Diseño de Componentes

Objetivos

- Estudiar los principales estilos arquitectónicos utilizados en arquitecturas de componentes
- Estudiar los principales tipos de conectores

Estilos Arquitectónicos

Estilos

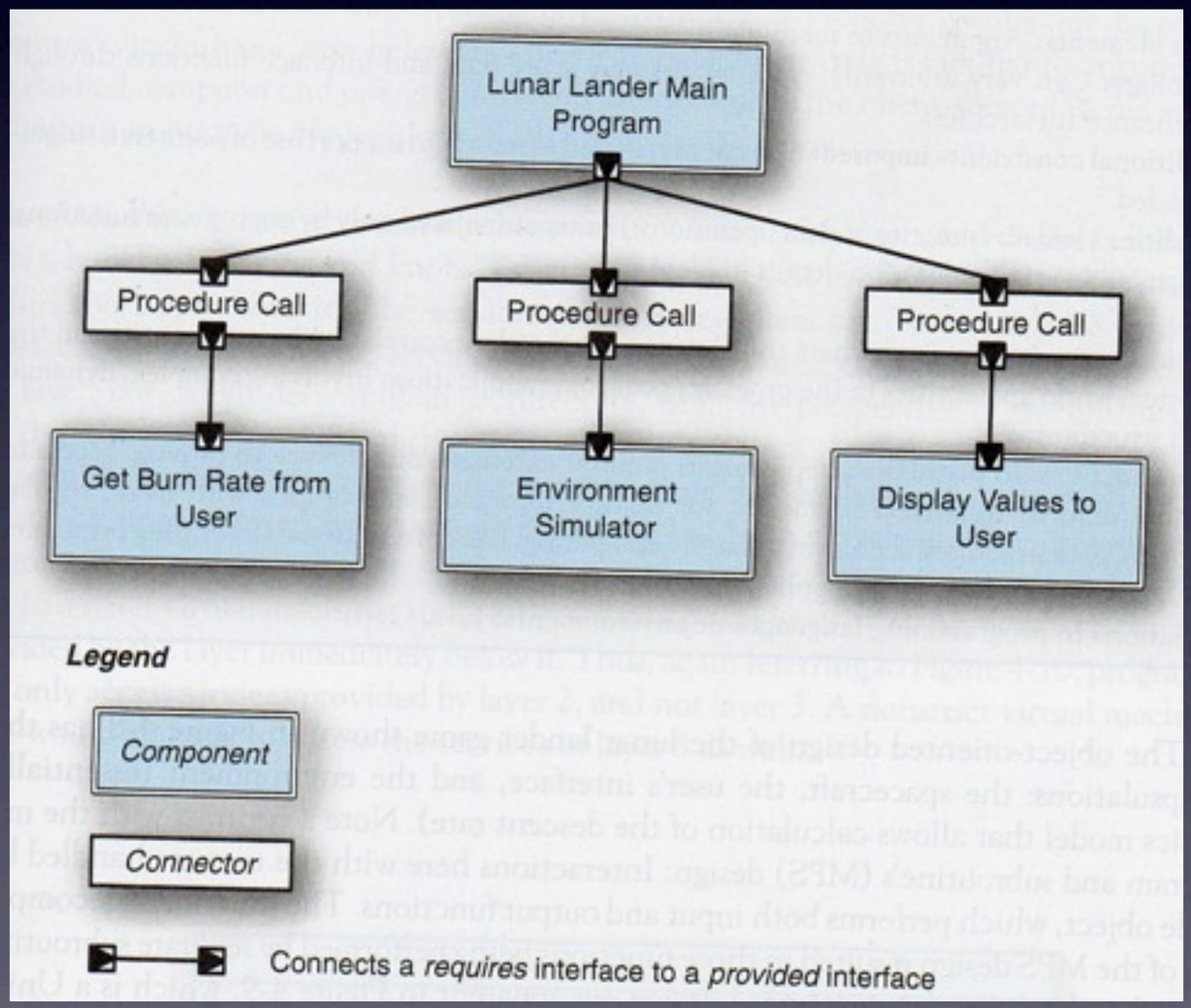
- Main program and subroutines

Summary: Decomposition based upon separation of functional processing steps.
Components: Main program and subroutines.
Connectors: Function/procedure calls.
Data elements: Values passed in/out of subroutines.
Topology: Static organization of components is hierarchical; full structure is a directed graph.
Additional constraints imposed: None.
Qualities yielded: Modularity: Subroutines may be replaced with different implementations as long as interface semantics are unaffected.

Typical uses: Small programs; pedagogical purposes.
Cautions: Typically fails to scale to large applications; inadequate attention to data structures. Unpredictable effort required to accommodate new requirements.
Relations to programming languages or environments: Traditional imperative programming languages, such as BASIC, Pascal, or C.

Estilos

- Main program and subroutines



Estilos

- Object-Oriented

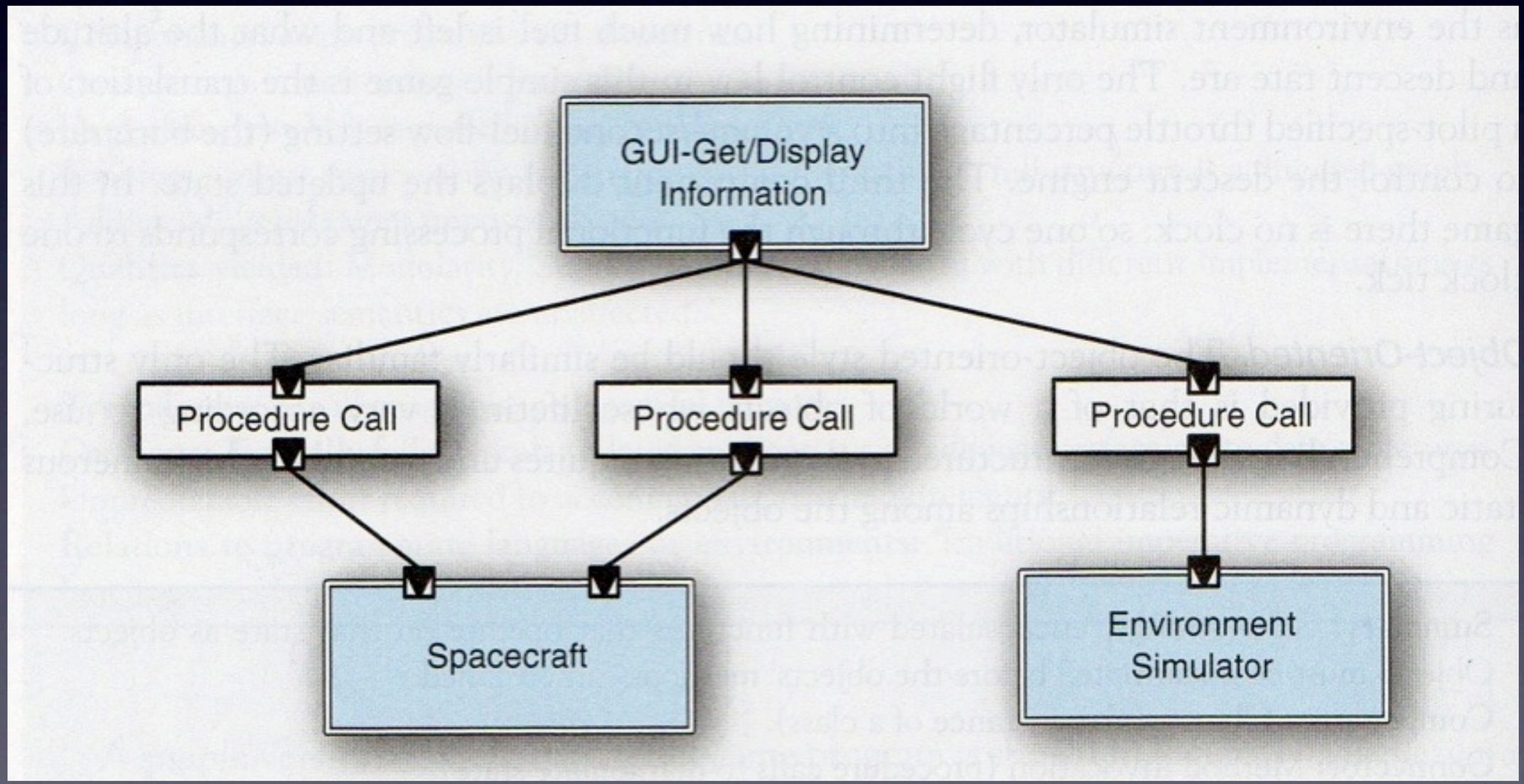
Summary: State strongly encapsulated with functions that operate on that state as objects. Objects must be instantiated before the objects' methods can be called.
Components: Objects (aka. instance of a class).
Connector: Method invocation (procedure calls to manipulate state).

Data elements: Arguments to methods.
Topology: Can vary arbitrarily; components may share data and interface functions through inheritance hierarchies.
Additional constraints imposed: Commonly: shared memory (to support use of pointers), single-threaded.
Qualities yielded: Integrity of data operations: data manipulated only by appropriate functions. Abstraction: implementation details hidden.

Typical uses: Applications where the designer wants a close correlation between entities in the physical world and entities in the program; pedagogy; applications involving complex, dynamic data structures.
Cautions: Use in distributed applications requires extensive middleware to provide access to remote objects. Relatively inefficient for high-performance applications with large, regular numeric data structures, such as in scientific computing. Lack of additional structuring principles can result in highly complex applications.
Relations to programming languages or environments: Java, C++.

