Chapter 8, Object Design: Object Constraint Language

OCL: Object Constraint Language

- Formal language for expressing constraints over a set of objects and their attributes
- Part of the UML standard
- Used to write constraints that cannot otherwise be expressed in a diagram
- Declarative
  - No side effects
  - No control flow
- Based on Sets and Multi Sets

OCL Basic Concepts

- OCL expressions
  - Return True or False
  - Are evaluated in a specified context,
    - either a class or an operation
  - All constraints apply to all instances

OCL Simple Predicates

Example:
context Tournament inv:
self.getMaxNumPlayers() > 0

In English:
"The maximum number of players in any tournament should be a positive number."

Notes:
- "self" denotes all instances of "Tournament"
- OCL uses the same dot notation as Java.

OCL Preconditions

Example:
context Tournament::acceptPlayer(p) pre:
not self.isPlayerAccepted(p)

In English:
"The acceptPlayer(p) operation can only be invoked if player p has not yet been accepted in the tournament."

Notes:
- The context of a precondition is an operation
- isPlayerAccepted(p) is an operation defined by the class Tournament

OCL Postconditions

Example:
context Tournament::acceptPlayer(p) post:
self.getNumPlayers() =
self@pre.getNumPlayers() + 1

In English:
"The number of accepted player in a tournament increases by one after the completion of acceptPlayer()."

Notes:
- self@pre denotes the state of the tournament before the invocation of the operation.
- Self denotes the state of the tournament after the completion of the operation.
OCL Contract for `acceptPlayer()` in `Tournament`

```plaintext
context Tournament::acceptPlayer(p) pre:
    not isPlayerAccepted(p)
context Tournament::acceptPlayer(p) pre:
    getNumPlayers() < getMaxNumPlayers()
context Tournament::acceptPlayer(p) post:
    isPlayerAccepted(p)
context Tournament::acceptPlayer(p) post:
    getNumPlayers() = @pre.getNumPlayers() + 1
```

OCL Contract for `removePlayer()` in `Tournament`

```plaintext
context Tournament::removePlayer(p) pre:
    isPlayerAccepted(p)
context Tournament::removePlayer(p) post:
    not isPlayerAccepted(p)
context Tournament::removePlayer(p) post:
    getNumPlayers() = @pre.getNumPlayers() - 1
```

Java Implementation of `Tournament` class (Contract as a set of JavaDoc comments)

```java
public class Tournament {
    /** The maximum number of players * is positive at all times. *
     * Invariant: maxNumPlayers > 0 */
    private int maxNumPlayers;
    /** The players List contains >= references to Players who are * are registered with the * Tournament. */
    private List players;
    /** Returns the current number of * players in the tournament. */
    public int getNumPlayers() { ... }
    /** Returns the maximum number of * players in the tournament. */
    public int getMaxNumPlayers() { ... }
    /** The acceptPlayer() operation * assumes that the specified * player has not been accepted * in the Tournament yet. *
     * @pre !isPlayerAccepted(p)
     * @pre getNumPlayers() < maxNumPlayers
     * @post isPlayerAccepted(p)
     * @post getNumPlayers() = @pre.getNumPlayers() + 1 */
    public void acceptPlayer(Player p) { ... }
    /** The removePlayer() operation * assumes that the specified player * is currently in the Tournament. *
     * @pre isPlayerAccepted(p)
     * @post !isPlayerAccepted(p)
     * @post getNumPlayers() = @pre.getNumPlayers() - 1 */
    public void removePlayer(Player p) { ... }
}
```

Constraints can involve more than one class

How do we specify constraints on on a group of classes?

Starting from a specific class in the UML class diagram, we navigate the associations in the class diagram to refer to the other classes and their properties (attributes and Operations).

Example from ARENA: League, Tournament and Player

Constraints:
1. A Tournament’s planned duration must be under one week.
2. Players can be accepted in a Tournament only if they are already registered with the corresponding League.
3. The number of active Players in a League are those that have taken part in at least one Tournament of the League.

Instance Diagram: 2 Leagues, 5 Players, 2 Tournaments
3 Types of Navigation through a Class Diagram

1. Local attribute
   - Tournament
     - start: Date
     - end: Date

2. Directly related class
   - League
     - Player

3. Indirectly related class
   - Tournament

Any constraint for an arbitrary UML class diagram can be specified using only a combination of these 3 navigation types!

Specifying the Model Constraints in OCL

Local attribute navigation
- context Tournament
- inv:
  - end - start <= 7

Directly related class navigation
- context Tournament::acceptPlayer(p)
- pre:
  - league.players->includes(p)

OCL Sets, Bags and Sequences

- Sets, Bags and Sequences are predefined in OCL and subtypes of Collection. OCL offers a large number of predefined operations on collections. They are all of the form:
  - collection->operation(arguments)

OCL-Collection

- The OCL-Type Collection is the generic superclass of a collection of objects of Type T
- Subclasses of Collection are:
  - Set: Set in the mathematical sense. Every element can appear only once
  - Bag: A collection, in which elements can appear more than once (also called multiset)
  - Sequence: A multiset, in which the elements are ordered
- Example for Collections:
  - Set(Integer): a set of integer numbers
  - Bag(Person): a multiset of persons
  - Sequence(Customer): a sequence of customers

OCL-Operations for OCL-Collections (1)

- size: Integer
  - Number of elements in the collection
- includes(o:OclAny): Boolean
  - True, if the element o is in the collection
- count(o:OclAny): Integer
  - Counts how many times an element is contained in the collection
- isEmpty: Boolean
  - True, if the collection is empty
- notEmpty: Boolean
  - True, if the collection is not empty

The OCL-Type OclAny is the most general OCL-Type

OCL-Operations for OCL-Collections (2)

- union(c1:Collection)
  - Union with collection c1
- intersection(c2:Collection)
  - Intersection with Collection c2 (contains only elements, which appear in the collection as well as in collection c2)
- including(o:OclAny)
  - Collection containing all elements of the Collection and element o
- select(expr:OclExpression)
  - Subset of all elements of the collection, for which the OCL-expression expr is true
How do we get OCL-Collections?

