ΗΥ351:
Ανάλυση και Σχεδίαση Πληροφοριακών Συστημάτων
Information Systems Analysis and Design

OCL: Object Constraint Language (cont)

Γιάννης Τζίτζικας

Διάλεξη : 14
Ημερομηνία :

Αρχικά και Παραγόμενα Γνωρίσματα
Initial and **Derived** Attributes

An OCL expression may be used to indicate the initial or derived value of an attribute or association end.

```
context Typename::attributeName: Type
init: -- some expression representing the initial value

context Typename::assocRoleName: Type
derive: -- some expression representing the derivation rule
```

The expression must conform to the result type of the attribute.

If the context is an association end the expression must conform to the classifier at that end when the multiplicity is at most one, or Set or OrderedSet when the multiplicity may be more than one. Initial, and derivation expressions may be mixed together after one context.

```
Context Person::income: Integer
init: parents.income->sum()*1%            -- pocket allowance
derive: if underAge
        then  parents.income->sum()*1% -- pocket allowance
        else  job.salary               -- income from regular job
       endif
```

Let Expressions

Sometimes a sub-expression is used more than once in a constraint.
The **let expression** allows one to define a variable which can be used in the constraint.

context Person **inv**:
let income : Integer = self.job.salary->sum() **in**
if isUnemployed **then**
    income < 100
else
    income >= 100
endif

A let expression may be included in any kind of OCL expression. It is only known within this specific expression.

«definition» expressions

The Let expression allows a variable to be used in one OCL expression.
To enable reuse of variables/operations over multiple OCL expressions we can use the stereotype «definition».

All variables and operations defined in the «definition» constraint are known in the same context as where any property of the Classifier can be used.

The syntax of the attribute or operation definitions is similar to the Let expression, but each attribute and operation definition is prefixed with the keyword 'def'.

context Person
def: income : Integer = self.job.salary->sum()
def: nickname : String = 'Little Red Rooster'
def: hasTitle(t : String) : Boolean = self.job->exists(title = t)

The names of the attributes / operations in a let expression may not conflict with the names of respective attributes/ associationEnds and operations of the Classifier.

Re-typing or casting

In some circumstances, it is desirable to use a property of an object that is defined on a subtype of the current known type of the object. Because the property is not defined on the current known type, this results in a type conformance error.

When it is certain that the actual type of the object is the subtype, the object can be re-typed using the operation **oclAsType(OclType)**. This operation results in the same object, but the known type is the argument OclType.

When there is an object obj of type Type1 and Type2 is a subtype of Type1, then it is allowed to write:

obj1.oclAsType(Type2) --- evaluates to object with type Type2

Accessing overridden properties of supertypes

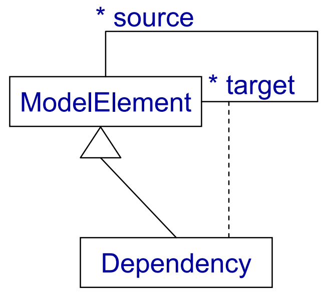
Whenever properties are redefined within a type, the properties of the supertypes can be accessed using the oclAsType() operation.

Whenever we have a class B as a subtype of class A, and a property p1 of both A and B, we can write:

context B inv:

self.oclAsType(A).p1 -- accesses the p1 property defined in A

self.p1 -- accesses the p1 property defined in B



In this model fragment there is an ambiguity with the OCL expression on Dependency:

context Dependency inv:

self.source <> self

This can either mean normal association navigation, which is inherited from ModelElement, or it might also mean navigation through the dotted line as an association class. Both possible navigations use the same role-name, so this is always ambiguous. Using oclAsType() we can distinguish between them with:

context Dependency

inv: self.oclAsType(Dependency).source->isEmpty()

inv: self.oclAsType(ModelElement).source->isEmpty()

Predefined properties on all objects

There are several properties that apply to all objects, and are predefined in OCL.

- **oclIsTypeOf (t: OclType) : Boolean**
  - returns true if the type of **self** and **t** are the same,e.g.

    context Person
    inv: self.oclIsTypeOf(Person **)** -- is true
    inv: self.oclIsTypeOf(Company**)** -- is false
- **oclIsKindOf (t: OclType) : Boolean**
  - The oclsIsTypeOf deals with the direct type of an object. The oclIsKindOf property determines whether t is either the direct type or one of the supertypes of an object.
- **oclInState (s: OclState) : Boolean**
  - will be discussed later on
- **oclIsNew () : Boolean**
  - It returns true if, used in a postcondition, the object is created during performing the operation. i.e., it didn't exist at precondition time.
- **oclAsType (t : OclType) : instance of OclType**
  - *we have discussed this already*

.allInstances()

All properties discussed until now in OCL are properties on instances of classes. The types are either predefined in OCL or defined in the class model. In OCL, it is also possible to use features defined on the types/classes themselves. These are, for example, the class-scoped features defined in the class model. Furthermore, several features are predefined on each type.

