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

![Diagram showing the relationship between problem description and implementation, with informal language and machine-parseable models.]
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
CSP:

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

CSP:

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

Example:

$X = \{a, b\}; \quad D = \{\mathbb{N}, \mathbb{N}\}$

$C \equiv a < 22 \land a \leq b \land b > 12$
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
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
Support In Other Disciplines:

- Semantic Data Models in Database Design
- Semantic Networks in Knowledge Engineering
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  natural language  Conceptual Model

Source Code
Part 2a: UML
The 4+1 view of UML

**Vocabulary Functionality**

- Design View
- Implementation View
- Use Case View
- Process View
- Deployment View

**System Assembly Configuration Management**

- Vocabulary
- Functionality
- Behaviour
- Scalability
- Performance
- Distribution
- Installation
- System Topology

Modelling Constraints – p.11/42
The 4+1 view of UML

- **Design View**
  - Process View
  - Use Case View
- **Implementation View**
- **Deployment View**

**Vocabulary Functionality**
- Configuration Management
- Distribution
- System Topology

**Behaviour**
- Scalability
- Performance

**System Assembly**
- Installation
- System Topology

Modelling Constraints – p.11/42
Types of UML Diagrams

- **UML diagrams**
  - static view
    - class diagram
    - object diagram
  - implementation diagrams
    - component diagram
    - deployment diagram
  - interaction diagrams
    - use case diagram
    - activity diagram
    - statechart diagram
  - dynamic view
    - sequence diagram
    - collaboration diagram
Types of UML Diagrams

- **UML diagrams**
  - static view
    - class diagram
    - object diagram
classification
Building Blocks (UML)

▷ classification

▷ association
Building Blocks (UML)

- classification
- association
- aggregation

Modelling Constraints – p.14/42
Building Blocks (UML)

- **classification**

- **association**

- **aggregation**

- **attributes**
Building Blocks (UML)

- **classification**
- **association**
- **aggregation**
- **attributes**
- **ISA** (specialization / generalization)
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

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}) }
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, ...)

```
model royloy

context Person
  inv: Set{1,2,3,4} -> isEmpty
  inv: age >= 0

context Person
  inv: age. div (7) < 7
  inv: age. mod (7) < 7
  inv: age. max (7) < 7
  inv: age. min (7) < 7

context Company
  inv: employees -> iterate (p: Person ; b: Bag (Person) = Bag () | b -> including (p)) -> includes (manager)

context Company
  inv: self. numberOfEmployees = employees -> size

Inv_Actor_Graphs.ocl  royloy.ocl
```

Activated project untitled
Opened file /root/work/Modeling/OCLE/examples/Royal_&_Loyal/royloy.ocl.
Opened file /root/work/Modeling/OCLE/examples/Royal_&_Loyal/royloy.ocl.
Attribute access

context Nqueen inv:
    column > 0 and column <= N and
    row > 0 and row <= N
A Flavour of OCL: N-queens

**Attribute access**

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

**Navigation**

```ocl
class Nqueen
context Nqueen inv:
  nQProblem.numQueens >= 3
```
### Pseudo - attributes

--simplifies the other expressions

```ocl
class Nqueen
  def:
  attr N : Integer = nQProblem.numQueens

context Nqueen
def:
  attr N : Integer = nQProblem.numQueens
```
Pseudo - attributes

--simplifies the other expressions

```oclass
context Nqueen def:
attr N : Integer = nQProblem.numQueens
```

Set expressions

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

--no two queens attack another:

context NQProblem inv:
    nqueen->forall(q1, q2 : Nqueen | not q1.attacks(q2) )
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
schedule the setup of a 5 - segment bridge

Martin Bartusch 1983, Massimo Paltrinieri 1994
- 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 $\Rightarrow$ 77 constraints
11 bridge components, 46 tasks, 7 resources

**Number of Constraints:**

- **Resource Constraints:**
  - most tasks multiply on $n = 2\ldots6$ components
  - $\frac{n(n-1)}{2}$ constraints each $\Rightarrow$ 77 constraints

- **Precedences among tasks:** 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 $\Rightarrow 77$ constraints

- Precedences among tasks: 66 constraints

- Five additional requirements: 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 $\Rightarrow 77$ constraints

- Precedences among tasks: 66 constraints

- Five additional requirements: 25 constraints

**Sum: 168 constraints**
Bridge - Building: Constraint Graph

Modelling Constraints – p.26/42
Disjunction: Unary Resources
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 ) )
### Precedence Constraints

**context** Task **inv:**

```plaintext
Task.allInstances->forAll(t: Task|
    t.previous->forAll(prev: Task|
        prev.start +
        prev.duration <=
        t.start ) )
```
- generic constraints apply to all 14 subtypes
- 2 instead of 143
context Erection \textbf{inv}: \\
\text{start} + 6 \leq \text{formwork}.\text{start}
Part 4 – Outlook and Conclusion
**Outlook: Application Development**

XML & OCL Parser

CP Code in Host Language

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

class foo : public baz {
    // ...
};
```
Outlook: Distributed CSPs

Constraint Repository

Constraint Repository

constraint fusing mediator

problem specification

solutions

solutions

XML

XML

XML

XML

CSP

constraint solver

user agent

XML

XML

XML
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 ↔ 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 ↔ 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 ↔ 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. \( R \subseteq A \times B \):
  \[
  (\forall a \in A) \ (\exists b \in B) \ (a, b) \in R
  \]

\[
R(A, B)
\]

\[
\array{A \xrightarrow{R(A, B)} B}
\]
Problems of UML

- informal notation ↔ 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. \( R \subseteq A \times B: \)
  \[
  (\forall a \in A) (\exists b \in B) (a, b) \in R
  \]

Modelling Constraints – p.35/42
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
Thank you for your attention.
Configuration Knowledge Base in UML

- **RootComponent**
  - **PC**
    - type: \{standard, deLuxe\}
    - floppy->size() > 0

- **ComponentType**
  - **HardDrive**
    - gig: \{20, 40, 80\}
  - **Floppy**
    - size: \{3.5, 5.25\}

- **Motherboard**
  - clockRate: \{800, 2200\}
  - requires: <<incompatible>>

- **ComponentType**
  - **MBoard-1**
  - **MBoard-2**

- **CPU**
  - 486
  - Pentium

Modelling Constraints – p.38/42
MOF

M3: Meta-metamodel

M2: Metamodel

M1: User Model

M0: User Data

Hard-wired Meta-metamodel

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

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

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

......
MOF is a simple language for defining languages

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

<table>
<thead>
<tr>
<th>Level</th>
<th>MOF Terms</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>M3</td>
<td>meta-metamodel</td>
<td>MOF Model</td>
</tr>
<tr>
<td>M2</td>
<td>meta-metadata</td>
<td>UML Metamodel</td>
</tr>
<tr>
<td>M1</td>
<td>metadata</td>
<td>Class Diagram</td>
</tr>
<tr>
<td>M0</td>
<td>data</td>
<td>User Objects</td>
</tr>
</tbody>
</table>
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

```xml
<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

\[
\text{XMI 1.0, 1.1} \Leftrightarrow \text{UML 1.1, 1.3, 1.4}
\]