Estilos

- Object-Oriented



Estilos

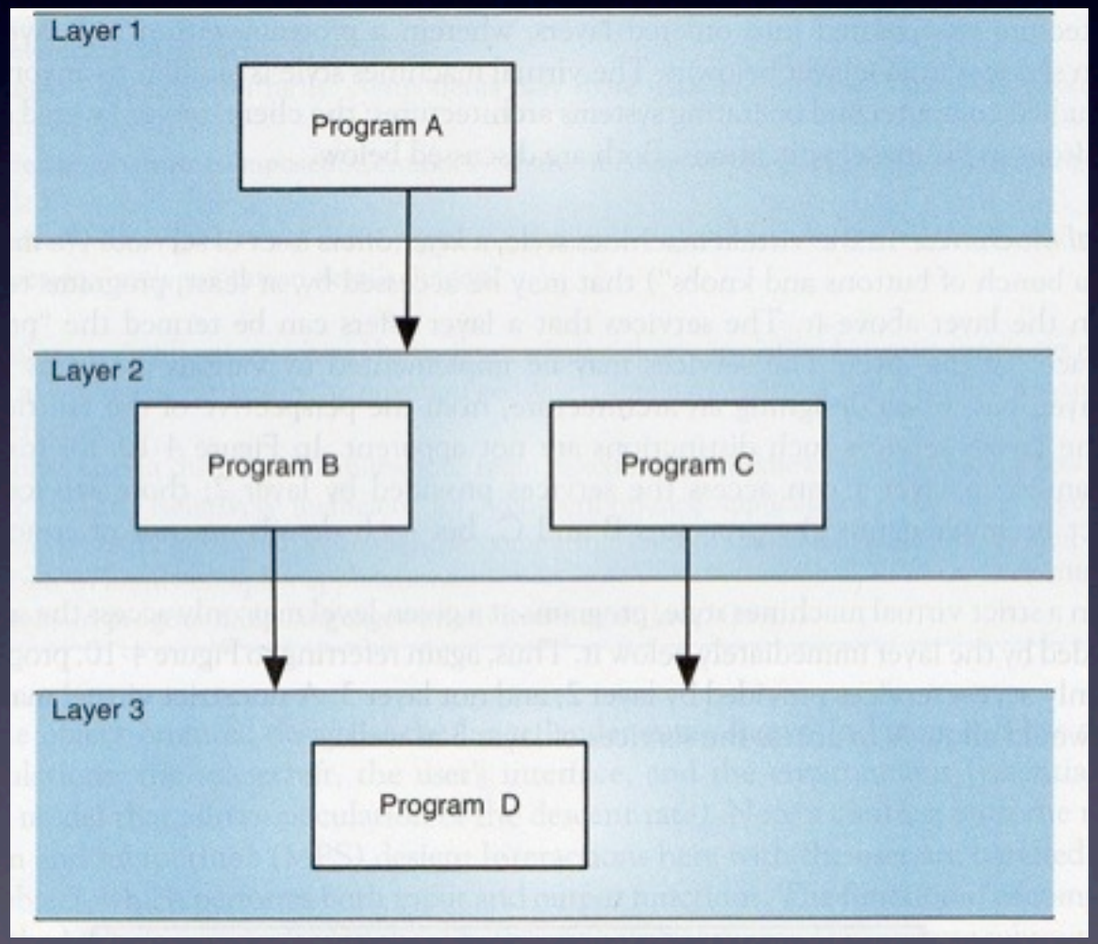
- Layered (Virtual Machine)

Summary: Consists of an ordered sequence of layers; each *layer*, or virtual machine, offers a set of services that may be accessed by programs (subcomponents) residing within the layer above it.
Components: Layers offering a set of services to other layers, typically comprising several programs (subcomponents).
Connectors: Typically procedure calls.
Data elements: Parameters passed between layers.
Topology: Linear, for strict virtual machines; a directed acyclic graph in looser interpretations.
Additional constraints imposed: None.
Qualities yielded: Clear dependence structure; software at upper levels immune to changes of implementation within lower levels as long as the service specifications are invariant. Software at lower levels fully independent of upper levels.

Typical uses: Operating system design; network protocol stacks.
Cautions: Strict virtual machines with many levels can be relatively inefficient.

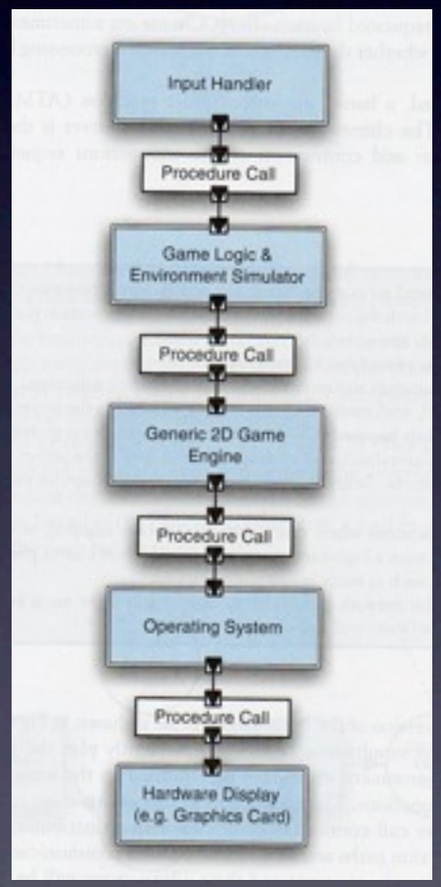
Estilos

- Layered (Virtual Machine)



Estilos

- Layered (Virtual Machine)



Estilos

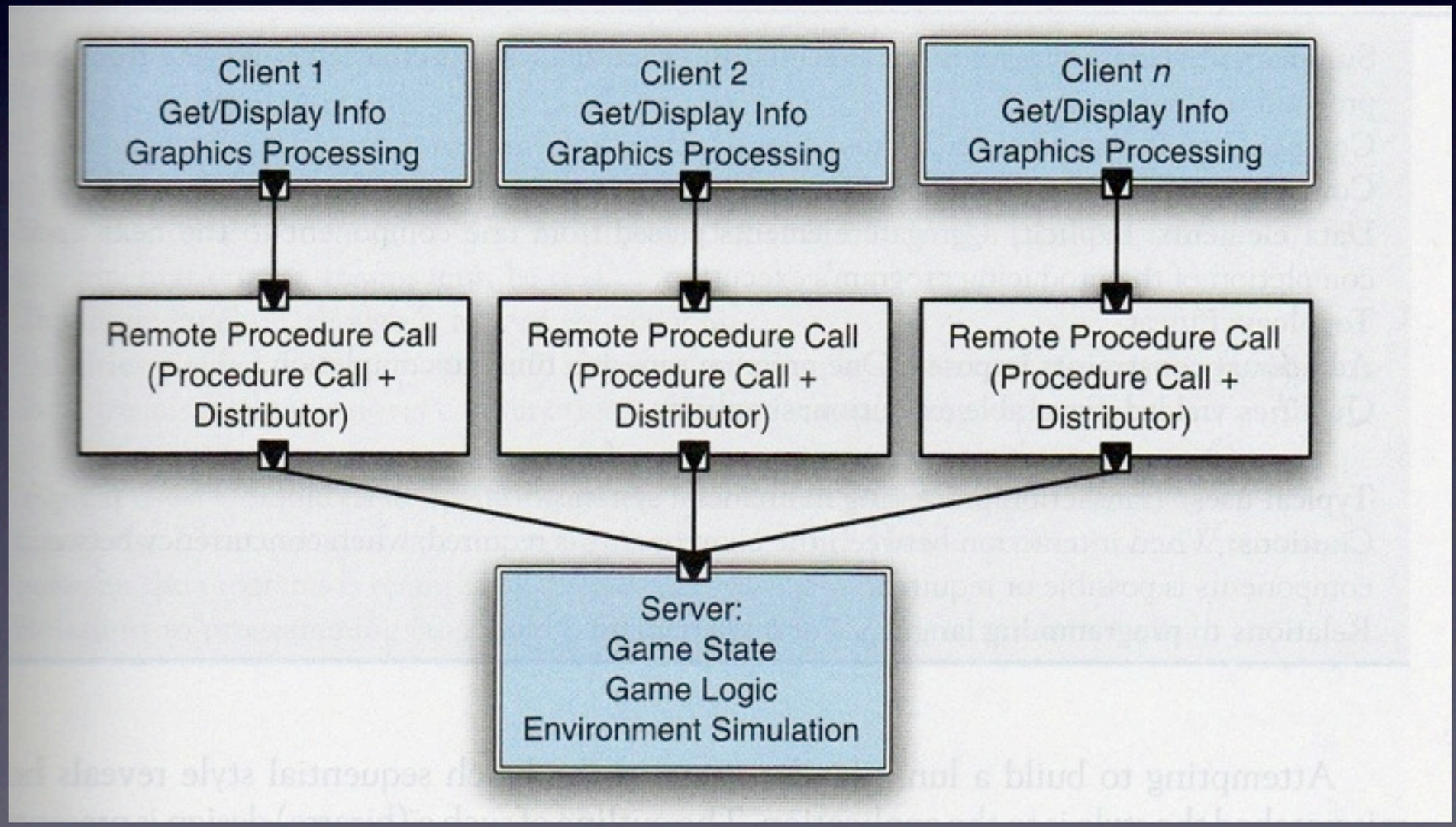
- Layered (Client-Server)

Summary: Clients send service requests to the server, which performs the required functions and replies as needed with the requested information. Communication is initiated by the clients.
Components: Clients and server.
Connectors: Remote procedure call, network protocols.
Data elements: Parameters and return values as sent by the connectors.
Topology: Two-level, with multiple clients making requests to the server.
Additional constraints imposed: Client-to-client communication prohibited.
Qualities yielded: Centralization of computation and data at the server, with the information made available to remote clients. A single powerful server can service many clients.

Typical uses: Applications where centralization of data is required, or where processing and data storage benefit from a high-capacity machine, and where clients primarily perform simple user interface tasks, such as many business applications.
Cautions: When the network bandwidth is limited and there are a large number of client requests.

