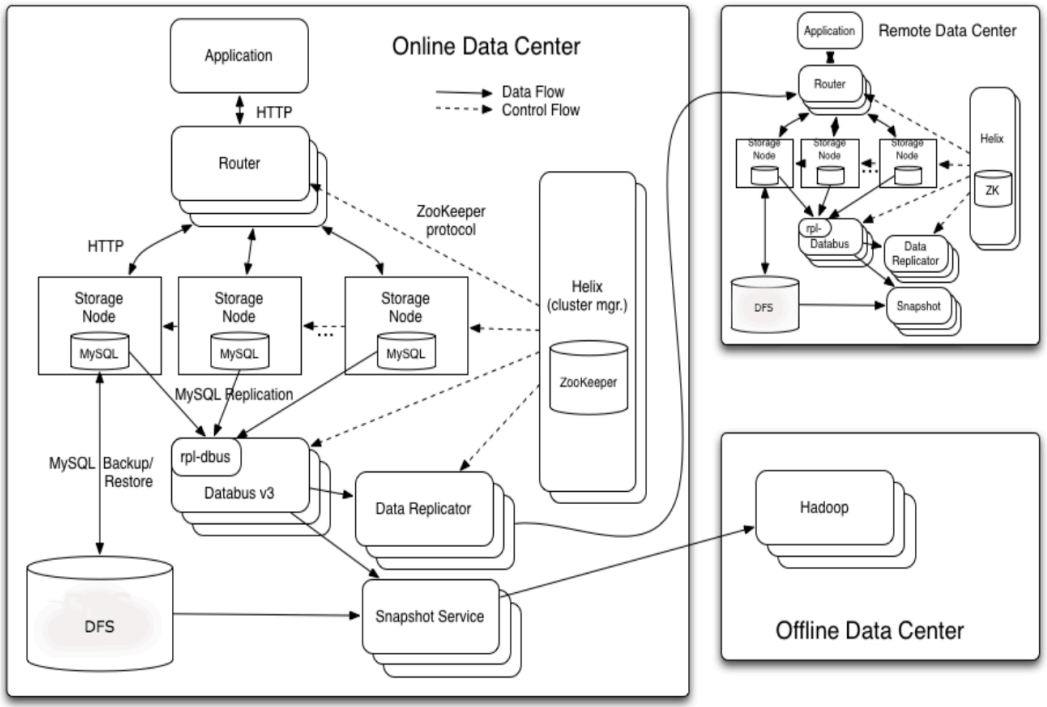
Software Connectors 1

Based on Nenad Medvidovic's slides

Software Connectors

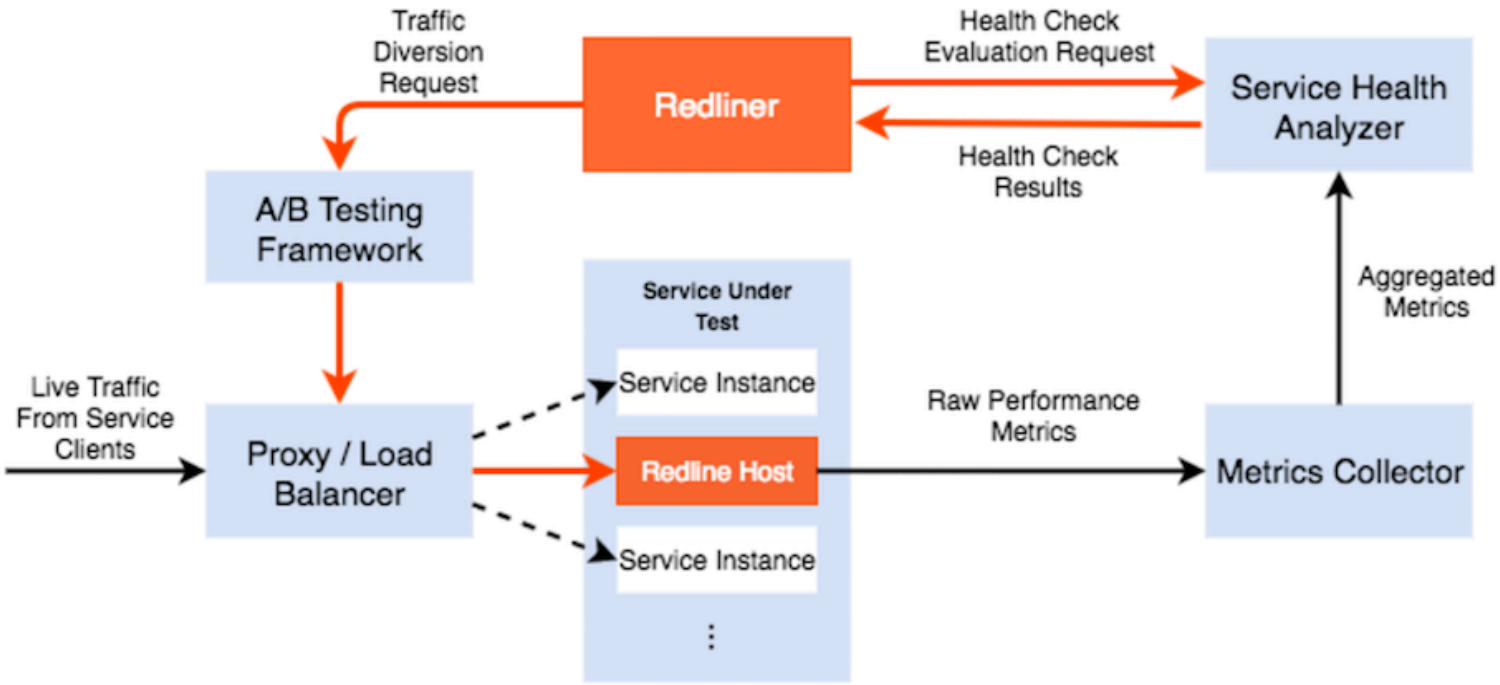
- Key part of Architectures (static, dynamic, deployment views)
- Connect components and define the rules of interaction between components
  - Simple: shared variable access; method calls; …
  - Complex: database access; client-server; scheduler; load balancer
- Connectors provide: Interaction ducts;

