Conformance of Implementation to a Security Architecture

Marwan Abi-Antoun, Jonathan Aldrich, Nagi Nahas, Bradley Schmerl and David Garlan

An architectural approach to security

• The security architecture of a system shows its runtime components, and how those components communicate, with various security invariants on where authentication occurs or untrusted input is validated, etc. It matches how experts think about security, vs. a purely code-based strategy.
• However, the Achilles heel of a security architecture is the lack of any enforced or checked architectural conformance between the actual code and the intended security architecture of the system.
• We should be able to trust the architecture to reflect the actual implementation: an incorrect model can be worse than no model at all, as it may mislead us into an incorrect understanding of the system’s security.

Enforcing architectural conformance

• Enforcement of architectural intent at the programming language level using a small number of annotations at architectural boundaries;
• Automated abstraction of an instance-based hierarchical runtime architectural view from annotated code;
• Semi-automated incremental synchronization between the implementation-level view and the conceptual-level view to detect and correct divergences.

Checking conformance: case study 1

The tool automatically matched renamed elements and detected extra connectors marked with X

Matching two incompatible type structures at two different levels of abstraction

The tool detects structural differences:
• Match ( ),
• Insert ( ),
• Delete ( ),
• Rename ( )

The synchronized diagram reveals a missing sub-system and a callback

Original informal diagram

Checking conformance: case study 2

The as-built architecture: the connection marked with X is a violation of the documented architecture and of the EJB specification!

On the right side:

Synchronizing architectural views

• Detect structural differences:
  • No Unique Identifiers
  • Detect Renames
  • Detect Hierarchical Moves
  • Support Manual Overrides
  • Type Information Optional
  • Disconnected/Stateless Operation
  • Detect inserts and deletes

Key Idea: Using the tree-hierarchy in hierarchical views gives scalability over general graph matching.

Specify conformance of implementation to a security architecture

Example: no cycles allowed in a system following the pipe-and-filter architectural style

Detect additional violations:
• Uncover additional non-structural violations by enriching the up-to-date architectural model with
• Architectural types and styles
• Constraints (general predicates)

Specifying architectural intent in code

Initial attempts: extend the underlying language

• ArchJava: enforce the control flow architectural structure, with language extensions

• AliasJava: annotations to describe the data sharing architecture, i.e., data:
  • Confined within component
  • Passed linearly from one component to another
  • Shared temporarily or persistently

• Case studies: several studies were conducted on non-trivial systems to evaluate the usability of ArchJava and AliasJava.

Current research: impose architectural structure on arbitrary object graphs

• Use annotations instead of extensions
• Statically approximate runtime graph
• Add hierarchy using ownership

A’s components and connectors

B’s components and connectors

Connector X

Logic tier

User tier

Data tier

Architect’s diagram showing components and connectors

Having view extracted from annotated code can better match how experts think about security, vs. a purely code-based strategy.