Proceedings,
The 2013 ASEE North Midwest Section Conference,
October 17-18, 2013, Fargo, North Dakota.

ASEE-NMWSC2013-0050

Integrating Modern Model-based Development Concepts and Tools in a Programming Tools course

Nannan He, Han-Way Huang
Department of Electrical, Computer Engineering and Technology
Minnesota State University, Mankato, MN 56001

Abstract

Software programming is often considered to be difficult for many engineering students. Nowadays, many control and automation systems are facing the increasingly sophisticated functional and non-functional demands. In such systems, software portion is always expected to have the greater impact. Therefore, educators continue to face great challenges in getting students to be capable of conducting efficient software development. In the last decade, model-based design (MBD) is an emerging development methodology for modern software. Its efficiency has been demonstrated in large scale software engineering projects. This paper presents our experience of integrating modern MBD concepts and tools into a Programming Tools (PT) course. First, the basic components in the MBD process are exposed to students, especially its two unique components - automated code generation and model-based verification and validation (V&V). Second, three modeling languages: Matlab/Simulink, LabVIEW and SCADE are exposed to students. They all have been widely applied in embedded control and automation domains. Third, input programming languages of these selected tools are introduced to students to help them apply the tools in the laboratory assignments and class project.

Introduction

Knowledge of computing and software programming is important to all engineering and technology students. The US Bureau of Labor Statistics predicts that computing will be one of the fastest-growing U.S. job markets in STEM through 2020: about 73% of all new STEM jobs will be computing related [1]. More importantly, software development training could be a valuable experience for all engineer students, as it can cultivate student' problem solving and process development capability.

However, software programming is often considered to be difficult for engineering students. Engineering students usually study the syntax and semantics of low-level programming languages (PL) such as C or assembly in one or two semesters. They have fewer opportunities to apply the learnt programming skills compared with computer science or software engineering students. It is common for engineering students to forget the syntax of C language. When a class project involves software programming, students often spent a large amount of time in debugging syntax and semantics errors, with little time left for algorithm development and verification. Some engineering students consider writing a small program with 300 to 500 lines of code as a painful experience. And a large percentage of the junior or senior design projects that could not be accomplished on time are due to the prolonged software implementation stage.

Model-based design (MBD) is an emerging methodology for developing complex software, especially embedded software. Its efficiency has been demonstrated in software engineering. For example, the Matlab/Simulink language from MathWorks that supports MBD has become the predominant software modeling language in many motion controls, aerospace and automotive applications. By promoting the use of domain-specific notations to graphically represent specifications and designs, MBD can identify design flaws at the early stage and avoid costly design fixes during the late stage. The implementation of the software system is either generated or derived manually from high-level models. Multiple large EU-funded research projects have been initiated to promote the application of MBD in industry, and target at solving the challenges encountered in different real-world application domains [2, 3, 4].

This paper presents our experience of integrating the MBD knowledge into a Programming Tools (PT) course. This course is an elective for junior and senior computer engineering or electrical engineering students. Before taking this course, students have already had some programming experience. They have already learned to enter, compile, run, test, and debug programs. The objectives of the course include teaching students modern programming tools, and their usage in the design and implementation of electronic control systems or special-purpose digital systems such as digital signal processing, etc. Traditionally, C/C++ was the programming language used by students to write programs to control or augment hardware, or perform numerical analysis in the course. LabVIEW as a graphical programming tool was also exposed to students. To improve this PT course, we add three MBD related topics: (i) MBD workflow especially the automated code generation and model-based verification. (ii) Three programming tools that support MBD: Matlab/Simulink from Mathworks, LabVIEW from National Instruments and Scade by Esterel Technologies; and (iii) Tools that support programming with these languages are introduced to students and applied in the laboratory assignments and class projects.