LinkedIn Espresso Doc Store



https://engineering.linkedin.com/espresso/introducing-espresso-linkedins-hot-new-distributed-document-store

LinkedIn Redliner



https://engineering.linkedin.com/blog/2017/02/redliner--how-linkedin-determines-the-capacity-limits-of-its-ser

Connectors in Software Arch

- In coding often connectors are implicit
- In software architecture:
  - They are identified and have an identity
  - Capture system interaction (at the level of components)
  - They have a specification that can be complex

What is Different about Connectors?

- Components have application-specific functionality.
- Connectors provide interaction mechanisms that are generic across different applications.
- Interaction may involve multiple components
- Interaction may have a protocol associated to it.

Benefits of Explicit Connectors

- Interaction is defined by the arrangement of the connectors (as far as possible)
- Component interaction is defined by the pattern of connectors in the architecture
- Interaction is “independent” of the components

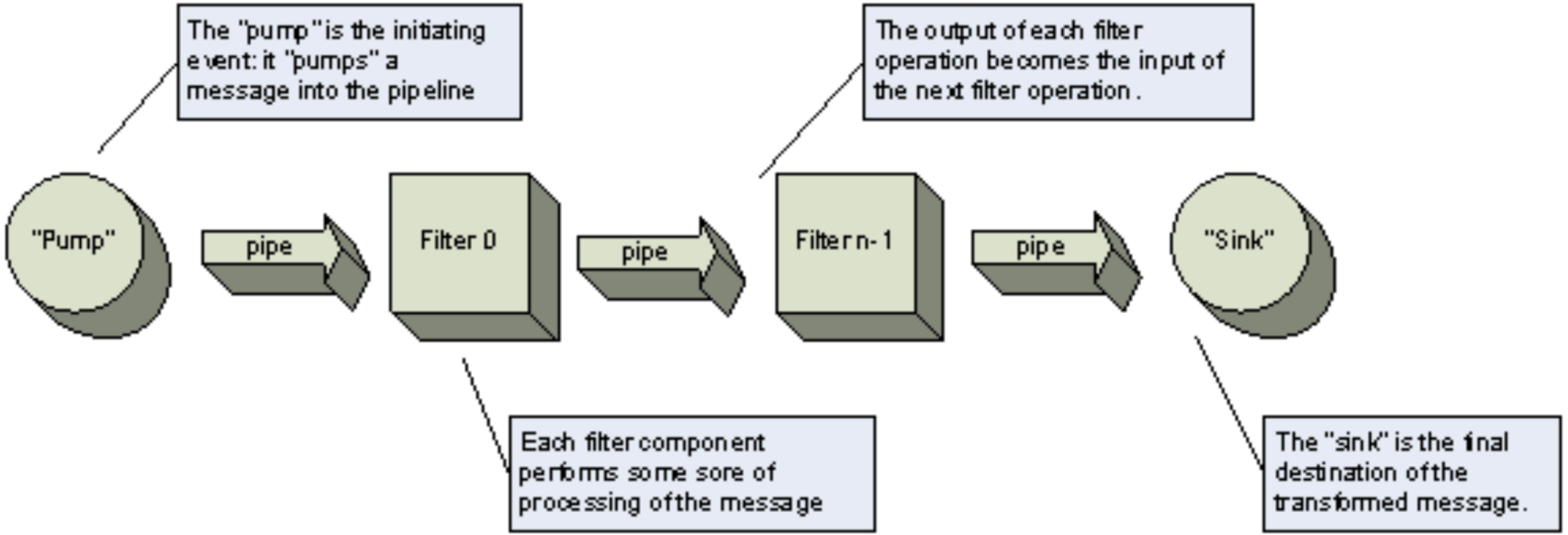
Roles played by Software Connectors

- The specification of the connector protocols determine: the types of interface that it works with; properties of interaction; rules about ordering of interaction; measurable features of interaction.
- The main roles are: Communication; Coordination; Conversion; Facilitation
- Connectors often have multiple roles

