Software Connectors

Software Architecture

Intro

- making architectural design decisions that pertain to integrating those components effectively and ensuring the components' proper interaction in a system is just as important.
- assembling functional components and supporting interactions between components.
- perform transfer of control and data among components.
- provide services, such as persistence, invocation, messaging, and transactions, that are largely independent of the interacting components' functionalities.

Intro

- These services are usually considered to be "facilities components" in widely used middleware standards such as COREA, DCOM, and RMI.
- Recognizing these facilities as connectors helps to clarify an architecture and keep the components' focus on application- and domain-specific concerns.
- Treating these services as connectors rather than components can also foster their reuse across applications and domains.
- Connectors allow architects and engineers to compose heterogeneous functionality, developed at different times, in different locations, by different organizations.
- Connectors can be thought of quite appropriately as the guards at the gate of **separation of concerns**

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 [*JDBC*]
  - Asynchronous event multicast
- Each connector provides
  - Interaction duct(s)
  - Transfer of control and/or data

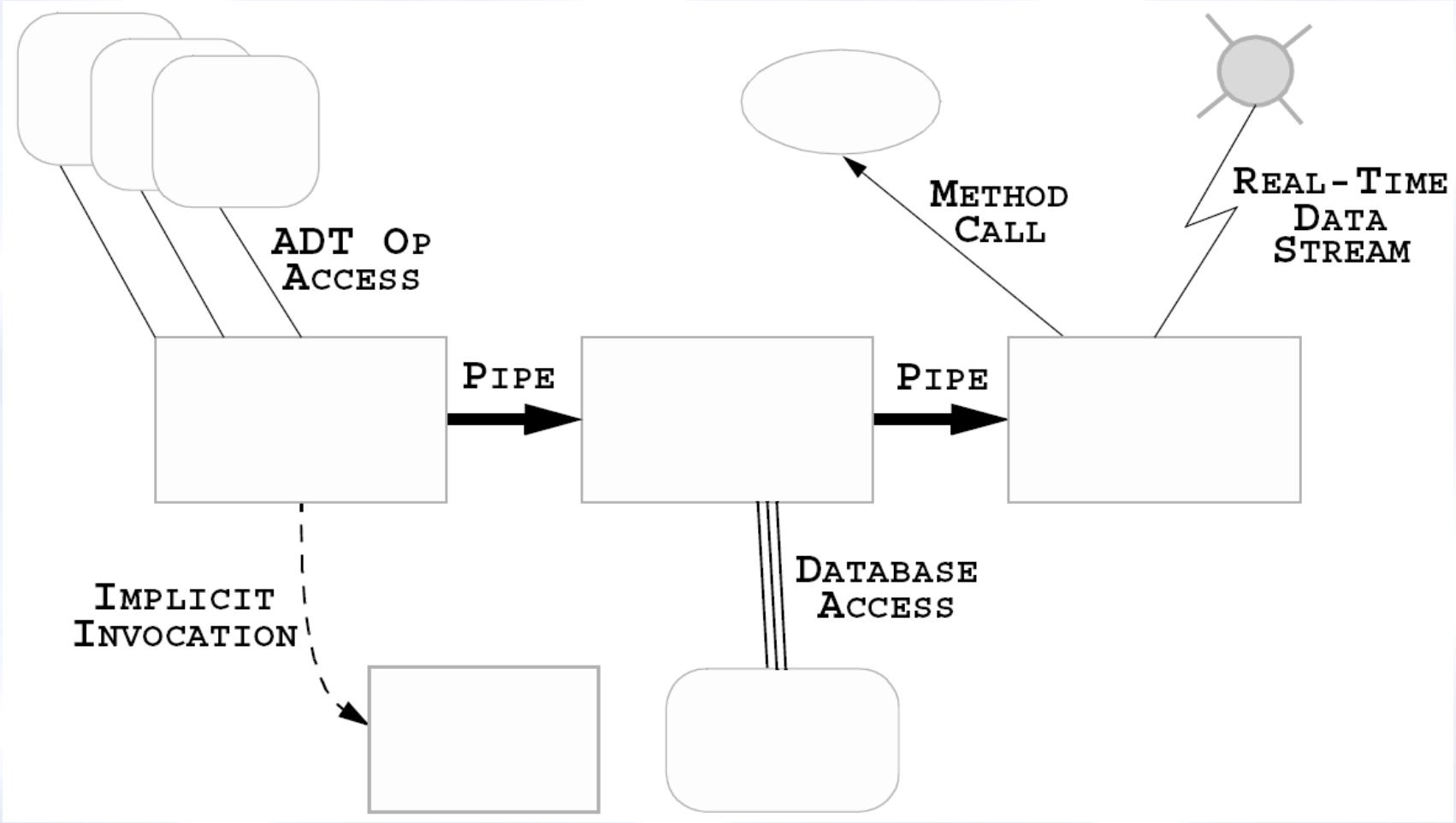
What is a Software Connector?

- Traditional software engineering approaches embrace a rather narrow view of connectors that is unlikely to be successfully applicable across all development situations.
- This problem is further exacerbated by the increased emphasis on development using large, off-the-shelf components originating from multiple sources.
- As components become more complex and heterogeneous, the interactions among them become more critical determinants of system properties.

What is a Software Connector?

- Very important dimension of a software architect's job is to:
  - Understand the component interaction needs.
  - Carefully identify all of the relevant attributes of that interaction.
  - Select the candidate connectors.
  - Assess any trade-offs associated with each candidate.
  - Analyze the key properties of the resulting component, connector assemblies.

