

Enabling the Refinement of a Software Architecture into a Design

Marwan Abi-Antoun and Nenad Medvidovic

{marwan, neno}@sunset.usc.edu

Computer Science Department

University of Southern California

Los Angeles, CA 90089-0781, USA

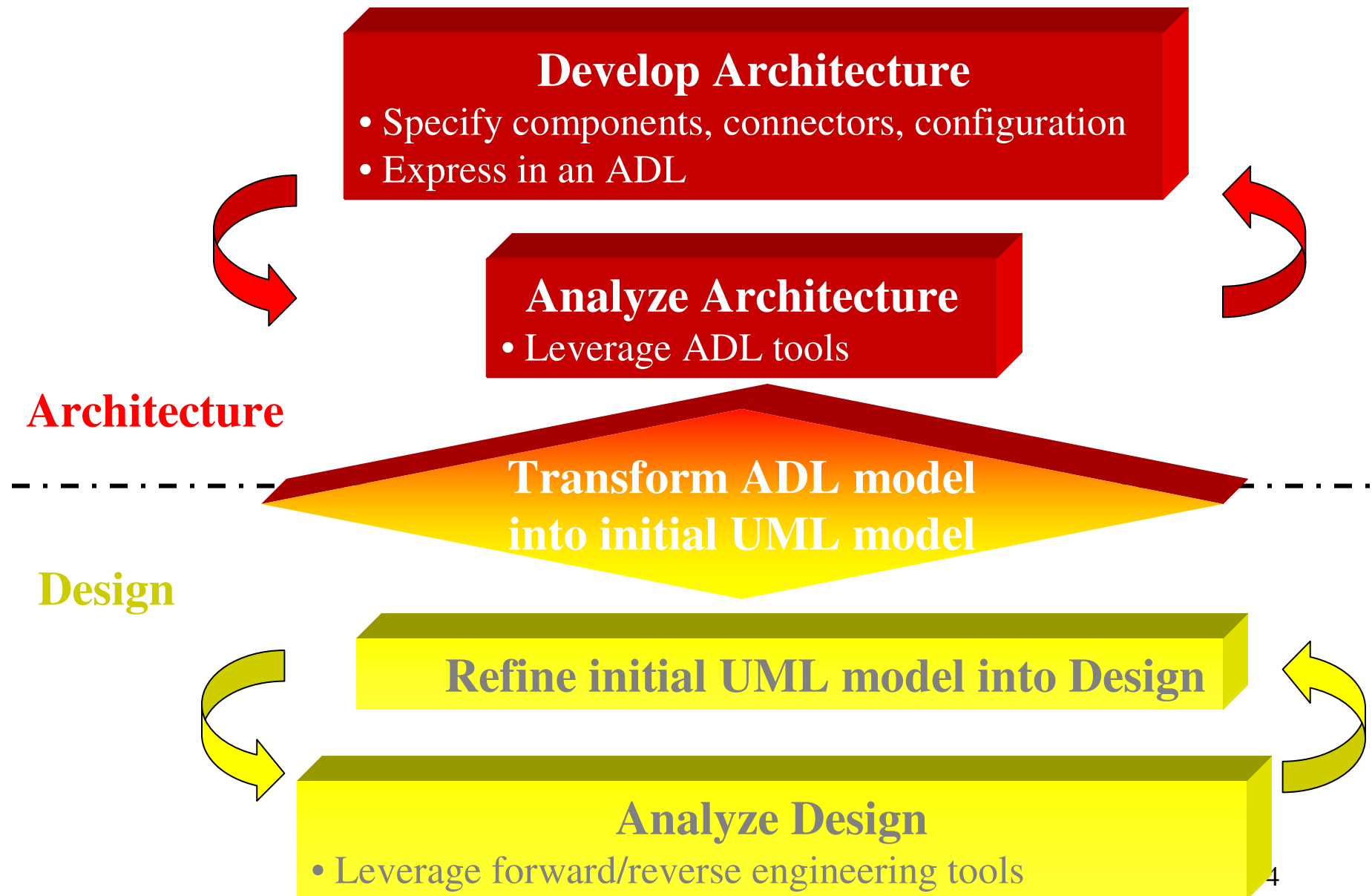
Problem Statement

- Software architecture research has addressed **formal specification** and analysis of coarse-grained software models using **rigorous modeling notations, Architecture Description Languages (ADLs)**.
- The industrial software community has been standardizing on a general purpose solution, the **Unified Modeling Language (UML)**, which provides a family of models that address the entire software lifecycle.
- How can we capitalize on the strengths of both approaches? How can we augment UML with support to specific problems? How can we make ADLs transition into the mainstream?

Contributions

- We will describe an approach that combines the benefits of ADLs with those of UML:
 - Use ADLs for **architecture-level** analyses
 - Use the UML for **design**, and downstream activities
- For a selected ADL (C2SADEL), we will:
 - Define a set of rules to **transform** an architectural representation in an ADL into an **initial UML model** that can then be further **refined**, while enforcing the **architectural constraints**
 - Leverage the available **tool support for the ADL**, and **integrate with UML tool support**

