

# **An Approach to Architecture-Based Software Integration**

**Nenad Medvidovic**

Computer Science Department  
University of Southern California  
Los Angeles, CA 90089-0781

**`nen@usc.edu`**

**`http://sunset.usc.edu/~nen/`**

## Motivation

- The reality of large-scale software development
    - component-based software construction
    - OTS reuse
    - continuous system evolution
    - distributed world
    - heterogeneous world
- Architecture to the rescue

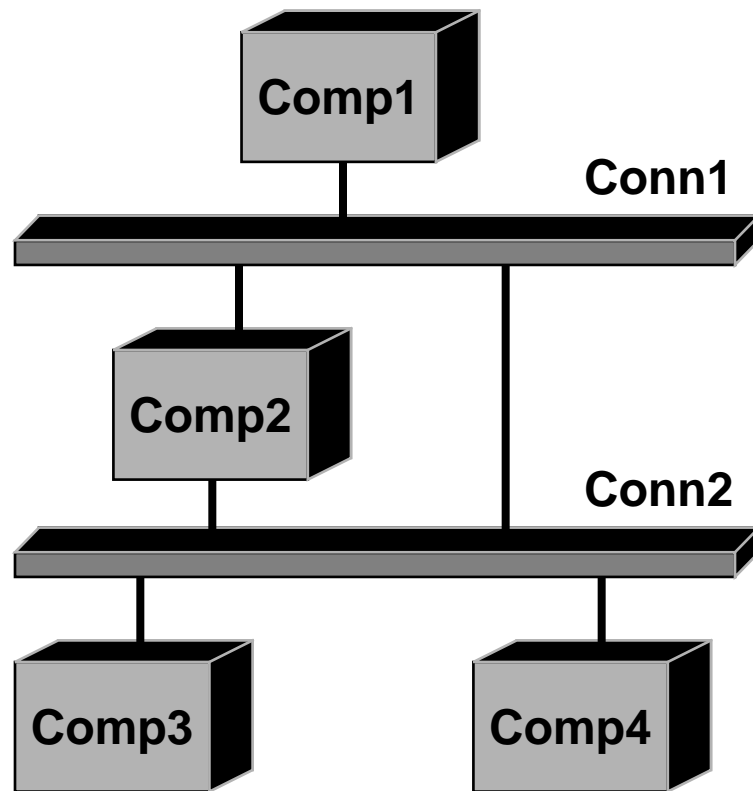
## Why Architecture?

- Separation of concerns
- Coarse-grain system decomposition
- Components as software building blocks
- Heterogeneity addressed via OTS reuse
- Distribution and heterogeneity addressed via connectors

## All Architectures Are Not Created Equal

- Reuse, heterogeneity, and distribution are *not* guaranteed
  - one-of-a-kind, homogeneous, monolithic systems also have architectures
  - rigid connectors
  - architectures influenced by non-architectural issues
  - architectural mismatch (Garlan et al.)
    - explicit focus on desired properties is needed
- Architectural styles
  - domain-specific abstractions
  - effective domain-independent patterns and idioms

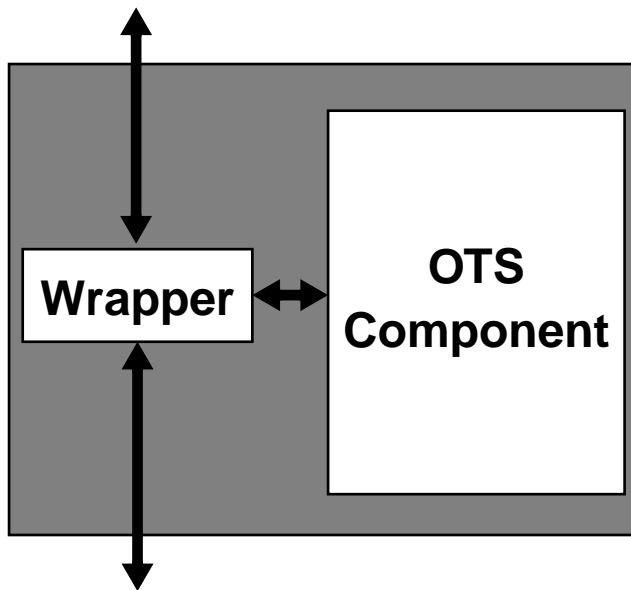
## An Architectural Style that Supports Heterogeneity, Reuse, and Distribution



