Software Design, Modelling and Analysis in UML

Lecture 09: Class Diagrams III

2013-11-25

Prof. Dr. Andreas Podelski, **Dr. Bernd Westphal**

Albert-Ludwigs-Universität Freiburg, Germany

Contents & Goals

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

$\langle r : \langle role_1 : C_1, \_, P_1, \_, \_, \_ \rangle, \ldots, \langle role_n : C_n, \_, P_n, \_, \_, \_ \rangle$

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

Definition. Let $\mathscr{D}$ be a structure of the (extended) signature $\mathscr{S} = (\mathscr{T}, \mathscr{C}, V, atr)$. for associations

A system state of $\mathscr{S}$ wrt. $\mathscr{D}$ is a pair (σ, λ) consisting of

- a type-consistent mapping

$$\sigma : \mathscr{D}(\mathscr{C}) \nrightarrow (atr(\mathscr{C}) \nrightarrow \mathscr{D}(\mathscr{T})),$$

values for basic type attributes only

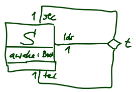
- a mapping λ which assigns each association $\langle r : \langle role_1 : C_1 \rangle, \ldots, \langle role_n : C_n \rangle \rangle \in V$ a relation

$$\lambda(r) \subseteq \mathscr{D}(C_1) \times \cdots \times \mathscr{D}(C_n)$$

(i.e. a set of type-consistent n-tuples of identities).

Example

$\langle t : \langle ldr : S \rangle$
$\langle sec : S \rangle$
$\langle ter : S \rangle \rangle$



$\sigma = \{1_S \mapsto \{aw \mapsto 1\}, 2_S \mapsto \{aw \mapsto 0\}, 3_S \mapsto \{aw \mapsto 1\}, 27_S \mapsto \{aw \mapsto 0\}\}$

$$\lambda = \{ t \mapsto \{ \overset{ldr}{(1_S}, \overset{sec}{2_S}, \overset{ter}{3_S}), (1_S, 27_S, 3_S), (2_S, 5_S, 6_S), (3_S, 3_S, 3_S) \} \}$$

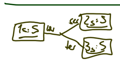
$\supseteq \mathscr{D}(S) \times \mathscr{D}(S) \times \mathscr{D}(S)$

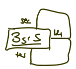
students may join multiple groups

links may also have dangling references

one student may assume all roles (add a constraint if this is not desired)

OBJECT DIAGRAMS:





↳ we need hyperedges in general

WE WILL NOT FORMALLY DEFINE THAT

Association/Link Example

Signature:

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

by convention

by default

A **system state** of $\mathscr{S}$ (some reasonable $\mathscr{D}$) is (σ, λ) with:

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

$$\lambda = \{A\_C\_D \mapsto \{\underbrace{(1_C, 3_D), (1_C, 7_D)}_{\text{object } 1_C \text{ is related to } 3_D \text{ and } 7_D \text{ by } A\_C\_D}\}\}$$

Associations and OCL

OCL and Associations: Syntax

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

$$\begin{array}{llll} expr ::= \dots & \mid r_1(expr_1) & : \tau_C \to \tau_D & r_1 : D_{0,1} \in atr(C) \\ & \mid r_2(expr_1) & : \tau_C \to Set(\tau_D) & r_2 : D_* \in atr(C) \end{array}$$

Now becomes

$$\begin{array}{llll} expr ::= \dots & \mid role(expr_1) & : \tau_C \to \tau_D & \mu = 0..1 \text{ or } \mu = 1 \\ & \mid role(expr_1) & : \tau_C \to Set(\tau_D) & \text{otherwise} \end{array}$$

if there is

C participates in assoc. r

$\langle r : \dots, \langle role : D, \mu, \_, \_, \_, \_\rangle, \dots, \langle role' : C, \_, \_, \_, \_, \_\rangle, \dots\rangle \in V$ or

