Outline

- **Background**
  - Architecture-Based Approaches
  - Current Tool Support

- **Integrating Tool Support**
  - Alternatives
  - Tradeoffs
  - Proposed Architecture
  - Illustrations
Definitions of Software Architecture

High-level structure:

- description of elements from which the system is built (components)
- description of interactions among those elements (connectors)
- composition of the components and connectors into configurations or topologies
- constraints
Key Software Architecture Concepts

- Canonical building blocks:
  - components
    - unit of computation and state
    - Example: Abstract Data Type (ADT), ...
  - connectors (or buses)
    - model interactions among components
    - Examples: procedure call, shared variable access...
  - configurations
    - connected graph of components and connectors
    - Example: “Layered” configuration
Architectural Styles

- Architectural Style:
  - recurring organizational pattern
  - constrains selection of components/connectors
  - determines allowed compositions and interactions of components and connectors

- Examples:
  - Pipe and Filter
  - Client/Server
Architecture-Based Approaches

- **Research:**
  - Rigorous modeling notations
  - Powerful analysis techniques
  - Special-purpose solutions

- **Practice:**
  - Choose practicality over rigor
  - Address entire software life cycle
  - General-purpose solutions
Describing an Architecture

- Architecture description languages (ADLs):
  - modeling and analysis of architectures
  - formal notations
  - limited support for development
  - limited tool support

- Unified Modeling Language (UML):
  - general purpose visual modeling notation
  - semi-formal (ambiguous)
  - expressive and extensible
  - standardized tool support
Integrating Architecture-Based Approaches

- **Coarse-Grained Architecture:**
  - Specify components, connectors, ...
  - Express in an ADL
  - Use tool support for modeling and analysis
- **Transform from an ADL to UML**
- **Design and Implementation:**
  - Specify design constructs (classes, ...)
  - Express in UML
  - Use tool support for development
Integrating Both Approaches

Architecture-based specification, modeling and analysis environment (e.g., DRADEL)

Design environment (e.g., Rational Rose®)

Code Generation and Development Environment

class Class 1 extends Window {
    public ....
}

Architecture in ADL       Architecture in UML       Design in UML       Implementation
Current State of Tool Support

- **Research-Off-The-Shelf (ROTS) tools:**
  - Overlap in functionality
  - Highly specialized
  - Poorly integrated
  - Poorly documented and supported
  - Mostly prototypes

- **Commercial-Off-The-Shelf (COTS) tools:**
  - Address entire life cycle
  - Mature and extensible
  - Well supported
Need for Tool Support

- **Abstract complexity from the user**
  - Formalism can be hidden behind a graphical user interface

- **Automate repetitive activities**
  - Apply transformation rules
  - Generate (Semi-) Formal constructs

- **Integrating Heterogeneous Environments**
  - Overlap in functionality
  - Architectural “mismatches”
How to Integrate Tools?

- **Architecture-Centric Approaches**
  - Top-down decomposition
  - Miss out on reuse

- **Component-Centric Approaches**
  - Bottom-up composition
  - May not achieve “optimal” architecture

- **Middleware (CORBA, D/COM, …)**
  - Imposes constraints on architecture
Middleware

- Infrastructure that supports (distributed) component-based application development
  - standard for constructing and interconnecting components
- Microsoft (Distributed) Component Object Model (D/COM)
  - OLE Automation
- C2 Implementation Framework
Overview of OLE Automation

Microsoft's technology for cross-application macro programming

Automation:
- a protocol
- any automation controller can use every automation object
- any automation object can be integrated with every automation controller
**Key Concepts in OLE Automation**

- **Automation Object:**
  - programmable component
  - described in a type library

- **Automation Controller (e.g., Visual Basic):**
  - programming (“scripting”) environment
  - drive or integrate Automation Objects

- **Automation Server:**
  - make Automation Objects available and accessible to one or more Automation Controllers
Dynamic Link Libraries (DLL)
- Exported Functions
- Interface Pointers
- In-process objects (get loaded in the same process space as calling application)

Tradeoffs:
👍 No need for marshalling
👍 Better performance
👎 Can crash host application
👎 Possible security breaches
👎 Cannot run standalone
EXE COM Servers

- Windows Executable (*.EXE)
  - Out-of-process
- Tradeoffs:
  - Cannot easily crash host application
  - Require lower security “clearance”
  - Can run standalone
  - Require marshalling
  - Slower performance
Overview of C2 Style

- Network of components and connectors:
  - concurrent (multi-threaded) components
  - no assumption of a shared address space
  - component heterogeneity

- 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
Communication by exchanging asynchronous messages:

- message = name + typed parameters
- notifications sent downward
  - announcements of state changes of the internal object of a component
- requests sent upward
  - directives from components below requesting that an action be performed by some set of components above
C2 Implementation Infrastructure

- C2 Class Framework
  - components
  - connectors
  - messages, ...

