Developing Reliable Component-Based Software in *Mediator*

Yi Li(✉)

LMAM and Department of Informatics, School of Mathematical Sciences, Peking University, Beijing, China
liyi_math@pku.edu.cn

Abstract. Component-based development is widely used to reduce the development cost of complex systems. In this pattern, software features are organized, encapsulated and reused as components. In this report, we present a component-based modeling framework based on the modeling language *Mediator* that aims to build formally verified software, both on model-level and code-level. This work is the core part of a Ph.D. thesis.

Keywords: Component-based · Modeling language · *Mediator*

1 Introduction

Modern software systems are becoming more and more complex. To simplify the development phase, software developers encapsulate the features in smaller components that are easier to be developed and tested. The correctness of components are important since they are often reused by other software, hence any small vulnerabilities may lead to dozens of potential bugs. In this report, we present a formal modeling and code-generation framework based on *Mediator* where *Mediator* is a new modeling language proposed in [12]. With this framework, we can easily design high-level models and specify their properties, automatically generate runnable codes and verify both of them.

Part of this work has been published, including the modeling language *Mediator* and its formal semantics [12], and a code generator to C language [13]. We have also built a model checking integration with help of NuSMV [5] and another code generator to System C. These two works have been developed but still unpublished.

2 Related Work

Component-based software engineering has been prospering for decades. Currently, there are various tools, both formal and informal, that supports component-based modeling. For example, NI LabVIEW [16], MATLAB Simulink [7] and Ptolemy [9] provide powerful modeling platforms and a large number of built-in component libraries to support commonly-used platforms.

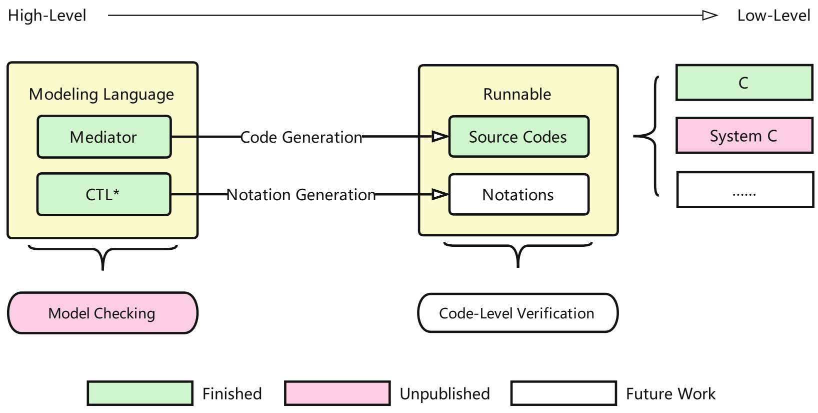

Fig. 1. The reliable development framework based on *Mediator*

However, due to the complexity of models, such tools mainly focus on synthesis and simulation, instead of formal verification. There is also a set of formal tools that prefer simple but verifiable model, e.g. Esterel SCADE [1] and rCOS [14] (Fig. 1).

In the recent years, formal method has shown its power in industrial use [8,10,15]. These works proved that formal verification techniques are capable of handling large-scale component-based embedded systems. However, the unfamiliarity of formal specifications is still one of the main obstacles hampering programmers from using formal tools. For example, even in the most famous formal modeling tools with perfect graphical user interfaces (like UPPAAL [2] and PRISM [11]), sufficient knowledge about automata theory is necessary to properly encode the models.

Importance of code generation has also been uncovered for a long time. A large number of formal and industrial code generation tools have been built for different target platforms. For example, Rodin for Event-B [4] and SCADE [3] are very popular formal tools that can generate executable codes from abstract models.

3 *Mediator*

Mediator is a component-based modeling language [12], which provides proper formalism for both high-level *system* layouts and low-level *automata*-based behavior units. Both automata and systems are encapsulated with *a set of input or output ports* (which we call an *interface*) and *a set of template parameters* so that they can be easily reused in multiple applications.

Mediator is designed to serve both software engineers and formal researchers. On the one hand, the behavior of automata is captured by guarded transitions, whose semantics is clear and self-contained. On the other hand, interfaces of automata and systems are precisely defined by ports and their types, where engineers can easily design reliable software systems through reusing. For example, a widely-used data structure *queue*, a popular *leader election* algorithm in distributed computing and a controller for Arduino-based wheeled vehicles are encoded as *Mediator* models in [12] and [13].

4 Design of the Framework

4.1 Automatic Code Generation

Manual encoding is exceedingly time consuming and error prone, and has become a huge obstacle between reliable software models and trustworthy computer programs. To deal with this problem, we present a code generation framework for *Mediator*.

