

# An Architectural Approach Maximizing Platform Portability and User Interface Component Reuse

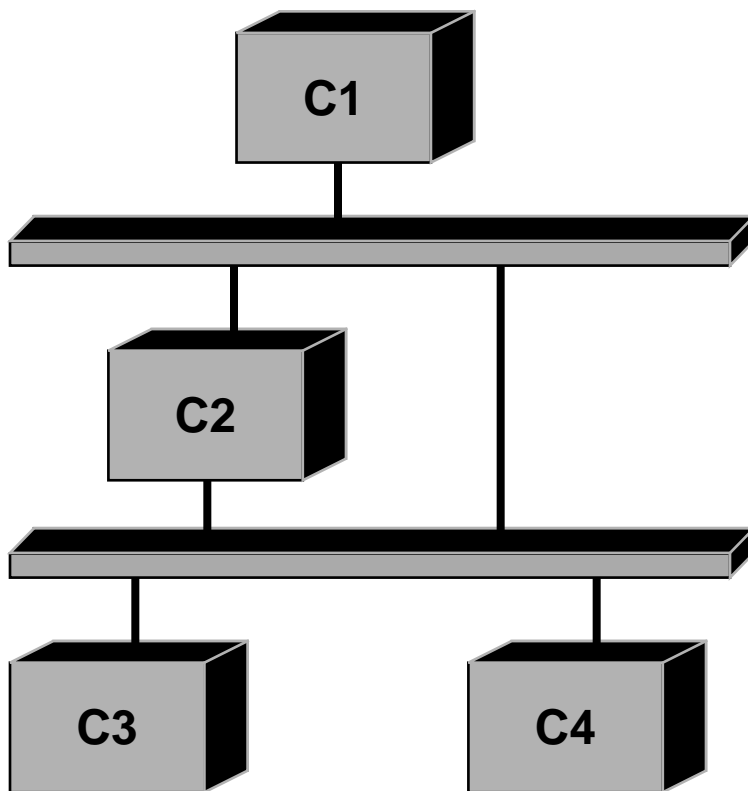
Richard N. Taylor

Department of Information and Computer Science  
University of California, Irvine  
Irvine, California 92697-3425  
`taylor@ics.uci.edu`