Course Description

Background

Programming Tools (or with the similar name) is usually a required course for computer engineering or computer science major students, but an elective course for other engineering major students in many universities. It can be offered at the introductory level or the system level. At the introductory level, the PT course typically emphasizes the basic methodology and tools supporting program compiling, linking, test, debug and source code management [5]. The PT course at the systems level often focuses on key concepts of low-level programming and explicit memory management (e.g., C or C++); tool chains for group software development; and advanced topics on software system design, implementation, testing strategies and documentation [6].

The PT course presented in this paper is closer to the systems level. It is organized as 2 hours of lecture and 2 hours of laboratory per week. At the end of the course, students are capable of utilizing existing programming tools to develop a complete hardware/software embedded system. Such a system is required to consist of three basic functions: (i) collect and store data inputs from a variety of sensor devices which are sensitive to the external environment, (ii)

perform analysis and control algorithms, and (iii) control actuators to react to certain scenarios correctly and timely with respect to the requirement specification. In this course, C/C++, an imperative paradigm programming language, was solely chosen for coding. In many cases, after initiating the course project, students quickly move to the implementation stage after a brief design phase, and start the C programming and debugging iterations using an IDE. Although this approach works for the small-scale course project, students have reported that it is very time consuming and inefficient. And the behavior of the created system often deviates from the original design plan. Educators have recognized the need to introduce some efficient and cost-effective programming tools to students [7]. The goal is to equip students with the knowledge for developing complex engineering systems with a large number of constraints.

Experts in the software engineering and computer science communities advocate introducing the MBD methodology to students. It provides students with the insights, techniques and tools to alleviate the difficulties of developing complex software systems. Educators have either integrated MBD into the existing software design course [8] or proposed a new project-based course to solely teach MBD [9]. However, as these courses are mainly for computer science or software engineering students, their contents are too theoretical for engineering students who have limited software development background.

The intent of the PT course presented in this paper is to convey the practical instead of theoretical knowledge related to programming to students. We added materials on MBD from the engineering practitioner's point of view to the course with three objectives in mind. The first is to improve students' awareness of the advanced MBD methodology. The second is for students to develop an appreciation for the MBD that will contribute to the efficient and cost-effective application development. The third is to give students the opportunity to learn modern programming tools enabling MBD. The following subsections present the three MBD topics added to the PT course, with the emphasis on the teaching approach and lab assignment design.

Model-based Design Concept

We introduced key MBD concepts that are important for an engineering practitioner to our students during the first week. Five basic steps in MBD approach from requirement analysis, system design, implementation, integration to continuous verification, are covered. Based on the MBD process illustrated in Figure 1, we discussed the differences between the MBD and the conventional software development processes like waterfall model to encourage active learning. For instance, without being taught, students can summarize by themselves that testing and verification is conducted continuously during each of the other four steps, not until their completion. Students are also guided to learn new concepts along each basic MBD step. Using System Design step as an example, students learn the concept of Executable Specification (in terms of models shown in Figure 1). It can unambiguously model the entire system functionality, including the environment, physical component and design algorithm. The created models have the benefits of improving communication and collaboration in the development team via sharing of models, and supporting early validation and testing via models simulation.

After introducing the basic concepts of MBD, we taught automated code generation and model-based V&V. Both can contribute to the improvement of the design, implementation and verification of safety-critical and security-critical embedded software.

Automated code generation

The use of automated code generation tools has been increasing in the last ten years. This is mainly because they help engineers faster and better develop documented software in comparison to hand coded development. It has two outstanding advantages: (i) Eliminate errors from hand-coding; (ii) Regenerate easily for different targets. Many engineers with limited programming experience could be greatly relieved from low-level programming, and focus on domain-specific problems. Moreover, depending on the application purposes, system design models are synthesized to different implementation languages. For example, programs coded in Structured Text (a language) are generated for the PLC applications; VHDL or Verilog code is generated for the models of the FPGA or ASIC. For MCU control or DSP application, the models are translated to C/C++.

