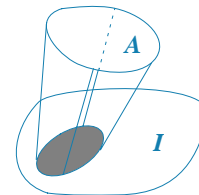
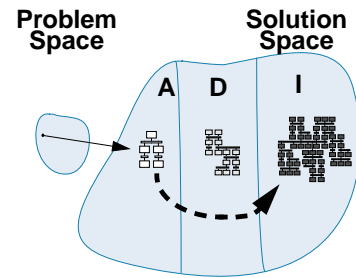


## Review — From Architecture to Implementation

- Directly implementing an architecture is infeasible
  - reduces to transformational programming
- Possible by limiting the target space
  - middleware platforms
  - software bus technologies



## Review — What is Middleware?

- Infrastructure that supports (distributed) component-based application development
  - a.k.a. distributed component platforms
  - mechanisms to enable component communication
  - mechanisms to hide distribution information
  - (large) set of predefined components
- Elements of Middleware
  - software components
    - component interfaces
  - containers
  - metadata
  - integrated development environment

## Review — Architectures and Middleware

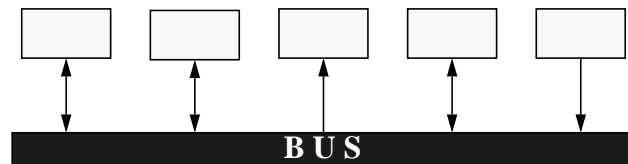
- Architecture-based development is top-down
    - decomposition
  - Component-based development is bottom-up
    - composition
  - Middleware imposes certain architectural constraints
    - usage of middleware and predefined components can influence the architecture
  - Architecture can impose constraints on middleware
    - specifying the architecture restricts the number of middleware options
  - Architecture  $\approx$  Framework + Framework Methodology
- *Finding suitable middleware can aid in the implementation of an architecture*

## Software Interconnection Technologies

- Middleware mostly focuses on components
    - components are homogeneous
    - intra-middleware OTS reuse
    - connectors are hidden in the middleware infrastructure
  - Software interconnection (bus) technologies focus on component interactions
    - components may be heterogeneous
    - wider reuse possible
- *Components do not “live” in the bus’s world; they attach to it*

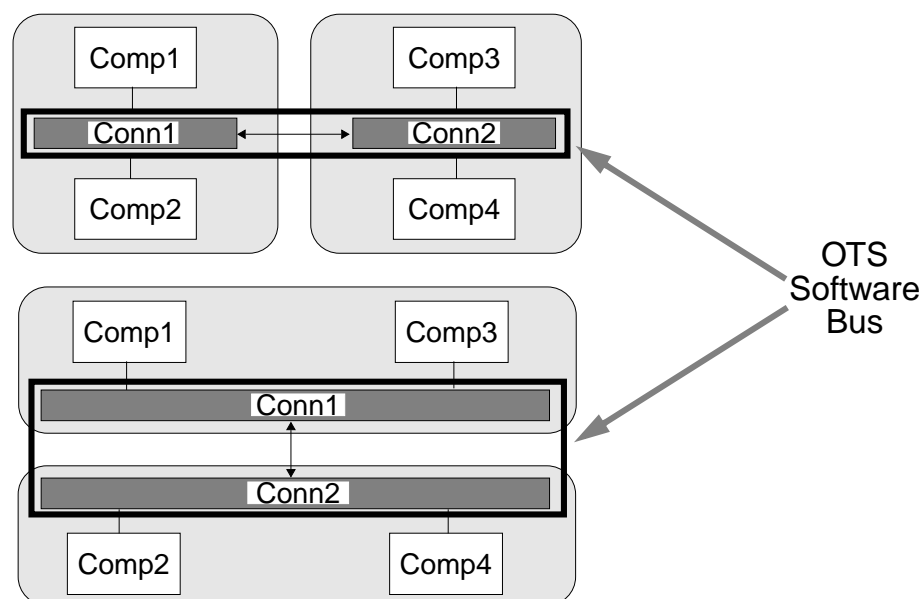
## Software Buses and Architectures

- Buses still require an architectural approach to application development
  - they are the assembly language of software composition
  - e.g., what is the architecture of the below system?

