Abstract: This paper intends to introduce the Object constraint language and where this language is applicable. It also gives a brief introduction to the formal language Z which has been widely used in the industry, and tries to compare both the languages. Finally, the tools used in evaluating the OCL expressions are discussed in brief.

Introduction:

As the name suggests, OCL is a language used to specify the constraints of the objects present in the model. The Object Constraint Language (OCL) is a part of the specification of the Unified Modeling language, UML. Though it’s a part of the specification, it’s an extension of the UML. It overcomes the drawbacks, where UML has failed to deliver.

The History of OCL dates back to the days of syntropy method, it was developed at the IBM insurance division by Jos Warmer as a formal language which is easy to be read. OCL is the contribution of IBM towards building up of the UML standard.

Unlike, other formal languages which are mathematical in nature, OCL remains easy to read, write and understand. It is pure expression language and hence does not have any side effects on the model.

This paper discusses the OCL version 1.1 which has few setbacks, but these setbacks have been overcome in the latter versions. The latest OCL version is the 1.4 and a draft of OCL 2.0 included in the UML2.0 is being reviewed at present.

Usage of OCL.

The graphical model representation is not enough for a precise and unambiguous specification. This can be verified by the example given in fig 1. It does not specify the initial values of the attributes and the conditions which are to be fulfilled in order to perform operations in the classes. OCL helps in reducing these ambiguities.

OCL is used for different purposes:

1. To describe the invariants on classes and types in the class model.
2. To specify the pre and post conditions on operations, methods and attributes.
3. To describe the guards on the state transitions.
4. As a navigation tool.
5. To prescribe constraints on operations.

This paper considers the diagram in Figure 1 for the examples in this paper and this example in from [1].
Types of Constraints and their relation with UML.

Different types of constraints exist as mentioned above. The definitions for each of the constraints are given below.

There are four types of constraints:

1. An **invariant** is a constraint that states a condition that must always be met by all instances of the class, type, or interface. An invariant is described using an expression that evaluates to true if the invariant is met. Invariants must be true all the time.

2. A **precondition** to an operation is a restriction that must be true at the moment that the operation is going to be executed. The obligations are specified by postconditions.

3. A **postcondition** to an operation is a restriction that must be true at the moment that the operation has just ended its execution.

4. A **guard** is a constraint that must be true before a state transition fires.
Before explaining the syntax of these constraints, it would be good to get acquainted with the syntax of the OCL.

Each constraint is linked to one item from the model. This item is called the context of the constraint. A special keyword self refers to the object that is the context of the constraint. If the constraint is an invariant the context is a class. If the constraint is a pre- or postcondition the context is an operation of a class. If the constraint is a guard then the context is the state from which the transition fires. Note that in all cases, the keyword self refers to the instance of that class for which the constraint is being evaluated, e.g. the class that defines the operation, or the class for which the state chart is specified.

The item for which the constraint is being specified is underlined and the property or attribute is specified by a colon next to the item. The constraints are specified in the next line.

Example:  Company
          self.numberofemployees

Types in OCL.

The version 1.1 has a standard library which is mentioned below.

OCL Standard Library:
– Basic types: Integer, String, Real, Boolean.
– Collection types: Set, Bag, Sequence.
Model classes
   – Classes, interfaces, datatypes from the UML diagrams.
The latter versions boasts of
Tuple types
   – To support random queries.
The basic data types are similar to the data types of the programming languages and we will not include them in detail.

OCL defines a number of operations on the predefined datatypes
They are as follows

<table>
<thead>
<tr>
<th>Types</th>
<th>Operations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Integers</td>
<td>* , / , + , - , abs</td>
</tr>
<tr>
<td>Real</td>
<td>* , / , + , - , floor</td>
</tr>
<tr>
<td>Boolean</td>
<td>and , or , xor , not , implies , if-then-else</td>
</tr>
<tr>
<td>String</td>
<td>toUpper , concat</td>
</tr>
</tbody>
</table>
Collections

Apart from these basic data types, we have other data types called as collections which are an essential part of the OCL expressions.

