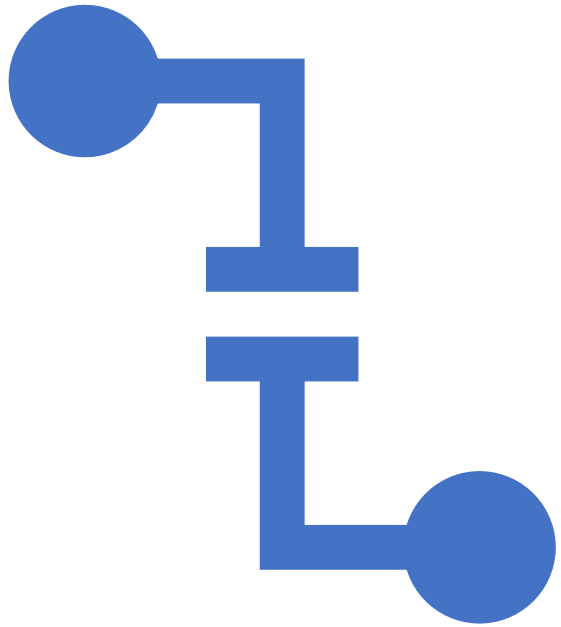
Software Architecture and Design Thinking 116U01C701

Module 2

Module 2: Connectors

2.1 Connector Foundations, Connector Roles

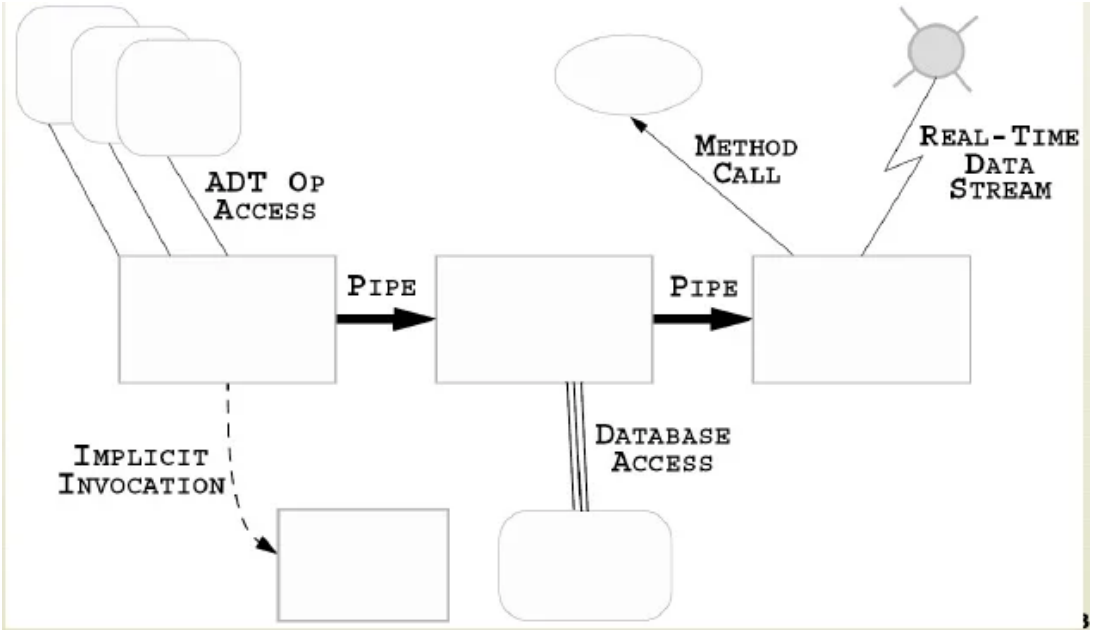
2.2 Connector Types and Their Variation Dimensions



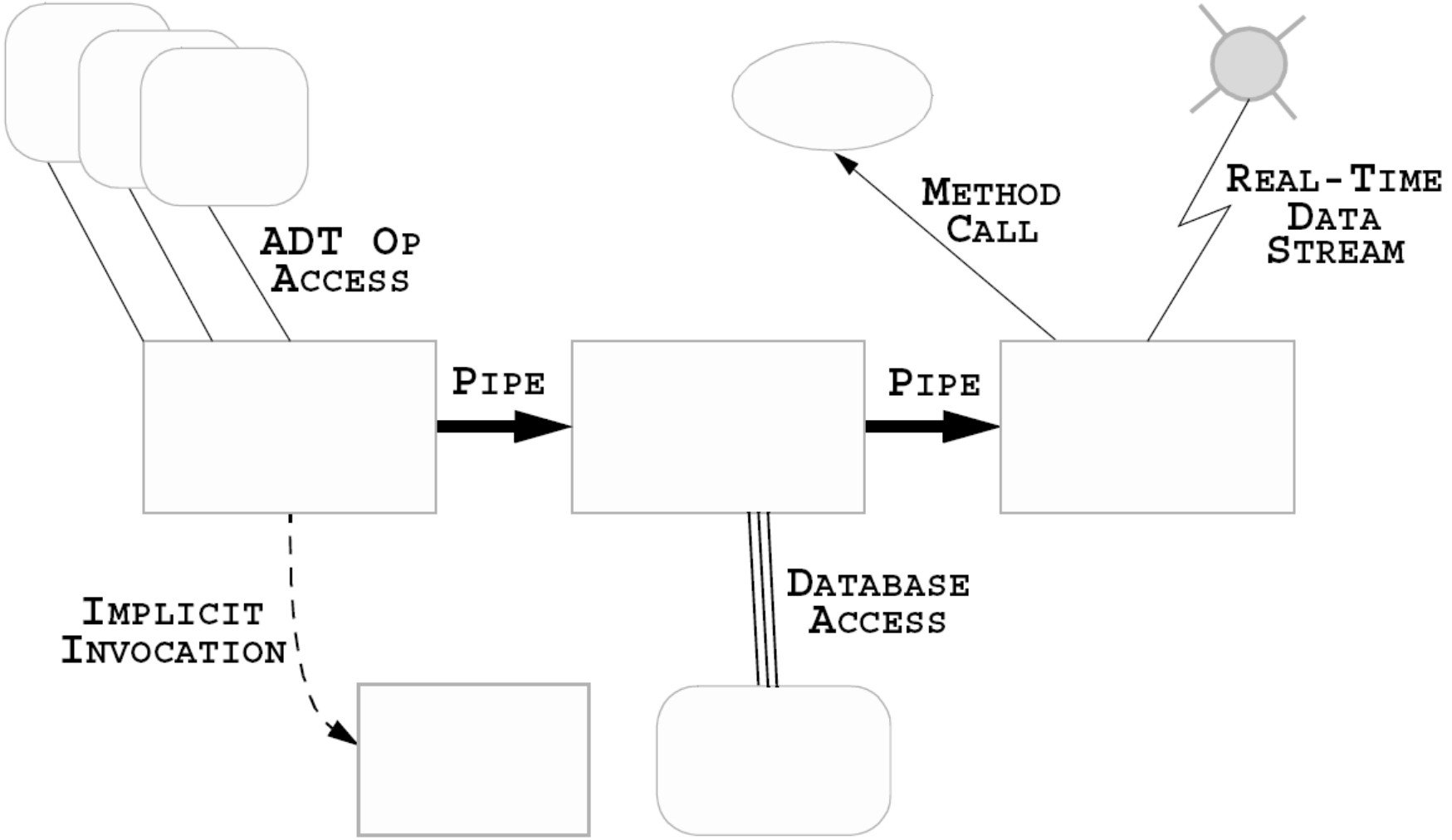
Connectors

What is a Software Connector?

- **Architectural element that models**
  - Interactions among components
  - Rules that govern those interactions
- **Simple interactions**
  - Procedure calls
  - Shared variable access
- **Complex & semantically rich interactions**
  - Client-server protocols
  - Database access protocols
  - Asynchronous event multicast
- **Each connector provides**
  - Interaction duct(s)
  - Transfer of control and/or data



Where are Connectors in Software Systems?