Estilos

- Layered (Client-Server)



Estilos

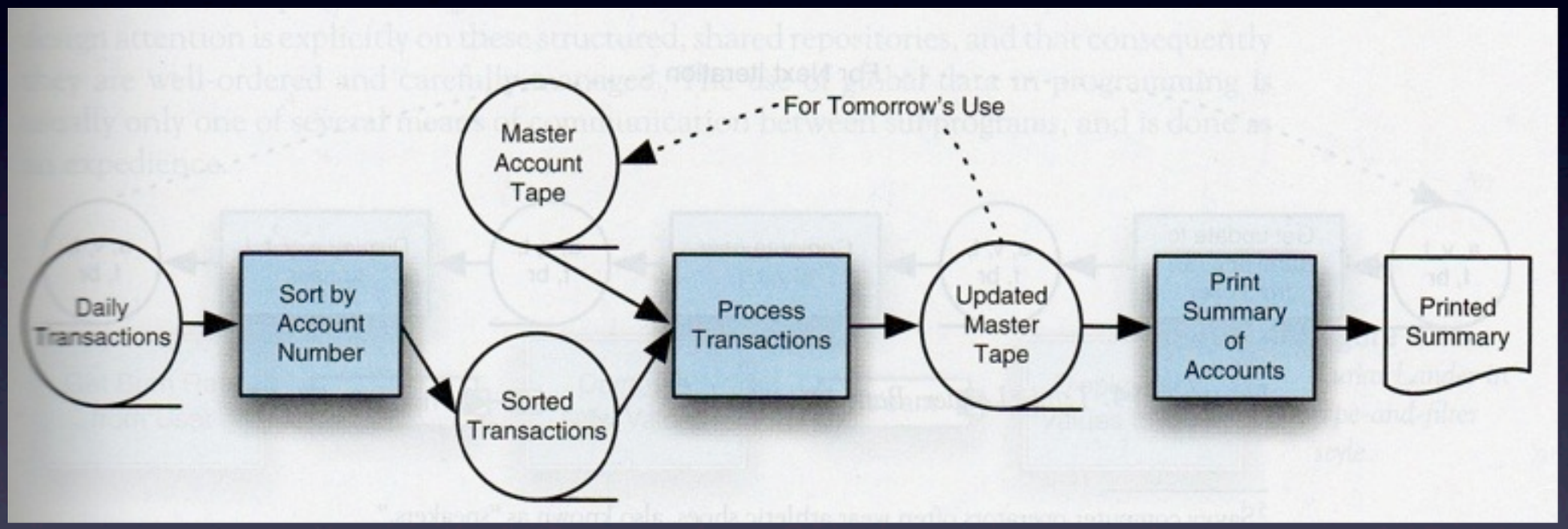
- Dataflow (Batch-Sequential)

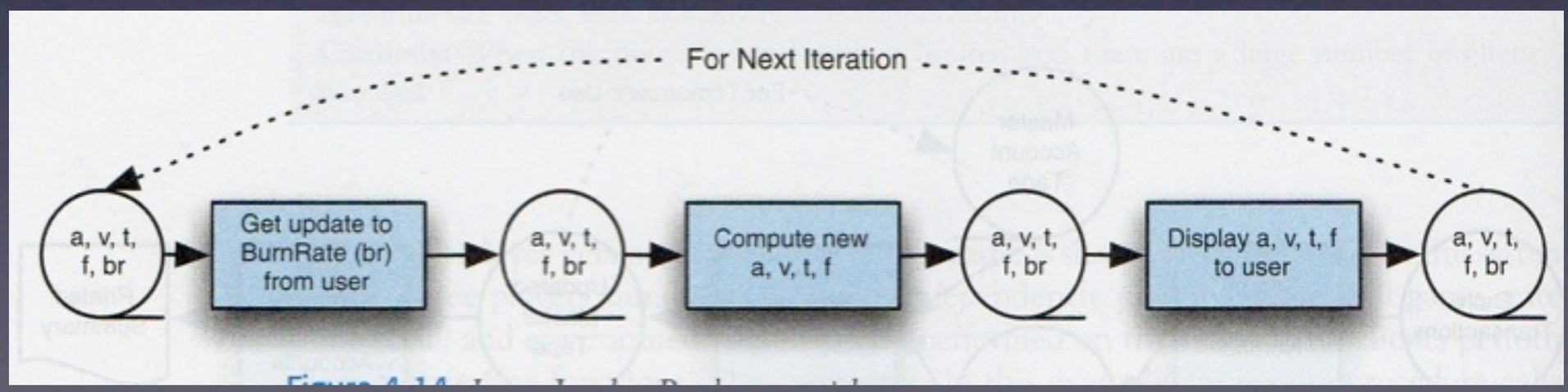
Summary: Separate programs are executed in order; data is passed as an aggregate from one program to the next.
Components: Independent programs.
Connectors: The human hand carrying tapes between the programs, aka "sneaker-net."[5]
Data elements: Explicit, aggregate elements passed from one component to the next upon completion of the producing program's execution.
Topology: Linear.
Additional constraints imposed: One program runs at a time, to completion.
Qualities yielded: Severable execution; simplicity.

Typical uses: Transaction processing in financial systems.
Cautions: When interaction between the components is required; when concurrency between components is possible or required.
Relations to programming languages or environments: None.

Estilos

- Dataflow (Batch-Sequential)





Estilos

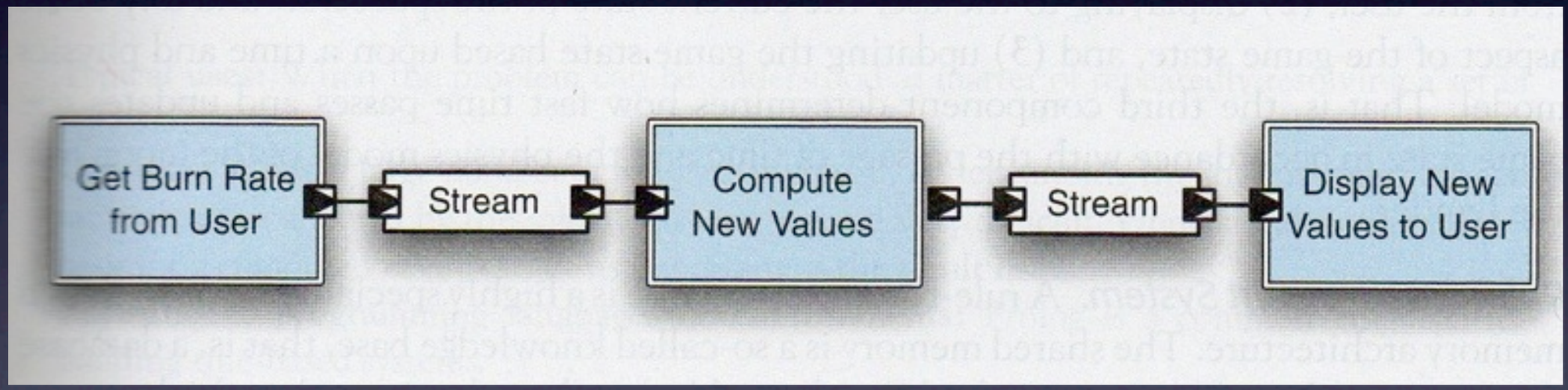
- Dataflow (Pipe-and-Filter)

Summary: Separate programs are executed, potentially concurrently; data is passed as a stream from one program to the next.
Components: Independent programs, known as filters.
Connectors: Explicit routers of data streams; service provided by operating system.
Data elements: Not explicit; must be (linear) data streams. In the typical Unix/Linux/DOS implementation the streams must be text.
Topology: Pipeline, though T fittings are possible.
Qualities yielded: Filters are mutually independent. Simple structure of incoming and outgoing data streams facilitates novel combinations of filters for new, composed applications.

Typical uses: Ubiquitous in operating system application programming.
Cautions: When complex data structures must be exchanged between filters; when interactivity between the programs is required.
Relations to programming languages or environments: Prevalent in Unix shells.

Estilos

- Dataflow (Pipe-and-Filter)



Estilos

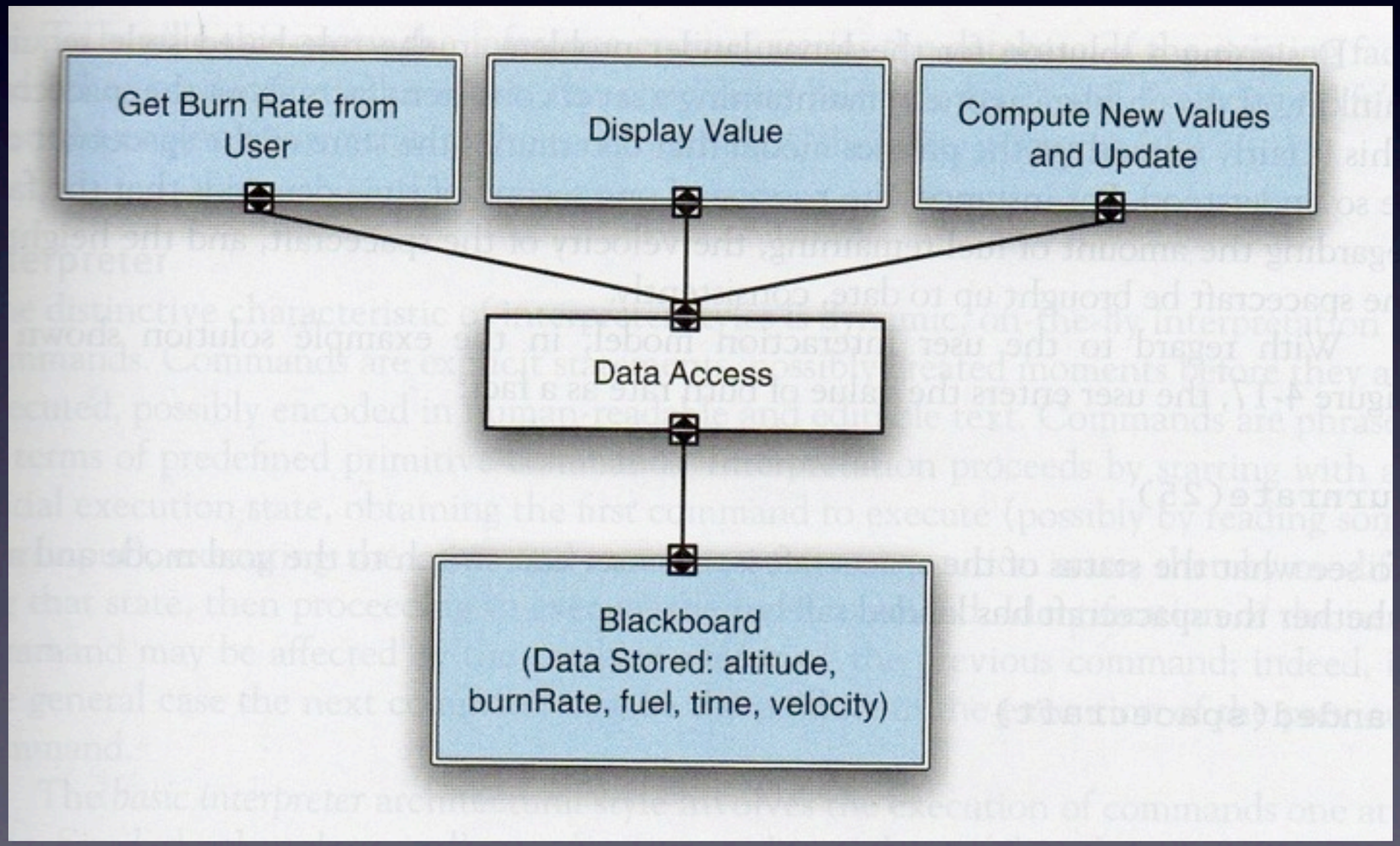
- Shared Memory (BlackBoard)