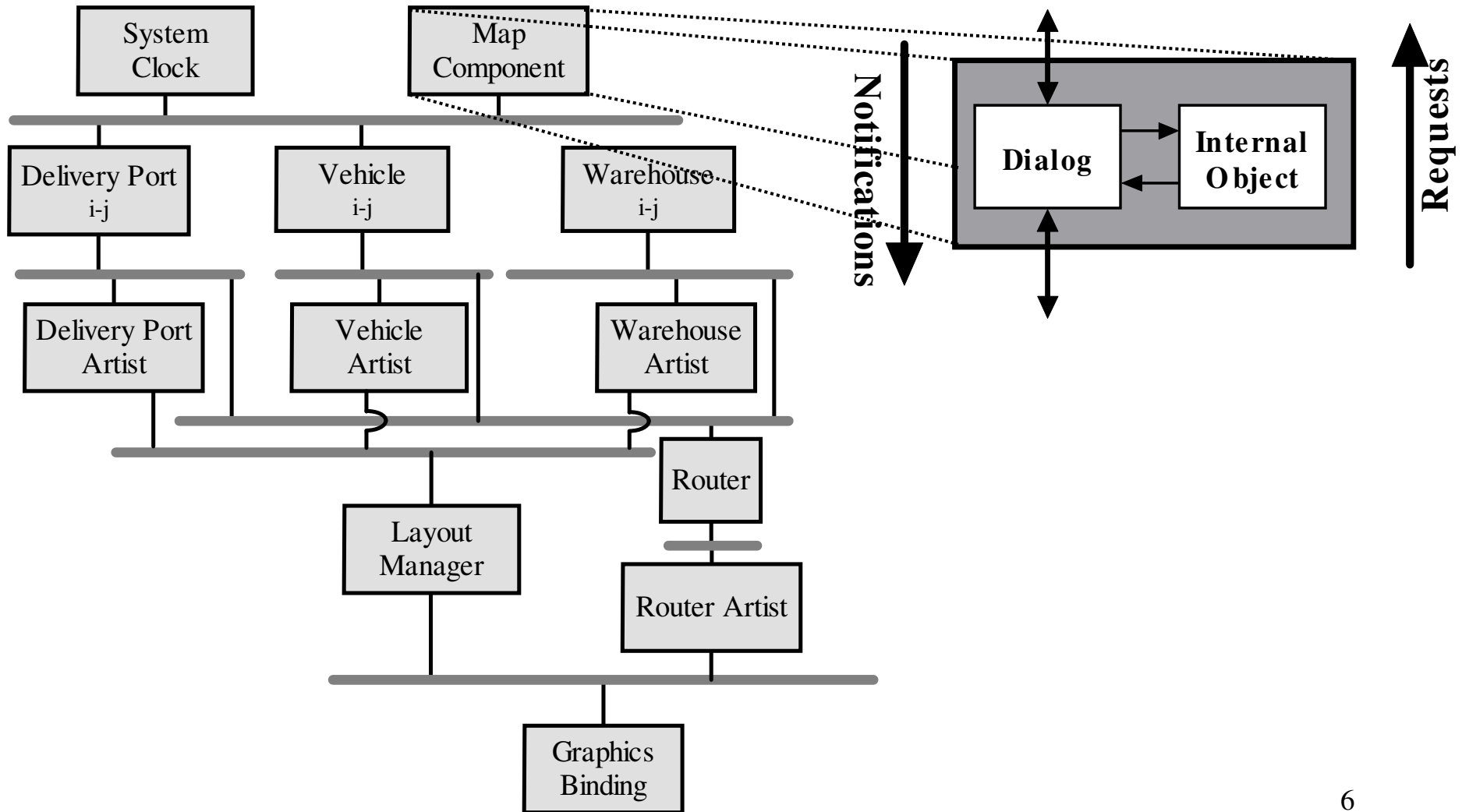
Proposed Approach



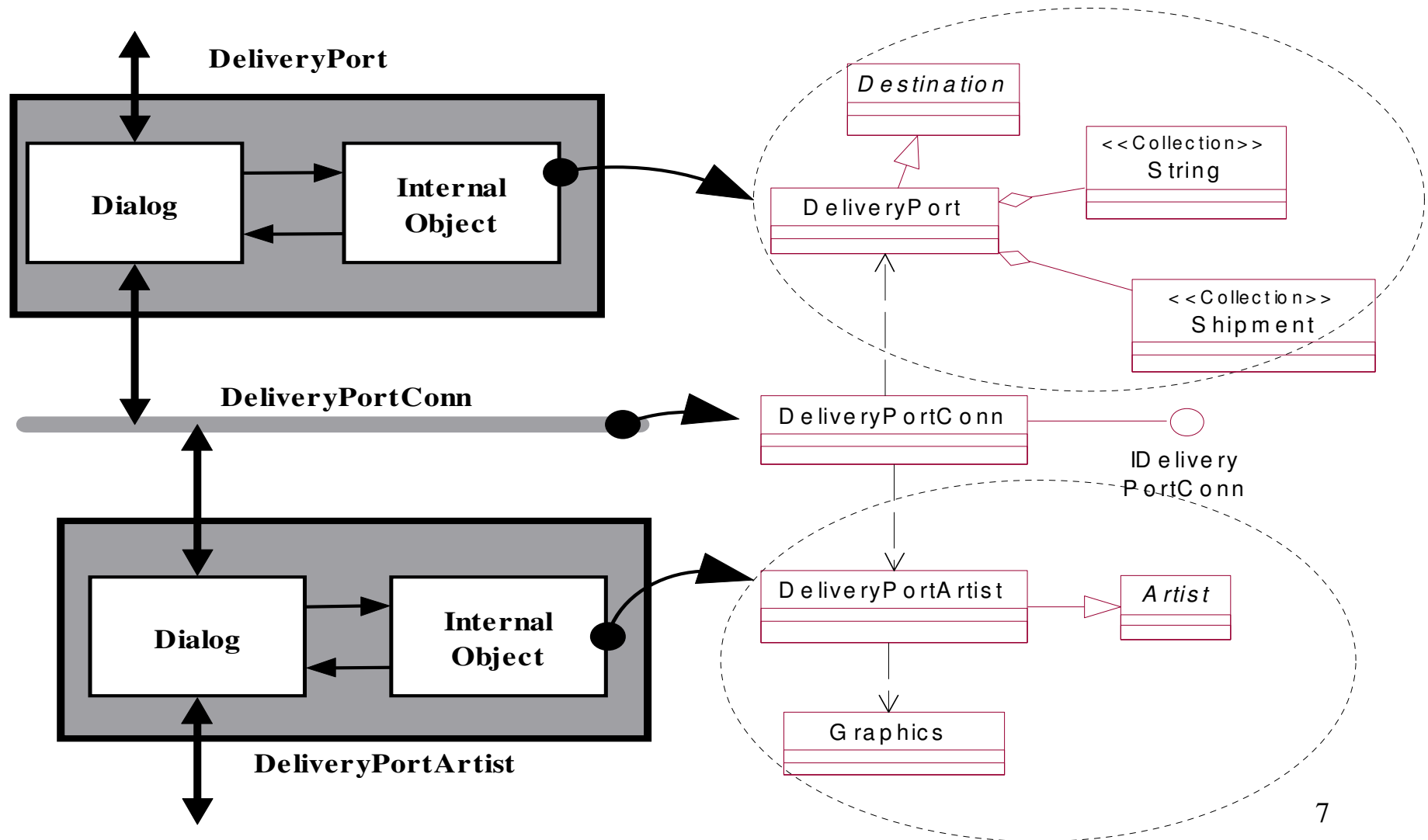
Transform ADL representation into initial UML model

- **Prepare for the Transformation**
 - Understand the rules and **constraints** of architectural style, ...
 - Tailor transformation rules to the selected ADL
- **Apply Transformation**
 - Start transformation using **native UML constructs** (classes, interfaces, ...) to model non-style-specific constructs, typically component and connector “internals”
 - Complete transformation using **stereotypes** to model **architectural constructs** (components, connectors, ...) and define **constraints** to ensure **conformance** to architectural style
- **Caveats**
 - Enforce rules of architectural style
 - ☞ Minimize human error by providing tool support!

