Review — Architectural Style

- Definitions
  - Architectural styles are recurring organizational patterns and idioms.
  - Established, shared understanding of common design forms is a mark of a mature engineering field.
  - Architectural style is an abstraction of recurring composition and interaction characteristics of a set of architectures.
  - Styles are key design idioms that enable exploitation of suitable structural and evolution patterns and facilitate component, connector, and process reuse.
- Two categories of styles
  - idioms & patterns
  - reference models

Review — Style Properties and Benefits

- Properties of styles
  - a *vocabulary* of design elements
  - a set of *configuration rules*
  - a *semantic interpretation*
  - *analyses* possible in a style
- Benefits of styles
  - design and code reuse
  - understandability of system organization
  - interoperability
  - style-specific analyses
  - visualizations

Review — General Observations

- Different styles result in architectures with greatly differing properties
- A style does not fully influence the resulting architecture
    - considerable room for individual judgement
- Open issue:
    - → what is the relationship between domains and styles?

Review — Some Common Architectural Styles

- “Basic” styles
    - pipe and filter
    - object-oriented
    - implicit invocation
    - layered systems
    - blackboard
    - client-server
    - state transition
- “Derived” styles
    - → GenVoca
    - → C2

GenVoca

- A domain-independent model (*a style*) of hierarchical software composition based on
  - interchangeable software components
  - large-scale reuse
- “Lego” paradigm of software design and construction
- Extrapolated from the characteristics of systems built in two domains
  - Genesis — database management systems (DBMS)
  - Avoca — network software suites
- The two domains are well understood
  - automated support for component-based development

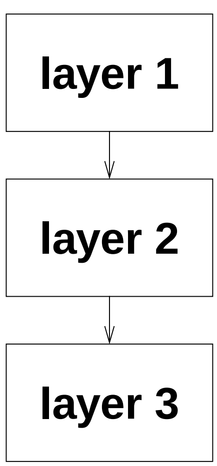
GenVoca Model Framework

- A *component* is a closely-knit cluster of classes
  - classes act as a unit
  - components may be parameterized
- A *realm* is a set of components that realize the same interface in different ways
  - different behaviors
  - different implementations
  - components in a realm are plug-compatible
  - → e.g., **`R = {c1,c2,c3}`**
- An architecture (a system) is a type expression
  - a composition of components
  - component interconnections are implicit in parameters
  - hierarchical composition is possible
  - → e.g., **`c[x:R1,y:{R2}]`**

GenVoca Model Framework (cont.)

- Principle of *design encapsulation*
  - components do not rely on implementations of components in their parameter lists (i.e., “below them”)
  - reminiscent of virtual machines
- A component is *symmetric* iff one of its parameters is in the component’s realm
  - symmetric components may be composed in any order
  - composition semantics may differ substantially
- A *domain* is a set of all systems that present the interface of a realm

Example — GenVoca Realms and Systems




```
LayerStack = layer1 [ layer2 [ layer3 ] ]

RealmBottom = { layer3 }

RealmMiddle = { layer2 [ x:RealmBottom ] }

RealmTop = { layer1 [ y:RealmMiddle ] }
```