Implemented vs. Conceptual Connectors

- Connectors in software system implementations
  - Frequently no dedicated code
  - Frequently no identity
  - Typically do not correspond to compilation units
  - Distributed implementation
    - Across multiple modules
    - Across interaction mechanisms

Implemented vs. Conceptual Connectors (cont'd)

- Connectors in software architectures
  - First-class entities
  - Have identity
  - Describe all system interaction
  - Entitled to their own specifications & abstractions

Reasons for Treating Connectors Independently

- Connector ≠ Component
  - Components provide application-specific functionality
  - Connectors provide application-independent interaction mechanisms
- Interaction abstraction and/or parameterization
- Specification of complex interactions
  - Binary vs. N-ary
  - Asymmetric vs. Symmetric
  - Interaction protocols

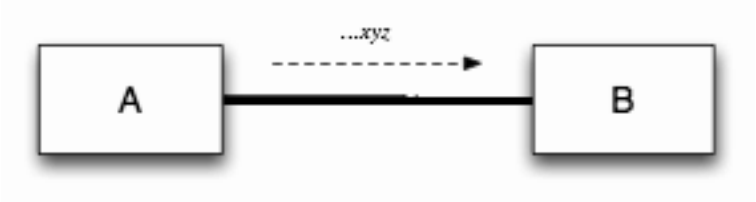
Treating Connectors Independently (cont'd)

- Localization of interaction definition
- Extra-component system (interaction) information
- Component independence
- Component interaction flexibility

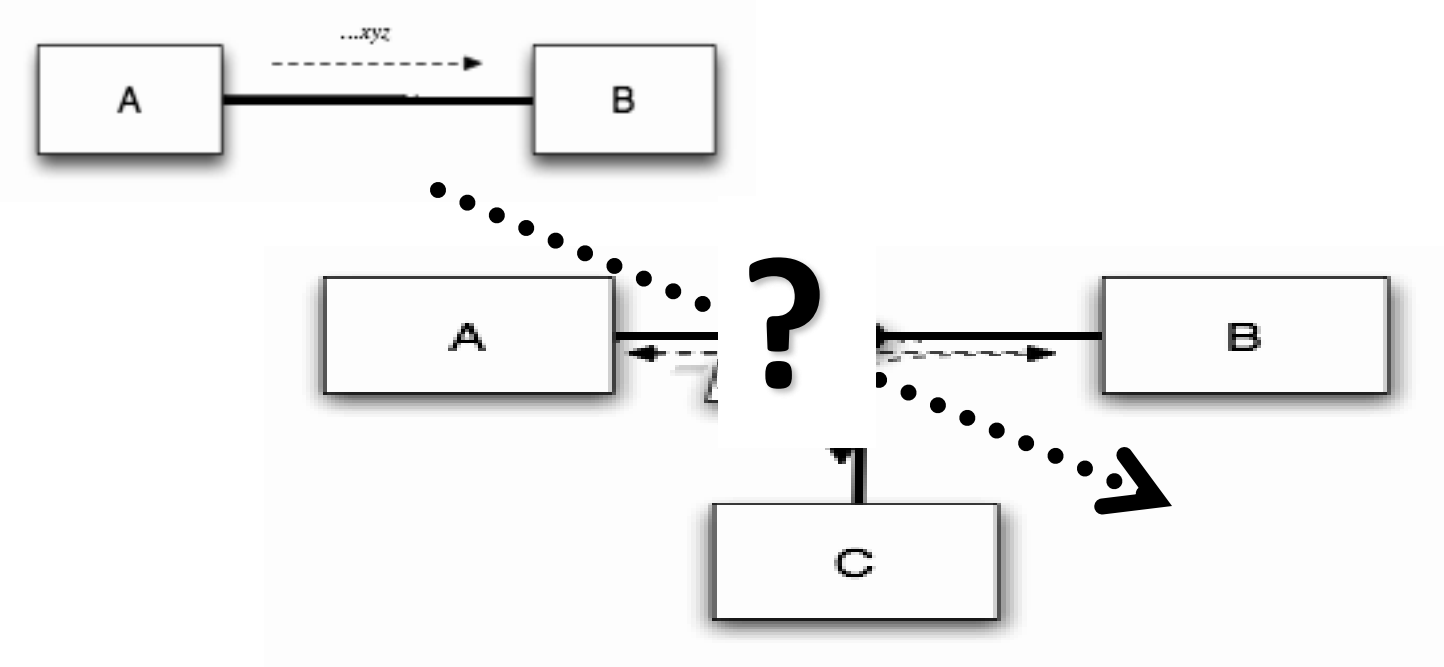
Benefits of First-Class Connectors

- Separate computation from interaction
- Minimize component interdependencies
- Support software evolution
  - At component-, connector-, & system-level
- Potential for supporting dynamism
- Facilitate heterogeneity
- Become points of distribution
- Aid system analysis & testing

An Example of Explicit Connectors



An Example of Explicit Connectors (cont'd)



Connector foundations

- Connectors mainly used for:
  - Flow of control: calling functions/ procedure or other programs
  - Flow of data: memory access
- Additionally:
  - Maintains one channel/ duct used to link interacting components
  - Support flow of data and control between them
- Simple connectors (module linker) provide services by forming duct between components
- Connector argument ducts with some combination of data and control flow will provide richer interaction services (connecting specific components based on the service needed)
- Very complex connectors can have internal architecture that includes computation and information storage.

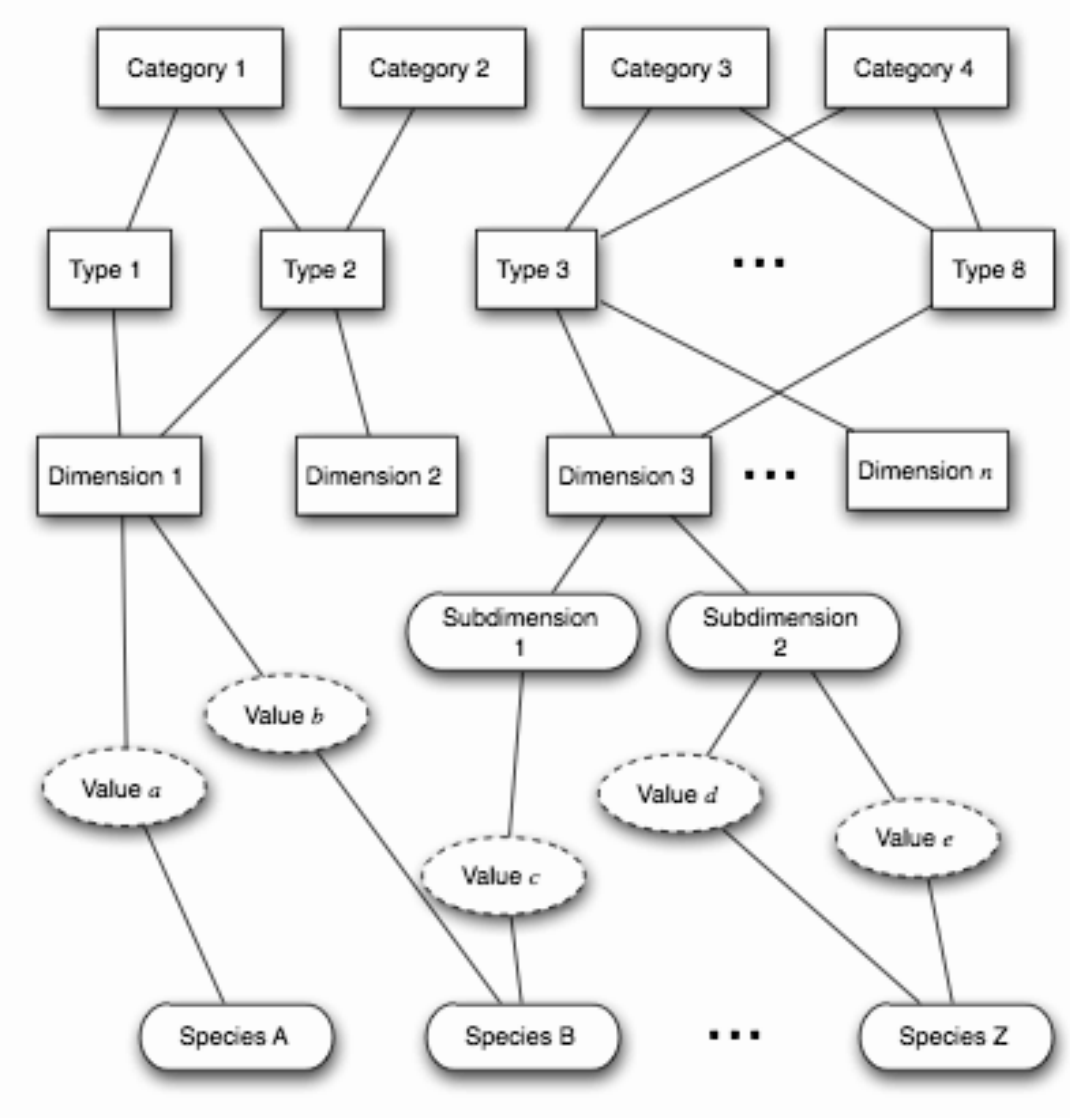
Connector foundations

- **Simple connectors (module linker)**
  - Provide services by forming duct between components
  - Provide one type of interaction service
  - Implemented in programming languages

