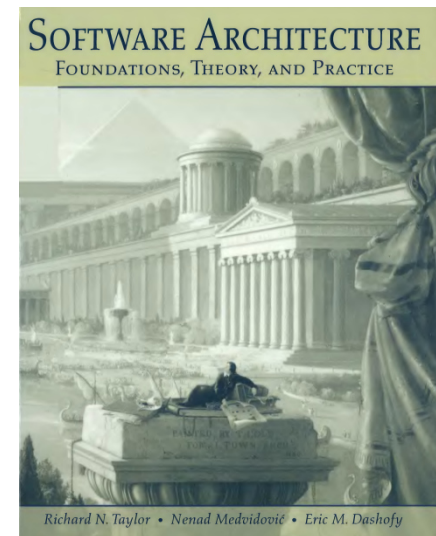


Connettori Software

Rif. Cap. 6 –Richard N. Taylor, Nenad Medvidovic, and Eric M. Dashofy

Corso PSSS- Prof. Fasolino - 2020



Outline

- Generalità sui Connettori
- Ruoli
- Classificazione dei Connettori


Premessa

- Progettare Dati e Funzionalità di un sistema è indubbiamente importante, ma ...
- Progettare l'interazione fra i componenti e la loro efficace integrazione lo è altrettanto!
- Questo compito è svolto da **Connettori** che:
 - Realizzano il passaggio di controllo e di dati fra componenti, ma
 - Possono occuparsi anche di servizi, come:
 - Persistenza, invocazione, scambio di messaggi, transazioni
 - Questi servizi sono in genere considerati facility di base ed offerti automaticamente da un middleware (come CORBA, DCOM, RMI), ma occorre pensare esplicitamente ai componenti che li realizzano
 - Per permettere il riuso di connettori, e per supportare lo sviluppo basato sull'integrazione di componenti eterogenei

Premessa

- Esistono diversi tipi di connettore, con caratteristiche e comportamenti diversi.
- Ad es. un connettore può semplicemente inviare un messaggio ad uno specifico componente, oppure...
- Può inviare in broadcast la notifica di un evento a più componenti sconosciuti
- Può richiedere al componente mittente di sospendersi, finchè il destinatario non abbia ricevuto il messaggio, oppure...il mittente può procedere senza aspettare
- Può inoltrare i messaggi nell'ordine di arrivo... oppure ordinarli, filtrarli, combinarli etc...
-

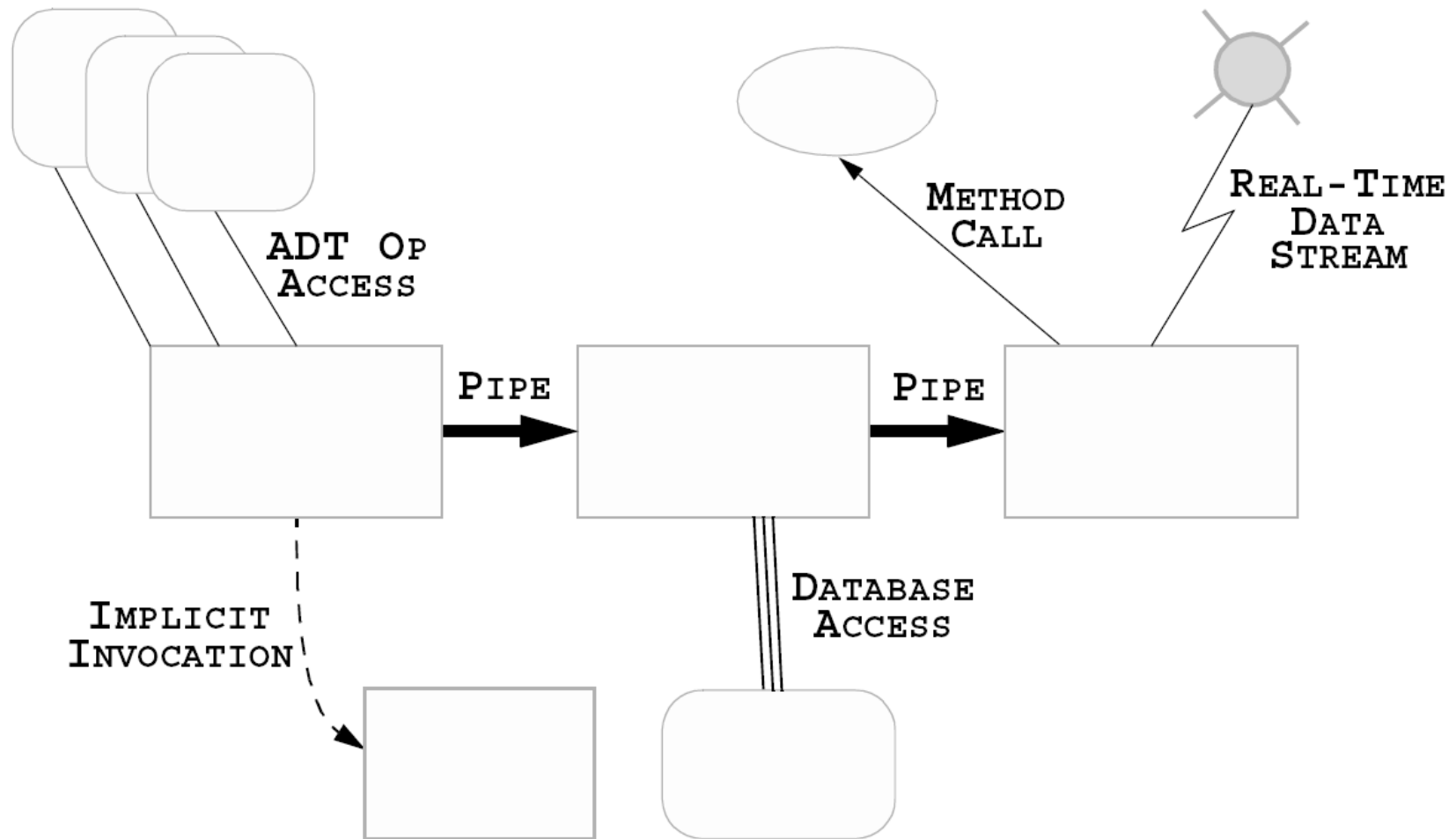
Progettare i Connettori

- L'architetto avrà anche i seguenti compiti:
- Comprendere I bisogni di interazione fra componenti
- Identificare attentamente tutti gli attributi dell'interazione
- Selezionare I connettori candidati
- Verificare I trade-off associati ai vari candidati 
- Analizzare le proprietà della combinazione componente-connettore risultante

Che cosa sono i Connettori Software?