A predefined feature on classes, interfaces and enumerations is allInstances, which results in the Set of all instances of the type in existence at the specific time when the expression is evaluated.

Example

We want to make sure that all instances of Person have unique names:

context Person
inv: Person.**allInstances()->forAll(**p1, p2 | p1 <> p2 **implies** p1.name <> p2.name**)**

The Person.allInstances() is the set of all persons that exist in the system at the time that the expression is evaluated and is of type Set(Person).

Type conformance rules:

- Type1 conforms to Type2 when they are identical or when Type1 is a subtype of Type2 (standard rule for all types).
- Collection(Type1) conforms to Collection(Type2), when Type1 conforms to Type2. This is also true for Set(Type1)/ Set(Type2), Sequence(Type1)/ Sequence(Type2), Bag(Type1)/Bag(Type2)
- The types Set (X), Bag (X) and Sequence (X) are all subtypes of Collection (X).

Type conformance is transitive: if Type1 conforms to Type2, and Type2 conforms to Type3, then Type1 conforms to Type3 (standard rule for all types).

For example, if Bicycle and Car are two separate subtypes of Transport:

Set(Bicycle) conforms to Set(Transport)
Set(Bicycle) conforms to Collection(Bicycle)
Set(Bicycle) conforms to Collection(Transport)

However

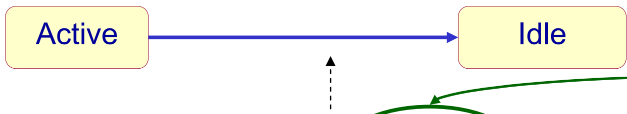
Set(Bicycle) does not conform to Bag(Bicycle), nor the other way around.

They are both subtypes of Collection(Bicycle) at the same level in the hierarchy.

Χρήση εκφράσεων OCL στα μοντέλα UML (πέραν των διαγραμμάτων κλάσεων)

Use of OCL expressions in UML models (apart from class diagrams)

OCL και Διαγράμματα Καταστάσεων
OCL and State Diagrams



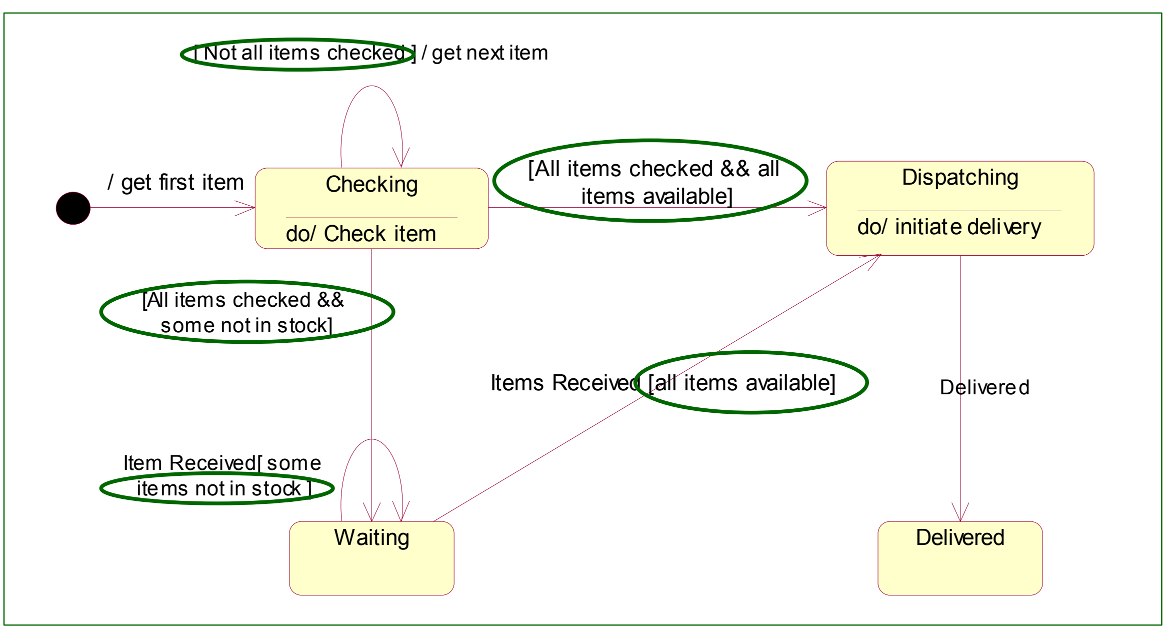
Transition labels: **Event[Condition]/Action**
- all three are optional