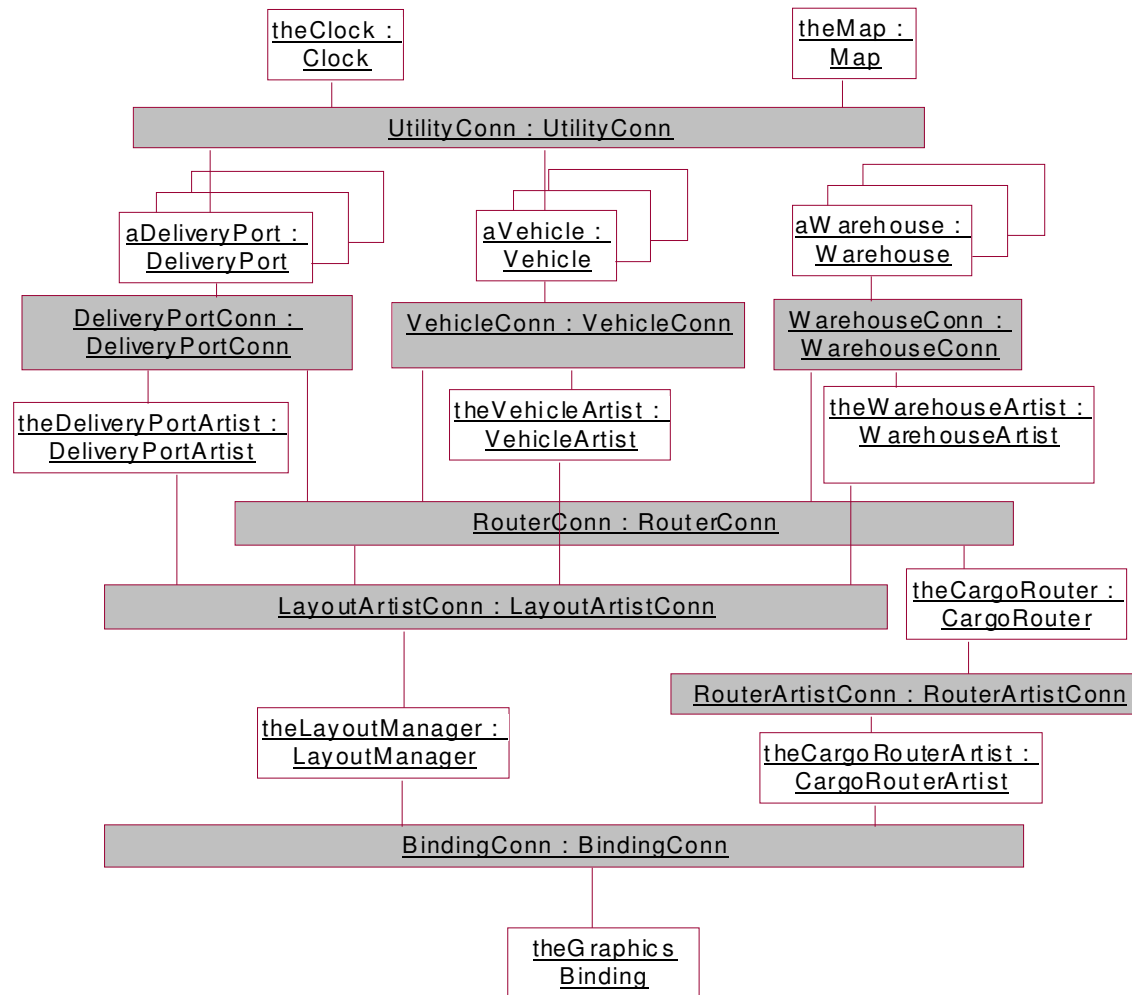
Overview of the Architectural Approach – C2



Transform into Standard UML: Components and Connectors



Transform into Standard UML: Architectural Configuration



Transform into Standard UML: Transformation Rules

Internal Object → Class

State Variable → Class Private Attribute

Component Invariant → Tagged Value + Class Documentation

Provided Operation → Class Operation

Required Operation → Class Documentation

Operation Pre/Post Condition → Pre/Post Condition on Class Operation

Message Return Type → Return Type on Class Operation

Message Parameter → Parameter (Name + Type) on Class Operation

Connector → Interface (<<Interface>> Class)

Connector Interface → Union of Operations of attached Objects/Interfaces

Message Originator → Operation <<Stereotype>>

Architecture Configuration (explicit invocation) → Object Diagram

Component Instance → Internal Object Class Instance

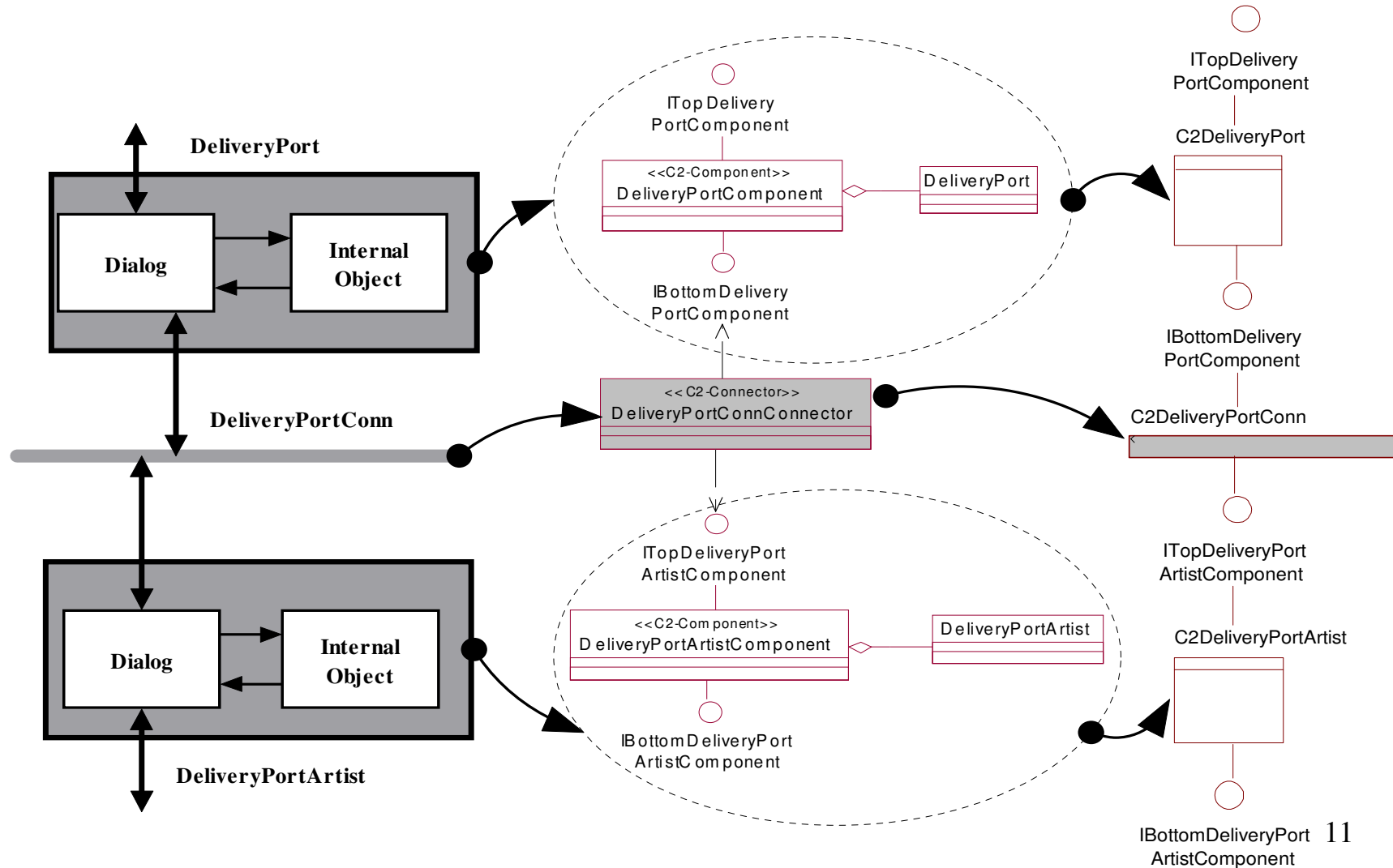
Connector Instance → <<Interface>> Class Instance