- Graphics Component
  - accept notifications from C2 components above it and translate as calls to graphics toolkit methods (e.g., Java AWT)
  - transforms user events, generated in the graphics toolkit into C2 requests
Selected Integration:
- Candidate ADL: C2SADEL
- Selected UML Tool: Rational Rose

iDRADEL-Rose
- DRADEL:
  - analysis (constraints checking, ...) in ADL
  - select transformation rules from ADL to UML
  - generate initial UML model in Rose
- Rose
  - refine initial model into a design
  - perform code generation, ...
Rational Rose ®

- Commercial-off-the-shelf (COTS)
- No source code available
- Capabilities:
  - Visual modeling tool
  - Support for UML and its extensions
    - stereotypes
    - tagged values
  - Code Generation (C++, Java, ...)
  - Reverse Engineering (C++, Java, ...)

22
Rose Extensibility Interface (REI)

- Automation Objects provide read/write access:
  - model elements (packages, classes, …)
  - properties or name-value pairs (user-extensions, …)
  - diagrams (graphical properties of model elements)
  - application (scripts, addins…)

- Rose Script
  - Automation Controller

- Rose Automation
  - Automation Server
Rational Rose Add-ins

- Framework for Rose to accept an 'add-in' feature for inclusion in Rose
- Basic Add-in: supplies its own responses for events to execute third-party scripts or executables
- Add-ins register for Rose Events through settings in Windows Registry
Rational Rose Events

- In response to certain user actions
  - Examples:
    - `OnApplInit`: when Rose is started
    - `OnActivate`: on Rose startup and when add-in is activated via the Add-In Manager
    - `OnDeactivate`: on Rose shutdown and when add-in is deactivated via the Add-In Manager

- **Script** events invoke Rose Scripts

- **Interface** events fire registered COM Servers (DLL)
Rational Rose® Environment

- **Rose Extensibility Interface (REI)**
  - Rose Application
  - Diagrams
  - Model Elements
  - Add-Ins
  - Properties

- **Add-In**
  - RoseScript
    - Report Generation
    - Custom Dialogs
    - Consistency Checking
    - Project Estimation
    - Metrics
    - Wizards
    - Import
    - Export

- **Events**
  - RoseScripts
  - OLE Server

- **Registries for Add-Ins**
  - Events
  - Stereotype Files
  - Property Files
  - Menu File
  - Data Types

- **Menus**
  - Rose Scripts
  - External Applications

- **Stereotypes**
  - Icons
  - Strings

- **Extensible Properties**
  - New tools/tabs
  - New properties

- **OLE Clients**
  - Visual Basic Application
  - Visual C++ Application
  - Visual J++ Application
  - Microsoft Office 97 VBA Scripts

- **OLE Servers**
  - Visual Basic Applications
  - Visual C++ Applications
  - Visual J++ Applications
  - Microsoft Office 97 Applications

- **Exchange Data Files**
  - External Applications
  - Development Environments
  - Data Modeling Tools
  - Persistence
  - ...
DRADEL

- Research-off-the-shelf (ROTS)
- Source Code available (Java)
- Designed in the C2 style
- Implemented as a multi-threaded architecture using the C2 framework
DRADEL Environment

- Checking:
  - consistency
  - topological constraints
  - type conformance
  - ...
- Skeleton Application Generation
  - C2 implementation infrastructure
Alternatives Considered

- Avoid Heterogeneity Problem
- Exchange Data Files
- Bring DRADEL to Rose
- Re-implement DRADEL in Rose
- DRADEL as Rose Add-In
- DRADEL as Automation Controller

Avoid Heterogeneity Problem
Option #1: Avoid Heterogeneity

- Use UML tools with available source code (e.g., Argo/UML in Java)
- Tradeoffs:
  - Not as mature as commercial products
  - Poorly documented code
  - Simple Application Programming Interface (API) does not provide advanced capabilities (introspection, programming language independence, …)
Option #2: Exchange Data Files