`http://www.ics.uci.edu/~taylor/`  
`http://www.ics.uci.edu/pub/arch/`

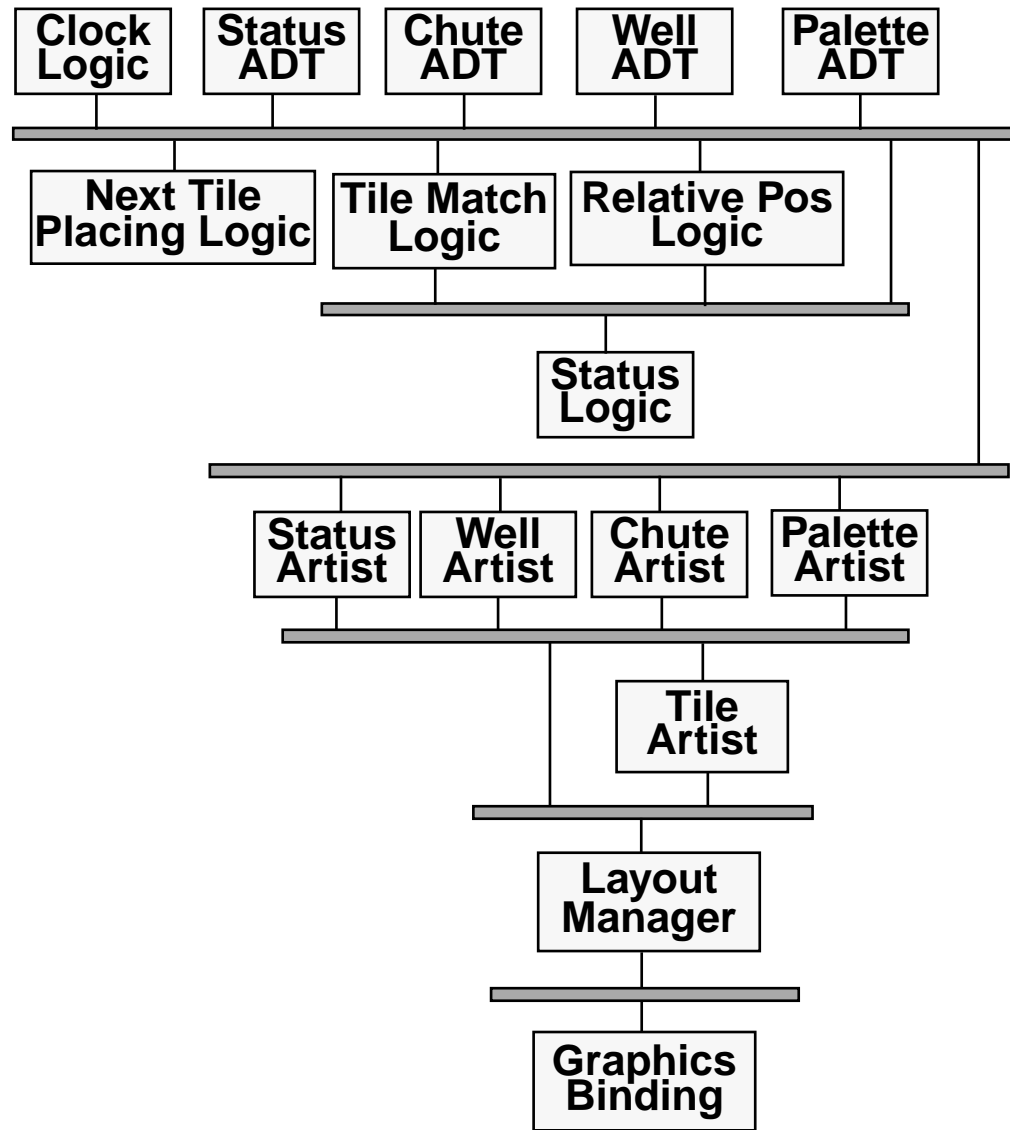
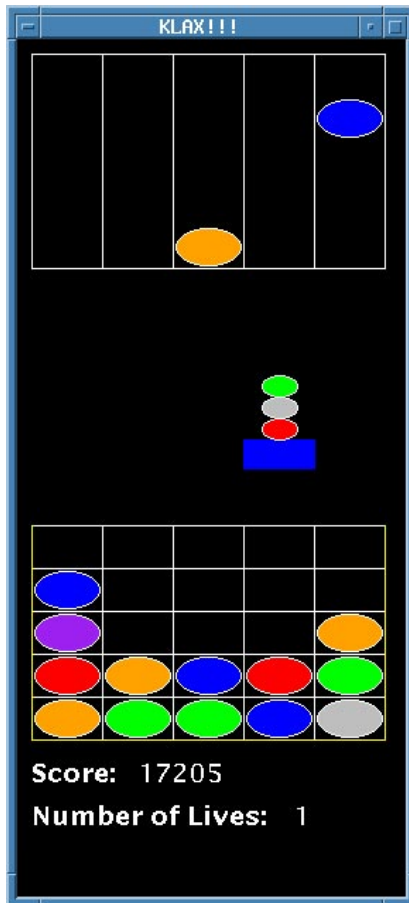
## Overview of C2

- A component- and message-based architectural style
- C2 architectures are networks of concurrent components hooked together by connectors