- A collection can be generated by explicitly enumerating the elements
- A collection can be generated by navigating along one or more 1:N associations
  - Navigation along a single 1:n association yields a Set
  - Navigation along a couple of 1:n associations yields a Bag (Multiset)
  - Navigation along a single 1:n association labeled with the constraint \{ordered\} yields a Sequence

Navigation through a 1:n-Association

Example: A Customer should not have more than 4 cards

\[
\text{context Customer inv:}
\begin{align*}
\text{card->size} & \leq 4 \\
\end{align*}
\]

**card** denotes a set of **customercards**

Alternative writing style

\[
\text{Customer card->size \leq 4}
\]

Navigation through several 1:n-Associations

Example:

\[
\text{programPartner nrcustomer = bonusprogram.customer->size}
\]

**bonusprogram** denotes a set of **Bonussprograms**

Conversion between OCL-Collections

- OCL offers operations to convert OCL-Collections:
  - **asSet** transforms a multiset or sequence into a set
  - **asBag** transforms a set or sequence into a multiset
  - **asSequence** transforms a set or multiset into a sequence.

Example of a Conversion

\[
\text{programPartner nrcustomer = bonusprogram.customer->asSet->size}
\]

This expression may contain customer multiple times, we can get the number of unique customers as follows:
Operations on OCL-Type Sequence

- **first**: T
  - The first element of a sequence
- **last**: T
  - The last element of a sequence
- **at(index:Integer): T**
  - The element with index `index` in the sequence

Example: „The first Level, you can reach in the bonusprogram has the name "Silber"." 

OCL-Invariant:

```
Bonusprogram::
  level->first.name = "Silber"
```

Specifying the Model Constraints: Using `asSet`

- **Local attribute navigation**
  - `context Tournament inv:`
  - `end - start <= Calendar.WEEK`
- **Directly related class navigation**
  - `context Tournament::acceptPlayer(p)`
    - `pre:` `league.players->includes(p)`
- **Indirectly related class navigation**
  - `context League::getActivePlayers`
    - `post:` `result=tournaments.players->asSet`

Evaluating OCL Expressions

The value of an OCL expression is an object or a collection of objects.

- **Multiplicity of the association-end is 1**
  - The value of the OCL expression is a single object
- **Multiplicity is 0..1**
  - The result is an empty set if there is no object, otherwise a single object
- **Multiplicity of the association-end is * 1-Many**
  - The result is a collection of objects
  - By default, the navigation result is a Set
  - When the association is {ordered}, the navigation results in a Sequence
  - Multiple "1-Many" associations result in a Bag

OCL supports Quantification

- **OCL forall quantifier**
  - /* All Matches in a Tournament occur within the Tournament’s time frame */
    - `context Tournament inv:`
      - `matches->forAll(m:Match | m.start.after(t.start) and m.end.before(t.end))`
- **OCL exists quantifier**
  - /* Each Tournament conducts at least one Match on the first day of the Tournament */
    - `context Tournament inv:`
      - `matches->exists(m:Match | m.start.equals(start))`

Specifying invariants on Tournament and Tournament Control

- **English: “All Matches of in a Tournament must occur within the time frame of the Tournament”**
  - `context Tournament inv:`
    - `matches->forAll(m| m.start.after(start) and m.start.before(end))`
- **English: “No Player can take part in two or more Tournaments that overlap”**
  - `context TournamentControl inv:`
    - `tournament.players->forAll(p| p.tournaments->forAll(t| t <> tournament implies not t.overlap(tournament)))`
Specifying invariants on Match

English: “A match can only involve players who are accepted in the tournament”

\[
\begin{align*}
\text{context Match inv:} & \quad \forall p. \forall t. \forall m. (p \text{.tournaments} \rightarrow \exists t. (t \text{.matches} \rightarrow \exists m. (m \text{.includes(self))})) \\
\text{context Match inv:} & \quad \forall p. \forall t. \forall m. (p \text{.tournaments} \rightarrow \exists t. (t \text{.matches} \rightarrow \exists m. (m \text{.includes(self))})) \\
\end{align*}
\]

Additional Readings

- J.B. Warmer, A.G. Kleppe
- B. Meyer
- B. Meyer,
- C. A. R. Hoare,

Summary

- Constraints are predicates (often boolean expressions) on UML model elements
- Contracts are constraints on a class that enable class users, implementors and extenders to share the same assumption about the class (“Design by contract”)
- OCL is the example of a formal language that allows us to express constraints on UML models
- Complicated constrains involving more than one class, attribute or operation can be expressed with 3 basic navigation types.