MANTECH Publications

Journal of Embedded Systems & Its Applications
Volume 7 Issue 1, January-June 2022

Use of Embedded Systems for Model-Verified Code Generation

P. Rahila [1], Varsha Narmadha [2]

Professor[1], Student[2]

Department of ECE

Sethu Institute of Technology, Pulloor, Kariapatti, Tamil Nadu, India

Corresponding Author's Email:- *varshanarmadha76@yahoo.com*

Abstract

The cost of testing the system is reduced when requirements are verified early in the development process. In this work, we examine the various methodologies for verifying model-based embedded system requirements. Different modeling languages have their own development and code generating tools. The developer may grasp the needs of hardware and software to create the system by selecting the appropriate tool during the design stage of an embedded system. For the development of safety-critical embedded systems, verification and code creation will be more useful. To create dependable embedded software code, modeling languages might be employed. However, the applicability of validated produced code is dependent on embedded software developers.

Keywords: *Model-based design, verification, and code generation are all aspects of embedded systems.*

INTRODUCTION

Addressing the expanding diversity and complexity of software in embedded systems while ensuring adequate product quality is a major problem for embedded software engineering Structured embedded software engineering and automation are required to achieve this [4]. An embedded system is a specially built computing device that is installed into a device. An embedded system in a microwave oven, for example, receives user input from the panel, operates the LCD display, and controls the microwave heating components. Microprocessors, which include many operations of a computer on

a single device, are commonly used in embedded systems (i.e. System-on-chip). Embedded software is frequently found in extremely complicated devices. Medical device software, automobile software, avionics software, military software, and railway software are all used to operate things that people rely on for their life.

A flaw in the programme may be more than simply inconvenient; it could be disastrous. Traditional programming languages like as C and C++ are used by many embedded software developers. It makes advantage of the language's built-in procedures and approaches to increase dependability and decrease security issues. The validated code generation using model-based architecture (MBA) [4] method, on the other hand, has shown growing success.

Embedded software development relies heavily on modeling. Software requirement requirements are realized at runtime by expected behaviors, and software components can be specified as component mode. MBA separates application logical design from software execution. As a core MBD technique, Unified Modeling Language (UML) is extensively used to represent software architecture, components, objects, and their interactions. Non-functional aspects of embedded software include real-time, dependability, and security.

UML was used to define safety. Historically, embedded software development approaches have evolved to focus on tools that assist the embedded software developer with system configuration, integration, and, most importantly, testing. We attempt to demonstrate the usage of modeling languages for validated code development for embedded systems in this research.

Models must be articulated in a modeling language with a well-defined grammar and semantics that convey both static structure and dynamic behavior at an abstract level distinct from the programming domain [11]. [1],[3],[7] offers a collection of diagrams with semantic meaning that enable users to convey the structure and behavior of a design using the Unified Modeling Language (UML) and the Systems Modeling Language (Sys ML).

BACKGROUND

Embedded System

An embedded system is a customized computing device based on a microcontroller that is utilized as part of another system or machine. An embedded

system is often built on a single microprocessor board, with software stored in ROM. Telecommunications, vehicles, consumer electronics, and plant control are some frequent uses of embedded systems.

Despite the fact that the application areas are distinct, they have a common organisational structure in terms of functional configuration. Figure 1 depicts a layered embedded system topology that includes application programming interfaces, hardware-dependent software, application software, and a hardware platform. [5][6]. Application programmed interfaces (APIs) are required for communication between hardware-dependent software (System Software) and the software application layer (Application programs). Hardware-dependent software is linked to external physical hardware and the network.

The system's real-time operating system (RTOS) and device drivers are inextricably linked to its hardware platform. Performance and size are often the restrictions that determine the hardware platform in an application domain. A CPU and memory system must fulfil a minimal requirement depending on the application.