Component/Connector Binding → Object Link (instance of an association)

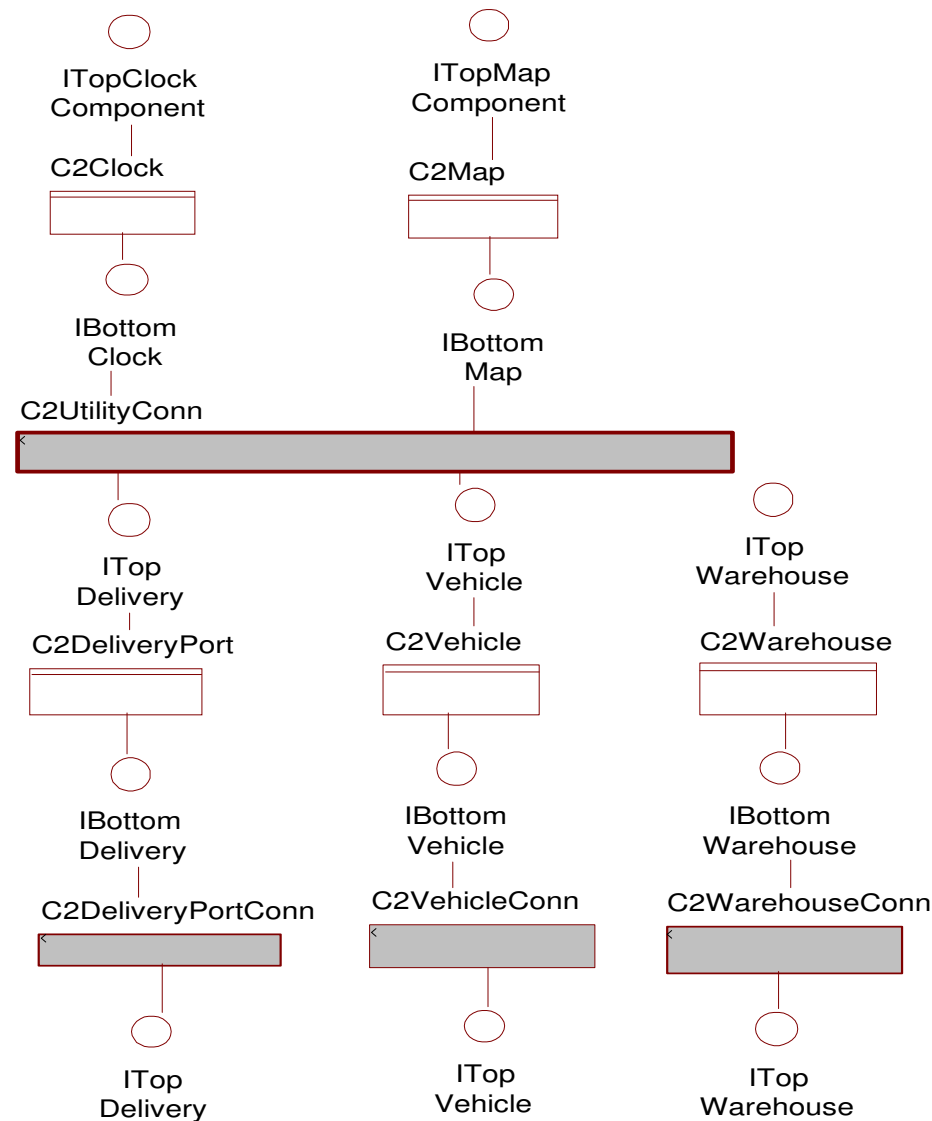
Transform into Standard UML: Tradeoffs

- Advantages:
 - Common notation
 - Representation can be manipulated by standard tools
- Limitations:
 - Violation of architectural constraints
 - Incomplete transformation: style-specific constructs,
...

Transform into UML with Extensions: Components and Connectors



Transform into UML with Extensions: Architectural Configuration



Transform into UML with Extensions: Transformation Rules

Component → <<C2-Component>> Class

Internal Object → <<C2-Component>> Class Attribute

Component Top Interface → <<Interface>> Class

Component Bottom Interface → <<Interface>> Class

Outgoing Request → <<Interface>> Class <<out>> Operation

Incoming Notification → <<Interface>> Class <<in>> Operation

Connector → <<C2-Connector>> Class

Connector Top Interface → Union of Bottom Interfaces of attached Components/Connectors

Connector Bottom Interface → Union of Top Interfaces of attached Components/Connectors

Architecture Configuration (implicit invocation + event notification) → Component Diagram

Component Instance → Component realizing...

Connector Instance → Component realizing...

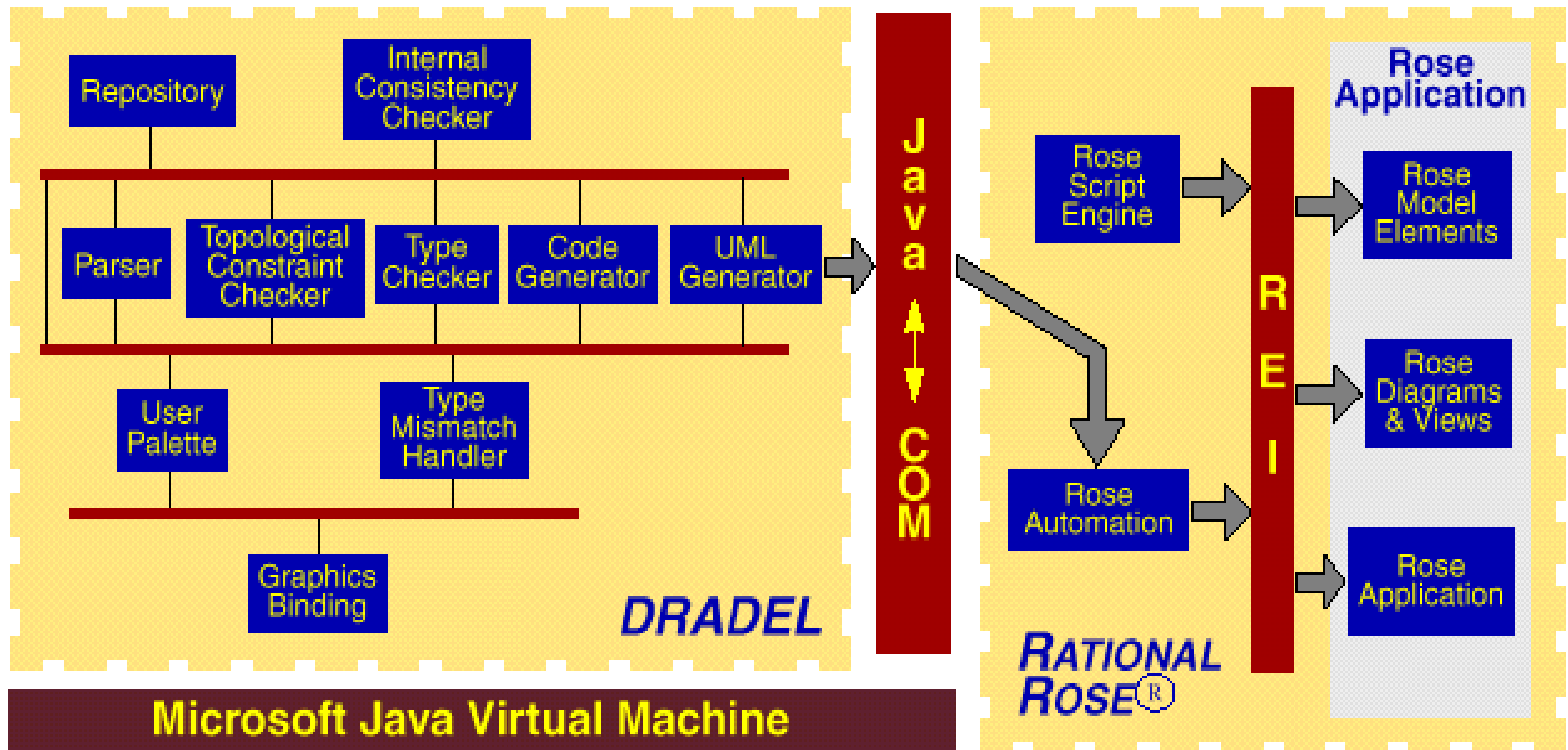
Transform into UML with Extensions: Tradeoffs