Connector foundations

- **Composite Connectors:**
  - Connector argument ducts with some combination of data and control flow will provide richer interaction services (connecting specific components based on the service needed)
  - Very complex connectors can have internal architecture that includes computation and information storage.
  - Achieved through composition of several connectors (and possible components)
  - Provided as libraries and frameworks
  - Combine many kind of interactions
  - Can help to overcome the limitation so f modern programming language
  - Necessary to understand underlying, low-level interaction mechanism, identify appropriate design choice, direct potential mismatch among components

Framework for studying Connector



Category: Primary service / role provided

Type: way in which interaction services are realized

Dimension/ subdimension: architectural relevant details

Value: dimension instances

Species: values from different types

Software Connector Roles

- **Locus of interaction among set of components**
- **Protocol specification (sometimes implicit) that defines its properties**
  - Types of interfaces it is able to mediate
  - Assurances about interaction properties
  - Rules about interaction ordering
  - Interaction commitments (e.g., performance)
- **Roles**
  - Communication
  - Coordination
  - Conversion
  - Facilitation

Connectors for Communication

- **Main role associated with connectors**
- **Support:**
  - Different communication mechanisms
    - e.g. procedure call, Remote Procedure Calls, shared data access, message passing
  - Constraints on communication structure/direction
    - e.g. pipes
  - Used to pass messages, exchanges data to be processed and communicate results of computation

Connectors for Coordination

- Determine computation control mechanism
- Components interact by passing the thread of execution to each other
- Function calls and method invocation
- High order connectors such as signals and load balancing connectors provide richer, more complex interactions built around coordination services
- Control delivery of data

Connectors for Conversion

- Transform the interaction required by one component to that provided by another
- Enable heterogeneous components to interact
- Enable interaction of independently developed, mismatched components
- Mismatches based on interaction
  - Type
  - Number
  - Frequency
  - Order
- Examples of converters
  - Adaptors
  - Wrappers

Connectors for Facilitation

- Mediate and streamline components interaction
- Enable interaction of components intended to interoperate
- Govern access to shared information
- Ensure proper performance profiles
  - e.g., load balancing
- Provide synchronization mechanisms
  - Critical sections
  - Monitors

Connector Types/ levels

1. Procedure call
2. Data access
3. Event
4. Stream
5. Linkage
6. Distributor
7. Arbitrator
8. Adaptor

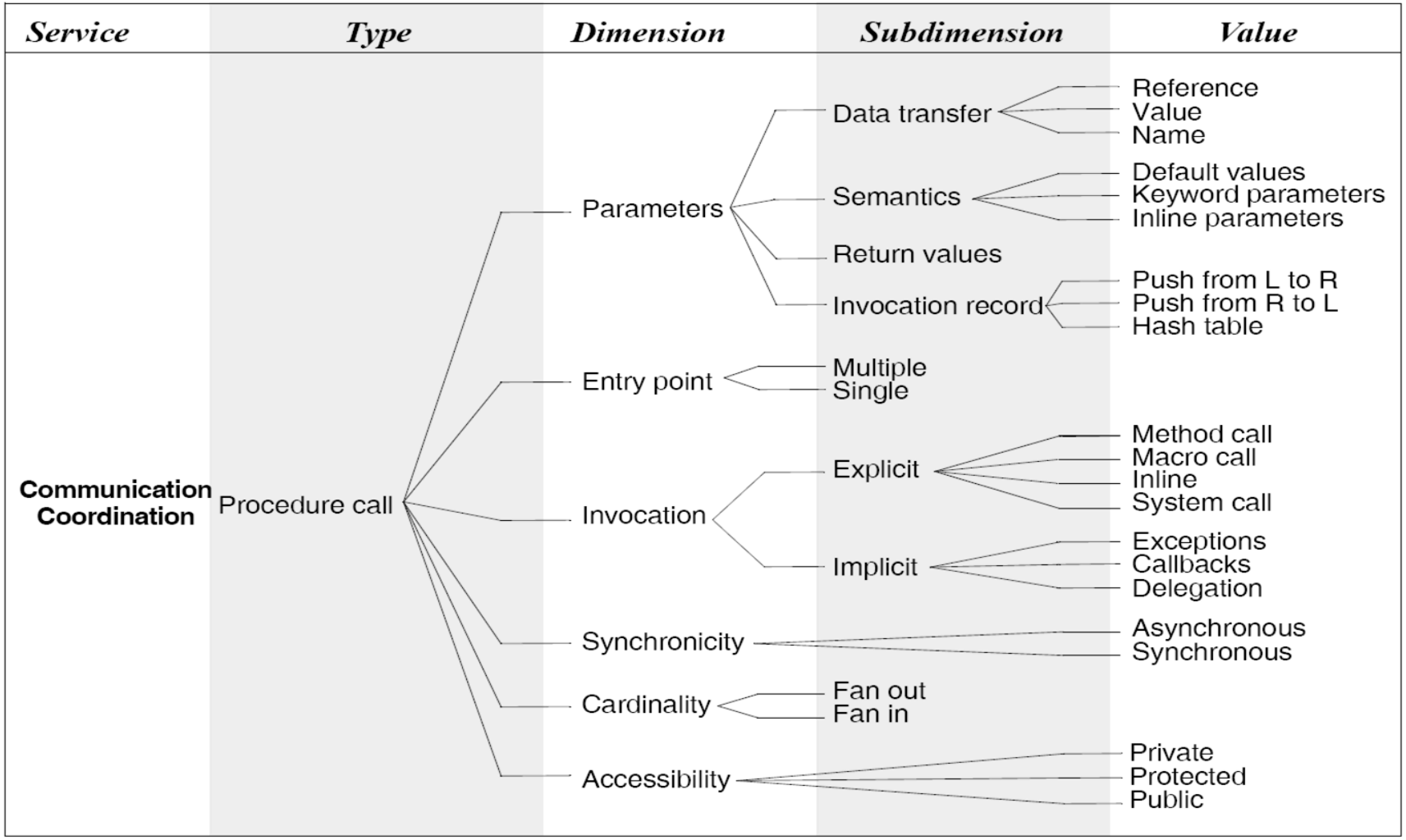
Procedure Call Connectors

- **Coordination connectors:**
  - Model the flow of control among the components through various invocation techniques
- **Communication connectors:**
  - Perform transfer of data through parameters and return values
- Most widely used and best understood connectors
- E.g. procedure call included in OO methods, fork and exec in Unix like environment , call-back invocation in event-based systems and operating system calls
- Frequently used as the basis of composite connectors such as remote procedure call (RPC)

Procedure Call Connectors

- Values of dimension
- Multiple entry versus Single entry point
- Fan- in (how many can be supported) and fan-out (how many can be called)