- Have DRADEL generate a file in the format expected by Rose
- Have Rose read/write files in the format expected by DRADEL
- Tradeoffs:
  - Formats are likely to change
  - Works only for text (ASCII) files
  - Requires writing parsers
Option #3: Bring DRADEL to Rose

- Package DRADEL as a Java Bean®, then bridge DRADEL Bean with Rose using Sun's Active X Bridge
- Tradeoffs:
  - Non-trivial modifications to DRADEL: may limit the extensibility of the framework
  - Current support exists only for packaging a Java Bean as an ActiveX, but not the other way around
Option #4: Re-implement DRADEL

- Re-implement DRADEL functionality in Rose using Scripts or AddIns
  - Parser
  - Type Checker
  - ...

- Tradeoffs:
  - UML cannot describe all architectural information described in an ADL
  - Duplicating instead of reusing functionality
Option #5: Automation Controller

- Use DRADEL as Automation Controller
  - Port DRADEL to Microsoft Visual J++ 6.0
  - Generate Java-Callable Wrappers (JCW) for Rose Automation Objects
  - Add *UMLGenerator* Component to DRADEL
    - traverse architectural representation
    - initialize Rose application,
    - create Rose model elements and diagrams ...
  - Modify *UserPalette* Component

- Use Rose as Automation Server
Out-of-process DRADEL

- Repository
- Internal Consistency Checker
- Parser
- Topological Constraint Checker
- Type Checker
- Code Generator
- UML Generator
- User Palette
- Type Mismatch Handler
- Graphics Binding
- DRADEL
- Rose Script Engine
- Rose Extensibility Interface (REI)
- Rose Automation
- Rose Elements
- Rose Diagrams and Views
- Rose Application

Microsoft Java Virtual Machine
Microsoft Windows

modified component
new component
process boundary
Option #6: Rose Add-In

- Use DRADEL as Automation Server:
  - Java object as COM-Callable Wrapper (CCW)
  - Package as a Dynamic-Link Library
  - Register as Rose AddIn
  - Subscribe to Rose events:
    - OnNewModel
    - OnDeActivate
    - OnApplInit

- Automation Controller (same as before)

- Use Rose as Automation Server/Controller
DRADEL as Rose Add-In

COM Automation Server (DLL) and COM Automation Client

Repository
Internal Consistency Checker
Parser
Topological Constraint Checker
Type Checker
Code Generator
UML Generator
User Palette
Type Mismatch Handler
Graphics Binding
DRADEL-Rose Add In

Rational Rose ®

Add-In

Rose Extensibility Interface (REI)
- Rose Application
- Diagrams
- Model Elements
- Add-Ins
- Properties

Registry for Add-Ins
- Events
- Stereotype Files
- Property Files
- Menu File
- Data Types

Events
- COM Server

Menus
- COM Server

Stereotypes
- Icons
- Strings

Properties
- New tools-tabs
- New properties
Integrating Java and COM

- Microsoft Java Virtual Machine
  - Java objects accessible as COM objects
  - COM objects accessible as Java objects
  - handles object references, cleanup, and interface queries for any COM object used in Java
  - language independence: the component can be written in Java or another COM-compliant language.
Microsoft Java Virtual Machine
Using COM objects in Java

- Generate a Java-Callable Wrapper (JCW) for the COM object

- jactivex tool:
  - provided in the Microsoft SDK for Java
  - used for hosting ActiveX controls as JavaBeans components
  - generates JCW Java classes from a type library file (e.g., RationalRose.tlb) produced from a compiled Interface Definition Language (IDL) specification
**@com directives**

- Generated by jactivex tool
- Specify how the Java object maps to the COM equivalent and vice versa
- Can be manipulated to suit the particular application’s needs
- Converted into Java class attributes needed by the Java/COM integration subsystem in the Microsoft Win32 Virtual Machine for Java
import rationalrose.*;    // import Rational Rose Automation Objects package

public class UMLGenComponent extends ComponentThread {
    private IRoseApplication iRoseApp;
    private IRoseModel iRoseModel;
    ...
    private void Initialize()
        throws ComException
    {
        try
        {
            // An important restriction on using Java classes that wrap COM
            // classes is that you must use the instance through an interface.
            iRoseApp = (IRoseApplication) new RoseApplication();
            iRoseModel = iRoseApp.NewModel();
            ...
        }
        ...
        catch (ComException e)
        {
            ...
        }
    }
}
Advantages of Integrated Environment

