Tsmart-BIPEX: An Integrated Graphical Design Toolkit for Software Systems

Huafeng Zhang[1], Yu Jiang[1], Han Liu[1], Ming Gu[1], and Jiaguang Sun[1]

School of Software, Tsinghua University, China

Abstract. To help build reliable software systems efficiently, the component based model-driven design approach is widely used, and lots of modeling languages have been designed. In this paper, we propose an integrated graphical development toolkit **Tsmart-BIPEX** [1] in support of building complex systems in the **BIP** modeling language, which features a rich semantics for composing sub-systems. First, we build a graphical interface for model construction and simulation, which is more intuitive than the command-line based toolchain. Furthermore, to enhance the original **BIP** verification tool **RTD-Finder**, we translate the graphical model to a labeled transition system for a thorough verification in verifier **VCS**. Finally, we can generate executable C++ code directly from the graphical model. **Tsmart-BIPEX** has been successfully applied in the development of a real-world train network controller.

1 Introduction

As embedded devices gain more and more computing power, the systems on these platforms are expected to accomplish much more sophisticated tasks, which brings new challenges to ensure the correctness of embedded softwares in the traditional development approaches [11]. According to the report from National Institute of Standards and Technology, 70% of faults are introduced in the forepart of the life-cycle, and 80% of them are not discovered until integration and system test or later, which leads to 10X or higher repair cost [14].

A promising way to alleviate the pain of detecting design defects in later phases is the component-based model-driven design approach [9, 10]. The key idea is to build a structural model that decompose a complex system into subsystems with independent functions, then provide coordination semantics to assemble these sub-systems together, and finally synthesize executable code from the verified model. During the last decades, lots of component based modeling languages such as **safety state machine**[1], **stateflow**[6], and **BIP**[2] have been proposed and widely used with their corresponding supporting toolkits.

Among them, **BIP** features its semantics with the support of complex interaction and dynamic priorities which can be further used to express complex scheduling policies. It brings **BIP** strong expressiveness that cannot be matched by other modeling languages, but also results in the limited verification support

[1] Demo Video: `https://youtu.be/xNd6N7DJC-s`
Tool Download: `https://github.com/ronhuafeng/tsmart-bipex`

of the tool **RTD-Finder** [3], where lots of temporal properties can not be verified. More complex semantics is harder to be formalized for a comprehensive verification. Furthermore, during the real engineering practice, we found that **BIP** is a textual modeling language based on a command-line based toolchain for model simulation and synthesis, which is inconvenient for model construction and interactive debugging.

In this paper, we improve the convenience and verification efficiency of **BIP** by developing an integrated graphical design toolkit **Tsmart-BIPEX**. The overall structure of the toolkit is shown in Fig. 1. First, graphical model editor and simulator are implemented to assist the engineers according to the **BIP** syntax and semantics. Then, a translator is developed to translate the model to an equivalent labeled transition system, which can be directly verified by the **VCS** model checker. Finally, the original code generator is seamlessly integrated for C++ code generation. While the original supported compiler and engine are only provided on Linux with a command-line interface for textual model construction, **Tsmart-BIPEX** can be installed on Mac-OSX, Linux, and Windows.

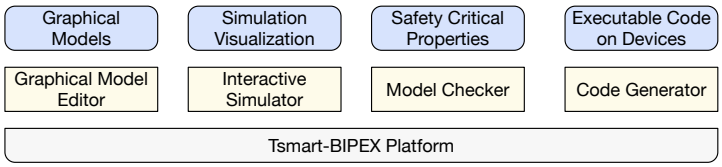

Fig. 1. Structure of the integrated graphical design toolkit Tsmart-BIPEX.

Related work is presented in Section 2. Some backgrounds on the computation model **BIP** are presented in Section 3. The implementation of **Tsmart-BIPEX** is introduced in Section 4. A case study of a real-world system design is presented in Section 5, and we conclude with Section 6.

2 Related Work

During the last decades, lots of component based modeling languages such as **safety state machine**, **statechart**, **stateflow**, and **BIP** have been proposed for the modeling of complex systems. Based on those modeling languages, many design toolkits have been implemented, such as **SCADE**[5], **Statemate**[7], **Simulink Stateflow**[4] and **RTD-Finder**. These toolkits have been successfully applied in both academic and industrial applications.

