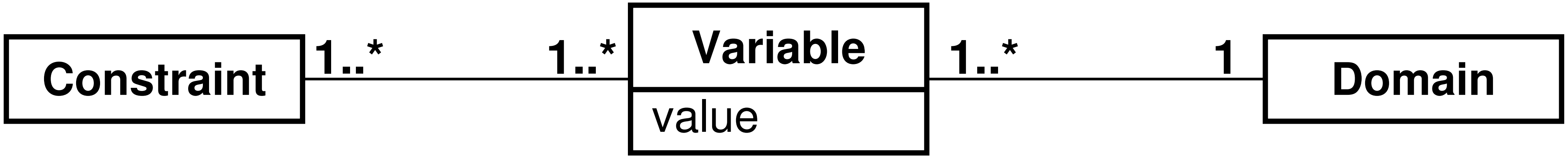
Classical CSP Definition



CSP:

- *variables* $X := \{x_1, ..., x_n\}$
- *domains* $D := \{d_1, ..., d_n\}$
- *constraints* $C \subseteq d_1 \times d_2 \times \ \ldots \ \times d_{n-1} \times d_n$

Classical CSP Definition



CSP:

- *variables* $X := \{x_1, ..., x_n\}$
- *domains* $D := \{d_1, ..., d_n\}$
- *constraints* $C \subseteq d_1 \times d_2 \times \ ... \ \times d_{n-1} \times d_n$

Example:

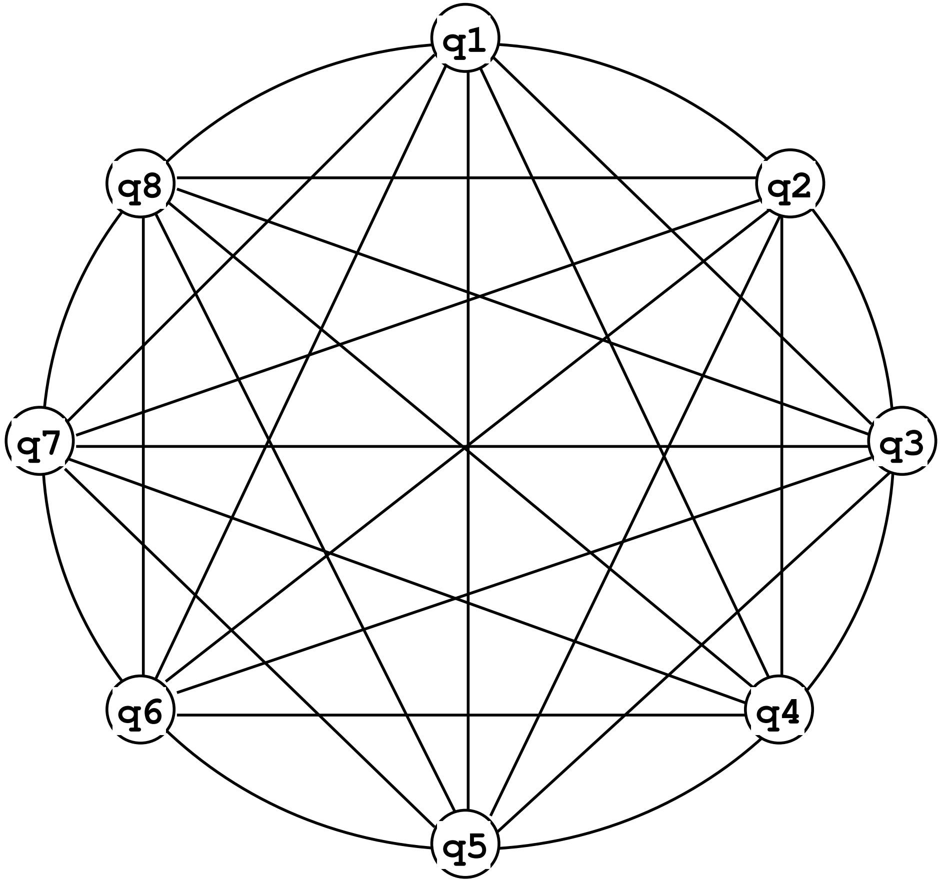
$$X = \{a, b\}; \ D = \{\mathbb{N}, \mathbb{N}\}$$
$$C \equiv a < 22 \wedge a \leq b \wedge b > 12$$

Purpose: Why use class models

- *objective:* complex data in terse format

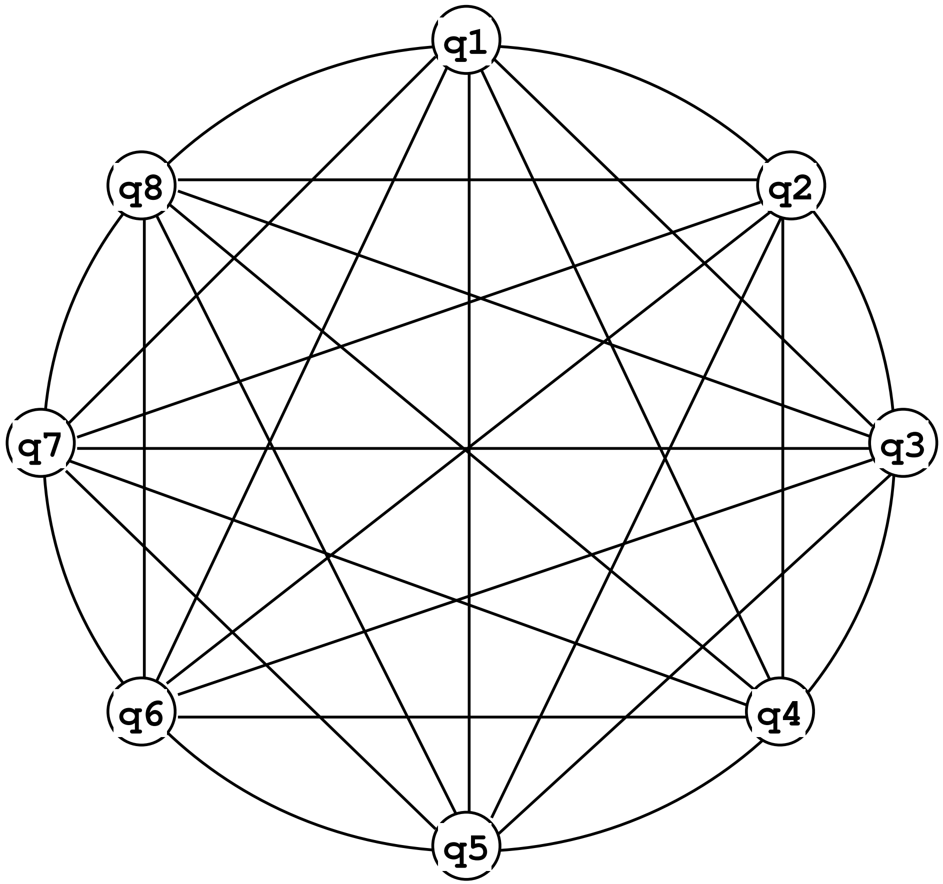
Purpose: Why use class models

- *objective:* complex data in terse format
- constraint (hyper-) graph inappropriate



Purpose: Why use class models

- *objective:* complex data in terse format
- constraint (hyper-) graph inappropriate



- $\Rightarrow$ no structural abstraction

Purpose: Modelling Support

Support In Other Disciplines:

- Semantic Data Models in Database Design

Purpose: Modelling Support

Support In Other Disciplines:

- Semantic Data Models in Database Design
- Semantic Networks in Knowledge Engineering

Purpose: Modelling Support

Support In Other Disciplines:

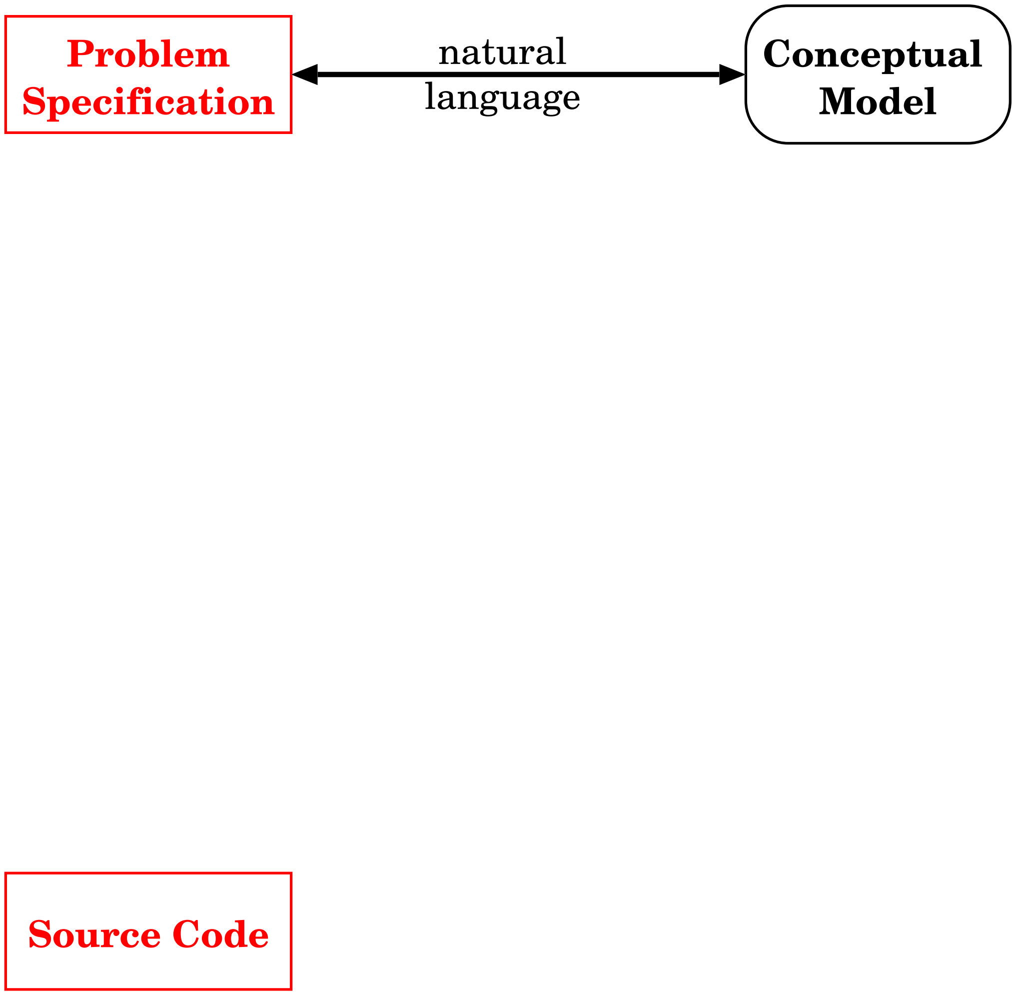
- Semantic Data Models in Database Design
- Semantic Networks in Knowledge Engineering
- Ubiquitous modelling in:
  - Software development
  - Hardware development
  - Engineering in general