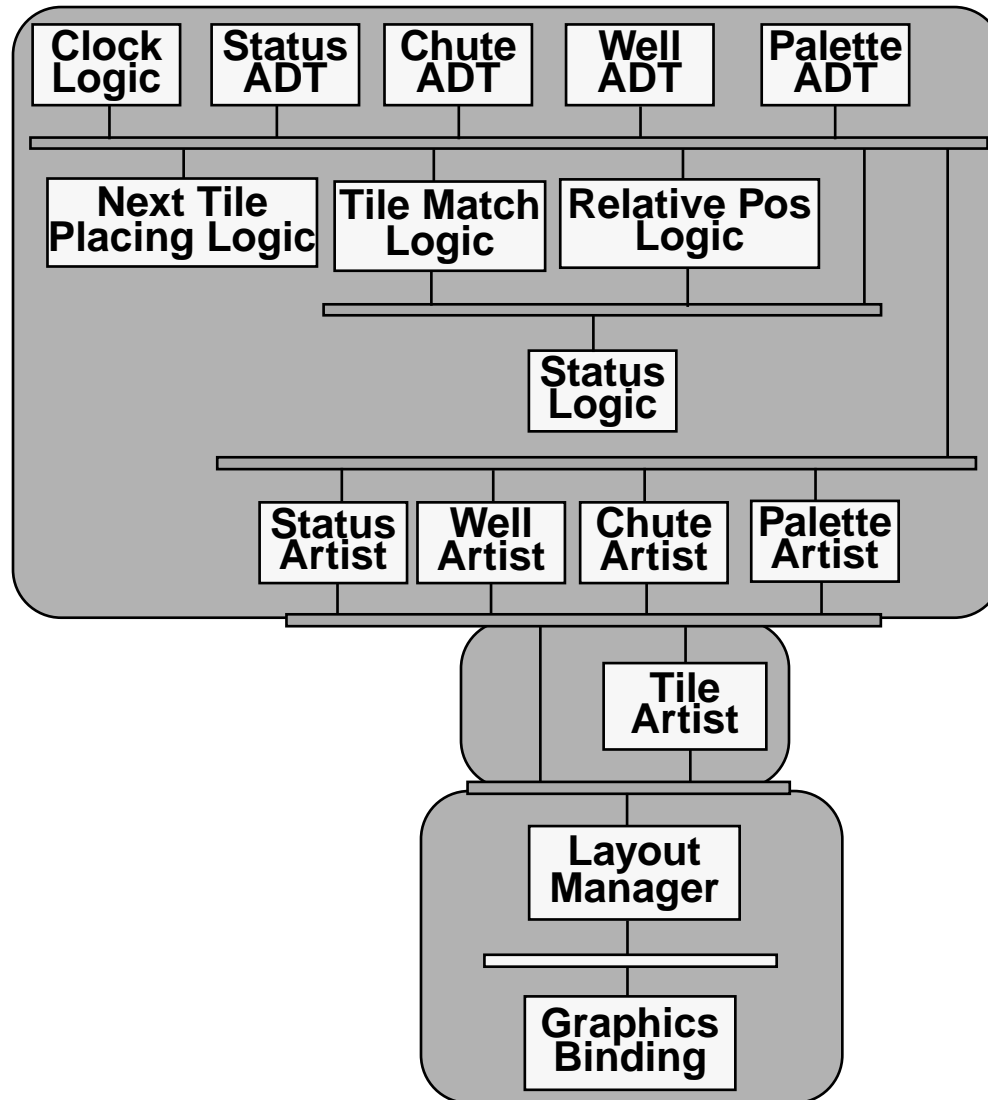


- no component-to-component links
- “one up, one down” rule for components
- connector-to-connector links are allowed
- “many up, many down” rule for connectors
- all communication by exchanging messages
- *substrate independence*