- Elementi architetturali che modellano:
 - Le **interazioni** fra componenti
 - **Regole** che governano queste interazioni
- Le interazioni possono essere di tipi diversi:
- Interazioni semplici
 - Procedure calls
 - Shared variable access
- Interazioni Complesse & Ricche semanticamente
 - Protocolli Client-server
 - Protocolli di accesso a Database
 - Asynchronous event multicast
- Ciascun connettore fornisce
 - Canali per l'interazione
 - Trasferimento di controllo e/o di dati

Dove sono i Connettori nei sistemi Software?



Implemented vs. Conceptual Connectors

- Nell'implementazione software, i connettori sono spesso “invisibili” o trascurati:
 - Non hanno codice dedicato ad essi in maniera esclusiva
 - Non hanno identità
 - Non corrispondono a nessuna unità di compilazione
 - Nelle implementazioni distribuite:
 - Sono sparsi attraverso più moduli
 - O realizzati con diversi meccanismi di interazione

Implemented vs. Conceptual Connectors (cont'd)

- I Connettori nelle architetture software invece:
 - Sono entità di Prima Classe
 - Hanno identità propria
 - Descrivono tutte le interazioni nel sistema
 - Hanno diritto a proprie specifiche e astrazioni

Ragioni per trattare i Connettori indipendentemente

- **Connettori \neq Componenti**
 - I Componenti forniscono funzionalità *specifiche per l'applicazione* (application-specific functionality)
 - I Connettori forniscono meccanismi di interazione *indipendenti dall'applicazione*
- Permettono di astrarre sull'interazione e/o sulla parametrizzazione
- Permettono di specificare adeguatamente le interazioni
 - Binary vs. N-ary
 - Asymmetric vs. Symmetric
 - Interaction protocols

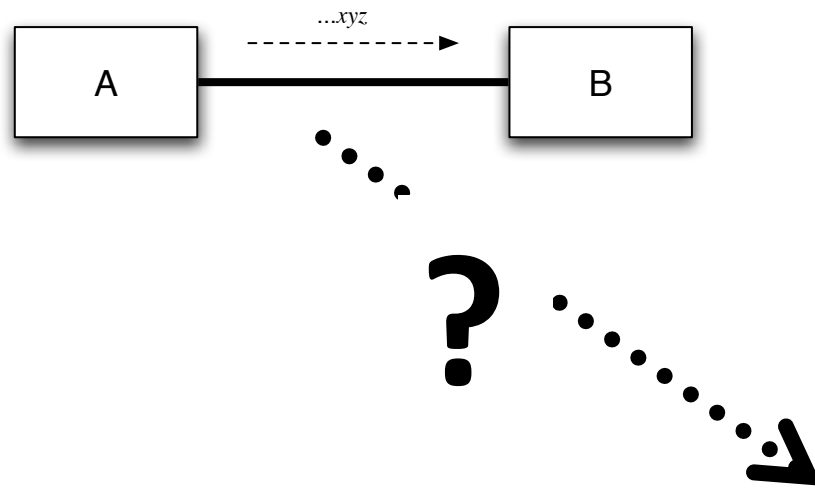
Ragioni per trattare I Connettori indipendentemente (continua)

- Localizzazione della definizione dell'interazione
- Specificano informazioni sul sistema (ossia l'interazione) extra rispetto ai componenti
- Consentono Indipendenza dei Componenti
- Maggiore flessibilità delle interazioni fra componenti

Benefici nel trattare i Connettori esplicitamente

- Tenere separata l'elaborazione dall'interazione
- Minimizzano le interdipendenze fra componenti
- Supportano meglio l'evoluzione del sistema
 - At component-, connector-, & system-level
- Possono supportare dinamicità
- Facilitano l'eterogeneità
- Diventano punti di distribuzione
- Aiutano l'analisi ed il testing dei sistemi software

Necessità di connettori espliciti: un esempio



In questo esempio, A e B sono connessi mediante una pipe monodirezionale.

Proprietà della pipe:

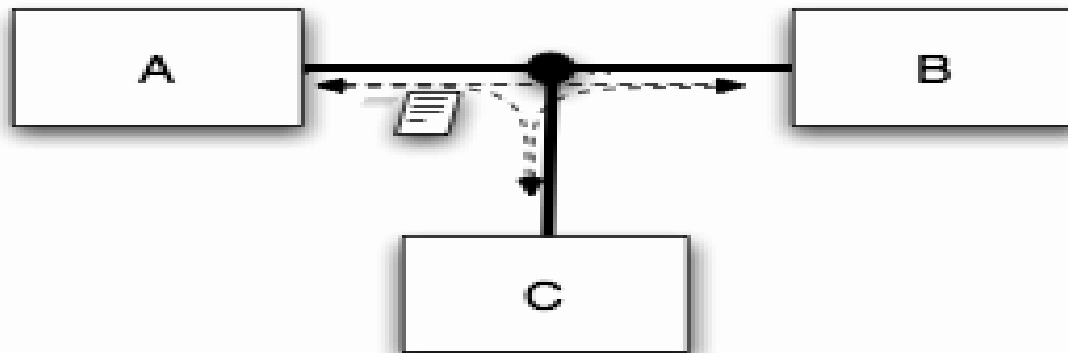
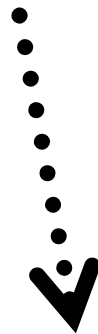
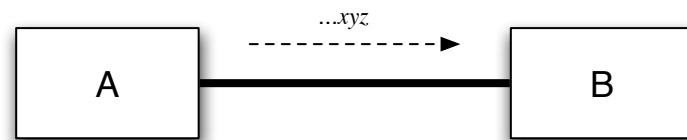
La pipe ha cardinalità 1-1.

A non sa nulla di B.

La pipe non è bufferizzata, ossia se A scrive lo stream e B non lo preleva, lo stream verrà perso con la successiva scrittura.


In alternativa, si può aggiungere una seconda pipe (per la comunicazione da B ad A), oppure si può introdurre una pipe bufferizzata).

Un altro esempio



Se è necessario spedire dati strutturati, il connettore più adatto ed efficiente non sarà una pipe, ma un **Bus di Eventi** (v. eventbus in Java). Il bus è sempre monodirezionale, ma può avere cardinalità 1-n. Può consentire di bufferizzare i dati e di aggiungere più componenti a runtime.