Summary: Independent programs access and communicate exclusively through a global data repository, known as a blackboard.
Components: Independent programs, sometimes referred to as "knowledge sources," blackboard.
Connectors: Access to the blackboard may be by direct memory reference, or can be through a procedure call or a database query.
Data elements: Data stored in the blackboard.
Topology: Star topology, with the blackboard at the center.
Variants: In one version of the style, programs poll the blackboard to determine if any values of interest have changed; in another version, a blackboard manager notifies interested components of an update to the blackboard.
Qualities yielded: Complete solution strategies to complex problems do not have to be preplanned. Evolving views of the data/problem determine the strategies that are adopted.

Typical uses: Heuristic problem solving in artificial intelligence applications.
Cautions: When a well-structured solution strategy is available; when interactions between the independent programs require complex regulation; when representation of the data on the blackboard is subject to frequent change (requiring propagating changes to all the participating components).
Relations to programming languages or environments: Versions of the blackboard style that allow concurrency between the constituent programs require concurrency primitives for managing the shared blackboard.

Estilos

- Shared Memory (BlackBoard)



Estilos

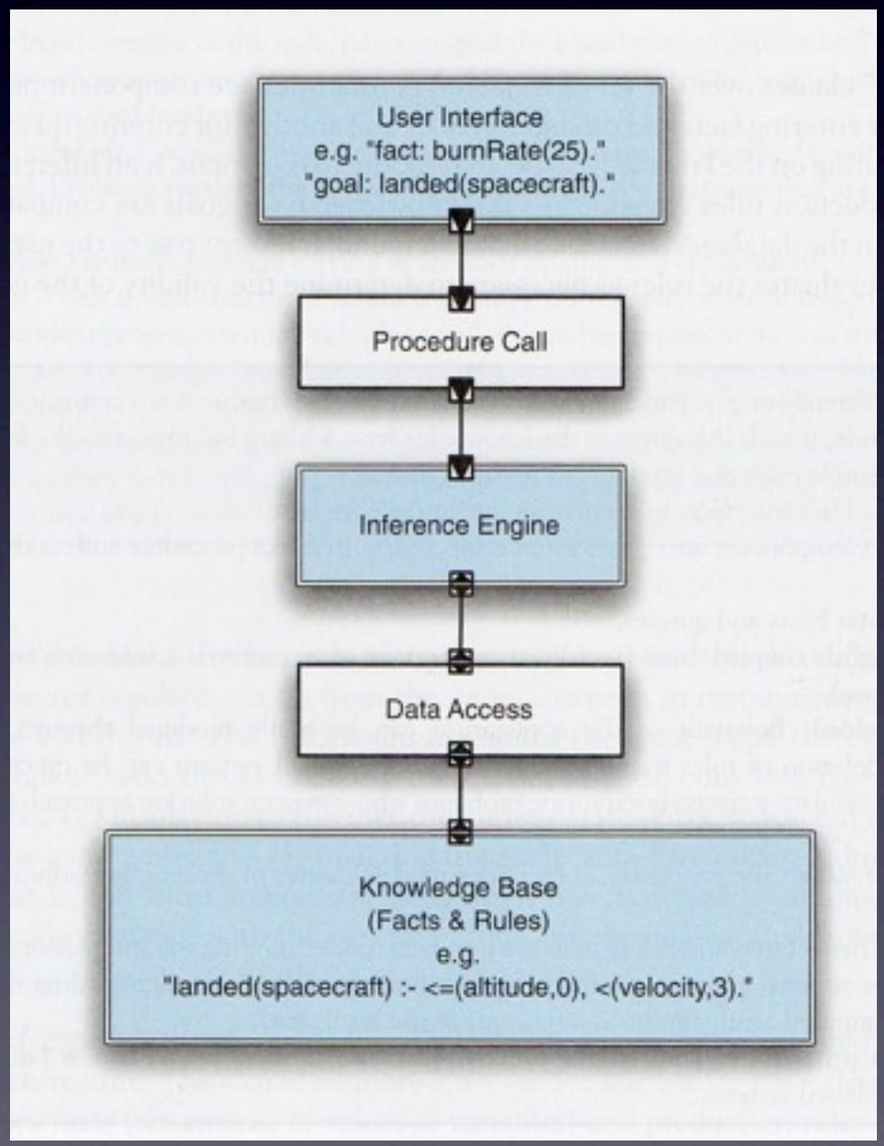
- Shared Memory (Rule-Based)

Summary: Inference engine parses user input and determines whether it is a fact/rule or a query. If it is a fact/rule, it adds this entry to the knowledge base. Otherwise, it queries the knowledge base for applicable rules and attempts to resolve the query.
Components: User interface, inference engine, knowledge base.
Connectors: Components are tightly interconnected, with direct procedure calls and/or shared data access.
Data Elements: Facts and queries.
Topology: Tightly coupled three-tier (direct connection of user interface, inference engine, and knowledge base).
Qualities yielded: Behavior of the application can be easily modified through dynamic addition or deletion of rules from the knowledge base. Small systems can be quickly prototyped. Thus useful for iteratively exploring problems whose general solution approach is unclear.

Typical uses: When the problem can be understood as matter of repeatedly resolving a set of predicates.
Cautions: When a large number of rules are involved, understanding the interactions between multiple rules affected by the same facts can become very difficult. Understanding the logical basis for a computed result can be as important as the result itself.
Relations to programming languages or environments: Prolog is a common language for building rule-based systems.

Estilos

- Shared Memory (Rule-Based)



Estilos

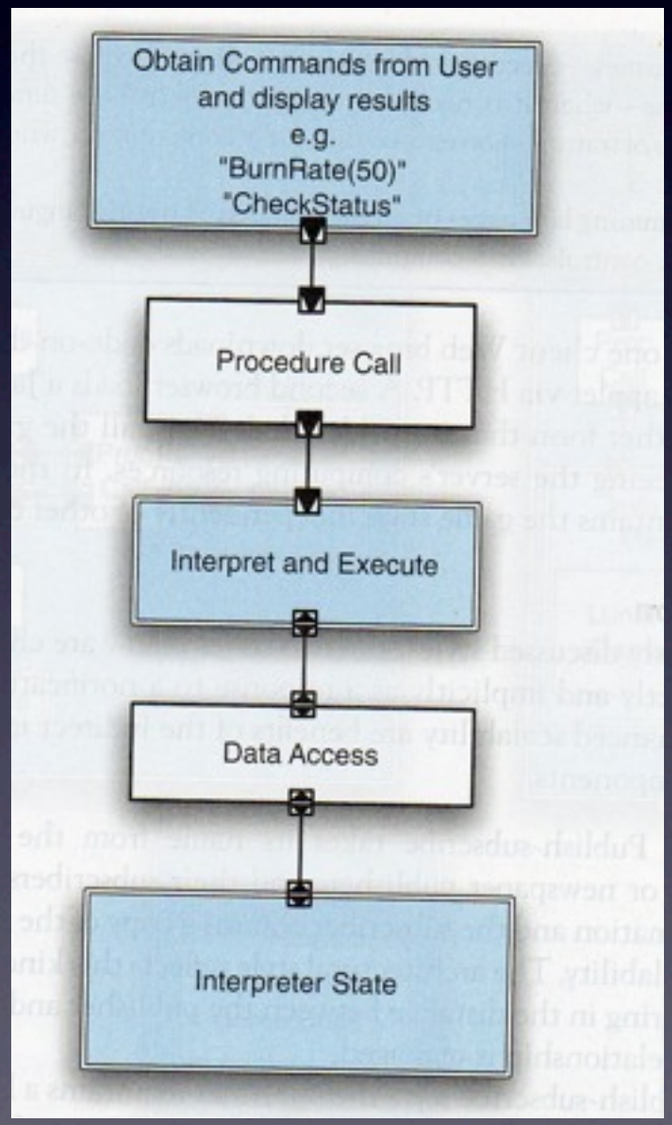
- Interpreter (Interpreter)

Summary: Interpreter parses and executes input commands, updating the state maintained by the interpreter.
Components: Command interpreter, program/interpreter state, user interface.
Connectors: Typically the command interpreter, user interface, and state are very closely bound with direct procedure calls and shared state.
Data elements: Commands.
Topology: Tightly coupled three-tier; state can be separated from the interpreter.
Qualities yielded: Highly dynamic behavior possible, where the set of commands is dynamically modified. System architecture may remain constant while new capabilities are created based upon existing primitives.

Typical uses: Superb for end-user programmability; supports dynamically changing set of capabilities.
Cautions: When fast processing is needed (it takes longer to execute interpreted code than executable code); memory management may be an issue, especially when multiple interpreters are invoked simultaneously.
Relations to programming languages or environments: Lisp and Scheme are interpretive languages, and sometimes used when building other interpreters; Word/Excel macros.

