CHAPTER 03

USING OBJECT CONSTRAINT LANGUAGE FOR MODEL DRIVEN ARCHITECTURE

3.1 INTRODUCTION

The development of software models are utilizing the OCL, hence regarded as a modeling language. OCL [64, 65] when used along with UML supports a lot more additional features. OMG is a standard for analyzing and designing in object oriented manner. Each expression which is represented in OCL is based on the types (whether classes or interfaces or any other) which are explained in the UML diagrams. Therefore UML will definitely be used in the OCL based applications.

Global information can be added to the object oriented models via the OCL expressions along with some object modeling features. It may not be possible to represent this information diagrammatically. In UML 1.1, it was assumed that this information can also be up to defining constraints, where a constraint refers to the restrictions imposed on some values of object oriented model whereas UML 2 enables to add a lot
more information in the model rather than just constraints. The additional information that can be presented in these expressions can be query definitions, references to the values, mentioning conditions and business rules into the model. OCL can be considered as a standard language to represent precise and unambiguous expressions.

A strong relation exists among the OMG standards. Models serve as building block for the development of software, a significant feature of MDA. Thus it is necessary that we have consistent models. A combined effort of OCL and UML produces consistent models.

Thus, we can have an efficient and faster implementation of consistent model developed in UML and OCL.

### 3.2 CHARACTERISTICS OF OCL

A model has more than one class, required OCL statements to be a consistent model. With the help of UML diagrams many inconsistencies would persist. Therefore, OCL is a necessary language for developing efficient models.

#### 3.2.1 Query and Constraint Language Description

Earlier, OCL was assumed to be just limited to constraint implementation where constraints define the conditions when system data values and objects would be valid.
As in example, Fig. 3-3, where there could be any number of passengers in a flight, but on applying constraint, number of passengers are restricted to less than or equal to available seats in the plane. In specified system, flights for which conditions does not hold are invalid.

In UML 2, constraints and statements in the UML designs can be defined using OCL. Taking an example, say we have a statement 2+7. It is a genuine OCL statement of integer type, depicting integer value 9. An OCL expression can be used as a constraint, if it is of type Boolean.

OCL expressions represent a value. Which can be a simple value, (like an integer), a collection of values, may be a reference to an object, or a collection of references to objects. A Boolean value which represents a message in an interaction diagram or a constraint in a state chart can also be represented using an OCL expression. For example, an expression stated below, defines the availableSeats() operation of the Flight class:

```
context Flight::availableSeats() : Integer

body: plane.numberofSeats - passengers->size()
```
Examples of OCL expressions include the derivation of a derived attribute, association/specification of the initial value of attributes/associations.

Any system value/values can be represented using OCL expression. OCL has same capabilities as SQL. We can use a single OCL expression representing the whole body of a query operation. But, SQL is not a constraint language whereas OCL is a both a query and a constraint and language at the same time.

3.2.2 Foundation with Mathematical Logic

OCL is based on predicate logic and set theory, making it significant. But no mathematical symbols are used. One who is familiar with mathematical notation can easily use to represent accurate statements but not everyone can interpret it. Hence, a mathematical notation cannot be considered as a standard language.

For a modeling language, we need to have precision using mathematics, as well as it should be as simple as natural language. Both the above requirements are conflicting, so we need to look for a balance between these requirements. OCL provides this balance using mathematical concept and simple ASCII words. Therefore, we have an unambiguous language
which is easy for both, users of object technology and their customers.

One can also define syntax of OCL for himself provided that syntax should map to the language structures as defined in the standard.

### 3.2.3 Strongly Typed Language

As OCL expressions specify the type of data hence it’s a typed language. It is not necessary that every model is directly executed and hence there would we many OCL expression written for which corresponding systems will don’t have executable versions. Even then, one can test an OCL expression. These expressions can be checked before execution (i.e. while modeling phase) and hence errors can be removed at modeling stage only.

### 3.2.4 Declarative Language

OCL expressions describe what is to be accomplished rather than how to do it hence it comes under declarative language category. When an OCL expression is evaluated, it does not affect the system’s state.

The model developers do not go deep into how computations should be done, rather decisions are made at high level of abstraction. How something should be accomplished depends
on the implementation approach. One can mention the associations in the coding.

Let us consider an example, suppose we need to know all the passengers on the flight. There could be many implementation approaches such as: flight object can have references to all of its passengers or passengers can be looked for in the database (say Passenger Table) or an additional attribute can be added to the flight object which would specify the number of passengers. For all above possible implementations we have OCL, as it does not require to specify how whereas a procedural language has to be based on the implementation method. Thus OCL is acceptable for PIM development.