I Ruoli dei Connettori Software

- Sono il luogo dell'interazione fra insiemi di componenti
- Specifica del protocollo (talora implicita) che definisce le sue proprietà
 - Tipi di interfacce che esso è capace di mediare
 - Assicurazione sulle proprietà delle interazioni
 - Regole sull'ordine delle interazioni
 - Impegni sulle interazioni (es. performance)
- Quattro possibili Ruoli dei Connettori:
 - Comunicazione
 - Coordinazione
 - Conversione 
 - Facilitazione (agevolazioni)



1. Connectors as Communicators

- È il ruolo principale dei Connettori: consentire lo scambio di informazioni fra componenti (es. dati, messaggi, risultati)
 - Supportano differenti meccanismi di comunicazione
 - es. procedure call, RPC, shared data access, message passing
 - Vincolano la struttura su cui avviene la comunicazione e la direzione della comunicazione
 - Es. pipes
 - Vincoli sulla qualità del servizio
 - Es. Persistenza
- Servono a separare *l'elaborazione dalla comunicazione*
- Possono influenzare caratteristiche non funzionali del sistema
 - Es. performance, scalabilità, sicurezza

2. Connectors as Coordinators

- Supportano il trasferimento del flusso di controllo fra componenti
 - Il flusso di controllo passa fra i vari componenti con vari meccanismi
 - Esempi: function call, o invocazione di metodi
- Servono a separare *il controllo dall'elaborazione*
- N.B. Elementi di controllo possono essere presenti anche nella comunicazione, conversione e facilitazione

3. Connectors as Converters

- Consentono l'interazione anche di componenti incompatibili, sviluppati indipendentemente.
 - Necessari quando un componente ritiene di poter interagire con un altro componente in modo diverso da quello previsto dall'altro
- Le possibili incompatibilità dell'interazione riguardano:
 - Tipo
 - Numero
 - Frequenza
 - Ordine
- Esempi di convertitori
 - Adattatori (di formato) 
 - Wrappers 

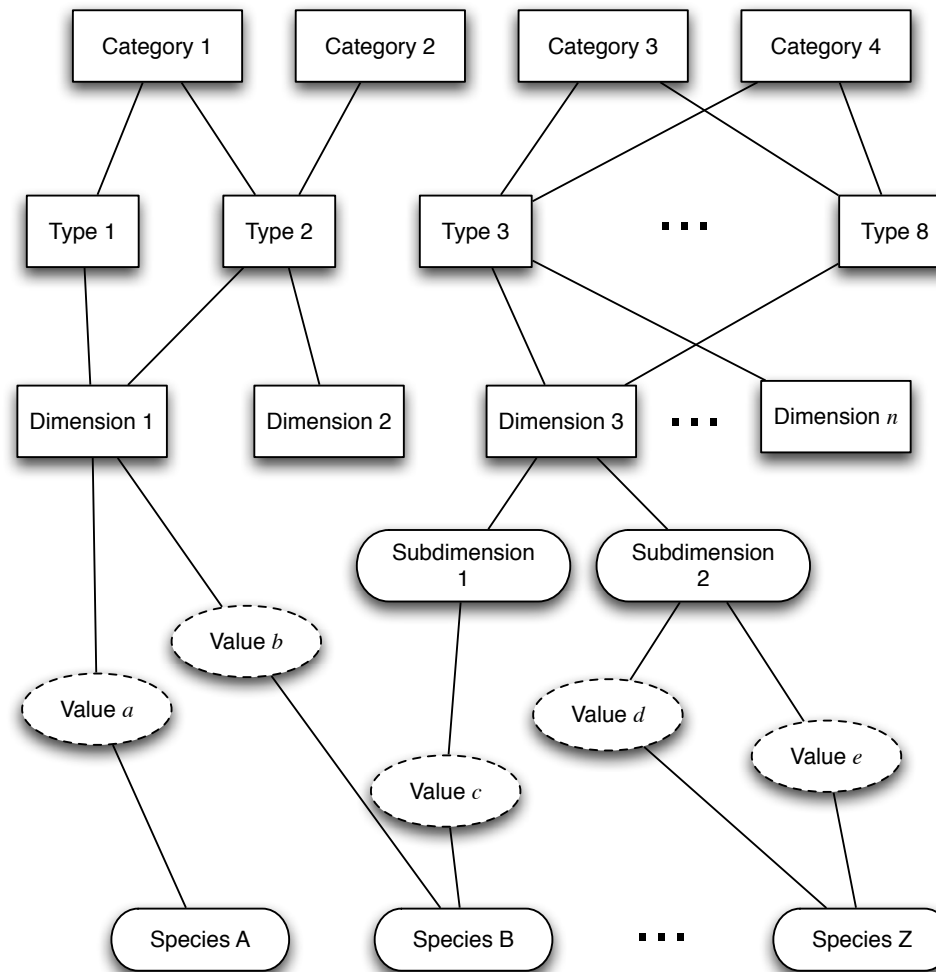
4. Connectors as Facilitators

- I connettori offrono servizi per la facilitazione e ottimizzazione dell'interazione,
 - Es. load balancing, scheduling di servizi, controllo della concorrenza
- Abilitano l'interazione dei componenti interagenti
 - Mediazione e serializzazione dei dati
- Governano l'accesso a informazioni condivise:
 - Scheduling di servizi
- Assicurano adeguati profili prestazionali:
 - e.g., load balancing
- Forniscono servizi di sincronizzazione
 - Critical sections
 - Monitors