Communication

- Information is transmitted between components (e.g. message passing; method call; remote procedure call,…).
- Connectors constrain things:
  - Direction of flow (e.g. pipes)
  - Capacity, rates of flow, etc.
- May have other effects e.g. coordination (e.g. blocking I/O)
- Influences measurable Quality Attributes of the system
- Separates communication from functional aspects (components do the functional part).

Pipes


The "pump" is the initiating event: it "pumps" a message into the pipeline
The output of each filter operation becomes the input of the next filter operation.
"Pump"
pipe
Filter 0
pipe
Filter n-1
pipe
"Sink"
Each filter component performs some sore of processing of the message
The "sink" is the final destination of the transformed message.

Coordination

- Controls the timing relationship of functional aspects of the system
- E.g. coordinating the arrival of data at a collection of components
- Different from communication, conversion or facilitation.
- Some connectors in other classes also include coordination in their definition.

Conversion

- How to get components to interact that don’t have the right means of interaction.
- Incompatibilities might be related to: datatypes, ordering, frequency, structure of parameters etc.
- Examples of types of converters: Wrappers (deal with structural issues), Adaptors (deal with datatype incompatibilities)

Facilitation

- Enable interaction among a group of components that are intended to interact.
- Help manage the interaction
- Examples: load balancer; replication management; redundancy management; scheduler
- Can also relate to coordination e.g. synchronization: critical sections; monitors;

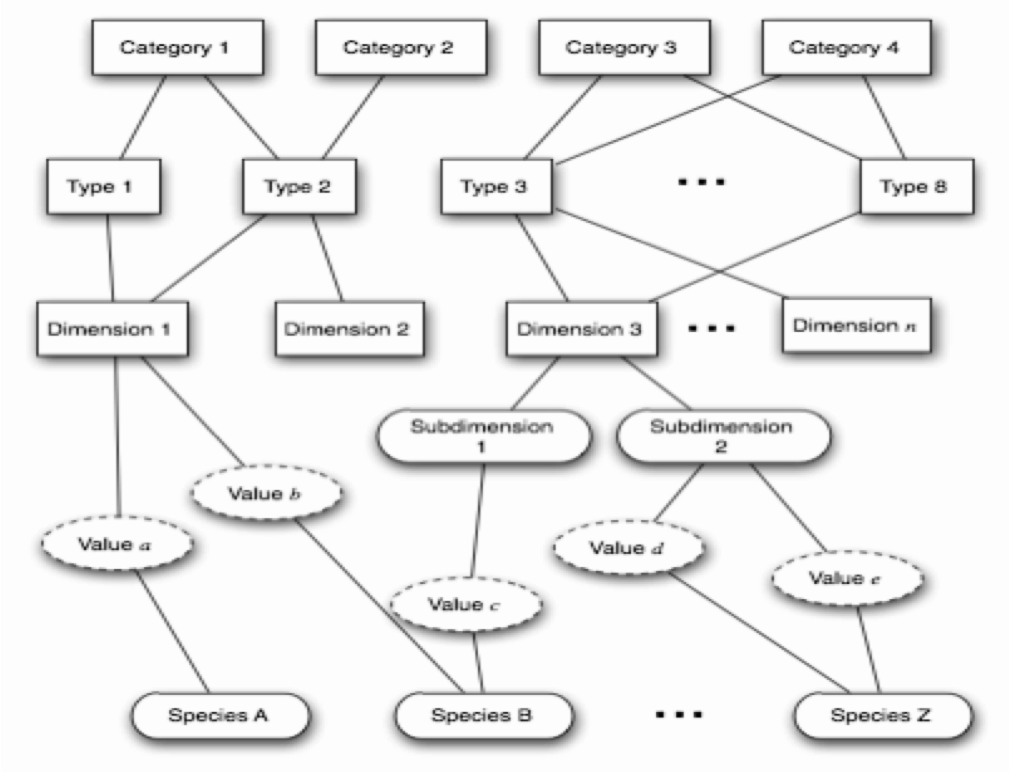
Taylor, Medvidovic and Dashofy

- Have a complex categorization of conectors
- This has some utility since it helps us understand potential different types and uses of connectors.
- We will not go into great depth but it is worth considering briefly