3.3 MODEL DRIVEN ARCHITECTURE

Model Driven Development [66, 67, 68, 69] play a significant role in software development process. Under the supervision of Object Management Group (OMG), the MDA framework [70, 71, 72, 73] has been developed. OMG defines how models can be converted from one language into another. For example, UML model can be used to develop database schemas. Herein conversion takes place from UML language to SQL language. In this scenario UML source code is treated as a model. Thus we have another type of transformation of UML model into a code.
3.3.1 Explaining Platform Independent Model (PIMs) and Platform Specific Models (PSMs)

The major aspect of Model Driven Architecture (MDA) is the significance of these models in software development. Inside MDA, the designing and development of software is governed by modeling the proposed software system.

There are three steps in MDA process:

1. Development of a platform independent model (PIM) i.e, the model has high level of abstraction and does not depend on any technology for its implementation.

2. Transformation of this PIM model into some models which are changed in such a way so that they represent our system which is available for some specified approaches of implementation. Such models are considered as Platform Specific Models (PSMs). For example, Enterprise Java Beans (EJB) model and a database model

3. The third step includes transferring the PSM into code.

The third step is easy but the most difficult one is involving the transformation of a PIM into a PSM [6]. Fig 3.1 depicts the relations between the following: PIM, PSM, source code and the transformations among these.
3.3.2 Automation of Transformations

One more major feature of MDA is - it has tools to execute these transformations. Multiple tools are available for transforming a PSM into code. But now PIM to PSM transformation is also automated. This shows the advantages we get from MDA. People involved in software development business can understand the time span it takes to carry out tasks which are almost in routine. To quote on example, using object-oriented design for development of a database model, developing a Common Object Model (COM) component model etc from exiting high-level design. Thus, Model Driven Architecture aims at automating the transformations involved in software development process.
3.3.3 MDA Building Blocks

A number of strongly connected parts are there in the MDA framework. To interpret the framework, the independent parts and the mutual relationships among these parts must also be known. MDA framework has following independent parts: models, modeling languages, transformation tools and related definitions, as shown in Fig 3.2.

![Fig 3.2. The MDA Framework](image)

1. Descriptions of Models

High level models (PIMs) and low level models (PSMs) are included to design MDA architecture. Approach of MDA is transformation of a PIM into few numbers of PSMs, where each PSM is available for a different implementation technology. In case PIM was designed for only one target, then transforming it into PSMs for different technologies would not have been possible. Therefore PIM must be designed independent of implementation technology.
But, PSM should represent the features of corresponding technology. For example, say a PSM is targeted as a database, so the features of a database like table, columns, keys should be recognizable easily. Best relation between the PSM and its technology requires that PSM to code transformation is precise, effective and efficient.

Both of the models, PSM and PIM, must be accurate, consistent and should have maximum possible system information. This is the advantage of OCL over UML as it provides far more information about the system as compared to UML.

2. Description of Modeling Languages

They are an important part of the MDA framework. We need a well-defined modeling language for PIMs and PSMs so that this language can be interpreted by the automated tools used for transformations of PIMs and PSMs. Since before a system is built a person only knows the purpose of a PIM thus, a PIM is hand-written therefore it is necessary to write a PIM in clear, precise form so that other person can interpret it. Thus, the need for a modeling language for PIM is highly demanded which should be interpretable by humans as well as machines.
Since PSMs are generated using automated tools, thus PSMs must be interpretable by machines and by the experts of the PSM’s targeted technology. Thus there is less demand for modeling languages for PSM as compared to PIM. At present, many specified profiles for UML define UML-like languages for specified approaches, for example, EJB profile.

3. Description of Transformation Tools

The market of transformations is growing rapidly. These tools carry out the core part of the MDA approach, by providing automation to a significant part of the software development process. Multiple tools are available for transforming PSM into code. But, very few are available for implementing PIM to PSM transformations. Almost all the tools available for PIM to PSM transformation can also carry out PSM to code transformation. Thus, these tools are flexible according to transformation need of the user.

4. Transformation Definitions

In MDA framework, it is important to concentrate on the definitions for conversions among PIM into PSM and PSM into code. Since, we need to reuse these definitions so they must be separated from the tools which are used to implement these definitions. It is not a good practice to build a new
transformation definition every time, rather definition should be build in a specific language so that it can be used again and again whenever required.

It may be possible that a developer develops a user-defined transformation definition as per requirements. But these definitions would be available in public domain and hence can be tailored by the users according to their own needs.

Some vendors develop certain transformation definitions which are hidden inside the tools and thus cannot be used by the users.

3.3.4 Advantages of MDA

Advantages of the MDA approaches are identified as follows:-

1. The major advantage is Portability, application re-use, minimization of cost and complexity of application development along with its management. Since PIMs are developed independent of implementing technology therefore can develop PSMs for different platforms thus, MDA approach supports portability.

2. The Productivity has been increased by making capable to make developers, designers and system administrators to apply language approaches which are much more interpretable across the whole team
members. A productivity profit could be obtained by applying fully automated tools for PIM-to PSM development and PSM to code development.