Tipi di Connettori

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

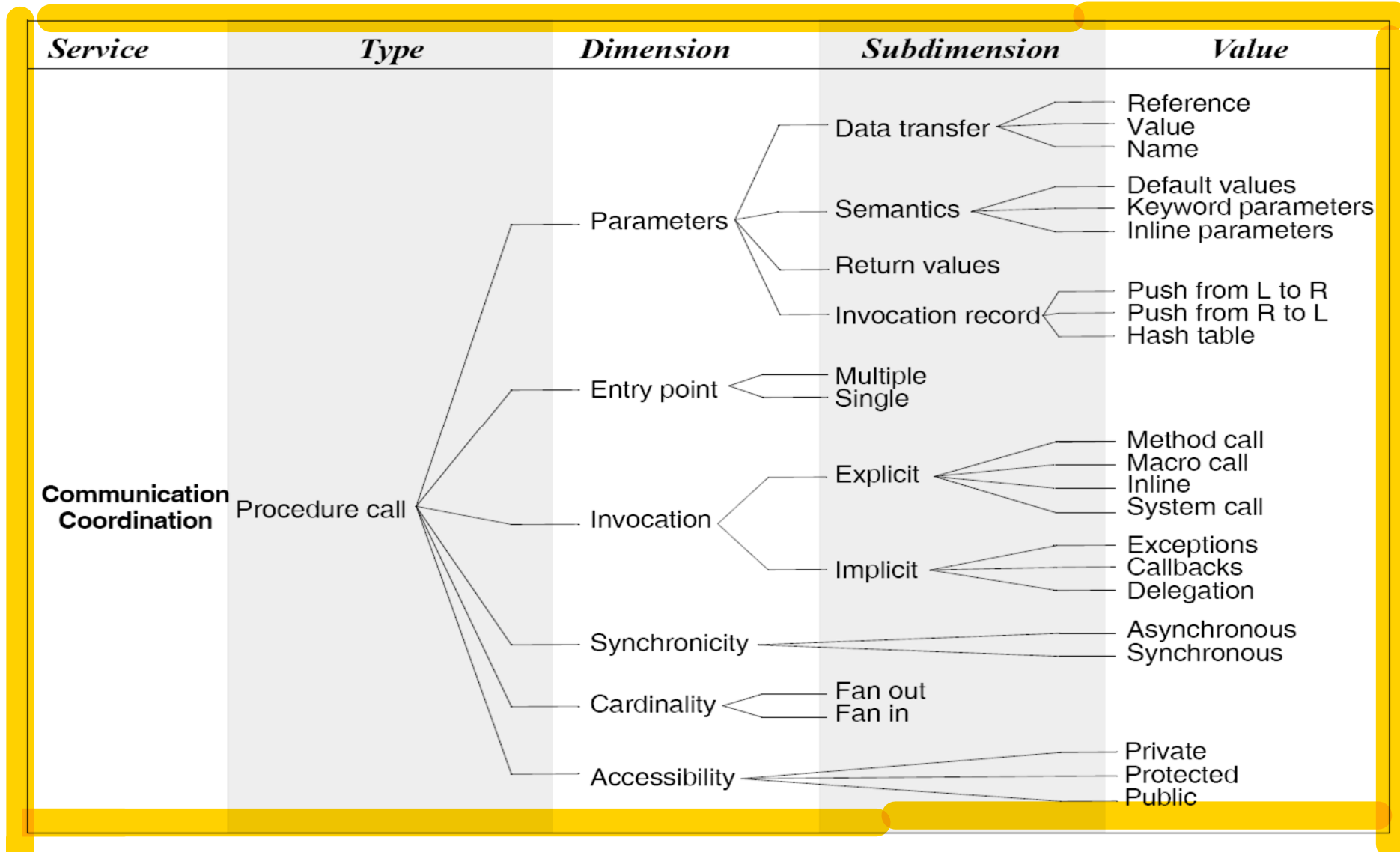
Uno schema per la classificazione di Connettori



Procedure Call Connector

- Modellano il passaggio del flusso di controllo fra vari componenti attraverso vari meccanismi.
- Esempi:
- Object oriented methods call
- Exec e fork in Unix
- Operating System Call
- Call Back nei sistemi ad eventi

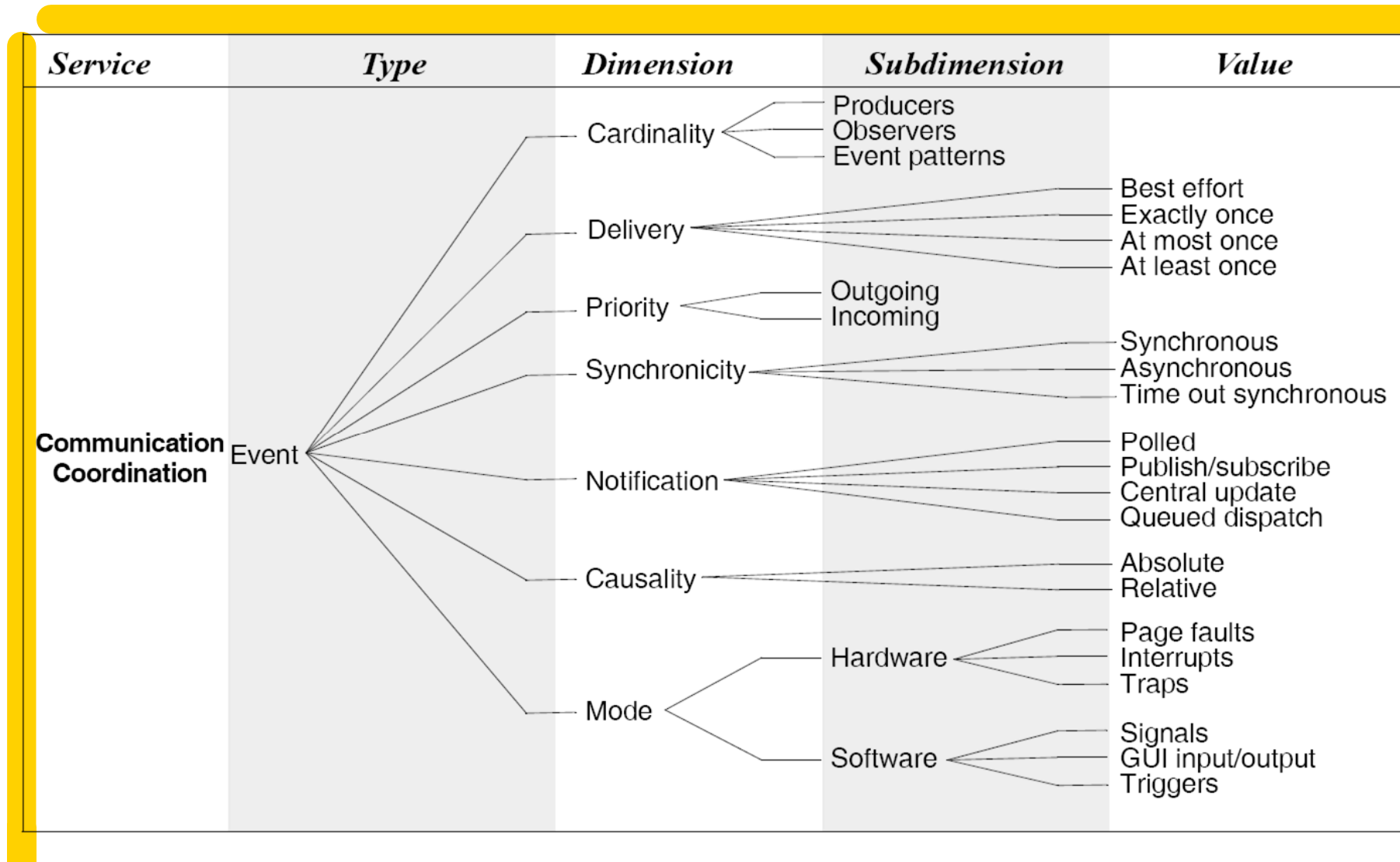
Procedure Call Connectors



Event connectors

- Sono simili ai connettori di procedure call perchè gestiscono il passaggio del flusso di controllo fra componenti (*ruolo di coordinamento*).
 - In questo caso il flusso è governato da un evento.
- Quando il connettore sa dell'occorrenza di un evento, genera notifiche (messaggi) a tutte le parti interessate e sposta il controllo ai componenti che gestiscono quei messaggi.
 - I messaggi possono essere generati al verificarsi di un singolo evento, oppure di uno specifico pattern di eventi.
 - I contenuti di un evento possono essere strutturati (es. Tempo e luogo dell'evento, ..)
 - In questo modo, I connettori di evento forniscono anche *servizi di comunicazione*.