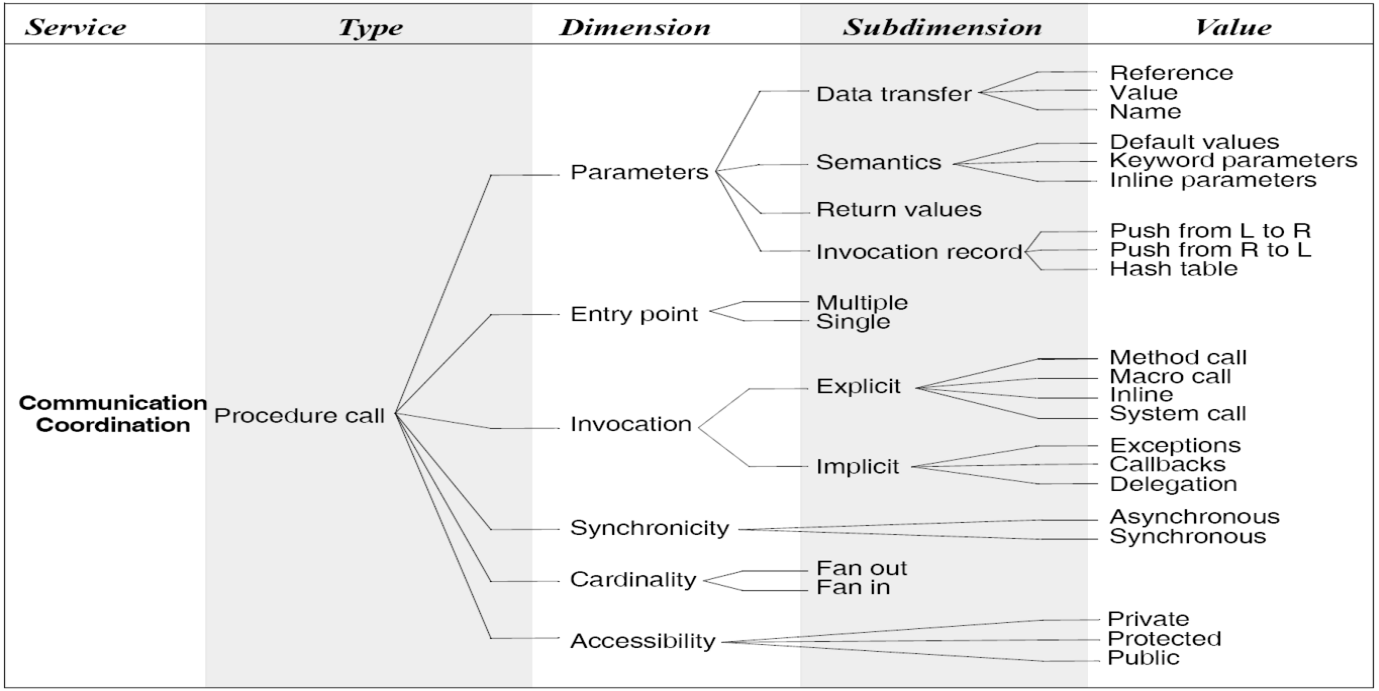
Types of Connector

- Method/Procedure call
- Data access
- Events
- Stream
- Distributor
- Arbitrator
- Adaptor

