Breeze: A Modeling Tool for Designing, Analyzing, and Improving Software Architecture

Luxi Chen, Linpeng Huang, Hao Zhong, Chen Li, Xiwen Wu
Department of Computer Science and Engineering, Shanghai Jiao Tong University, Shanghai, China
{leen1988, lphuang, zhonghao, lcroy, wuxiwen}@sjtu.edu.cn

***Abstract*—One of the key challenges in the software engineering lies in requirement engineering. As an important technique for modeling and analyzing requirements, software architecture has been intensively studied in recent years. Although various modeling tools have been proposed in both academy and industry, these tools typically provide limited support for analyzing non-functional requirements at architecture level. To address this problem, in this tool demo, we present a tool, called Breeze, that models, analyzes, and improves software architecture, with an emphasis on its non-functional requirements. In particular, Breeze has three key modules: (1) a modeling module that facilitates the modeling for software systems, (2) an analysis module that verifies non-functional requirements (*e.g.* safety, reliability and correctness) at the architecture level, and (3) a reconfiguration module that allows users to repair defects or to further improve architectures.**

I. Introduction

In practice, requirements can easily become incomplete, inconsistent, and ambiguous, and such poor-quality requirements have a critical negative impact on the quality of software [9]. As an important way to improve the quality of the software system, software architecture has been intensively studied, but there is still adequate space for improvement. For example, Schneider *et al.* [8] propose Unified Requirements Modeling Language (URML) that supports elicitation of both functional and non-functional requirements. Pandey [7] complains that URML cannot describe software at the architecture level, since it is derived from Unified Modeling Language (UML). Ameller *et al.* [1] propose ArchiTech that suggests alternative architectural decisions. Although their tool can improve some non-functional requirements, their suggestions may be not fully reliable, since their tool does not have a solid mathematical foundation. Considering the above limitations, Xiao *et al.* [10] admit that there is still a strong need for better software architecture tools, especially for analyzing NFRs (*e.g.*, safety, reliability, and correctness).

As pointed out by Clements [4], software architecture allows understanding impacts of requirements before implementation. From an architecture of a software system, it is feasible to determine to what degree the software system satisfies NFRs. The architecture-level analysis can detect defects at the design phase. If such defects are found, it reduces the cost of developing a software system, since it is less expensive to fix defects at the early phase of software development.

In this tool demo, we present our Breeze tool. It supports our architecture description language, Breeze/ADL [6], and well integrates with existing NFR analysis tools. As it is based on the Eclipse framework, it is compatible with the state-of-the-art UML. Breeze has the following key novel features:

- Breeze supports both modeling and analyzing requirements at the architecture level. It allows users to model NFRs, and it supports three NFR analysis such as correctness analysis, safety analysis, and reliability analysis.
- Breeze allows users to repair defects in their designed architectures by defining reconfiguration rules. The mechanism allows users to further improve their architectures based on their analysis results.

Breeze is an open source tool, and more information can be obtained from https://github.com/BreezeCSA/Breeze.

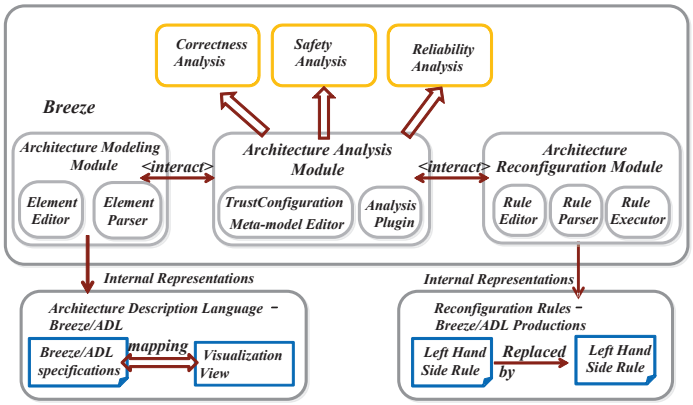

Fig. 1. The overview of Breeze.

II. Tool Description

Figure 1 shows the overview of Breeze, and it consists of the following three key modules.

Architecture modeling module. Breeze models architectures of software systems with Breeze/ADL [6]. Breeze/ADL is an XML-based architecture description language, which specifies the properties of software system in terms of components, connectors and ports. It can model deployment architectures, but is not limited to, since its XML-schema can be easily extended. For example, Li *et al.* [5] extend Breeze to model software systems in the big data era. Figure 2 shows the screenshot of Breeze. It has a GUI that allows users to define software architecture feature elements (*e.g.*, components, connectors, and links), through the corresponding icon buttons.

Architecture analysis module. For each designed architecture, Breeze generates a meta-model based on the specifica-