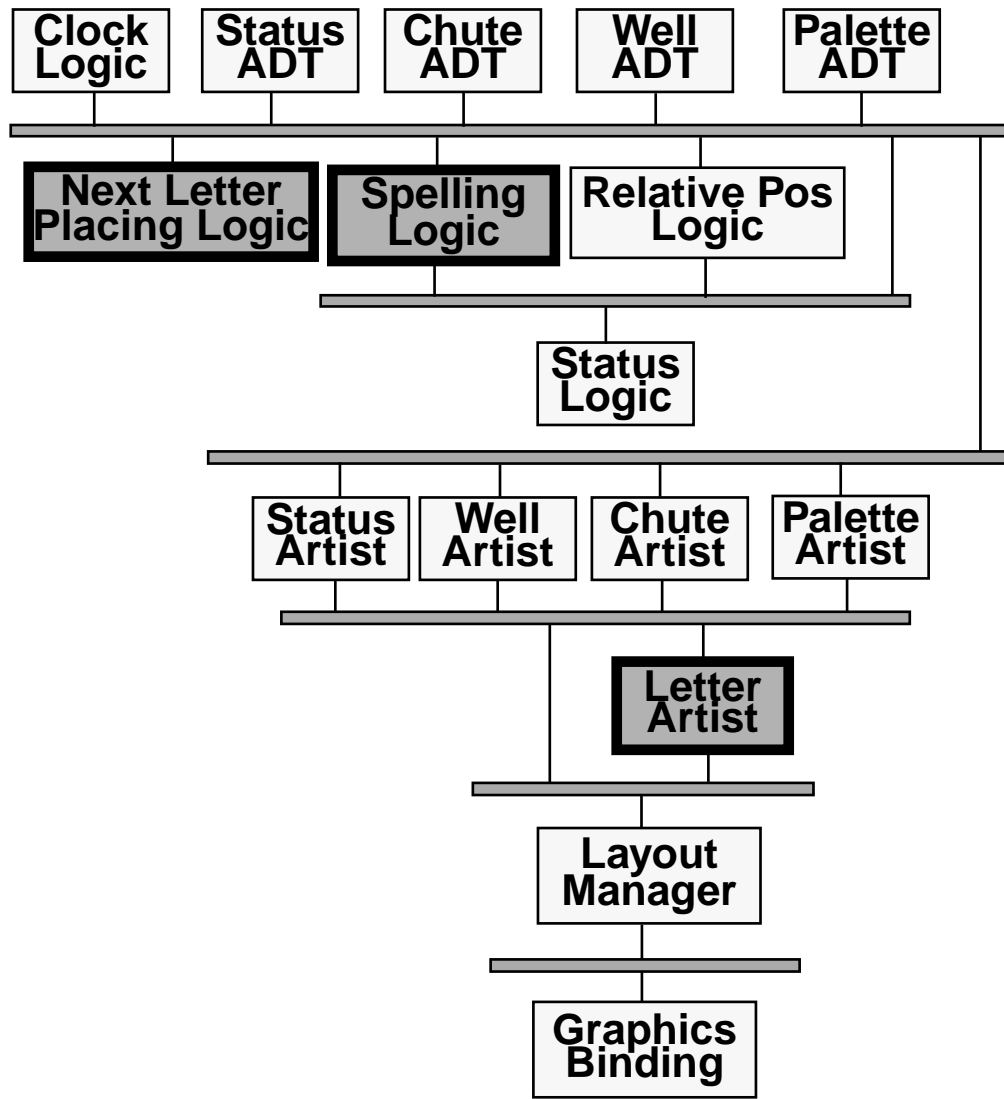
# KLAX: Example Application in the C2 Style