- Advantages:
 - Use UML's built-in extension mechanisms
 - Architectural style rules and constraints can be checked
- Limitations:
 - Requires complete specification of architectural style
 - Requires UML tools to support constraint definition languages (e.g., Object Constraint Language or OCL)
 - Limitations of UML to express all the information represented in an ADL

Leveraging Tool Support for ADLs and UML

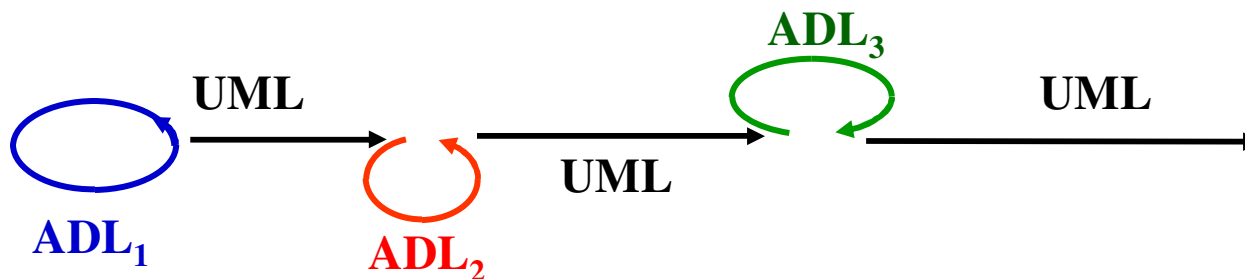
- Motivation
 - Abstract away complexity and automate repetitive tasks
 - Automatically add/translate constraints so that the designer can focus on refinement and not on violations to the architectural style
- iDRADEL-Rose
 - Analyze architecture using DRADEL capabilities
 - Select transformation rules
 - Generate initial UML model in Rose
- UML: Rational Rose
 - Refine initial model into a design
 - Perform code generation, reverse engineering, ...

Integrated Environment Architecture



Summary of Contributions

- Proposed an approach that combines benefits of ADLs and UML
 - Strength of ADLs for architecture-based analyses
 - Strength of UML for design
- Integrated tool support of ADL and UML:
 - ADL tools: type checking, constraint checking
 - UML tools: refinement, code generation, ...
- Similar approach can be applied to any ADL



Future Research Directions

- Analysis of a design represented in UML for conformance to a given architectural style
- Reverse engineering an architecture from a design or implementation
- Modeling dynamic behavior, e.g., using UML statechart diagrams
- Automating ADL-to-UML transformation for other candidate ADLs
- Adapt approach to proposed changes in UML, e.g., *Profiles*

Rational Rose - (untitled) - [Class Diagram: Logical View / Main]

File Edit View Browse Report Query Tools Add-Ins Window Help

Use Case View Logical View Main java sun Basic Classes Collection Classes C2 Components DeliveryPort CargoRouter LayoutManager Clock Warehouse CargoRouterArtist DeliveryPortArtist Vehicle WarehouseArtist Map VehicleArtist BindingConn RouterConn LayoutArtistConn DeliveryPortConn RouterArtistConn VehicleConn WarehouseConn UtilityConn PlannerSystem Component View Main java sun PlannerSystem C2DeliveryPort C2CargoRouter C2LayoutManager C2Clock C2Warehouse C2CargoRouterArtist C2DeliveryPortArtist

DeliveryPort

- cargo_val : StringCollection
- cargo : ShipmentCollection
- max_capacity : Integer
- capacity : Integer
- name : String
- selected
- internal_

Warehouse

- cargo_val : StringCollection
- cargo : ShipmentCollection
- max_capacity : Integer

CargoRouter

- current_trips : TripInfoCollection
- dist_start_to_end : Integer
- dist_end_by_start : String
- veh_in_transit : StringCollection
- dist_start : String

iDRADEL-Rose

Development of Robust Architectures using a Description and Evolution Language

File: D:\Marwan\DRADEL\CargoRouter\PlannerSystem.c2

Parse File Check Constr Type Check Generate Code Generate UML Exit

C2 ==> UML Transformation Rules:

```

***** Representing the C2 architecture *****
C2 Component -> UML <<Component>> Class
C2 Internal Object -> UML <<Component>> Class Attribute
C2 Component Top Interface -> UML <<Interface>> Class
C2 Outgoing Request -> UML <<Interface>> Class <<out>> method
C2 Incoming Notification -> UML <<Interface>> Class <<in>> method
C2 Component Bottom Interface -> UML <<Interface>> Class
Incoming Request -> UML <<Interface>> Class <<in>> method
Outgoing Notification -> UML <<Interface>> Class <<out>> method
C2 Connector -> UML <<Connector>> Class
C2 Connector Top Interface -> Union of Bottom Interfaces of attached Comp
C2 Request -> UML <<Connector>> Class <<request>> method
C2 Notification -> UML <<Connector>> Class <<notification>> method
C2 Connector Bottom Interface -> Union of Top Interfaces of attached Comp
C2 Request -> UML <<Connector>> Class <<request>> method
C2 Notification -> UML <<Connector>> Class <<notification>> method
C2 Architecture -> UML Component Diagram
C2 Component -> UML Component realizing <<Component>> class and its T
C2 Connector -> UML Component realizing <<Connector>> class, bottom <<
C2 Component Types -> Not Supported
C2 Connector Types -> Not Supported

```

Select All Deselect All

Status Log:

```

Parsing ...
Complete
Checking Topological Constraints ...
Complete
Type Checking ...
Complete
Generating UML model ...
Complete

```

UML Generation Options:

Open JDK 1.1.4 as template

For Help, press F1

Supporting Slides

Rules of the C2 Style

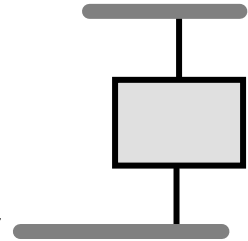
- Components, connectors (buses), and their configurations
- Substrate independence:
 - a component is only aware of components “above” it and is completely unaware of components “beneath” it
- Implicit Invocation:
 - “listeners” register interest in events
 - “announcers” are unaware of the listeners

