XI. The Object Constraint Language

The Object Constraint Language (OCL)

Examples

Invariants

Set-Theoretic Constraints

Pre-/Post-Conditions

Acknowledgment: these slides are based on Prof. John Mylopoulos slides which are used to teach a similar course in the University of Toronto – St. George campus. Used with Permission.

The Object Constraint Language

- Some constraints can be adequately expressed in the graphical language (e.g., multiplicity of an association).
- Some can not. For example, constraints within operation specifications (pre- and post-conditions)
- The Object Constraint Language (OCL) provides a formal language for specifying constraints which can supplement the models created in terms of UML diagrams.
- The language has a precise syntax that enables the construction of unambiguous statements.
- Each expression has an associated ***context***, which is usually the class to which the expression is attached.

OCL Examples

OCL expression	Interpretation
Person self.age	In the context of a specific person, the value of the property 'age' of that person—i.e. a person's age.
Person self.income >= 5,000	The property 'income' of the person under consideration must be greater than or equal to 5,000.
Person self.wife->notEmpty **implies** self.wife.sex = female	If the set 'wife' associated with a person is not empty, then the value of the property 'sex' of the wife must be female. The boldface denotes an OCL keyword, but has no semantic import in itself.
Company self.employee->size <= 50	The size of the set of the property 'employee' of a company must be less than or equal to 50. That is, a company cannot have more than 50 employees.
Company self.employee->select (age > 50)	This specifies the set of employees of a company whose age is greater than 50.

Invariants

- ***Invariants*** can be associated with classes and describe properties that must hold true for all the instances of the class.
- For example, for an LCBO store with a customer database, represented by a Customer class

  Customer

  ---

  age ≥ 18

  says that every customer must have an age attribute value greater than 18.
- For a CustomerCard class, the invariants

  CustomerCard

  ---

  validFrom.isBefore(today)

  expiresAt.isAfter(today)

  make sure that the card is valid at the time of use.

More on Invariants

- Instead of writing

  validFrom.isBefore(today)

  we can write

  validFrom --> isBefore(today)

  isBefore is a binary operation associated with dates.
- Sometimes the value of one attribute can be computed from those of others (***derived attribute***):

  Customer

  printedName = firstName.concat(lastName)

Invariants Between Classes

- We can also specify invariants between the instances of two or more classes.
- For example, the Customer class may have an invariant

  card.customer = customer

  We assume here that card is an attribute of Customer and customer is an attribute of CustomerCard, and we want to make sure that the values of these attributes match.
- Likewise, for the CustomerCard class we may have an invariant

printedName = customer.title.concat(customer.name)

which states that the value of printedName of CustomerCard should be the same with the concatenation of customer.name and customer.title.

Set-Theoretic Constraints

- Attributes are single-valued in UML, but associations are not (unless their multiplicity specifies so.) We want to define constraints on sets of objects too.
- For example, if we have a class GoodCustomer which a specialization of Customer, and Customer has an association bought with an attribute amount,then we may want a constraint

  bought.amount --> sum ≤ $5000

  which says that the sum of all products bought by a good customer is greater than $5K.
- One-product customers have the constraint

  bought --> size = 1 (or, bought.size = 1)

Set-Theoretic Functions and Predicates

size(set) - returns the size (cardinality) of the set
sum(set) - returns the sum of the set (assumed to contain numbers)
average(set) - returns the average of the set
min(set) - returns the minimum of the set
max(set) - returns the maximum of the set
notEmpty(set) - true if the set is not empty
includes(object) - true if the set includes the object
union(set) - returns the union of two sets
intersection(set) - returns the intersection of two sets

Pre- and Post-conditions in OCL

- Pre-condition and post-condition expressions are associated to an operation/method and they describe
  - ✓ What must be true before the operation is executed (pre-condition);
  - ✓ What will be true once the operation is executed (post-condition).
- For example, we may want to say:
  - ✓ Customer::buy(product)

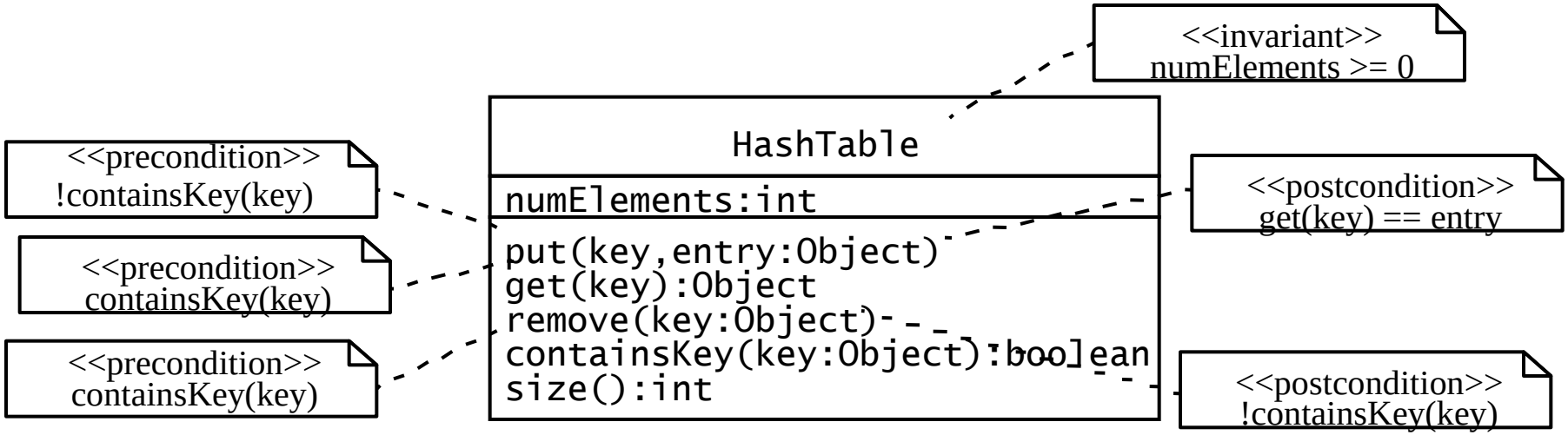
    pre: acctBal-product.price > 0

    post: acctBal = acctBal@pre - product.price

The value of acctBal ***before*** the operation

Expressing Constraints in UML Models

- A constraint is depicted as a note attached to the constrained UML element by a dependency relationship.



Additional Readings

- [Warmer99] Warmer, J. Kleppe, A. *The Object Constraint Language: Precise Modeling with UML,* Addison-Wesley 1999.
- http://dec.bournemouth.ac.uk/dec_ind/swebster/UML_OCL/index.htm