Estilos

- Interpreter (Interpreter)



Estilos

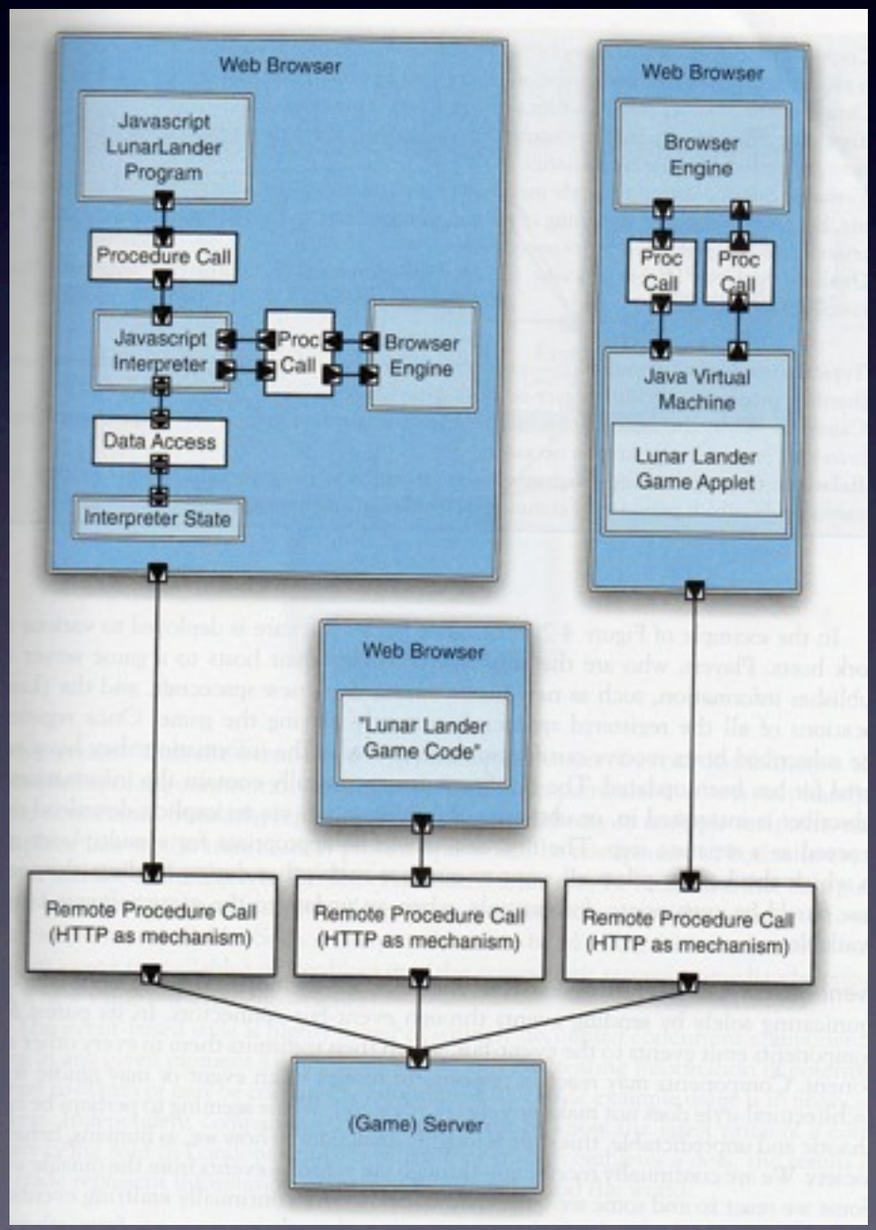
- Interpreter (Mobile code)

Summary: Code moves to be interpreted on another host; depending on the variant, state does also.
Components: Execution dock, which handles receipt and deployment of code and state; code compiler/interpreter.
Connectors: Network protocols and elements for packaging code and data for transmission.
Data elements: Representations of code as data; program state; data.
Topology: Network.
Variants: Code-on-demand, remote evaluation, and mobile agent.
Qualities yielded: Dynamic adaptability. Takes advantage of the aggregate computing power of available hosts; increased dependability through provision of migration to new hosts.

Typical uses: When processing large data sets in distributed locations, it becomes more efficient to have the code move to the location of these large data sets; when it is desirous to dynamically customize a local processing node through inclusion of external code.

Estilos

- Interpreter (Mobile code)



Estilos

- Implicit Invocation (Publish-subscribe)

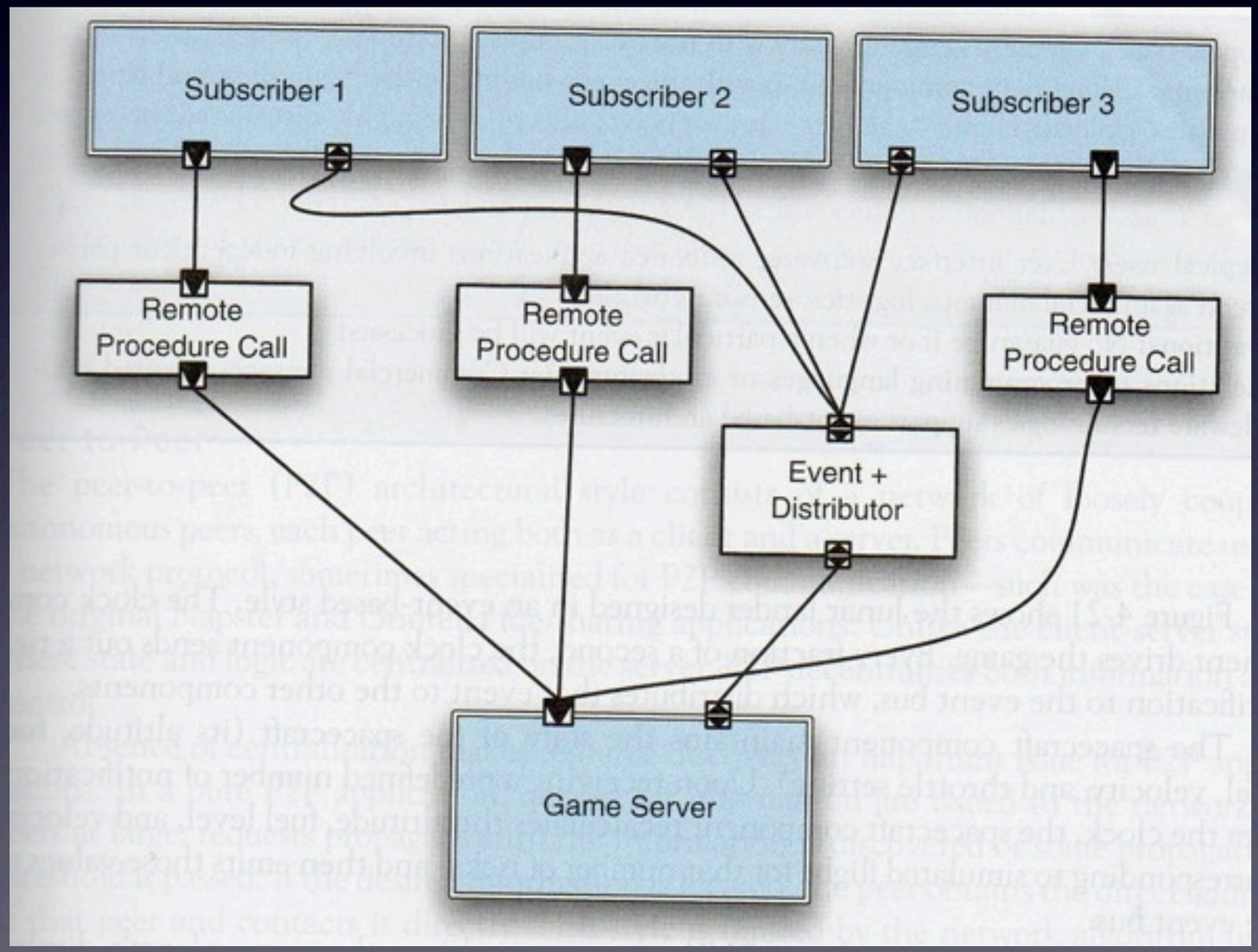
Summary: Subscribers register/deregister to receive specific messages or specific content. Publishers maintain a subscription list and broadcast messages to subscribers either synchronously or asynchronously.
Components: Publishers, subscribers, proxies for managing distribution.

Connectors: Procedure calls may be used within programs, more typically a network protocol is required. Content-based subscription requires sophisticated connectors.
Data elements: Subscriptions, notifications, published information.
Topology: Subscribers connect to publishers either directly or may receive notifications via a network protocol from intermediaries.
Variants: Specific uses of the style may require particular steps for subscribing and unsubscribing. Support for complex matching of subscription interests and available information may be provided and be performed by intermediaries.
Qualities yielded: Highly efficient one-way dissemination of information with very low coupling of components.

Typical uses: News dissemination—whether in the real world or online events. Graphical user interface programming. Multiplayer-network–based games.
Cautions: When the number of subscribers for a single data item is very large a specialized broadcast protocol will likely be necessary.
Relations to programming languages or environments: In large-scale systems support for publish-subscribe is provided by commercial middleware technology.