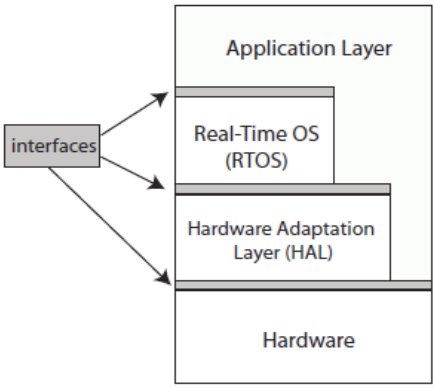

Fig.1. Architecture of an embedded system

Designing Embedded System

The design of an embedded system consists of four major phases [7]. The fundamental steps are as follows:

- Specification of requirements
- Partitioning of hardware and software
- Software development
- Design of hardware
- Designing an interface
- System integration and testing

After examining system requirements, system developers can derive needed functionalities. These functions are taken into account while allocating hardware or software. Hardware and software development occurs along with interface design. Following the construction of all essential hardware and software components, they are combined to form a system and tested. Figure 2 depicts the system level design phases.

Modeling with the Unified Modeling Language (UML) and the Systems Modeling Language (Sys ML) provides a comprehensive collection of diagrams with semantic meaning, allowing users to explain the structure and behavior of a design while maintaining consistency among user perspectives [8].