Concept

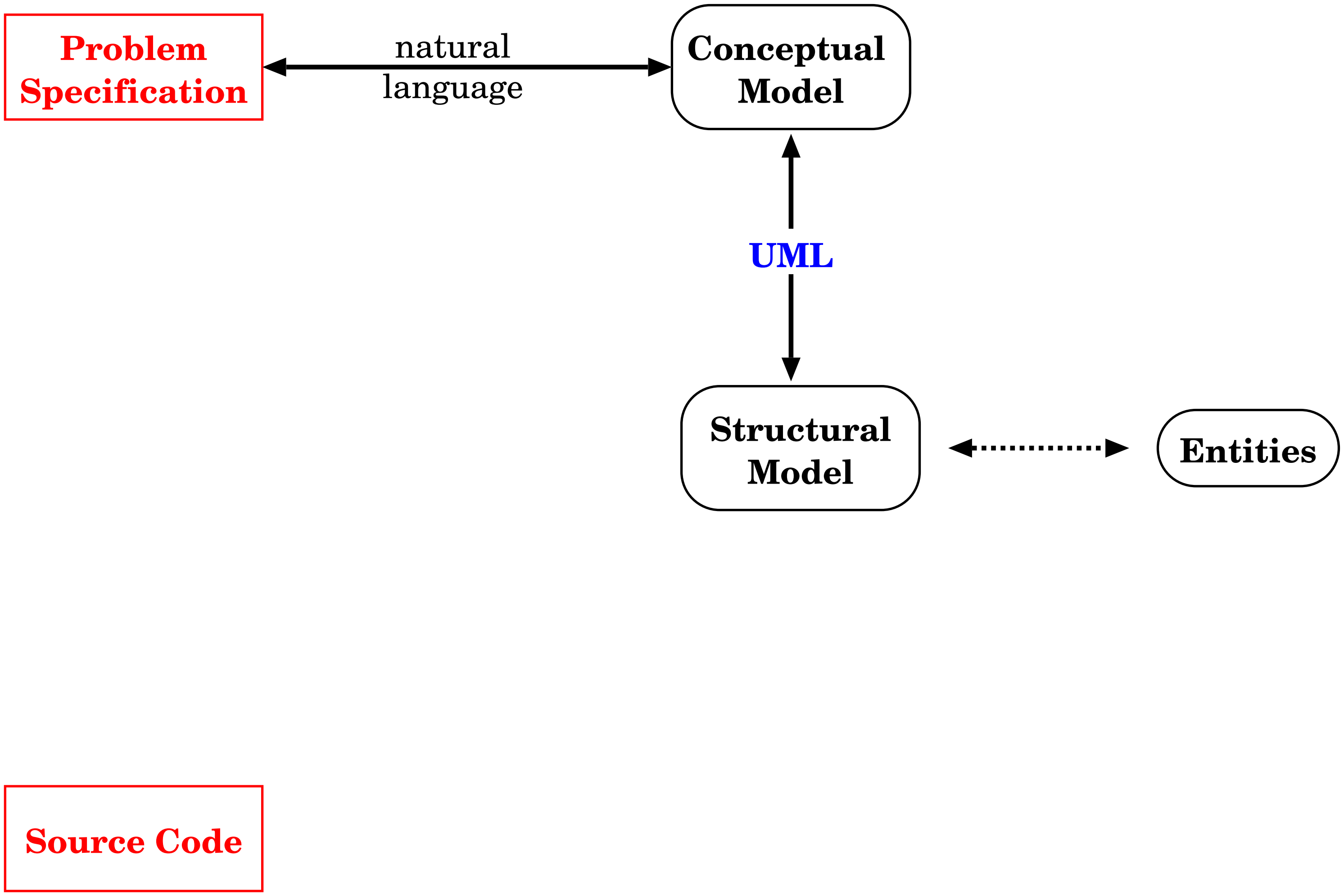
Problem Specification

Source Code

Concept

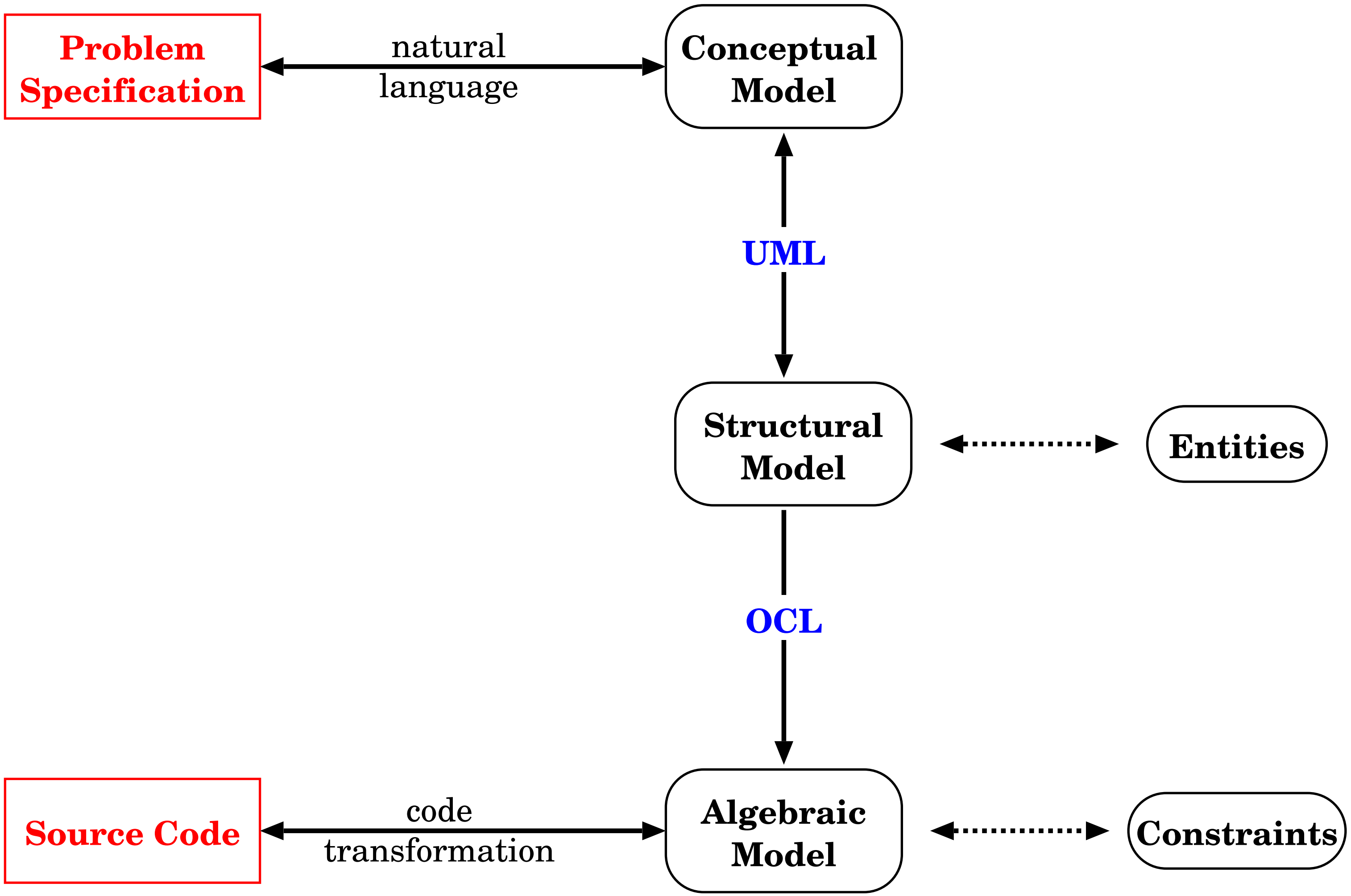

Problem Specification
natural language
Conceptual Model
Source Code

Concept



Problem Specification
natural language
Conceptual Model
UML
Structural Model
Entities
Source Code

Concept



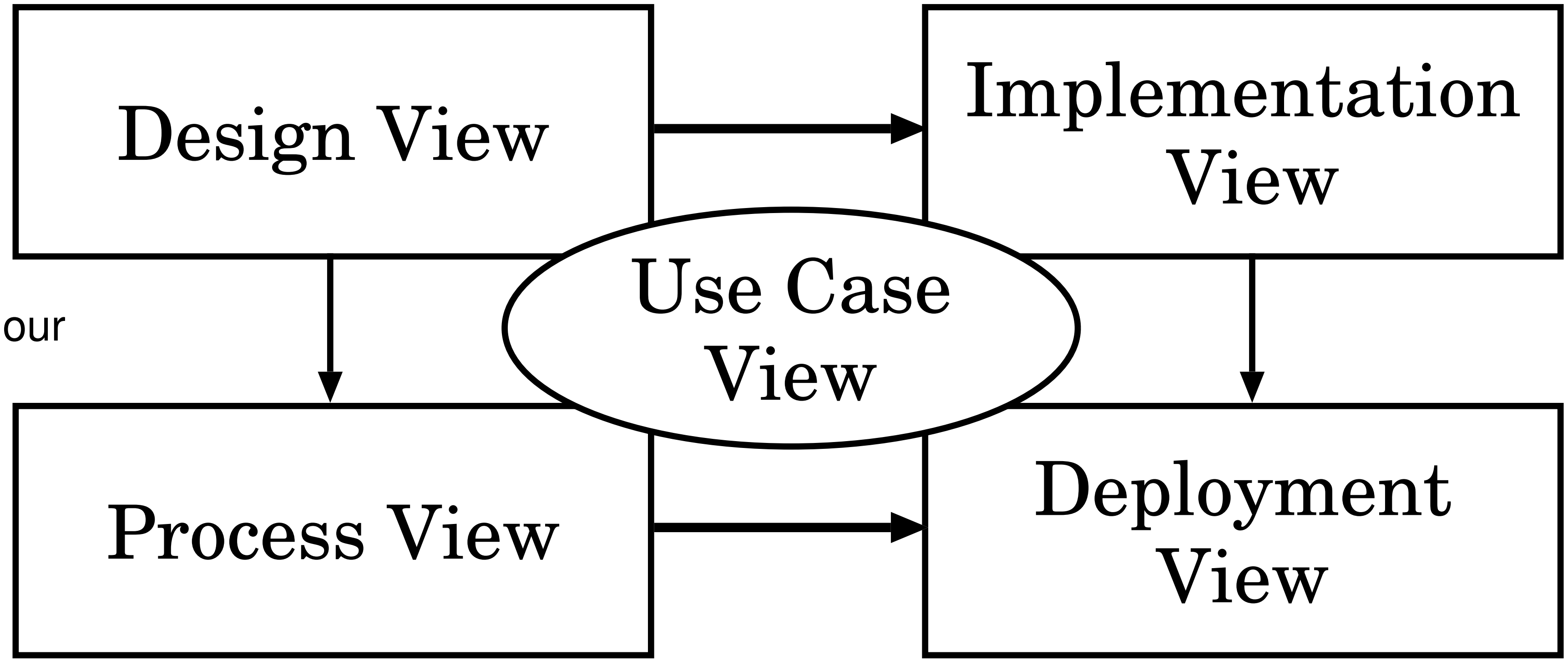
Problem Specification
natural language
Conceptual Model
UML
Structural Model
Entities
OCL
Source Code
code transformation
Algebraic Model
Constraints

