Published by :
http://www.ijert.org

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 5 Issue 06, June-2016

Code Generation for Verified Model based Embedded Systems

Pravin Y. Karmore
Research Scholar
RTM Nagpur University
Nagpur, INDIA

Pradeep K. Butey
Dept. of Computer Science
Kamla Nehru College
Nagpur, INDIA

***Abstract*— Verification of requirements in the early stages of system development reduces the cost of testing the system. In this paper we analyses the different approaches for verification of model based embedded system requirements. Various modeling languages have their own tools for the development and code generation. The selection of proper tool in the design stage of embedded system, developer can understand the requirements of hardware and software to build the system. Verification and code generation will be more helpful for the development of safety-critical embedded systems. Modeling languages can be used to develop reliable embedded software code. However, verified generated code usability depends on embedded software developers.**

Keywords*— *Embedded System, Model based designing, Verification and code generation.

I. INTRODUCTION

A big challenge for embedded software engineering is to address this growing variety and complexity of the software in the embedded system and ensure sufficient product quality. In order to achieve this, structured embedded software engineering and automation are inevitable [4]. An embedded system is a specifically designed computing device which is used inside of a device. For example, an embedded system in a microwave oven accepts user input from the panel, manages the LCD display, and controls the heating elements of microwave. Embedded systems generally use microprocessors that contain many functions of a computer on a single device (i.e. System-on-chip). Embedded software is often integrated in highly complex devices. Medical device software, automotive software, avionics software, military software and railway software are all used to control devices on which people's lives depend. A fault in that software may not just be inconvenient, it could be disastrous. The many of embedded software developers uses traditional programming languages such as C and C++. It uses inbuilt processes and techniques in the language to improve reliability and reduce security flaws. However, the verified code generation with model based architecture (MBA) [4] approach met with increasing success. Modeling has a major role during embedded software development. In runtime, software requirement specifications are implemented by expected behaviors, and the software components of software could be specify as component mode. Application logical design is separated from software implementation with MBA. Unified Modeling Language (UML) as a key MBD approach is widely used to model software architecture, component, object, and relationships among them. Some non-functional properties of embedded software are real-time, reliability and safety were described by UML. Embedded software development methodologies historically have amended to concentrate on tools that support the embedded software developer with system configuration, integration, and particularly testing. In this paper, we try to illustrate the use of modeling languages for verified code development for embedded systems. Models must be articulated in a modeling language with a properly defined grammar and semantics accomplish by expressing both static structure and dynamic behavior at an abstract level removed from the programming domain [11]. Through the Unified Modeling Language (UML) and the Systems Modeling Language (SysML) [1],[3],[7] provides a set of diagrams with semantic meaning that enable users to communicate the structure and behavior of a design.

II. BACKGROUND

A. *Embedded System*

An embedded system is a specialized microcontroller based computing device used as a part of another system or machine. Normally, an embedded system is built on a single microprocessor board with the software stored in ROM. Some common examples of applications of embedded systems are telecommunications, automobiles, consumer electronics, and plant control. Even though the application domains are dissimilar from each other, they have universal organization in functional configuration. A layered embedded system structure, including application programming interfaces, hardware-dependent software, application software, and hardware platform is shown in Fig. 1 [5][6]. Application program interfaces are essential for communication between the hardware-dependent software (System Software) and the application layer of software (Application programs). The hardware-dependent software is attached with the external physical hardware and network. Real-time Operating System (RTOS) and device drivers are closely attached to the hardware platform of the system. According to an application domain, performance and size are the constraints that usually influence the hardware platform. As per specific application, a processor and memory system must meet a minimum requirement.