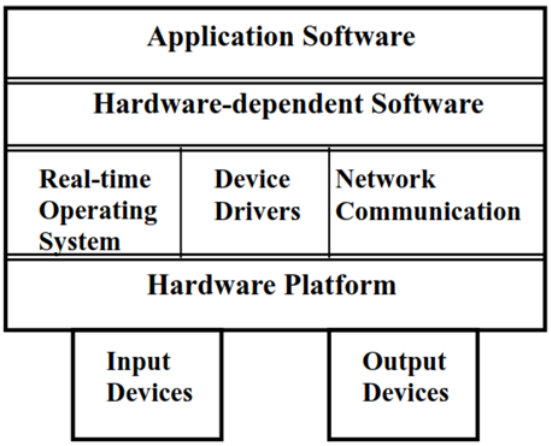

Figure:-2 System-level design processes

OVERVIEW OF UML AND SYSML

Unified Modeling Language

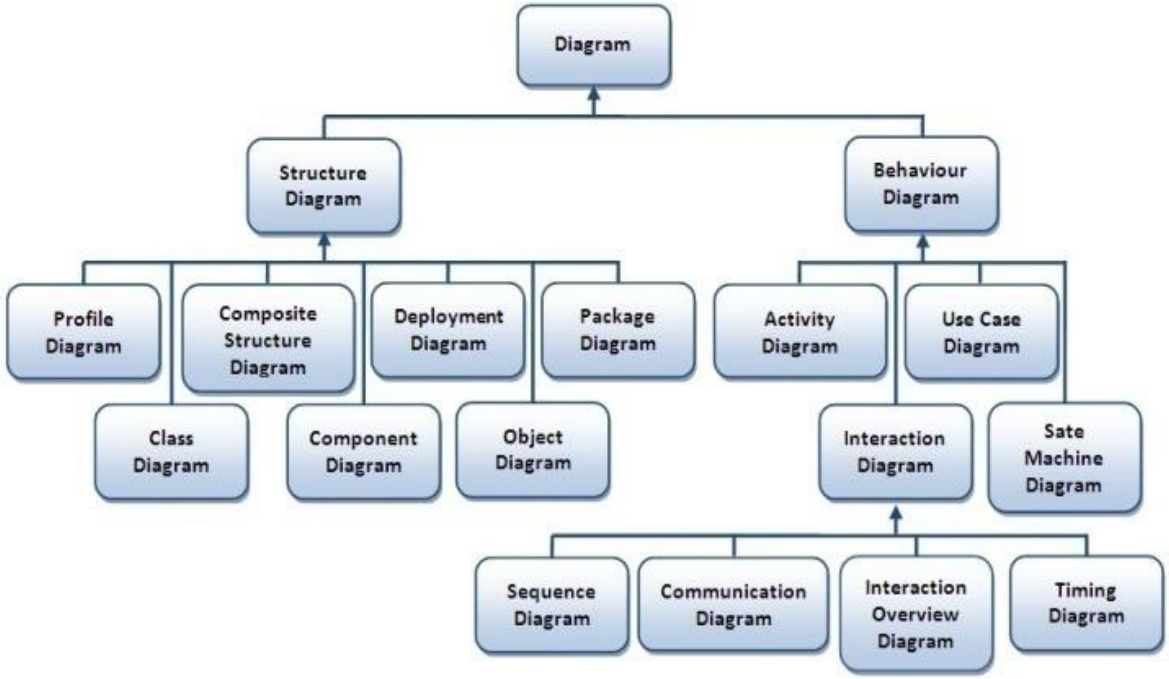

Fig.3. The latest version of UML allows 14 types of diagrams

Since 1997, the Unified Modeling Language (UML) [1] has grown in popularity and widespread use as a visual modeling language in software engineering. Figure 3 displays the many types of diagrams accessible in the most recent version of UML. The most recent version of UML allows system developers to construct 14 different types of diagrams. It is classified into two types: structural diagrams and behavior diagrams. There are seven diagram kinds in the structural information category: profile, class, composite structure, component, deployment, object, and package. The second category is used to build activity, interaction, use case, and state machine diagrams for general forms of behavior. In addition, the interaction category includes sequence, communication, interaction summary, and time diagrams.

Systems Modeling Language

Other modeling tools, ranging from Visio to Verilog, were utilized to model and then integrate new system-engineering principles. This will make integrating from many perspectives and gaining traceability challenging, hence the Object Management Group (OMG) decided to build UML for systems engineering. In 2003, the OMG's System Engineering Domain Special Interest Group (SE DSIG) created a modified version of UML suited for systems engineering to facilitate modeling of a wide range of systems, which may comprise data, hardware, software, processes, persons, and facilities. As a result, the Sys ML Partners collaboration proposed the Systems Modeling Language (Sys ML) [3]. Many new designs were examined at first, but only two were chosen: the Requirements Diagram and the Parametric Equations Diagram.

Requirements Diagrams

The Requirements Diagram is supported by a requirements model. According to Sys ML, "the requirements model demonstrates Sys ML support for specifying textual needs and relating them to specification models, analysis models, and design models." A need is the behavior, structure, and/or qualities that a system, component, or other model component must have." [9]

Parametric Equations Diagrams

Equations with Parameters Typically, diagrams are used to represent attributes and their connections. The diagrams define the range values that may be used for sophisticated mathematical and logical statements, as well as limitations. To state the expressions and limitations, a reference to the language is usually employed.

