Last Lectures:
- Studied syntax and semantics of associations in the general case.

This Lecture:
- **Educational Objectives:** Capabilities for following tasks/questions.
  - Cont’d: Please explain this class diagram with associations.
  - When is a class diagram a good class diagram?
  - What are purposes of modelling guidelines? (Example?)
  - Discuss the style of this class diagram.

- **Content:**
  - Effect of association semantics on OCL.
  - Treat “the rest”.
  - Where do we put OCL constraints?
  - Modelling guidelines, in particular for class diagrams (following [Ambler, 2005](#))
  - Examples: modelling games (made-up and real-world examples)
Links in System States

\[ \langle r : \langle \text{role}_1 : C_1, \ldots, P_1, \ldots \rangle, \ldots, \langle \text{role}_n : C_n, \ldots, P_n, \ldots \rangle \rangle \]

Only for the course of lectures 08/09 we change the definition of system states:

**Definition.** Let \( \mathcal{D} \) be a structure of the (extended) signature \( \mathcal{I} = (\mathcal{T}, \mathcal{C}, V, \text{atr}) \).

A **system state** of \( \mathcal{I} \) wrt. \( \mathcal{D} \) is a pair \((\sigma, \lambda)\) consisting of

- a type-consistent mapping
  \[ \sigma : \mathcal{D}(\mathcal{C}) \leftrightarrow (\text{atr}(\mathcal{C}) \leftrightarrow \mathcal{D}(\mathcal{T})) \],

- a mapping \( \lambda \) which assigns each association \( \langle r : \langle \text{role}_1 : C_1 \rangle, \ldots, \langle \text{role}_n : C_n \rangle \rangle \in V \) a relation
  \[ \lambda(r) \subseteq \mathcal{D}(C_1) \times \cdots \times \mathcal{D}(C_n) \]
  (i.e. a set of type-consistent \( n \)-tuples of identities).
Example

$\gamma = \{ 1_s \mapsto \{ 1s \mapsto 0, 2_s \mapsto 2s \mapsto 03, 3_s \mapsto 3s \mapsto 01 \}, 2 \mapsto \{ 2s \mapsto 03, 00 \} \}$

$\lambda = \{ t \mapsto \{ (1s, 2s, 3s), (1s, 2s, 3s), (2s, 5s, 6s), (3s, 3s, 3s) \} \}$

Students may join multiple groups

Links may also have dangling references

One student may assume all roles

(add a constraint if this is not desired)

OBJECT DIAGRAMS:

WE WILL NOT FORMALLY DEFINE THAT
**Association/Link Example**

**Signature:**

\[ \mathcal{S} = (\{\text{Int}\}, \{C, D\}, \{x : \text{Int}\}, \langle A\_C\_D : \langle c : C, 0..*, +, \{\text{unique}\}, \times, 1\rangle, \langle n : D, 0..*, +, \{\text{unique}\}, >, 0\rangle\rangle, \{C \mapsto \emptyset, D \mapsto \{x\}\}) \]

A system state of \( \mathcal{S} \) (some reasonable \( \mathcal{D} \)) is \((\sigma, \lambda)\) with:

\[
\sigma = \{1_C \mapsto \emptyset, 3_D \mapsto \{x \mapsto 1\}, 7_D \mapsto \{x \mapsto 2\}\}
\]

\[
\lambda = \{A\_C\_D \mapsto \{(1_C, 3_D), (1_C, 7_D)\}\}
\]

Object \( C \) is related to \( 3_D \) and \( 7_D \) by \( A\_C\_N \).
Associations and OCL
**OCL and Associations: Syntax**

**Recall:** OCL syntax as introduced in Lecture 03, interesting part:

\[
expr ::= \ldots \mid r_1(expr_1) : \tau_C \rightarrow \tau_D \quad r_1 : D_{0,1} \in atr(C) \\
| r_2(expr_1) : \tau_C \rightarrow Set(\tau_D) \quad r_2 : D_* \in atr(C)
\]