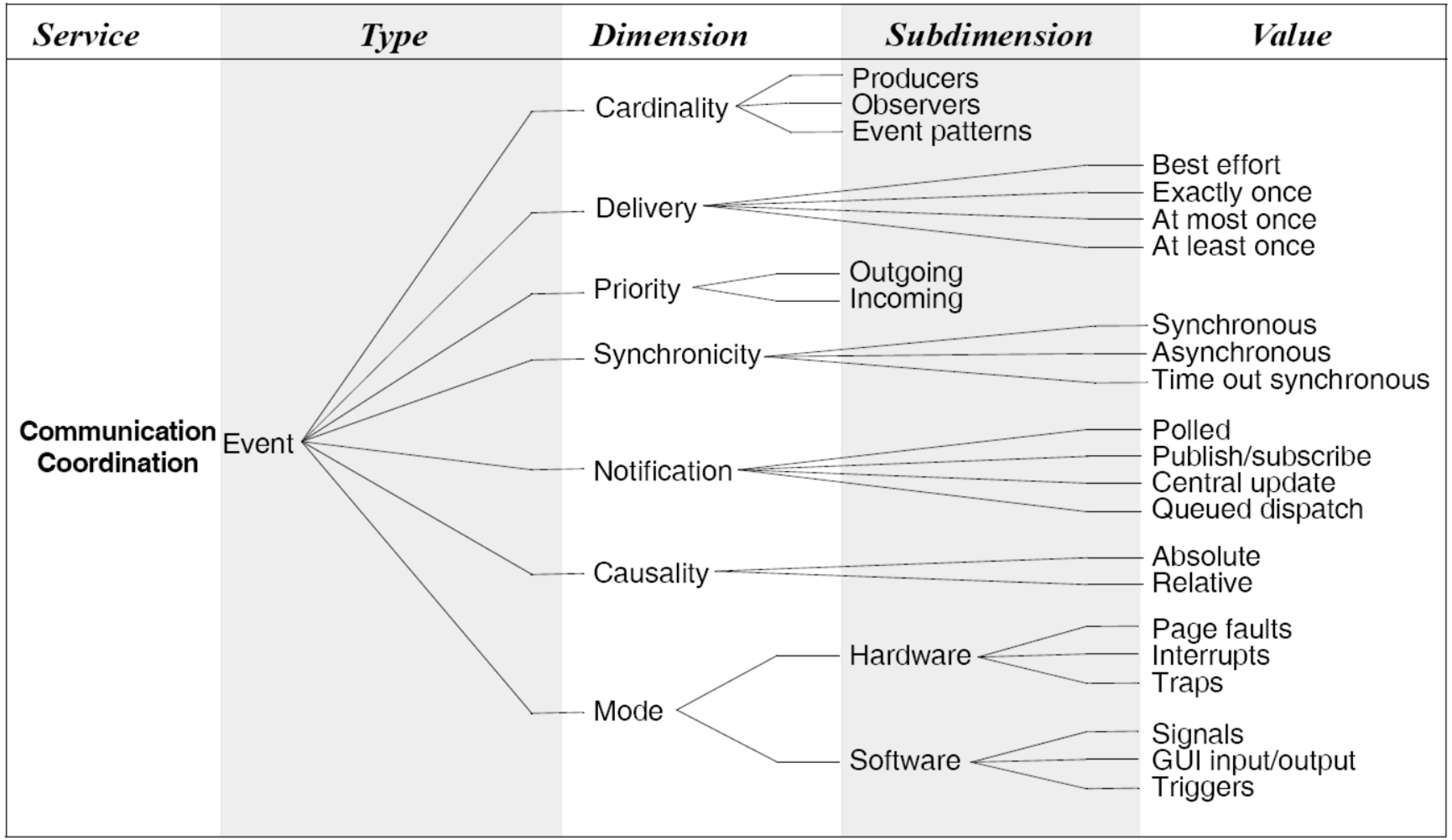
Procedure Call Connectors



Event Connectors

- An event is the instantaneous effect of the normal or abnormal termination of the invocation of an operation on an object, which occurs at that object's location.
- They are **similar to procedure call connectors** as they affect the flow of control among components providing coordination services.
- Flow is participated by an event.
- After recognizing the occurrence of an event, event notification is generated which concludes into component handling the event gaining the control.
- They form **virtual connection** between components interested in the same event topics.
- Typically found in distributed applications that requires asynchronous communication.

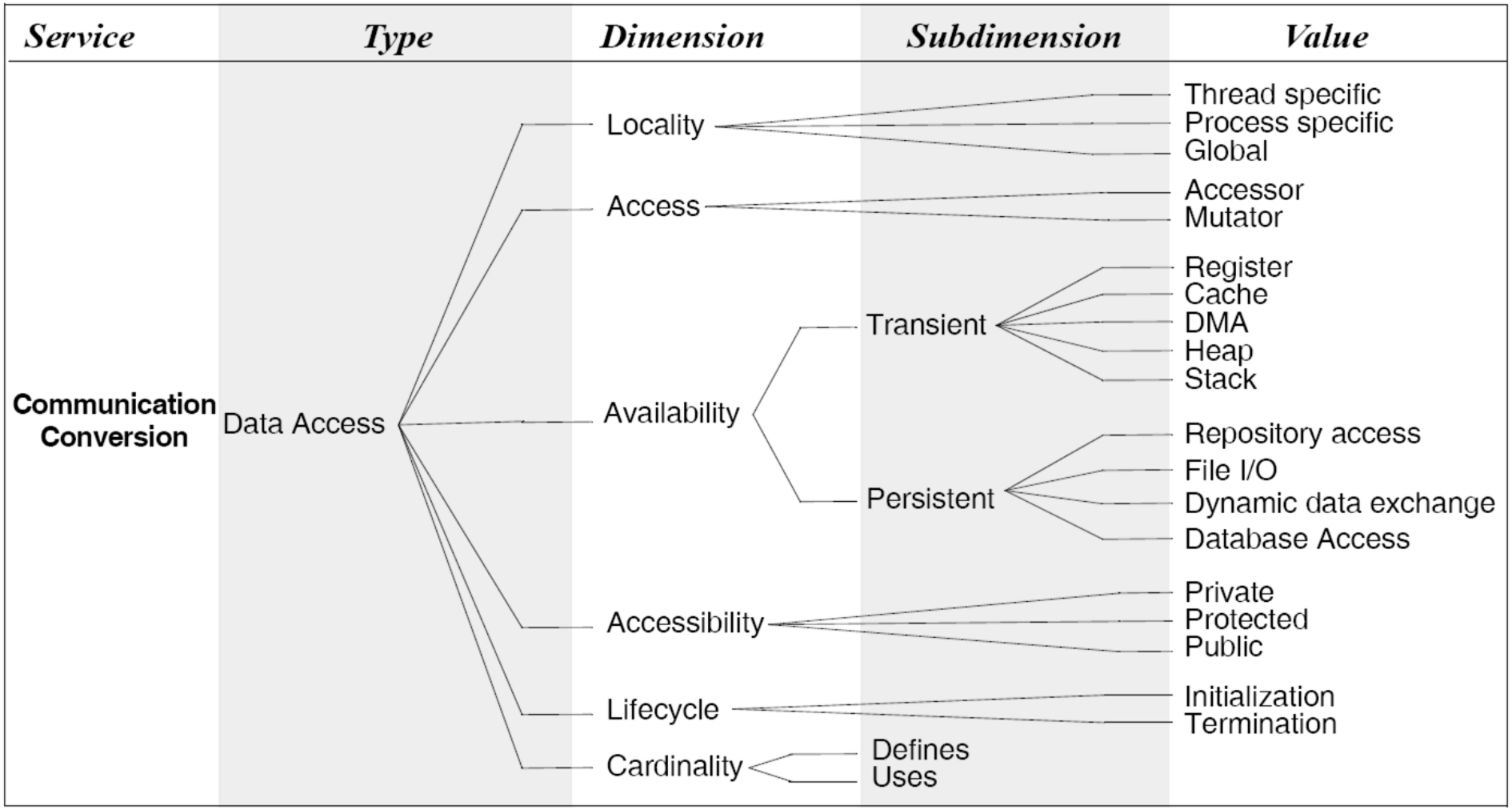
Event Connectors



Data Access Connectors

- Allow components to access data maintained by data store components as a communication service
- Needs preparation of data store before and cleanup after access
- Format may be different before and after access, hence connector may transform data from one format to another
- **Persistent data:** data access including query mechanism, such as SQL for data access, accessing information in software components repositories.
- **Transient data:** data access through heap, stack memory , information caching etc.
- Could enable global access, Mutating (changing data) access