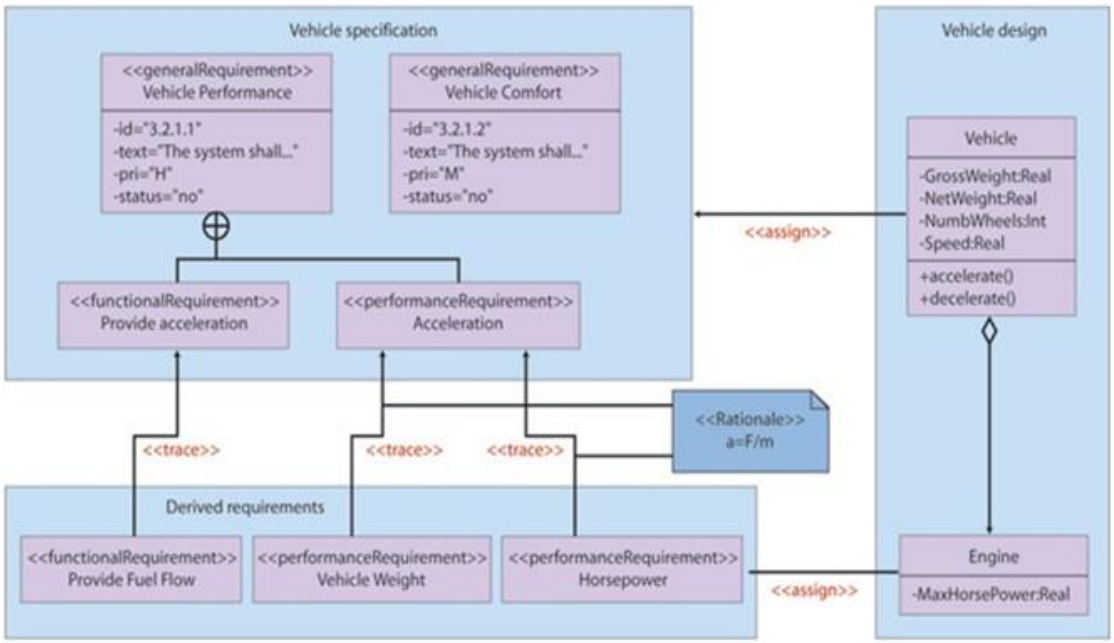

Fig. 4. Requirements diagram for vehicle

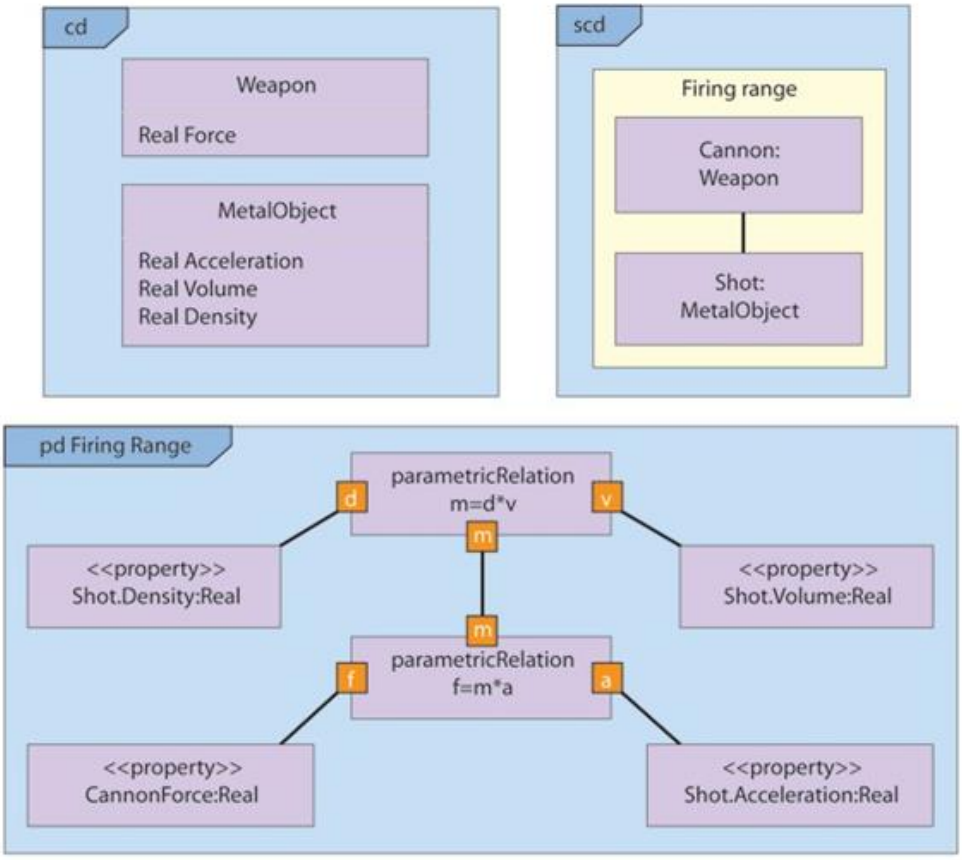

Fig. 5. Parametric equations diagram for weapon and firing

The parametric model can incorporate logical expressions, differential equations like {when Y=7 or X<1}, or other restrictions like {Y <3x+7}, all written in a specific language like Math ML or a programming language. Parametric models are typically represented in analytical models to support performance models, feedback and control models, and engineering models for safety, reliability, mass characteristics, and design to cost [9]. Figure 6 depicts the Sys ML specification of a simplified model of the antilock braking system in the Car.

MODEL BASED DESIGN

MBD is built on the efficient usage of models as a main goal across the software engineering life cycle. The basic goal of MBD is to provide functional models a major role in software specification, design, integration, and validation. Model driven development use models to depict the pieces of a system, their structural links, and their dynamic interactions and behavior Design exploration and system partitioning are aided by structural connection modeling.