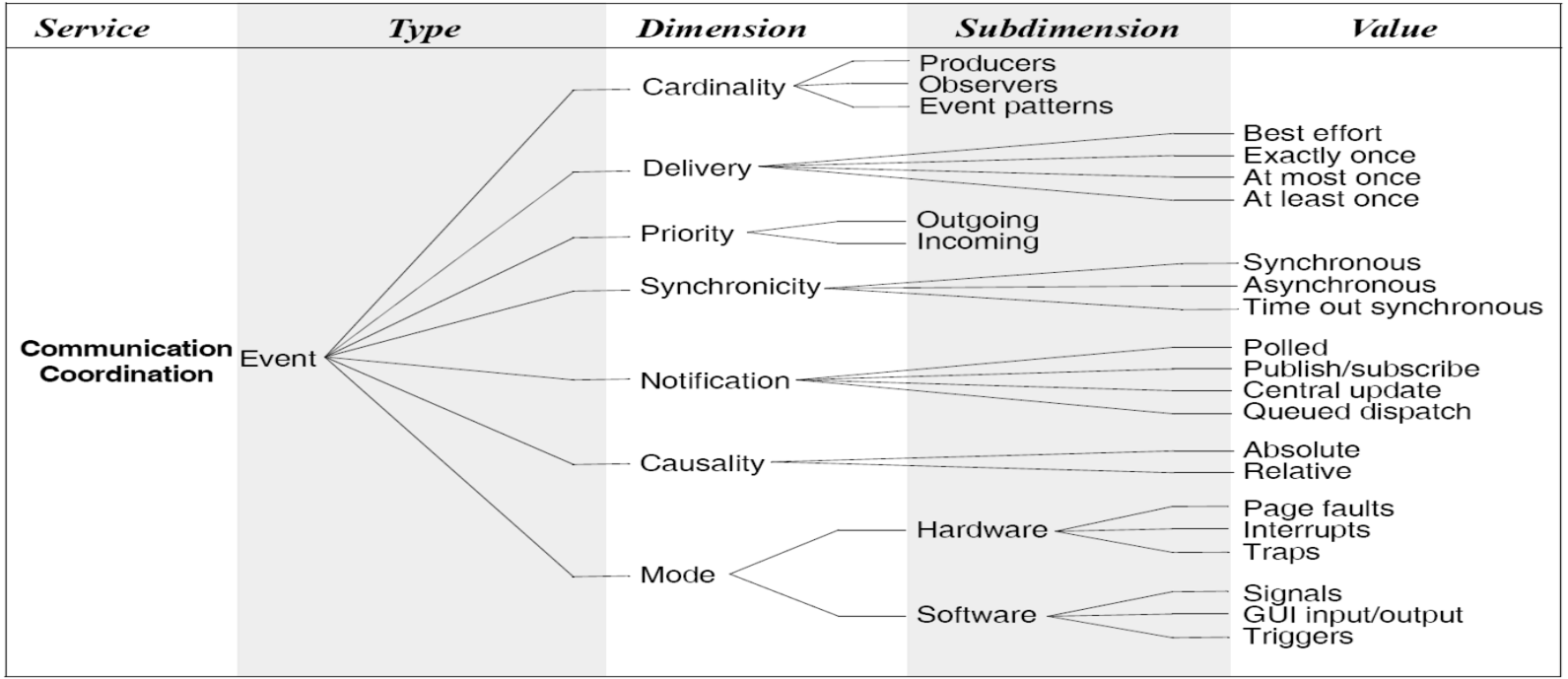
A Framework for Classifying



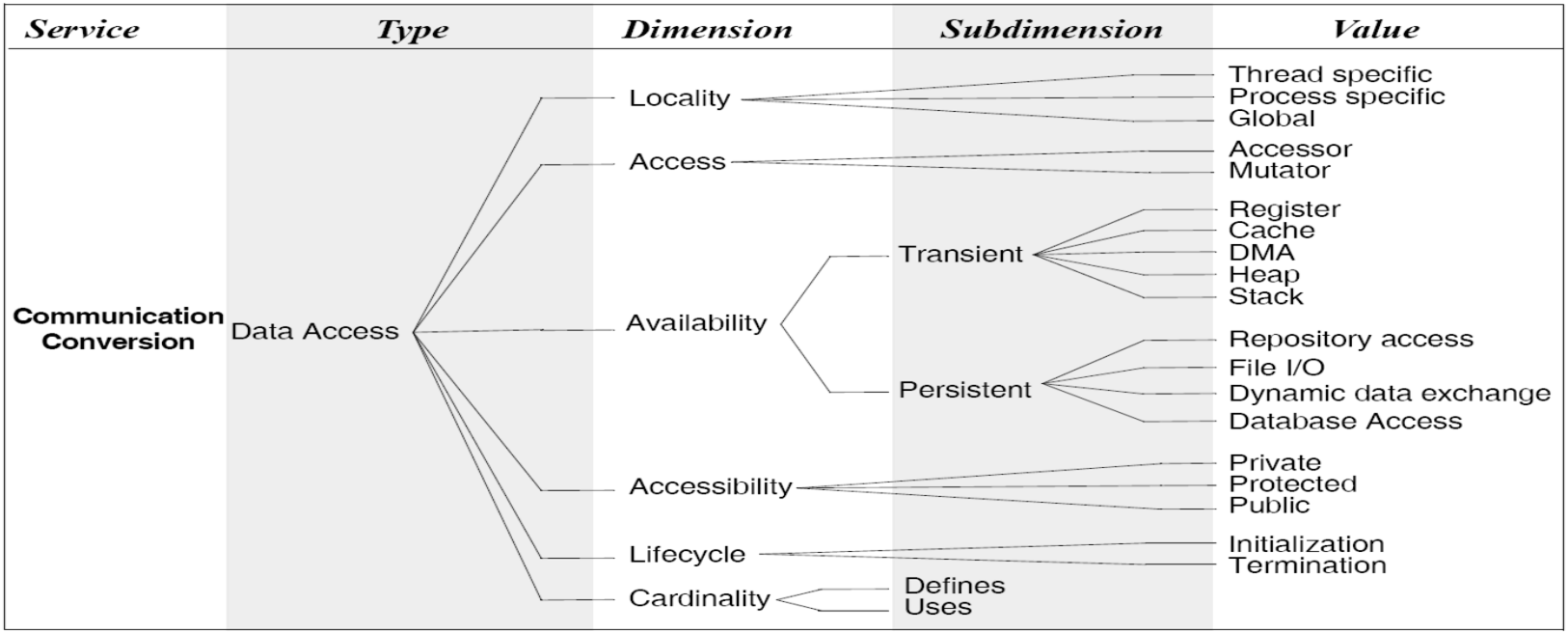
Procedure Call Connectors