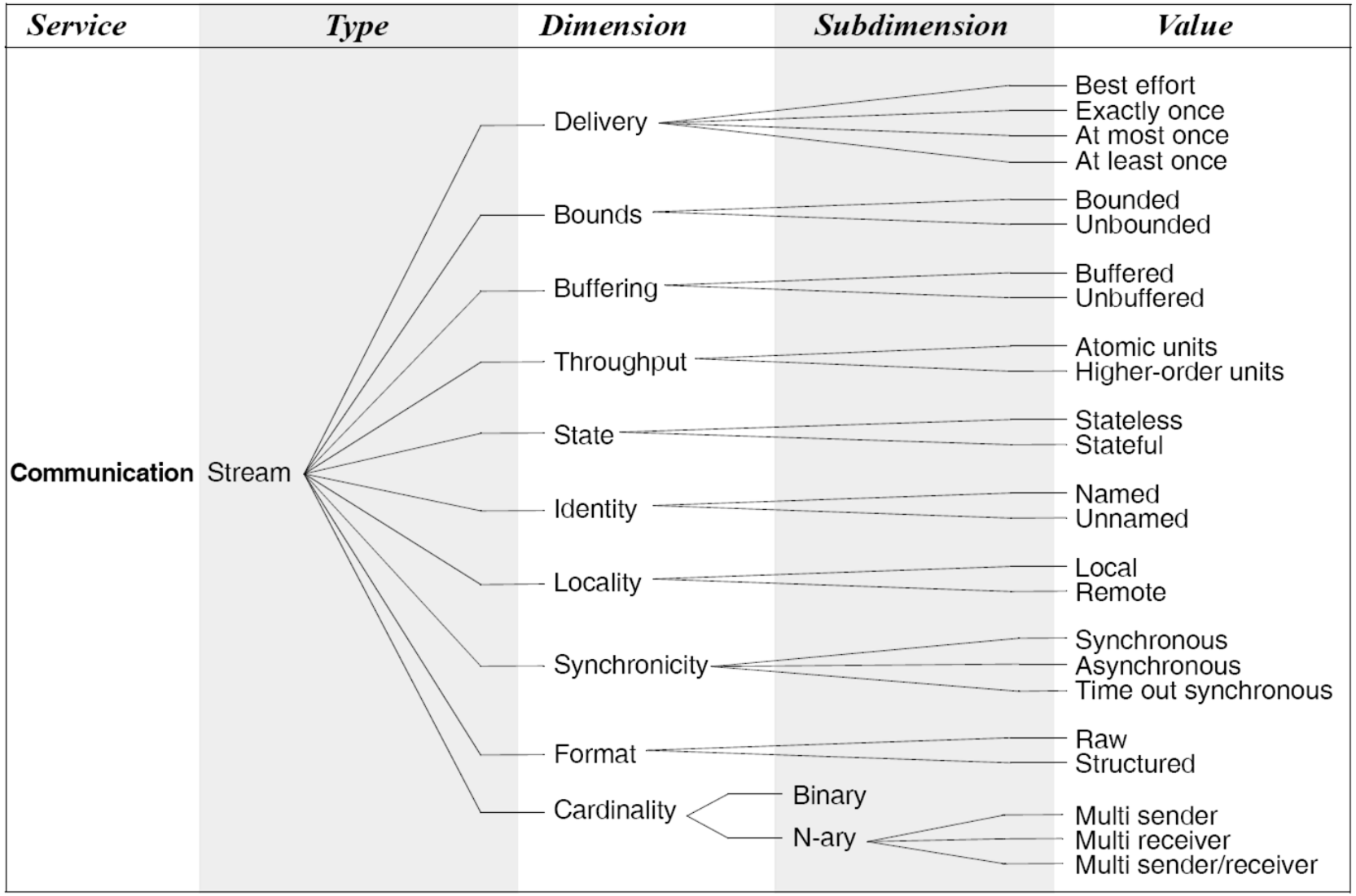
Data Access Connectors



Stream Connectors

- Used for transferring large amount of data between autonomous processes (as a communication service)
- Used in client-server system using data transfer protocols to deliver results of computation
- Can be combined with other types of connectors to provide composite type connectors

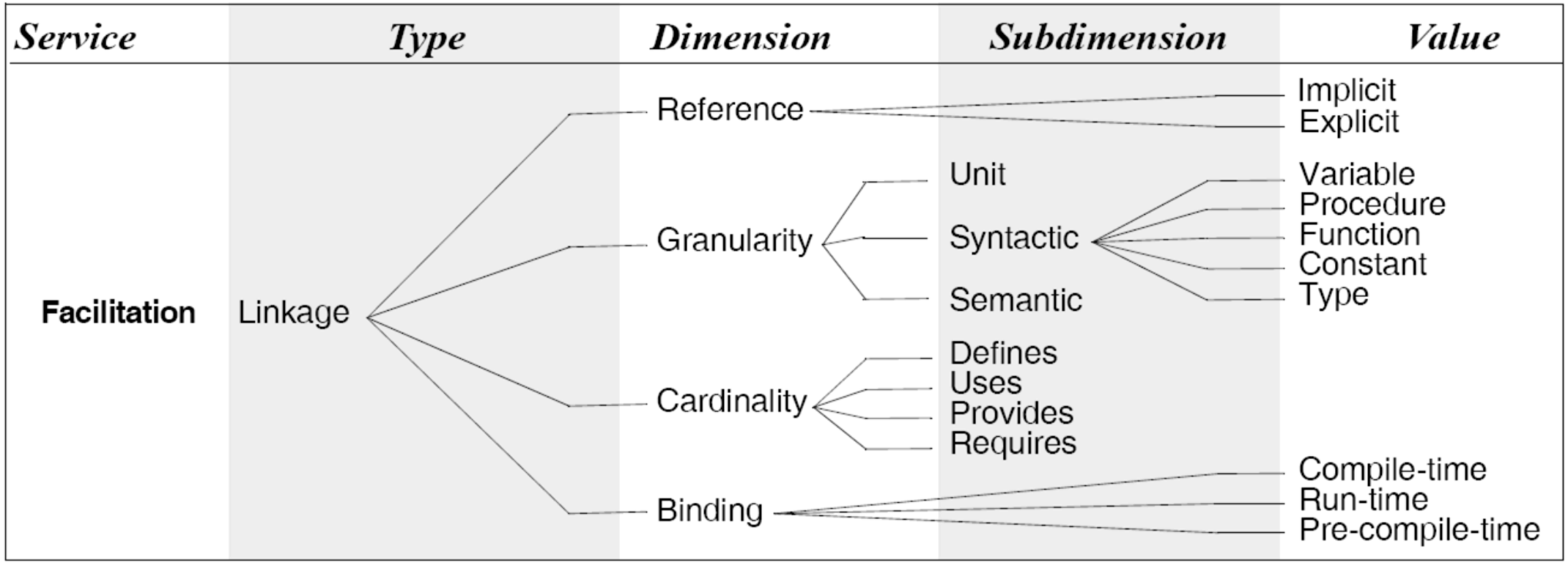
Stream Connectors



Linkage Connectors

- Used to tie the system components together and hold in the same state during operation
- Establishes channels/ ducts for communication and coordination through higher-order connectors to enforce certain semantics.
- Works as facilitations services.
- Used to establish the link and later may disappear
- Semantic interconnections specifies how the linked components are supposed to interact.

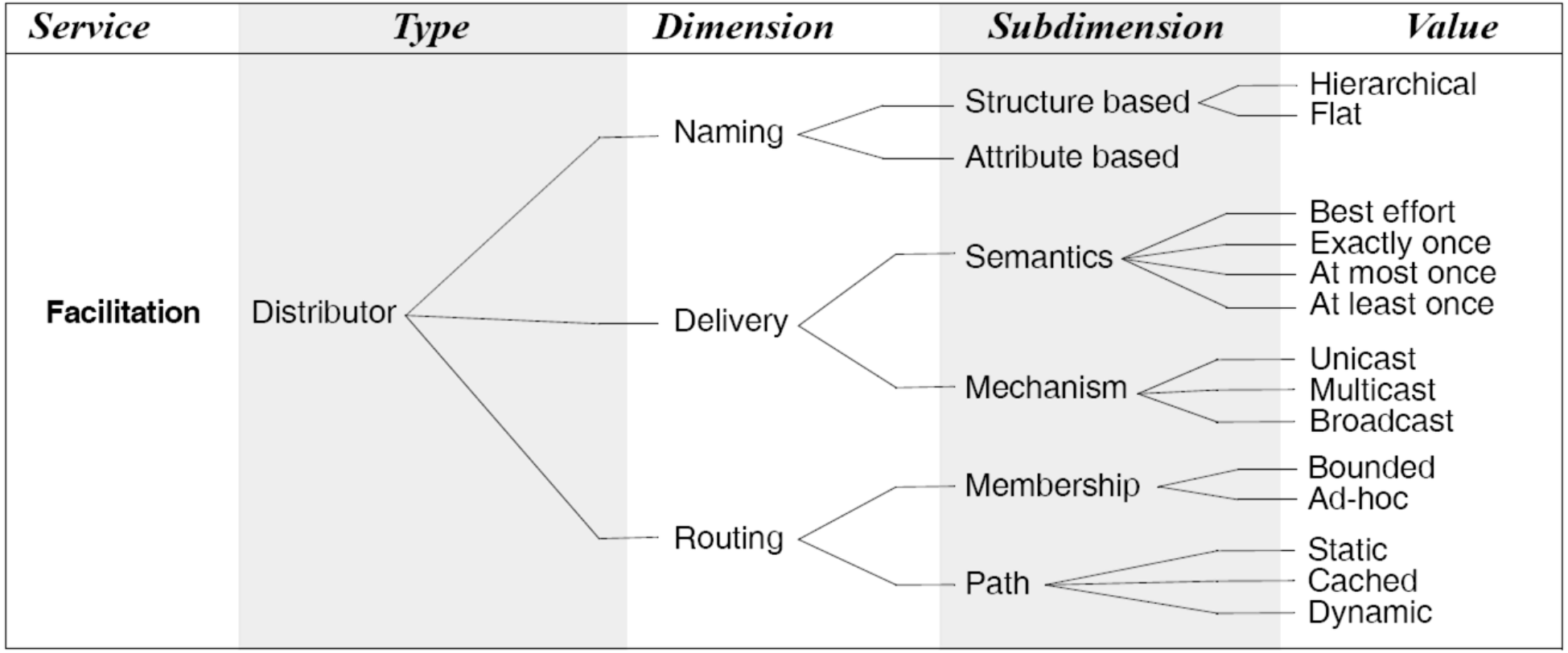
Linkage Connectors



Distributor Connectors

- Performs identification of interaction paths and subsequent routing of communication and coordination information among components along these paths.
- Provides facilitation services
- Always works in association with other connectors such as stream or procedural call