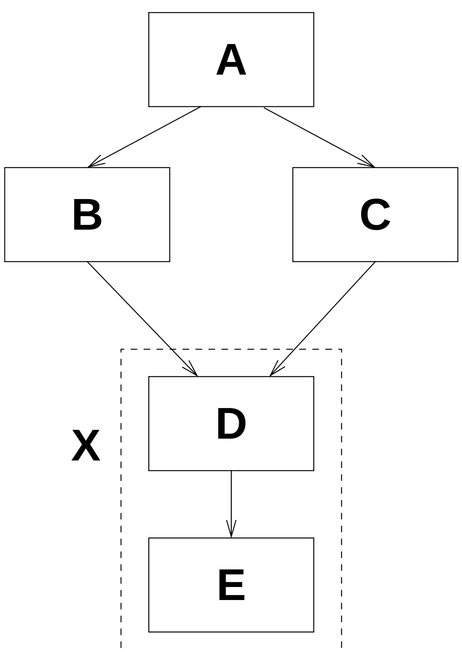
Example — GenVoca Symmetric Components

- Arbitrarily composable

`a[x:Q]`
`b[y:Q]`
} →
`a[b[y:Q]]`
`b[a[x:Q]]`
`a[b[a[x:Q]]]`
`a[a[a[a[x:Q]]]]`

- UNIX filters are symmetric components
  - all filters have the same interface

Example — GenVoca Hierarchical Systems



`S = A[B[X],C[X]]`

`X = D[E]`

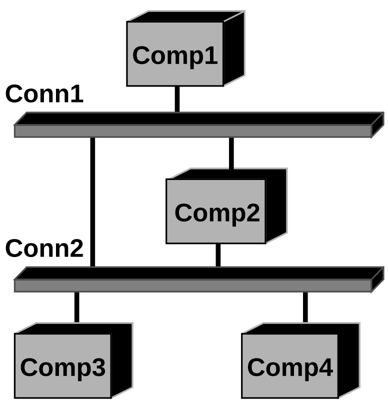
`S = A[B[D[E]],C[D[E]]]`

GenVoca Component Interactions

- Components communicate by direct calls
  - no support for implicit invocation or event multicast
  - a single address space is assumed
- No support for concurrency
- No support for heterogeneous interactions
  - components may need to be custom (re)built to fit the GenVoca model

C2

- A component- and message-based style
  - for highly distributed software systems
- Generalized from GUI intensive systems' architectures
- C2 architectures are networks of concurrent components hooked together by connectors



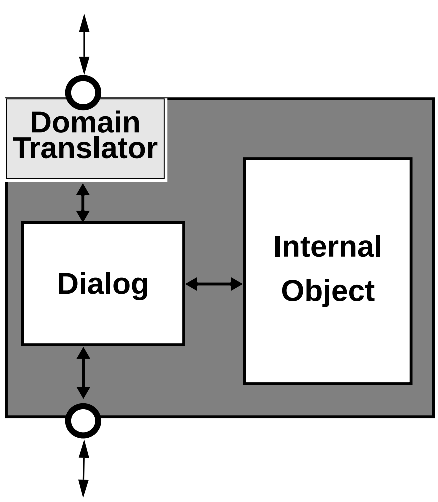
  - no component-to-component links
  - "one up, one down" rule for components
  - connector-to-connector links are allowed
  - "many up, many down" rule for connectors
  - all communication by exchanging messages
  - *substrate independence*

Goals of C2

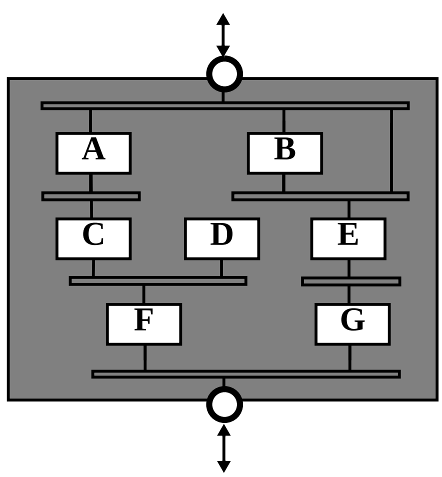
- Reuse
  - components and connectors
- Heterogeneity
  - distributed environment
  - multi-lingual components
  - multiple component granularities
  - multiple address spaces
  - multiple threads and/or processes
  - multiple users
  - multiple toolkits and media types
- Evolvability
  - static and dynamic