Event Connectors

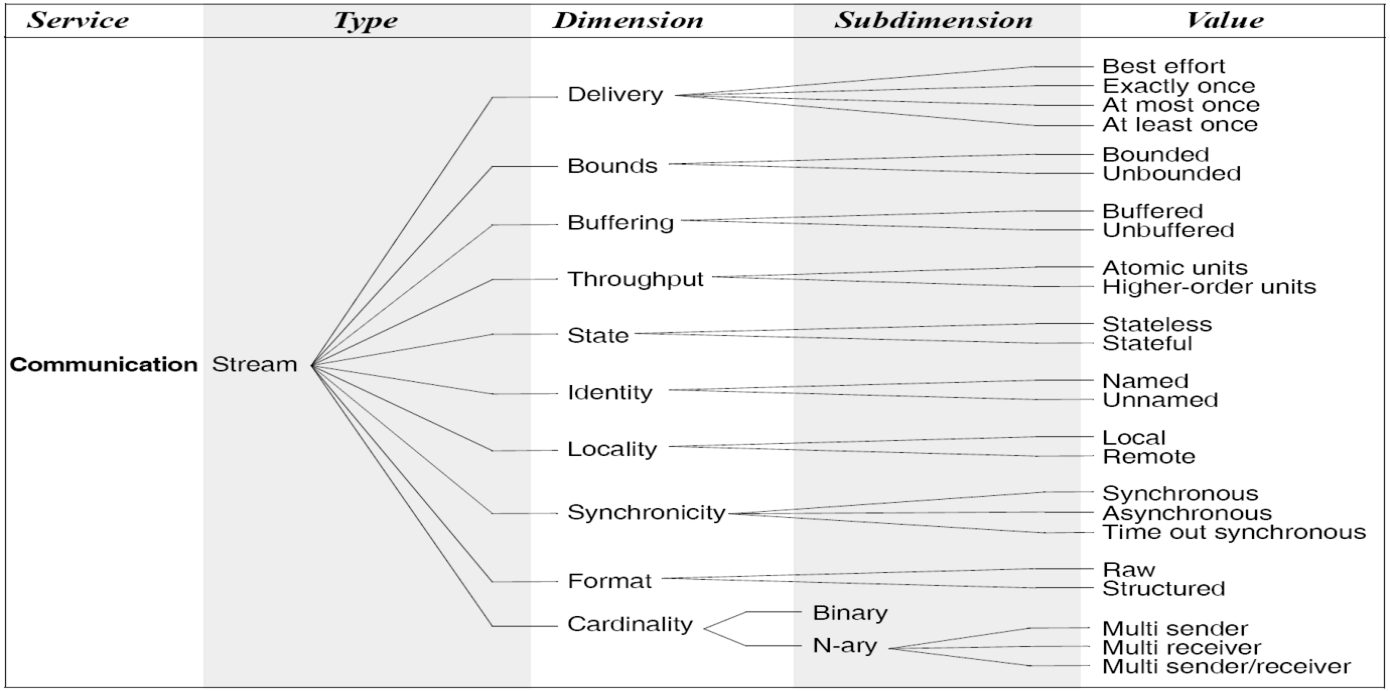


Data Access Connectors

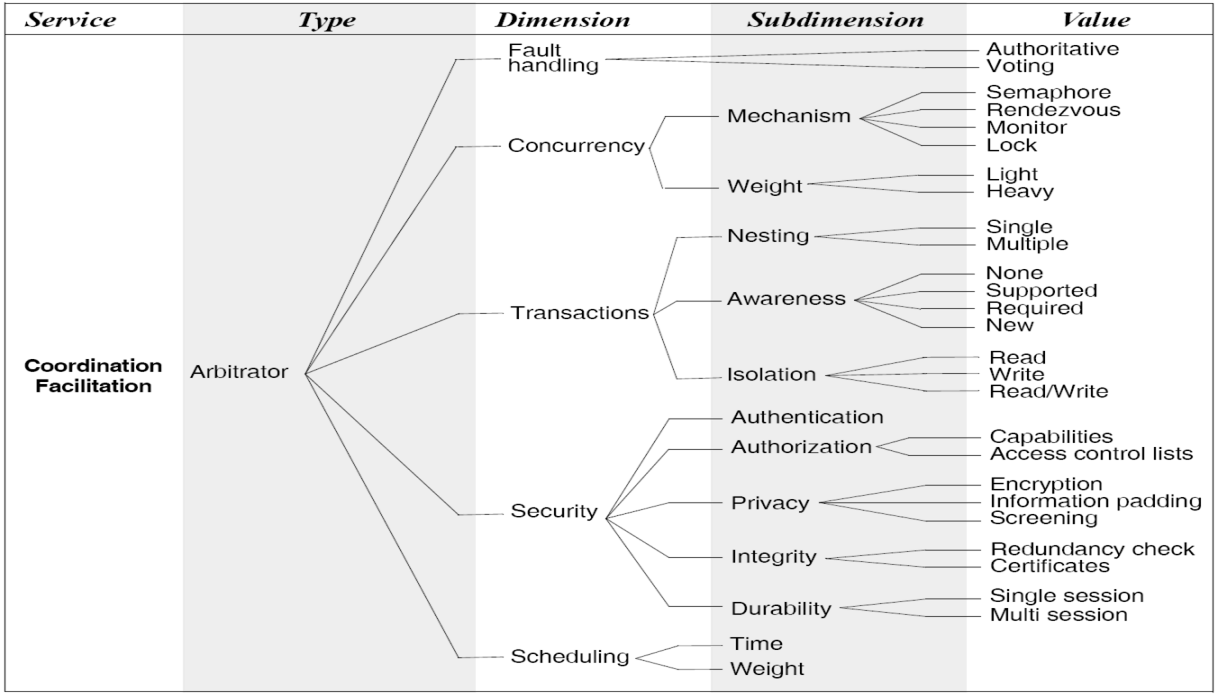


Software Architecture: Foundations, Theory, and Practice; Richard N. Taylor, Nenad Medvidovic, and Eric M. Dashofy; © 2008 John Wiley & Sons, Inc. Reprinted with permission.