Part 2a: UML

The 4+1 view of UML

Vocabulary
Functionality

System Assembly
Configuration
Management

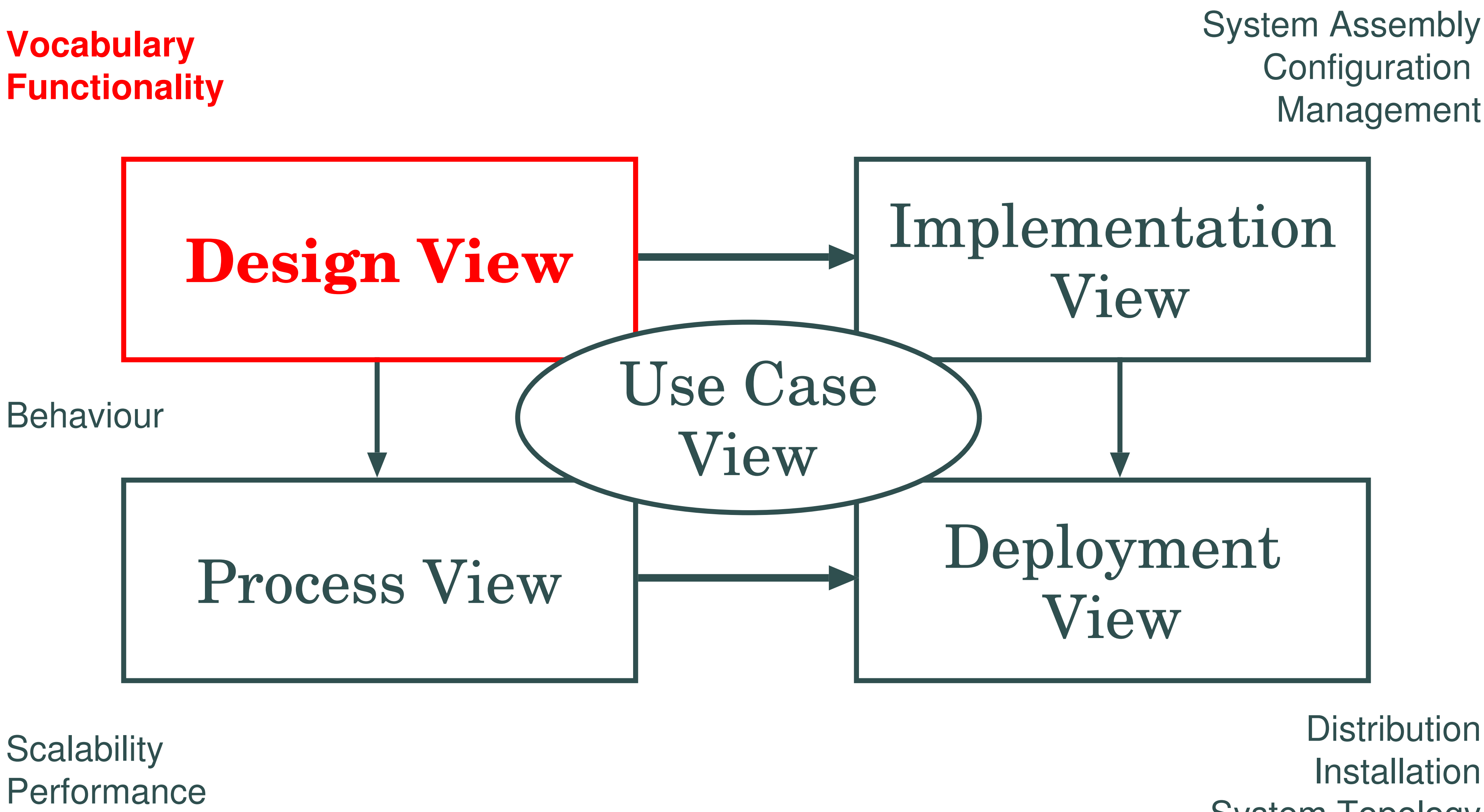


Behaviour

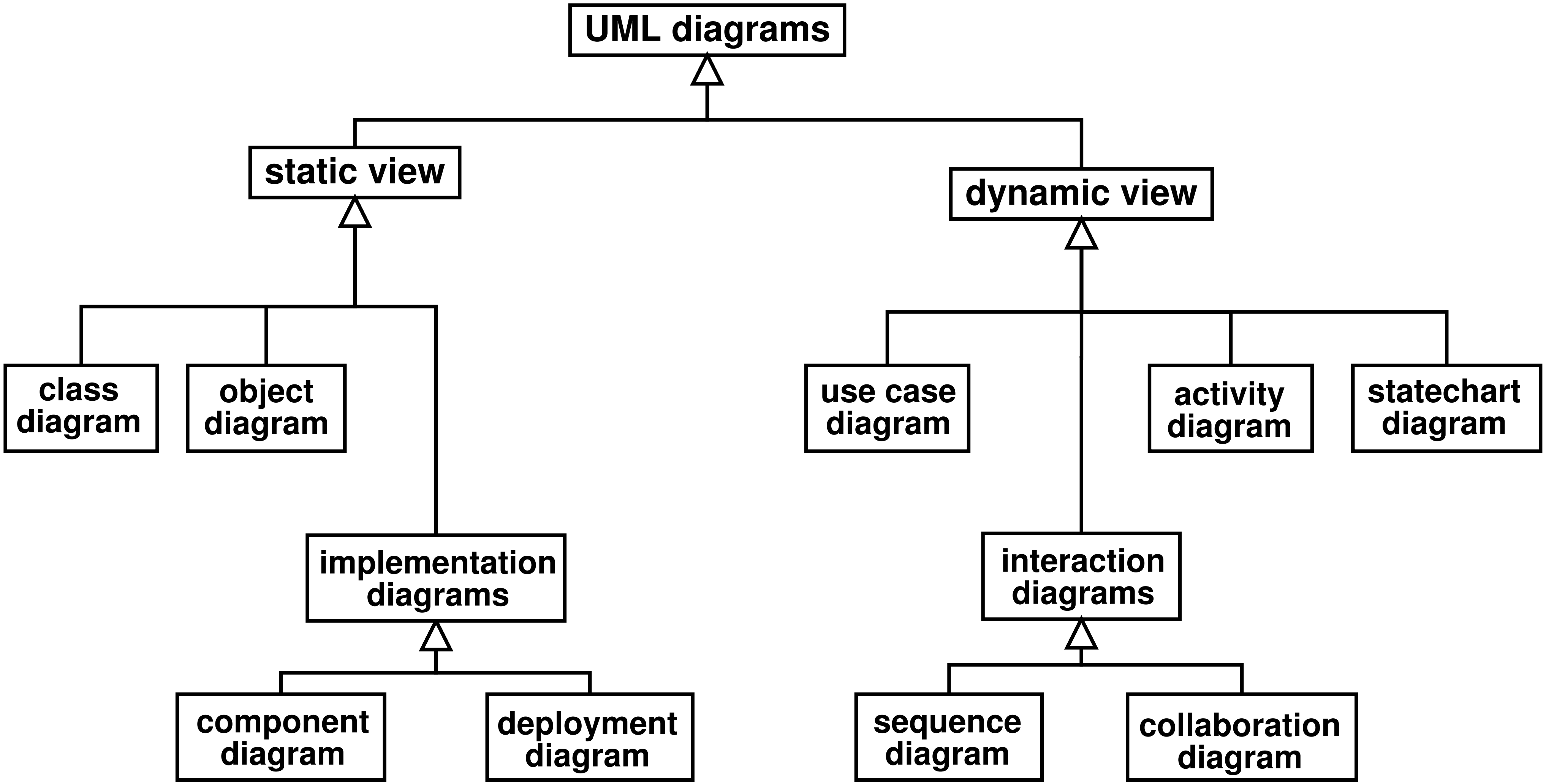
Scalability
Performance

Distribution
Installation
System Topology

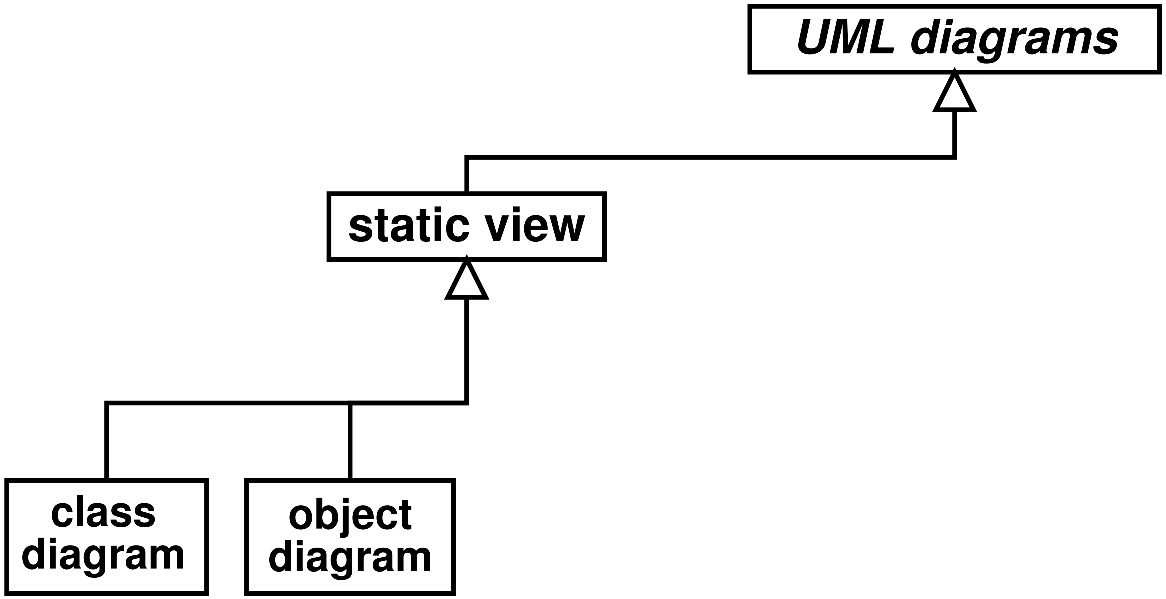
The 4+1 view of UML



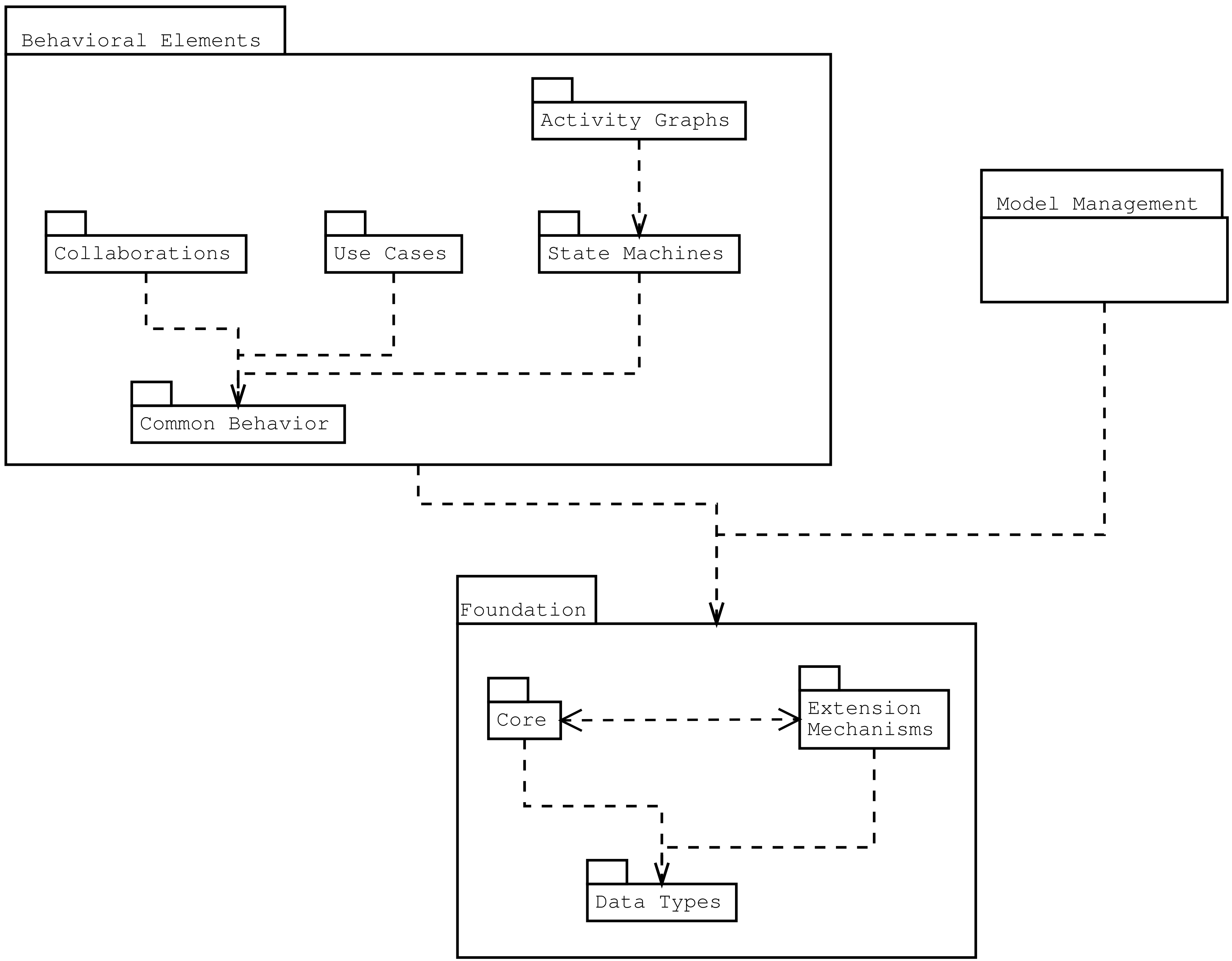
Types of UML Diagrams



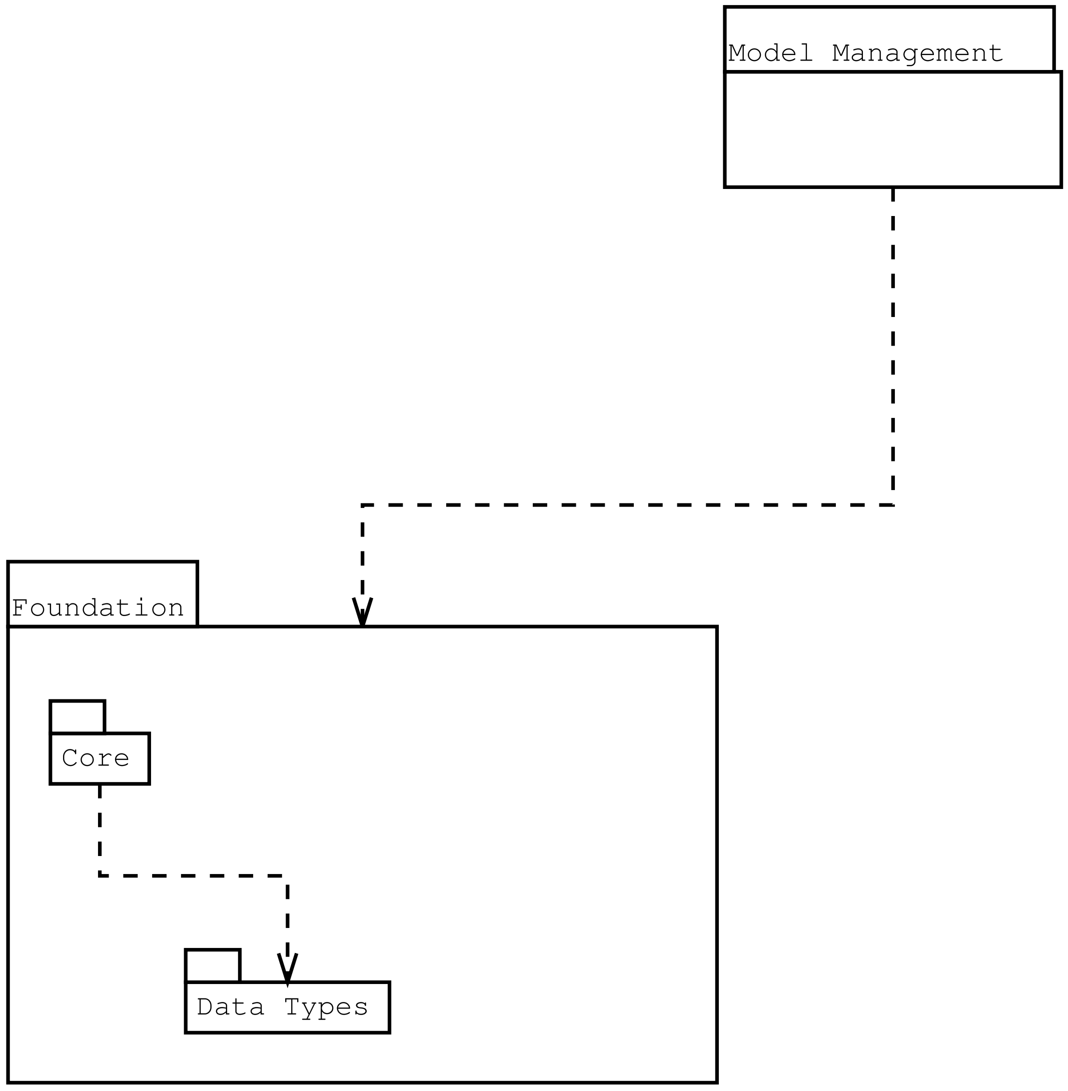
Types of UML Diagrams



UML Packaging

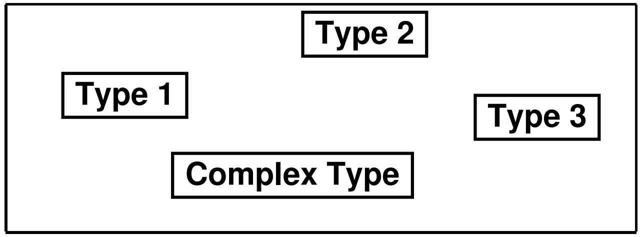


UML Packaging



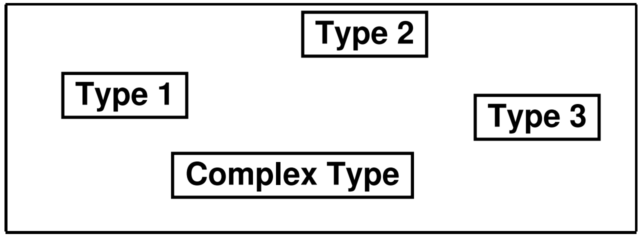
Building Blocks (UML)