- Event:
  - if nil then when the task is completed we continue
- Condition
  - logical condition (transition occurs if its value is True)
  - the guards of transitions from a state must be mutually exclusive so that to have a unique next state
- Action
  - processes that occur quickly and are not interruptible

OCL expression

An OCL expression acting as value of a guard is of type Boolean.
The expression is evaluated at the moment that the transition attached to the guard is attempted

OCL and State Diagrams (II)



OCL and State Diagrams (III)

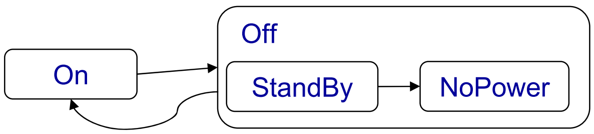
Predefined properties on all objects

oclInState (s : OclState) : Boolean

oclInState (s : OclState) : Boolean

This operation returns true if the object is in the state s.

Values for s are the names of the states in the statemachine(s) attached to the Classifier of object. For nested states the statenames can be combined using the double colon.



Here the values for s can be

- On
- Off
- Off::Standby
- Off::NoPower.

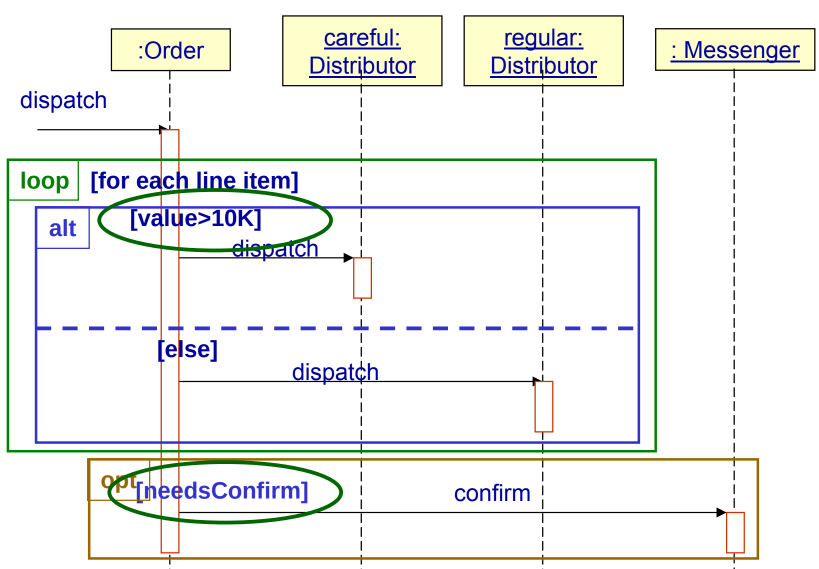
If the classifier of object has the above associated statemachine valid OCL expressions are:

- object.oclInState(On)
- object.oclInState(Off)
- object.oclInstate(Off::Standby)
- object.oclInState(Off::NoPower)

If there are multiple statemachines attached to the object's classifier, then the statename can be prefixed with the name of the statemachine containing the state and the double colon '::', as with nested states.

OCL και Διαγράμματα Αλληλεπίδρασης
OCL and Interaction Diagrams

OCL can be used for expressing the conditions under which a message (in a sequence or communication diagram) is sent



Ποια εργαλεία CASE υποστηρίζουν την OCL και πως; Which UML CASE Tools support OCL and how?

- We can attach OCL constraints to our diagrams using an appropriate stereotype and a dashed line should connect it to its contextual element
- OCL constraints are exchanged using XMI
- Tools that support OCL
  - ArgoUML allows expressing them
  - OCL Evaluator (a tool for editing, syntax checking & evaluating OCL)
  - Octopus OCL 2.0 Plug-in for Eclipse
  - Enterprise Architect
    - Allows writing OCL expressions, however they are not actually used to enrich the model (with constraints that cannot be expressed in the diagrams), but for model checking. The parser also seems to allow expressions that are not correct. Overall, OCL support is currently very limited in EA.