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 5 Issue 06, June-2016

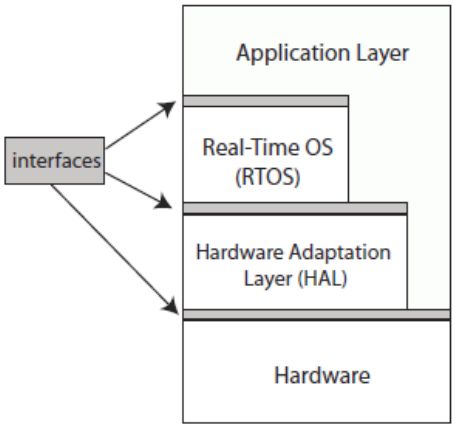

Fig.1. Architecture of an embedded system

B. *Designing Embedded System*

An embedded system designing includes main four steps [7]. The basic steps are the following:

- Requirements specification
- Hardware and software partitioning
- Software design
- Hardware design
- Interface design
- System integration and test

System developers can derive required functions after evaluating system requirements. These functions are considered for allocation of hardware or software. Development of hardware and software is done parallel with the interface design. After development of all required hardware and software components, they are integrated to build a system and go ahead for the testing of system. The system level design steps are shown in Fig. 2.

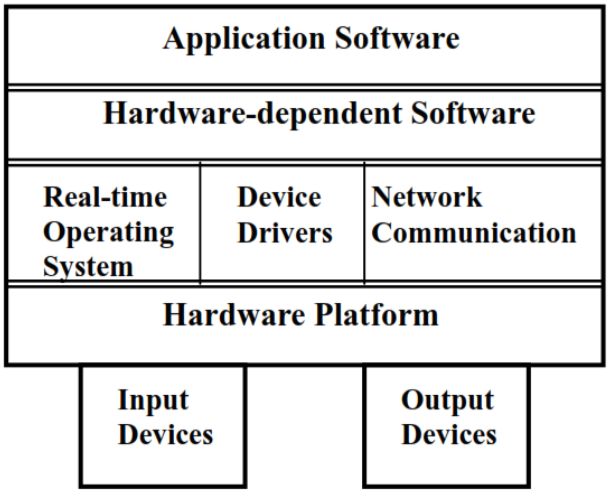

Fig. 2. System-level design processes

III. Overview of UML and SysML

The modeling using the Unified Modeling Language (UML) and the Systems Modeling Language (SysML) provides a rich set of diagrams with semantic meaning that allow users to communicate the structure and behavior of a design as well as maintain consistency across user views [8].

C. *Unified Modeling Language*

In software engineering, the Unified Modeling Language (UML) [1] has become more popular and widely used as a visual modeling language since 1997. Fig. 3 depicts the different types of diagrams available with the latest version of UML. The latest version of UML allows the system developer to create 14 types of diagrams. It is divided into two main categories: structure diagram and behaviour diagram.

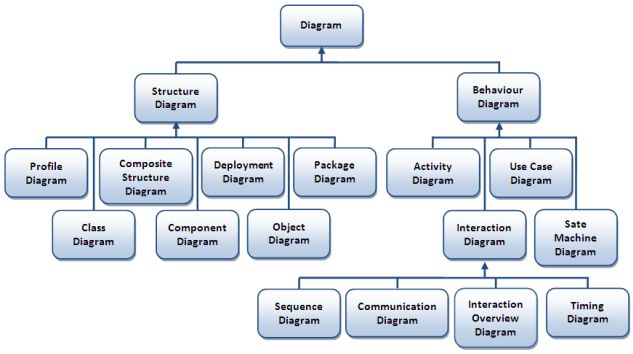

Fig.3. The latest version of UML allows 14 types of diagrams

The structural information category has seven diagram types such as profile, class, composite structure, component, deployment, object and package. The second category is for general types of behavior and used to create activity, interaction, use case and state machine diagrams. Further, interaction category has sequence, communication, interaction overview and timing diagrams.

D. *Systems Modeling Language*

For additional system-engineering concepts in other modeling tools ranging from Visio to Verilog used to model and then integrated them. This will difficult in integrating as per different viewpoints and obtaining traceability, therefore the Object Management Group (OMG) decided to design UML for systems engineering. In the year 2003, a customized version of UML suitable for systems engineering to be specially made by the OMG's System Engineering Domain Special Interest Group (SE DSIG) for systems engineering was intended to support modeling of a broad range of systems, which may include data, hardware, software, procedures, personnel, and facilities. As a result of this, a consortium called SysML Partners proposed the Systems Modeling Language (SysML) [3]. Initially, many new diagrams were considered, but finally just two diagrams the Requirements Diagram and the Parametric Equations Diagram were considered.