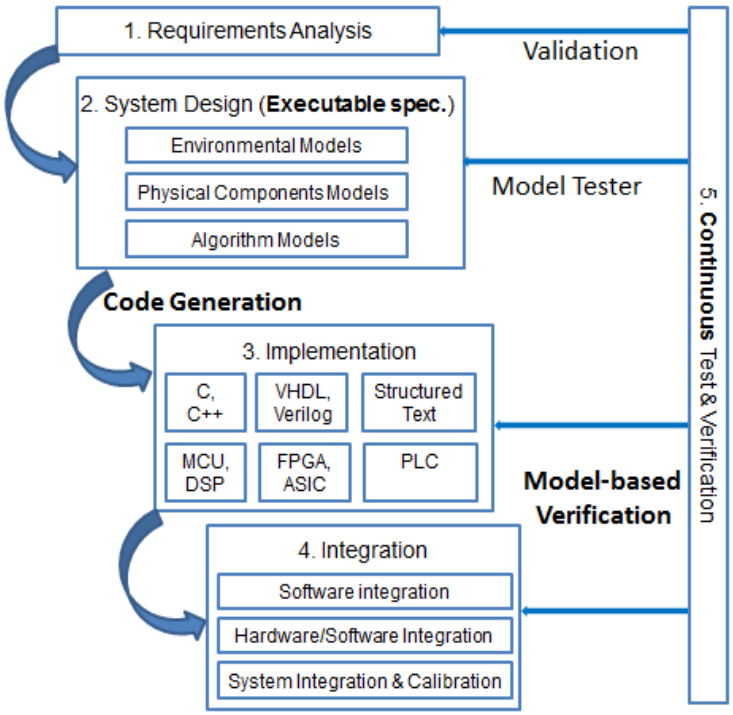

Figure 1 Main steps in Model-based design process

The purpose of formal verification of the specification models is to ensure the correctness of the design. Most high-level PLs like Matlab/Simulink are easy to use, but lack of rigorous semantics. Some researchers have investigated the automated transformation from Simulink to a formal language like Lustre, and applied the existing formal verification tools for checking the Lustre programs to formally verifying Simulink models [10].

To give students a direct experience of automated code generation, the C code generated by a commercial tool Simulink Coder™ (formerly Real-Time Workshop®) and an open-source tool Gene-auto [11] are exposed to students. First, the comparison of the generated code is discussed in class. The C code generated from Simulink Coder is complex and hard to read. This is mainly due to the additional code injected into the C Code for the performance optimization or debugging purposes. The resultant code can be used for real-time and non-real-time applications, including simulation acceleration, rapid prototyping, etc. In contrast, the C code derived by Gene-auto could be clean and easy to trace back to the corresponding Simulink model blocks. The C code translated by Gene-auto is mainly for program verification. Thus, the focus is on the functional correctness rather than the execution performance. Next, the C code generated by the Gene-auto tool from Simulink design models is further studied for students to understand the details of automated code generation mechanism. Two translation examples are selected for case

study: (i) the translation of basic logical operation blocks, arithmetic operation blocks and simple subsystems composed of these two kinds of blocks to C functions or statements; (ii) the translation of states and transitions in the Stateflow charts to C functions. For example, an addition "+" block with three inputs and one output could be translated to the C statement "o1 = in1 + in2 + in3". Students are asked to prepare a class report on the comparison of different code generation tools. Some relevant research papers were also offered to the students who want to explore this topic further [12]. This topic not only helps students understand automated code generation mechanism, but also convinces students the great enhancements led by the MBD approach in system development for engineers.

Model-based validation and verification (V&V)

Model-based V&V represents a set of V&V techniques continuously applied through the MBD. All of them contribute to three important goals/benefits: (i) Detect errors early in development; (ii) Reuse test throughout development process. (iii) Reduce use of physical prototypes. In this course, model-based V&V techniques/tools in three areas are provided to students. First, conventional quality control techniques in software engineering [13, 14] are recapped and compared. Validation targets at answering the question "Are we developing the *right* system?", while verification aims at answering a different question "Are we developing the system *right*?" Formal methods and testing are two most popular approaches for verification. In mission-critical systems, where bugs may incur disastrous effects, formal methods are employed to guarantee the correct behavior with respect to the safety-critical requirement. In comparison, testing is scalable and easy to apply although it is limited to detect the bugs in a system, but cannot ensure the correctness. Unit testing, integration testing and system testing are three common testing practices in software systems development. The purpose is for students to clarify the typical usage and differences of these techniques.