- classification

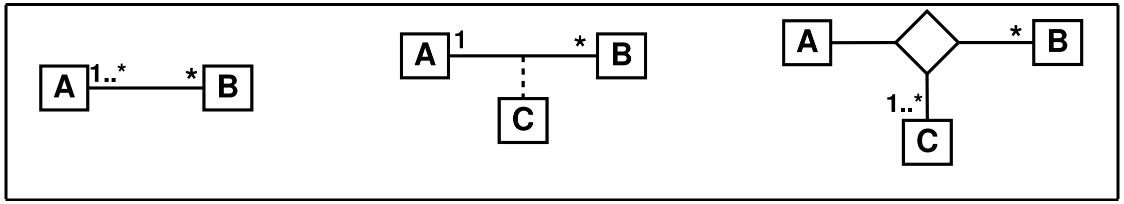


Building Blocks (UML)

- ▷ classification

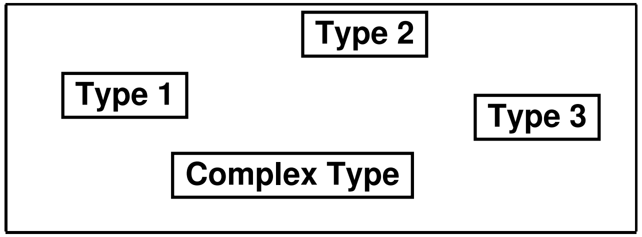


- ▷ association

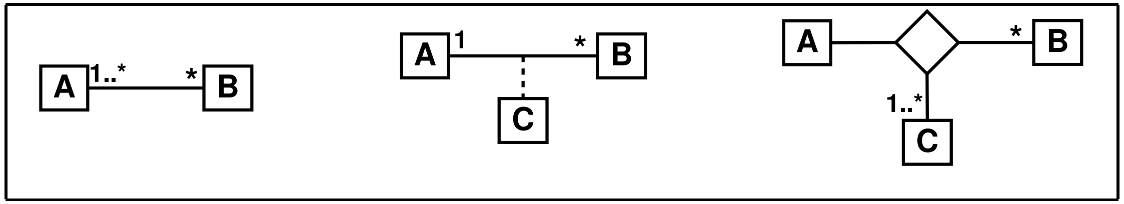


Building Blocks (UML)

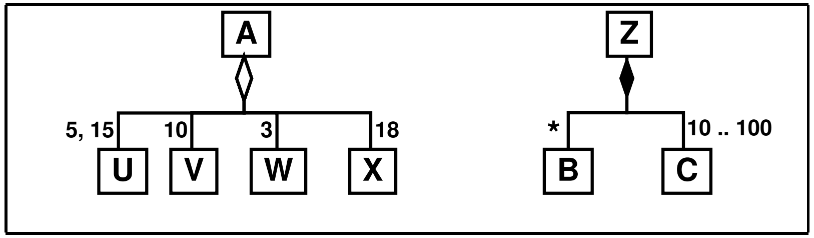
- ▷ classification



- ▷ association

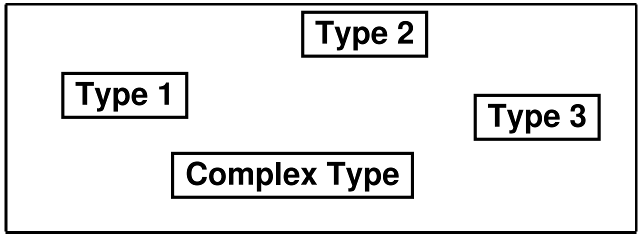


- ▷ aggregation

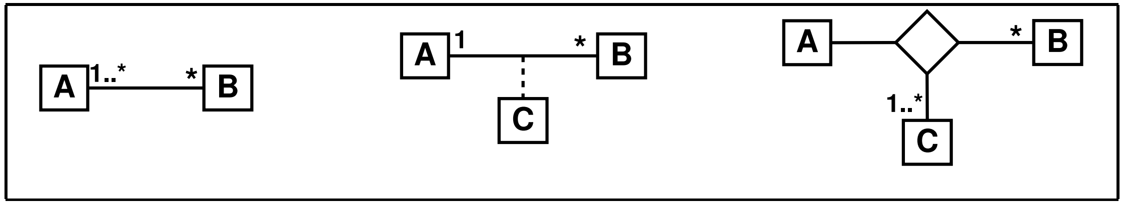


Building Blocks (UML)

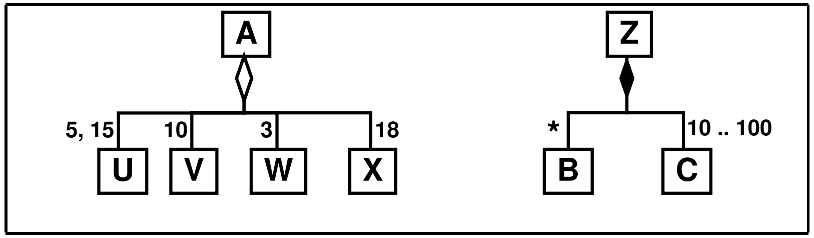
▷ classification



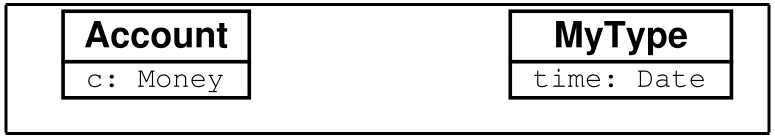
▷ association



▷ aggregation

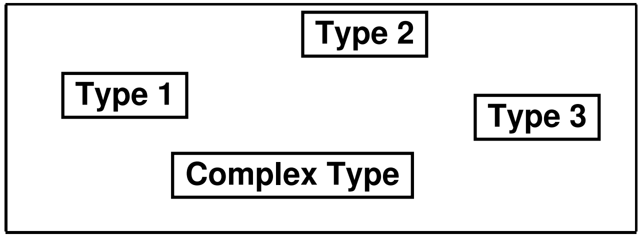


▷ attributes

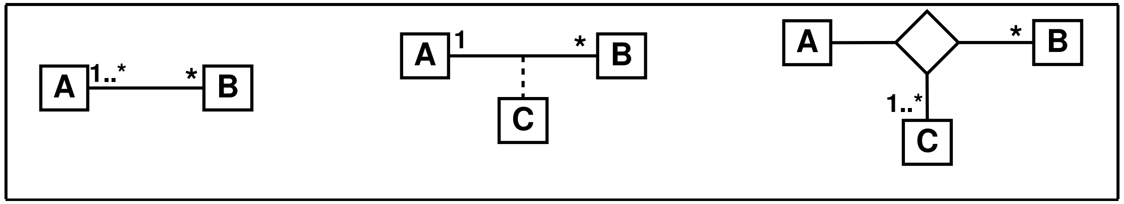


Building Blocks (UML)

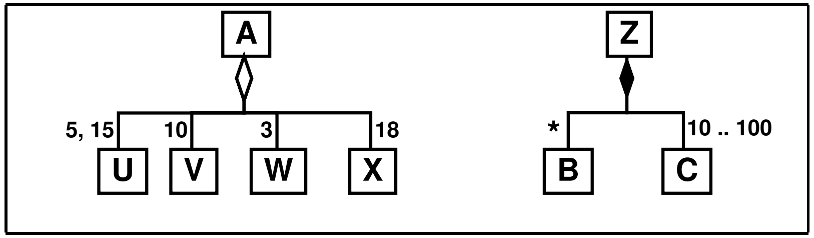
- ▷ classification



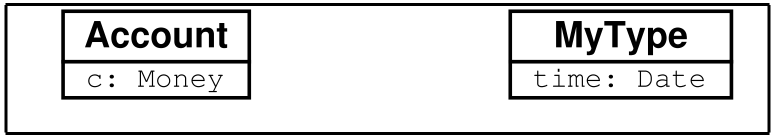
- ▷ association



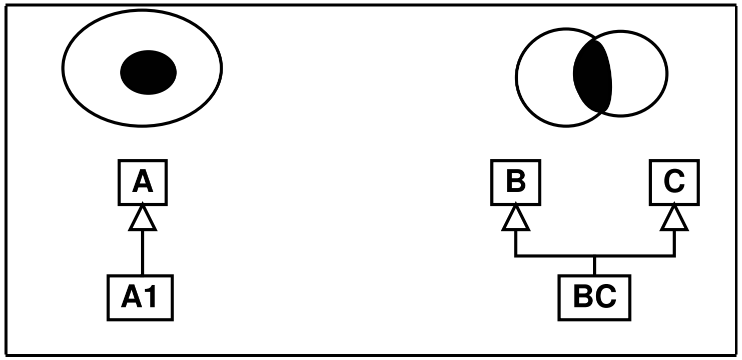
- ▷ aggregation



- ▷ attributes

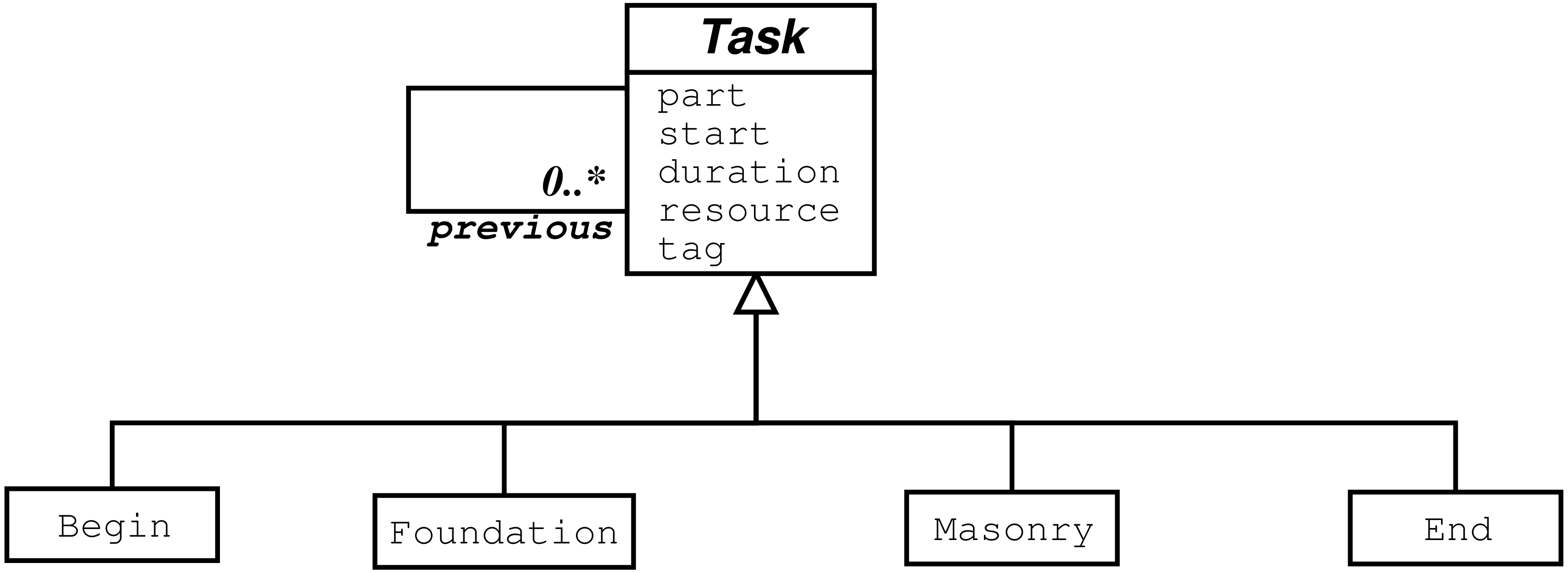


- ▷ ISA (specialization / generalization)



Example IS-A Relation

- properties explained in terms of structural differences



- inheritance of:
  - attributes
  - associations
  - constraints

Part 2b: OCL

The Object Constraint Language (OCL)

- textual complement to UML
- first order logic with navigation

The Object Constraint Language (OCL)

- textual complement to UML
- first order logic with navigation

Use of Types

- *basic types:* **`Real, Integer, Boolean`**
- *container types:* **`Collection, Set, Sequence, Bag`**
- *user-defined:* **`Tuple{}`**
  - e.g. **`Tuple{ Set(1,3,4), ``Joe Bloggs'', 1999,`**
    `Sequence(Tuple(2,1}, Tuple{2,-1}) }`

The Object Constraint Language (OCL)