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

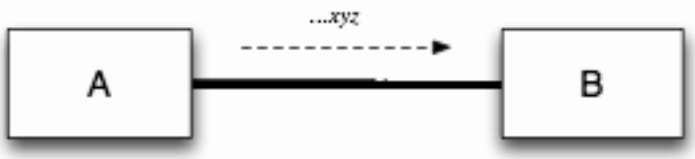
Treating Connectors Independently (cont'd)

- 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

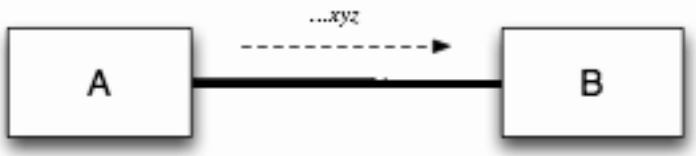
An Example of Explicit Connectors



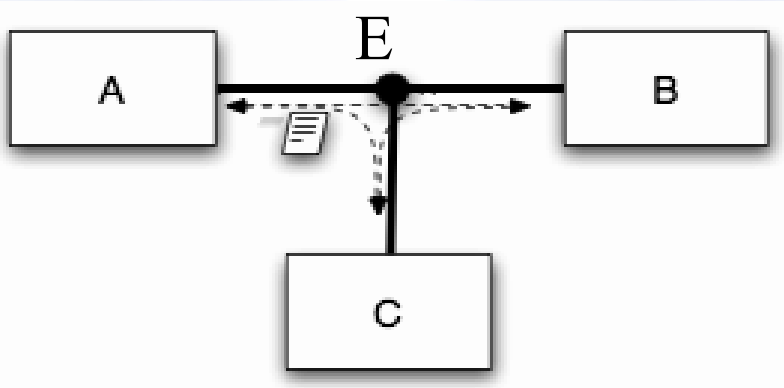
A simple Pipe-and-Filter architecture consisting of two filters, A and B, communicating via untyped data streams through the unidirectional pipe connector P

If, however, we want to change the nature of the data from an unformatted byte stream to discrete, typed packets that can be processed more efficiently by the interacting components, pipes will not suffice .

An Example of Explicit Connectors (cont'd)



?



An *event-based architecture consisting of two components,* A *and* B, *communicating via typed discrete data packets (events) through the event bus* connector E, *The* connector *allows on-the-fly system modifications, such as adding a new component* C,

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 通信
  - Coordination 协调
  - Conversion 转换
  - Facilitation 便利

Connectors as Communicators

- Main role associated with connectors
- Supports
  - Different communication mechanisms
    - e.g. procedure call, RPC, shared data access, message passing
  - Constraints on communication structure/direction
    - e.g. pipes
  - Constraints on quality of service
    - e.g. persistence
- Separates communication from computation (SOC)
- May influence non-functional system characteristics
  - e.g. performance, scalability, security

Connectors as Coordinators

- Determine computation control
- Control delivery of data
- Separates control from computation
- Orthogonal to communication, conversion, and facilitation

  Elements of control are in communication, conversion and facilitation

support transfer of control among components.
passing the thread of execution to each other.
Function calls and method invocations are examples of coordination connectors.

Connectors as Converters

- Enable interaction of independently developed, mismatched components
- Mismatches based on interaction
  - Type
  - Number
  - Frequency
  - Order
- Examples of converters
  - Adaptors (adapts input to other interface, a bridge)
  - Wrappers (simplify API)

Connectors as Facilitators

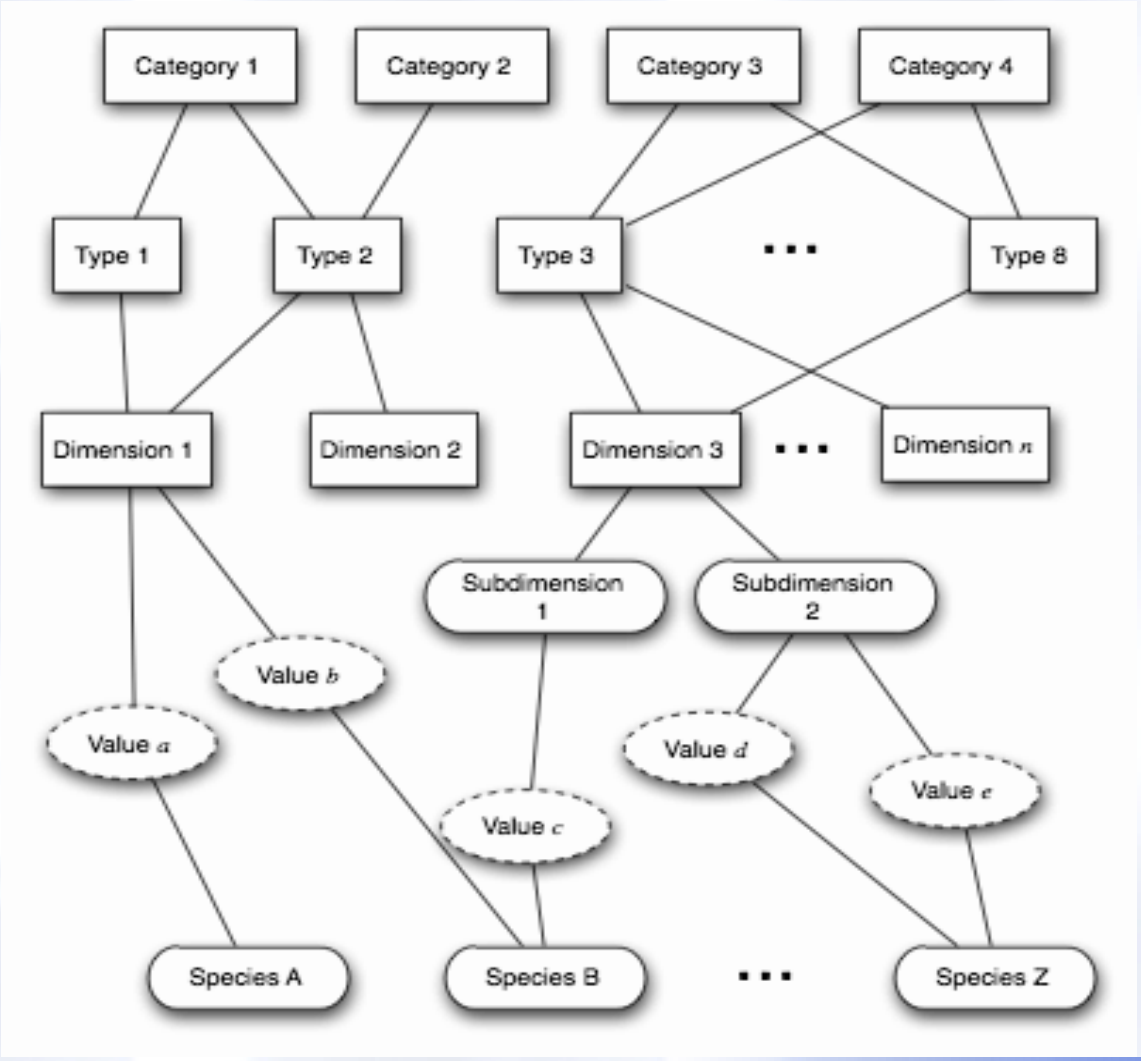
- Enable interaction of components intended to interoperate
  - Mediate and streamline interaction
- Govern access to shared information
- Ensure proper performance profiles (non-functional req.)
  - e.g., load balancing, scheduling services, concurrency control
- Provide synchronization mechanisms
  - Critical sections
  - Monitors

further facilitating and optimizing components' interactions

Connector Types

- Procedure call
- Data access
- Event
- Stream
- Linkage
- Distributor
- Arbitrator
- Adaptor