Now becomes

\[
expr ::= \ldots \mid role(expr_1) : \tau_C \rightarrow \tau_D \quad \mu = 0..1 \text{ or } \mu = 1 \\
| role(expr_1) : \tau_C \rightarrow Set(\tau_D) \quad \text{otherwise}
\]

if there is
\[
\langle r : \ldots, \langle \text{role} : D, \mu, \ldots \rangle, \ldots, \langle \text{role} : C, \ldots \rangle, \ldots \rangle \in V \quad \text{or} \\
\langle r : \ldots, \langle \text{role} : C, \ldots \rangle, \ldots, \langle \text{role} : D, \mu, \ldots \rangle, \ldots \rangle \in V, \text{role} \neq \text{role}'.
\]

Two rows for tech. reasons: order matters

\[
\overline{C \rightarrow n(D) : \text{never } n(self_D)} \\
\overline{C \rightarrow n : \text{self}_C \text{ is ob}}
\]
**OCL and Associations: Syntax**

**Recall:** OCL syntax as introduced in Lecture 03, interesting part:

\[
expr ::= \ldots \mid r_1(expr_1) : \tau_C \rightarrow \tau_D \quad r_1 : D_{0,1} \in atr(C) \\
| r_2(expr_1) : \tau_C \rightarrow Set(\tau_D) \quad r_2 : D_* \in atr(C)
\]

Now becomes

\[
expr ::= \ldots \mid role(expr_1) : \tau_C \rightarrow \tau_D \quad \mu = 0..1 \text{ or } \mu = 1 \\
| role(expr_1) : \tau_C \rightarrow Set(\tau_D) \quad \text{otherwise}
\]

if

\[
\langle r : \ldots, \langle role : D, \mu, \_\ldots \ldots \_ \rangle, \ldots, \langle role' : C, \_\ldots \ldots \_ \rangle, \ldots \rangle \in V \text{ or} \\
\langle r : \ldots, \langle role' : C, \_\ldots \ldots \_ \rangle, \ldots, \langle role : D, \mu, \_\ldots \ldots \_ \rangle, \ldots \rangle \in V, \text{role} \neq \text{role}'.
\]

**Note:**
- Association name as such doesn’t occur in OCL syntax, role names do.
- \(expr_1\) has to denote an object of a class which “participates” in the association.
OCL and Associations Syntax: Example

\[
\begin{align*}
expr & ::= \ldots \\
& | role(expr_1) : \tau_C \to \tau_D \\
& | role(expr_1) : \tau_C \to Set(\tau_D)
\end{align*}
\]

\[
\mu = 0..1 \text{ or } \mu = 1
\]

\[
\text{if } \langle r: \ldots, \langle \text{role} : D, \mu, \ldots, \ldots \rangle, \ldots, \langle \text{role}' : C, \ldots, \ldots \rangle, \ldots \rangle \in V \text{ or } \langle r: \ldots, \langle \text{role}' : C, \ldots, \ldots \rangle, \ldots, \langle \text{role} : D, \mu, \ldots, \ldots \rangle, \ldots \rangle \in V, \text{ role } \neq \text{ role}'.
\]

Figure 7.21 - Binary and ternary associations [OMG, 2007b, 44].

- context Player inv: size(year(sself)) > 0
- context Player inv: self.p \to size > 0
- context Player inv: self.season \to size > 0
- context Player inv: self.b \to size > 0

OK

NOT OK

OK

OK
OCL and Associations: Semantics

Recall: (Lecture 03)

Assume \( expr_1 : \tau_C \) for some \( C \in \mathcal{C} \). Set \( u_1 := I[expr_1](\sigma, \beta) \in \mathcal{D}(\tau_C) \).

- \( I[r_1(expr_1)](\sigma, \beta) := \begin{cases} u , & \text{if } u_1 \in \text{dom}(\sigma) \text{ and } \sigma(u_1)(r_1) = \{u\} \\ \bot , & \text{otherwise} \end{cases} \)

- \( I[r_2(expr_1)](\sigma, \beta) := \begin{cases} \sigma(u_1)(r_2) , & \text{if } u_1 \in \text{dom}(\sigma) \\ \bot , & \text{otherwise} \end{cases} \)

Now needed:

\[ I[role(expr_1)]((\sigma, \lambda), \beta) \]

- We cannot simply write \( \sigma(u)(role) \).
  Recall: \( role \) is (for the moment) not an attribute of object \( u \) (not in \( atr(C) \)).

- What we have is \( \lambda(r) \) (with \( r \), not with \( role \! \)) — but it yields a set of \( n \)-tuples, of which some relate \( u \) and other some instances of \( D \).