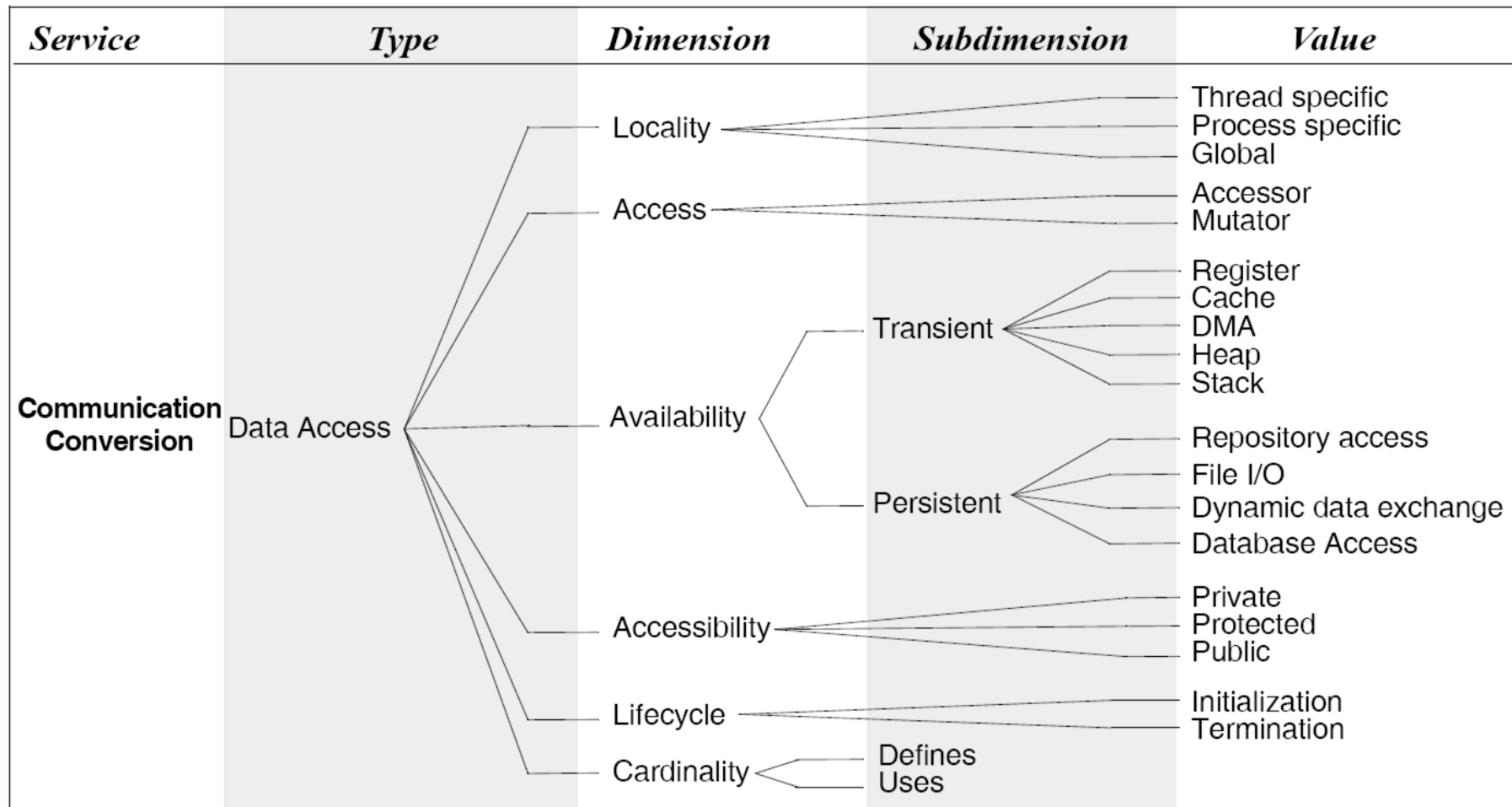
Event Connectors



Data Access Connectors

- Permettono di accedere a dati mantenuti in un componente di tipo data store.
- Il connettore può variare, in funzione della persistenza o meno dei dati acceduti.
 - Es. *accesso a dati persistenti*: query, come query SQL per accesso a database, o accesso a dati contenuti in un repository.
 - Es. di *accesso a dati transienti*: accesso a pile o code e caching di informazioni in memoria.
- Esempi: un connettore può permettere un accesso ai dati di tipo globale, in sola lettura o anche in scrittura, con cardinalità diverse (un solo produttore di dati e più lettori), etc..