Assertions and Programming Languages

Assertions and Programming Languages

- Assertion techniques (preconditions, postconditions, invariants)
- History of assertion techniques:
  - Hoare 1972
  - Meyer 97a (he proposed the idea Design by Contract)
- Assertions support in Programming Language:
  - Eiffel supports them
  - In Java it is also possible (e.g. using JAF, standard from J2SE 1.4)

Techniques for adding Assertion Support in a PL

- Built in
  - Syntactic correctness of assertions is checked by the compiler
  - The runtime environment performs the runtime assertion checks
- Preprocessing
  - Formulate assertions separate from the program or include the assertions as comments. A preprocessor translates the assertions to program code
  - Pros : separation (separation of programmatic logic from contracts)
  - Con: the original program code is modified (e.g. the line numbers of compiler errors do not fit the line numbers of the program)
- Metaprogramming
  - Traditionally this is possible only in dynamically typed and interpreted languages
  - Programs that have the possibility to reason about themselves have so called **reflective capabilities** **(**Java has a reflection API)
  - The main advantage of metaprogramming approaches is that no specialized preprocessor has to be used but the native compiler. Nevertheless a specialized runtime environment has to be used to enable assertion checking

Assertions and Java

- "An *assertion* is a statement containing a boolean expression that the programmer believes to be true at the time the statement is executed".
- It is a facility provided within the java programming language to test the correctness or assumptions made by your program. Assertions are checks provided within the system to ensure the smooth running of the program.
- ***Why Assertions?***
- *Why we need another level of checking when exceptions can do the job?*
- Exceptions are primarily used to handle unusual (abnormal) conditions arising during program execution.
  - They do not guarantee smooth or correct execution of the program.
- Assertions are used to specify conditions that a programmer assumes are true.
  - If a programmer can swear that the value being passed into a particular method is positive no matter what a calling client passes, it can be documented using an assertion to state it. Assertions help state scenarios that ensure the program is running smoothly. Assertions can be efficient tools to ensure correct execution of a program. They also improve the confidence about the program.
  - We can turn them off

Assertions and Java

Syntax `assert` *expression1*;

The expression is the one we wish to assert as true.
If the assumption fails, the expression evaluates to be false which means the assertion failed. In case the expression succeeds the program continues normally.

When an assertion fails the program throws an AssertionError on to the stack trace.

Examples:

```
assert i<0;
assert (!myString.equals(""));
```

Java Assertion Facility (JAF) (builtin since Java 1.4)

Assertions and Java

Syntax `assert expression1 : expression2;`

The first argument takes a Boolean expression, while the second expression would be the resulting action to be taken if the assertion fails. The *Expression2* should be a value and can also be a result of executing a function. The compiler would throw an error if the second expression returns a void value.

When an assertion fails the program throws an AssertionError on to the stack trace. The program creates an object AssertionError with the return type of *Expression2.* The overloaded AssertionError constructor would then convert the returned data type into String and dump it on the stack trace with a meaningful message.

Examples:

```
assert age>0 : "The value of age cannot be negative" +age;
assert ((i/2*23-12)>0):checkArgumentValue();
assert isParameterValid():throw IllegalParameterError();
```

In the second example the method checkArgumentValue() must return a value

Assertions and Java

- for the javac compiler to accept code containing assertions, you must use the -source 1.4 command-line option
  - javac -source 1.4 MyClass.java
- By default, assertions are disabled at runtime
  - enable assertions at runtime:
    - -enableassertions or –ea
  - disable assertions at runtime:
    - -disableassertions or –da

OCL Constraints and Java

Context Account:withdraw(amount:Real)
pre: amount <= balance
post: balance = balance@pre - amount

Context Account:getBalance():Real
post: result = balance

```
class Account {
  private float balance = 0 ;
  public void withdraw(float amount){
      assert amount<= balance;
      balance = balance - amount;
  }

 public float getBalance(){
      return balance;
}
```


OCL Constraints in Java (2)

Context Employee:SetAge (age)
pre: age > 0

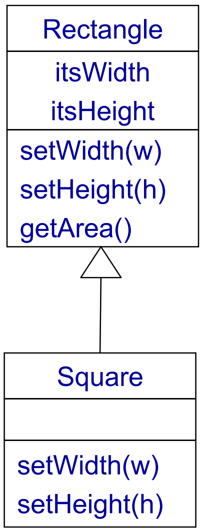
```
class Employee {
  public void SetAge(int age){
      assert age>0;
      this.age = age;
  }
}
```