- \( role \) denotes the position of the \( D \)'s in the tuples constituting the value of \( r \).
Assume \( \text{expr}_1 : \tau_C \) for some \( C \in \mathcal{C} \). Set \( u_1 := I[\text{expr}_1]\((\sigma, \lambda), \beta) \in \mathcal{D}(\tau_C) \).

- \( I[\text{role}(\text{expr}_1)]\((\sigma, \lambda), \beta) := \begin{cases} u, & \text{if } u_1 \in \text{dom}(\sigma) \text{ and } L(\text{role})(u_1, \lambda) = \{u\} \\ \bot, & \text{otherwise} \end{cases} \)

- \( I[\text{role}(\text{expr}_1)]\((\sigma, \lambda), \beta) := \begin{cases} L(\text{role})(u_1, \lambda), & \text{if } u_1 \in \text{dom}(\sigma) \\ \bot, & \text{otherwise} \end{cases} \)

where

\[
L(\text{role})(u, \lambda) = \{ (u_1, \ldots, u_n) \in \lambda(r) | u \in \{u_1, \ldots, u_n\} \} \downarrow i
\]

if

\[
\langle r : \ldots \langle \text{role}_1 : \_ \_ \_ \_ \_ \_ \_ \rangle, \ldots \langle \text{role}_n : \_ \_ \_ \_ \_ \_ \_ \rangle, \ldots \rangle, \text{role} = \text{role}_i.
\]

Given a set of \( n \)-tuples \( A \), \( A \downarrow i \) denotes the element-wise projection onto the \( i \)-th component.
\( I[\text{role}(\text{expr}_1)]((\sigma, \lambda), \beta) := \begin{cases} L(\text{role})(u_1, \lambda), & \text{if } u_1 \in \text{dom}(\sigma) \\ \bot, & \text{otherwise} \end{cases} \)

\( L(\text{role})(u, \lambda) = \{(u_1, \ldots, u_n) \in \lambda(r) \mid u \in \{u_1, \ldots, u_n\}\} \downarrow i \)

\( \sigma = \{1_C \mapsto \emptyset, 3_D \mapsto \{x \mapsto 1\}, 7_D \mapsto \{x \mapsto 2\}\} \cup \{(c \mapsto \emptyset)\} \)

\( \lambda = \{\text{\textcolor{red}{\underline{1}}} \mapsto \{(1_C, 3_D), (1_C, 7_D)\}\} \cup \{(2_c, r_D)\} \)

\( I[\text{self} \cdot n]((\sigma, \lambda), \{\text{self} \mapsto 1_C\}) = \{3_D, r_D\} \)

\( = L(n)(1_C, \lambda) \)

\( = \{(1_C, 3_D), (1_C, r_D)\} \downarrow 2 \)

\( = \{3_D, r_D\} \)

\( \langle r: \langle c:C, 0..*\rangle \rangle \)

\( \langle n:D, 0..*\rangle \)
\[ \lambda : \mathcal{F} \rightarrow \mathcal{F} \{ (1_c, 3_{D}, 1_c), (6_c, 7_{D}, 2_c), (7_c, g_{D}, 2_c), (5_c, 6_{D}, 6_c) \} \]

\[ \mathcal{L}(\eta)(1_c, \lambda) = \{ (1_c, 3_{D}, 1_c), (7_c, g_{D}, 2_c) \} \downarrow 2 = \{ 3_{D}, 8_{D} \} \]

\[ \mathcal{L}(\eta)(2_c, \lambda) = \{ 3_{D}, 8_{D} \} \]

\[ \mathcal{L}(\eta)(5_c, \lambda) = \{ 9_{D} \} \]
Associations: The Rest
Visibility

Not so surprising: Visibility of role-names is treated completely similar to visibility of attributes, namely by **typing rules**.

**Question:** given

![Diagram](image)

is the following OCL expression well-typed or not (wrt. visibility):

```ocl
class C {
  inv : self.role.x > 0
}
```

Basically same rule as before: (analogously for other multiplicities)

\[(\text{Assoc}_1) \quad \frac{A, B \vdash expr_1 : \tau_C}{A, B \vdash role(expr_1) : \tau_D}, \quad \mu = 0..1 \text{ or } \mu = 1, \quad \xi = +, \text{ or } \xi = - \text{ and } C = B \]

\[
\langle r : \ldots \langle role : D, \mu, -, \xi, -, - \rangle, \ldots \langle role' : C, -, -, -, -, - \rangle, \ldots \rangle \in V
\]
Navigability is similar to visibility: expressions over non-navigable association ends \((\nu = \times)\) are basically type-correct, but forbidden.

