Software Verification and Validation with UML in Safety Related System of Nuclear Power Plant

Yun Goo Kim*, Young-Chul Shin*, Jin Young Choi**, Dae Yon Hwang**
* *MMIS Team, Nuclear Engineering and Technology Institute, KHNP Co., 25-1, Jang-dong, Yueseong-gu, Daejeon, Korea, 305-343 goodguy@khnp.co.kr and ycshin@khnp.co.kr*
** *Korea Univ., Anam, Sungbuk, 136-701, Seoul, Korea, choi@formal.korea.ac.kr and dyhwang@formal.korea.ac.kr*

1. Introduction

Digital I&C system with computer is being used widely in Nuclear Power Plant(NPP) because of the development of digital and IT technologies. Also, windows OS and object oriented application are adopted in the reason of user-friend characteristic and easy development environment. Various software development techniques in object oriented environment are introduced but software V&V technique for NPP Safety Related System in object oriented environment was not seriously considered. So the software V&V methodology for the development of object oriented NPP application, especially with UML, has been developed.

2. S/W V&V with UML in each life cycle.

UML has intuitive and easy-understanding notation and its diagrams give the transparent view of the requirement and functions to stake holders such as user, developer, buyer and so on. Figure 1 shows concept of S/W V&V with UML and formal verification.

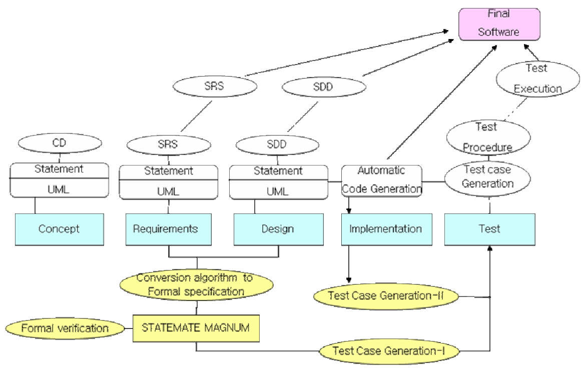

Figure 1 S/W V&V with UML and formal verification

2.1. Requirement Phase

Use case diagram is used in parallel with textual form of requirement to realize it. These use case diagram assists software requirement evaluation whose acceptance criteria are correctness, completeness, accuracy, testability, consistency and so on. Also, System V&V test plan and Acceptance V&V test plan can be generated from use case diagram.

2.2. Design Phase

In Design phase Class diagram, Sequence diagram, Collaboration diagram, Activity diagrams, and State diagram are used for detail design according to requirement. Class diagram can be used directly in Object Oriented development, and program source code can be implemented automatically form State diagram.

Most software logic and functions are designed by State diagram and UML State diagram is similar to formal State chart. So, convert algorithm from UML state diagram to formal State chart, developed and formal verification is done by STATEMATE [1].

2.3. Implementation Phase

Rhapsody is used for UML modeling and Rhapsody can generate code from class diagram and State diagram. If model and design are verified well, no error can be induced in implementation phase.

2.4. Test Case generation

Various UML diagrams at each life cycle can be used to generate test case, for example component test case from State chart, Integration test case from collaboration diagram and system test case from use case diagram. Formal model also generate test case from state chart and, these two cases complement each other.

3. Formal Verification

Although the syntax of Rhapsody state diagram and STATEMATE statechart is very similar, semantics of Rhapsody state diagram is quite different from that of STATEMATE. One main reason is that Rhapsody allows implicit C++ function to be inserted into state transitions. Another big difference is how they interpret the notion of a step. In STATEMATE, a variable value change or events that occurred do not take effect in the current state. On the contrary, in Rhapsody, a step is divided into micro steps and value changes or events will give effect to the system on the next micro step. This leads to different interpretation of concurrency also. In STATEMATE all transition actions are done simultaneously, when in Rhapsody they are executed sequentially.