Estilos

- Implicit Invocation (Publish-subscribe)



Estilos

- Implicit Invocation (Event-based)

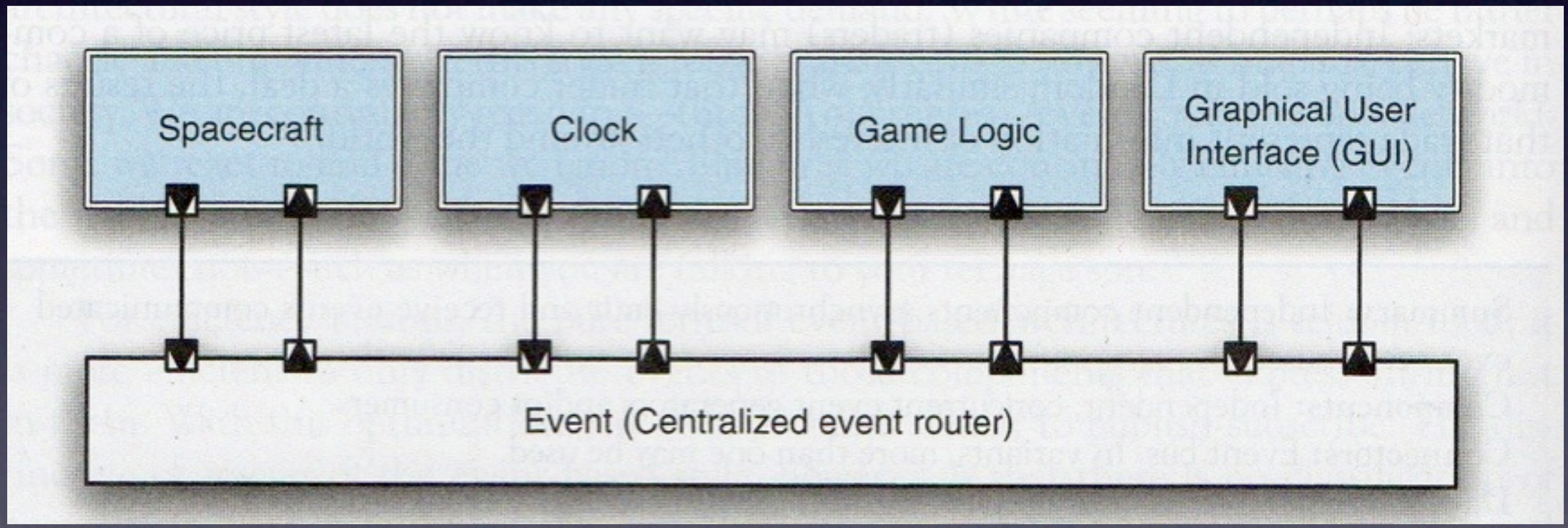
Summary: Independent components asynchronously emit and receive events communicated over event buses.
Components: Independent, concurrent event generators and/or consumers.
Connectors: Event bus. In variants, more than one may be used.
Data elements: Events—data sent as a first-class entity over the event bus.

Topology: Components communicate with the event-buses, not directly to each other.
Variants: Component communication with the event-bus may either be push or pull based.
Qualities yielded: Highly scalable, easy to evolve, effective for highly distributed, heterogeneous applications.

Typical uses: User interface software, wide-area applications involving independent parties (such as financial markets, logistics, sensor networks).
Cautions: No guarantee if or when a particular event will be processed.
Relations to programming languages or environments: Commercial message–oriented middleware technologies support event-based architectures.

Estilos

- Implicit Invocation (Event-based)



Estilos

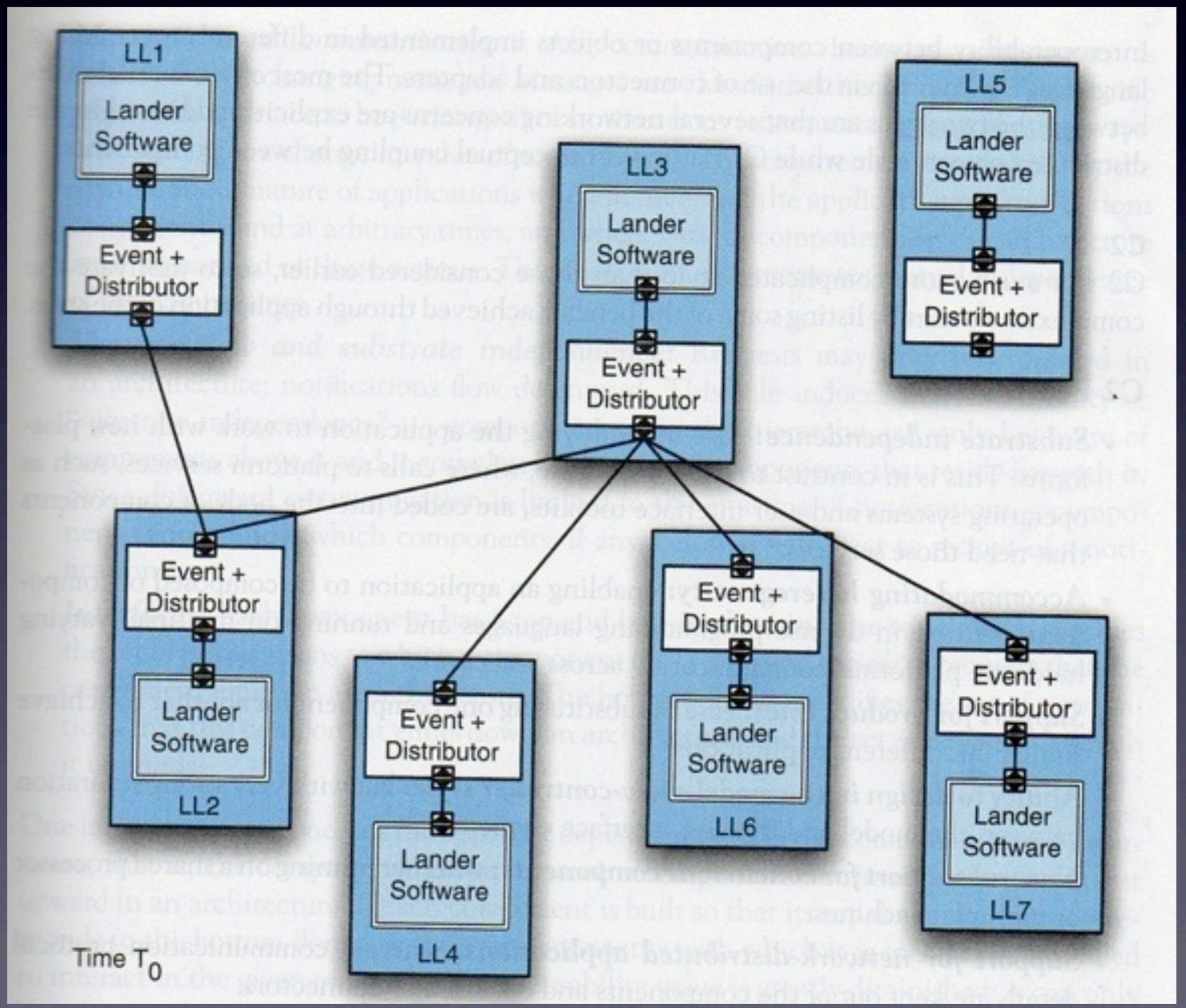
- Peer-to-Peer

Summary: State and behavior are distributed among peers that can act as either clients or servers.
Components: Peers—independent components, having their own state and control thread.
Connectors: Network protocols, often custom.
Data elements: Network messages.
Topology: Network (may have redundant connections between peers); can vary arbitrarily and dynamically.
Qualities yielded: Decentralized computing with flow of control and resources distributed among peers. Highly robust in the face of failure of any given node. Scalable in terms of access to resources and computing power.

Typical uses: Where sources of information and operations are distributed and network is ad hoc.
Cautions: When information retrieval is time critical and cannot afford the latency imposed by the protocol. Security—P2P networks must make provision for detecting malicious peers and managing trust in an open environment.

Estilos

- Peer-to-Peer



Estilos

- C2

Summary: An indirect invocation style in which independent components communicate exclusively through message routing connectors. Strict rules on connections between components and connectors induce layering.
Components: Independent, potentially concurrent message generators and/or consumers.
Connectors: Message routers that may filter, translate, and broadcast messages of two kinds—notifications and requests.
Data elements: Messages—data sent as first-class entities over the connectors. Notification messages announce changes of state. Request messages request performance of an action.
Topology: Layers of components and connectors, with a defined top and bottom, wherein notifications flow downward and requests upward.

Additional constraints imposed:

- All components and connectors have a defined top and bottom. The top of a component may be attached to the bottom of a single connector and the bottom of a component may be attached to the top of a single connector. No direct component-to-component links are allowed; there is, however, no bound on the number of components or connectors that may be attached to a single connector. When two connectors are attached to each other, it must be from the bottom of one to the top of the other.
- Each component has a top and bottom *domain*. The top domain specifies the set of notifications to which a component may react and the set of requests that the component emits up an architecture. The bottom domain specifies the set of notifications that this component emits down an architecture and the set of requests to which it responds.
- Components may be hierarchically composed, where an entire architecture becomes a single component in another, larger architecture.
- Each component may have its own thread(s) of control.
- There can be no assumption of a shared address space among components.

Qualities yielded:

- Substrate independence: Ease in moving the application to new platforms.
- Applications composable from heterogeneous components running on diverse platforms.