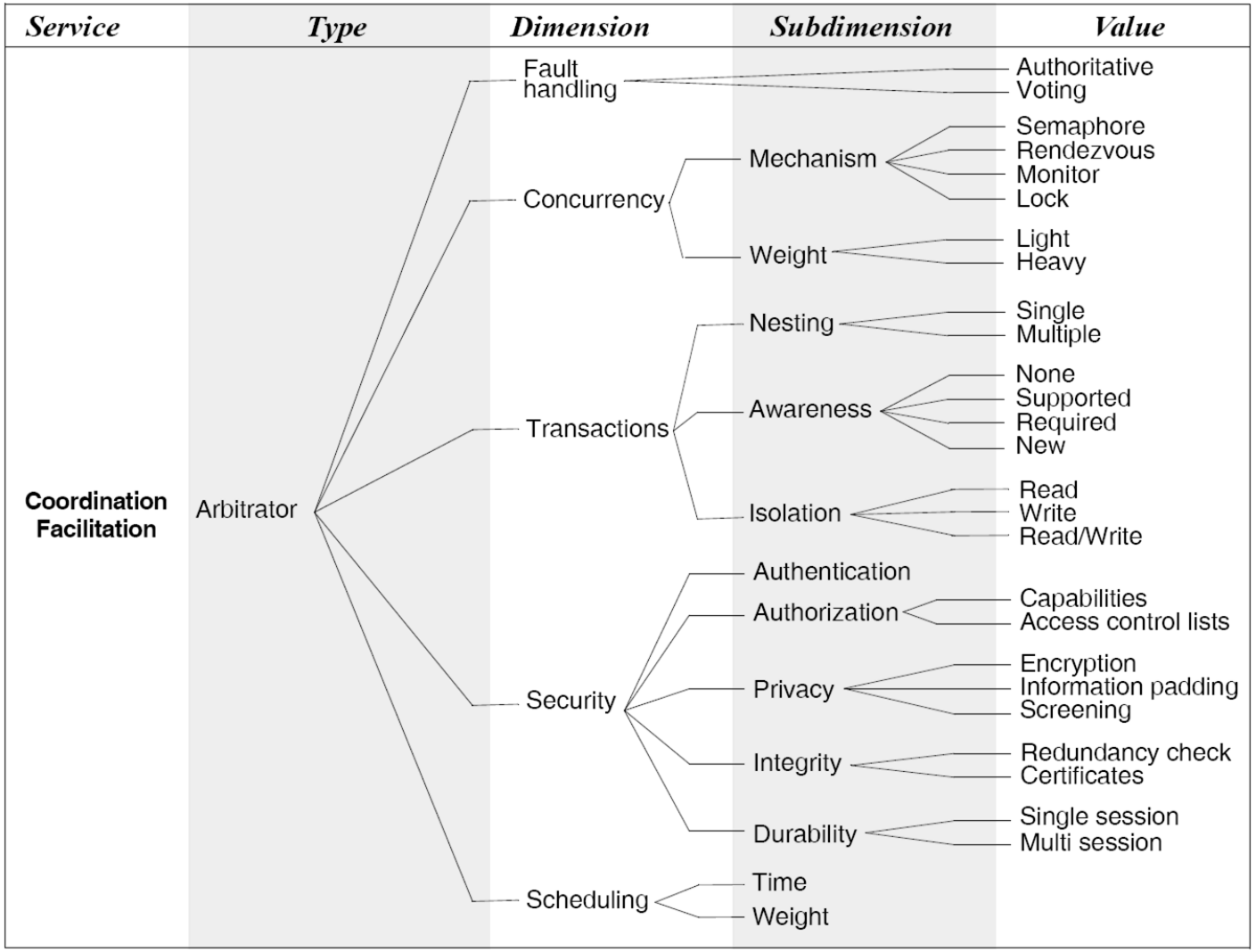
Distributor Connectors



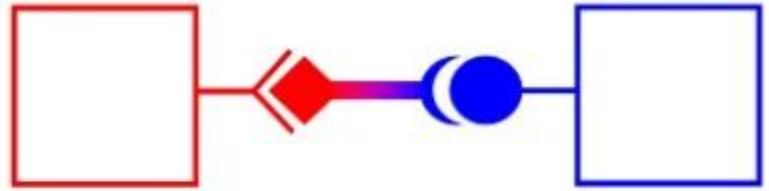
Arbitrator Connectors

- Streamlines system operations provide facilitation by resolving conflicts
- Coordination by redirecting flow of control
- E.g. in multithreaded environment shared memory access is provided through synchronization and concurrency control

Arbitrator Connectors

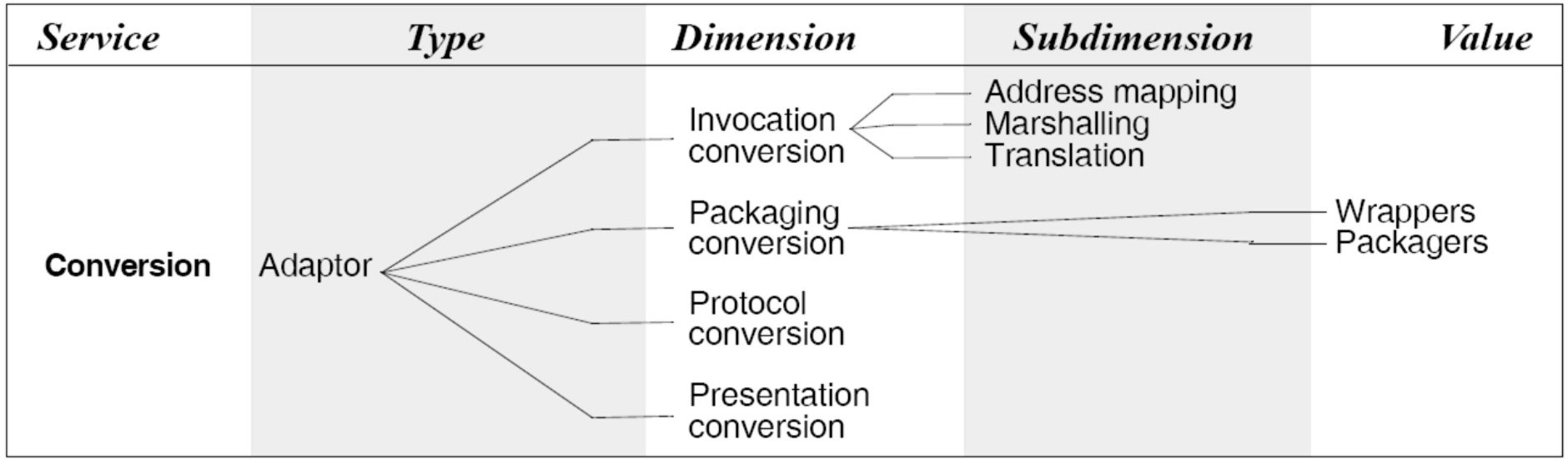


Adaptor Connectors



- Provide facilities to support interaction between components those may mismatch
- Interoperation in heterogeneous environment such as different programming languages or computing platforms

Adaptor Connectors

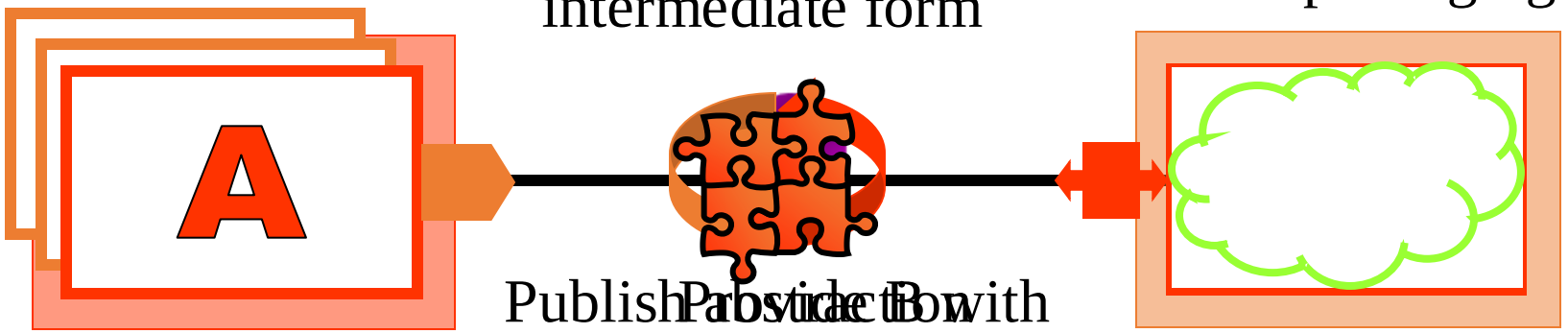


Merits of Connectors

- Connectors allow modeling of arbitrarily complex interactions
- Connector flexibility aids system evolution
  - Component addition, removal, replacement, reconnection, migration
- Support for connector interchange is desired
  - Aids system evolution
  - May not affect system functionality
- Libraries of OTS connector implementations allow developers to focus on application-specific issues
- Difficulties
  - Rigid connectors
  - Connector “dispersion” in implementations
- Key issue
  - Performance vs. flexibility

Role and Challenge of Software Connectors

How do we enable components A and B to interact?