Data Access Connectors



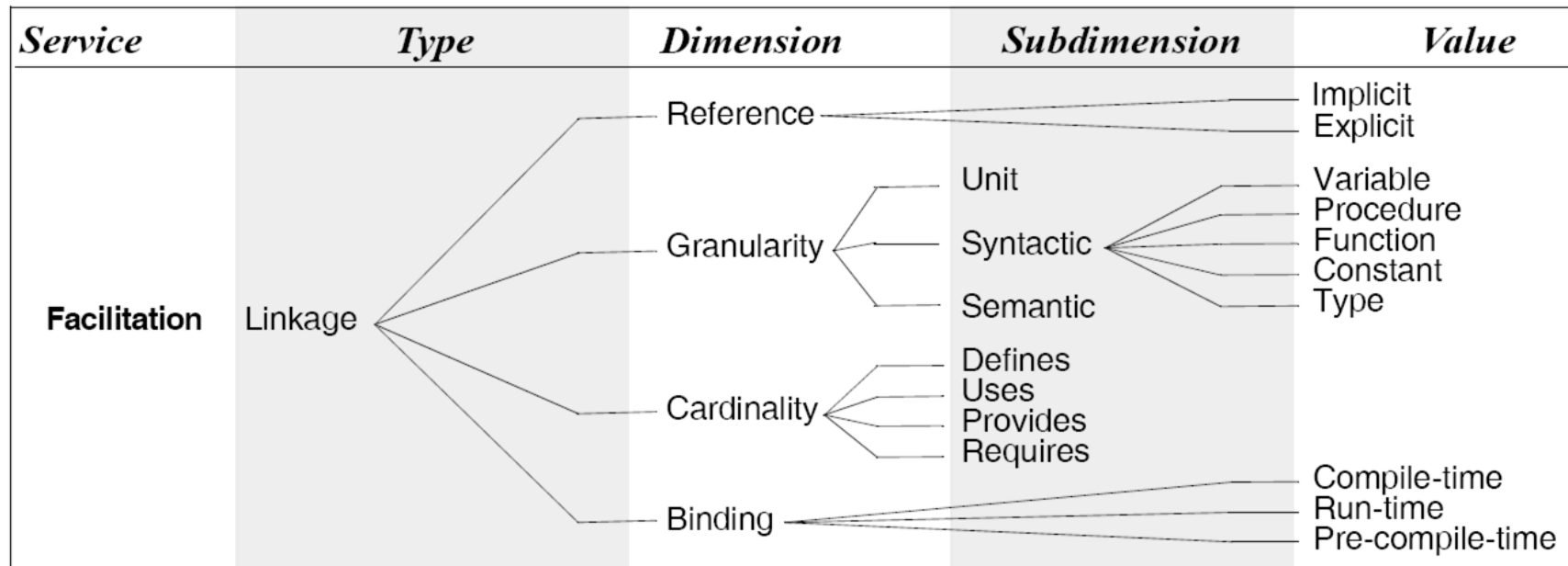
Connettori di Linkage

- Forniscono servizi di facilitazione, per connettere insieme componenti diversi durante il funzionamento (es. componenti e bus in architetture C2) , oppure moduli, oggetti, o file nel build di un sistema.
- *Creano un canale di comunicazione* tra tali componenti.
- Possono avere diversa granularità (in base a dimensione del componente e livello di dettaglio richiesto per effettuare il collegamento) :
 - Unità: si specificano solo le unità da connettere
 - Sintattico: si specificano anche i componenti interni da mettere in corrispondenza fra le unità connesse (es. procedure, funzioni, costanti, variabili, etc.)
 - Semantico: specifica anche le regole dell'interazione fra le unità (i protocolli)

Connettori di Linkage

- La cardinalità del connettore indica il numero di posti in cui una risorsa (es. procedura, variabile) può essere definita, usata, fornita, o richiesta.
- Il binding può avvenire prima della compilazione, in compilazione o a runtime.
- Es. il programma *make*, per *system building*, nei sistemi di *Configuration management*

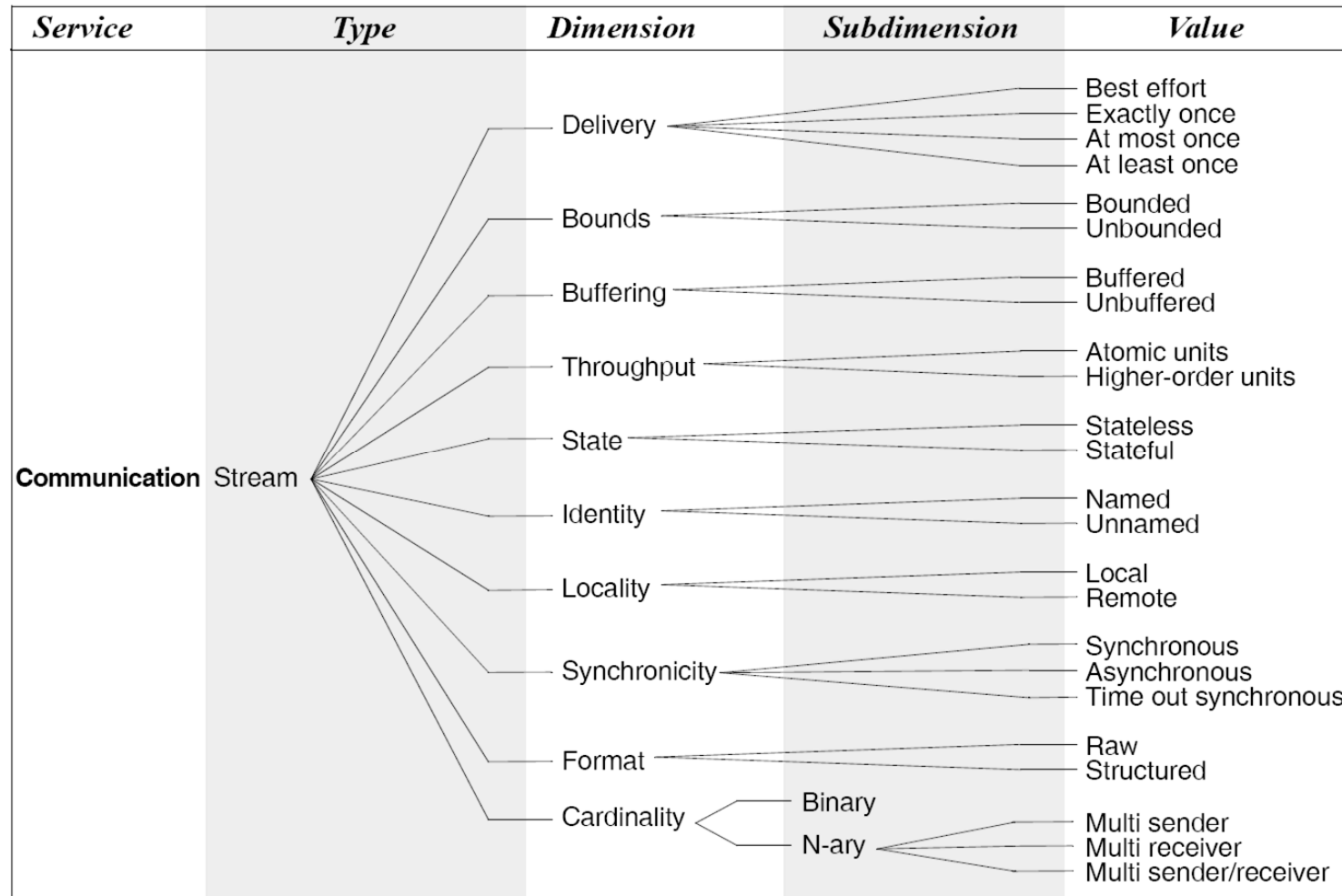
Linkage Connectors



Connettori di stream

- Gli Stream sono usati per trasferire grandi quantità di dati fra processi autonomi (*ruolo di comunicazione*).
- Si usano anche in sistemi client-server, con protocolli di data transfer (come FTP, UDP, UDT,...), per consegnare i risultati di una elaborazione.
- Possono essere combinati con altri tipi di connettori, per fornire connettori compositi: es. Possono combinare quelli di data access per eseguire accessi a database e file storage, e connettori di eventi per la consegna multipla di un grande numero di eventi.
- **Esempi:** Unix pipes, TCP/UDP communication sockets, e protocolli client/server proprietari.