1) *Requirements Diagrams*

The Requirements Diagram has an underlying requirements model. SysML states: "The requirements model illustrates the SysML support for describing textual necessities and concerning them to the specification models, analysis models, and design models. A requirement represents the behavior, structure, and/or properties that a system, component, or other model components must satisfy." [9]

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 5 Issue 06, June-2016

2) Parametric Equations Diagrams

Parametric Equations Diagrams are usually used to model properties and their relationships. The diagrams specify the allowable range values for complex mathematical and logical expressions as well as constraints. Generally a reference to the language is used to state the expressions and constraints.

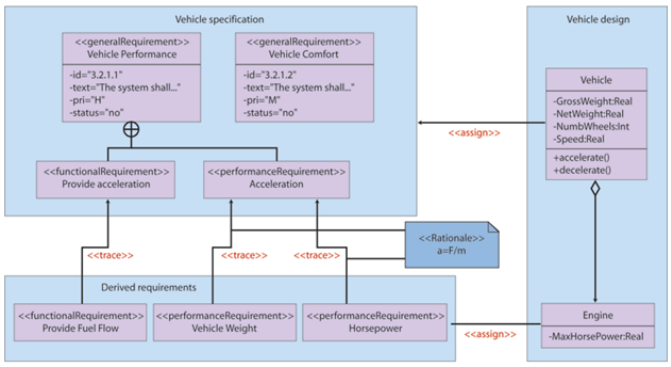

Fig. 4. Requirements diagram for vehicle

The parametric model can include logical expressions, differential equations such as {when Y=7 or X<1}, or other constraints such as {Y< 3x+7}, expressed in a specific language, such as MathML or a programming language. Generally, parametric models are captured in analysis models to support performance models; feedback and control; and engineering models for safety, reliability, mass properties, and design to cost [9]. The SysML specification of a simplified model of the antilock braking system in the Car is shown in Fig. 6.

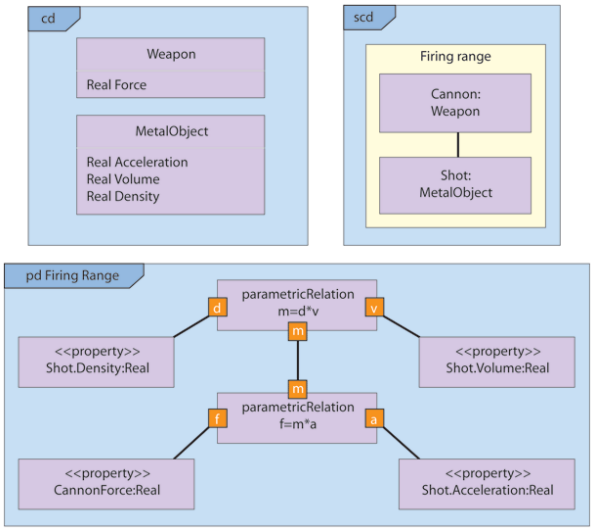

Fig. 5. Parametric equations diagram for weapon and firing

IV. Model Based Design

Model based design (MBD) is based on the efficient use of models as a primary objective throughout the software engineering life cycle. The main objective of MBD is to provide a central role to functional models in the specification, design, integration, and validation of software. Model driven development uses models to represent a system's elements, the structural relationships between them and their dynamic interactions and behavior. Modeling structural relationship supports design exploration and system partitioning. The modeling behavior and interactions are required to verify designs by verifying models and for code generation. But many embedded software developers hesitate to accept the generated code. The rejection of code by developers means loss of MBD advantages. Use of the MBD approach means accepting automatic code generation from models. A properly defined grammar and semantics models is capable of expressing both static structure and dynamic behavior. Such type of models at an abstract level differentiated from the programming domain must be articulated in a modeling language. These languages divided into two groups:

A. Vendor-specific language – It is developed and promoted by a specific vendor of an MBD platform such as Esterel from Esterel Technologies, MatLab and Simulink from MathWorks, and the ASD language used in Verum Software Technologies' Analytical Software Design (ASD): Suite [10] (see Fig. 7 below). ASD Suite allows the system developer to create an initial set of requirements. There is no requirement of coding, testing, and refining each component in separate steps. In view of this, the modeling tool identifies both the external and internal performance of the elements using its two basic model types: interface models and design models.

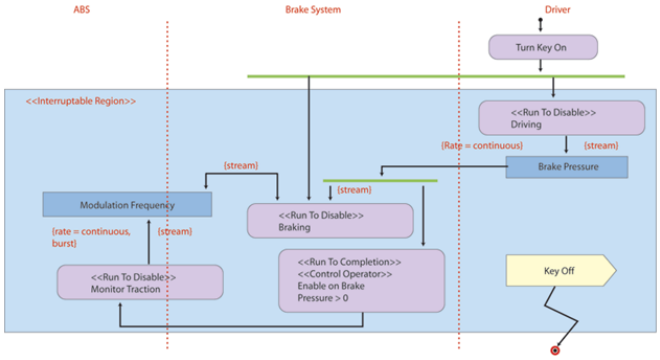

Fig. 6. SysML specifications for antilock breaking system in the Car

B. Standardized languages – A group of interested industry users and MBD platform vendors defined languages, which are most commonly based on the Unified Modeling Language (UML).

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 5 Issue 06, June-2016

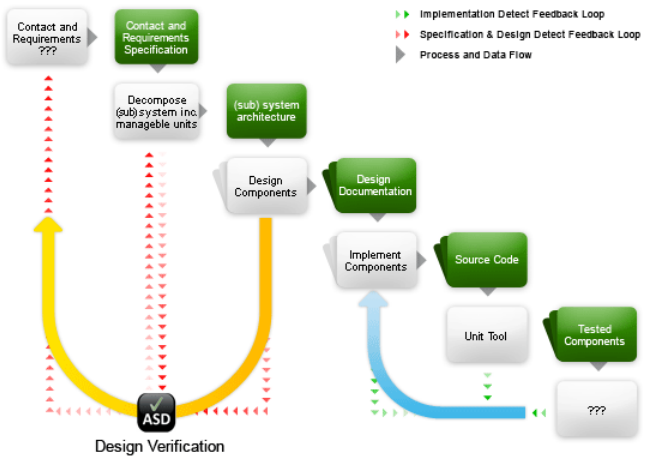

Fig. 7. ASD: Suit Model Driven Design [10]

II. Test Driven Development

Test driven development (TDD) provides various advantages over the traditional software development/test cycle. The test driven development using modeling provides the developer to create an initial set of requirements. In TDD, a developer finds out ways to build the system testable, designs as per the specifications, writes tests and builds, testing strategies, and then writes the functional code to meet the specified requirements of the test-spawned design [12] [13].

Advantages of TDD in embedded software:

1. The code is always tested. Testing drives the design of the code. The code is improved because of the decoupling required to create testable code.
2. The system grows organically as more knowledge of the system is gained. The tests are "living" documentation, because the knowledge is gained in tests.
3. The developers can alter existing code or add new features with confidence because automated regression testing will reveal failures and unexpected results.
4. Because of the inconsistency of hardware and software during development, bugs are due to software, hardware, or a combination of the two.
5. The software bugs can be removed to such an extent that it becomes easier to locate, by method of elimination, the cause of the unexpected system.

V. Verification of Model Driven Design

Using MBD, we can eliminate specification and design errors early in the development cycle where they are cheapest and easiest to rectify. It helps to increase the degree of automation that can be applied to the development process by means of automatic verified code generation. The test driven development of MBD facilitate parallel hardware/software design by enabling system models to be verified using a simulated execution mechanism on development hosts before the target system is available (see Fig. 8) [13]. Such a development reduces the required testing effort by applying automated formal verification methods to the functional models in collaboration with simulating execution behavior instead of relying solely on testing the implemented program code [12].