# An Implementation Architecture for KLAX

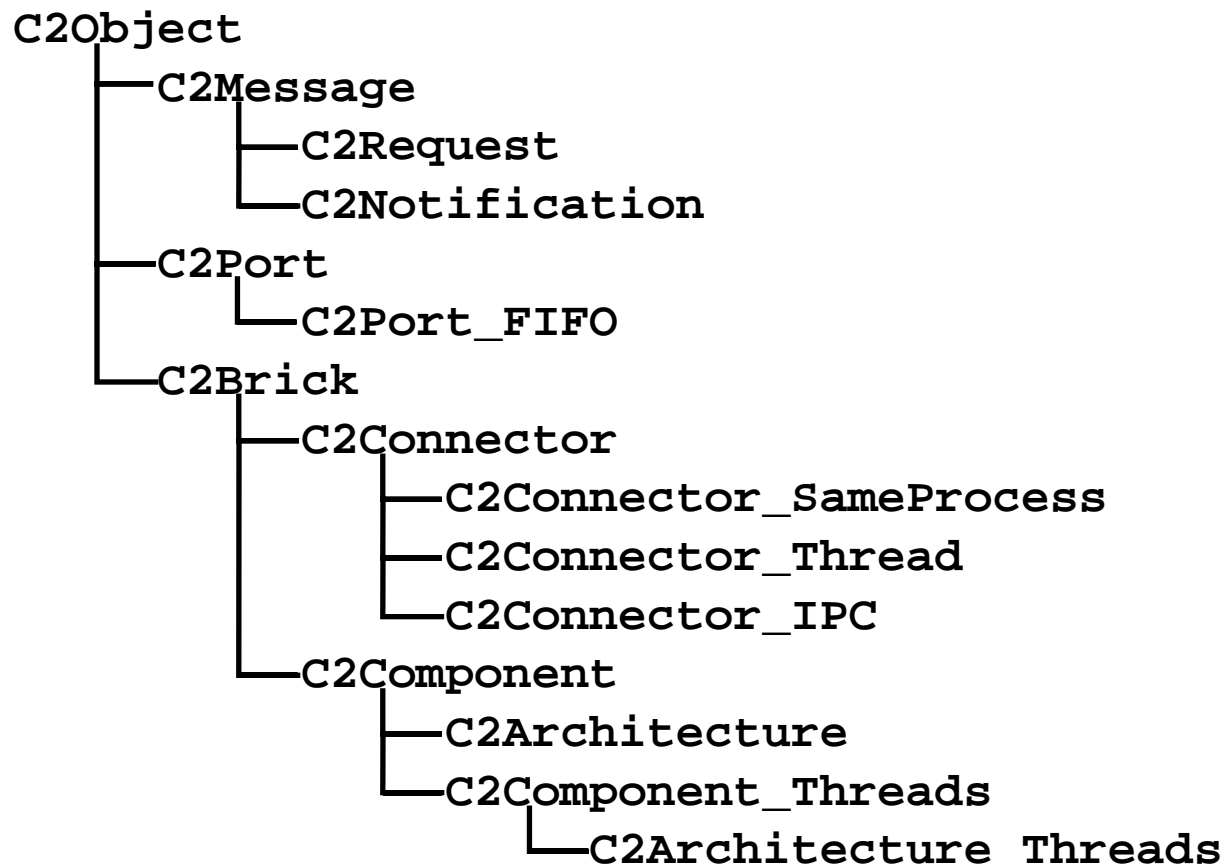


# Spelling KLAX



## Implementing C2 Architectures

- Extensible framework of abstract classes for C2 concepts
- Implemented in Java, C++, and Ada (partially)

