Tool Support for Statically Checking the Structural Conformance of an Object-Oriented System to its Runtime Architecture

Marwan Abi-Antoun Jonathan Aldrich
School of Computer Science, Carnegie Mellon University
{marwan.abi-antoun, jonathan.aldrich}@cs.cmu.edu

Abstract

Maintaining the conformance of an implementation to its architecture is difficult in practice since developers often make changes that degrade the architectural structure.

We present tools for statically checking the structural conformance of a system to its runtime architecture.

Categories and Subject Descriptors D.2.2 [*Software Engineering*]: Design Tools and Techniques

General Terms Design, Documentation

Keywords runtime architecture, conformance checking

1. Introduction

Ensuring conformance of the code architecture or *module view* has received much attention [5]. The code architecture is useful for analyzing quality attributes such as modifiability. Architecture-based analyses for performance, security and reliability, however, use *runtime views*. A Component-and-Connector (C&C) architectural view models *runtime* entities and their potential interactions. Thus, in an object-oriented system, a component is one or more *objects*.

2. Approach Overview

Following the extract-abstract-check model [5], we check an implementation's conformance by structurally comparing an as-designed view to an abstracted as-built view. In the terminology of Murphy et al. [5], the analysis identifies:

- **Convergence:** a node or edge that is both in the as-built view and in the as-designed view;
- **Divergence:** a node or edge that is in the as-built view but not in the as-designed view;
- **Absence:** a node or edge that is in the as-designed view but not in the as-built view.

OOPSLA'08, October 19–23, 2008, Nashville, Tennessee, USA.
ACM 978-1-60558-220-7/08/10.

Methodology. The approach involves the following steps, illustrated with CourSys, a prototypical three-tiered system:

1. Document the as-designed architecture using an Architecture Description Language such as Acme [4] (Fig.3(a));
2. Abstract an as-built architecture from the implementation, using the following sub-steps:
   (a) Add ownership domain annotations to the code [1];
   (b) Extract an Ownership Object Graph (OOG), which is a sound hierarchical representation of any possible runtime object graph (Fig. 1) [1];
   (c) Map the extracted OOG to an as-built C&C view, with architectural types and traceability information;
3. Structurally compare the as-built and the as-designed architectures to check their conformance; and
4. Obtain a measure of conformance.

Architectural Conformance. A system conforms to its as-designed architecture if the latter is a conservative abstraction of the system's runtime structure. By the *communication integrity* principle, *each component in the implementation may only communicate directly with the components to which it is connected in the architecture*.

View synchronization assumes that both views are equally important and attempts to make them identical [3]. In contrast, conformance checking is asymmetric because the as-built view often contains details that are irrelevant to the as-designed view. Guided by the principle above, conformance checking accounts for any communication in the as-built view that is not in the as-designed view, without cluttering the as-designed view with implementation details [2].

Displaying Conformance. The analysis produces a view of the as-designed architecture (Fig.3(b)), to show the results of the conformance check, in terms of convergences, divergences and absences (Fig. 2). The analysis also enriches the conformance view with traceability to the code.

Measuring Conformance. The analysis also computes conformance metrics based on the graph edit distance between the as-built and the as-designed views [2].

Example. When checking the conformance for the CourSys example, the analysis detects numerous renames between the two views (Fig. 3(b)). For instance, `dataTier` in the as-designed view corresponds to the `DATA` domain in the