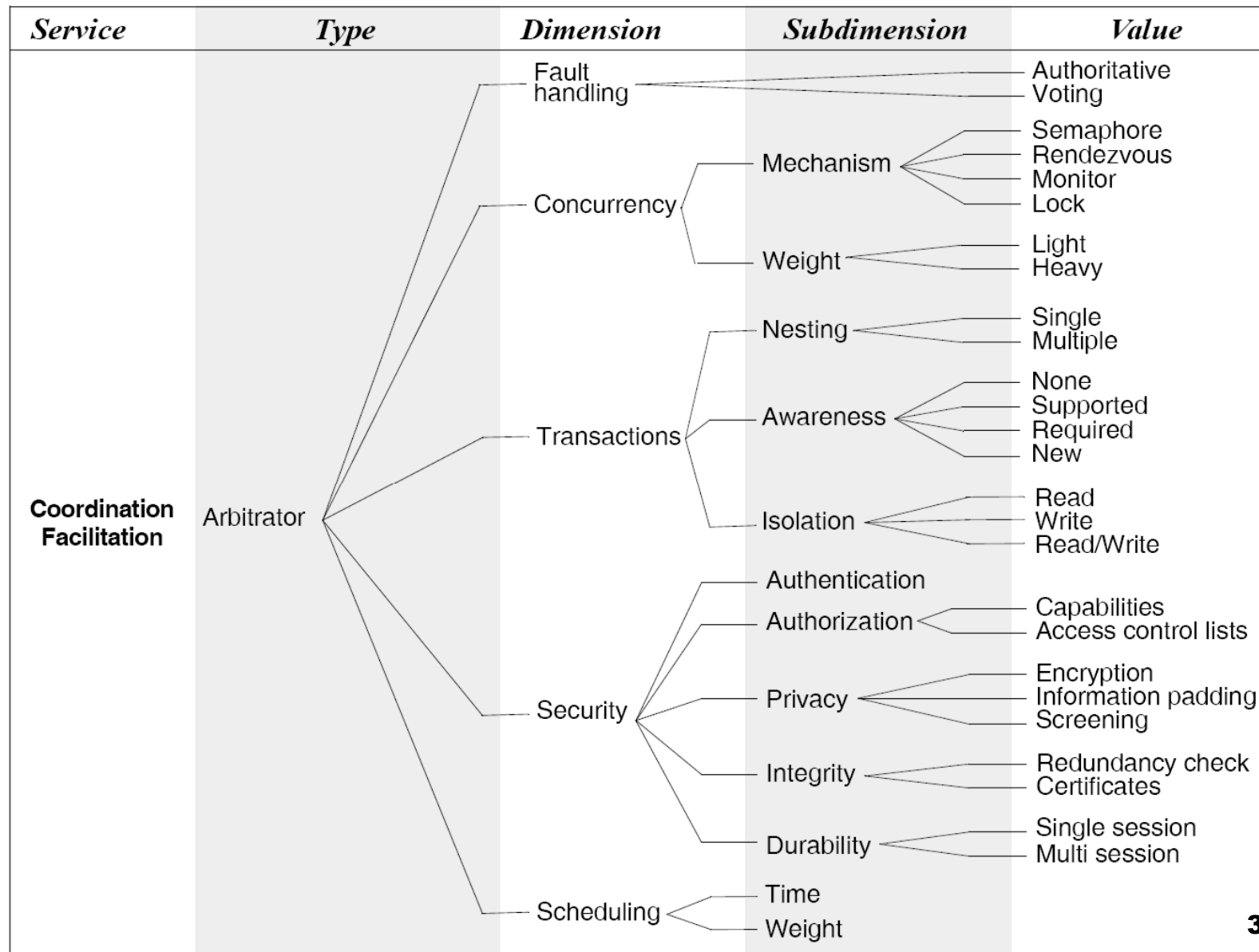
Stream Connectors



Connettori di Arbitraggio

- Quando più componenti sono contestualmente presenti, i connettori possono sequenziare le chiamate, risolvere conflitti (funzione di facilitazione), passare il flusso di controllo (coordinamento)...
- Per esempio, in sistemi multithreaded, con accesso a memoria condivisa, possono usare controlli di *sincronizzazione e concorrenza* per garantire consistenza ed atomicità delle operazioni.
- Possono offrire facilities per *negoziare / service level* e per mediare interazioni che richiedono garanzie di affidabilità e atomicità. Forniscono anche servizi di *scheduling e bilanciamento del carico*.
- Possono supportare la dependability (reliability, safety, security).

Arbitrator Connectors



Adaptor Connectors

- Supportano l'interazione fra componenti non progettati per l'interazione, o da ambienti eterogenei (per linguaggio di programmazione o piattaforma).
- Forniscono servizi di conversione, relativi ai protocolli di interazione o alle politiche di comunicazione.
- Possono anche servire per ottimizzare le interazioni fra componenti, scegliendo dinamicamente la modalità di interazione (es. fra RPC e chiamata locale)