Rules of the C2 Style (continued)

- Communication by exchanging asynchronous messages:
 - notifications of completed services sent downward
 - announcements of state changes of the internal object of a component
 - service requests sent upward
 - directives from components below requesting that an action be performed by some set of components above

C2 Components

- Connection points: "top" and "bottom"
 - Top (bottom) of a component can only be attached to bottom (top) of one bus.
 - Components only communicate via connectors: direct communication is disallowed.
- Component cannot be attached to itself



C2 Components (continued)

- Canonical internal architecture:

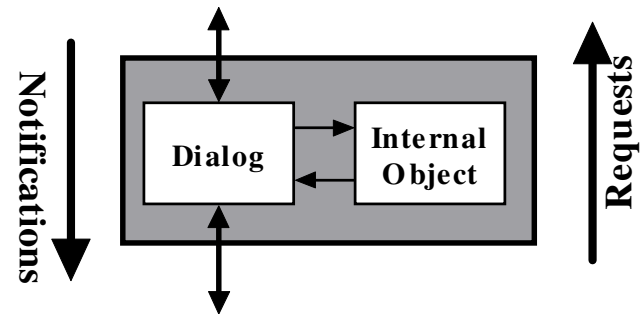
- Internal object

- arbitrarily complex
- has a defined interface

- Dialog

- invokes access routines of the object
- is in charge of interacting with the rest of the architecture via events.

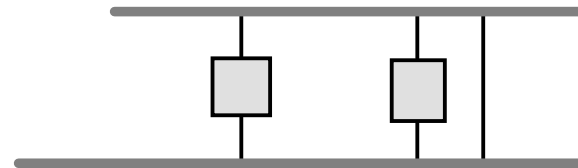
- Separates communication from computation



C2 Connectors

- Communication message routing and filtering devices

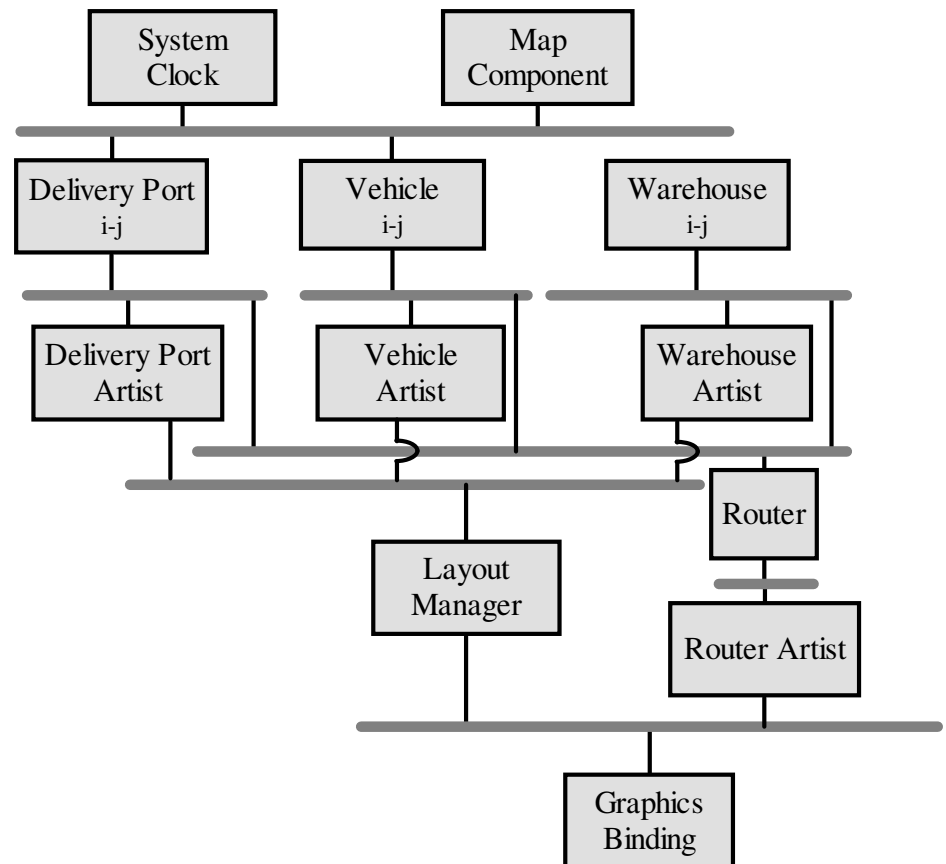
- multicast
- point-to-point



- Connector-to-connector links allowed
- No bound on number of components or connectors attached to a connector
- Context-reflective interfaces:
 - function of attached components/connectors

C2 Architecture: Cargo Routing System

- Logistics system for routing incoming cargo to a set of warehouses
- *DeliveryPort*, *Vehicle*, and *Warehouse* keep track of the state of a port, a transportation vehicle, and a warehouse



C2SADEL Specification

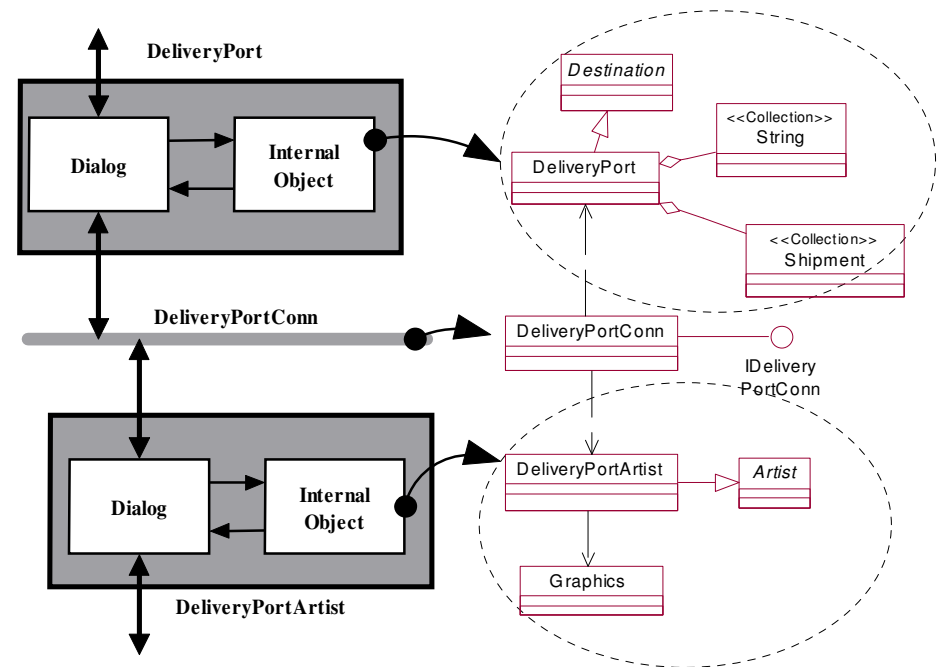
```
architecture CargoRoutingSystem is {
  component_types {
    component DeliveryPort is extern {DeliveryPort.c2;}
    ...
  }
  connector_types {
    connector FilteringConnector is {filter msg_filter;}
    ...
  }
  architectural_topology {
    component_instances {
      aDeliveryPort : DeliveryPort;
      theDeliveryPortArtist: DeliveryPortArtist;
      ...
    }
    connector_instances {
      UtilityConn : FiltConn;
      ...
    }
    connections {
      ...
      connector DeliveryPortConn {
        top aDeliveryPort;
        bottom theDeliveryPortArtist;;
      }
      ...
    }
  }
}
```

