Trinity: A Language for Multi-View Architecture Description and Control

Maddie Kirwin
kirwinma@grinnell.edu

Selva Samuel
ssamuel@cs.cmu.edu

Jonathan Aldrich
Jonathan.Aldrich@cs.cmu.edu

Software Architecture

the "fundamental organization of a system embodied in its components, their relations to each other, and the environment"

Architecture Views

Module or Code

Modules: principal units of implementation

- Used to explain system functionality + structure of code base
- A blueprint for code construction and incremental development
- Analysis of code dependency

Component and Connector (CnC)

Components: elements that have some runtime presence (processes, objects, clients, servers).
Connectors: components' pathways of interaction (protocols, information flows).

- Show how the system works
- Guide development around structure & behavior of runtime elements
- To reason about performance and reliability

Deployment

Deployment View: a mapping between software and nonsoftware elements in the former's environment.

- Analyzing actual runtime performance, reliability, and security
- SW elements: CnC elements
- Environmental elements: hardware, network elements, and their capabilities

The Problem

It is hard to determine whether the logical relationships between entities in architecture diagrams are present in system implementations.

Previous Solutions

Architecture Description Languages (ADLs)

- (-) *Description:* Inferred by the name, ADLs only describe software architectures; they do not prescribe, or **enforce conformance** to them
- (+) *Analysis:* ADLs are focused on system analyses
- (+) *Formal Notation:* Currently, ADLs are the most formal mainstream architecture tools available

ArchJava Java extension unifying architecture and implementation

- (-) *Application:* Does not check for conformity to architecture
- (-) *Distributed Systems:* No support for distributed systems
- (-) *Multiple Views:* Lacks support for multiple architecture views; focuses only on Component-and-Connector view

Trinity's Approach

- Make software architecture a **"live" component** of Trinity systems
- Trinity enforced **architecture conformance** complements ADL analyses
- Support architecture conformance and **communication integrity** in **distributed systems**
- Directly translate the conceptual entities from multiple views into **code-enforced constructs**
- Support **all three software architecture views** (module or code, CnC, and deployment)

Design

Implementation Concepts

Architecture concepts are translated into untime entities in Trinity

<This is an explanation of the different architecture entities in Trinity, not specific to an example>
Component: a runtime entity that may interact with other components through ports.

Connector: interaction pathways that join two compatible component ports.

Port: component access points that enable interaction with other components.

entryPoints: a program starting point that enables execution.

<INSERT EXAMPLE TRINITY CODE OF EXAMPLE ARCH.>
*** describe each component**

<INSERT SOFTWARE ARCHITECTURE DIAGRAM OF EXAMPLE>

Demonstrated Principles

Trinity's design demonstrates the following principles:

- Readability
- Reuse/adaptability
- Communication integrity, especially in distributed systems

Example

```
component Client
    port getInfo: requires CSIface

component Server
    port sendInfo: provides CSIface

external component DB
    port dbIface: target DBModule

connector JDBCCtr
    val connectionString: String

    architecture
        components
            RequestHandler ch
            DB db

        connectors
            JDBCCtr jdbcCtr

        attachments
            connect rh.dbIface and db.dbIface
            with jdbcCtr

        bindings
            sendInfo is rh.sendInfo
```

```
architecture
    components
        Client client
        Server server

    connectors
        JSONCtr jsonCtr
    attachments
        Connect client.getInfo and
        server.sendInfo with jsonCtr

    entryPoints
        Client: start
```