$\langle r : \dots, \langle role' : C, \_, \_, \_, \_, \_\rangle, \dots, \langle role : D, \mu, \_, \_, \_, \_\rangle, \dots\rangle \in V$, $\underline{role \neq role'}$. ?

two rows for tech. reasons: order matters

$C \xrightarrow{c \quad n} D$: never $n(self_D)$

C n p : $n(self_C)$ is ok

OCL and Associations: Syntax

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

$$\begin{array}{llll} expr ::= \dots & \mid r_1(expr_1) & : \tau_C \to \tau_D & r_1 : D_{0,1} \in atr(C) \\ & \mid r_2(expr_1) & : \tau_C \to Set(\tau_D) & r_2 : D_* \in atr(C) \end{array}$$

Now becomes

$$\begin{array}{llll} expr ::= \dots & \mid role(expr_1) & : \tau_C \to \tau_D & \mu = 0..1 \text{ or } \mu = 1 \\ & \mid role(expr_1) & : \tau_C \to Set(\tau_D) & \text{otherwise} \end{array}$$

if

$\langle r : \dots, \langle role : D, \mu, \_, \_, \_, \_\rangle, \dots, \langle role' : C, \_, \_, \_, \_, \_\rangle, \dots\rangle \in V$ or

$\langle r : \dots, \langle role' : C, \_, \_, \_, \_, \_\rangle, \dots, \langle role : D, \mu, \_, \_, \_, \_\rangle, \dots\rangle \in V, role \neq role'$.

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

$$\begin{array}{llll} expr ::= \dots & \mid role(expr_1) & : \tau_C \to \tau_D & \mu = 0..1 \text{ or } \mu = 1 \\ & \mid role(expr_1) & : \tau_C \to Set(\tau_D) & \text{otherwise} \end{array}$$

if

$\langle r : \dots, \langle role : D, \mu, \_, \_, \_, \_\rangle, \dots, \langle role' : C, \_, \_, \_, \_, \_\rangle, \dots\rangle \in V$ or

$\langle r : \dots, \langle role' : C, \_, \_, \_, \_, \_\rangle, \dots, \langle role : D, \mu, \_, \_, \_, \_\rangle, \dots\rangle \in V, role \neq role'$.

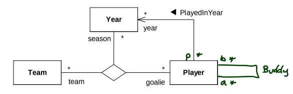


Figure 7.21 - Binary and ternary associations [OMG, 2007b, 44].

- context Player inv: size(year(self)) > 0 OK
- context Player inv: self.p → size > 0 NOT OK
- context Player inv: self.season → size > 0 OK
- context Player inv: self.b → size > 0 OK

OCL and Associations: Semantics

Recall: (Lecture 03)

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

- $I[\![r_1(expr_1)]\!](\sigma, \beta) := \begin{cases} u & \text{, if } u_1 \in \mathrm{dom}(\sigma) \text{ and } \sigma(u_1)(r_1) = \{u\} \\ \perp & \text{, otherwise} \end{cases}$
- $I[\![r_2(expr_1)]\!](\sigma, \beta) := \begin{cases} \sigma(u_1)(r_2) & \text{, if } u_1 \in \mathrm{dom}(\sigma) \\ \perp & \text{, otherwise} \end{cases}$ $\subseteq \mathscr{D}(D)$

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.

OCL and Associations: Semantics Cont'd

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

- $I[\![role(expr_1)]\!]((\sigma, \lambda), \beta) := \begin{cases} u & \text{, if } u_1 \in \mathrm{dom}(\sigma) \text{ and } L(role)(u_1, \lambda) = \{u\} \\ \perp & \text{, otherwise} \end{cases}$ $\mu = 1$ or $\mu = 0..1$
- $I[\![role(expr_1)]\!]((\sigma, \lambda), \beta) := \begin{cases} L(role)(u_1, \lambda) & \text{, if } u_1 \in \mathrm{dom}(\sigma) \\ \perp & \text{, otherwise} \end{cases}$