A Framework for Classifying Connectors

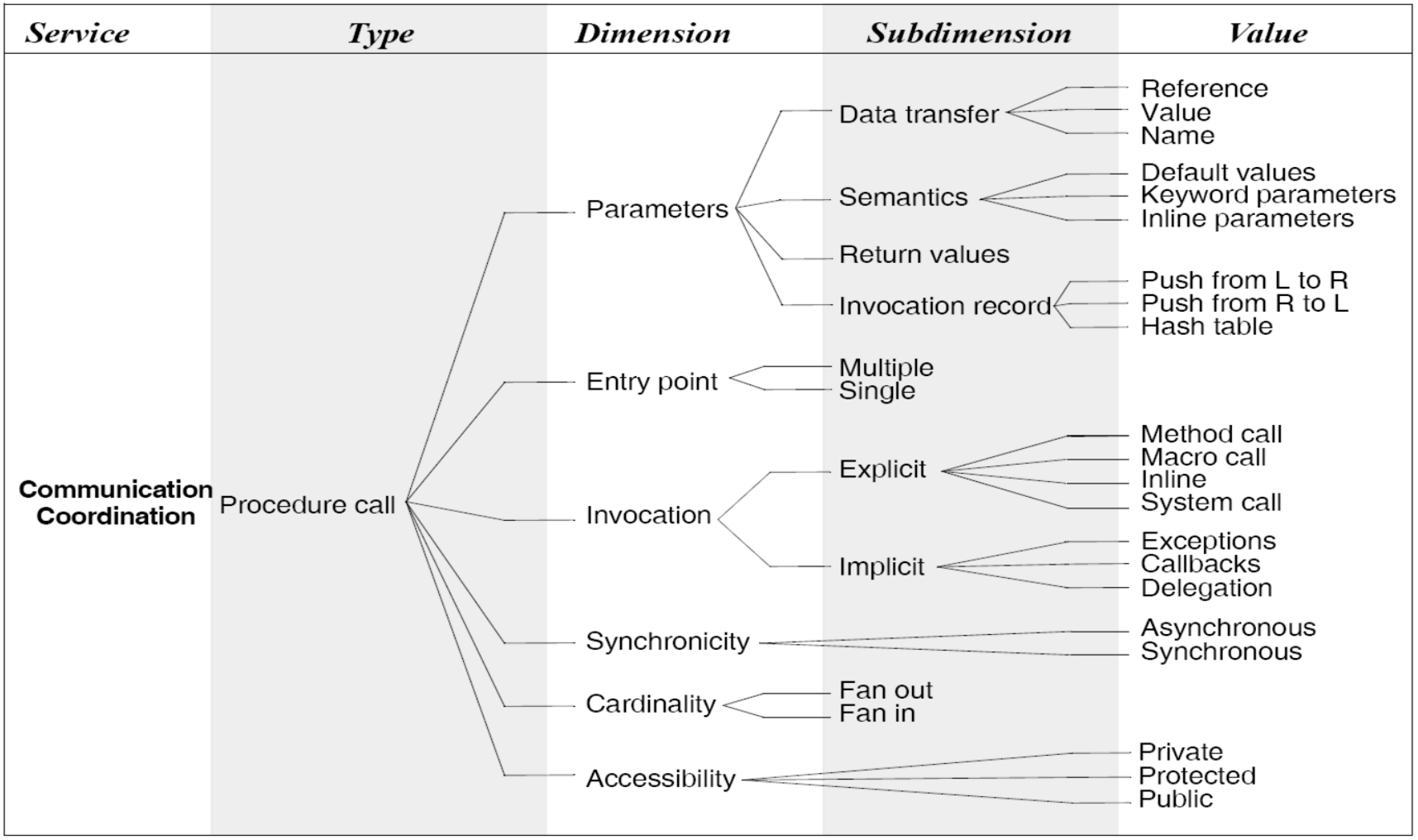


- Each connector is identified by its primary service category and further refined based on the choices made to realize these services.
- A particular connector instance (that is, *species)* can take a number of values from different types.
- does not result in a strict hierarchy, but rather in a directed acyclic graph.

Procedure Call Connectors

- model the flow of control among components through various invocation techniques.
- perform transfer of data among the interacting components through the use of parameters and return values.
- the most widely used and best understood connectors
- Examples include object-oriented methods, fork and exec in Unix-like environments, call-back invocation in event-based systems, and operating system calls.
- frequently used as the basis for composite connectors, such as remote procedure calls RPC, which also perform facilitation services.