The first code generator in this framework aims to generate *Arduino* C programs that can be directly downloaded to the hardware without any manual adaption [13]. As an open-sourced embedded hardware platform, various Arduino motherboards are applied in different domains, robots and quad-copters, for example. Another code generator for System C is already developed by not published yet. The framework is designed to be extensible so that users can easily develop code generators themselves.

4.2 Verification

The presented framework plans to support multi-level verification on both high-level models and low-level codes. For high-level models, we can specify properties as CTL* formulae to both *automata* and *systems*. *Mediator* models and these properties are exported into NuSMV and checked. For low-level source codes, we plan to transform the property formulae to code notations that are supported by many code-level verifiers, such as Frama-C [6], etc.

5 Conclusion and Future Work

In this report, we summarize the current status of our research on *Mediator* and its corresponding component-based modeling and verification framework which forms the core part of the presented Ph.D. thesis. Driven by this topic, we have two publications [12,13] and another two submitted. At least three more publications on this topic are planned.

In the remaining years, we will complete this framework, mainly the code-level verification part and work on more practical case studies. We are investigating the notation language of Frama-C [6], and plan to generate these notations directly from our models and CTL* properties.

Acknowledgements. The work is supervised by Prof. Meng Sun, and partially supported by the National Natural Science Foundation of China under grant no. 61532019, 61202069, 61272160 and 61772038.

References

1. Abdulla, P.A., Deneux, J., Stålmarck, G., Ågren, H., Åkerlund, O.: Designing safe, reliable systems using scade. In: Margaria, T., Steffen, B. (eds.) ISoLA 2004. LNCS, vol. 4313, pp. 115–129. Springer, Heidelberg (2006). https://doi.org/10.1007/11925040_8
2. Amnell, T., et al.: UPPAAL - now, next, and future. In: Cassez, F., Jard, C., Rozoy, B., Ryan, M.D. (eds.) MOVEP 2000. LNCS, vol. 2067, pp. 99–124. Springer, Heidelberg (2001). https://doi.org/10.1007/3-540-45510-8_4
3. Berry, G., Gonthier, G.: The Esterel synchronous programming language: design, semantics, implementation. Sci. Comput. Program. **19**(2), 87–152 (1992)
4. Cataño, N., Rivera, V.: EventB2Java: a code generator for Event-B. In: Rayadurgam, S., Tkachuk, O. (eds.) NFM 2016. LNCS, vol. 9690, pp. 166–171. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-40648-0_13
5. Cavada, R., et al.: The nuXmv symbolic model checker. In: Biere, A., Bloem, R. (eds.) CAV 2014. LNCS, vol. 8559, pp. 334–342. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-08867-9_22
6. Cuoq, P., Kirchner, F., Kosmatov, N., Prevosto, V., Signoles, J., Yakobowski, B.: Frama-C. In: Eleftherakis, G., Hinchey, M., Holcombe, M. (eds.) SEFM 2012. LNCS, vol. 7504, pp. 233–247. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-33826-7_16
7. Hahn, B., Valentine, D.T.: SIMULINK toolbox. In: Essential MATLAB for Engineers and Scientists, pp. 341–356. Academic Press (2016)
8. Jeannin, J., et al.: Formal verification of ACAS X, an industrial airborne collision avoidance system. In: Proceedings of EMSOFT 2015, pp. 127–136. IEEE (2015)
9. Kim, H., Lee, E.A., Broman, D.: A toolkit for construction of authorization service infrastructure for the internet of things. In: Proceedings of IoTDI 2017, pp. 147–158. ACM (2017)
10. Klein, G., et al.: seL4: formal verification of an OS kernel. In: Proceedings of SOSP 2009, pp. 207–220. ACM (2009)
11. Kwiatkowska, M., Norman, G., Parker, D.: PRISM 4.0: verification of probabilistic real-time systems. In: Gopalakrishnan, G., Qadeer, S. (eds.) CAV 2011. LNCS, vol. 6806, pp. 585–591. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-22110-1_47
12. Li, Y., Sun, M.: Component-based modeling in mediator. In: Proença, J., Lumpe, M. (eds.) FACS 2017. LNCS, vol. 10487, pp. 1–19. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-68034-7_1
13. Li, Y., Sun, M.: Generating arduino C codes from *mediator*. In: de Boer, F., Bonsangue, M., Rutten, J. (eds.) It's All About Coordination. LNCS, vol. 10865, pp. 174–188. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-90089-6_12
14. Liu, Z., Morisset, C., Stolz, V.: rCOS: theory and tool for component-based model driven development. In: Arbab, F., Sirjani, M. (eds.) FSEN 2009. LNCS, vol. 5961, pp. 62–80. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-11623-0_3
15. Miller, S.P., Whalen, M.W., Cofer, D.D.: Software model checking takes off. Commun. ACM **53**(2), 58–64 (2010)
16. National Instruments: Labview. http://www.ni.com/zh-cn/shop/labview.html