Modeling behavior and interactions are necessary for design verification through model verification and code generation. Many embedded software developers, however, are hesitant to accept the resulting code. Developers rejecting code implies that MBD benefits are lost.

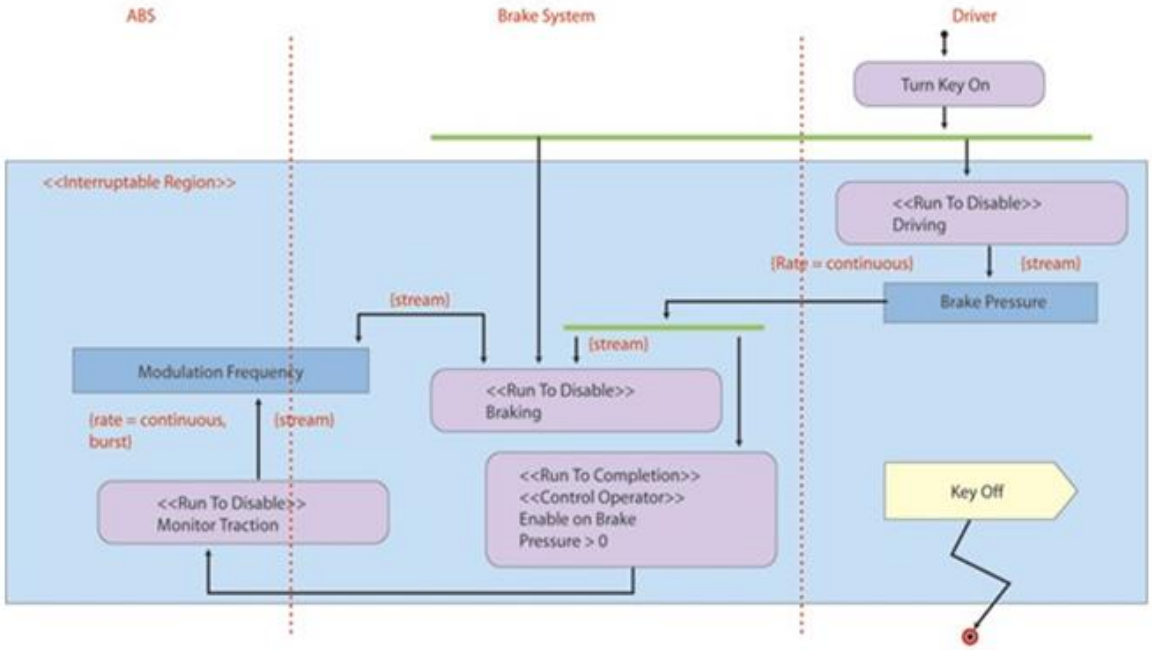

Fig. 6. Sys ML specifications for antilock breaking system in the Car

Accepting automated code creation from models implies using the MBD technique. A correctly designed language and semantics model may represent static structure as well as dynamic behavior. Such models, which are abstracted from the programming domain, must be stated in a modeling language. These languages are classified into two groups:

Vendor-specific language

It is created and marketed by a specific vendor of an MBD platform, such as Esterel Technologies' Esterel, Mat Lab and Simulink from MathWorks, and the ASD language used in Verum Software Technologies' Analytical Software Design (ASD): Suite [10]. (see Fig. 7 below).

The ASD Suite enables the system developer to build a preliminary set of requirements. There is no obligation to code, test, and refine each component separately. As a result, the modeling tool detects the elements' external and internal performance using its two core model types: interface models and design models.