C2 Component Specification

```
component DeliveryPort is subtype CargoRouteEntity (int \and beh) {
  state {
    cargo          : \set Shipment;
    selected       : Integer;
    ...
  }
  invariant {
    (cap >= 0) \and (cap <= max_cap);
  }
  interface {
    prov ip_selshp: Select(sel : Integer);
    req  ir_clktck: ClockTick();
    ...
  }
  operations {
    prov op_selshp: {
      let num : Integer;
      pre num <= #cargo;
      post ~selected = num;
    }
    req or_clktck: {
      let time : STATE_VARIABLE;
      post ~time = time + 1;
    }
    ...
  }
  map {
    ip_selshp -> op_selshp (sel -> num);
    ir_clktck -> or_clktck ();
    ...
  }
}
```

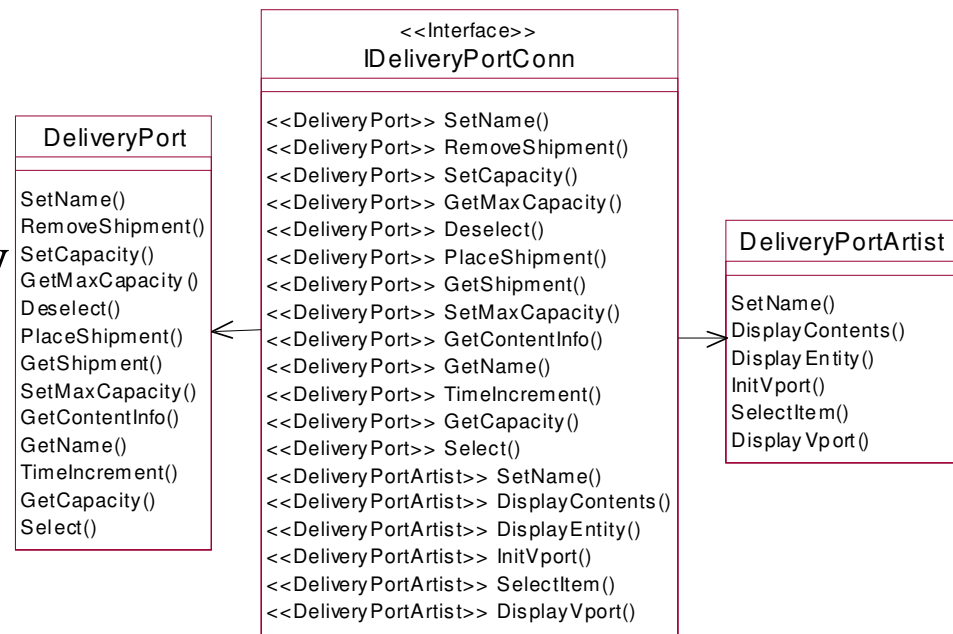
Transform into Standard UML

- Internal objects → UML classes
- Connectors → UML Interfaces
- Express arbitrary complexity using *native* UML constructs (aggregation, inheritance, ...)



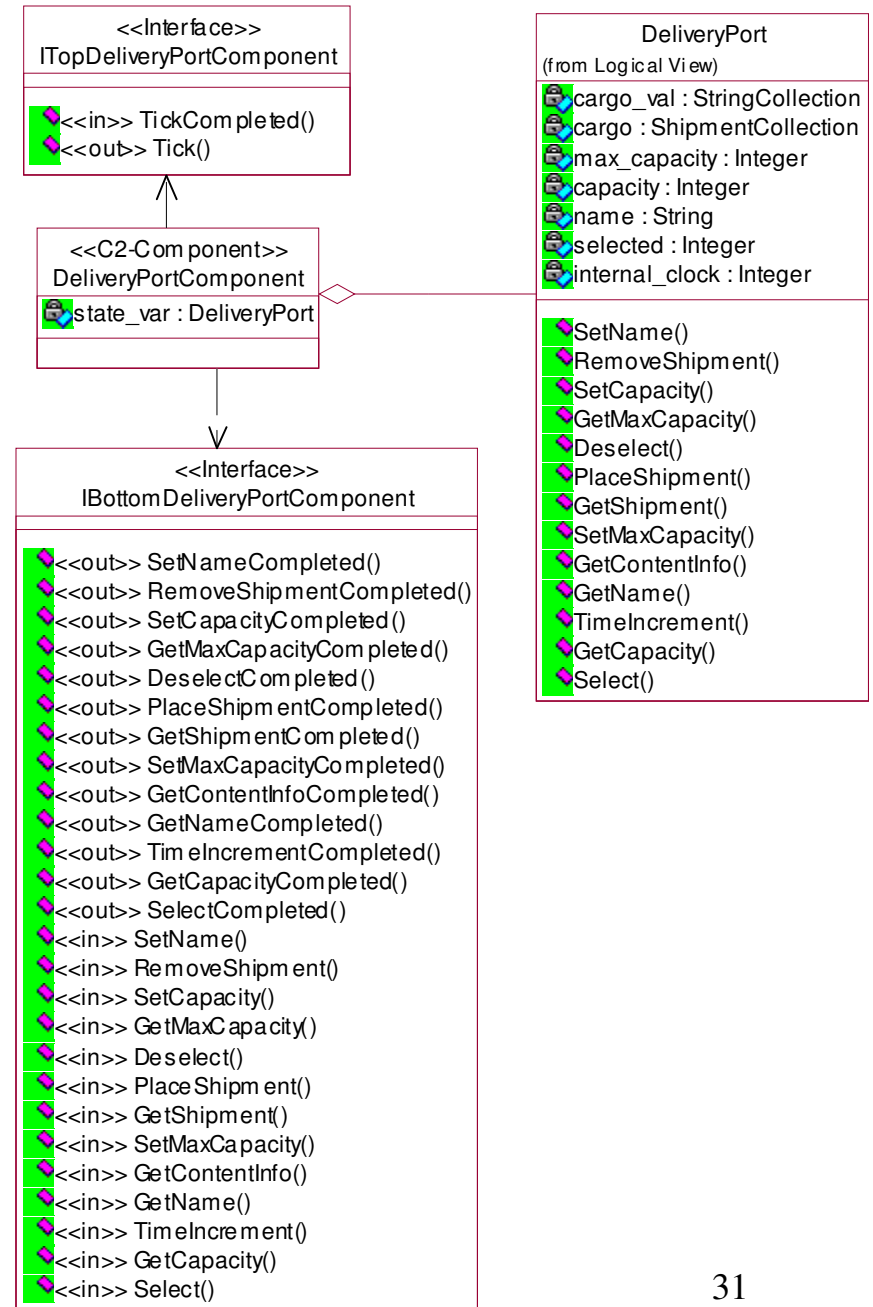
Enforcing the Rules of the Architectural Style