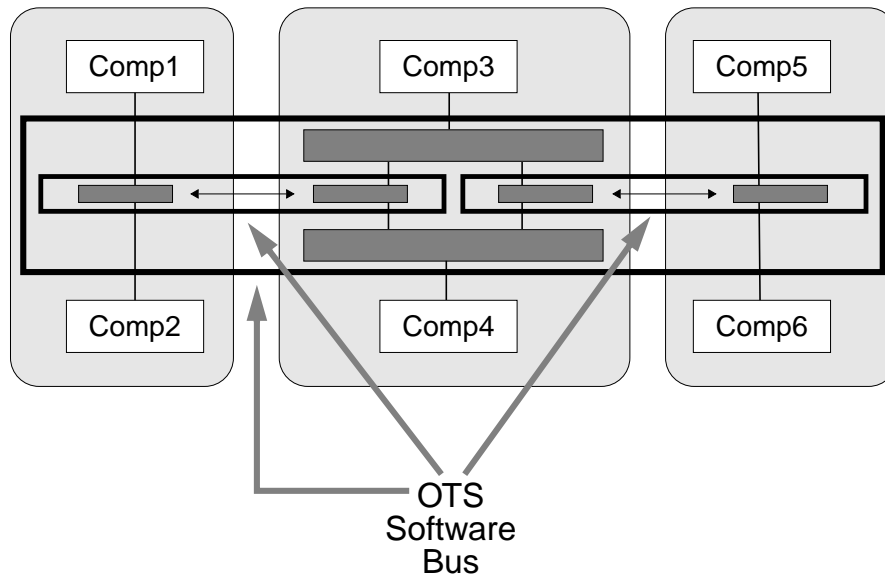


- Buses exhibit properties reminiscent of connectors
  - they represent an implementation counterpart to conceptual connectors at the level of architecture
  - such technologies can be exploited in architectures
  - aid with distribution, heterogeneity, interaction mismatches

## Connectors as Facilitators of Distribution and Heterogeneity



## Connectors as Facilitators of Distribution and Heterogeneity (2)



## Interaction Mismatches

- A direct by-product of reuse and heterogeneity
- Components must be undisturbed
- *The responsibility is connectors'*
- Interaction mismatch handling techniques
  - pairwise information reformatting/conversion
  - interchange to and from a single shared format
  - data conversion modules
    - buffers
    - wrappers
    - adaptors

## Existing Interconnection Technologies

- Field
    - developed at Brown University
    - precursor to HP's SoftBench
  - Q
    - developed at University of Colorado
    - provided interoperability for the Arcadia environment
  - Polyolith
    - developed at University of Maryland
- *Examples of successful transitions of research ideas into practice*

## The Field Environment

- Built on workstations
  - intended for large-scale programming with “classical” PLs
- Uses selective broadcasting to integrate software tools
  - central message server with distributed message handling
- Goals
  - teaching
  - research
  - development
- Achieved by
  - ease of use
  - ability to handle moderate-sized systems
  - flexibility
  - extensibility

## The Field Toolset

- Annotation editor
  - associates annotations with source code
  - for C and Pascal
- Cross referencer
  - collects relations about the current system and stores them in an updatable, relational database
    - e.g., functions, calls, declarations...
  - cross reference interface contains a front-end to the cross-referencer
- Data structure display
  - visualization of complex data structures

## The Field Toolset (2)

- Debugger
  - a message-based interface to other Field tools
  - GUI front end
- Flow-graph viewer
  - visualization of the hierarchical flow graph, obtained from the cross-referencer
  - locates routines and calls
  - highlights execution
- Menu-driven interface to Unix make
  - performs compilations
  - informs of errors
  - possible to extend Unix make
    - e.g., automatic dependency analyzer

## The Field Toolset (3)

- Profiler interface
  - visualization of a system's execution profile
  - used, e.g., for detecting execution bottlenecks
- Viewer
  - generic viewing facility
  - used for different system aspects
    - runtime stack
    - traced variables and expressions
    - debugger events (e.g., breakpoints)

## Field's Integration Framework

- Uses a central message bus **Msg** for tool integration
  - selective broadcast (synchronous and asynchronous)
  - tools register message patterns of interest
  - all message traffic goes via Msg to registered tools
- Field messages
  - text strings of arbitrary length
  - information extracted according to patterns
  - communication either synchronous or asynchronous
- Four tool integration criteria
  - tools must be able to interact directly
  - tools must share dynamic information
  - programmer must access source code via a common editor
  - Field must make static, specialized information available to all tools that need it