Standardized languages

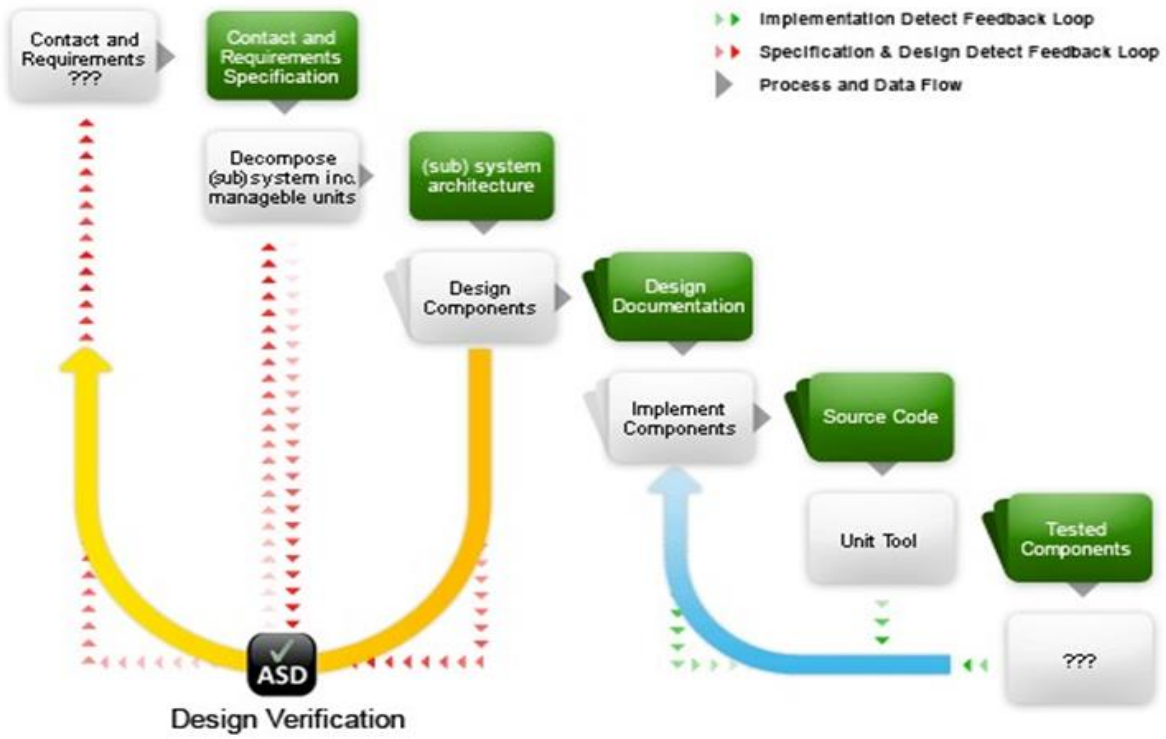

Fig. 7. ASD: Suit Model Driven Design [10]

A consortium of interested industry users and MBD platform suppliers developed languages based on the Unified Modeling Language (UML).

Test Driven Development

TDD has several advantages over the typical software development/test cycle. The use of modeling in test driven development allows the developer to construct an initial set of requirements.

TDD requires a developer to identify ways to make the system testable, design according to specifications, create tests and builds, testing methodologies, and finally write functional code to match the defined needs of the test-spawned design [12] [13].

TDD has the following advantages in embedded software:

1. The code is constantly tested. The design of the code is driven by testing. The decoupling necessary to produce testable programming improves the code.

2. As more information about the system is gathered, the system grows naturally. Because information is developed through testing, the tests are "living" documentation.

3. Developers may confidently modify old code or add new functionality since automated regression testing will show errors and unexpected outcomes.

4. Bugs are caused by software, hardware, or a mix of the two, due to the inconsistency of hardware and software throughout development.

5. Software defects can be eradicated to the point where it's easier to find the source of the unexpected system using the elimination approach.

VERIFICATION OF MODEL DRIVEN DESIGN

We can remove specification and design flaws early in the development cycle, when they are the cheapest and easiest to fix, thanks to MBD. Automatic validated code creation aids in increasing the degree of automation that can be used to the development process. MBD facilitates concurrent hardware/software design by allowing system models to be tested on development hosts using a simulated execution method before the target system is ready (see Fig. 8) [13].