For example, **SCADE** is a development suite for building safety-critical embedded systems based on the synchronous modeling language **safe state machine**, and is widely used in safety-critical applications such as avionics and train control. **Simulink Stateflow** is a modeling and simulation platform based on the event-driven modeling language **Stateflow**, which highlights its tight integration with Matlab computing environment. It also provides numerous toolboxes such as **Design Verifier** and **Polyspace** for model construction and synthesized code analysis. **Statemate** developed by IBM is a working environment for the development of complex reactive software based on the reactive modeling

language **Statechart**, where the software can be modeled and synthesized from a set of parallel and synchronized automata.

Except for those famous but expensive industrial tools, there are also many academical prototypes for system design such as **Ptolemy**, **LabView**, **BIP** and **ForSys**. They add new modeling and analysis features to the original industrial tools. For example, **Ptolemy** supports the design of heterogenous systems. **BIP** features its semantics with the support of complex interaction and dynamic priorities which can be further used to express complex scheduling policies, which cannot be matched by other modeling languages. In our work, we try to improve the convenience and verification efficiency of **BIP** by developing an integrated graphical design toolkit **Tsmart-BIPEX**.

3 Background

In **BIP**, a system is specified in textual description with the composition of components and the interaction between components. There are two types of components, where the atomic components are a class of components which specify the behavior of independent sub-systems in labeled transition graphs, and the compound components schedule the communication between atomic components by defining interactions connecting atomic components and priorities of the interactions. The textual model can be compiled, validated and synthesized with command-line support on Linux system.

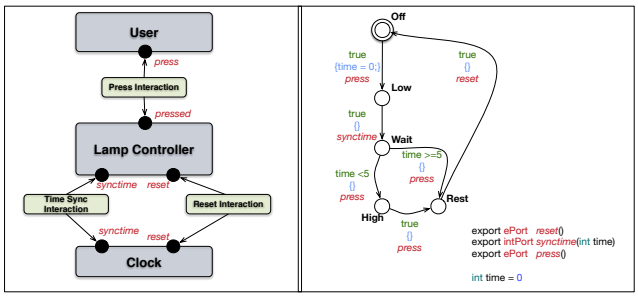

Fig. 2. A visual example of BIP model, including the compound and atomic component, interactions, ports, states, transitions with actions and guards, local variables.

We depict the main features by an example constructed in the graphical interface of **Tsmart-BIPEX** in Fig. 2. This model represents how a user controls the brightness of a lamp. The left part shows the top-level model with three components: User, Lamp Controller, and Clock. Components communicate with each other through interactions. The interaction between User and Lamp Controller is *Press Interaction*. The interactions between Lamp Controller and Clock are *Time Sync Interaction* and *Reset Interaction*. Interactions and components are connected via ports (the black dots attached to components). When an interaction is fired, it triggers the computation of its connected components via related ports. The right part shows the detailed behavior of the Lamp Controller by an automaton. To obtain a full description for **BIP**, please refer to [2].

4 Tsmart-BIPEX Implementation

The overall structure of **Tsmart-BIPEX** is presented in Fig.1, and is implemented on the Eclipse Rich Client Platform. The whole project contains 74,719 lines of Java codes and 2,358 lines of Clojure code, where 36,278 lines are inherited from Tsmart-GalsBlock[12] which contributes to the common graphical user interface and pure hardware design. The four components contained in the toolkit are described as follow.

Graphical Model Editor. The graphical model editor is based on Eclipse RCP and Eclipse Graphical Modeling Framework which implements the **BIP** syntax defined in [2]. The editor supports the graphical creation and editing of compound & atomic components and interactions and its graphical interface is shown in Fig. 3. The editor's four views exhibits how the sub-components and interactions in a compound component are created. (1) The project view lists the models contained in the current project. (2) The editor view shows a canvas where the selected model's diagram can be edited. (3) The palette view provides a bundle of predefined components and interactions which can be dragged into the editor view as a template for the creation of a new model element. (4) The model structure view displays the hierarchical structure of the current model in editing. The interface for editing atomic components provides similar view layout with elements for automata construction, as presented in Fig.2.

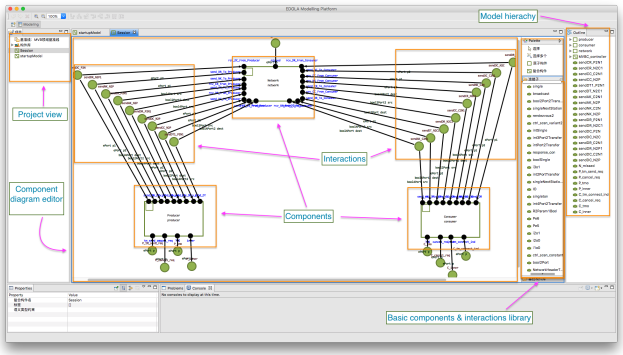

Fig. 3. Graphical editor interface for component and interaction construction.