Design by contract

```
class Employee {
  public void SetAge(int age) throws ArgumentException {
      if (age<=0) {
        throw new ArgumentException(“negative age”);
      }
     this.age = age;
  }
}
```

Defensive Programming
(throwing exceptions)

Example: Rectangle and Square




```
class Rectangle {
public:
   virtual void setWidth(double w) {itsWidth=w;}
   virtual void setHeight(double h){itsHeight=h;}
   double getArea() {return itsHeight * itsWidth;}
private:
  double itsWidth;
  double itsHeight;
};
```

```
class Square: public Rectangle {
public:
   virtual void setWidth(double w);
   virtual void setHeight(double h);
};
```

```
void Square::setWidth(double w)
  {Rectangle::setWidth(w);  Rectangle::setHeight(w); };
void Square::setHeight(double h)
   {Rectangle::setWidth(h);  Rectangle::setHeight(h); };
```


Example: Rectangle and Square (II)

```
class Rectangle {
public:
   virtual void setWidth(double w) {itsWidth=w;}
   virtual void setHeight(double h){itsHeight=h;}
   double getArea() {return itsHeight * itsWidth;}
private:
  double itsWidth;
  double itsHeight;
};
```

```
class Square: public Rectangle {
public:
   virtual void setWidth(double w);
   virtual void setHeight(double h);
};
```

```
void Square::setWidth(double w)
  {Rectangle::setWidth(w);  Rectangle::setHeight(w); };
void Square::setHeight(double h)
   {Rectangle::setWidth(h);  Rectangle::setHeight(h); };
```

```
void g(Rectangle* r)
{
    r->setWidth(5);
    r->setHeight(4);
   assert(r->getArea()==20);
}
```

It will function correctly if **r** is a rectangle.

It will not function correctly if **r** is a square

The class Square actually violates an "invariant" of the class Rectangle,

specifically the width-height independence.

Example: Rectangle and Square (III)

This could be expressed in the class Rectagle using OCL by
a post condition of setWidth: *i.e. the height is the old value of height;*
and a post condition of setHeight *i.e. the width is the old value of width.*

Context Rectangle:setWidth(w)
post: itsWidth = w and
itsHeight = itsHeight@pre

Context Rectangle:setHeight(w)
post: itsHeight = h and
itsWidth = itsWidth@pre

[Meyers]:
When we override a method A with a method B
the precondition of B should be that of A or a weaker condition, and
the postcondition of B should be that of A or a stronger (more strict) condition.

This reveals the problem in our example: the postcondition of Square:setWidth is weaker (although it should be stronger according to the above rule).
So, if for example we had copied the postconditions of the Rectangle's methods to the methods of Square, we would have seen the problem while testing the class Square .

Where things go

- At the beginning each application was dependent on the hardware of the machine (machine code programming)
- With compilers (i.e. programming languages) each application can be compiled for different machines and OSs assuming there is a compiler for them (one compiler is needed, thousands of applications exploit it)
- With Java and bytecodes even the compilation is somehow "bypassed"
- With UML and OCL the specification of an application can be independent even from the PL

Χρήσιμες αναφορές

- Που μπορείτε να βρείτε την προδιαγραφή (specification) της UML 2.0 OCL
  - http://www.omg.org/cgi-bin/doc?ptc/2005-06-06
  - http://www.omg.org/technology/documents/modeling_spec_catalog.htm#UML
- Εργαλεία
  - OCL Evaluator (a tool for editing, syntax checking & evaluating OCL)
  - Octopus OCL 2.0 Plug-in for Eclipse
- Μια διαδικτυακή πύλη για την OCL
  - http://www-st.inf.tu-dresden.de/ocl/
- Ένα ενδιαφέρον άρθρο
  - J. Warmer and A. Kleppe, “The Object Constraint Language: Precise Modeling with UML”, Addison-Wesley 1999.
- Άλλες πηγές
  - http://en.wikipedia.org/wiki/Object_Constraint_Language
  - http://www.omg.org/docs/ad/99-12-05.pdf
  - http://www.brucker.ch/projects/hol-ocl/
  - http://www.eclipse.org/articles/Article-EMF-Codegen-with-OCL/article.html (for EMF)