Procedure Call Connectors



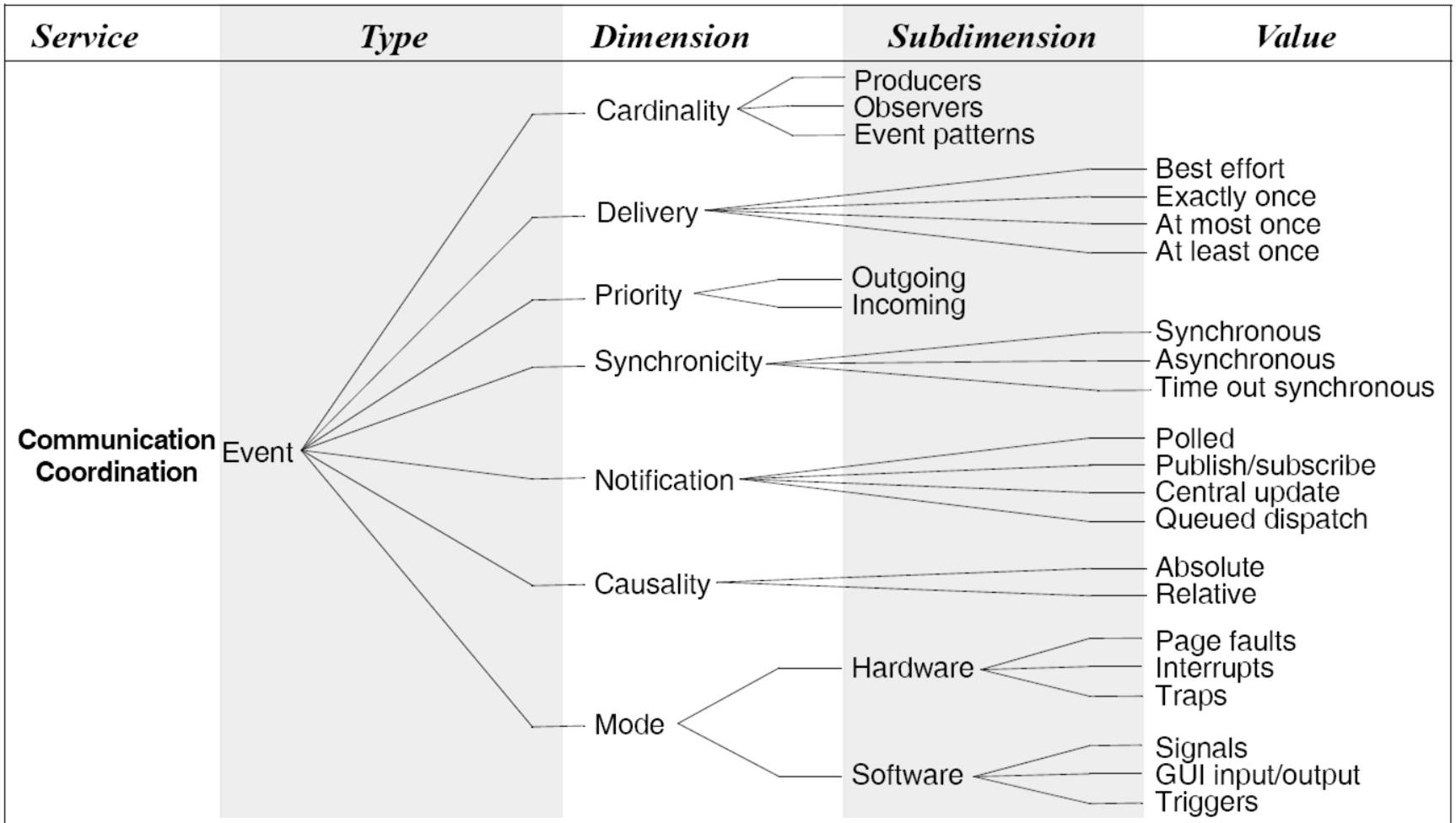
Event Connectors

- Similar to procedure call connectors in that they affect the flow of control among components. The flow is precipitated by an event.
- Once the event connector learns about the occurrence of an event, it generates messages (event notifications) for all interested parties and yields control to the components for processing those messages.
- The contents of an event can be structured to contain more information about the event, such as the time and place of occurrence, and other Application-specific data.
- Event connectors therefore also provide communication service.

Event Connectors

- Event connectors are also different from procedure calls in that virtual connectors are formed between components interested in the same event topics.
- Those connectors may appear and disappear dynamically depending on the components' changing interests.
- Event-based distributed systems rely on the notion of time and ordering of actions. Dimensions such as causality, atomicity, and synchronicity play a critical role.
- Event connectors are found in distributed applications that require asynchronous communication.

Event Connectors



Eg. the cardinality of a multicasting event will be a single producer and multiple observer components.