A. Executable models

The MBD leads to design system rigorous and precise enough to allow for verified code generation, in addition, executable models can be executed on the development system by means of simulation very early in the development life cycle. Executable model provides the rapid and early feedback on specifications and requirements that are verified with the actual requirements of the system. It allows functional verification to perform on the development host design without accessing software running on the target machine. This is very important in case of parallel development of hardware and software development. There are two popular platforms are available, IBM's Rational Rhapsody and Graphics Bridge Point. Both use UML-based modeling languages. The model can be "generated" into code for the model-based design which would run on an embedded target. Using the high end tools, such as IBM Rational Rhapsody, the structure and behavior of the complete model-driven design can be automatically created (see Fig. 9). The model execution with Rational Rhapsody enables early design validation when bugs are less costly to fix.

B. Scalability

The symbolical testing in early design phase examines the overall state space of an embedded software design to identify whether or not the particular properties hold under all possible inputs. There are different approaches to for different model driven design platforms. Some MBD platforms limit the class of designs that can be verified. For example, SCADE Suite of Esterel Technologies deals with synchronous, deterministic designs. A compositional verification approach by ASD of Verum Software Technologies provides verification of entire system component by component to prove the properties still hold when the components are integrated to form the complete system (see Fig. 9). Using this approach completely concurrent and asynchronous design was properly tested for compliance with the specified properties. It also describes the absence of typical asynchronous and concurrent design bugs such as race conditions, live locks and deadlocks. Simulation and testing are normally useless at reducing such errors.

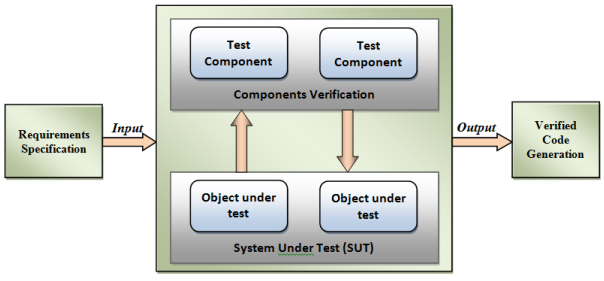

Fig. 8. Model based testing of embedded system components

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 5 Issue 06, June-2016

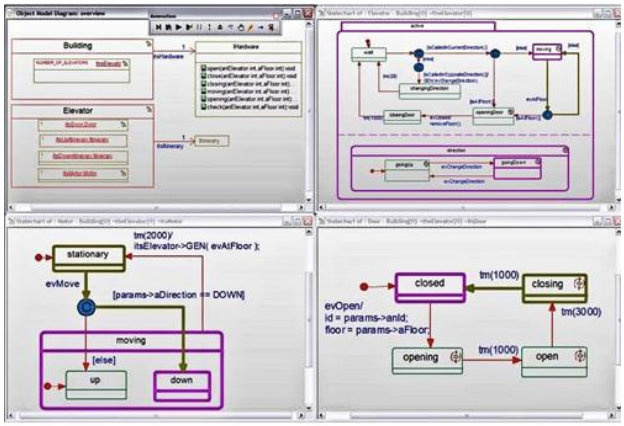

Fig. 9. Model execution with IBM Rational Rhapsody

The MBD platforms widely used in the safety, security and mission-critical domains such as rail transportation, aerospace, automotive and military applications is gradually increasing. If an MBD platform is used with sufficient verification and testing facilities on design models, the unit testing requirement can be reduced. The MBD standards differentiate between the operational embedded software changed as part of the system from the modeling tools used to build that software. The different standards for safety-critical embedded software often needs the tool users to carry out an evaluation of the tool to classify it according to whether or not the tool itself can introduce errors into the operational embedded software and to perform an evaluation of the tool against the appropriate criteria for safe and secure use.