Second, V&V techniques applied at different MBD steps are discussed in class. During the initial requirement analysis step, a validator is applied to ensure that the extracted requirements correctly match the intended use. In the system design, a model tester or a simulator can be utilized to check whether Executable Specification satisfies the requirements obtained in the initial step. Unit testing is typically applied to check if the implementation coded in some low-level languages is consistent with the design models. Integration testing and system testing are initiated from the integration step. Formal methods are applied to check critical components in the implementation. A translation validation tool, which formally verifies the translation from Simulink models to C, is introduced to students [16].

Third, the latest advances of model-based testing MBT in both academy and industry are exposed to students. This is one of our new teaching endeavors in integrating an on-going research results into the advanced level or graduate level courses.

MBD tools

In the past, this PT course introduces some graphical programming tools used by electrical, control or automation engineers in the real-world [17], like LabVIEW to students. Although some of these tools have been extended to support the MBD approach, they were offered as the

replacement of text-based programming only. Nowadays, there are many MBD tools available from both commercial and research communities. Some tools are designed with specific application in mind. For example, a survey of MBD tools used in the User Interface Design area has been reported [18]. Most students taking the PT class have some background in embedded systems, such as digital hardware, digital signal processing, or automation control. Thus, we are particularly interested in a class of MBD tools which have been used in the embedded systems area. Table 1 show a list of tools which are either open-source or free for education.

Table 1. A list of MBD tools

Tools	**Vendor**	**Models**	**Code Generation**	**Brief description**
LabVIEW	NI	"G", a visual PL	C/C++ etc.	A system-design platform and development environment for dataflow and graphical programming widely used in data acquisition and embedded control and monitoring applications. http://www.ni.com/labview/
Simulink	Mathworks	Simulink	C, Verilog, Structured text	A block diagram environment for multi-domain simulation and MBD of dynamic systems, including a graphical editor and customizable block libraries. http://www.mathworks.com/products/simulink/
Scade	Esterel Tech	Lustre	C	A Lustre-based IDE for designing safety critical embedded software applications in reactive systems, and building formal models. http://www.esterel-technologies.com/products/scade-suite/
Ptolemy II [19]	Berkeley	Variety of models	Java, NesC	A framework for hierarchical heterogeneity, i.e. heterogeneous modeling, simulation and design of concurrent embedded systems. http://www.ptolemy.eecs.berkeley.edu/ptolemyII/
eTrice	Eclipse	ROOM[i]	Java, C, C++	An Eclipse project for embedded Model Driven Software Development based on ROOM. http://www.eclipse.org/etrice/
TOPCASED	Airbus	SysML[ii], SAM[iii], AADL[iv] UML	Java, C, Python	Eclipse based software environment dedicated to the realization of critical embedded systems; supporting formal checking. http://www.topcased.org/
Rational Rhapsody Designer	IBM	SysML, UML	C, etc.	A model-based system engineering environment included in IBM Rational Rhapsody family. http://www.ibm.com/developerworks/rational/products/rhapsody/
EZRealtime	UFAM/EMF	PNML[v]	C	A MDE-based tool based on timed Petri Net formalism for developing embedded and real-time systems. http://code.google.com/p/ezrealtime/

Tools	Vendor	Models	Code Generation	Brief description
UPPAAL	Uppsala Univ. & Aalborg Univ	Networks of timed automata	Real-time Java, etc.	An integrated tool environment based on networks of timed automata for modeling, V&V of real-time systems. http://www.uppaal.org/
Qsys/SOPC Builder	Altera	IP functions, Subsystems	Verilog, VHDL	An integration tool for the hierarchical FPGA design by automated generation of interconnection logic and HDL code. http://www.altera.com/