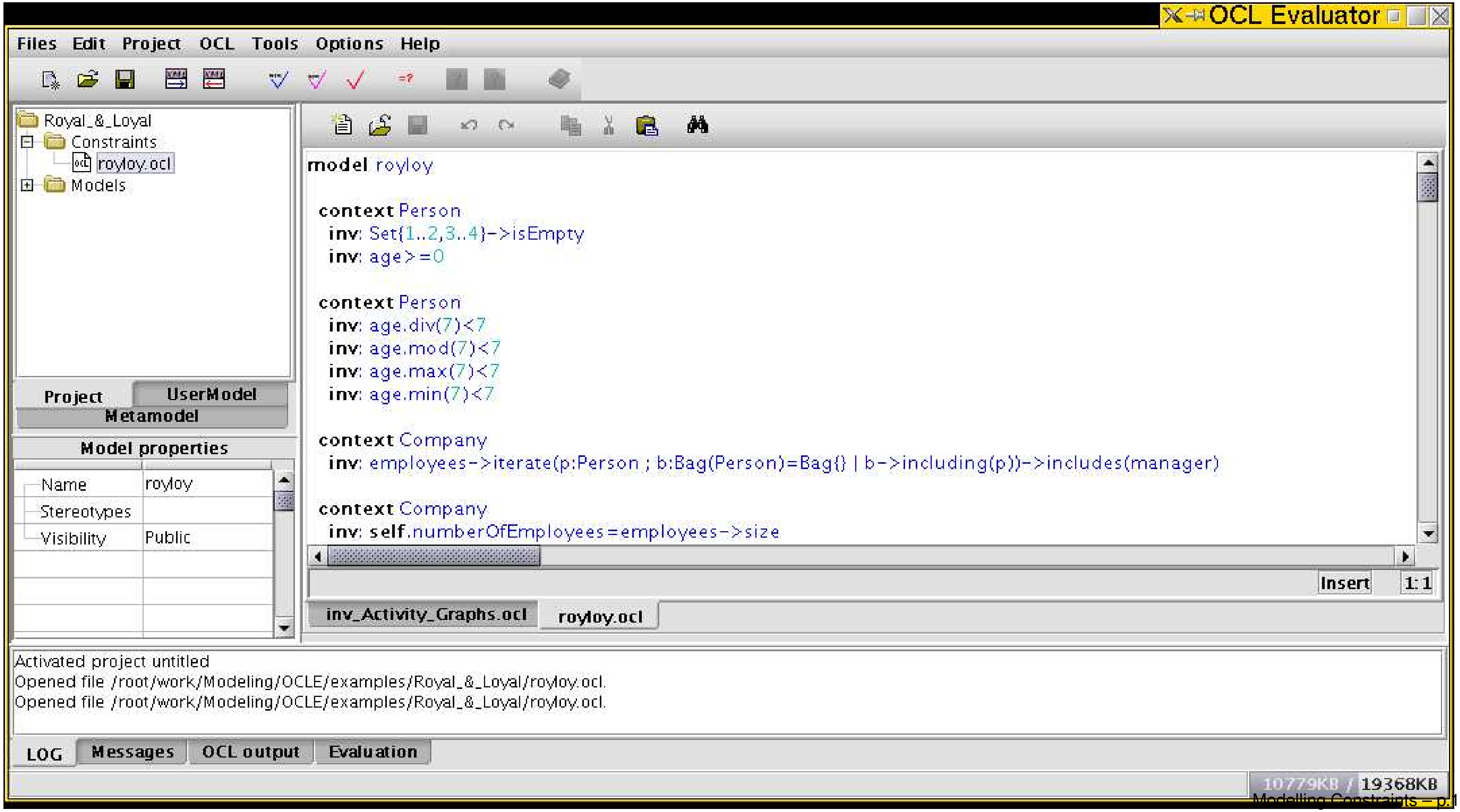
- textual complement to UML
- first order logic with navigation

Use of Types

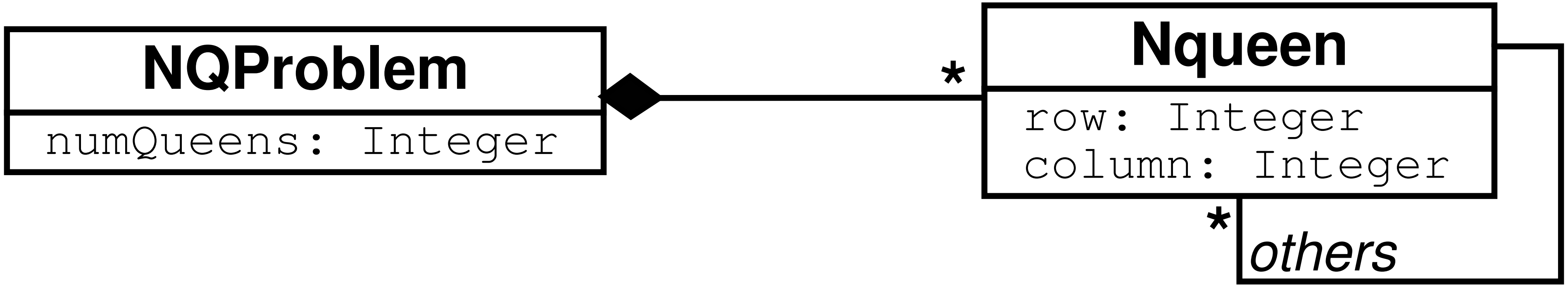
- *basic types:* `Real, Integer, Boolean`
- *container types:* `Collection, Set, Sequence, Bag`
- *user-defined:* `Tuple{}`
  - e.g. `Tuple{ Set(1,3,4), ``Joe Bloggs'', 1999,`
    `Sequence(Tuple(2,1}, Tuple{2,-1}) }`
- *UML model:* all classes accessible

Status of the OCL

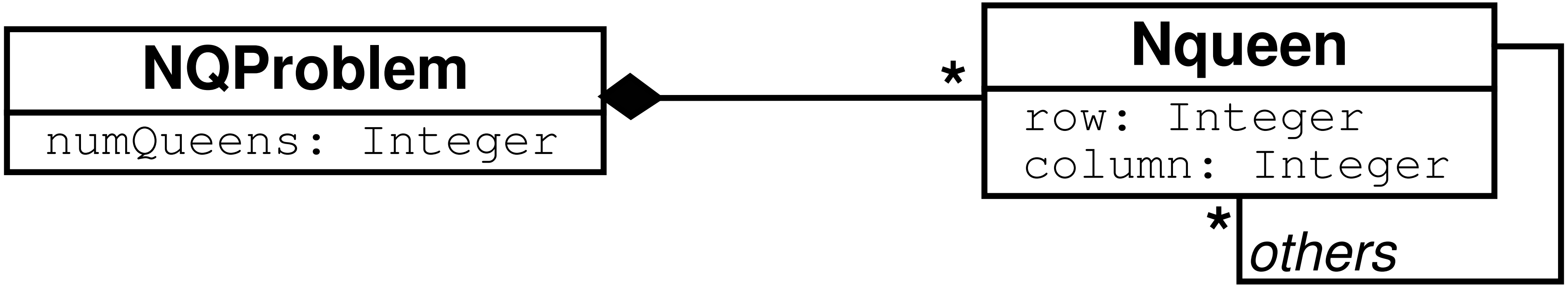
- fully integrated into UML 1.4
- several implementations (Boldsoft, Argo/UML, ...)



A Flavour of OCL: N-queens



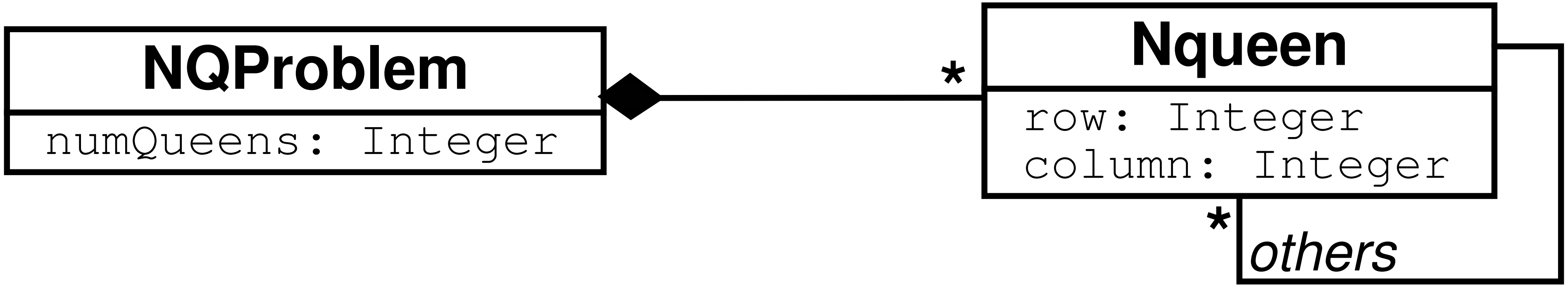
A Flavour of OCL: N-queens



- Attribute access

```
context Nqueen inv:
   column > 0 and column <= N and
   row    > 0 and row    <= N
```

A Flavour of OCL: N-queens



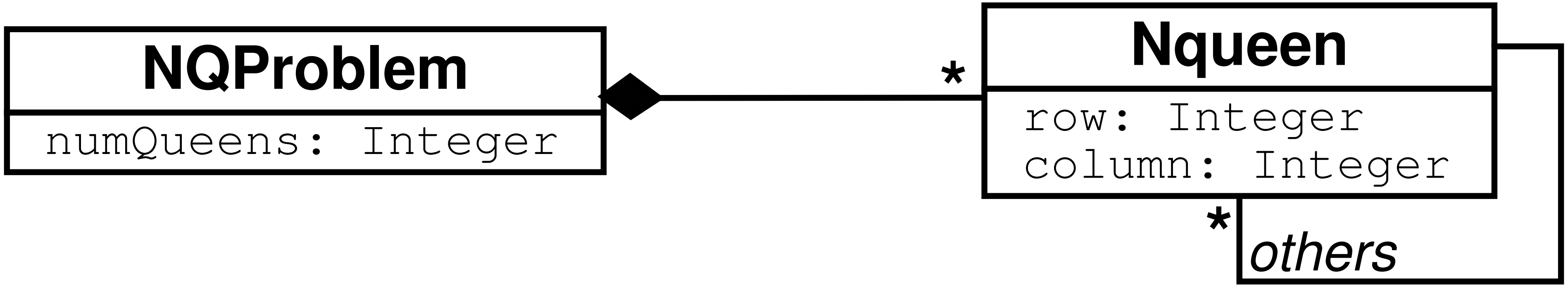
- Attribute access

```
context Nqueen inv:
   column > 0 and column <= N and
   row    > 0 and row    <= N
```

- Navigation

```
context Nqueen inv:
   nQProblem.numQueens >= 3
```

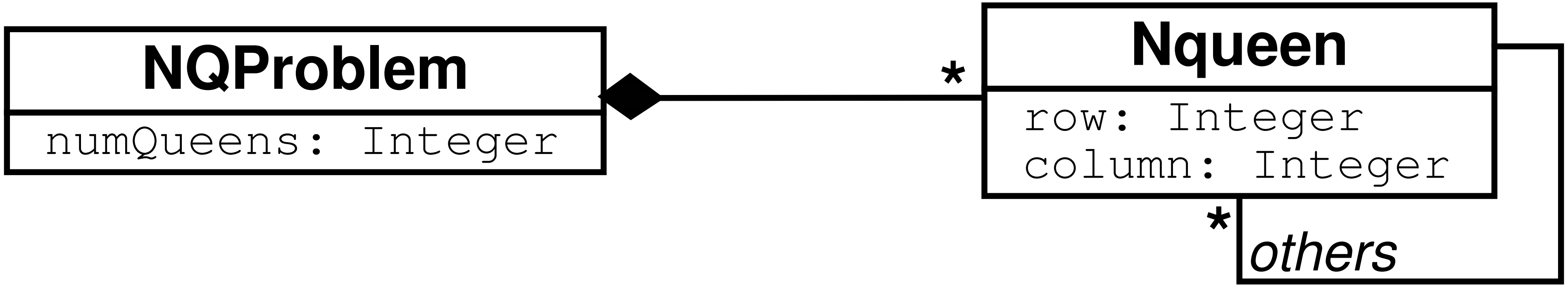
A Flavour of OCL



- Pseudo - attributes

```
--simplifies the other expressions
context Nqueen def:
attr N : Integer = nQProblem.numQueens
```

A Flavour of OCL



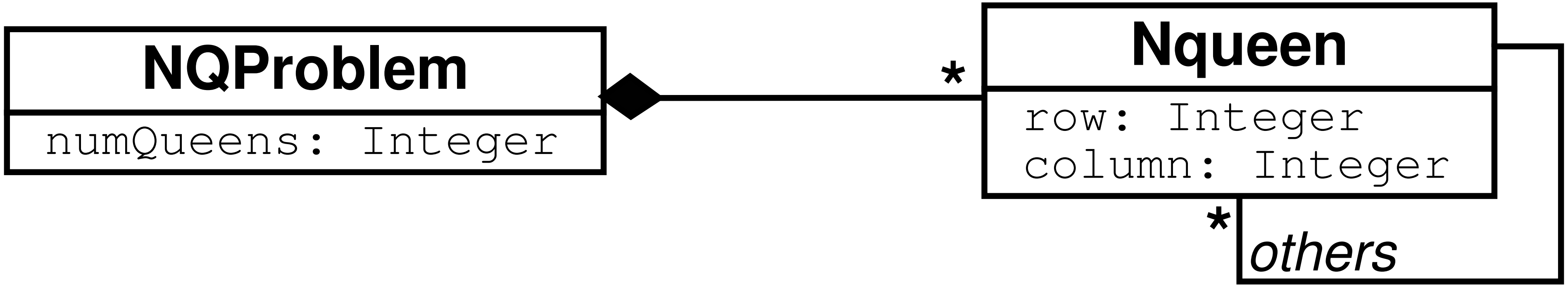
- Pseudo - attributes

```
--simplifies the other expressions
context Nqueen def:
attr N : Integer = nQProblem.numQueens
```

- Set expressions

```
context NQProblem inv:
   numQueens = nqueen->size()
```

A Flavour of OCL

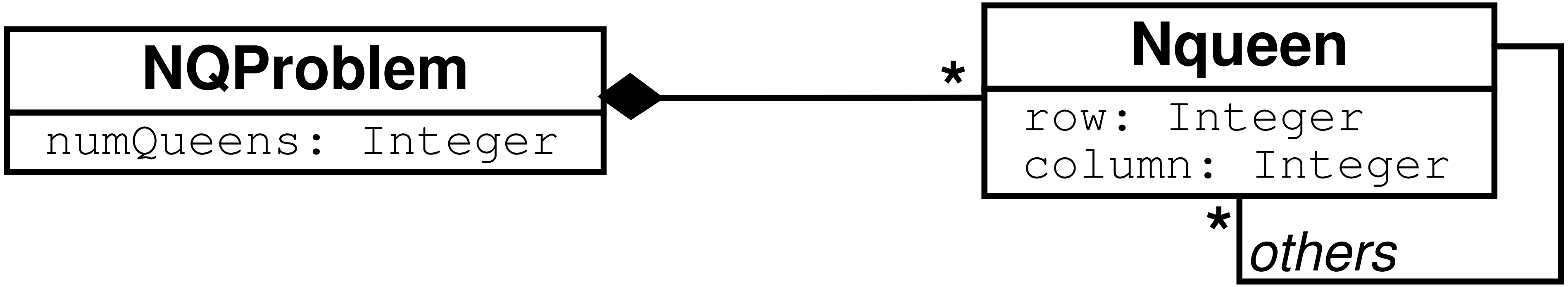


- Quantification

```
--no two queens attack another:

context NQProblem inv:
   nqueen->forAll(q1, q2 :  Nqueen |
                         not q1.attacks(q2) )
```