VI. CONCLUSION

Model driven design is an important system development technique worth considering for embedded systems that have safety, security, and/or reliability requirements. The consideration of the most appropriate MBD platform for a particular component and for the overall system requires that developers understand the major technical advantages and disadvantages of available tools. The verified code generation for embedded system using modeling languages is more efficient to reduce cost of testing. Verification early in the design phase ensures that the software uses the full capability of the hardware and thus avoids the redesign of hardware. The requirements verification approach to model based design of embedded system has potential development cost and time efficiencies and ability to reduce the occurrence of software design flaws. The development of universal tool that can be used to model and generate verified software code for all the needs of embedded system development is the future work. The care should be taken while selecting the tool for the designing of such a system. In general, test driven models that are developed using modeling language tools works as a right path for the specified target.

REFRENCES

[1] Object Management Group. OMG Unified Modeling Language (OMG UML), Infrastructure, Version 2.4.1; August 2011.www.sysml.org.
[2] Michael J. Karlesky, William I. Bereza, and Carl B. Erickson, Effective Test Driven Development for Embedded Software, Ph.D. Thesis.
[3] SysML Object Management Group (OMG), 2003. UMLTM for Systems Engineering Request For Proposal OMG Document: ad/03-03-41.
[4] Pravin Karmore and Pradeep Butey, Analysis of Model-based Testing Methodology for Embedded Systems, International Journal of Advanced Research in Computer Science and Software Engineering, Volume 6, Issue 5, P.P 308-314, May 2016.
[5] Alberto Sangiovanni-Vincentelli and Grant Martin, "Platform-Based Design and Software Design Methodology for Embedded Systems," IEEE Design & Test of Computers, November-December, 2001, pp.23-33.
[6] M. Sgroi, L. Lavagno, and A. SangiovanniVincentelli, "Formal Models for Embedded Systems Design," IEEE Design & Test of Computers, April-June, 2000, pp.2-15.
[7] Byeongdo Kang, Young-Jik Kwon, Roger Y. Lee, "A Design and Test Technique for Embedded Software", Proceedings of the 2005 Third ACIS IEEE Int'l Conference on Software Engineering Research, Management and Applications (SERA'05), 2005.
[8] Matthew Hause, Francis Thom, and Alan Moore, "An overview of Systems Modeling Language", December 2005.
[9] Mellor SJ, Balcer MJ. Executable UML, A Foundation for Model-Driven Architecture. Reading, MA, Addison-Wesley; 2002.
[10] Guy Broadfoot,*Using model-driven development to reduce system software security vulnerabilities*, Verum Software Technologies, March 2014.
[11] Padma Iyenghar, ElkePulvermueller and Clemens Westerkamp, "Towards Model-Based Test Automation for Embedded Systems Using UML and UTP", IEEE, ITFA, 2011.
[12] Deepak A. Mathaikutty, SumitAhuja and AjitDingankar, "Model-driven Test Generation for System Level Validation". IEEE, 1-4244-1480, 2007.
[13] David Astels, Test Driven Development: A Practical Guide, Upper Saddle River, NJ: Prentice Hall PTR, 2003.

Authors Profile

Pravin Y. Karmore pursuing Ph.D. in Computer Science from RTM Nagpur University, Nagpur. He is obtained Master in Computer Applications degree from RTM Nagpur University, Nagpur. And M.Phil. degree in Computer Science from Alagappa University, Karaikudi. At present he is working as Assistant Professor at Dept. of Computer Applications, Shri Ramdeobaba College of Engineering and Management, Nagpur. His research area is Software Engineering, Embedded Systems and Neural Networks.

Dr. Padeep K. Butey obtained M.Sc. degree and PGDCS&A from RTM Nagpur University, Nagpur. He obtained his Ph.D. degree in Computer Science from RTM Nagpur University, Nagpur. Now he is working as Associate Professor and Head at Dept. of Computer Science, Kamla Nehru College, Nagpur. He has published more than 35 research papers in various national and international conferences and journals.