Estilos

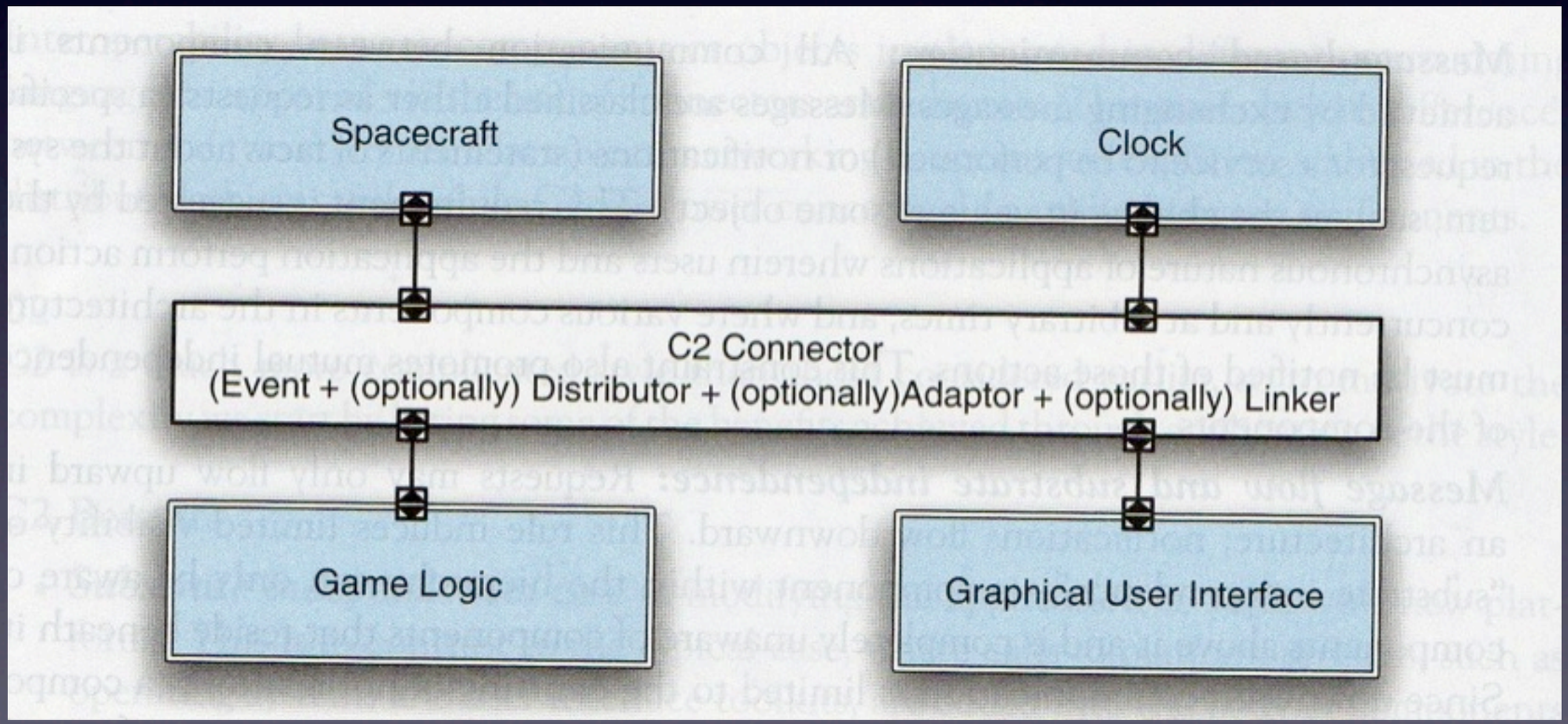
- C2

- Support for product lines.
- Ability to program in the model-view-controller style, but with very strong separation between the model and the user interface elements.
- Support for concurrent components.
- Support for network-distributed applications.

Typical uses: Reactive, heterogeneous applications. Applications demanding low-cost adaptability.
Cautions: Event-routing across multiple layers can be inefficient. Overhead high for some simple kinds of component interaction.
Relations to programming languages or environments: Programming frameworks are used to facilitate creation of implementations faithful to architectures in the style. Support for Java, C, Ada.

Estilos

- C2


Spacecraft
Clock
C2 Connector
(Event + (optionally) Distributor + (optionally)Adaptor + (optionally) Linker
Game Logic
Graphical User Interface

Conectores

Conectores

- Realizan transferencia de control y datos entre componentes
- Proveen servicios
  - Persistencia
  - Invocación
  - Mensajería
  - Transaccionalidad
- Normalmente considerados “Componentes”

Conectores

- Roles Principales
  - Comunicación
  - Coordinación
  - Conversión
  - Facilitadores

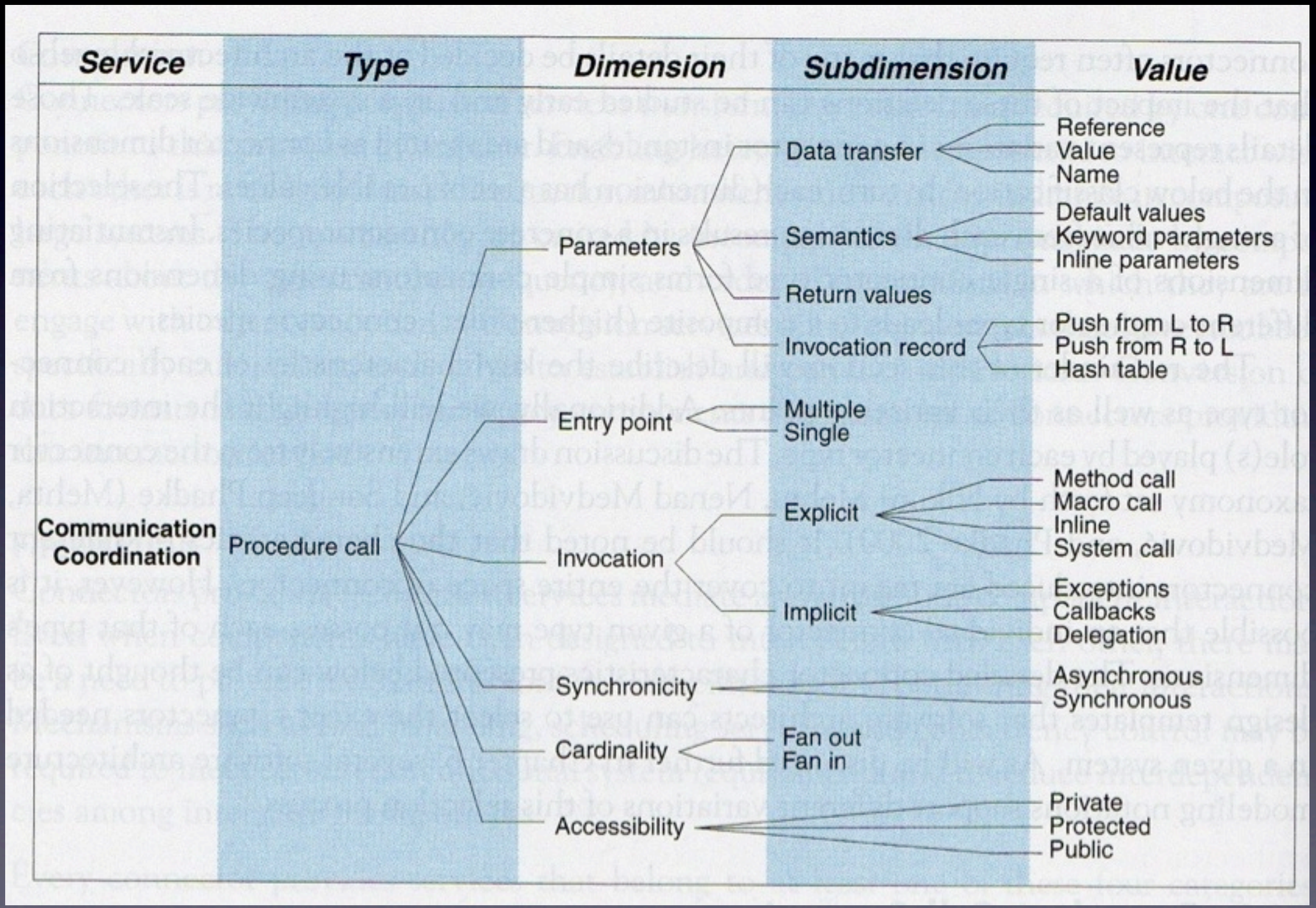
Conectores

- Tipos de Conectores
  - Procedure Call
  - Event
  - Data access
  - Linkage
  - Stream
  - Arbitrator
  - Adaptor
  - Distributor

Conectores

- Procedure Call Connectors
  - Modela el flujo de control entre componentes (Coordinador)
  - Flujo de datos - Parámetros (Comunicación)