- DRADEL is still independently extensible
- By the rules of the C2 style:
  - can include new components without affecting interaction with Rose
  - can be considered as a Composite Component
Advantages of Integrated Environment

- Rose is still independently extensible
- Automation
  - Manipulate objects exposed by other COM-enabled applications from within Rose applications
  - Manipulate Rose objects from other COM-enabled applications
  - Write new applications that manipulate Rose objects and other COM objects

- Rose Scripts
  - Model Checking, Report Generation, etc…
Architectural Mismatches

- Mismatched assumptions a reusable part makes about the system it is to be part of
- Often conflict with the assumptions of other parts
- Are almost always implicit, making them extremely difficult to analyze before building the system
Multi-threading and Garbage Collection:

- DRADEL is multi-threaded
- Garbage Collection of COM objects:
  - Some COM objects can be called only on the thread on which they were created
  - Garbage collection occurs at unpredictable times
  - The required thread may have expired or may be no longer responding to messages by the time garbage-collection reclaims the object
    - COM Object does not get released: memory leak!
    - Solution: explicitly release COM object
C2 implementation infrastructure problems
- Routing messages between an overall architecture and a composite component in that architecture
Architectural Mismatches (continued)

- One-way communication (DRADEL → Rose)
  - Not a Connectable Object (*IConnectPoint*)
  - No notifications (e.g., ClassAdded) in response to requests (e.g., AddClass)

**Diagram:**

1. *IConnectionPointContainer::FindConnectionPoint* to get *IConnectionPoint* interface
2. *IConnectionPoint::Advise* passing sink Interface pointer
3. Object calls members in sink interface as appropriate until *IConnectionPoint::Unadvise* is called
Enabling two-way communication

- Solution #1:
  - traverse Rose Model Elements
  - extract architecturally-relevant information
  - compare with the current representation
  - Very computationally expensive

- Solution #2:
  - have Rose store model in external repository
  - have repository generates triggers in response to changes in model elements
  - Extract architecturally-relevant information
  - Update architectural representation in DRADEL
Limitations of Integrated Environment

- Synchronous Procedure Calls
  - C2 style uses asynchronous events
  - COM interfaces make calls synchronous

- Performance:
  - Depends on amount of information passed
  - In-process is almost TWICE as fast as out-of-process!
Limitations of Integrated Environment (continued)

- Managing Component Lifetimes
  - Many components could potentially terminate the Java Virtual machine!
  - Could not release “in time” Rose COM Object
  - Shutdown has to be done in a specific order
  - Shutdown might lead to deadlock

- Vendor/Platform Dependence
  - Java/COM integration requires Microsoft Java Virtual Machine
  - COM not available on all UNIX platforms (currently only Solaris)
Summary of Contributions

- Integrated environment to assist software architects in refining a coarse grained architecture into a design, and eventually an implementation.
- Uncovered various contradictory assumptions (architectural mismatches) implicit in the frameworks, middleware, tools and environments.
Conclusions

- Similar approach can be followed with:
  - other Architecture Description Languages
  - other Architecture-Based tools

- This approach will be adapted to:
  - changes in our understanding of UML
  - changes in UML itself
Example: **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
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;;
            }
            ...
        }
    }
}
Overview of 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**
Overview of C2 Components

- 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
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 ()
    ...
  }
}
Overview of 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
ADL to UML Transformation

- Non-architectural concerns become UML constructs (+ OCL)
- Architectural concerns become UML with extensions

- stereotypes (available on most model elements)
- tagged-values (as Rose Properties)
Representing Non-Architectural Concerns in UML

- Internal objects → UML classes
- Connectors → UML<<interface>> classes
- Express arbitrary complexity using *native* UML constructs (aggregation, inheritance, …)
Transformation Rules - I

Internal Object → Class
- State Variable → Class Private Attribute
- Component Invariant → Tagged Value + Class Documentation
- Provided Operation → Class Method
- Required Operation → Class Documentation
- Operation Pre/Post Condition → Pre/Post Condition on Class Method
- Message Return Type → Return Type on Class Method
- Message Parameter → Parameter (Name + Type) on Class Method

Connector → <<Interface>> Class
- Connector Interface → Union of Methods of attached Objects/Interfaces
- Message Originator → Method <<Stereotype>>

