CHAPTER 01

INTRODUCTION

1.1 OVERVIEW

Software Engineering makes capable the programmers to develop systems that function reliability in spite of this complexity. Formal methods [1, 2, 3] are utilized to get the objective above discussed which are mathematical languages, approaches and tools for system verification and specifications. Formal methods are used to explain the hardware/software characteristics of the systems. Framework of formal methods can be used to specify complex and large systems. The guarantee of accuracy in the use of formal method cannot be ensured. The inconsistencies, ambiguities, and incompleteness of the system can be expressed by using this method that might otherwise go undetected. Model Verification & Refinement is a technique for verifying correctness properties of safety – critical reactive system.

1.2 AIMS & OBJECTIVES

The main objective of our research work is to perform a refinement of existing model with the help of formal verification methods. Software verification is classified into two parts:
The first case deals with the implementation of software product that is mapped mechanically to description language of a verification approach.

The second case highlights, a verification tool that can be developed for implementation level descriptions. To achieve and match the given implementation language requirements the verification tools is rewritten.

The important challenges in the above two discussed methods are as follow:

1. In the first case, a model extractor construction is done which can change implementation level program descriptions into model of detailed verification. To ensure the above change correctly, it is required to interpret the semantic content of code at implementation level and change it into similar representation for model of verification.

2. In the second method, a verifier has been developed that can pass precise decisions on the validity of system execution. The development of an interpretable verification system for any formal language can be an important challenge.

The major focus of the proposed approach is on the “Control Software” specific software that communicates
with a continuous environment. These software are developed using some periodic tasks which are executed on a fixed schedule.

Our approach is to look for possible developments in the following fields as follows:

**1.3 FORMAL VERIFICATION OF REAL TIME SYSTEMS**

The functioning of the Real Time System [4, 5] is carried out by formal verification for ensuring efficient performance and functioning.

2. Mobile Data Base Management System verification by Sequence Diagrams in the form of a formal language.

Formal Description Languages support language standardization program verification and software reliability. There are three distinct approaches to specify a programming language – Operational, Semantics, Axiomatic semantics and Devotional semantics. A language in the form of operation of an abstract machine is called operation semantics, which gives an abstract interpreter of the language. A language in terms of assertions and inference rules for reasoning about
the programs is known as axiomatic semantics. The Axiomatic specification of a programming language will be mathematical theory for that language. A Devotional semantics is most useful devotional semantics of the well known programming language ADA, PASCAL and LISP can be found. The formal specification language is often used as a Meta - language for describing programming language semantics. Recently, Object Constraint Language (OCL) [7] is being used for formalization of Object Oriented Language’ semantics.

Apart from great acceptability of Unified Modeling Checker (UMC), Object Constraint Language (OCL) has got an excellent identification of applicability. OCL is mainly used for specifying constraints in UML diagrams (mainly on classes) and in behavioral diagrams (on guards). But still OCL cannot provide sufficient specification for the constraints over dynamic behavior of diagrams. i.e. OCL cannot sufficiently specify constraints over state configurations of states, their evolution and transitions with time. Therefore, real time specifications are not possible using OCL. Equivalence and model checking method in formal verification have been used for few applications. A Nodal checking has been widely accepted in electronic system and protocol verification. For model checking we need to have description of system and a
specification of properties. Generally these properties are represented in Computational Tree Logic (CTL). For model specification we can have OCL based approach wherein we can have OCL specifications instead of CTL specifications.

OCL extensions have been introduced by concepts based on time bounded variant of CTL. Hence, real time and model checking specifications are carried out by OCL. Once the constraints are specified for UML State Chart Diagrams (state space) they can be used for past oriented temporal logics and Activity Diagrams without any modifications.

The key role has been played by description techniques with a modeling method which defines a various views of the systems. Few description techniques from UML, have been used which are specialized for enabling definition of semantics. They are as follows:

1. Informal Text and Diagrams (ITD)
2. Programming Language (PL)
3. State Transition Diagram (STD)
4. Specification Language (SL)
5. Message Sequence Chart (MSE)
6. Object Model (OM)

The above discussed documents are providing with mathematical system model based semantics. Using this
semantics, a precise semantics for documents has achieved along with an integrated one. This permits to represents transformation among documents along with the rigorous context conditions among various description techniques. The system development has been represented by the development graph which consist the documents in the form of nodes and dependencies among the direct arcs. Each document will contain all the information of its predecessor so the document state is captured to analyze whether it is redundant or important. This information is necessary for development process so that it can analyze requirements, design decisions and to permit changes in the requirements in a prescribed way.

1.4 FORMAL VERIFICATION IN MODEL REFINEMENT

The model refinement using formal verification [8] can be best explored if models are created using the language with following features:

1. Language has formal refinement mechanism.

2. Language can evaluate all refinements possible from a given model.

3. Language can prove that a particular model is outcome of refinements on a given model.
Languages like Object-Z, OCL etc. are equipped with such refinement machinery. There is a high applicability of formal software verification in the development of security related software in the previous 10 years.

Various quality levels are defined by ISO in ISO 15408 standard different levels of quality for software testing and verification. Common criteria project has been representing this standard with the members of security organization in the whole world. Various formal specifications and verifications tools have been introduced for quality and standard software development process. The area of concentration includes projects of security from verification of hardware circuit to verification of software driver. Specially, model tool checking has been seen as a successful tool.