What is the right answer?

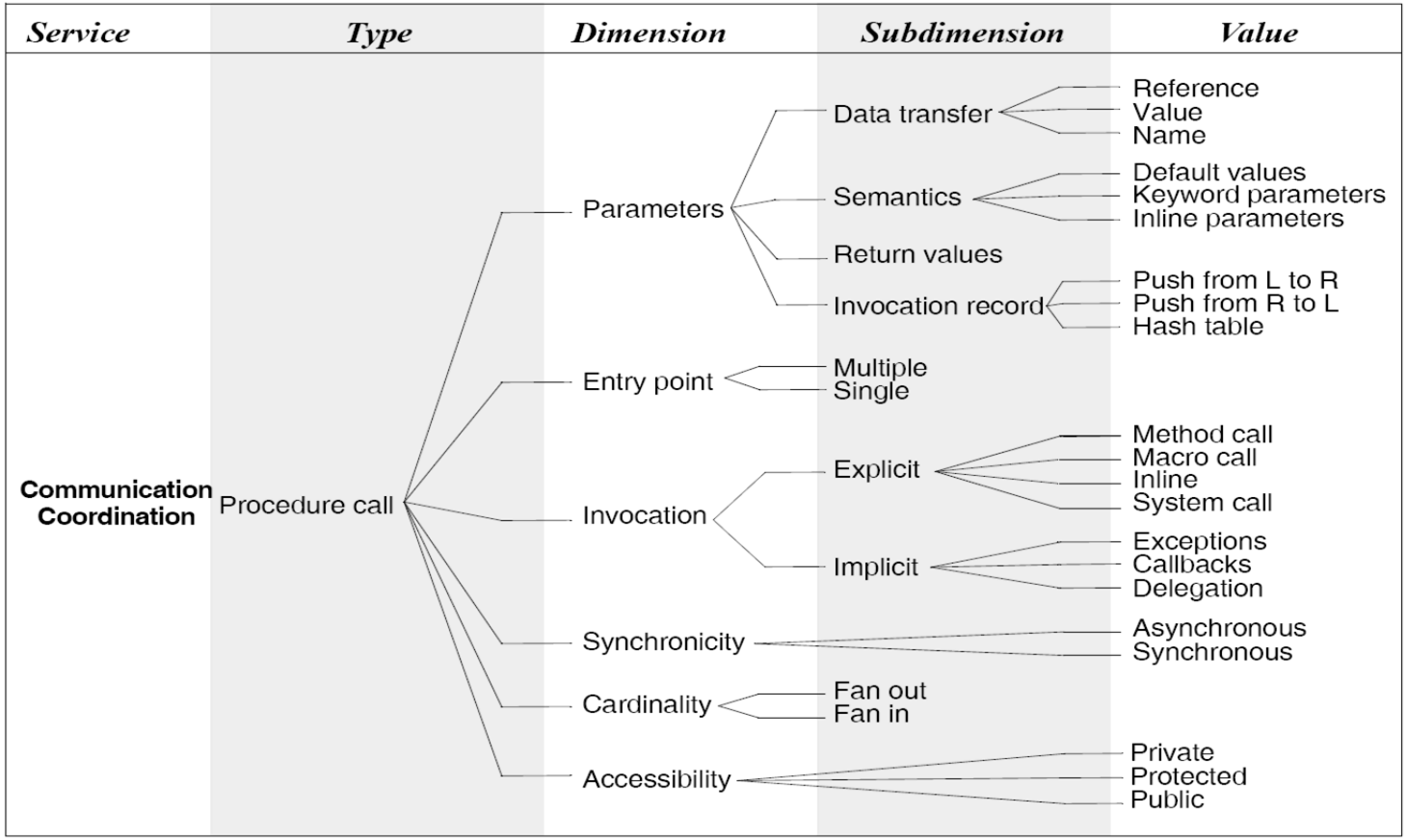
How Does One Select a Connector?

- Determine a system's interconnection and interaction needs
  - Software interconnection models can help
- Determine roles to be fulfilled by the system's connectors
  - Communication, coordination, conversion, facilitation
- For each connector
  - Determine its appropriate type(s)
  - Determine its dimensions of interest
  - Select appropriate values for each dimension
- For multi-type, i.e., composite connectors
  - Determine the atomic connector compatibilities

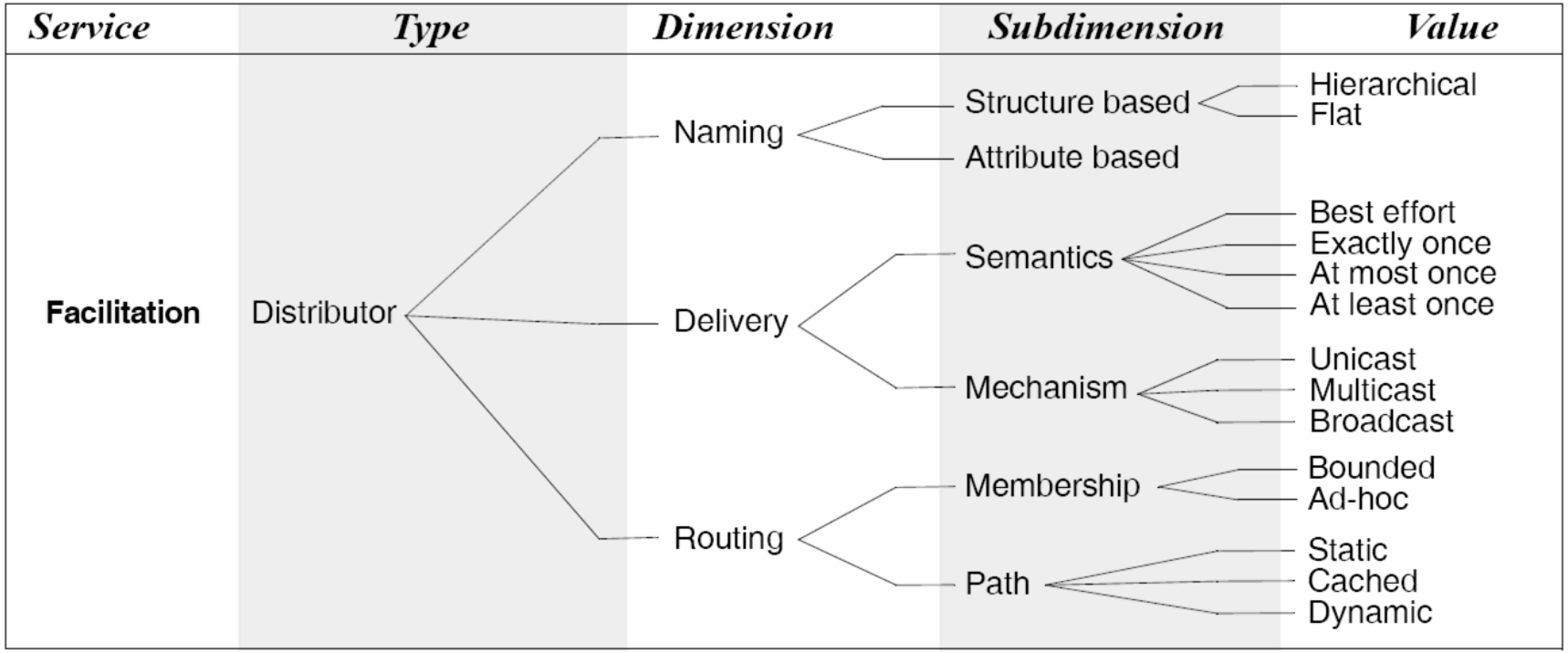
Simple Example

- System components will execute in two processes on the same host
  - Mostly intra-process
  - Occasionally inter-process
- The interaction among the components is synchronous
- The components are primarily computation-intensive
  - There are some data storage needs, but those are secondary
- Select procedure call connectors for intra-process interaction
- Combine procedure call connectors with distributor connectors for inter-process interaction
  - RPC
- Select the values for the different connector dimensions
  - What are the appropriate values?
  - What values are imposed by your favorite programming language(s)?