**Question:** given

```
context D inv : self.role.x > 0
```

is the following OCL expression well-typed or not (wrt. navigability):

The standard says:
- \('-'\): navigation is possible
- \('\times'\): navigation is not possible
- \('>'\): navigation is efficient

So: In general, UML associations are different from pointers/references!

But: Pointers/references can faithfully be modelled by UML associations.
Recapitulation: Consider the following association:

\[ \langle r : \langle \text{role}_1 : C_1, \mu_1, P_1, \xi_1, \nu_1, o_1 \rangle, \ldots, \langle \text{role}_n : C_n, \mu_n, P_n, \xi_n, \nu_n, o_n \rangle \rangle \]

- Association name \( r \) and role names/types \( \text{role}_i / C_i \) induce extended system states \( \lambda \).
- Multiplicity \( \mu \) is considered in OCL syntax.
- Visibility \( \xi \) and navigability \( \nu \) give rise to well-typedness rules.

Now the rest:

- Multiplicity \( \mu \): we propose to view them as constraints.
- Properties \( P_i \): even more typing.
- Ownership \( o \): getting closer to pointers/references.
- Diamonds: exercise.
**Multiplicities as Constraints**

**Recall:** The multiplicity of an association end is a term of the form:

\[ \mu ::= * \mid N \mid N..M \mid N..* \mid \mu, \mu \quad (N, M \in \mathbb{N}) \]

**Proposal:** View multiplicities (except 0..1, 1) as additional invariants/constraints.

**Recall:** we can normalize each multiplicity to the form

\[ \mu = N_1..N_2, \ldots, N_{2k-1}..N_{2k} \]

where \( N_i \leq N_{i+1} \) for \( 1 \leq i \leq 2k \), \( N_1, \ldots, N_{2k} \in \mathbb{N}, N_{2k} \in \mathbb{N} \cup \{*\} \).

**Define**

\[ \mu_{OCL} = \text{context } C \text{ inv : } \]

\[ (N_1 \leq \text{role } \rightarrow \text{size()} \leq N_2) \lor \ldots \lor (N_{2k-1} \leq \text{role } \rightarrow \text{size()} \leq N_{2k}) \]

for each

\[ \langle r : \ldots, \langle \text{role : } D, \mu, \ldots \rangle, \ldots, \langle \text{role' : } C, \ldots \rangle, \ldots \rangle \in V \text{ or } \]

\[ \langle r : \ldots, \langle \text{role' : } C, \ldots \rangle, \ldots, \langle \text{role : } D, \mu, \ldots \rangle, \ldots \rangle \in V, \text{ role } \neq \text{ role'}. \]

**Note:** in n-ary associations with \( n > 2 \), there is redundancy.
Context C inv:

\( n \rightarrow \text{size} \geq 3 \)

and \( n \rightarrow \text{size} \leq 5 \)

or \( n \rightarrow \text{size} \geq 7 \)
Multiplicities as Constraints of Class Diagram

Recall:

\[ \mathcal{CD} = \{ CD_1, \ldots, CD_n \} \]

signature \( \mathcal{I}(\mathcal{CD}) \)

invariants \( \text{Inv}(\mathcal{CD}) \)

basic
(classes and attributes)

distinguish

extended (visibility)

\[ \text{from now on: } \text{Inv}(\mathcal{CD}) = \{ \text{constraints occurring in notes} \} \cup \{ \mu_{\text{OCL}} \mid \]

\[ \langle r : \ldots, \langle \text{role} : D, \mu, \_ , \_ , \_ \rangle, \ldots, \langle \text{role}' : C, \_ , \_ , \_ , \_ \rangle, \ldots \rangle \in V \text{ or } \]

\[ \langle r : \ldots, \langle \text{role}' : C, \_ , \_ , \_ , \_ \rangle, \ldots, \langle \text{role} : D, \mu, \_ , \_ , \_ \rangle, \ldots \rangle \in V, \]

\[ \text{role} \neq \text{role}', \mu \notin \{0..1, 1\} \}. \]
Multiplicities as Constraints Example

\[ \mu_{OCL} = \text{context } C \text{ inv} : \]
\[ (N_1 \leq \text{role } \rightarrow \text{size()} \leq N_2) \text{ and } \ldots \text{ and } (N_{2k-1} \leq \text{role } \rightarrow \text{size()} \leq N_{2k}) \]