3. Third major advantage is interoperability across different platforms ensure that specific business function must be implemented by all the standards of different implementation technologies. The objective of cross-platform interoperability is that tools should develop PSMs as well as bridge these PSMs among themselves and with other platforms.

4. MDA provides documentation, by embedding the information in the PIM itself. Also it provides easy maintenance as developing a PIM is much easier than writing it manually.

3.4 INTEGRATING OCL AND MDA FRAMEWORK

MDA can only be applied provided the, models are at maturity level 4, so that they have direct link with the source code. That is, from PIM to PSM and PSM to code, it is necessary to have high level models.

3.4.1 Why Integrate UML and OCL?

Software modeling is thought as process of developing diagrams, since models have bubbles, arrows, figures along
with some text. But such models don’t depict consistent information. There are many limitations of these diagrams, which weakens the model.

There are some statements which must be a part of detailed specification, but these statements cannot be expressed using a diagram. For example, fig 3.3 shows a UML model, with two classes, Flight and Person. It shows that there are some specified numbers of persons which are passengers on a flight. So, diagram shows (0.....*), i.e., there could be any number of passengers, whereas the truth is that only a finite number of passengers are allowed as per number of seats. Now, this constraint over number of passengers cannot be expressed via diagram alone. OCL constraint notation is added to improve this diagram written as:

context Flight

inv: passengers->size() <= plane.numberOfSeats

Fig 3.3: A model expressed in a diagram
Statements, if expressed mathematically, like OCL offer many advantages for representing the systems, like those statements will not be comprehensible differently by different persons. For example, say a statement will be interpreted identically by the programmer or an analyst. Other benefits are like: unambiguous, accurate, correct and well-defined. Consistency of the statements with the other parts of the model can be checked using Automated tools, also Code generation becomes more powerful. If a model is developed that contains only statements, then it becomes difficult to interpret. Although, source code can serve as a model, but diagrammatic view of a model is more acceptable by most of the people.

Bubbles and arrows, when used in a diagram, makes interpretation easier.

Integration of OCL and UML provides the best solutions. Models can be represented with many different UML diagrams accompanied with OCL statements. Both of them are equally necessary. Absence of UML diagrams will result in that OCL statements will refer to non-existing elements and absence of OCL statements would result in underspecified model. So, integration of both provides the fully specified model.
3.4.2 Representing OCL

Another example is taken in Fig 3.4 where three classes are taken namely: Person, House, and Mortgage, along with their associations. Anyone who depicts this model will assume certain rules:

1. Anyone can hold a mortgage of his house if and only if he himself is the owner (not of any neighbor or friend).
2. For any mortgage, date of starting must proceed date of ending.
3. All persons have a unique social security number.
4. If a person has sufficient income, only then he will be permitted to have a new mortgage.
5. If one’s house has sufficient counter value then only he will be permitted to have a new mortgage.

![Diagram of mortgage system](image)

Fig 3.4: The "mortgage system" expressed in a diagram
These above mentioned rules can in no way be expressed by diagrams. It is necessary to document these rules correctly else different people will predict it differently resulting into faulty implementation.

Even if these rules are just expressed in English which can result into different interpretations? So the problem still continues.

OCL statements are the best solution to get an unambiguous interpretation of these rules. These rules in OCL would be expressed as follows:

```plaintext
context Mortgage
inv: security.owner = borrower

context Mortgage
inv: startDate < endDate

context Person
inv: Person::allInstances()->isUnique(socSecNr)

context Person::getMortgage(sum : Money, security : House)
pre: self.mortgages.monthlyPayment->sum() <= self.salary * 0.30

context Person::getMortgage(sum : Money, security : House)
```
There are many reasons by which these rules must be expressed in form of OCL statements. First of all, users will correctly interpret the model, and errors would be detected at an early stage. Also programmers can easily comprehend the intended meaning of the analyst. OCL statements can be best served as input to automated tools. These tools can be used for simulations, testing, consistency checks, MDA transformations etc. This is only possible, if model depicts entire information required. Since tools cannot understand English rules, so OCL statements are best solution.

### 3.5 Relation of OCL to MDA

MDA framework has the following building blocks: models, languages, transformation definitions, and transformation tools. Out of these building blocks three building blocks can be created using OCL as shown in Fig 3.5:

1. **Models:** models can be built up to maturity level as OCL supports precise specifications.

2. **Transformation definitions:** These definitions need to be formal and precise for capable of being interpreted by the automated tools. OCL supports their requirements.
3. Languages: language definition should be precise and formal so that it is interpretable within the MDA framework, and OCL provides this.

One needs to focus on how OCL can be used for development of better models. Herein we focus on transformation definitions and modeling languages. For this we need to know meta level of modeling, i.e. meta models (using UML and OCL) and meta modeling considering an example of OCL can be used to develop transformation definitions.