Graphical Model Simulator. The graphical simulator is based on the execution semantics of **BIP** and provides interactive simulation of the constructed graphical model. In each step, an interaction or component is selected according to their state and execution priority, and the computation inside this candidate is executed. The graphical simulator helps users explore the system's behavior in a through way. Some basic functions are provided by the simulator, including model states traversal and execution traces recording, etc.

The simulation interface includes five main views. (1) The simulation action toolbar includes initialization, step-forward, step-backward and reset actions. (2) The canvas view visualizes the simulated model in the same layout as the editor, with the enabled components or interactions highlighted for choice. (3) The candidate view displays all candidates (enabled components or interactions)

with the highest priority in the model's current state, whose inner computation will be triggered if this element is selected. Users can manually select one candidate to trigger by double-clicking the corresponding item or let the simulation engine selects one randomly. (4) The data view lists values of local variables of every component in the current state. (5) The model structure view shows the hierarchical structure of the simulated model.

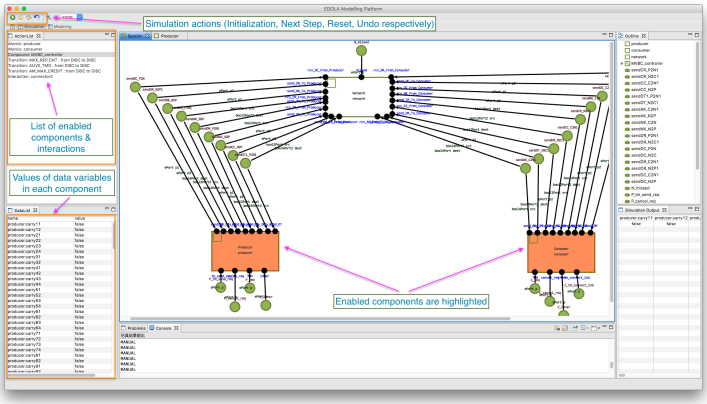

Fig. 4. Graphical simulator interface for the interactive model simulation,

Model Checker. The model checker is **VCS** [8], which takes the labeled transition system and corresponding properties written in computation tree logic formula as input, and export the verification output with optional counterexamples. Although **RTD-Finder** supports the verification of **BIP**, its computation efficiency for global invariant is not as good as **VCS**. For integration, we develop an engine to translate the graphical **BIP** model to a semantics equivalent labeled transition system, which can be verified by **VCS** directly. The formal verification complements the simulation to provide stronger support for correctness.

Code Generator. The code generator for **BIP** is integrated based on the source-to-source transformation architecture introduced in [2], which allows the generation of C++ code from **BIP** models. The generated source code can be executed on an embedded platform consisting of a **BIP** engine to schedule the computation of components and interactions. The original code generator engine is very good, and we integrate it into the graphical design environment. The user just needs to click the code generation button, and the code would be generated.

5 Experimental Results

We have applied **Tsmart-BIPEX** in the component-based model driven development of a network interface controller to evaluate the effectiveness of the graphical toolkit. Network interface controller is an embedded device widely used for the communication in several kinds of vehicle buses. The key component of the network interface controller is the multi-function vehicle bus controller (MVBC), which confirms with the protocol defined in IEC 61375 [13].

First, strictly following the description of the standard IEC 61375, we build a top-level model for the network layer of an MVBC device in **Tsmart-BIPEX** as shown in Fig. 3. The top-level model of the network layer contains three components: the producer of a message, the consumer of a message, and the router which transmits a message from the source device to its destination device. The router is modeled as an atomic component abstracted in Fig. 5, which extracts the source device address and the destination device address from the processed message and composes a new message to send.

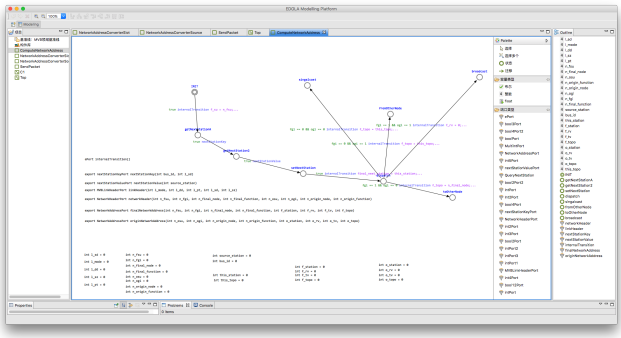

Fig. 5. Atomic component of the router function of the vehicle bus controller.