Adaptor Connectors

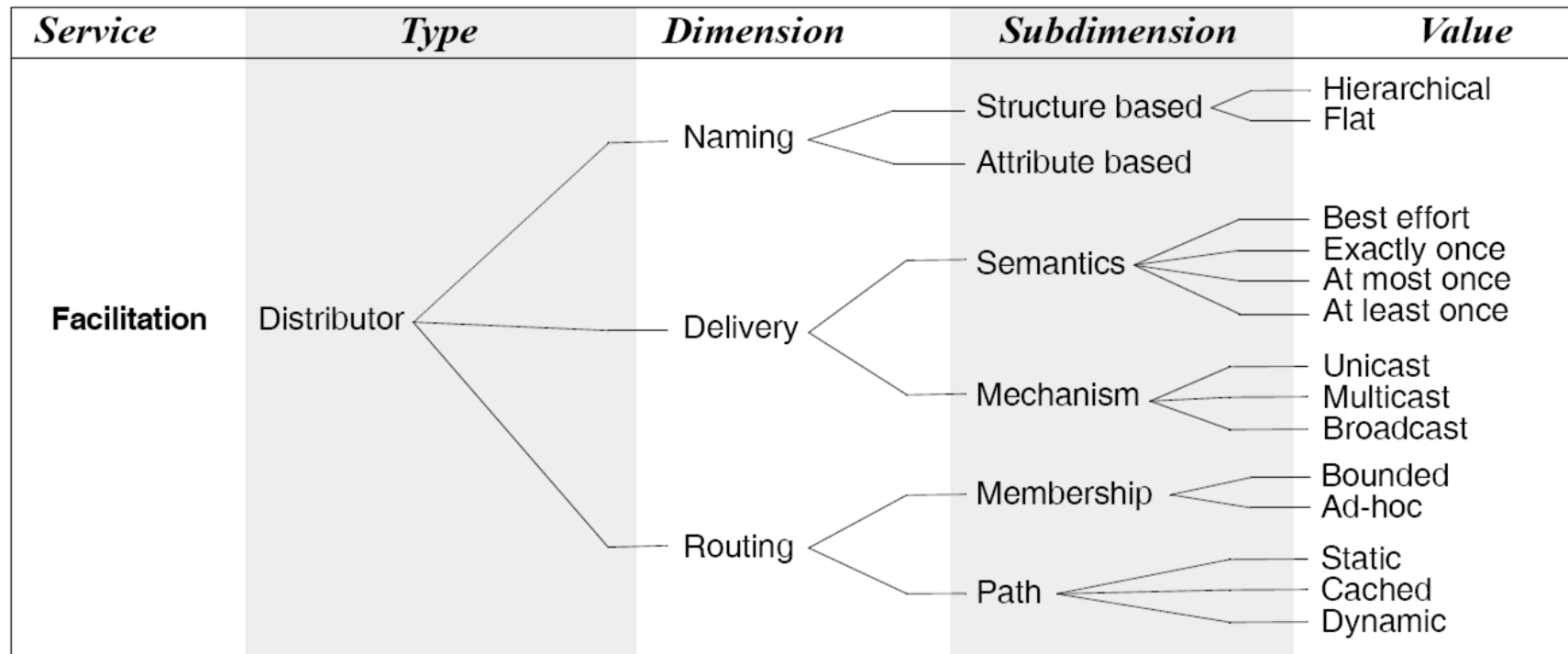
<i>Service</i>	<i>Type</i>	<i>Dimension</i>	<i>Subdimension</i>	<i>Value</i>
Conversion	Adaptor	Invocation conversion Packaging conversion Protocol conversion Presentation conversion	Address mapping Marshalling Translation Wrappers Packagers	



Distributor Connectors

- Identificano i percorsi di interazione e di conseguenza instradano informazioni (sia di comunicazione che di coordinamento) fra i componenti lungo tali percorsi.
- Non esistono mai da soli, ma sono al servizio di altri (come connettori di streams e procedure call).
- Es.: I sistemi distribuiti usano questi connettori per realizzare il flusso dei dati. Li usano per identificare la posizione di componenti e i percorsi per arrivarvi, a partire dal loro nome simbolico.
- Esempi: Domain name service (DNS), routing, switching, e molti altri servizi di rete.

Distributor Connectors



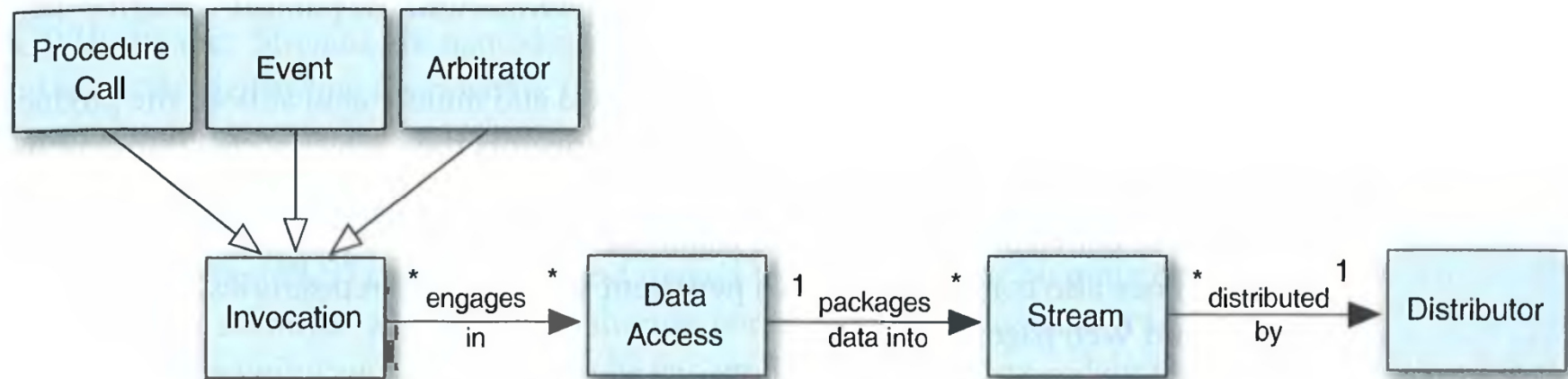
Discussione

- I connettori permettono di modellare interazioni complesse.
- La flessibilità dei connettori aiuta l'evoluzione del sistema
 - Grazie ad aggiunta, rimozione, sostituzione, riconnessione o migrazione di componenti
- Lo scambio fra connettori diversi può servire per:
 - Supportare l'evoluzione del sistema
 - Non impattare sulla funzionalità del sistema
- A livello implementativo, librerie di implementazioni di connettori permettono di concentrarsi su altri aspetti specifici dell'applicazione.

Esempi di connettori di distribuzione compositi

- Oggi la distribuzione di contenuti su Internet si basa su diversi tipi di connettori di distribuzione compositi:
 - event-based, client-server, peer-to-peer,...
- Questi connettori :
 - Eseguono accesso a dati (che prevedono letture di stream e impacchettamento di dati)
 - Distribuzione a vari end-users
 - Possono essere invocati o mediante procedure calls (es. nel client... server-based) o attraverso eventi (for example, event..based), o arbitration (for example, P2P..based).

Connettori di distribuzione compositi



Esempio: Connettori di distribuzione event-based

- Sono composti da quattro tipi di connettore: evento, data access, data stream, e distributore
- These connectors send and receive data through asynchronous notifications called *events*.
 - Events may arrive according to some fixed periodic schedule or at discrete aperiodic intervals.
- Typically, there are many producers and consumers of data in event..based distribution connectors.
- Event..based distribution connectors often employ an asynchronous best..effort delivery method for data, making no guarantees as to the completion or time of data deliveries.

- Data delivery events sent through the connectors can be prioritized.
 - Events can be delivered using prioritized threaded queues,
 - be locally or remotely managed,
 - or be delivered using user registered preferences and the publish..subscribe delivery mechanism.
- Event..based distribution connectors themselves access transient and persistent data, both public and private.

- They also communicate with persistent stores such as repositories, databases, file systems, and Web pages.
- Stream..based data is distributed from data producers to data consumers using a naming registry, via either a hierarchical or flat naming model.
- Data streams are delivered as events using best effort delivery and any combination of the unicast, broadcast, and multicast delivery mechanisms.