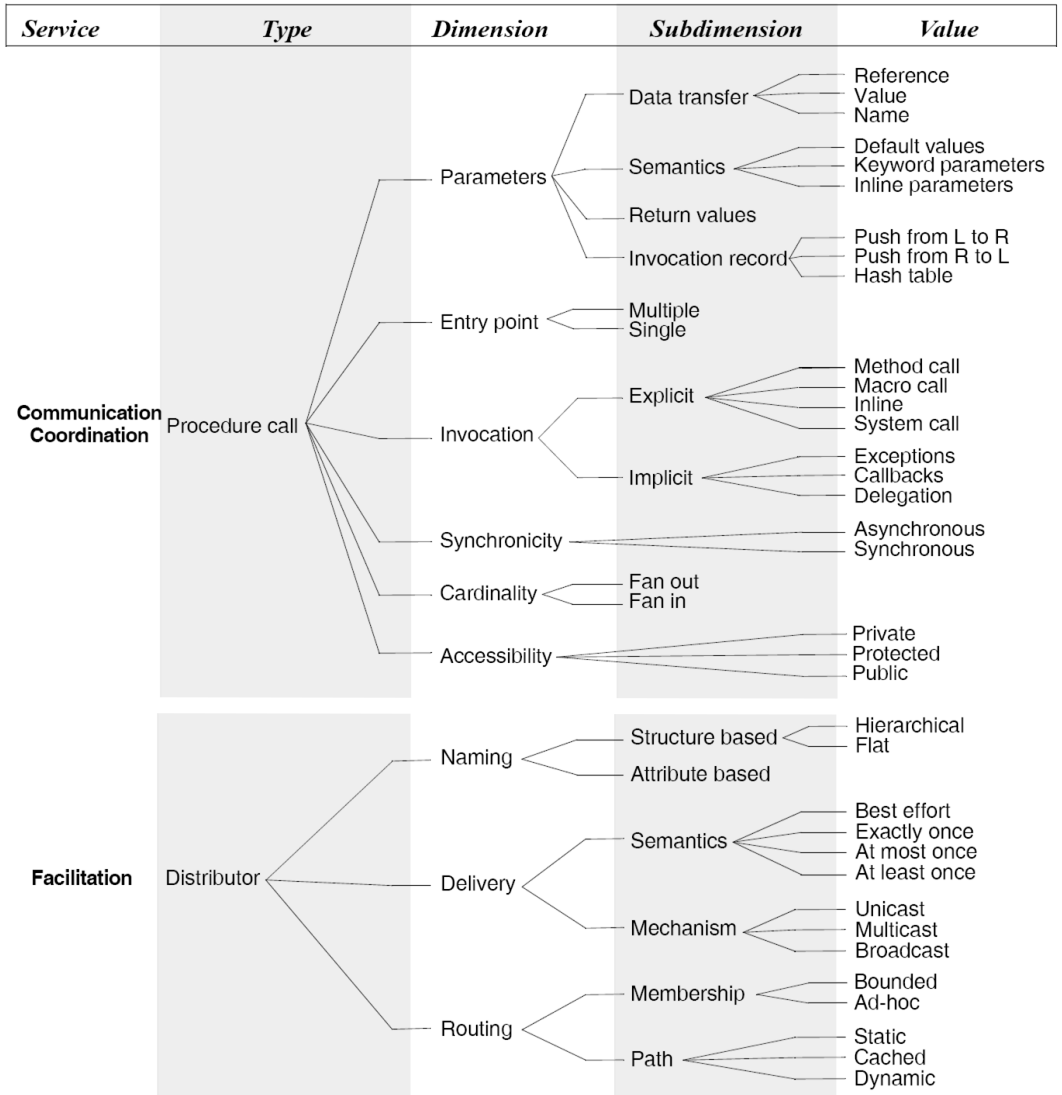
Procedure Call Connectors Revisited



Distributor Connectors Revisited



Two Connector Types in Tandem



Select the appropriate values for PC and RPC!

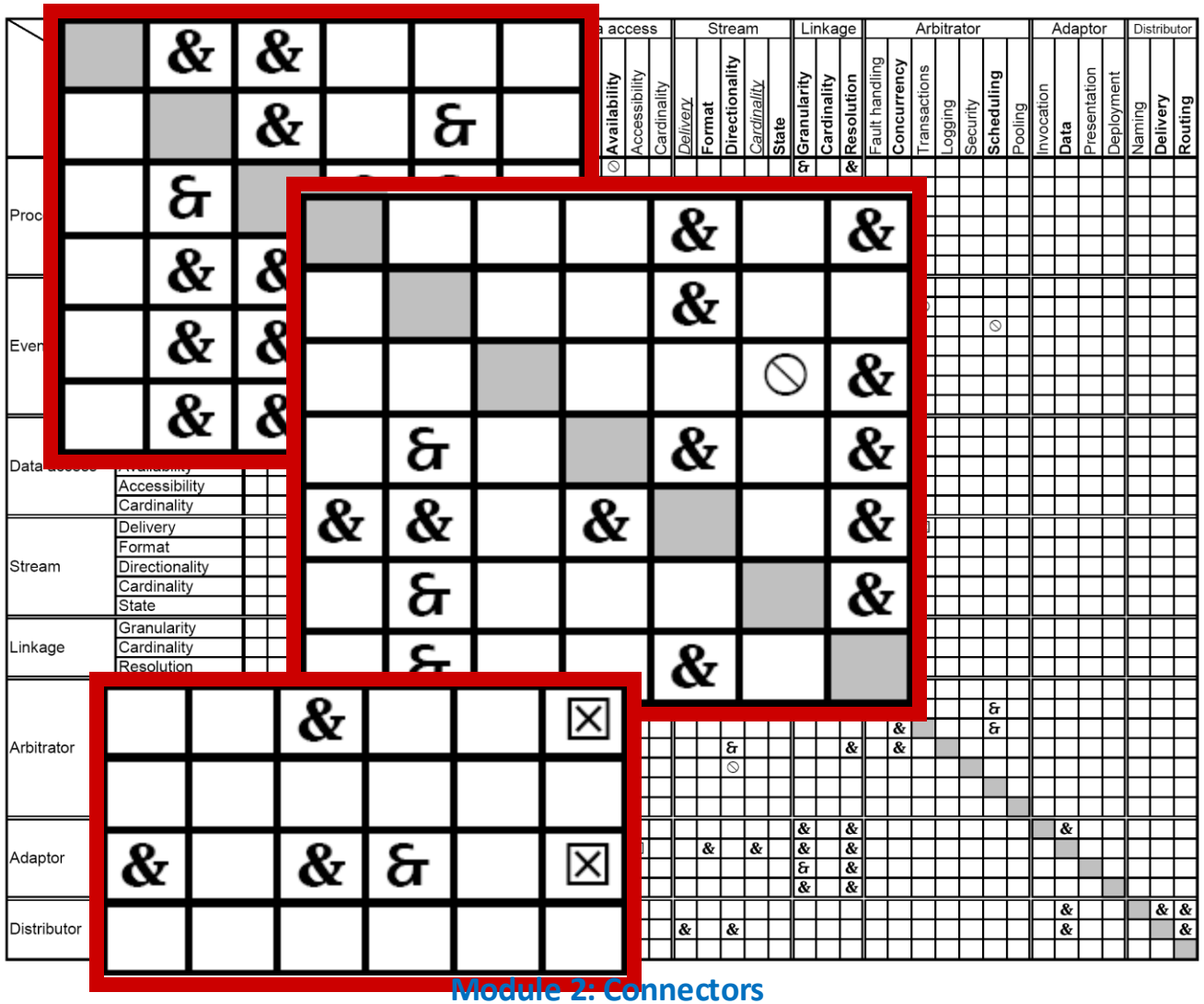
Selecting appropriate type of Connector

1. Select the specific set of interacting components.
2. Determine the interaction services the components need.
   a) Identify specific characteristics of the components' interaction
   b) Study components' architectural descriptions, implementation language and/ or framework
3. Based on the identified interaction services, determine a subset of connector types that comprise the initial candidate set for providing those services.
4. Evaluate each connectors type from the chosen subset based on the details of interaction requirements
5. For each of the remaining candidate connector types, set the values for the necessary dimensions and subdimensions as appropriate

Connector Dimension Inter-Relationships

- Requires – &
  - Choice of one dimension mandates the choice of another
- Prohibits – ☒
  - Two dimensions can never be composed into a single connector
- Restricts – ⊘
  - Dimensions are not always required to be used together
  - Certain dimension combinations may be invalid
- Cautions – &
  - Combinations may result in unstable or unreliable connectors

Dimension Inter-Relationships



Well Known Composite Connectors

- Grid connectors (e.g., Globus)
  - Procedure call
  - Data access
  - Stream
  - Distributor
- Peer-to-peer connectors (e.g., Bittorrent)
  - Arbitrator
  - Data access
  - Stream
  - Distributor
- Client-server connectors
- Event-based connectors