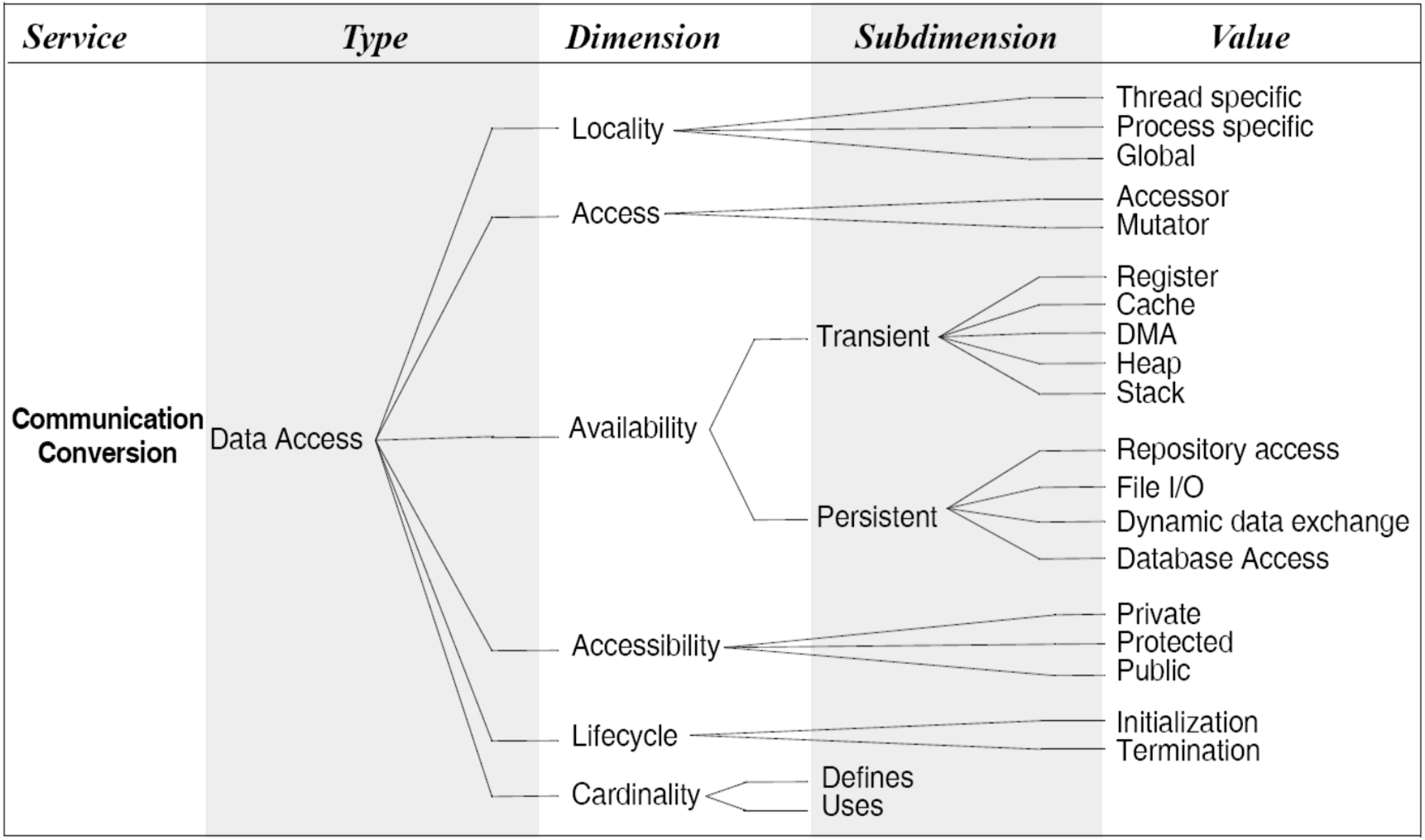
Data Access Connectors

- Data access connectors allow components to access data maintained by a datastore component (communication services).
- Data access often requires preparation of the data store before and cleanup after access has been completed.
- In case there is a difference in the format of the required data and the format in which data is stored and provided, data access connectors may perform translation of the information being accessed, that is, conversion.

Data Access Connectors

- The data can be stored either persistently or temporarily.
- Examples of persistent data access include query mechanisms, such as SQL for database access, and accessing information in repositories, such as a software component repository.
- Examples of transient data access include heap and stack memory access and information caching.

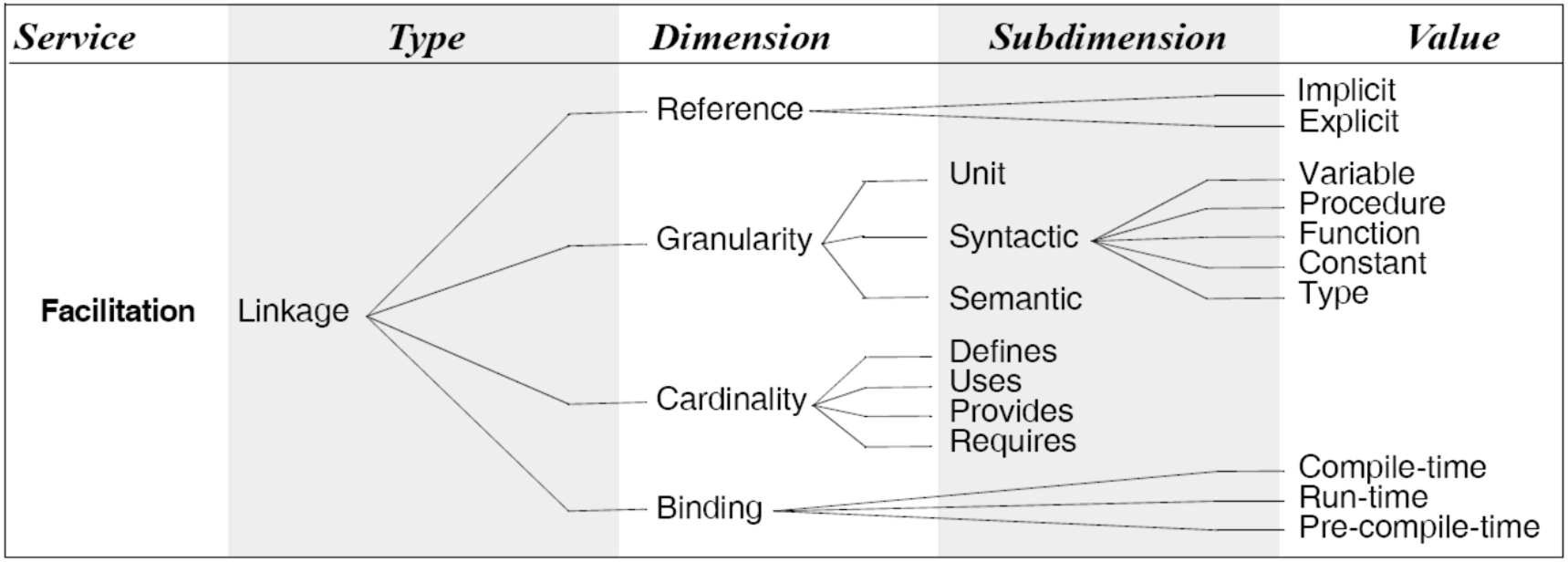
Data Access Connectors



Linkage Connectors

- are used to tie the system components together and hold them in such a state during their operation.
- enable the establishment of ducts-the channels for communication and coordination;
- then used by higher-order connectors to enforce interaction semantics. In other words, linkage connectors provide facilitation services.