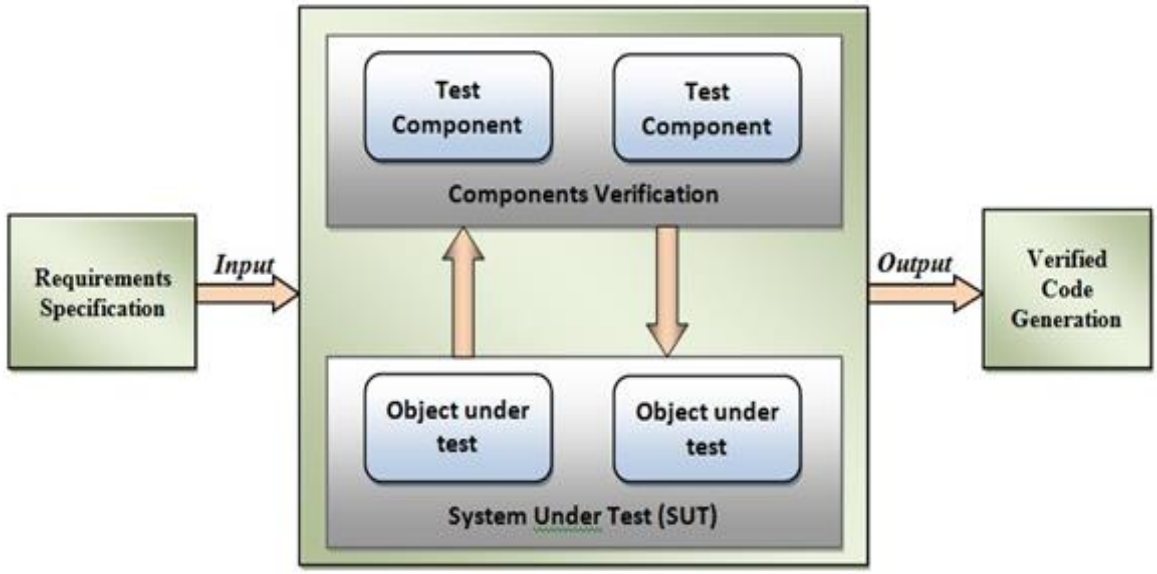

Fig. 8. Model based testing of embedded system components

Instead of depending just on evaluating the implemented programme code, this approach decreases the needed testing work by using automated formal verification methods to the functional models in conjunction with modeling execution behavior [12].

Executable models

The MBD results in a design system that is rigorous and exact enough to allow for validated code creation, as well as executable models that can be run on the development system via simulation early in the development life cycle The executable model gives quick and early feedback on specs and requirements that are cross-checked against the system's real requirements. It enables functional testing on the development host design without requiring access to software on the target system. This is critical in the case of hardware and software development running concurrently.

IBM's Rational Rhapsody and Graphics Bridge Point are two of the most popular systems. Both employ modeling languages based on UML. For a model-based design that runs on an embedded target, the model can be "produced" into code. The structure and behavior of the whole model-driven design may be automatically built using high-end technologies like IBM Rational Rhapsody (see Fig. 9). Rational Rhapsody's model execution allows for early design validation when errors are less expensive to repair.