After several rounds of graphical model simulation, the MVBC model is automatically translated to an equivalent labeled transition system for **VCS** verification. We check several key requirements on the model, such as"*the message should not be sent to the link-layer if the source device address equals the destination device address*". For the violated property, we need to check and revise the constructed model manually based on the counter example. This verification process strengthens the confidence in the constructed **BIP** model of MVBC.

When all properties pass the verification, we can automatically generate C++ code from the verified **BIP** model. The generated C++ source files contain 2,537 lines of code, including 623 lines of the BIP engine code to coordinate the components and interactions. Also, we write 146 lines of interface code to handle the embedded device's I/O communications. All codes are compiled by the arm-gcc-3.8 compiler to get an executable file running on the ARM processor of the MVBC device. Then, we connect the automatically synthesized MVBC device with the existing well-tested hand-written MVBC device, the communication between the two devices functions well.

6 Conclusion

In this paper, we present an integrated graphical development toolkit **Tsmart-BIPEX** to support the widely used **BIP** modeling language. With the integration of the graphical model editor interface and the model simulation interface, the convenience is greatly raised compared to the original textual model construction and command-line based simulation. By translating the **BIP** model to

an equivalent labeled transition system, the verification efficiency is increased compared to the original **RTD-Finder**. Our future work may focus on the development and integration of more efficient code generation algorithms.

7 Acknowledgment

This research is sponsored in part by NSFC Program (No. 91218302, No. 61527812), National Science and Technology Major Project (No. 2016ZX01038101), Tsinghua University Initiative Scientific Research Program (20131089331), MIIT IT funds (Research and application of TCN key technologies) of China, and The National Key Technology R&D Program (No. 2015BAG14B01-02).

References

1. André, C.: Semantics of safe state machine. I3S Laboratory 6070 (2003)
2. Basu, A., Bozga, M., Sifakis, J.: Modeling heterogeneous real-time components in bip. In: Fourth IEEE International Conference on Software Engineering and Formal Methods (SEFM'06). pp. 3–12. Ieee (2006)
3. Bensalem, S., Bozga, M., Nguyen, T.H., Sifakis, J.: D-finder: A tool for compositional deadlock detection and verification. In: International Conference on Computer Aided Verification. pp. 614–619. Springer (2009)
4. Dabney, J.B., Harman, T.L.: Mastering simulink. Pearson/Prentice Hall (2004)
5. Dormoy, F.X.: Scade 6: a model based solution for safety critical software development. In: Proceedings of the 4th European Congress on Embedded Real Time Software. pp. 1–9 (2008)
6. Hamon, G., Rushby, J.: An operational semantics for stateflow. In: International Conference on Fundamental Approaches to Software Engineering. pp. 229–243. Springer (2004)
7. Harel, D., Lachover, Hagi, e.: Statemate: A working environment for the development of complex reactive systems. IEEE transactions on software engineering 16(4), 403–414 (1990)
8. He, F., Yin, L.e.: Vcs: A verifier for component-based systems. In: Automated Technology for Verification and Analysis. pp. 478–481 (2013)
9. Jiang, Y., Li, Z., etc: Design and optimization of multiclocked embedded systems using formal techniques. IEEE Transactions on Industrial Electronics 62(2), 1270–1278 (2015)
10. Jiang, Y., Liu, H., etc: Design of mixed synchronous/asynchronous systems with multiple clocks. IEEE Transaction on Parallel and Distributed Systems pp. 2220–2232 (2015)
11. Jiang, Y., Song, H., etc: Data-centered runtime verification of wireless medical cyber-physical system. IEEE Transactions on Industry Informatics (2016)
12. Jiang, Y., Zhang, H.e.: Tsmart-galsblock: a toolkit for modeling, validation, and synthesis of multi-clocked embedded systems. In: Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering. pp. 711–714. ACM (2014)
13. Schifers, C., Hans, G.: Iec 61375-1 and uic 556-international standards for train communication. In: Vehicular Technology Conference Proceedings, 2000. VTC 2000-Spring Tokyo. 2000 IEEE 51st. vol. 2, pp. 1581–1585. IEEE (2000)
14. Tassey, G.: The economic impacts of inadequate infrastructure for software testing (2002)