**CD:**

\[ \text{Inv}(CD) = \]
\[ \bullet \{ \text{context } C \text{ inv} : 4 \leq \text{role}_2 \rightarrow \text{size()} \leq 4 \text{ or } 17 \leq \text{role}_2 \rightarrow \text{size()} \leq 17 \} \]
\[ = \{ \text{context } C \text{ inv} : \text{role}_2 \rightarrow \text{size()} = 4 \text{ or } \text{role}_2 \rightarrow \text{size()} = 17 \} \]
\[ \cup \{ \text{context } C \text{ inv} : 3 \leq \text{role}_3 \rightarrow \text{size()} \} \]
Why Multiplicities as Constraints?

More precise, can’t we just use types? (cf. Slide [29])

- $\mu = 0..1, \mu = 1$:
  many programming language have direct correspondences (the first corresponds to type pointer, the second to type reference) — this is why we excluded them.

- $\mu = *$:
  could be represented by a set data-structure type without fixed bounds — no problem with our approach, we have $\mu_{OCL} = true$ anyway.

- $\mu = 0..4$:
  use array of size 4 — if model behaviour (or the implementation) adds 5th identity, we’ll get a runtime error, and thereby see that the constraint is violated. **Principally acceptable**, but: checks for array bounds everywhere...?

- $\mu = 5..7$:
  could be represented by an array of size 7 — but: few programming languages/data structure libraries allow lower bounds for arrays (other than 0). If we have 5 identities and the model behaviour removes one, this should be a violation of the constraints imposed by the model. The implementation which does this removal is wrong. How do we see this...?
Well, if the **target platform** is known and fixed, and the target platform has, for instance,

- reference types,
- range-checked arrays with positions $0, \ldots, N$,
- set types,

then we could simply **restrict** the syntax of multiplicities to

\[ \mu ::= 1 \mid 0..N \mid * \]

and don’t think about constraints (but use the obvious 1-to-1 mapping to types)... 

In general, **unfortunately**, we don’t know.
Properties

We don’t want to cover association properties in detail, only some observations (assume binary associations):

<table>
<thead>
<tr>
<th>Property</th>
<th>Intuition</th>
<th>Semantical Effect</th>
</tr>
</thead>
<tbody>
<tr>
<td>unique</td>
<td>one object has at most one $r$-link to a single other object</td>
<td>current setting</td>
</tr>
<tr>
<td>bag</td>
<td>one object may have multiple $r$-links to a single other object</td>
<td>have $\lambda(r)$ yield multi-sets</td>
</tr>
<tr>
<td>ordered, sequence</td>
<td>an $r$-link is a sequence of object identities (possibly including duplicates)</td>
<td>have $\lambda(r)$ yield sequences</td>
</tr>
</tbody>
</table>

\[\lambda(r)\]
## Properties

We don’t want to cover association properties in detail, only some observations (assume binary associations):

<table>
<thead>
<tr>
<th>Property</th>
<th>Intuition</th>
<th>Semantical Effect</th>
</tr>
</thead>
<tbody>
<tr>
<td>unique</td>
<td>one object has <strong>at most one</strong> (r)-link to a single other object</td>
<td><strong>current setting</strong></td>
</tr>
<tr>
<td>bag</td>
<td>one object may have <strong>multiple</strong> (r)-links to a single other object</td>
<td>have (\lambda(r)) yield multi-sets</td>
</tr>
<tr>
<td>ordered, sequence</td>
<td>an (r)-link is a <strong>sequence</strong> of object identities (possibly including duplicates)</td>
<td>have (\lambda(r)) yield sequences</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Property</th>
<th>OCL Typing of expression (role(expr))</th>
</tr>
</thead>
<tbody>
<tr>
<td>unique</td>
<td>(\tau_D \rightarrow \text{Set}(\tau_C))</td>
</tr>
<tr>
<td>bag</td>
<td>(\tau_D \rightarrow \text{Bag}(\tau_C))</td>
</tr>
<tr>
<td>ordered, sequence</td>
<td>(\tau_D \rightarrow \text{Seq}(\tau_C))</td>
</tr>
</tbody>
</table>

For **subsets**, **redefines**, **union**, etc. see [OMG, 2007a, 127].
Ownership

Intuitively it says:

Association $r$ is **not a “thing on its own”** (i.e. provided by $\lambda$),
but association end ‘role’ is **owned** by $C$ (!).
(That is, it’s stored inside $C$ object and provided by $\sigma$).