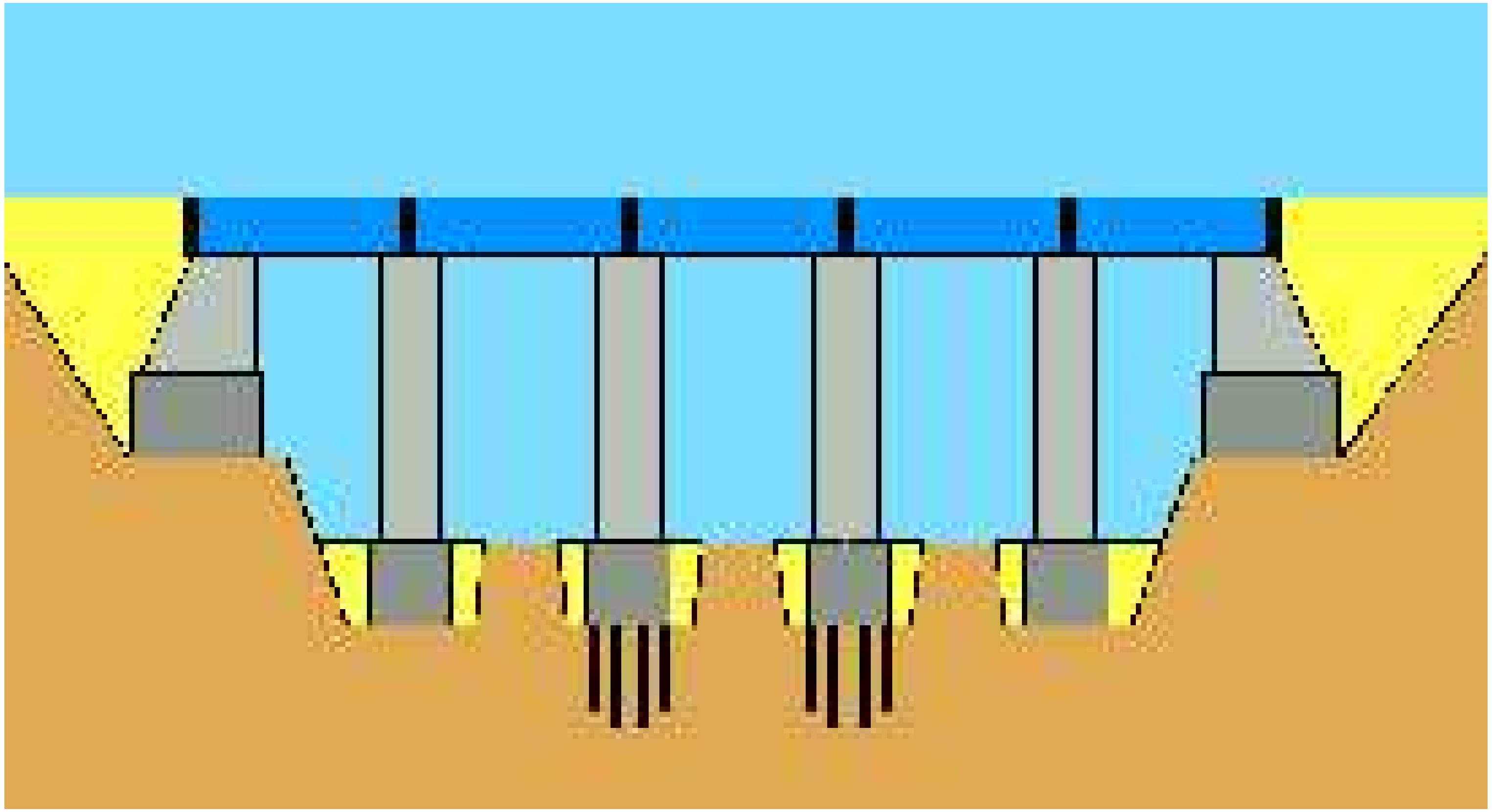
Pseudo - Operations in OCL




```
context Nqueen def:
oper attacks(other: Nqueen) : Boolean =
       self.column <> other.column and
       self.row    <> other.row
       and
       (self.row - other.row).abs <>
       (self.column - other.column).abs
```

Part 3 – Modelling Example

The Bridge - Building Problem



- schedule the setup of a 5 - segment bridge
- Martin Bartusch 1983, Massimo Paltrinieri 1994

Bridge - Building: Constraints

- 11 bridge components, 46 tasks, 7 resources

Number of Constraints:

Bridge - Building: Constraints

- 11 bridge components, 46 tasks, 7 resources

Number of Constraints:

- Resource Constraints:
  - most tasks multiply on $n = 2...6$ components
  - $\frac{n(n-1)}{2}$ constraints each $\Rightarrow$ $\boxed{\text{77 constraints}}$

Bridge - Building: Constraints

- 11 bridge components, 46 tasks, 7 resources

Number of Constraints:

- Resource Constraints:
  - most tasks multiply on $n = 2...6$ components
  - $\frac{n(n-1)}{2}$ constraints each $\Rightarrow$ $\boxed{\text{77 constraints}}$
- Precedences among tasks: $\boxed{\text{66 constraints}}$

Bridge - Building: Constraints

- 11 bridge components, 46 tasks, 7 resources

Number of Constraints:

- Resource Constraints:
  - most tasks multiply on $n = 2 ... 6$ components
  - $\frac{n(n-1)}{2}$ constraints each $\Rightarrow$ $\boxed{\text{77 constraints}}$
- Precedences among tasks: $\boxed{\text{66 constraints}}$
- Five additional requirements: $\boxed{\text{25 constraints}}$

Bridge - Building: Constraints

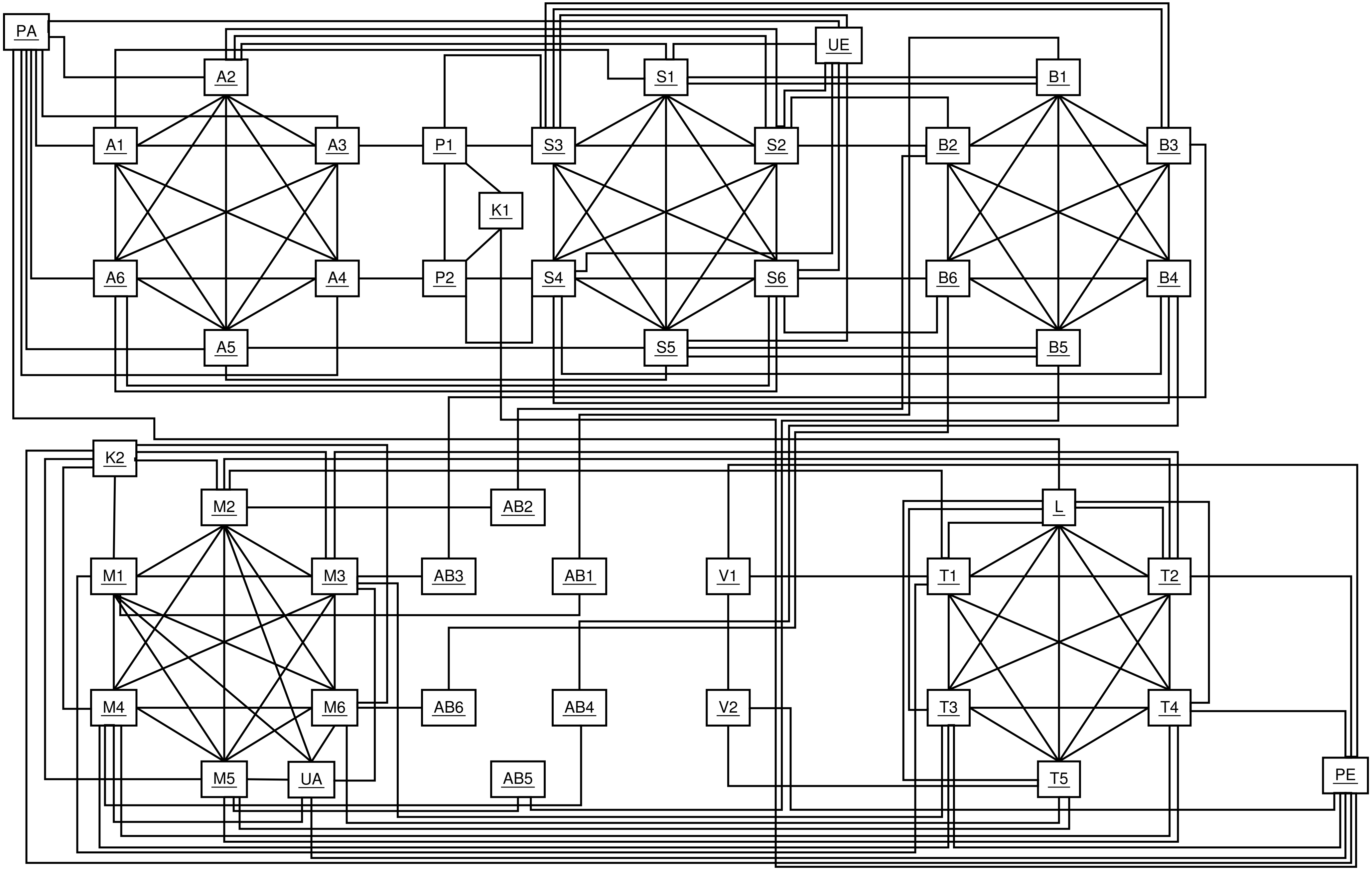
- 11 bridge components, 46 tasks, 7 resources

Number of Constraints:

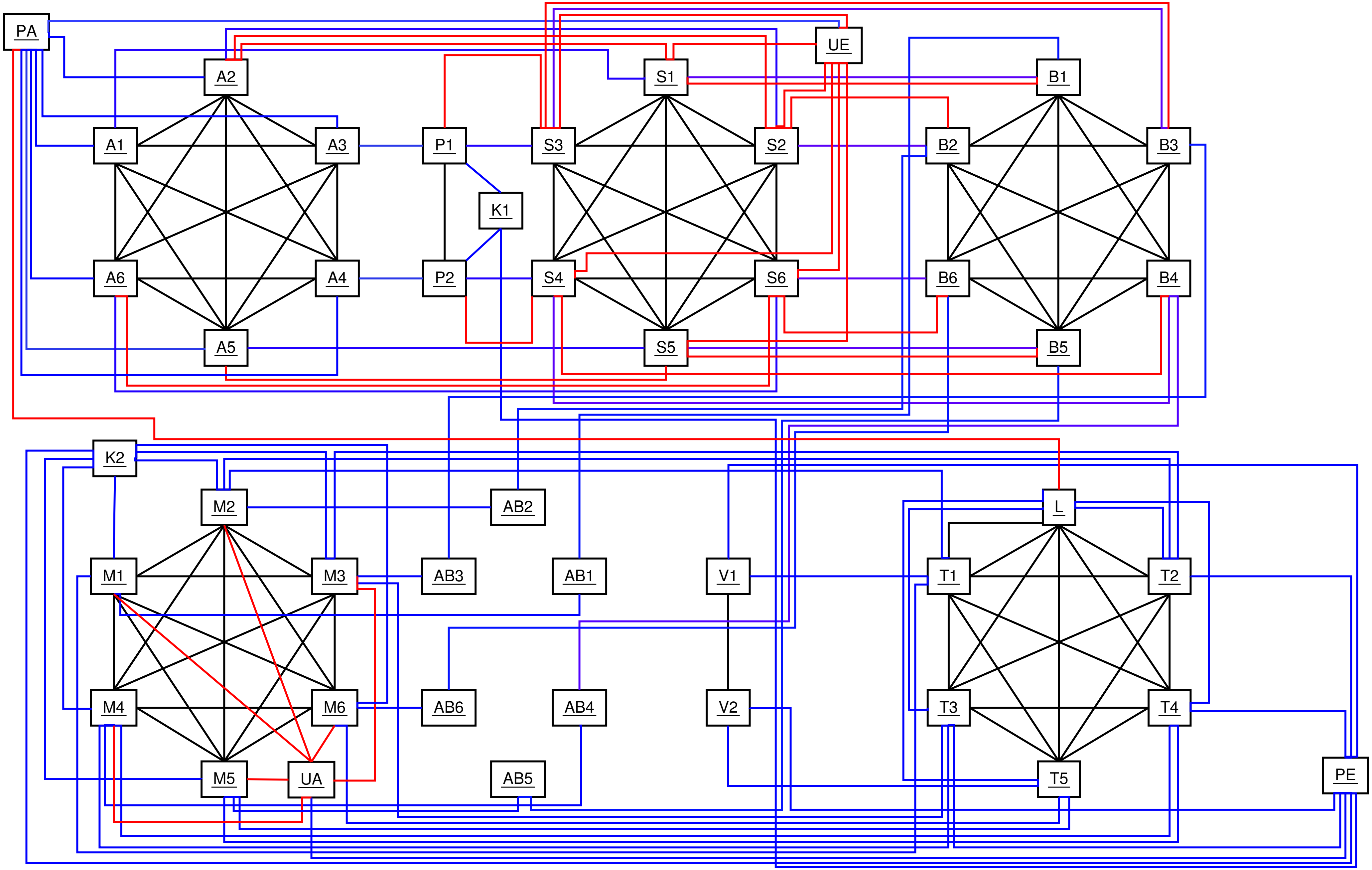
- Resource Constraints:
  - most tasks multiply on $n = 2 \ldots 6$ components
  - $\frac{n(n-1)}{2}$ constraints each $\Rightarrow$ $\boxed{\text{77 constraints}}$
- Precedences among tasks: $\boxed{\text{66 constraints}}$
- Five additional requirements: $\boxed{\text{25 constraints}}$