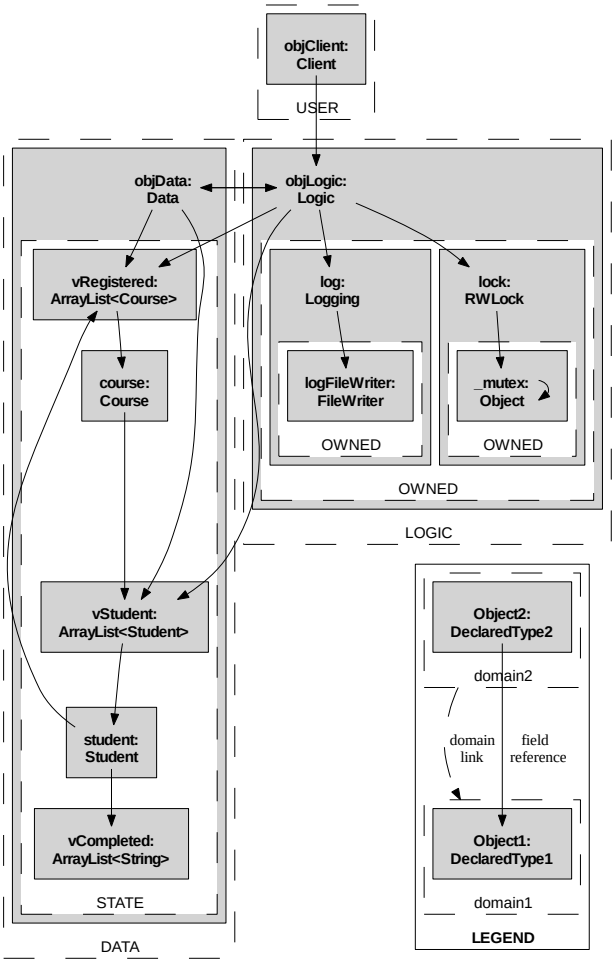


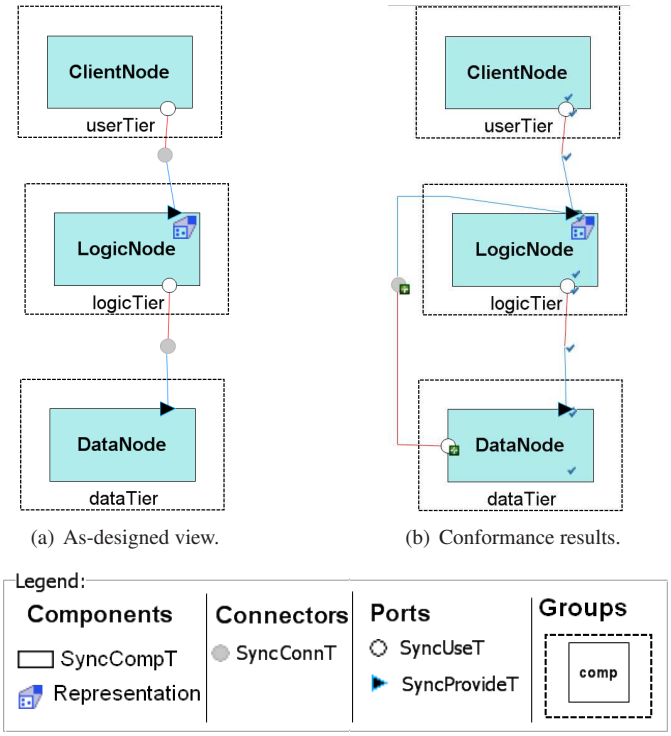
Figure 1. CourSys Ownership Object Graph.

as-built view. Similarly, `DataNode` maps to `objData`, and `LogicNode` maps to `objData`. Inside `LogicNode`'s substructure in the as-designed view, `Logging` is mapped to a `log` component in the as-built view. In the as-built view, `objData` has additional substructure missing from the as-designed view. The conformance check also detects a divergence in the form of a SyncUseT Port on `DataNode` and a Connector that attaches `DataNode` to the SyncProvideT port on `LogicNode`. Such an "upcall" from a data to a logic tier is often an architectural violation.

Using the traceability information in the as-built C&C view, a developer can trace the unexpected edge to the code without having to potentially review the entire code base. She determines that the upcall is due to a field reference of type `ILogic` declared inside class `Data` [1].

(a) (b) (c)

Figure 2. Fig. 2(a) indicates a *convergence*, Fig. 2(b) a *divergence*, and Fig. 2(c) an *absence*.



(a) As-designed view. (b) Conformance results.

Figure 3. CourSys conformance results.

3. Conclusion

We presented tool support for a semi-automated approach to statically check the structural conformance of an object-oriented system to its runtime architecture. The approach uses ownership annotations instead of using language extensions or mandating specific implementation frameworks.

From an annotated program, an analysis extracts a hierarchical view of the runtime object graph. Another analysis converts that representation into an as-built C&C view. A third analysis then structurally compares the as-built and the as-designed views, highlights the key differences between them and measures their structural conformance.

References

[1] M. Abi-Antoun and J. Aldrich. Tool Support for the Static Extraction of Sound Hierarchical Representations of Runtime Object Graphs. In *OOPSLA Companion*, 2008.

[2] M. Abi-Antoun and J. Aldrich. Static Conformance Checking of Runtime Architectural Structure. CMU-ISR-08-132, 2008.

[3] M. Abi-Antoun, J. Aldrich, N. Nahas, B. Schmerl, and D. Garlan. Differencing and Merging of Architectural Views. *ASE*, 15(8):35–74, 2008.

[4] D. Garlan *et al.* The Acme Architectural Description Language. `http://www.cs.cmu.edu/~acme`.

[5] G. C. Murphy, D. Notkin, and K. J. Sullivan. Software Reflexion Models: Bridging the Gap between Design and Implementation. *TSE*, 27(4):364–380, 2001.