- Context-reflective property:
 - operations provided by the interface are roughly the union of the provided operations of all components attached to the bus



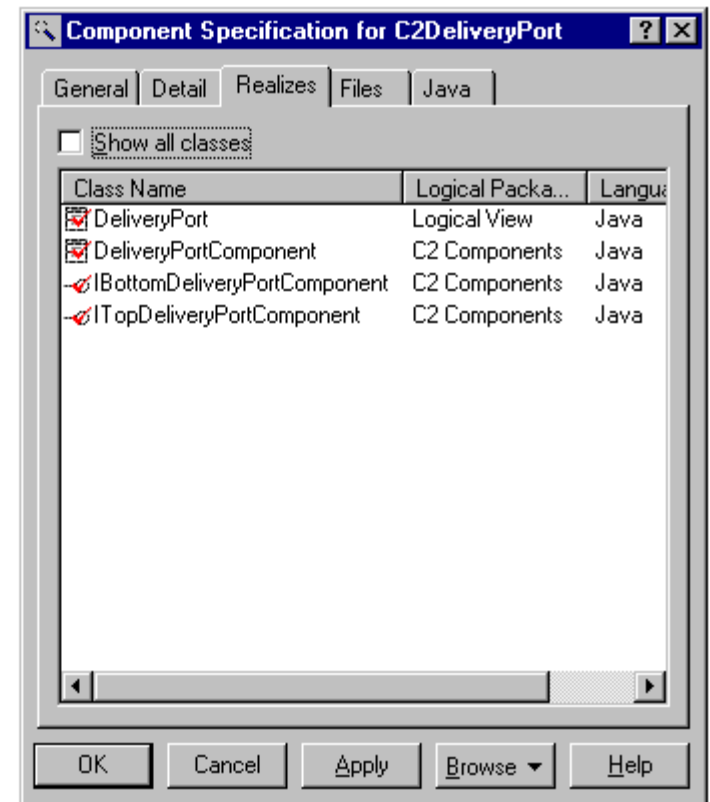
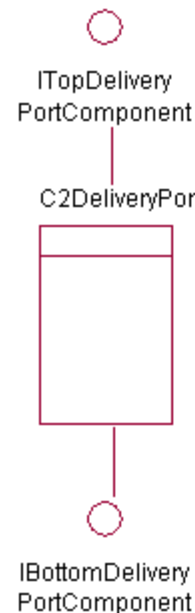
Transform into UML with Extensions

- *DeliveryPortComponent* class has top and bottom <<Interface>> classes, *ITopDeliveryPort-Component* and *IBottomDeliveryPort-Component*
- Intermediate connector is mapped to a <<Connector>> class, *DeliveryPortConn-Connector*



Transform into UML with Extensions: UML Component

- UML Component (C2DeliveryPort) for C2 Component (DeliveryPortComponent) Realizes:
 - <<C2-Component>> class (DeliveryPortComponent)
 - top and bottom interfaces (ITopDeliveryPortComponent, IBottomDeliveryPortComponent)
 - classes representing internal object (DeliveryPort, ...)

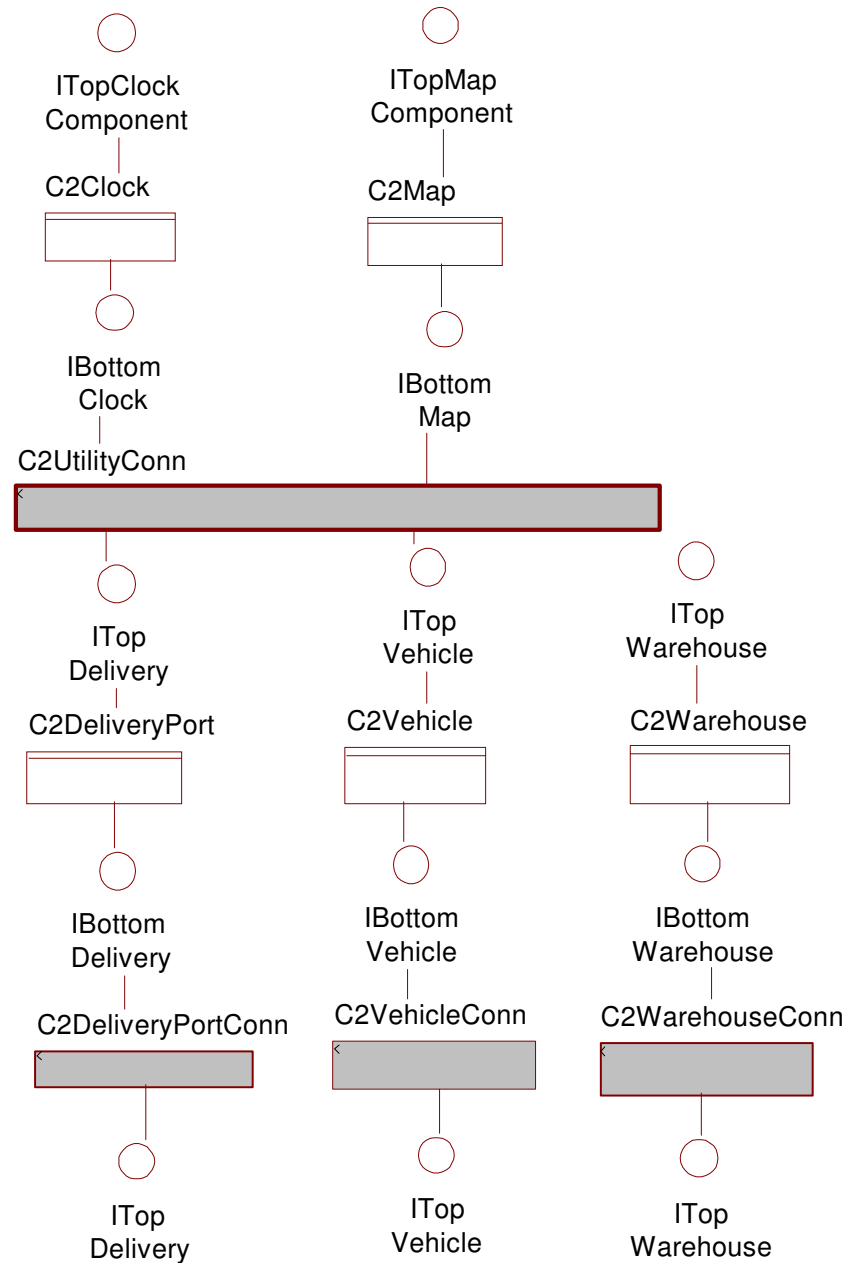


Transform into UML with Extensions: UML Component

- UML Component for a C2 connector realizes:
 - <<C2-Connector>> class
 - bottom interfaces of Components/Connectors above
 - top interfaces of all Components/Connectors below

UML Component Diagram

- UML Component for a C2 Connector realizes:
 - <<Connector>> class
 - bottom <<Interface>> classes of Components and Connectors above
 - top <<Interface>> classes of Components and Connectors below



Some UML Limitations

- UML cannot express all the information represented in an ADL
- UML support for sub-typing does not express the heterogeneous component subtyping mechanisms provided in C2SADEL
 - type v/s class in UML
 - interface/behavior/name/implementation inheritance in C2SADEL