where

role name, object identits, links, projection on the i-th component

$$L(role)(u, \lambda) = \Big(\{(u_1, \dots, u_n) \in \lambda(r) \mid u \in \{u_1, \dots, u_n\}\} \downarrow i\Big) \subseteq \mathscr{D}(\mathscr{C})$$

if

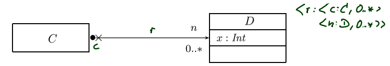
$$\langle r : \dots \langle role_1 : \_, \_, \_, \_, \_, \_\rangle, \dots \langle role_n : \_, \_, \_, \_, \_, \_\rangle, \dots\rangle, role = role_i.$$

Given a set of n-tuples A, $A \downarrow i$ denotes the element-wise projection onto the i-th component.

OCL and Associations Example

$$I[\![role(expr_1)]\!]((\sigma, \lambda), \beta) := \begin{cases} L(role)(u_1, \lambda) & \text{, if } u_1 \in \mathrm{dom}(\sigma) \\ \perp & \text{, otherwise} \end{cases}$$

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



$\langle r : \langle c : C, 0..* \rangle$
$\langle n : D, 0..* \rangle \rangle$

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

$$\lambda = \{\cancel{A\_C\_D}_{\,r} \mapsto \{(1_C, 3_D), (1_C, 7_D)\} \cup \{(2_C, 7_D)\}\}$$

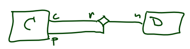
$I[\![self\,.\,n]\!]((\sigma, \lambda), \{self \mapsto 1_C\})$ $= \{3_D, 7_D\}$

all tuples where 1_C occurs

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

$= \{(1_C, 3_D), (1_C, 7_D)\} \downarrow 2$ position of role n in r

$= \{3_D, 7_D\})$



⟨r : ⟨c : C⟩, ⟨n : D⟩, ⟨p : C⟩⟩

$\lambda = \{ r \mapsto \{ (1_C, 3_D, 1_C), (6_C, 7_D, 2_C), (1_C, 8_D, 2_C), (5_C, 9_D, 6_C) \} \}$

$L(n)(1_C, \lambda) = \{ (1_C, 3_D, 1_C), (1_C, 8_D, 2_C) \} \downarrow 2 = \{ 3_D, 8_D \}$

$L(n)(2_C, \lambda) = \{ 7_D, 8_D \}$

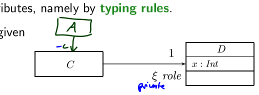
$L(n)(5_C, \lambda) = \{ 9_D \}$

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



$self_A . c$ OK
$self_A . c . role$ NOT OK
$self_C . role$ OK

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

context C inv : $self.role.x > 0$

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

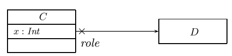
$$(Assoc_1) \quad \frac{A, B \vdash expr_1 : \tau_C}{A, B \vdash role(expr_1) : \tau_D}, \quad \begin{array}{l}\mu = 0..1 \text{ or } \mu = 1,\\ \xi = +, \text{ or } \xi = - \text{ and } C = B\end{array}$$

$$\langle r : \ldots \langle role : D, \mu, \_, \xi, \_, \_\rangle, \ldots \langle role' : C, \_, \_, \_, \_, \_\rangle, \ldots \rangle \in V$$

Navigability

Navigability is similar to visibility: expressions over non-navigable association ends ($\nu = \times$) are **basically** type-correct, but **forbidden**.

Question: given



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

context D inv : $self.role.x > 0$ NOT well-typed

The standard says:
- '–': navigation is possible
- '×': navigation is not possible
- '>': navigation is efficient

depends on context,
e.g. slow vs fast database,
needs to be communicated to developers

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

But: Pointers/references can faithfully be modelled by UML associations.

The Rest

Recapitulation: Consider the following association:

$$\langle r : \langle role_1 : C_1, \mu_1, P_1, \xi_1, \nu_1, o_1\rangle, \ldots, \langle role_n : C_n, \mu_n, P_n, \xi_n, \nu_n, o_n\rangle\rangle$$

- Association name r and role names/types $role_i / C_i$ induce extended system states λ.
- Multiplicity μ is considered in OCL syntax.
- Visibility ξ and navigability ν give rise to well-typedness rules.