Linkage Connectors

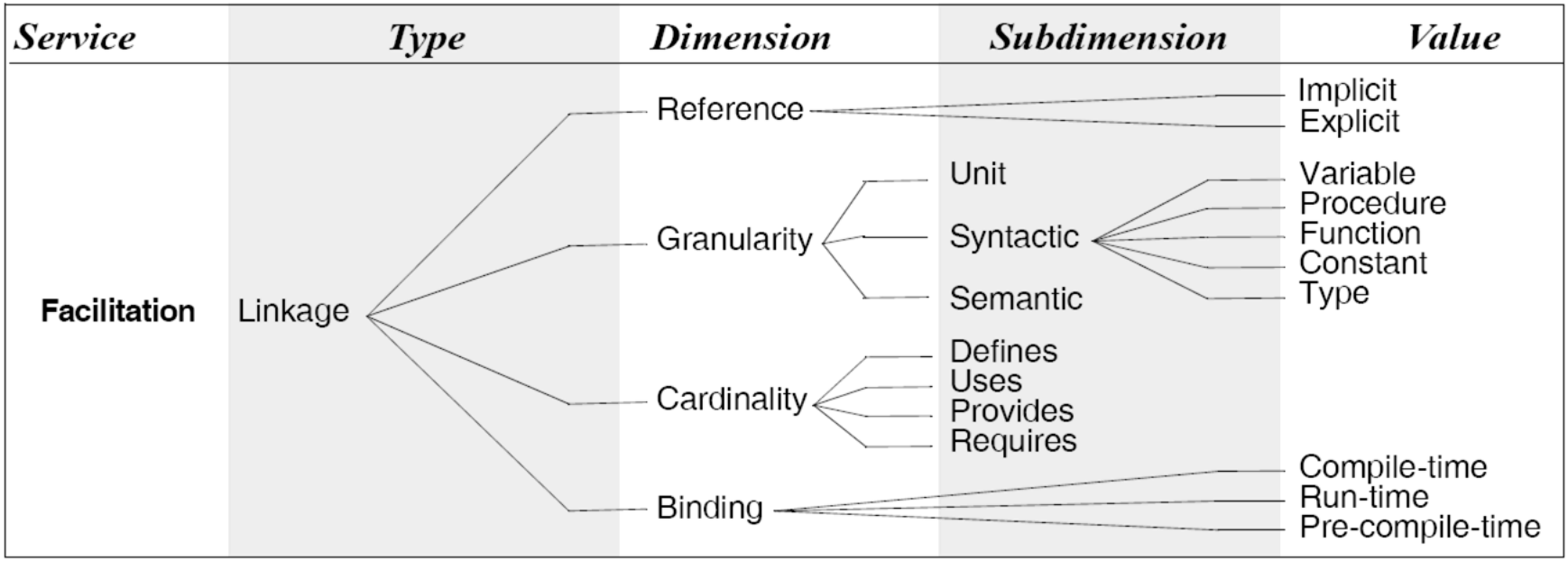


The granularity dimension refers to the size of components and level of detail required to establish a linkage.

Unit interconnection specifies only that one component-which can be a module, an object, or a file-depends on another. Eg. configuration management and system building facilities such as Make.

Semantic interconnection ensures that the interaction requirements and constraints are explicitly stated and satisfied.

Linkage Connectors



The cardinality of a linkage connector refers to the number of places in which a system resource-such as a component, procedure, or variable-is defined, used, provided, or required.

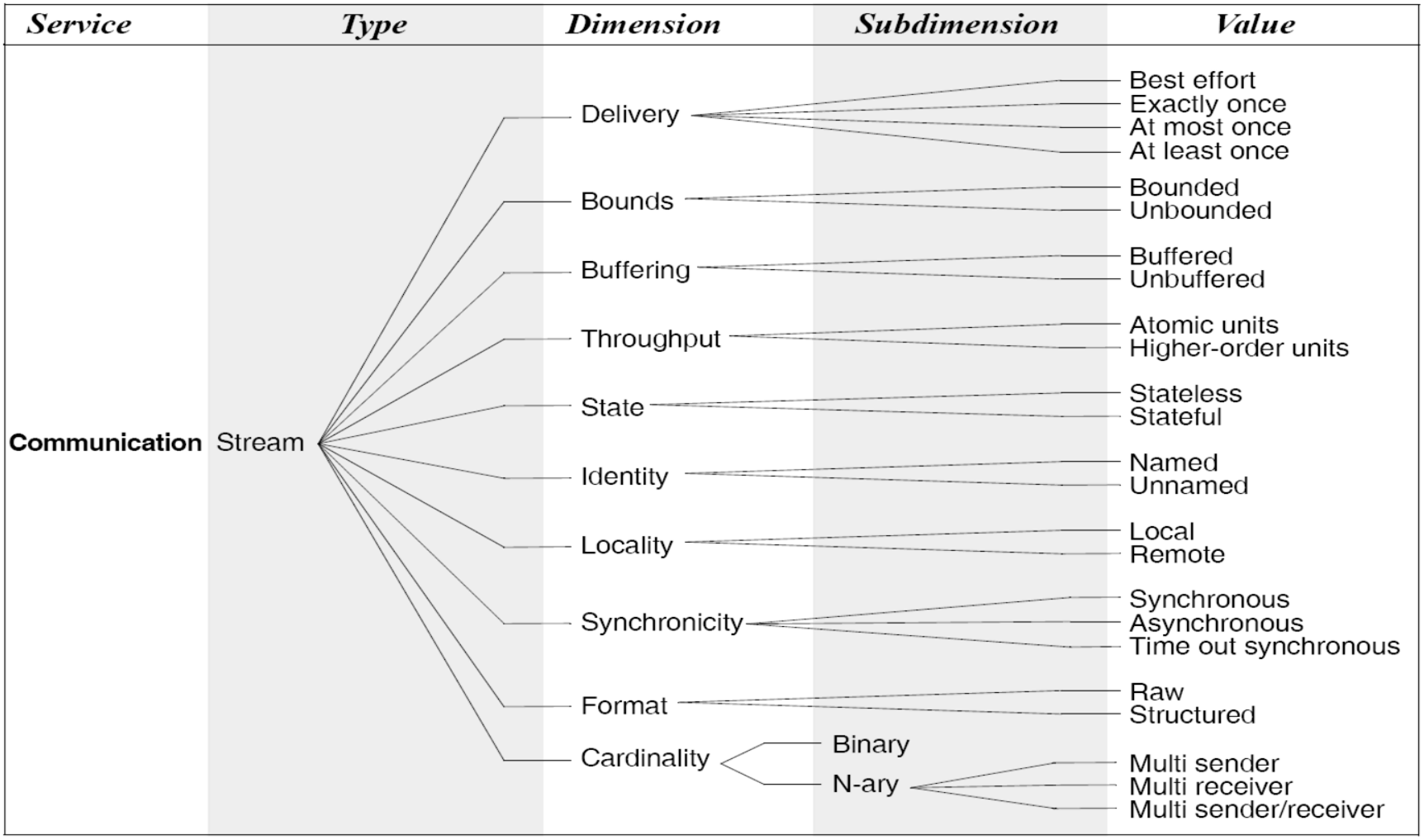
A resource is *defined* and/or *provided* in a single location and *used* or *required* from multiple locations.