The type Collection is predefined in OCL. Collection is an abstract type, with the concrete collection types as its subtypes. It has three different collection types namely:

Set: A Set is the mathematical set. It does not contain duplicate elements.
Ex: Company
    self.employee.
    This results in a set of employees working in a company.
Ex: Set = \{ 1, 5, 3, 8, 2, 0 \}

Bag: A bag is similar to a set but contains duplicate elements.
Ex: Bag = \{ 1, 5, 5, 5, 3, 8, 2, 2, 0 \}

Sequence: A sequence is similar to a bag and contains elements in an ordered set.
Ex: Sequence = \{ 0, 1, 2, 2, 3, 5, 5, 5, 8 \}

Collections and its operations.

The different types of operations available on collections are as follows:
1. select(…)
2. reject(…)
3. collect(…)
4. exists(…)
5. forAll(…)
6. iterate(…)
7. union(…)
8. size()
9. includes(…)
10. excludes(…)
11. including(…)
12. excluding(…)
13. isEmpty()
14. count(…)

The operations 1 to 7 result in a collection of data values and the rest result in a Boolean value or an integer value. The complete syntax for each of the operation looks like one of:

collection->operation( v : Type | boolean-expression-with-v )
collection->operation ( v | boolean-expression-with-v )
collection->operation ( boolean-expression )

The operations are explained below.
1. Select: selection from a specific collection
   Ex: company
       self.employee ->select(age>50)

2. Reject: The negation of selection
   Ex: company
       self.employee ->reject(age>50)

3. Collect: specify a collection derived from other collection.
4. **ForAll** results in Boolean value if all the elements of the collection satisfy the expression.
   Ex: `company`  
   `self.employee->forAll( forename = 'Jack' )`

5. **Exists** results in Boolean value if at least one element of the collection satisfies the constraint.
   Ex: `company`  
   `self.employee->exists( forename = 'Jack' )`

6. **Iterate** : all the above mentioned operations can be evaluated using this operation. Though it is a bit tough to operate using this operation.
   Ex:  
   `collection->collect(x : T | x.property)`
   -- is identical to:  
   `collection->iterate(x : T; acc : T2 = Bag{} | acc->including(x.property))`

7. The operations union, size, count, includes and others are similar in syntax.

**Predefined Types**

In addition to the basic data types such as integer, real OCL has three other predefined data types. They are OclType, OclExpression and OclAny.

**OclType:**
This is an instance of the Ocl type called the OCLtype. It is useful to modelers designing the meta model of the system because it gives access to the data types being used in the system. The types of each of the data types can be identified using this type. The instance of this type is called *type*.

Ex:  
   `type.name= String`
   Returns the type of the item in context.

Many other features are added to this type and they can be referenced in [1].

**OclExpression:**
This type identifies the features and semantics of the expressions used in the Ocl. Every expression is treated as an item in OCL and the instance of the item is called an expression.

Ex:  
   `expression.evaluationType : OclType`
   The result is type of the object that results from evaluating *expression*.

Many other features exist in this type and they can be referenced in [1].
OclAny:

It’s a supertype of the ocl type. Classes can be represented in the OclAny. The instance of this type is called an Object.

So far, we have discussed the datatypes and the syntax of this Language. Now we see how this data types and syntax can be applied to the constraints applicable to a part or whole of the object oriented modeling.

As there are four constraints discussed in the section Types of Constraints, each constrain is discussed in a brief manner.

**Invariants:**

Contextual instance of an OCL expression which is part of an Invariant is written as follows:

Every person’s age should be greater than zero.

**Person inv:**

```
self.age>0
```

**Company inv:**

```
self.stockPrice()>0
```

**Post and Pre-conditions.**

The syntax for pre or post condition on an Operation or method.

```
Typename:operationName(parameters):Returntype
Pre : parameter>.....
Post: result=
```

In a post condition, the expression can refer to two sets of values for a property of an object.