**So:** if multiplicity of role is 0..1 or 1, then the picture above is very close to
concepts of pointers/references.

Actually, ownership is seldom seen in UML diagrams. Again: if target platform
is clear, one may well live without (cf. [OMG, 2007b, 42] for more details).

**Not clear to me:**
Back to the Main Track
Back to the main track:

Recall: on some earlier slides we said, the extension of the signature is **only** to study associations in “full beauty”. For the remainder of the course, we should look for something simpler...

Proposal:

- **from now on**, we only use associations of the form

  - Form (i) introduces \( role : C_{0,1} \), and form (ii) introduces \( role : C_* \) in \( V \).
  
- In both cases, \( role \in atr(C) \).

- We drop \( \lambda \) and go back to our nice \( \sigma \) with \( \sigma(u)(role) \subseteq \mathcal{D}(D) \).
OCL Constraints in (Class) Diagrams
Where Shall We Put OCL Constraints?

Two options:
(i) Notes.
(ii) Particular dedicated places.

(i) Notes:
A UML note is a picture of the form

\[
\text{[}\begin{array}{c}
\text{text} \\
\text{[}\end{array}\text{]}
\]

\text{text} can principally be everything, in particular comments and constraints.

Sometimes, content is explicitly classified for clarity:

\[
\text{OCL:}
\begin{array}{c}
\text{expr}
\end{array}
\]
OCL in Notes: Conventions

stands for

context $C$ inv : expr
(ii) **Particular dedicated places** in class diagrams:  (behav. feature: later)

\[
C \\
\xi v : \tau \{p_1, \ldots, p_n\} \{expr\} \\
\xi f(v_1 : \tau, \ldots, v_n : \tau_n) : \tau \{p_1, \ldots, p_n\} \{\text{pre : } expr_1 \text{ post : } expr_2\}
\]

For simplicity, we view the above as an abbreviation for

\[
C \\
\xi v : \tau \{p_1, \ldots, p_n\} \\
\text{context } f \text{ pre : } expr_1 \text{ post : } expr_2
\]
Invariants of a Class Diagram

- Let \( CD \) be a class diagram.
- As we (now) are able to recognise OCL constraints when we see them, we can define

\[
\text{Inv}(CD)
\]

as the set \( \{\varphi_1, \ldots, \varphi_n\} \) of OCL constraints occurring in notes in \( CD \) — after unfolding all abbreviations (cf. next slides).

- As usual: \( \text{Inv}(\mathcal{D}) := \bigcup_{CD \in \mathcal{D}} \text{Inv}(CD) \).

- **Principally clear**: \( \text{Inv} (\cdot) \) for any kind of diagram.
If $\mathcal{CD}$ consists of only $CD$ with the single class $C$, then

- $\text{Inv}(\mathcal{CD}) = \text{Inv}(CD) =$
**Definition.** Let $\mathcal{CD}$ be a set of class diagrams. We say, the *semantics* of $\mathcal{CD}$ is the signature it induces and the set of OCL constraints occurring in $\mathcal{CD}$, denoted

$$[\mathcal{CD}] := \langle \mathcal{I}(\mathcal{CD}), \text{Inv}(\mathcal{CD}) \rangle.$$ 

Given a structure $\mathcal{D}$ of $\mathcal{I}$ (and thus of $\mathcal{CD}$), the class diagrams describe the system states $\Sigma^\mathcal{D}$, of which some may satisfy $\text{Inv}(\mathcal{CD})$.

**In pictures:**

$\mathcal{CD} = \{CD_1, \ldots, CD_n\}$

- **Basic** (classes and attributes)
- **Distinguish** (classes and attributes)
- **Extended** (visibility)

including constraints induced by multiplicities.

invovling
References
References