Now the rest:
- Multiplicity μ: 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:

$$\mu ::= * \mid N \mid N..M \mid N..* \mid \mu, \mu \qquad (N, M \in \mathbb{N})$$

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 $1 \leq i \leq 2k$, $\quad N_1, \ldots, N_{2k} \in \mathbb{N}$, $\quad N_{2k} \in \mathbb{N} \cup \{*\}$.

Define

$\mu_{\mathsf{OCL}} =$ context C inv :

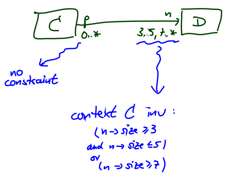
$$(N_1 \leq role \text{->} \mathrm{size}() \leq N_2) \ \overset{\text{or}}{\cancel{\text{and}}} \ \ldots \ \overset{\text{or}}{\cancel{\text{and}}} \ (N_{2k-1} \leq role \text{->} \mathrm{size}() \leq N_{2k})$$

for each

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

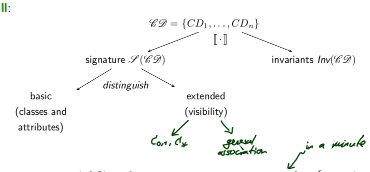
$$\langle r : \ldots, \langle role' : C, \_, \_, \_, \_, \_\rangle, \ldots, \langle role : D, \mu, \_, \_, \_, \_\rangle, \ldots\rangle \in V, role \neq role'.$$

Note: in n-ary associations with $n > 2$, there is redundancy.



Multiplicities as Constraints of Class Diagram

Recall:



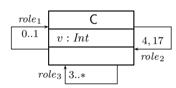
From now on: $Inv(\mathscr{CD}) = \{\text{constraints occurring in notes}\} \cup \{\mu_{\mathsf{OCL}} \mid$

$$\langle r : \ldots, \langle role : D, \mu, \_, \_, \_, \_\rangle, \ldots, \langle role' : C, \_, \_, \_, \_, \_\rangle, \ldots\rangle \in V \text{ or}$$
$$\langle r : \ldots, \langle role' : C, \_, \_, \_, \_, \_\rangle, \ldots, \langle role : D, \mu, \_, \_, \_, \_\rangle, \ldots\rangle \in V,$$
$$role \neq role', \mu \notin \{0..1, 1\}\}.$$

Multiplicities as Constraints Example

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

$\mathcal{CD}$:



$Inv(\mathcal{CD}) =$

- $\{$context C inv : $4 \leq role_2$ -> size$() \leq 4$ or $17 \leq role_2$ -> size$() \leq 17\}$
  $= \{$context C inv : $role_2$ -> size$() = 4$ or $role_2$ -> size$() = 17\}$
- $\cup \{$context C inv : $3 \leq role_3$ -> 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 $\mu_{\mathsf{OCL}} = \mathit{true}$ anyway.
- $\mu = 0..\cancel{3}^{4}$:
  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...?

Multiplicities Never as Types...?

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

$$\mu ::= 1 \mid 0..N \mid *$$

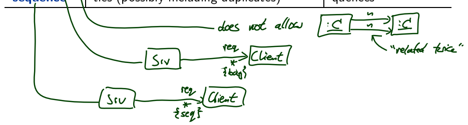
and don't think about constraints
(but use the obvious 1-to-1 mapping to types)...

In general, **unfortunately**, we don't know.