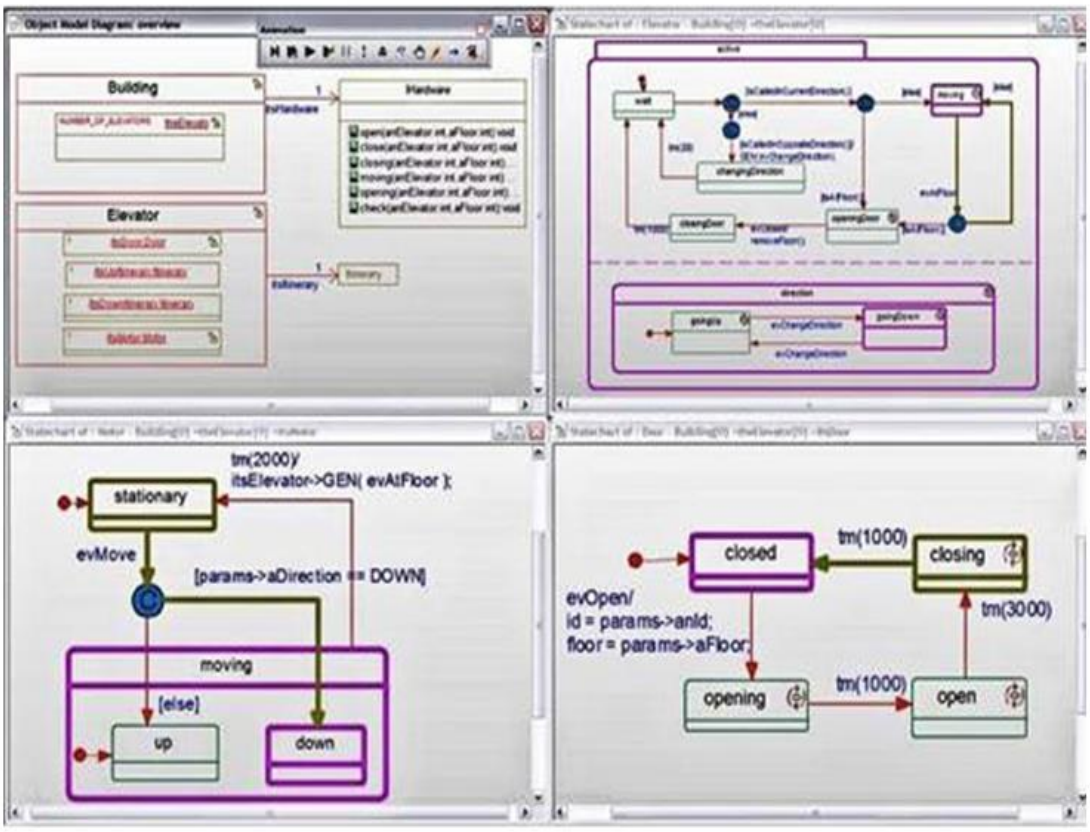

Fig. 9. Model execution with IBM Rational Rhapsody

Scalability

Symbolic testing in the early design phase looks at the whole state space of an embedded software system to see if the specific attributes hold up under all potential inputs. For various model-driven design platforms, there are several methodologies. The types of designs that can be validated on some MBD systems are limited. Esterel Technologies' SCADE Suite, for example, works with synchronous, deterministic designs.

A compositional verification technique developed by ASD of Verum Software Technologies verifies the entire system component by component to ensure that the characteristics are maintained after the components are combined to form the complete system (see Fig. 9).

Completely concurrent and asynchronous designs were thoroughly evaluated for compliance with the given characteristics using this method. It also refers to the lack of common asynchronous and concurrent design flaws such race situations, live locks, and deadlocks. In most cases, simulation and testing are ineffective in reducing such mistakes.

MBD systems are being more frequently employed in safety, security, and mission-critical sectors such rail transportation, aerospace, automotive, and military applications. The unit testing need can be minimized if an MBD platform is utilized with appropriate verification and testing facilities on design models. The MBD standards distinguish between operational embedded software that has been updated as part of the system and the modeling tools that were used to create that software.

The various standards for safety-critical embedded software frequently require tool users to evaluate the tool in order to classify it based on whether the tool can introduce errors into the operational embedded software and to evaluate the tool against the appropriate criteria for safe and secure use.

CONCLUSION

For embedded systems with safety, security, and/or reliability concerns, model driven design is an essential system development method to consider. Developers must grasp the primary technological benefits and disadvantages of different tools before deciding on the best MBD platform for a specific component and the whole system. The development of validated code for embedded systems using modeling languages is more efficient in terms of lowering testing costs. Verification early in the design process guarantees that the software makes full advantage of the hardware's capabilities, avoiding the need for hardware modification.

The requirements verification method to model-based embedded system design has the potential to minimize development costs and time, as well as the occurrence of software design defects. Future work will include the creation of a universal tool that can be used to model and create validated software code for all aspects of embedded system development. When choosing a tool for developing such a system, caution should be exercised. In general, test-driven models created with modeling language tools provide the best approach to the desired outcome.

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. Sangiovanni Vincentelli, "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 Broad foot, Using model-driven development to reduce system software security vulnerabilities, Verum Software Technologies, March 2014.

11. Padma Iyenghar, Elke Pulvermueller and Clemens Westerkamp, "Towards Model-Based Test Automation for Embedded Systems Using UML and UTP", IEEE, ITFA, 2011.

12. Deepak A. Mathaikutty, Sumit Ahuja and Ajit Dingankar, "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.