In this PT course, a survey of programming tools that support MBD, as shown in Table 1 is first introduced to students. Students are required to search for a new tool and prepare a 10-min presentation to introduce the tool they found to the whole class. The first three tools in the table are exposed to students. First, these three tools can support each step of the MBD process discussed in the previous subsection, including algorithm design, architecture design, system-level design, model edit, simulation, automated code generation, continuous V&V, code debugging and deployment, system integration, etc. Second, they are industry supported and have been widely used in the embedded systems development in the real-world. It is important to equip students with such knowledge, especially after they graduate and enter the workplace. Third, they have well-documented tutorials and user manual, and even exercises. These resources are very helpful for the lecture notes preparation and assignments design. Fourth, these tools include rich example projects from simple to complex scale. They are very useful references for designing student projects. In order to advocate active learning, pre-lab assignments are made in this course. Hands-on learning through class projects is highly encouraged and will be the basis for instruction in this course. So far, students have developed and completed 3 capstone projects. Students can get long-time benefits from the efforts they make in "playing with" these tools, especially in terms of improving their work efficiency.

Three Input Programming Languages

Compared with low-level text-based programming like C/C++, the graphical programming with these three MBD tools is much easier to start with. But, the efficient programming with high-level PLs is still a very demanding task. Besides a large amount of programming practices, it is also important to understand the library functions provided by the tools. In this paper, the primary experience of teaching design libraries is presented in the following.

Simulink: A set of commonly used block types belong to a variety of Simulink libraries. So, ten tool-provided libraries are introduced to students: Math, Logic and Bit, Sinks, Sources, Ports & Subsystems, Discrete, Discontinuities, Signal Routing, Signal Attributes and Model Verification. Most students find that basic block types included in the first four libraries are relatively simple. They can learn these blocks' specifications in a short time and quickly start doing exercises and lab assignments. The Ports & Subsystems library is reported being most difficult, at the same time the block types included in it are the most important to construct the complex and hierarchical designs. Thus, this library is the focus of our teaching. The block types belonging to the remaining five libraries are generally not complicated in terms of functionality, but the usage

they provide seldom occur in the text-based programming. Thus, it takes one lecture to introduce students these libraries.

LabView: To equip students with the new dataflow programming knowledge of Labview, we focused on event-driven programming and the Programming category in the Functions Palette. First, event-driven programming is greatly needed in designing real-time applications. Different from conventional procedure-oriented programming, the execution flow is determined by events. The event structure of LabVIEW and common types of static or dynamic events are introduced. From the lab exercises grading, we find that students usually find event-programming difficult at first, either missing relevant events or taking a wrong action with respect to an event. After students get familiar with this new programming paradigm, most of them find it very useful in building embedded systems with timing constraints.

Lustre/Scade: The main motivation of introducing Lustre is to help students be aware of a formal PL in MBD. The main instruction approach is using examples, instead of teaching its formal semantics, which is too overwhelming for non-CS students. The transformation from Simulink to Lustre and the supporting tools are also provided to students.

At the end of the semester we did formal course evaluation and student outcomes assessment. In summary, students commented that the automated code generation and MBD tools taught in this course are most beneficial to them. For example, some students, who have used some functions of LabVIEW before taking this course, reported that this course gave them wider and much deeper understanding of this software tool in system development. We will accumulate more student feedbacks each year and keep track of how this course may help graduates in their practical work.

Conclusion

MBD is cost effective for developing complex and reliable-critical embedded systems. This paper presents our teaching experiences of integrating this new MBD paradigm into a system-level Programming Tools course for CE and EE students. It mainly describes three new topics added to this PT course: MBD concepts, three common MBD tools and three input programming languages of these tools, especially from two aspects of course materials preparation and instruction approaches. In the future, students and our teachers will together create and gather more capstone projects related to MBD.