Sum: 168 constraints

Bridge - Building: Constraint Graph



Bridge - Building: Constraint Graph

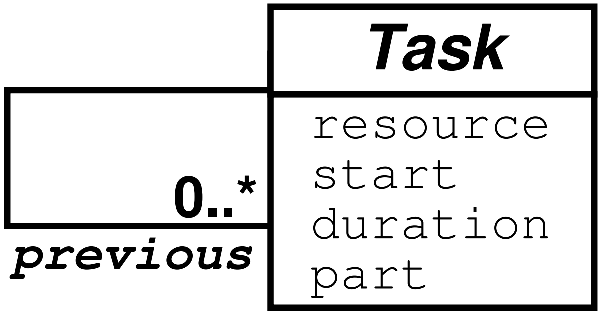


Bridge - Building: generic constraints 1/2

- Disjunction: Unary Resources

Bridge - Building: generic constraints 1/2

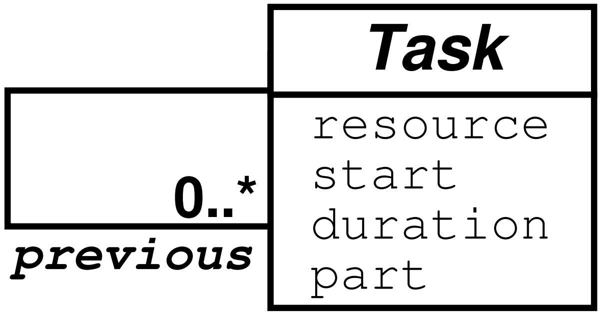
- Disjunction: Unary Resources




```
context Task inv:
  Task.allInstances->forAll(a,b:  Task|
     a <> b and
     a.name() = b.name() and
     a.resource = b.resource
   implies (
     a.start + a.duration <= b.start xor
     b.start + b.duration <= a.start )   )
```

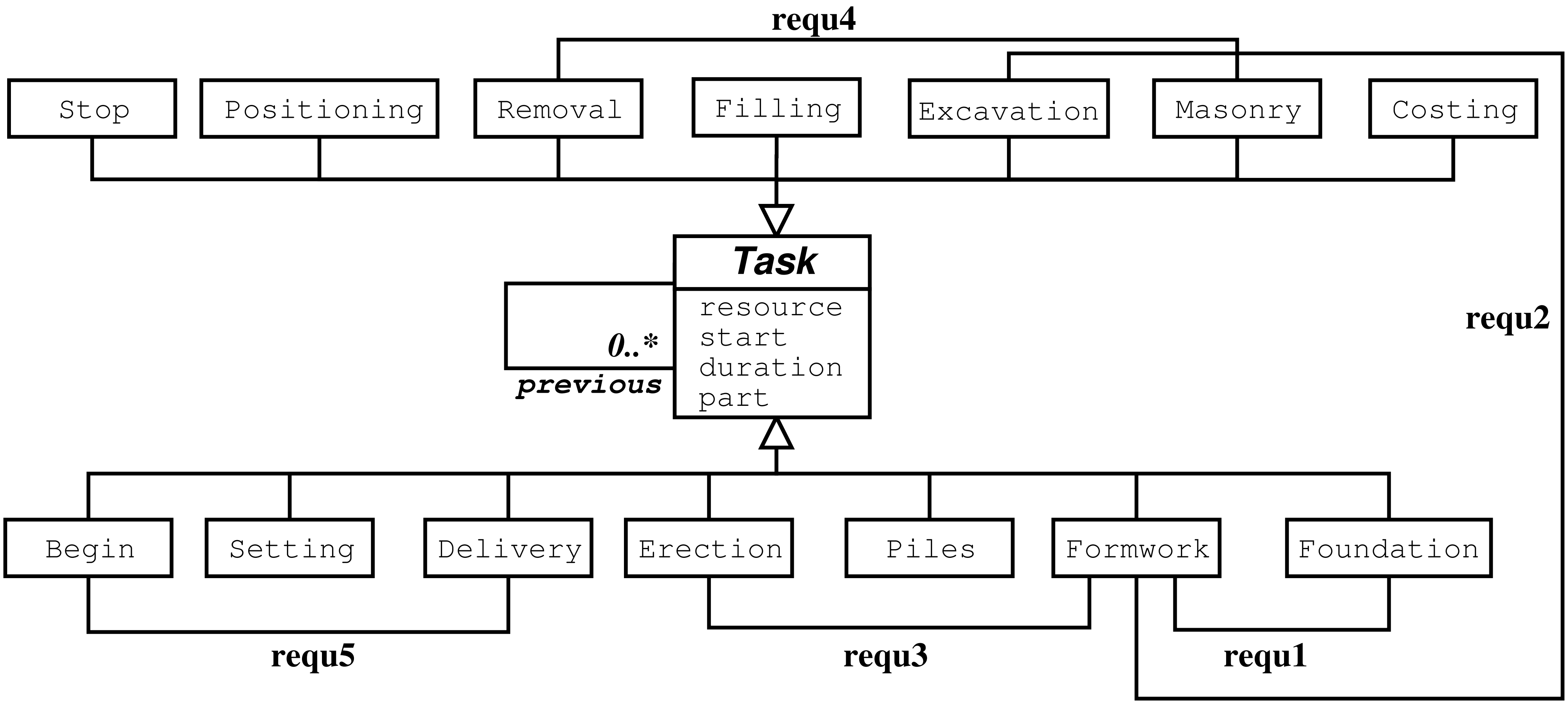
Bridge - Building: generic constraints 2/2

- Precedence Constraints



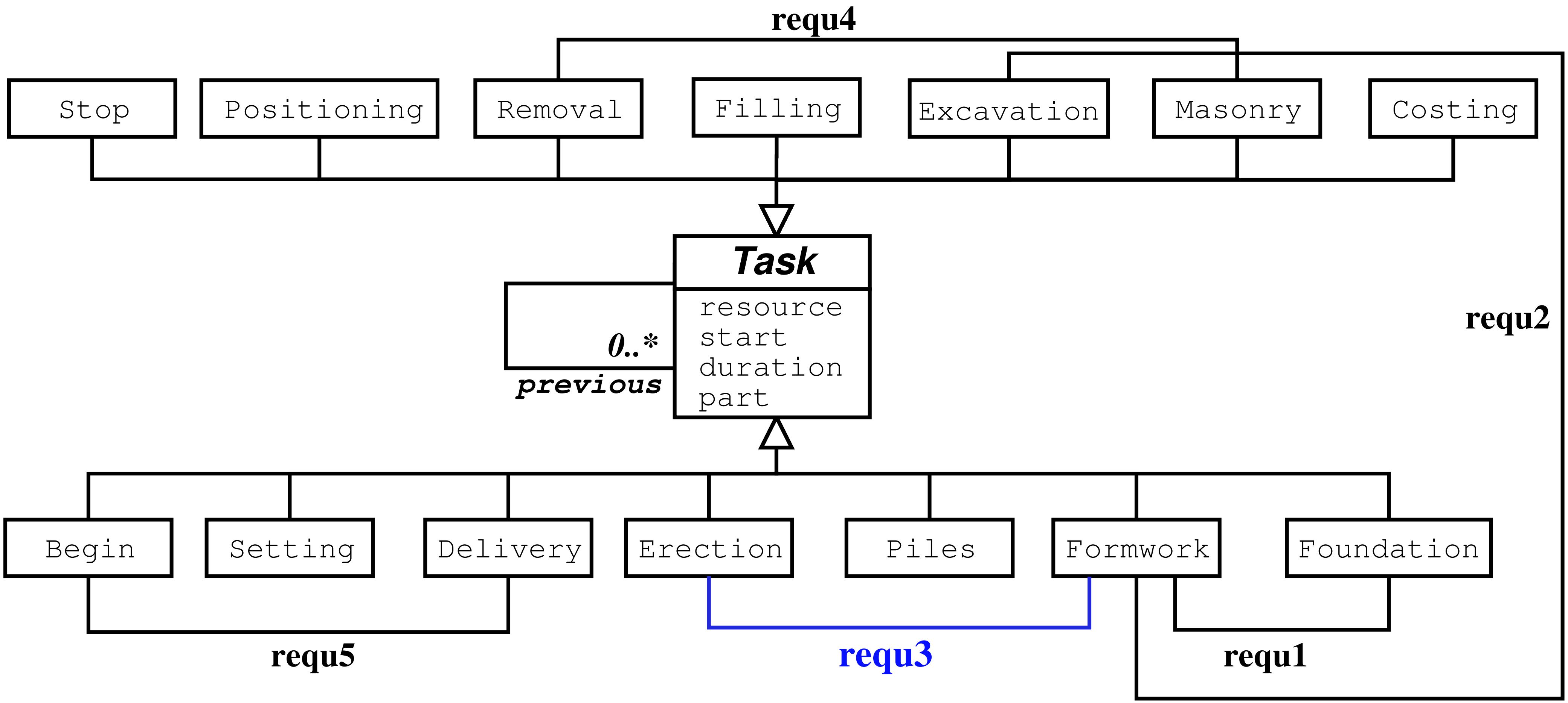

```
context Task inv:
  Task.allInstances->forAll(t:  Task|
       t.previous->forAll(prev:  Task|
         prev.start +
         prev.duration <=
         t.start )    )
```

Bridge - Building: All Tasks



- generic constraints apply to all 14 subtypes
- 2 instead of 143

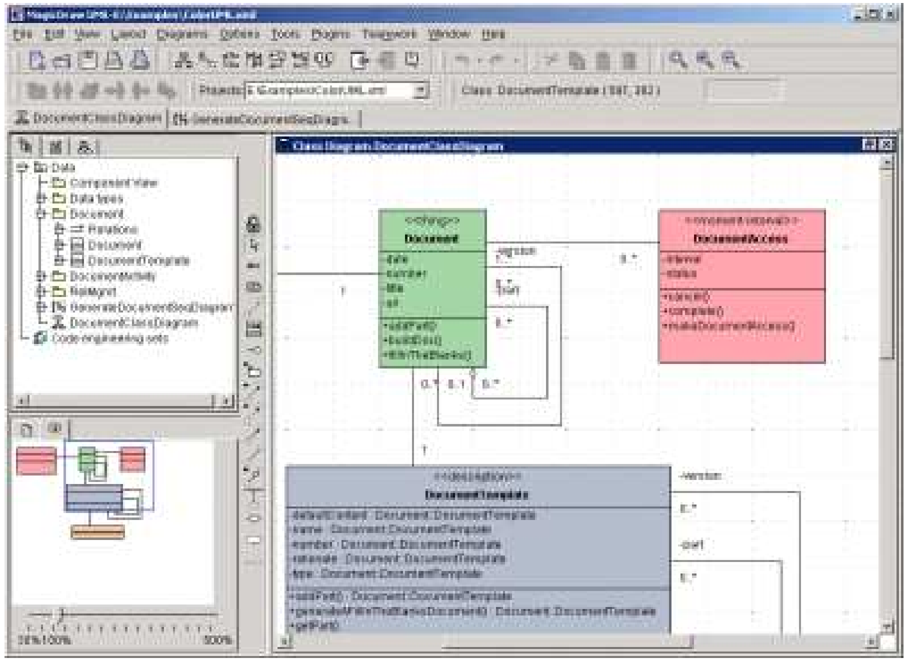
requ3: Erection of temporary housing




```
context Erection inv:
  start + 6 <= formwork.start
```

Part 4 – Outlook and Conclusion

Outlook: Application Development



```
<XMI.header>
  <XMI.documentation>
     <XMI.exporter>Together</XMI.exporter>
     <XMI.exporterVersion>4.0</XMI.exporterVersion>
  </XMI.documentation>
  <XMI.metamodel xmi.name = 'UML'
                 xmi.version = '1.1'/>
</XMI.header>
```