- Minimal component interdependencies
- Adaptable connectors
- Evolvable configurations
- Separation of architecture from implementation

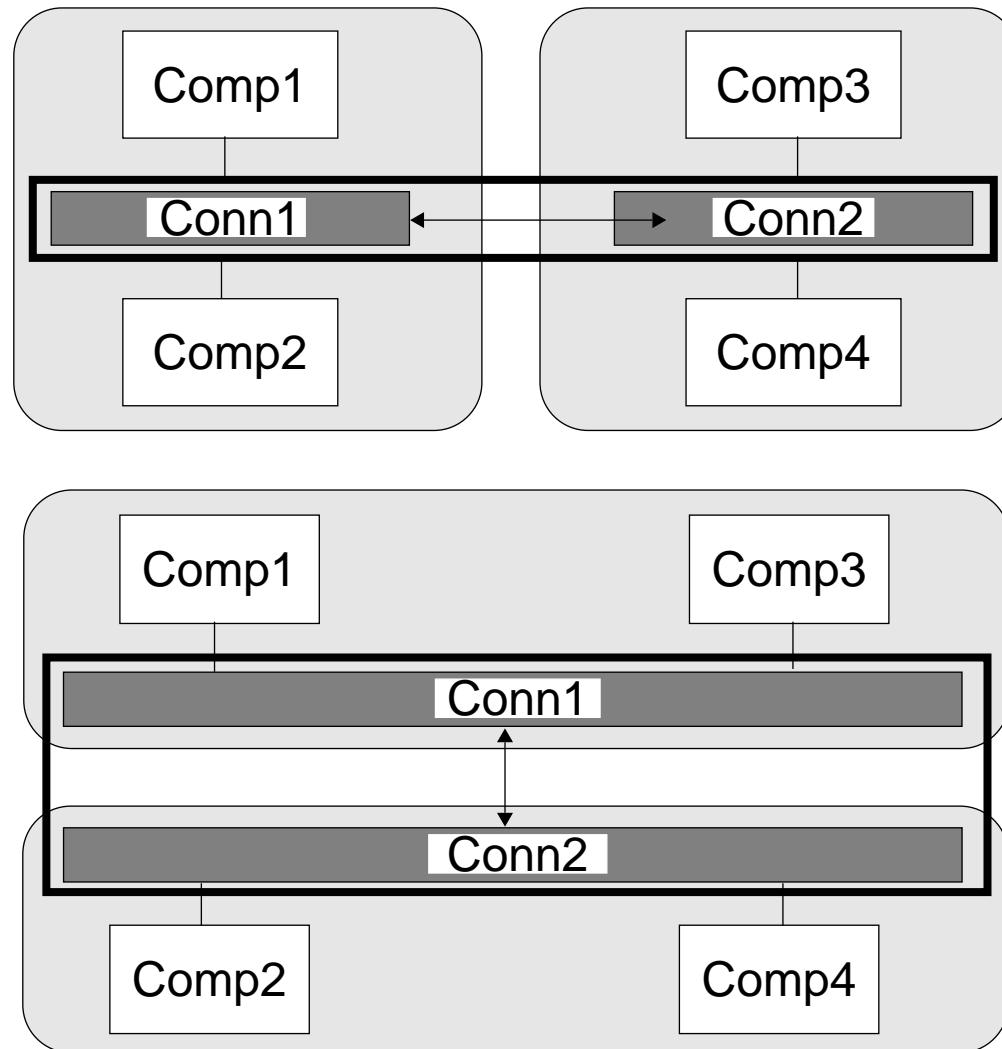
## OTS Reuse and Heterogeneity

- Accomplished via connectors and light-weight component wrappers