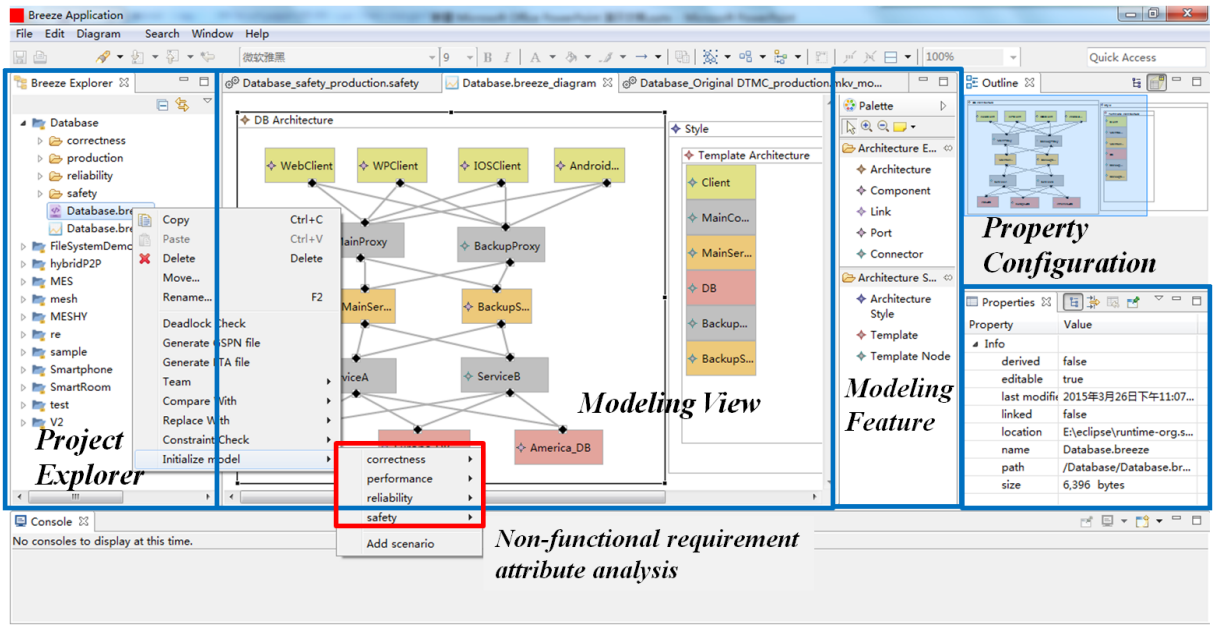

Fig. 2. The screenshot of Breeze.

tions in Breeze/ADL. The meta-model, called *TrustConfiguration*, focuses on NFRs, and it captures experience data and domain knowledge of requirement experts. The current version of Breeze supports the following analysis:

- Correctness analysis. Generating model checking specifications through Breeze/ADL to detect deadlock or inconsistency problems, with the support of NuSMV [3].
- Safety analysis. Weaving safety elements defined in requirements into Breeze/ADL model and identifying and prioritizing possible failure events with fault tree analysis [2], minimum path analysis, and cut set analysis.
- Reliability analysis. Mapping Breeze/ADL model to a discrete-time Markov Chain model to predict the reliability of the architecture.

The analysis results are useful to further improve the quality of software architecture.

Architecture reconfiguration module. Breeze allows users to repair defects in their architectures. To achieve this, Breeze implements a reconfiguration mechanism in Breeze/ADL *Productions* [6]. A *Production* is divided into two parts - the left hand side (LHS) rule and the right hand side (RHS) rule. Both LHS and RHS are Breeze/ADL specifications, and defined by users. The LHS rule presents the precondition and the RHS rule presents the results. As shown in Figure 1, this module compares the LHS rule with architectures, and replaces found matches by the RHS rules. In this way, if an NFR problem is found, they can fix the problem and improve the architecture automatically.

III. Conclusion and Future Work

In this paper, we introduce the Breeze that models, analyzes, and improves architectures of software systems. In the future, we will design meaningful graphical symbols to represent architectural modeling elements. We will also make efforts to implement source code generation from architectures to facilitate the development. Furthermore, we plan to evaluate the effectiveness of Breeze in a large scale industrial setting, and the results may help us understand to what degree our tool improve existing models.

Acknowledgment

This paper was supported by the National Natural Science Foundation of China under Grant No.91118004, 61232007 and the Innovation Program of Shanghai Municipal Education Commission (No. 13ZZ023).

References

[1] D. Ameller, O. Collell, and X. Franch. Architech: Tool support for nfr-guided architectural decision-making. In *Proc. 20th RE*, pages 315–316. IEEE, 2012.

[2] L. Chen, L. Huang, C. Li, L. Wu, and W. Luo. Design and safety analysis for system architecture: A breeze/adl-based approach. In *Proc. 38th COMPSAC*, pages 261–266, 2014.

[3] A. Cimatti, E. Clarke, E. Giunchiglia, F. Giunchiglia, M. Pistore, M. Roveri, R. Sebastiani, and A. Tacchella. NuSMV 2: An opensource tool for symbolic model checking. In *Proc. 14th CAV*, pages 359–364, 2002.

[4] P. Clements. *Software architecture in practice*. PhD thesis, Software Engineering Institute, 2012.

[5] C. Li, L. Huang, and L. Chen. Breeze graph grammar: a graph grammar approach for modeling the software architecture of big data-oriented software systems. *Software: Practice and Experience*, 2014.

[6] C. Li, L. Huang, L. Chen, and C. Yu. Breeze/ADL: Graph grammar support for an XML-based software architecture description language. In *Proc. 37th COMPSAC*, pages 800–805, 2013.

[7] R. Pandey. Architectural description languages (ADLs) vs UML: a review. *ACM SIGSOFT Software Engineering Notes*, 35(3):1–5, 2010.

[8] F. Schneider, B. Bruegge, and B. Berenbach. A tool implementation of the unified requirements modeling language as enterprise architect add-in. In *Proc. 21st RE*, pages 334–335, 2013.

[9] A. Van Lamsweerde. Requirements engineering in the year 00: A research perspective. In *Proc. 22nd ICSE*, pages 5–19, 2000.

[10] L. Xiao, Y. Cai, and R. Kazman. Titan: a toolset that connects software architecture with quality analysis. In *Proc. 22nd FSE*, pages 763–766, 2014.