Internal Architecture of a C2 Component

Simple

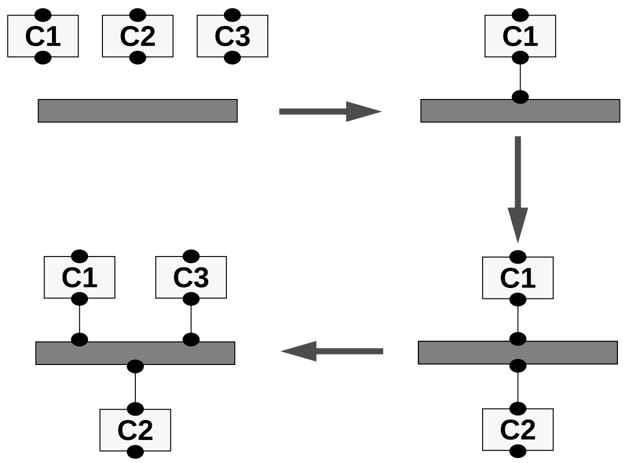


Composite

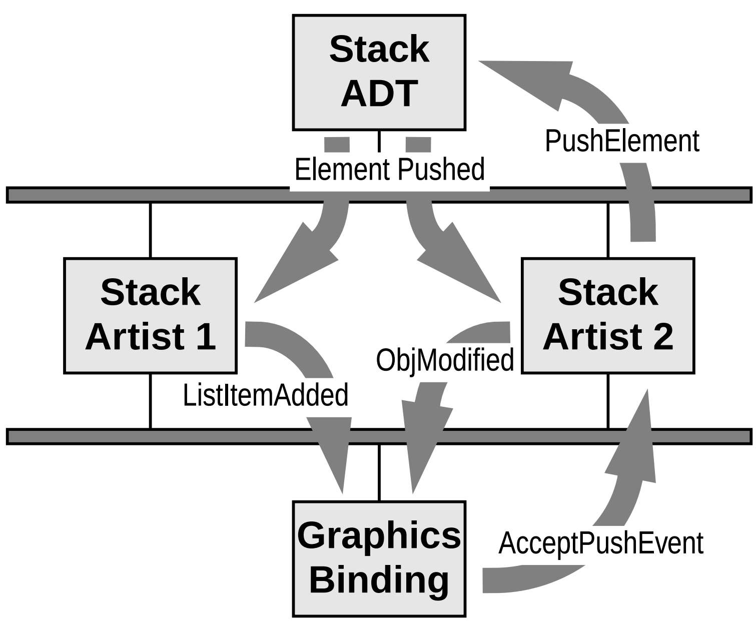


C2 Connectors

- Communication message routing and filtering devices
- C2 connector interfaces are context-dependent
  - a function of the interfaces of attached components
  - a function of the interfaces of attached connectors

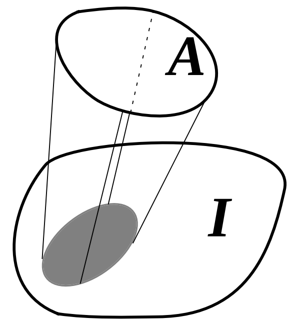


Simple C2 Architecture



Implementing C2 Architectures

- Extensible framework of abstract classes for C2 concepts
  - components
  - connectors
  - communication ports
  - messages
- Implements interconnection and communication protocols
- Enables rapid construction of C2-style applications
  - allows developers to focus on application-level issues
  - facilitates automated implementation generation
- Implemented in Java, C++, and Ada
  - extended to support multi-lingual development

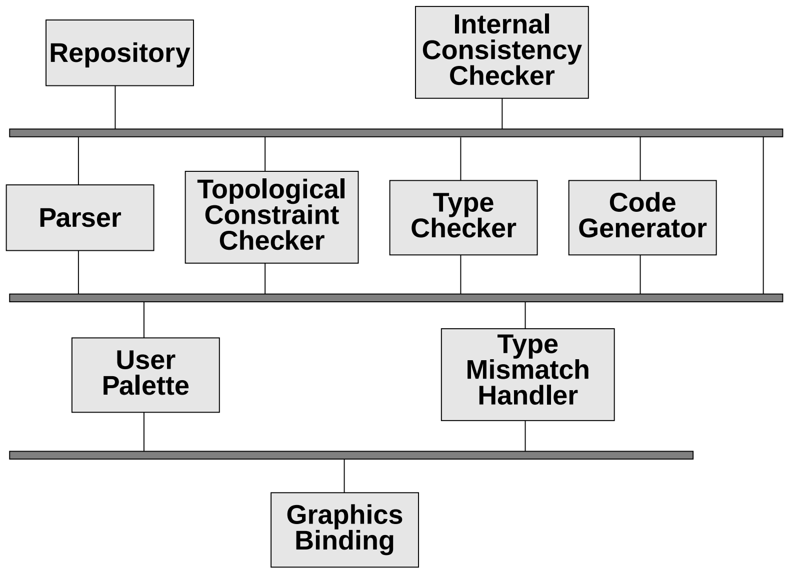


C2 Implementation Framework

- C2Object
  - C2Message
    - C2Request
    - C2Notification
  - C2Port
    - C2Port_FIFO
  - C2Brick
    - C2Connector
      - C2Connector_SameProcess
      - C2Connector_Thread
      - C2Connector_IPC
    - C2Component
      - C2Architecture
      - C2Component_Threads
        - C2Architecture_Threads

Example C2-Style Application — DRADEL

- An architecture-based development and evolution environment



Example C2-Style Application Family — KLAX