A linkage connector can **establish the binding** between components *very early*, *early*, or *late*.

Stream Connectors

- perform transfers of large amounts of data between autonomous processes, thus provide communication services in a system.
- Streams are also used in client-server systems with data transfer protocols to deliver results of computation. Streams can be combined with other connector types, such as data access connectors, to provide composite connectors for performing database and file storage access, event connectors, to multiplex the delivery of a large number of events.
- Examples: Unix pipes, TCP/UDP communication sockets, and proprietary client-server protocols.

Stream Connectors



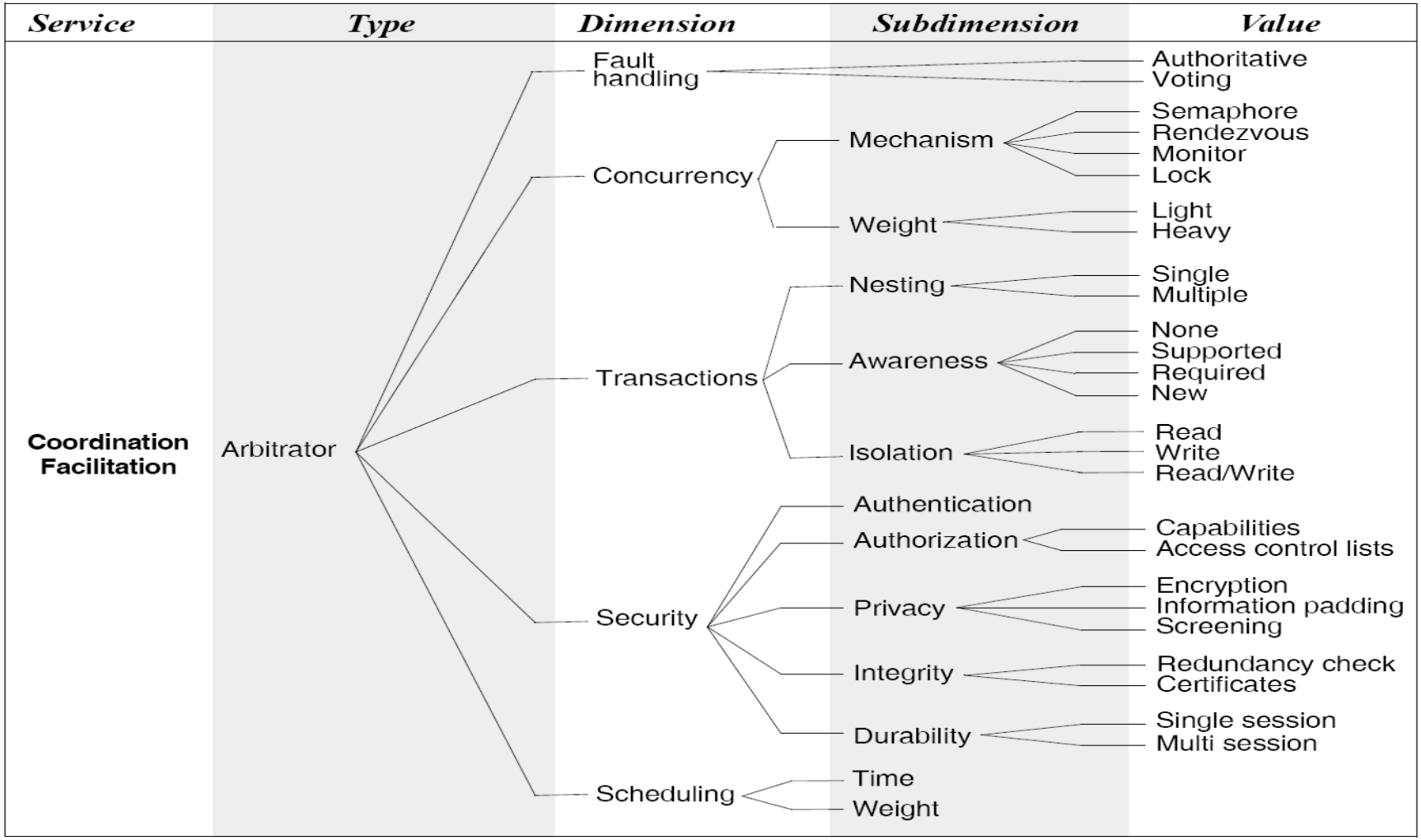
Arbitrator Connectors

- When components are aware of the presence of other components but cannot make assumptions about their needs and state, arbitrators streamline system operation and resolve any conflicts (thereby providing facilitation services), and redirect the flow of control (providing coordination services).

Arbitrator Connectors

- e.g.
  - use synchronization and concurrency control to guarantee consistency and atomicity of operations.
  - negotiate service levels and mediate interactions requiring guarantees for reliability and atomicity.
  - provide scheduling and load balancing services.
  - ensure system trustworthiness by providing crucial support for dependability in the form of reliability, safety, and security.