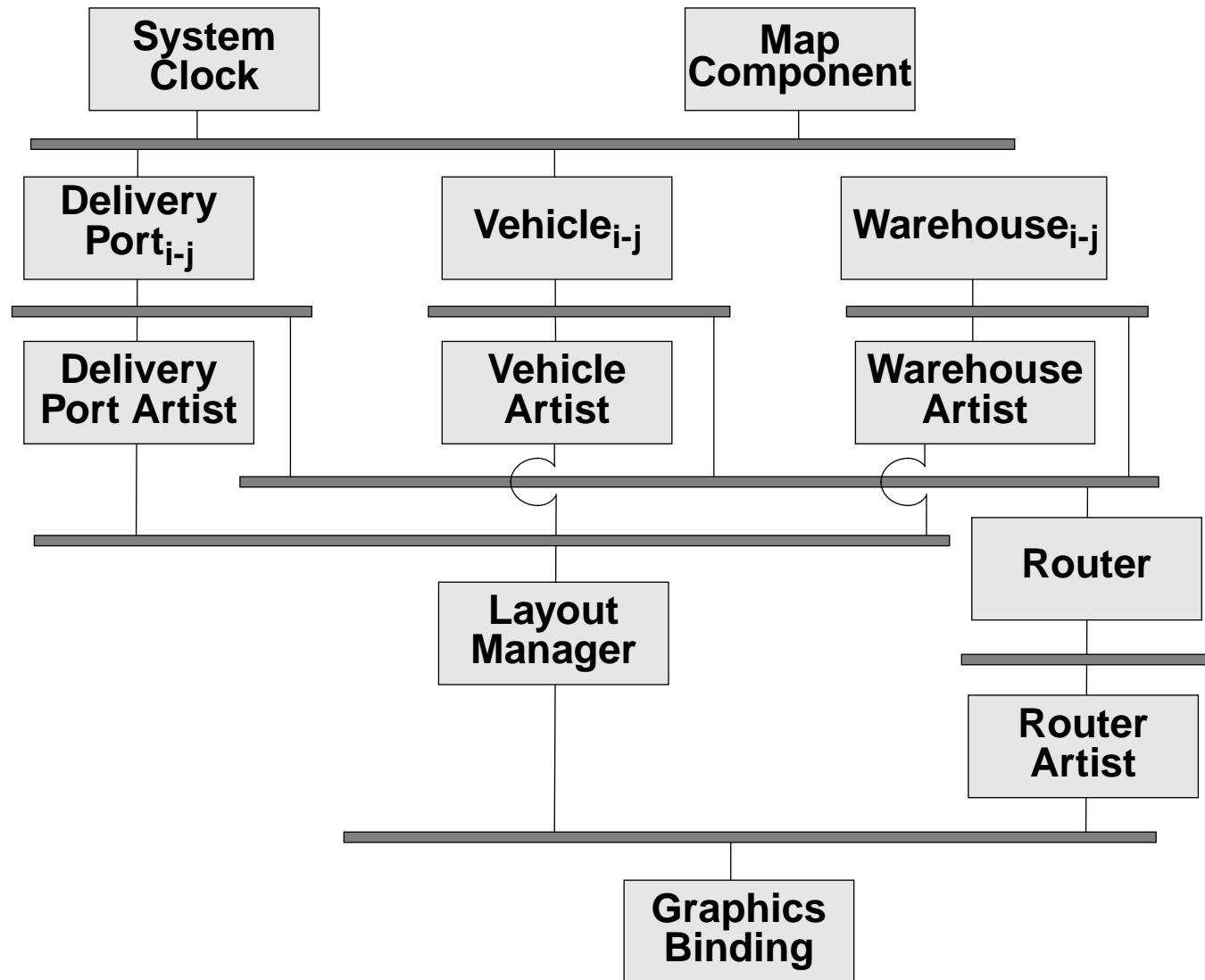


- inadequate functionality
  - source code modification
- explicit invocation
  - wrapper
- different thread of control
  - inter-thread connector
- different PL and/or OS process
  - IPC connector
- message interface mismatch
  - adaptor