XML & OCL Parser

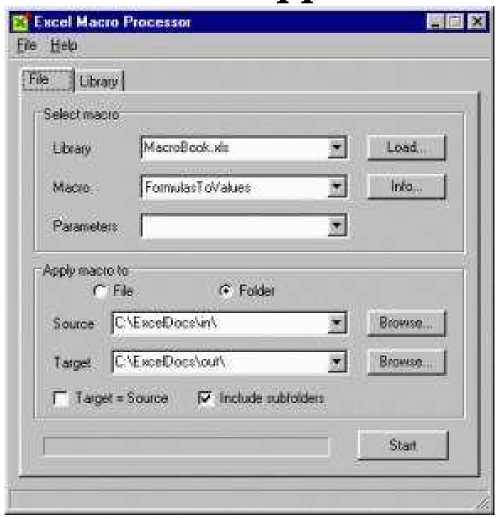
CP Code in Host Language

C++ Definitions

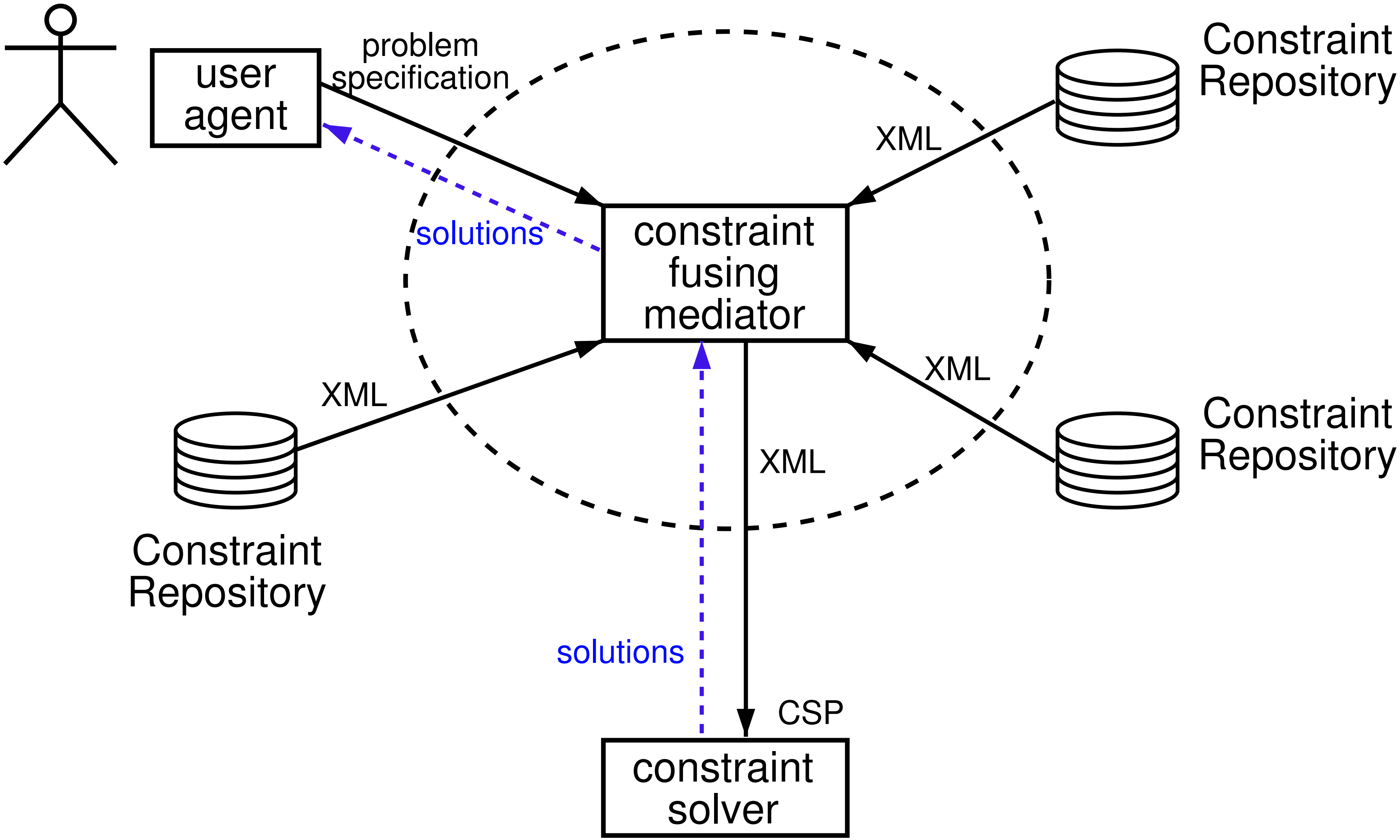
```
class Model {
  ComponentType pc = new Component;
  ...
};

class foo : public baz {
  // ...
};
```

Application



Outlook: Distributed CSPs



Benefits of using UML & OCL

- *simplified, comprehensible knowledge acquisition*
- seamless integration:
  - constraint-based reasoning
  - software development
- open to emerging developments (Semantic Web)

Benefits of using UML & OCL

- *simplified, comprehensible knowledge acquisition*
- seamless integration:
  - constraint-based reasoning
  - software development
- open to emerging developments (Semantic Web)

Experiences: UML/OCL in

- Software Architecture Description Languages (ADL)
- excellent experiences for configuration KBS
- Ontology Description
- CommonKADS, Rule-Based Systems

Problems of UML

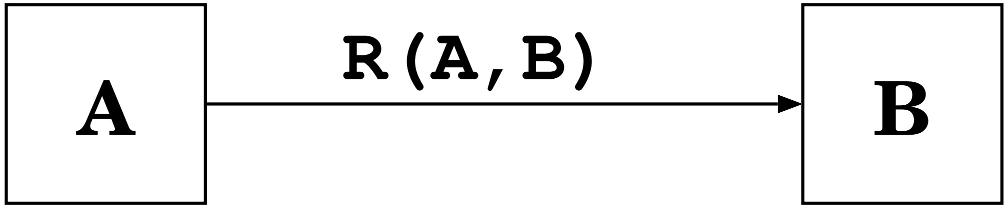
- informal notation $\leftrightarrow$ informal use
  - semantics in *'precise'* English
  - conflicting definitions, ambiguities, imprecision
- meta-circular: defines itself by itself
- commercial standard – dependent on OMG politics
  - slow adaptation (6-12 months)

Problems of UML

- informal notation $\leftrightarrow$ informal use
  - semantics in *'precise'* English
  - conflicting definitions, ambiguities, imprecision
- meta-circular: defines itself by itself
- commercial standard – dependent on OMG politics
  - slow adaptation (6-12 months)
- modelling *relations* is awkward, e.g. $R \subseteq A \times B$: $(\forall a \in A)\,(\exists b \in B)\,(a, b) \in R$

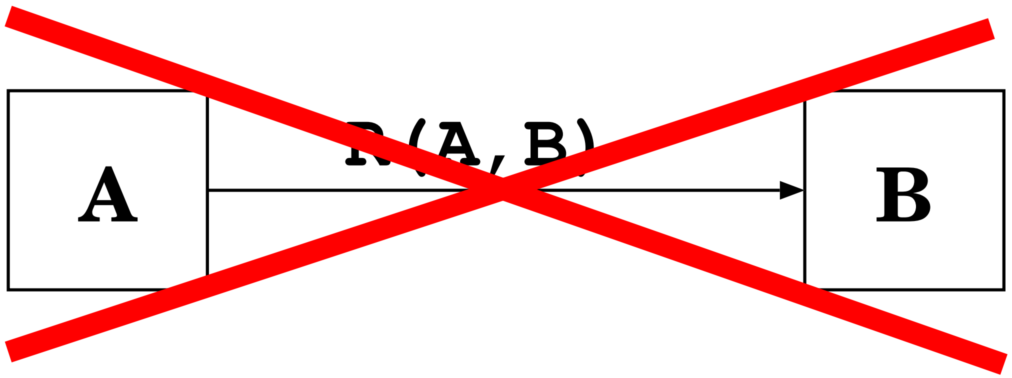
Problems of UML

- informal notation $\leftrightarrow$ informal use
  - semantics in *'precise'* English
  - conflicting definitions, ambiguities, imprecision
- meta-circular: defines itself by itself
- commercial standard – dependent on OMG politics
  - slow adaptation (6-12 months)
- modelling *relations* is awkward, e.g. $R \subseteq A \times B$: $(\forall a \in A)\,(\exists b \in B)\,(a, b) \in R$



Problems of UML

- informal notation $\leftrightarrow$ informal use
  - semantics in *'precise'* English
  - conflicting definitions, ambiguities, imprecision
- meta-circular: defines itself by itself
- commercial standard – dependent on OMG politics
  - slow adaptation (6-12 months)
- modelling *relations* is awkward, e.g. $R \subseteq A \times B$: $(\forall a \in A)\,(\exists b \in B)\,(a,b) \in R$



Conclusion

Useful for further work

- intuitively appealing notation
- informal notation as *'scratch pad'*
- abstraction / generalisation / packaging facilities
- translation to XML

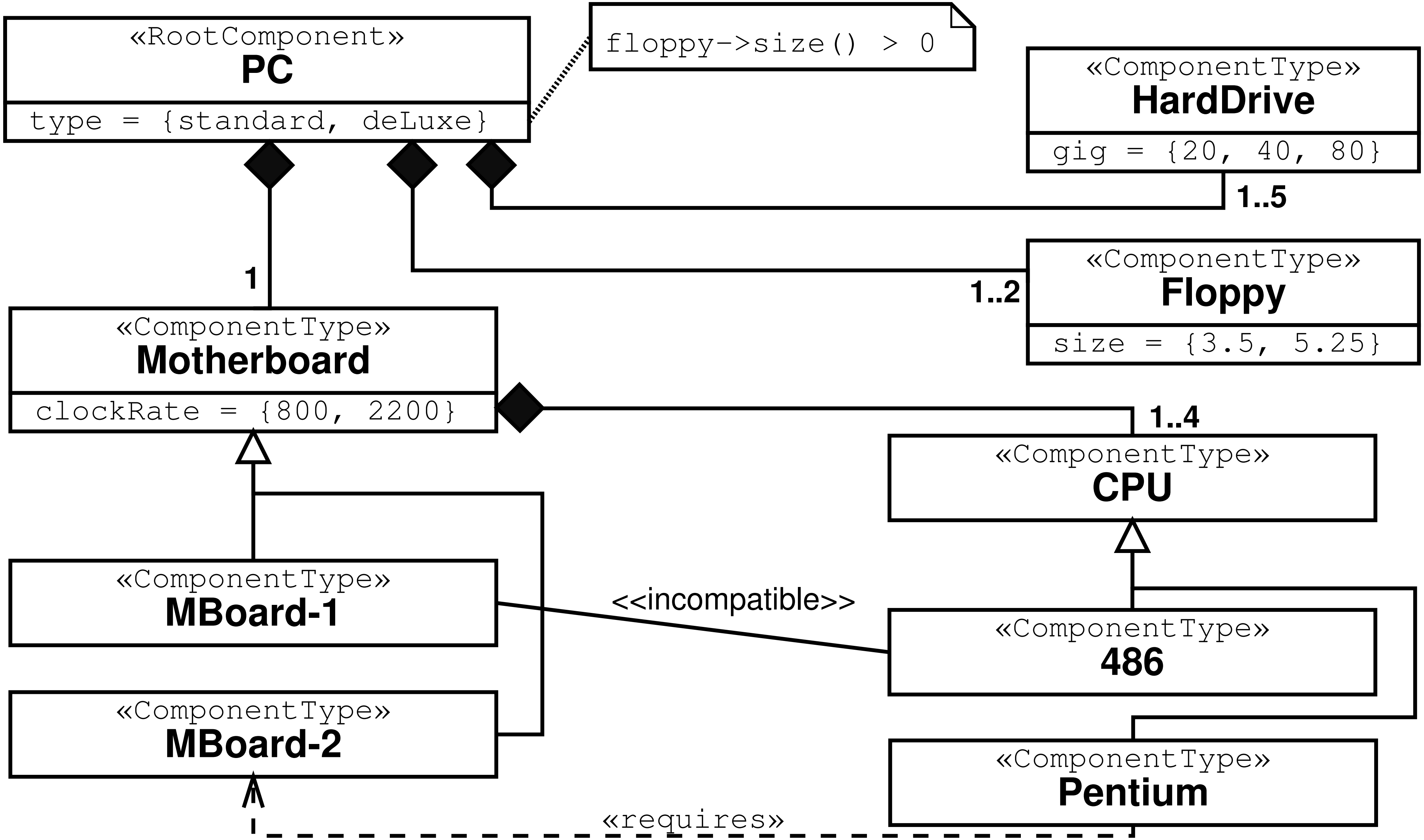
What to do next

- define a precise UML subset
- interfacing with eg. Localizer
- explore model transformation

Fin

Thank you for your attention.

Configuration Knowledge Base in UML



Additional Information: MOF

M3: Meta-metamodel

Hard-wired Meta-metamodel

M2: Metamodel

```
MetaModel( "RECORD_TYPES",
   MetaClass("RECORD",
      [MetaAttr("name", String),
      [MetaAttr("fields", List<"FIELD">) ])
   MetaClass("FIELD",  [ ... ] )
)
```

M1: User Model

```
RECORD("StockQuote",
 [ FIELD("company", String)
   FIELD("price", Real) ]
)
```

M0: User Data

```
StockQuote("Sunbeam Harvesters", 98.88)
StockQuote("Ilog PA", 118.88)
StockQuote("GreedyDudes", 2.13)
                 ......
```

Additional Information: MOF

Meta Object Facility (MOF)

- MOF is a simple language for defining languages
- UML is a MOF-based metamodel

Level	MOF Terms	Examples
M3	meta-metamodel	MOF Model
M2	meta-metadata	UML Metamodel
M1	metadata	Class Diagram
M0	data	User Objects

Additional Information: XMI

- XMI = XML Metadata Interchange (OMG)
- is a way to save UML models in XML
  - for *any* MOF-based metamodel
- ability to move UML models between tools

```
<XMI.header>
  <XMI.documentation>
     <XMI.exporter>Together</XMI.exporter>
     <XMI.exporterVersion>4.0</XMI.exporterVersion>
  </XMI.documentation>
  <XMI.metamodel xmi.name = 'UML' xmi.version = '1.1'/>
</XMI.header>
```

Additional Information: What's in XMI?

- *Document Type Definition* (DTD) rules for transforming MOF based models into XML DTDs
- XML Document production rules for MOF based data

UML can be regarded as:

- an XML document conforming to a DTD
- an XML DTD to which UML models must conform

Caveat: version differences between XMI and UML

- limits tool-to-tool interchange
- XMI 1.0, 1.1 $\Leftrightarrow$ UML1.1, 1.3, 1.4