Properties

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

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



Properties

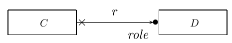
We don't want to cover association **properties** in detail,
only some observations (assume binary associations):

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

Property	OCL Typing of expression $role(expr)$
unique	$\tau_D \to Set(\tau_C)$
bag	$\tau_D \to Bag(\tau_C)$
ordered, **sequence**	$\tau_D \to Seq(\tau_C)$

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

Ownership



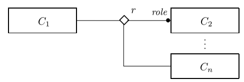
Intuitively it says:

Association r is **not a "thing on its own"** (i.e. provided by λ),
but association end '$role$' is **owned** by C (!).
(That is, it's stored inside C object and provided by σ).

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:



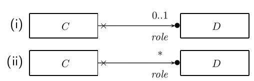
Back to the Main Track

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



(And we may omit the non-navigability and ownership symbols.)

- Form (i) introduces $role : C_{0,1}$, and form (ii) introduces $role : C_*$ in V.
- In both cases, $role \in atr(C)$.
- We drop λ and go back to our nice σ with $\sigma(u)(role) \subseteq \mathscr{D}(D)$.

OCL Constraints in (Class) Diagrams

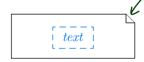
Where Shall We Put OCL Constraints?

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

(i) **Notes**:

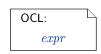
A UML **note** is a picture of the form

Eselsohr (dog's ear)

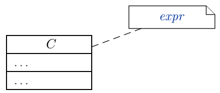


$text$ can principally be **everything**, in particular **comments** and **constraints**.

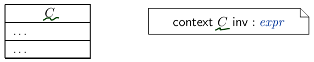
Sometimes, content is **explicitly classified** for clarity:



OCL in Notes: Conventions

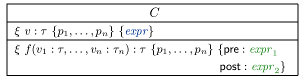


stands for

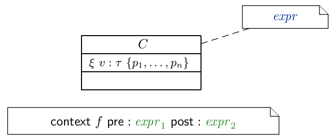


Where Shall We Put OCL Constraints?

(ii) **Particular dedicated places** in class diagrams: (behav. feature: later)



For simplicity, we view the above as an abbreviation for



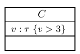
Invariants of a Class Diagram

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

  $$Inv(\mathcal{CD})$$

  as the set $\{\varphi_1, \dots, \varphi_n\}$ of OCL constraints **occurring** in notes in $\mathcal{CD}$ — after **unfolding** all abbreviations (cf. next slides).
- As usual: $Inv(\mathscr{CD}) := \bigcup_{\mathcal{CD} \in \mathscr{CD}} Inv(\mathcal{CD})$.
- **Principally clear:** $Inv(\cdot)$ for any kind of diagram.

Invariant in Class Diagram Example



If $\mathscr{CD}$ consists of only $\mathcal{CD}$ with the single class C, then

- $Inv(\mathscr{CD}) = Inv(\mathcal{CD}) =$

Semantics of a Class Diagram

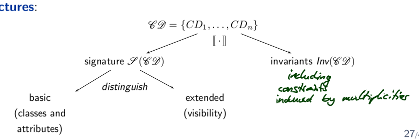
Definition. Let $\mathscr{CD}$ be a set of class diagrams.

We say, the semantics of $\mathscr{CD}$ is the signature it induces and the set of OCL constraints occurring in $\mathscr{CD}$, denoted

$$[\![\mathscr{CD}]\!] := \langle \mathscr{S}(\mathscr{CD}), Inv(\mathscr{CD}) \rangle.$$

Given a structure $\mathscr{D}$ of $\mathscr{S}$ (and thus of $\mathscr{CD}$), the class diagrams describe the system states $\Sigma_{\mathscr{S}}^{\mathscr{D}}$, of which **some** may satisfy $Inv(\mathscr{CD})$.

In pictures:



References

References

[Ambler, 2005] Ambler, S. W. (2005). *The Elements of UML 2.0 Style*. Cambridge University Press.

[OMG, 2007a] OMG (2007a). Unified modeling language: Infrastructure, version 2.1.2. Technical Report formal/07-11-04.

[OMG, 2007b] OMG (2007b). Unified modeling language: Superstructure, version 2.1.2. Technical Report formal/07-11-02.