Validation and Verification of Space Flight Software from Model Design

S.Manoj (M.E)¹, S.Kalraj (Assistant Professor)², S.Satheesh Kumar (Assistant professor)³
Department of Computer Science and Engineering,
Kongunadu College of Engineering and Technology, Trichy

Abstract — Architectural design, also known as top-level design, describes the system top-level structure and organization and identifies the various components. The application of design patterns is particularly important in this domain to improve the quality of software and reduce the flight software anomalies which lead to major losses. Architectural design patterns for reactive system components are customized to suit the flight software domain. The requirement and effectiveness of verification and validation techniques in guaranteeing that a final system reflects in the quality of final product. Many researcher derived the framework for requirement validation and verification. But most of them is concentrating on all the requirements of the system and using more scenarios testing which will lead to more complexity and resultant data. Analyzing all the resultant data will cause some error and utilize more resource. For that we planned to derive the novel simple framework for validating and verifying the software requirements based on state chart modeling. We hope that our proposed framework will be useful to validate and verify the requirements based on state chart modeling in a simple manner. We planned to evaluate our proposed framework using NASA mission flight software.

Keywords— Behavior, state chart assertion, verification and validation, software architecture, flight control system, UML modeling analysis, complex system design.

I. INTRODUCTION

The modeling is mainly used to develop automated techniques for validation and verification in the reactive system. A reactive system is a computer program that continuously interacts with its environment. Reactive denotes the interactive character of the software in the simplest case a reactive program interacts with its environment while more realistic systems the program is a collection of elements interacting with each other as well as their common environment. Reactive systems are different from computation systems. Computational system program transforms an input into an output. It should terminate to produce output. Non-termination is not allowed for computation program. In case of termination, the result produced should be unique.

Here consider space flight software as reactive system. Because the software need to be highly reliable. Designs are known as the description of communicating and classes that are customized to solve general problems in system. Flight software always known as complex system, where the errors can be occurred when the components are interacting with each other. Behavior of components will be explained using various modeling methods [1]. This approach not only generates the codes for entire system, but also provides error identification in the design stages too. We have developed a design pattern that is based on different models and that will overcome advantages of existing verification and test case generation technologies developed. The design pattern aims to provide automated techniques for analysis and test case generation for UML models of reactive systems. The modeling can be represented in two ways. First representation used to specify the design for modeling of various statechart formats. Main purpose is to execute state machine while arranging the diagram and represents its components as close as possible. The representation is designed so that it can represent many components and it can be adapted to specific statechart. The second representation is less abstract and closer to the auto generation of codes from the modeling.

Figure 1. Model Based Analysis and Testing

www.ijsret.org
II. ASSOCIATED TOOLS  
This paper uses a tool called Visual paradigm for Java to build and execute the state machines, also using various plug-in in eclipse to perform validation and verification in generated java codes. The visual paradigm is a behavior based modeling tool. This tool provides report generation and code engineering capabilities including code generation\(^3\). It can reverse engineer diagrams from modified codes. tool which supports every modeling formats of Unified Modeling Language. Here we are using the class diagram and state machine diagram to develop the modeling of a reactive system. Where initially a class diagram has been developed for the components and operations takes place on the system. from class diagram, the state machine diagram developed as a subdiagram, from the diagram only the JAVA codes are generated automatically. Also using PMD tool for validation and checkstyle tool for the verification.

```java
public proContext(pro owner)
{
    super();
    _owner = owner;
    setState(proFSM.prelaunchcheck);
    proFSM.prelaunchcheck.Entry(this);
}
public proState getState()
    throws StateUndefinedException
{
    if (_state == null)
    {
        throw(new StateUndefinedException());
    }
    return ((proState) _state);
}
protected pro getOwner()
{
    return (_owner);
}
```

Figure 2. Generated codes-sample

III. MODEL BASED SOFTWARE DEVELOPMENT  
The model is combining part of the software development process. Component combining can be captured in UML class diagrams, while dynamic behaviour can be captured in UML Statechart diagrams. A statechart contains rectangles that representing states, arrows represent transitions between states and explanation with the event that causes the transition to occur. Each state can specify actions that are taken on entry or exit. Transitions also can specify guards that are control the contexts in which the transition will occur. these statechart models can be automatically mapped into the application code by means of a statechart tool. statecharts are not just part of the design documentation, they can be maintained along with the source code. Statechart contains some of the features, that are composite states, events, actions, signals, guards, junctions, orthogonal regions, initial states, and deep history states. The tool expects as input UML consistent with that produced by the Visual Paradigm UML modeling tool. The generated C and Java code uses the standard statechart translations provided by the Framework. Each state is represented as a method. The tool generates code for all components of the statechart as well as added calls to external methods that implement low-level actions.