References

1. Link to US bureau of Labor Statistics: http://www.bls.gov/emp/ep_table_102.htm, a related Link to the market for computing careers: http://cs.calvin.edu/p/ComputingCareersMarket
2. EU CESAR project (Cost-Efficient Methods and Processors for Safety Relevant Embedded Systems) http://www.cesarproject.eu/
3. EU MOGENTES project (Model-based Generation of Tests for Dependable Embedded Systems) http://www.mogentes.eu/
4. SESAME project (A Model-driven Test Selection Process for Safety-critical Embedded Systems) http://wiki.lassy.uni.lu/projects/SESAME

5. Aleman, J.L.F., "Automated Assessment in a Programming Tools Course," *Education, IEEE Transactions on*, vol.54, no.4, pp.576-581, Nov. 2011.
6. Links to some system-level PT courses: http://www.cs.washington.edu/education/courses/cse374/; http://web.eecs.utk.edu/~huangj/cs360/ http://school.eecs.wsu.edu/undergraduate/cpts/courses/360
7. Paul G. Flikkema. "Approaching the Design of Complex Engineered Systems: A Model-based Approach Informed by System Thinking". *Proceedings of ASEE PSW Conference*, 2012.
8. Peter J. Clarke, Yali Wu, Andrew A. Allen, and Tariq M. King, "Experiences of Teaching Model-driven Engineering in a Software Design Course", *ACM/IEEE Intl. conference on Model Driven Engineering Language and Systems*, Oct. 2009.
9. Mireille Blay-Fornarino. "Project-based teaching for Model-Driven Engineering", in *Proceedings of the Promoting Software Modeling through Active Education*, pages 69-75, Sept 2008.
10. Joshi, A., Heimdahl, "Model-based safety analysis of Simulink models using SCADE design verifier". *Proceedings of Computer Safety, Reliability, and Security (SAFE-COMP)*. Volume 3688 of LNCS, Springer (2005).
11. Gene-auto project. http://geneauto.gforge.enseeiht.fr/
12. Coelho da Silva Stanisce Correa, G., da Cunha, A.M., Vieira Dias, L.A., Saotome, O., "A comparison between automated generated code tools using model based development," *Digital Avionics Systems Conference (DASC), 2011 IEEE/AIAA 30th* , vol., no., pp.7E4-1-9, Oct. 2011.
13. G. J. Myers, C. Sandler, T. Badgett, and T. M. Thomas, "*The Art of Software Engineering*", Hoboken, NJ:Wiley, 2004.
14. R. Pressman, "*Software Engineering: A Practitioner's Approach*", New York: McGraw-Hill, 2009.
15. Bran Selic, "The Pragmatics of Model-driven Development", *Software, IEEE*, vol.20, no.5, pp.19-25, Sept.-Oct. 2003.
16. O. Strichman, M. Ryabtsev, "Translation validation: from Simulink to C", *Proceeding of Intl' Computer Aided Verification conference*, pp 696-701, 2009.
17. Eduard Lunca, Silvlu Ursache and Oana Neascu, "Graphical Programming Tools for Electrical Engineering Higher Education", *International Journal of Online Engineering*, Vol 7, No 1, 2011.
18. J. L. Perez-Medina, S.D. Chessa, A. Front. "A survey of Model Driven Engineering Tools for User Interface Design". *Proceedings of Intl. conference on Task models and diagrams for user interface design*, pp 84-97, 2007.
19. Johan Eker, Jorn Janneck, Edward A. Lee, Jie Liu, Xiaojun Liu, Jozsef Ludvig, Sonia Sachs, Yuhong Xiong, Stephen Neuendorffer. "Taming heterogeneity - the Ptolemy approach". *Proceedings of the IEEE*, 91(1):127-144, 2003.

[i] ROOM: Real-Time Object-Oriented Modeling
[ii] SysML: Systems Modeling Language
[iii] SAM: System Architecture Modeling
[iv] AADL: *Architecture* Analysis and Design Language
[v] PNML: Petri Nets based Model Language