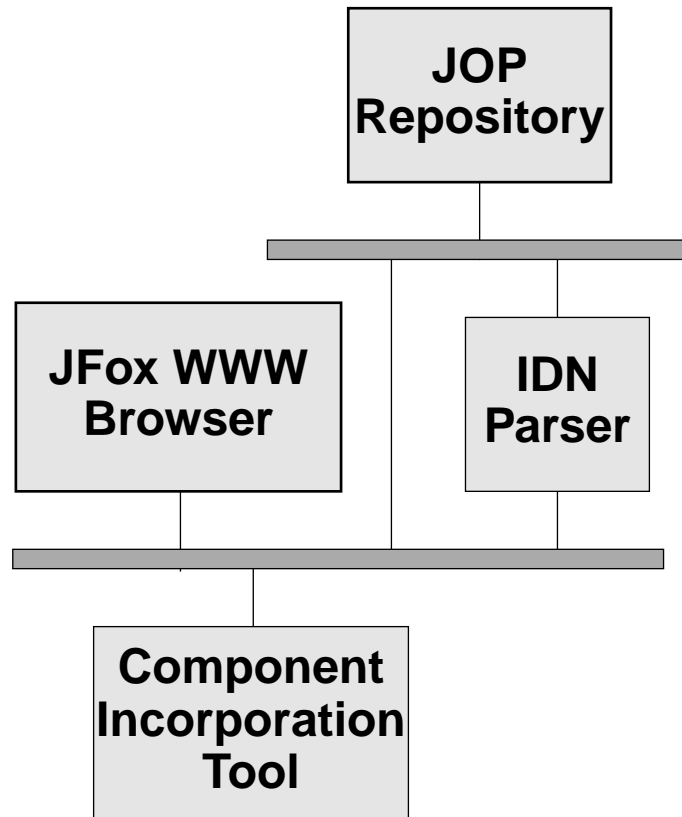


## Runtime Manipulation of C2 Architectures

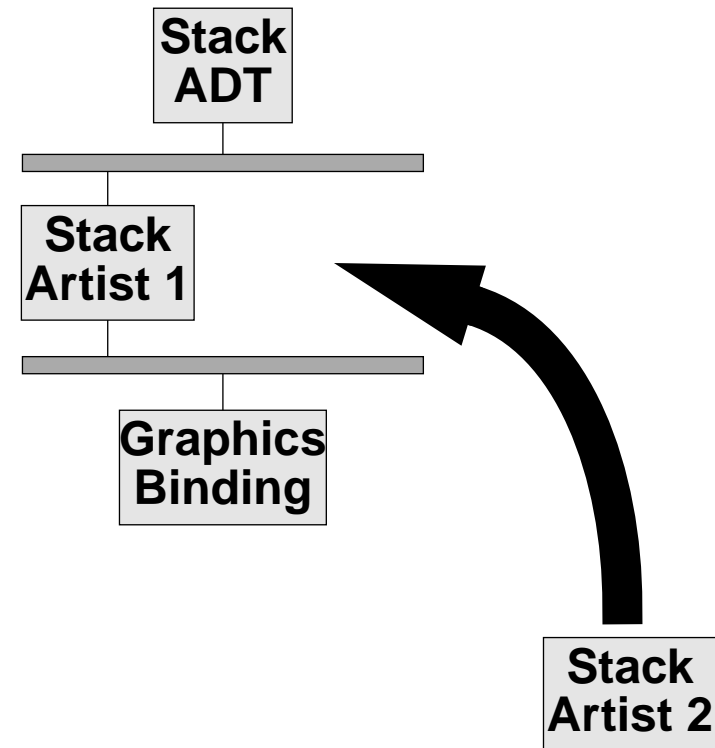
- *ArchShell* - tool for interactive construction, execution, and runtime modification of C2 architectures
  - current implementation supports the Java class framework
- Provides functionality similar to a UNIX shell
  - e.g., construction of pipe-and-filter architectures in *csH*
- Goes beyond support found in UNIX shells
  - dynamic loading and linking new architectural elements
  - sending C2 messages to running components interactively
  - testing runtime behavior of individual components
  - experimentation with partial architectures

