CHAPTER 04

REAL TIME CONSTRAINT MODELING USING OBJECT CONSTRAINT LANGUAGE (OCL)

4.1 INTRODUCTION

*Real Time Constraints (RTC) in the form of Strategies* design pattern decomposes real-time specified constraints [74] and behavior from the application service to which they are applicable. The application service is given by a class called service class. Real-time related concepts are delegated to approaches which implement these in a system-specific methodology.

Time bounded constraints (both physical and logical) exist in real time applications. Actions triggered by physical or logical events are supported by RTC. For, hard real time systems, actions must be completed till deadline, whereas for soft real time systems, some tolerance in terms of delay should be allowed.

For real time system based applications, it is difficult to guarantee the timing precision because of non-predictable behavior.
Language used for real time systems must be able to give precise representation of real time constraints which are required for scheduling and fault tolerant policy implementation.

The main objective of object oriented approach is to enhance software quality features like extensibility, reusability, modularity etc.

Herein we first explain the major concepts of object model and then temporal constraints are implemented using multiple program approaches.

In real time systems, preciseness of an operation is calculated by checking it logically as well as considering the time bounds (deadline). For hard real time system, if operation is not completed before/on deadline, it results in complete system failure, whereas in case of soft real time systems, some specific delay is allowed, but will result in decreased quality.

Hard real time systems are used for applications where meeting the deadline is necessary else it will result into critical outcomes (sometimes fatal for human lives). Hard real time system examples are industrial process controller, car engine control system, heart pacemakers etc.
In systems using RTC notation, generally pre-emptive scheduling policies are used. Another option can be Earliest Deadline First (EDF). A system having a combination of non real time applications and hard real time based applications “Adaptive Partition Schedulers” can are used.

Applications involving concurrent access and maintenance of current updates use soft real time systems. Examples, software maintain flight plans, live audio-video systems (in which latency can be tolerated compromising quality).

**4.2 UML PROFILE FOR RTC WITH OCL**

For a given UML model, model developers can define constraints using OCL. OCL can be used for representing pre and post operation conditions invariants for classes and many more. In case of dynamic behavior, constraints cannot be specified using OCL. Since, it is necessary that these constraints must be specified to be ensured of correct system behavior.

Till now, the OCL extension presented allowed model developers to give specifications of RTC [75, 76]. A heavyweight extension is applied over the OCL type metamodel on the current domain.
Although OMG has not yet accepted this proposal but one can use this, as it comprises recent OCL contributions.

4.3 OCL METAMODEL EXTENSIONS

There are two types of packages defined in OCL 2.0 namely ‘OCL type metamodel’ and ‘OCL expression metamodel’. Pre-defined OCL types, UML types are described by ‘OCL type metamodel’ and OCL expressions structures are defined by ‘OCL expression metamodel’. Characteristics of states, configuration of states and their ordering are discussed below:

4.3.1 States: User-defined classes have connected state chart diagrams from which states can be retrieved using OCL. OclModelElementType is the metaclass defined on layer M2 of OCL type metamodel. UML metamodel has some ModelElements(instances of subclass of abstract metaclass state) which are represented by this metaclass. Corresponding to OclModelElementType, we have OclState on layer M1. For every state in statechart diagram, we have an enumeration literal from OclState. A function named obj.oclInState(s) is used for accessing states. Here s denotes the state and obj is the object. This function returns a Boolean variable, whether state s is active(for object obj) in the statechart or not.
4.3.2 Configurations: Hierarchically ordered states serve as the building blocks of State charts. States like history states, stub, synch (Pseudo states) are not considered. A state with set of sub-states is regarded as a composite state. States with orthogonal region are called composite state. States which are neither composite nor pseudo are regarded as simple states. Since at a given time, we can have a number of active states, so it is difficult to identify current state from a given state chart.

Therefore, UML1.4 defines means for active state configurations. Say we have a State chart having simple state (which in turn is a constituent of a composite state), then composite states which have this simple state are also active. A tree structure, for depicting the states which are currently active is used. This tree has a unique root state and leaf nodes represent simple states. Valid (or basic) configuration (ie a list of simple states) can be used for identification of active state configuration. Basic configurations for a given state chart are depicted in Fig 4.1.
‘Configuration- Type’ is a new collection type introduced in ‘OCL type metamodel’ for basic configurations. This is shown in Fig 4.2-a Configuration Type instances are restricted sets (i.e they cannot have OclState type enumeration literals). This OCL property can be represented by following formula:

\[
\text{context ConfigurationType inv:} \\
\text{self.oclIsKindOf(SetType) and self.elementType.name = 'OclState'}
\]

**4.3.3 Execution Paths:** Valid configurations can be sequenced to represent state charts’ execution paths. PathType is introduced for modeling of these execution paths as shown in Figure 4.2-a. Instances of PathType are the sequences (constrained from having a specific element type) on layer M1. So, the element type must be an instance of Configuration- Type. Representing it in OCL, we need:

\[
\text{context PathType inv:}
\]
self.oclIsKindOf(SequenceType) and self.elementType.name = 'Set(OclState)'

**Fig. 4.2: OCL Types and Expressions**

**4.3.4 Execution Path Literals:** Literals are required for explicit specification of execution paths with annotated timing intervals as shown in Fig 4.2-a.

**4.3.5 Temporal Expressions:** TemporalExp is a new operation call introduced corresponding to OCL expression meta-model as shown in Fig 4.2-b. Execution paths referred by the temporal expressions are depicted using this operation call. It has two subclasses namely FutureTemporalExp and PastTemporalExp corresponding to future and past temporal expressions. For defining temporal operation semantics both above specified subclasses of TemporalExp are required.

***************
******