## Integration Criterion #1 — Direct Tool Interaction

- Messages are used as command interfaces to tools
- Allows integration of new tools into a system
  - *New ==> Existing*
    - requires message pattern registration only
  - *Existing ==> New*
    - requires no modifications
    - tools listen to events, not sources of those events

## Integration Criterion #2 — Dynamic Info Sharing

- Tools must know the current execution context
  - e.g., highlighting current line of execution in the editor
- Tools must know about other tools' states
  - e.g., editor must know where the debugger has set a breakpoint
- Dynamic information sharing is handled by Msg
  - each tool defines events that may be of interest
  - Msg notifies registered tools of event occurrences

### Integration Criterion #3 — Common Editor

- Different developer needs
  - program editing
  - compilation error correction
  - execution progress tracking
  - viewing bottlenecks identified by the profiler
  - debugging breakpoints
  - variable/expression tracing
- Field's annotation editor is closely tied to its message facility
  - can communicate with other tools
- Annotation set expandable via the start-up file from which annotations are read

### Integration Criterion #4 — Static Info Sharing

- Static information
  - system building rules
  - cross-reference information
  - profiling data
  - program information
    - e.g., variable types
- Static information is delivered via *active servers*
  - keep track of static information
  - receive requests from Msg
  - feed information back to tools
  - can be specialized for particular kinds of information

## Q

- Built to support distributed object infrastructure for the Arcadia project
  - there was no single “vision”
  - rather, the system was evolved to support emerging needs
- Q has tackled the prevailing problem of heterogeneity
  - multi-platform
  - multi-OS
  - multi-PL
- Predates or is co-developed with similar commercial approaches
  - initiated in 1988
- Surpasses support provided in commercial interoperability platforms
  - e.g., CORBA’s primary focus is C/C++

## The Architecture of Q 1.0

- Third-party RPC/XDR layer
  - remote procedure call (RPC)
  - external data representation (XDR)
  - inter-process typed data exchange
  - a generic data marshalling mechanism
  - inter-process communication (IPC)
  - supports only C
- Language-specific interfaces
  - Ada
  - C/C++
  - Lisp
  - Prolog

Ada	C	C++
RPC		XDR
Msg Transport		

## Critical Issue

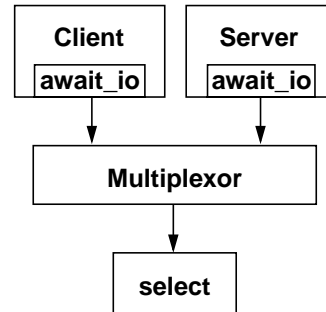
- XDR assumes a single type model
  - focus on heterogeneous architectures, not languages
- Concept of type concordance
  - representation of a type in one language
  - representation of the type in another language
  - marshaled representation of the type
- Supported types
  - integer
  - floating point
  - fixed point
  - enumeration
  - boolean
  - character
  - string
  - pointer

## Problem Encountered with Q 1.0

- Threading
  - RPC consists of two message sends (request—response)
  - all threads wait for messages on the same channels
    - uses Unix *select*
  - behavior of Q is unpredictable in the case of concurrency
    - e.g., Ada tasks

## The Architecture of Q 2.0

- Incorporates support for multiple clients/servers
- Exploits Ada-level *select*
- The multiplexor monitors message arrival and informs components *awaiting io*
- Problem encountered
  - Ada interface was reengineered to support multi-threaded architectures
  - RPC was not
    - uses global data structures
    - problem in multi-threaded development



## The Architecture of Q 3.0

- Non-blocking message passing interface on top of RPC
- Calls into it protected against reentrant access by semaphores
- Support for non-reentrant access partly implemented in language interfaces
  - eliminates the problem of language runtime system not recognizing blocking events and continuing to schedule threads

Ada	C	C++
Msg Int		Marsh
RPC		DR
Msg Transport		

## Lessons Learned from Q

- Real applications are not in a single style
  - drivers are reuse, heterogeneity, distribution
- RPC may not be sufficient to support them
  - client-server vs. peer-peer
- Large components are typically multi-threaded
- Interoperability mechanisms must have threading models compatible with components'
  - avoid deadlocks
  - provide efficient IPC service