## Example of Runtime Manipulation of C2 Architectures

### Environment Architecture



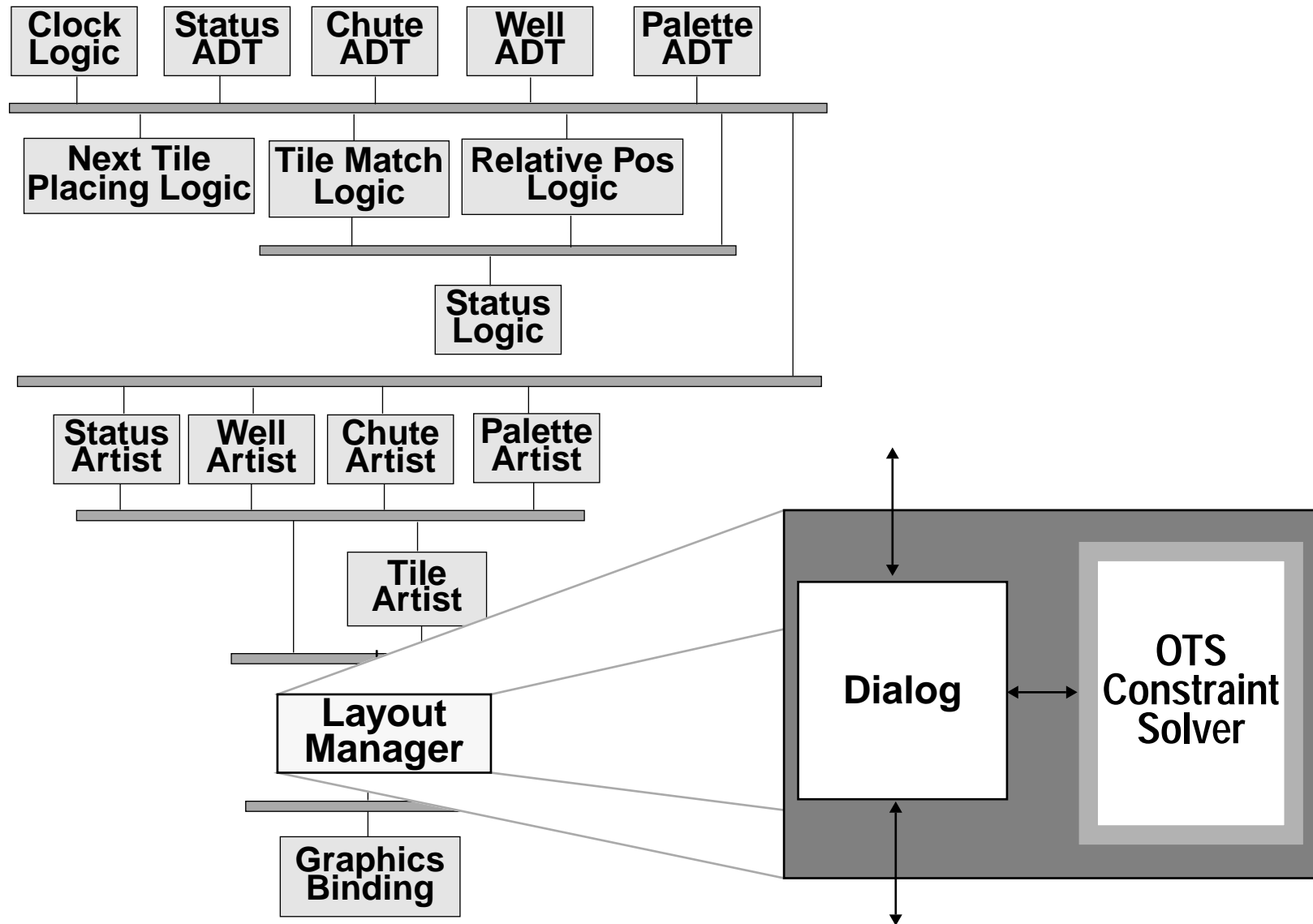
### Application Architecture



## Support for Reuse in C2

- Component heterogeneity
- Substrate independence
- Internal component architecture
- Asynchronous message passing via connectors
- No assumption of shared address space
- No assumption of single thread of control
- Separation of architecture from implementation

## Example of OTS Reuse in C2



## A Library of C2 Components

	Version Number	Constraints Maintained	Constraint Managers
<b>Palette ADT</b>	1	Palette Boundary	In-Line Code
	2	None	None
	3	Palette Boundary	SkyBlue
	4	Palette Boundary	Amulet
<b>Palette Artist</b>	1	Palette Location Tile Location Palette Size	In-Line Code
	2	None	None
	3	Palette Location Tile Location	SkyBlue
	4	Palette Location Tile Location	Amulet
<b>Chute Artist</b>	1	Chute Size	In-Line Code
	2	None	None
<b>Well Artist</b>	1	Well Size	In-Line Code
	2	None	None
<b>Layout Manager</b>	1	None	None
	2	All	SkyBlue
	3	All	Amulet
	4	Resizing	SkyBlue
	5	Resizing	Amulet
	6	All	SkyBlue & Amulet

## Conclusions

- Runtime manipulation of architectures and OTS component reuse facilitated by style features
  - isolation of changes inside components
  - component substitutability
  - partial communication and service utilization
- Component library and application family created easily
- Heuristics for OTS component integration

<b>Problem with OTS Component</b>	<b>Integration Method</b>
Inadequate Functionality	Source Code Modification
No Message-Based Communication	Wrapper
Different Programming Language	IPC Connector
Different OS Process	IPC Connector
Message Interface Mismatch	Domain Translator