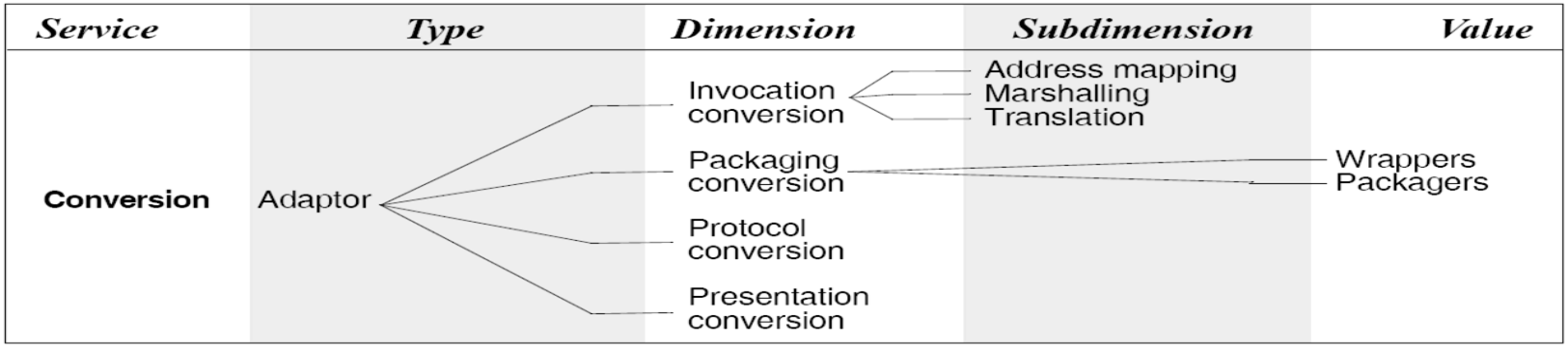
Stream Connectors



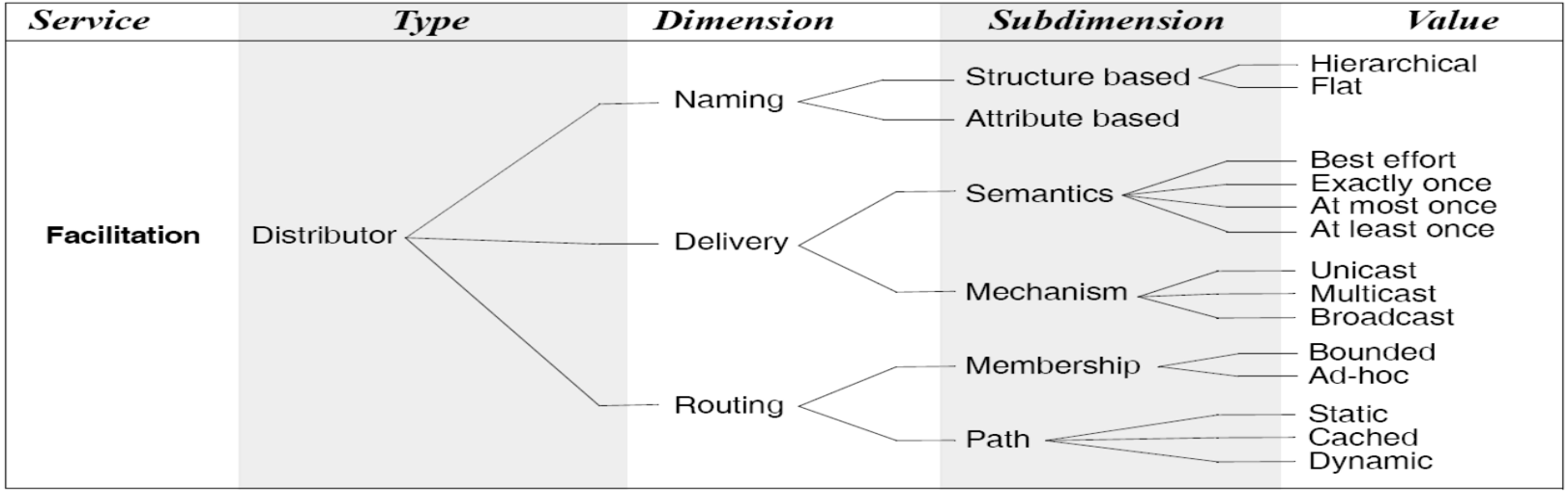
Arbitrator Connectors



Adaptor Connectors



Distributor Connectors



Discussion

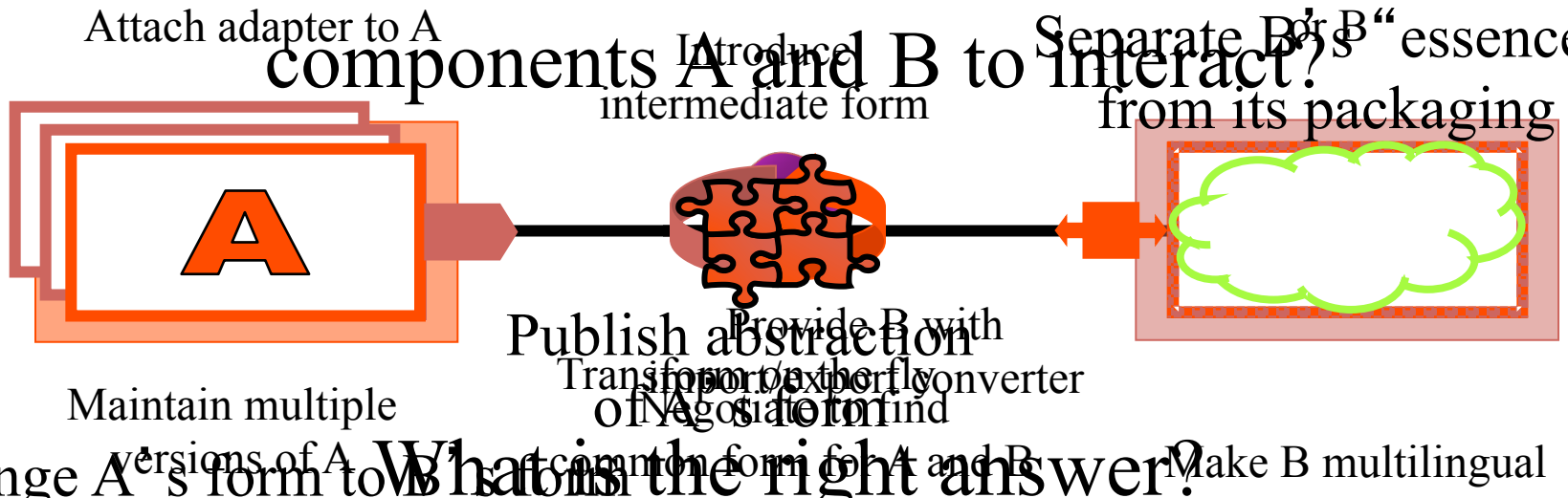
- Identifying connectors makes the interactions in the system explicit – allows reasoning, modelling.
- Defines the context a component operates in:
  - Places requirements on the component.
  - Flexible connectors that can be adapted easily give us freedom to substitute components with somewhat different interaction capabilities.
- Having an explicit structure allows us to consider replace some or all of the connectors:
  - Aids system evolution – replacing deprecated connectors with connectors that are fully supported.
  - May not require changes in component functionality and leave system functionality unchanged

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

Next time: Choosing Software Connectors

How do we enable components A and B to interact?



Publish abstraction of A's form

Change A's form to B's

What is the right answer?

Borrowed from Nenad Medvidovic's slides