Architecture Configuration → Collaboration Diagram
- Component Instance → Internal Object Class Instance
- Connector Instance → <<interface>> Class Instance
- Component/Connector Binding --> Object Link (instance of an association)
Specification for Internal Object

DeliveryPort

cargo_val : StringCollection
cargo : ShipmentCollection
max_capacity : Integer
capacity : Integer
name : String
selected : Integer
internal_clock : Integer

SetName()
RemoveShipment()
SetCapacity()
SetMaxCapacity()
Deselect()
PlaceShipment()
GetShipment()
SetMaxCapacity()
GetContentInfo()
GetName()
TimeIncrement()
Select()

<<Interface>>

<<DeliveryPort>>

DeliveryPortConn

<<DeliveryPort>> SetName()
<<DeliveryPort>> RemoveShipment()
<<DeliveryPort>> SetCapacity()
<<DeliveryPort>> GetMaxCapacity()
<<DeliveryPort>> Deselect()
<<DeliveryPort>> PlaceShipment()
<<DeliveryPort>> GetShipment()
<<DeliveryPort>> SetMaxCapacity()
<<DeliveryPort>> GetContentInfo()
<<DeliveryPort>> GetName()
<<DeliveryPort>> TimeIncrement()
<<DeliveryPort>> GetCapacity()
<<DeliveryPort>> Select()
<<DeliveryPortArtists>> SetName()
<<DeliveryPortArtists>> DisplayContents()
<<DeliveryPortArtists>> DisplayEntity()
<<DeliveryPortArtists>> InitVport()
<<DeliveryPortArtists>> SelectItem()
<<DeliveryPortArtists>> DisplayVport()

DeliveryPortArtist

width : Integer
entity_name : String
height : Integer
vport_xpos : Integer
vport_bg : String
contents : StringCollection
vport_name : String
vport_ypos : Integer
selection : Integer
vport_fg : String

SetName()
DisplayContents()
DisplayEntity()
InitVport()
SelectItem()
DisplayVport()
Specification for Internal Object (Continued)
Based on explicit method invocations and completely bypasses dialogs
- Instances of classes → internal objects
- Instances of interface classes → "connectors"
- object links → component/connector communications
Representing Architectural Concerns in UML

- Components/Connectors → Stereotyped Classes + Components realizing classes Internal Objects + Dialogs
Transformation Rules - II

Component $\rightarrow$ $<<$C2-Component$>>$ Class
  - Internal Object $\rightarrow$ $<<$C2-Component$>>$ Class Attribute
  - Component Top Interface $\rightarrow$ $<<$Interface$>>$ Class
  - Component Bottom Interface $\rightarrow$ $<<$Interface$>>$ Class
  - Outgoing Request $\rightarrow$ $<<$Interface$>>$ Class $<<$out$>>$ method
  - Incoming Notification $\rightarrow$ $<<$Interface$>>$ Class $<<$in$>>$ method

Connector $\rightarrow$ $<<$C2-Connector$>>$ Class
  - Connector Top Interface $\rightarrow$ Union of Bottom Interfaces of attached Components/Connectors
  - Connector Bottom Interface $\rightarrow$ Union of Top Interfaces of attached Components/Connectors

Architecture Configuration (implicit invocation + event notification) $\rightarrow$
  Component Diagram
  Component Instance $\rightarrow$ Component realizing...
  Connector Instance $\rightarrow$ Component realizing...
Specification for Component

- **DeliveryPortComponent** class has top and bottom **<<Interface>>** classes, **ITopDeliveryPortComponent** and **IBottomDeliveryPortComponent**

- Intermediate connector is mapped to a **<<Connector>>** class, **DeliveryPortConnector**
UML Component

- UML Component for C2 Component Realizes:
  - `<Component>` class
  - top and bottom `<Interface>` classes
  - classes representing internal object
Sequence Diagrams

- Represent component interactions, instead of object interactions
  - Example: Request/Notification Sequence

Diagram:

- : DeliveryPort Component
  - Request
    - 1: SetName(String)
  - Notification
  - 2: SetNameCompleted(String)
- : DeliveryPort ArtistComponent
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
For More Information

- Refer to paper submitted to UML ’99 conference:

*Enabling the Refinement of a Software Architecture into a Design*

by Marwan Abi-Antoun and Nenad Medvidovic