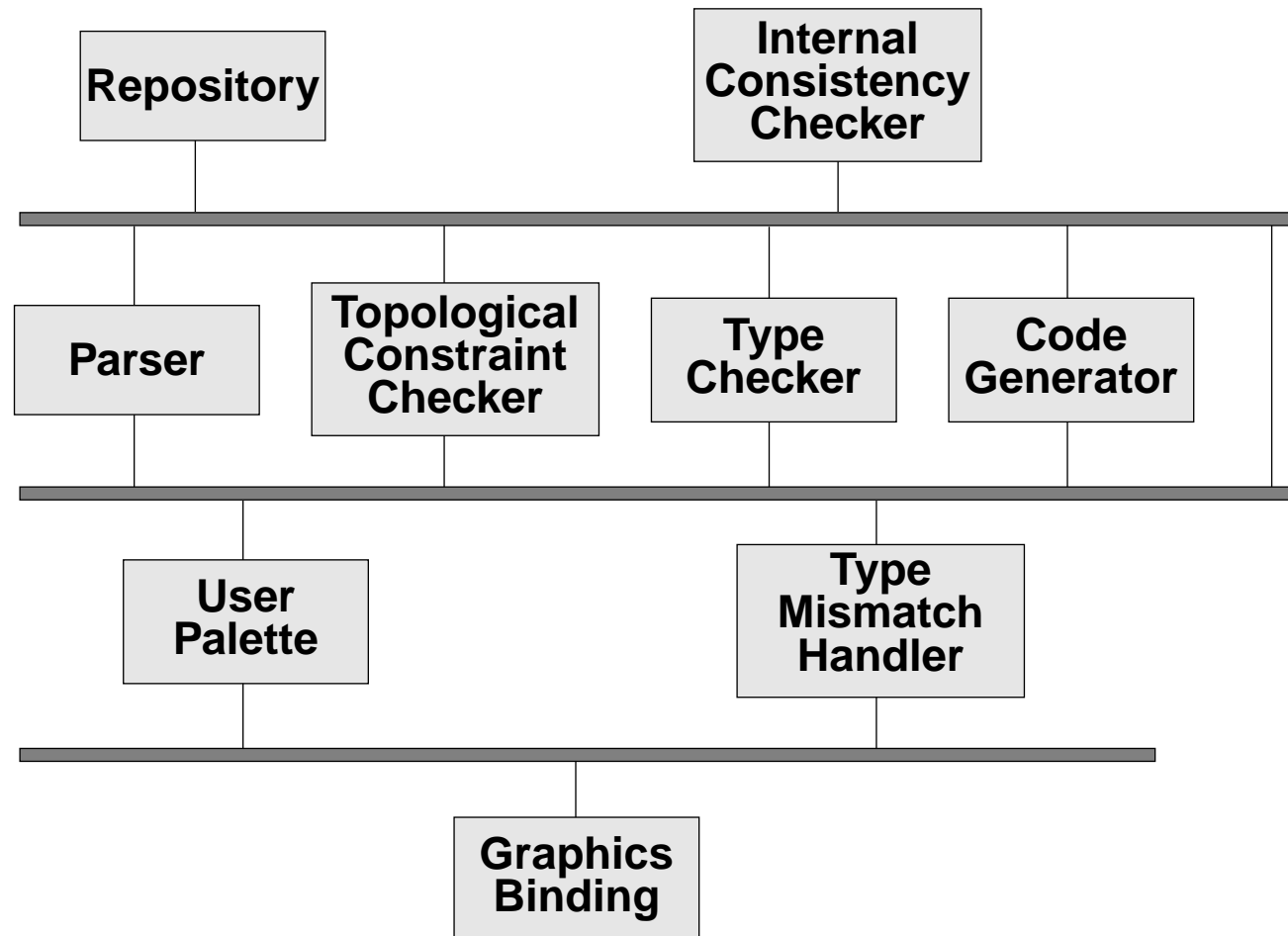
# Distribution and Heterogeneity



# Integrating an Application — Cargo Router



# Integrating a Tool Suite — DRADEL



## Discussion

- Component integration is no different from tool integration
  - tools have coarser granularity
  - lighter-weight wrappers
- Software environments applicable on themselves
- 80-20 rule for OTS component integration
  - loose tool integration is typically sufficient
  - open issue: “deep semantic” integration
- Some sources of architectural mismatch may be prevented
  - OTS reuse must be planned
  - architectural models are the necessary first step