1. The value of the property of an object before the operation.
2. The value of the property of the object after the operation

The value before the operation is denoted by the property followed by “@pre”.

**Person**: birthdayHappens()

```
post: age=age@pre+1
```

**Guards.**

The guards are used to fire when state transition takes place. The result is a Boolean expression.
So far, we have discussed the Object constraint language which can be used to express the constraints in an object and clear the ambiguities in the system. In order to appreciate the system, we discuss briefly about a formal language called Z(“Zed”). Thus giving the advantages of Ocl over Z.

**The Formal Language Z**

A language is formal if its syntax and semantics are defined formally (mathematically).

Formal languages allow for the design, the development, the verification, the testing and the maintenance of a system:
- remove ambiguities and introduce precision,
- structure information at an appropriate abstraction level,
- support the verification of design properties,
- are supported by tools and systems.

Z was developed at the Oxford University Computer Science Dept in 1989. It is a very expressive formal language based on first-order logic with equality (PL1=) and typed set-theory. It supports the structured modeling of a system, both static and dynamic:
- modeling/specification of data of the system,
- functional description of the system (state transitions).

Z consists of two sub-languages:
- mathematical language, which allows us to model design aspects, objects and their relations.
- schema language, which allows us structure, compose and split modelings (data, functions and predicates).

- Z-schemata
  - can be used as declarations, as types, and as predicates,
  - can model the state space (states and state transitions) of a system,
  - can be used to verify formal modelings.

It is purely a mathematical language and is used extensively by developers and is supported by tools and systems.

Ex: Referred from [3].

- **Vertical syntax:**

```
<table>
<thead>
<tr>
<th>name</th>
<th>declaration of typed variables</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>(represent observations of the state)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>relationships between values of vars</th>
</tr>
</thead>
<tbody>
<tr>
<td>(invariants of the system)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SchemaOne</th>
</tr>
</thead>
<tbody>
<tr>
<td>a : Z</td>
</tr>
<tr>
<td>c : P Z</td>
</tr>
<tr>
<td>a ∈ c</td>
</tr>
<tr>
<td>c ≠ Ø</td>
</tr>
</tbody>
</table>
```
Advantages of OCL over Z.

The major advantages of OCL could be attributed to the user-friendly approach it has over the formal language Z. The advantages are as follows:

1. Easy to read than a mathematical language.

2. Understood by all the people involved in the software Development. (customers, developers, stakeholders and all.).

3. The Z language does not support the Boolean data type which is supported by Ocl, it overcomes this by enumeration data type.

Disadvantages of OCL over Z.

1. The major disadvantage is that the Z code can be easily converted into program code because it is similar to programming code as it is mathematically oriented.

2. Supported by many tools and systems.

3. Implementation issues are not the focus of Ocl but larger systems need to give heed to the implementation because it would not be feasible to back off from a point in the middle of the development and start fresh.

Tools used for OCL.

Ocl is not a programming language, so we don’t expect programmable code from this language. Instead tools are required to check the syntax of the language to make it clear and easy in the modeling of the system. The following are some of the tools listed below to check the syntax of the language:

1. At the Dresden University of Technology an OCL parser, written in Java, is available.

2. The OCL 1.4 syntax checker is a tool from Klasse Objecten that performs a syntax check only.

Though there are many parsers available to check the syntax of the OCL language, these two would suffice the needs of a normal programmer and their simple interface would be helpful.
Conclusions

OCL is here to stay, being backed by UML - the industry wide standard for object oriented modeling. Though there are some setbacks, the future versions will come up with the best possible solutions for this language and increase the productivity of the language.

References.


2. On Formalizing the UML Object Constraint Language-Mark Richters and Martin Gogolla
   University of Bremen, Germany

3. Formal Modeling with Z: An Introduction
   http://www.informatik.uni-reiburg.de/~softech/teaching/ss02/st/

4. The company startup from the creator of the OCL, Jos Warmer
   http://www.klasse.nl/ocl/ocl-introduction-text.html