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

Maddie Kirwin kirwinma@grinnell.edu

> Selva Samuel ssamuel@cs.cmu.edu

Jonathan Aldrich Jonathan.Aldrich@cs.cmu.edu

# Software Architecture

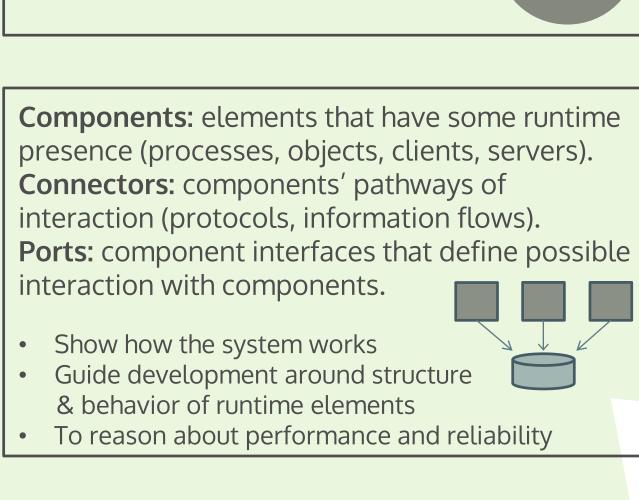
the "fundamental organization of a system embodied in its components, their relations to each other, and the environment"

### **Architecture Views**

# Module

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



**Deployment View:** a mapping between software and non-software 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

- (+) Conformance: Checks for architecture conformity
- (-) Distributed Systems: No conformance checks in distributed systems (ArchJava supports multiple systems via custom connectors, but does not enforce conformity)
- (-) 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

Deployment

#### Implementation Concepts

Architecture concepts are translated into runtime entities in Trinity

#### **Trinity Architecture Components**

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 can allow interaction with other components.

attachments: declarations that enable connections between compatible components to be made entryPoints: a program starting point that permit execution.

# **Demonstrated Principles**

Trinity's design demonstrates the following principles:

Readability

System architecture is contained in a single file and is prescriptive, uniting design and implementation.

Reuse and Adaptability

Compatibility checking and code generation make switching, adding, and removing architecture elements easier and more secure.

Communication Integrity in Distributed Systems

Wyvern's capability-based module system ensures that elements only have direct control of capabilities to which they have explicit access.

# 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

// Server and database are connected // using the JDBC Connector

CnC elements like components and connectors are Wyvern/Trinity objects.

Client-Server interface functions are provided by the programmer architecture

components

Client client Server server

connectors

JSONCtr jsonCtr

attachments

Connect client.getInfo and server.sendInfo with jsonCtr

entryPoints

Client: start