Modeling and Implementing Software Architecture with Acme and ArchJava

Marwan Abi-Antoun  
*Carnegie Mellon University*

Jonathan Aldrich  
*Carnegie Mellon University*

David Garlan  
*Carnegie Mellon University*

Bradley Schmerl  
*Carnegie Mellon University*

Nagi Nahas  
*Association of Computing Machinery*

*See next page for additional authors*

Follow this and additional works at: [http://repository.cmu.edu/compsci](http://repository.cmu.edu/compsci)

**Published In**
ABSTRACT
We demonstrate a tool to incrementally synchronize an Acme architectural model described in the Acme Architectural Description Language (ADL) with an implementation in ArchJava, an extension of the Java programming language that includes explicit architectural modeling constructs.

Categories and Subject Descriptors
D.2.11 [Software Architecture]: Languages

General Terms
Documentation, Design, Languages, Verification.

1. Introduction
The software architecture of a system defines its high-level organization as a collection of interacting components, connectors, and constraints on interaction, along with their additional properties defining the expected behavior. Over the past decade, numerous architecture description languages (ADLs) have been developed and applied to real-world systems. A crucial link is still missing, namely, ensuring that a software system is implemented according to its architectural design. We demonstrate tool support to refine architecture into code as well as maintain consistency between architecture and implementation.

2. Acme and AcmeStudio
We use Acme as an example of a mature general purpose architecture description language. Acme supports extensible styles for different domains, and extensible properties and analyses. AcmeStudio [2] is a domain-neutral architecture modeling environment for Acme, implemented as an Eclipse plug-in.

3. ArchJava
We have recently developed ArchJava [1], an extension to Java that enforces architectural structure within source code: developers can specify components, connectors, port constructs, and relate object instances, while completing the implementation using the Java programming language. However, ArchJava does not enforce other important architectural properties such as system behavior or architectural style.

4. Integration between Acme and ArchJava
We have developed additional Eclipse plug-ins with several capabilities to achieve better integration between the two models.

An architect can model an architecture using AcmeStudio, having access to AcmeStudio’s analyses to verify desired architectural properties. The architect can then generate ArchJava starter code using the refinement plug-in. As developers complete the implementation to provide the functionality of the system, ArchJava’s checks help ensure that the implementation conforms to the architect’s design. Furthermore, any changes made by the engineers are at least reflected in the ArchJava architecture.

If an existing ArchJava implementation does not have its architecture specified using an architectural description language, or if the documented architecture is severely out of date, we can import an Acme architecture from an existing ArchJava implementation. This makes it easier to get an overall view of the architecture, navigate between different levels of architectural modeling, and re-run the Acme architectural analyses to incorporate new insights and requirements into the architecture.

We also provide the capability to incrementally synchronize an Acme architecture and an ArchJava implementation, by pushing changes to Acme and/or to ArchJava, to keep architecture and implementation consistent during software evolution.

We build an intermediate representation of the Acme model and the ArchJava model that includes architectural types and instances. We then detect structural differences (Figure 2) between subsets of the two intermediate representations using our implementation of a tree-to-tree correction algorithm for unordered labeled trees based on [3]. The selection of the subset is under user control: if the Acme model does not specify some information that exists in ArchJava (such as method signatures), this information can be excluded from the comparison to avoid false positives. The structural comparison finds matches, and classifies the differences as inserts, deletes, and renames. The tool then generates an edit script to make one representation more consistent with the other. The user can specify additional

Copyright is held by the author/owner(s).
ICSE ’05, May 15–21, 2005, St. Louis, Missouri, USA.
ACM 1-58113-963-2/05/0005.
information that cannot be automatically retrieved from ArchJava, such as Acme types for the Acme elements to be created. We currently only support applying the edit script to the Acme model (Figure 3); we are still working on nicely laying out the generated Acme elements, and changing the ArchJava infrastructure to support making incremental changes to an existing ArchJava implementation.

We have validated this tool on additional examples and variations. When we compared the same ArchJava implementation used above with an Acme model where the capitalize component was replaced with its representation, the tool correctly detected most of the matching components. When we compared our earliest ArchJava implementation of the CaPiTaLiZe system to the current Acme model, the tool correctly detected a large number of differences, consisting of many names (all component and port names) and an additional buffer component in ArchJava: it correctly matched most of the names, except for the ArchJava components split, upper and lower: the ArchJava split was implemented with one output port, making it structurally undistinguishable from the ArchJava upper and lower components (the Acme split component has two output ports). Indeed, the tool detected a subtle architectural mismatch. The user can cancel the synchronization, correct the mismatch (e.g., modify the ArchJava split component to have two output ports), and resume the synchronization. We are also working on enabling the user to override the automatically detected differences (e.g., canceling delete edits) without leaving the synchronization tool.

5. Acknowledgments
This work is supported by a 2004 IBM Eclipse Innovation Grant and NSF grant CCR-0204047.

6. References