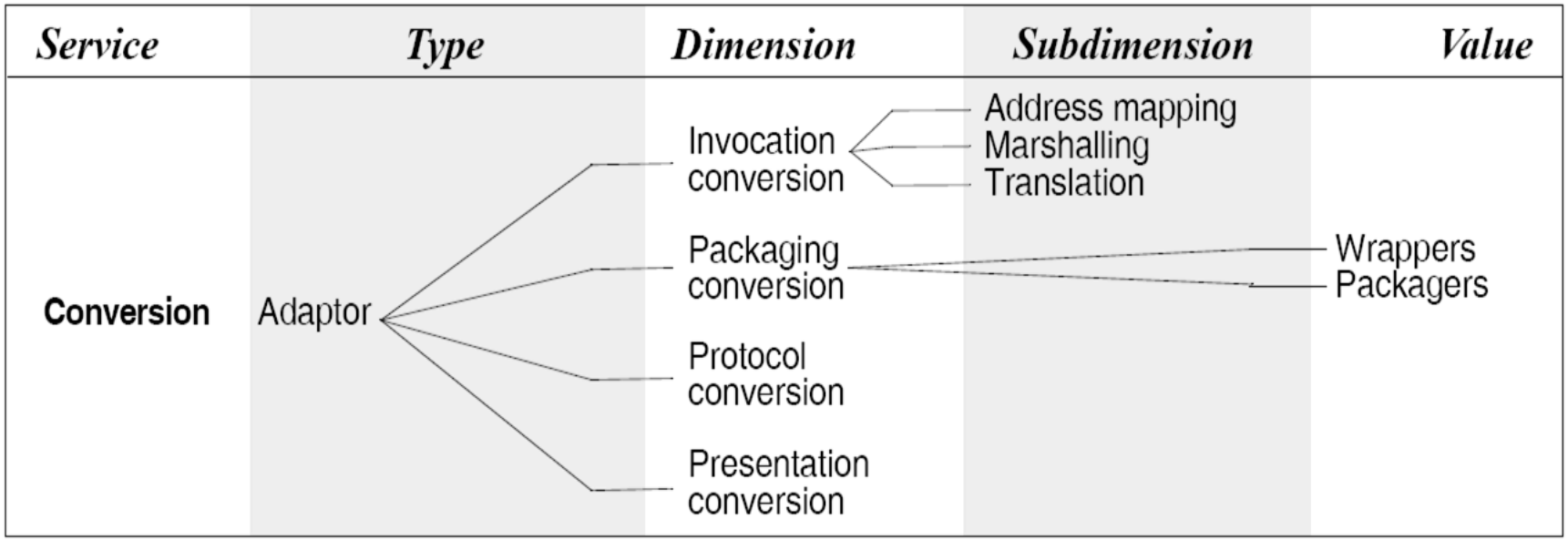
Arbitrator Connectors



Adaptor Connectors

- provide facilities to support interaction between components that have not been designed to interoperate.
- involve matching communication policies and interaction protocols among components, thereby providing conversion services.
- interoperation of components in heterogeneous environments.
- optimize component interactions for a given execution environment: remote call->convert to->local call

Adaptor Connectors

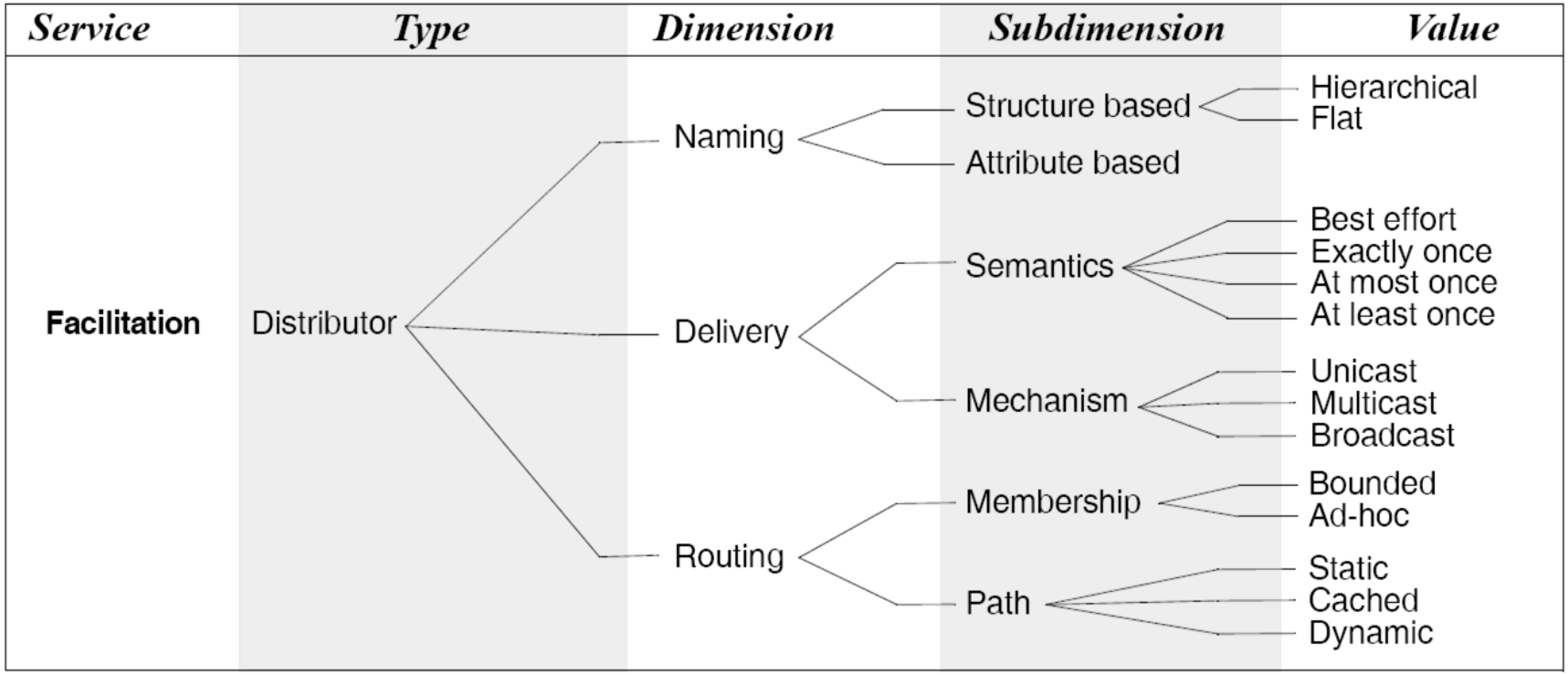


- virtual memory translation matches incompatible component interaction **protocols** （protocol conversion）
- virtual function tables used for dynamic dispatch of polymorphic method calls (invocation conversion)
- packagers, which separate a component's internal functionality from the manner in which it is accessed(packing conversion)

Distributor Connectors

- perform the identification of interaction paths and subsequent routing of communication and coordination information among components along these paths (facilitation services).
- provide assistance to other connectors, such as streams or procedure calls.
- Distributed systems use distributor connectors to direct the data flow.
- Domain name service (DNS), routing, switching, and many other network services belong to this type.
- have an important effect on system scalability and survivability.

Distributor Connectors



Discussion

- 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

Discussion

- 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