IV. STATE MACHINE  
In modeling components of a system specified with a UML modeling diagram, here class diagram and state machine diagrams are considered for the modeling. A class diagram is a type of static structure diagram that describes the structure of a system by showing the system’s classes, their attributes, operations and the relationships among objects. The class diagram is the main building block of object oriented modeling. It is used both for general conceptual modeling of the systematic of the application, and for detailed modeling translating the models into programming code. The classes in a class diagram represent both the main objects, interactions in the application and the classes to be programmed. A class with three sections. In the diagram, classes are represented with boxes which contain three parts. That are First top part contains the name of the class. It is printed in Bold, centered and the first letter capitalized, second middle part contains the attributes of the class. They are left aligned and the first letter is lower case, then bottom part gives the methods or operations the class can take or undertake. In the design of a system, a number of classes are identified and grouped together in a class diagram which helps to determine the static relations between those objects. With detailed modeling, the classes of the conceptual design are often split into a number of subclasses\(^3\). State diagrams require that the system described is composed of a number of states. Many forms of state diagrams exist. State diagrams are used to give an abstract description of the behavior of a system. This behavior is analyzed and represented in series of events, that could occur in one or more possible states. State diagrams can be used to graphically represent finite state machines.
V. SPECIFICATION, VALIDATION AND VERIFICATION

Specification is composed of a set of invariants and a set of Structured Reactive Systems statemachines. Invariants are used to concisely and precisely characterize the system behavior. These are derived as a result of the requirements analysis. The SRS statemachines are a variant of classical statemachines, designed to take advantage of characteristic structures of reactive systems and to eliminate problems with traditional statechart semantics. An SRS statemachine is produced for each system component that describes its dynamic behavior. The dynamic behavior of the system shows how components move between states. The specification is validated by the tool automatically checking completeness.

Data control flow diagrams are used to devise the system structure. Large systems can be decomposed into smaller subsystems that can be independently developed and tested. A number of decomposition approaches have been defined that can be applied automatically to define the system architecture. Verification is intended to check that a product, service, or system meets a set of design specifications. Verification procedures involve performing special tests to model or simulate a module. Verification procedures involve regularly repeating tests devised specifically to ensure that the product or system continues to meet the initial design requirements as time progresses.

Validation is intended to ensure a product, service, or system result in a product, service, or system that meets the operational needs of the user. A new development flow or verification flow, validation procedures may involve modeling either flow and using simulations to predict errors that might lead to incomplete verification or development of a product. Validation procedures also include those that are designed specifically to ensure that modifications made to an existing qualified development flow or verification flow will have the effect of producing a product, validation help to keep the flow qualified. This also involves acceptance of fitness for purpose with end users and other product stakeholders.

VI. FLIGHT SOFTWARE COMPONENTS SPECIFICATIONS

Space flight system contains various controlling operations like orbit control, radar control, attitude control. Where each operation is contains interconnection between them. This paper takes orbit control to develop it’s design using modeling and using only critical components of a system alone. It mainly used to reduce the complexity of developing design from components. Orbit control is used to manage the functions between flight launch to settle in earth orbit \(^4\). Here the orbit estimator used to control the actions takes place in navigation sensor and actuator. Both sensor and actuators contains simulators.

VALIDATION

Validation is requisite in the quality management process. It makes sure that the process or product meets the purpose intended. Where the validation done with PMD tool. PMD is a static Java source code analysis tool. It searches Java code for inefficient code, bugs, common coding problems, and other such issues. PMD can be used in the development environment through integrated development environment. PMD uses rules to perform the source code analysis, and the rules are grouped into rulesets. The rules are categorized by the sort of problem they check for thus the unused code ruleset finds unused local variables and private fields and methods, the strict exception ruleset finds methods that throw Exception and catch blocks that catch NullPointerException, and so forth. There are also library specific rulesets. For example, there’s a JUnit ruleset that finds common problems in suites. Each rule has a suite of unit tests to minimize the number of obvious false positives reported. The PMD reports reasonably clean. Currently PMD testing one source file at a time, which limits the scope of its rules.
PMD checks source code with some rules and setups a report. Like follows, passes a file name and a RuleSet into PMD in eclipse. PMD gives an Input codes to the file off to a Java separate. This process is used to divide the codes as various modules with different objects. The Report is now filled with RuleViolations. It will produce XML file or jar file from the inputs given to it as result after the validation process.

VERIFICATION

Verification gives the answer to the question whether the software is being developed in a correct way and validation provides the answer whether the right software is being produced. verification denotes precision whereas validation indicates value of the end or final product. Here the verification has been done with a tool called Checkstyle[2]. Checkstyle is a static code analysis tool used in software development for checking if Java source code complies with coding rules. The programming style adopted by a software development project can help to comply with good programming practices which improve the code quality, readability, re-usability, and reduce the cost of development. The performed checks mainly limit themselves to the presentation and don't analyse content, and do not confirm the correctness or completeness of the program it may be useful to determine which level of check is needed for a certain type of program. checkstyle used to test various things Javadoc comments for classes, attributes and methods, Naming conventions of attributes and methods, Limit of the number of function parameters, line lengths, Presence of mandatory headers, The use of packets imports, of classes, of scope modifiers and of instructions blocks, The spaces between some characters, The good practices of class onstruction, Duplicated code sections, Multiple complexity measurements, among which expressions.

VII. CONCLUSION

We described a framework for behavior based analysis and test case generation based on UML representation. We applied our framework to the analysis of various critical requirements of the space flight software for NASA space flight software. Our analysis and test case revealed various issues in the models and problems in the code generation environment. Also this tool chain is currently used for UML models, the underlying framework for translation and analysis is very flexible and could be customized to handle other formal semantics. In the future, we plan to make the framework more robust and to apply it further to the analysis of heterogeneous models. And also planned to predict the reliability the flight software which is designed here.

VIII. FUTURE ENHANCEMENT

In future planed to improve the developed framework in testing area. Then to combine some other modules with orbit module to develop a modular structure from the modeling after performing validation and verification. also planed to predict the reliability of a flight software from developed modular structure with a simple framework.

REFERENCE