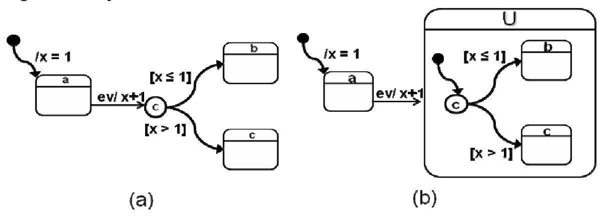

Figure 2 Differences between Rhapsody and STATEMATE

For example, in the Figure 2, Rhapsody and STATEMATE work same in case (a), but in case (b) they have different result. In the case of Rhapsody, the state go to c because of that the value of x changes when the state enters to U. But in STATEMATE the state go to b, because of that the value of x changes when the state enters to b.

There are several other cases which result of Rhapsody and STATEMATE, so below conversion algorithm and design rules has been developed to change rhapsody state diagram to STATEMATE statechart.

- Dynamic state transition.
- Priority of event between inside and outside of state
- Prevent multi related action in transition.
- Difference between event and trigger operation.
- Change of value of variables in event.

Because of these differences it is recommended that formal conversion is applied to core body of logical algorithm in software which is still good enough to illustrate the system. It is excluded that detailed information inserted for code generation.

4. V-model with UML and Formal Verification

V-model with UML and formal verification is proposed. In this model, modeling and implementation can be merged in the consequence of automatic code generation from UML model. And Formal verification path is generated under the use of STATEMATE.

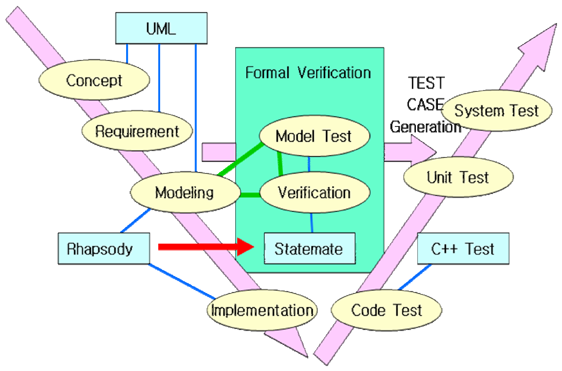

Figure 3 V-model with UML and formal verification

5. Conclusion

V&V method for safety related system software in Nuclear Power Plant, which is developed in object oriented environment, was reviewed and V&V characteristic for NPP software and object oriented software were considered. Also UML was used in every software life cycle and each activity requested in every cycle was developed with UML diagram. Developed UML diagram was changed to formal specification and the conversion algorithm for this conversion was developed. Also test case generation in this environment was suggested

REFERENCES

[1] D. Harel and M. Politi, Modeling Reactive Systems with Statecharts: The STATEMATE Approach, McGraw-Hill , 1998.
[2] M.A.Hennell, D.Hedley & I.J. Riddell, Automated Testing Techniques for Real-time Embedded Software,
[3] V. Encontre, Testing Embedded Systems: Do you have the GuTs for it?,
[4] A.M.R.Vincenzi, J.C.Maldonado, M.E.Delamaro, E.S.Spoto & W.E.Wong, Component-Based Software: An Overview of Testing, Component-Based Software Quality, LNCS 2693, Springer-Verlag, pp.99-127, 2003.
[5] W.Tsai, L.Yu, F.Zhu & R.Paul, Rapid Embedded System Testing Using Verification Patterns, IEEE Software, pp.68-75, July/August, 2005.
[6] D.Kaul & A.Gokhale, Middleware Specialization using Aspect Oriented Programming, ACM Software Engineering, March, 2006.
[7] J.Zander-Nowicka, Z.R.Dai & I.Schieferdecker, Model Driven Testing of Real- Time Embedded Systems - from Object Oriented towards Function Oriented development, Proceedings of Object-Oriented Modeling of Embedded Real-Time Systems, 2005.
[8] M.Utting, A.Pretschner & B.Legeard, A Taxonomy of Model-Based Testing, Working Paper, University of Waikato, 2006.
[9] M.Bjorkander, Model-Based Testing: Getting More Out of Modeling, Proceedings of Object-Oriented Modeling of Embedded Real-Time Systems, 2005.
[10] R.Wenner, Abstract Test Aspect: Testing with AOP, XP 2004, LNCS 3092, Springer-Verlag, pp.237-241, 2004.