The Object Constraint Language is capable of defining constraints over object-oriented system and thus is an expression language. In an object oriented system, the constraints are identified as restrictions imposed on the values.

Object Management Group (OMG) having about 700 companies aims at providing a framework for development of applications (with the help of object oriented programming methodologies) and standards for object oriented design and
analysis. Now-a-days software professionals are rapidly getting habitual with OCL.

At times, UML diagrams fail at representing constraints (like on ions, pre and post conditions) and at this point OCL plays a significant role. Expressions which involve accessing of attributes, invoking operations etc can be constructed easily using OCL. The characters used in OCL lie in normal alphanumeric set and hence is regarded as a simple language. OCL has proved to be significant in large number of application thus exploring many under-0-specified domains (involving UML and OCL).

OCL has been utilized in various forms:

1. specification of constraint on operations
2. to define guard
3. as a navigation language
4. to define pre- and post- conditions on Operations and Methods
5. Specification of type invariants for Stereotypes
6. Specification of invariants on classes and types in the class model

One of the choices of designers for developing real time systems is UML, for enhancing UML notation in modeling of real time application several approaches have been developed.
For applying additional constraints over UML model, the model developers use OCL. Presently, temporal constraints cannot be fully expressed using UML’s real time extension and OCL. Also dynamic behavior involved in UML models (like state evolutions and transitions) cannot be specified significantly using OCL. But, real time systems need to ensure system behavior correctly and thus need these constraints, to be specified correctly. For this, an OCL extension is proposed which is capable of expressing time bounded constraints.

UML provides a technique to present design decisions and requirements at various stages in the software development process. For modeling the behavior of system the level of abstraction should be used that gives adequate information for generating precise diagrams.

**Sequence Diagram (SD) and State Charts (SC).** Sequence Diagram (SD) is much interpretable for humans to produce and explain, while State chart describes system behavior in depth.

**1.5 REAL TIME CONSTRAINT NOTATION (RTCN)**

**MODEL TERMINOLOGY:**
Constraints in real time systems [10, 11, 12] are modeled using Real Time Constraint Notations (RTCN).

**1.5.1 Sequence Diagram and Message Sequence Charts:**
In UML, scenarios can be represented using two main kind of representing scenarios: **Sequence diagram** and **Collaboration diagrams**. Sequence Diagram focuses on order of events with respect to time and the structure of interactions among objects are represented by collaboration diagrams.

**1.5.2 Scenario Composition:**
It plays a significant role in description of Real Time System models. Program’s structure must be reflected by the sequence diagrams.

**1.5.3 Defining Finite State Machine and State Charts:**
FSM (Finite State Automata) defines a system by defining all the possible states and all the transitions that are possible among these states. But, for a complex system, we will have a large FSM which would be beyond the comprehension of people. FSM suffers with state explosion problems which can be controlled using State charts.

**1.5.4 Explaining State Chart Synthesis:**
To ensure the completeness of information for precise cohesive diagram, the real time system model has been
utilized. In software development process, first of all Sequence Diagrams are developed and hence, we compose the state chart from sequence diagram.

1.5.5 Representing Sequence Diagram Composition:

1.5.6 Deterministic Grammar:
The major issue in state chart composition utilizing just adequate information from couple of sequence diagrams for developing precise state charts. These state charts should not reflect ambiguous data.

For a given sequence diagram, corresponding Context Free Grammar (CFG) will constitute a set of message – response pairs. Unique response/ responses must be generated by an object corresponding to every message (or set of messages).

\[
SD \rightarrow \text{message response SD } | \epsilon \hspace{2cm} (1)
\]

Here \( \epsilon \) depicts that no message or response is present.

Objective here is to develop a State Diagram using context free grammar (CFG) of the form

message response \( \rightarrow \alpha \text{ResponseA } | \beta \text{ResponseB } | \chi \text{ResponseC} \hspace{2cm} (2)\)

where \( \alpha, \beta \) and \( \chi \) are specific sequences of messages.
1.6 Representation of State Information, Data & Timing Information:-

For generating a deterministic state chart, state information will be required. It may be possible that execution depends on some data values which are stored but does not represent any state or transition in the model. This additional information is depicted using pre and post conditions.

The timing information, like duration of simulation should create a response of a real time system. The major goal of models is to support engineers with the interpretable concepts of a system, before receiving the expense and problem of actually producing it. Software engineering has a well-established modeling approach and their application of models is identified as a useful and effective approach. Software engineering and specifically real-time embedded software is still an emerging area. It is applied to enhance the complex systems and its modeling approaches are neither prefect nor reliable. Software models have a specific and remarkable advantage over the other engineering approaches: they could be used to automatically produce executable programs for specific platforms.
Mathematical explanation of the modeling language semantics gives the effort to explain properly about system’s characteristics and to forecast its behavior and functioning. Adequate real-time software models could be developed, yet there is another problem of getting it automatically from a model and finally its implementation is a way that produces code which behaves appropriately as the model. An expressive, well founded transformation mechanism for automatic and proprietary accepting code generation design methodology has been developed.

The development of the system must be based on the Platform Independent Model. This is finally derived from Platform Specific Model (PSM). At the meta-level queries, views and transformations are areas that would be important for wellness of MDA.

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

******