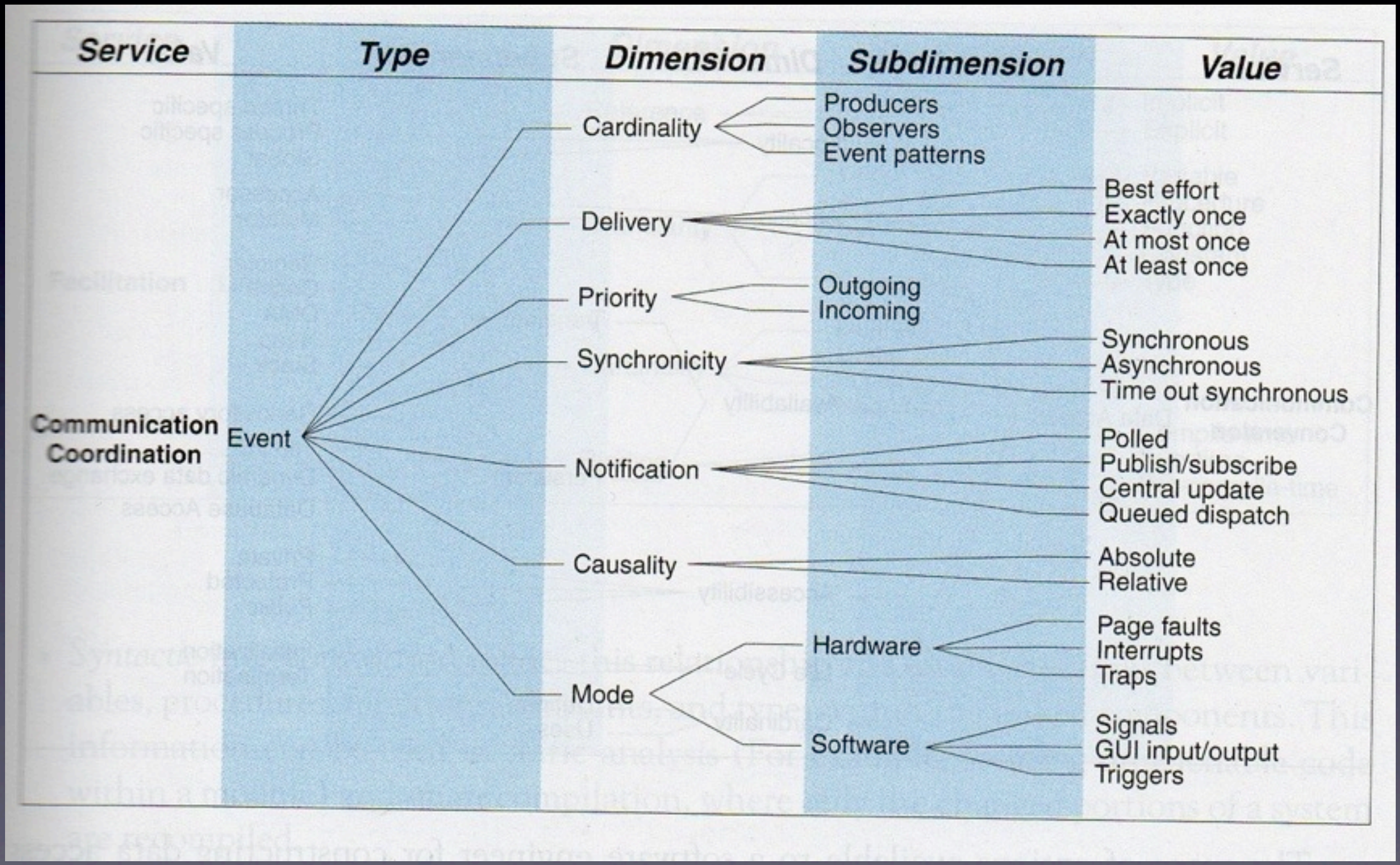
Conectores


Service
Type
Dimension
Subdimension
Value
Communication
Coordination
Procedure call
Parameters
Data transfer
Reference
Value
Name
Semantics
Default values
Keyword parameters
Inline parameters
Return values
Invocation record
Push from L to R
Push from R to L
Hash table
Entry point
Multiple
Single
Invocation
Explicit
Method call
Macro call
Inline
System call
Implicit
Exceptions
Callbacks
Delegation
Synchronicity
Asynchronous
Synchronous
Cardinality
Fan out
Fan in
Accessibility
Private
Protected
Public

Conectores

- Event Connectors
- Coordinan el flujo de control
- Basado en la ocurrencia de un evento y la generación de mensajes
- Usados en aplicaciones distribuidas

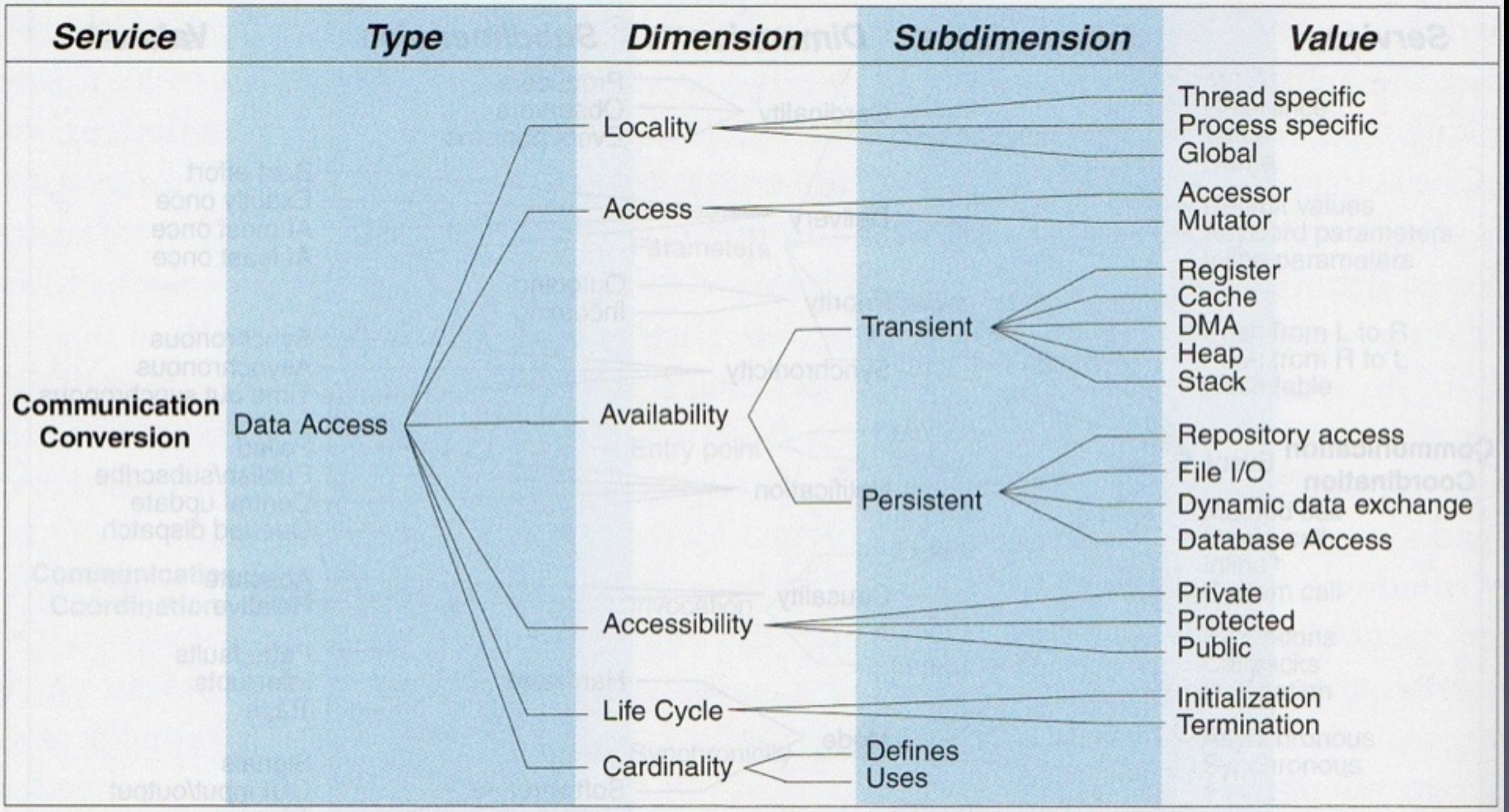
Conectores


Service
Type
Dimension
Subdimension
Value
Communication
Coordination
Event
Cardinality
Producers
Observers
Event patterns
Delivery
Best effort
Exactly once
At most once
At least once
Priority
Outgoing
Incoming
Synchronicity
Synchronous
Asynchronous
Time out synchronous
Notification
Polled
Publish/subscribe
Central update
Queued dispatch
Causality
Absolute
Relative
Mode
Hardware
Page faults
Interrupts
Traps
Software
Signals
GUI input/output
Triggers

Conectores

- Data Access Connectors
  - Permite a los componentes acceder información que reside en un componente de almacenamiento
  - Puede almacenar datos de manera persistente o temporal

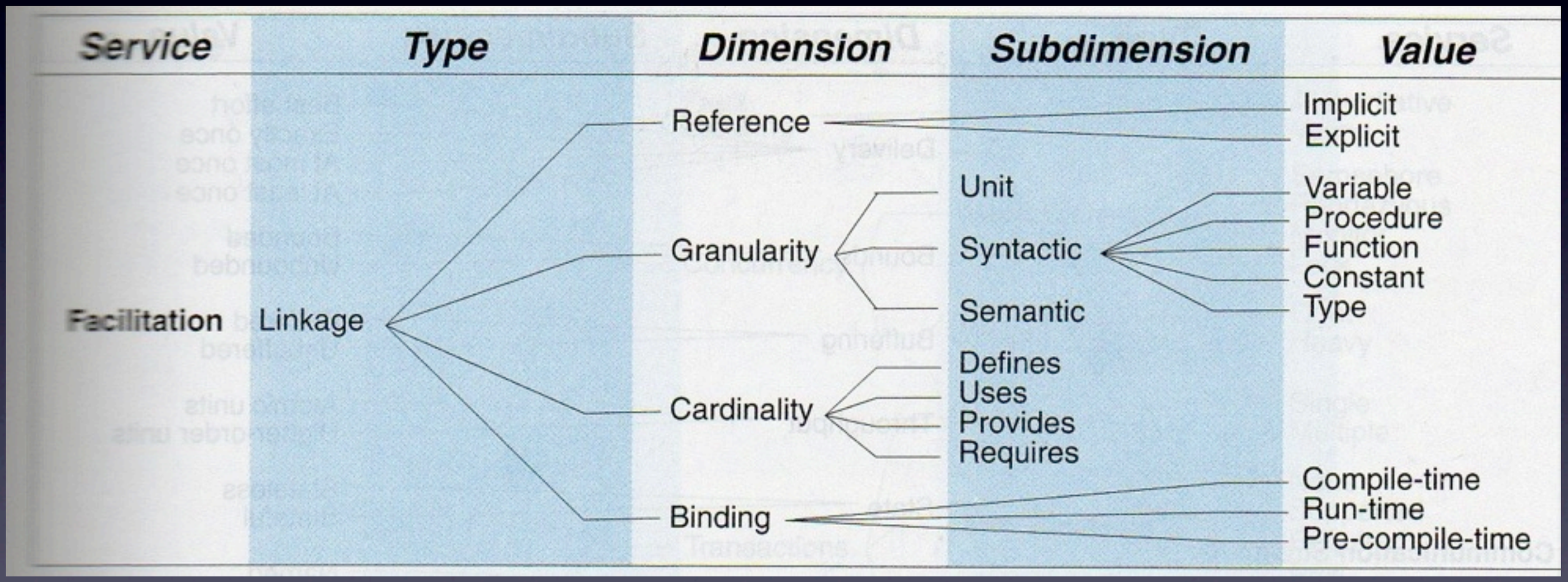
Conectores


Service
Type
Dimension
Subdimension
Value
Communication
Conversion
Data Access
Locality
Thread specific
Process specific
Global
Access
Accessor
Mutator
Availability
Transient
Register
Cache
DMA
Heap
Stack
Persistent
Repository access
File I/O
Dynamic data exchange
Database Access
Accessibility
Private
Protected
Public
Life Cycle
Initialization
Termination
Cardinality
Defines
Uses

Conectores

- Linkage Connectors
- Facilitan la creación de canales de comunicación y coordinación
- Utilizados por conectores de mas alto nivel

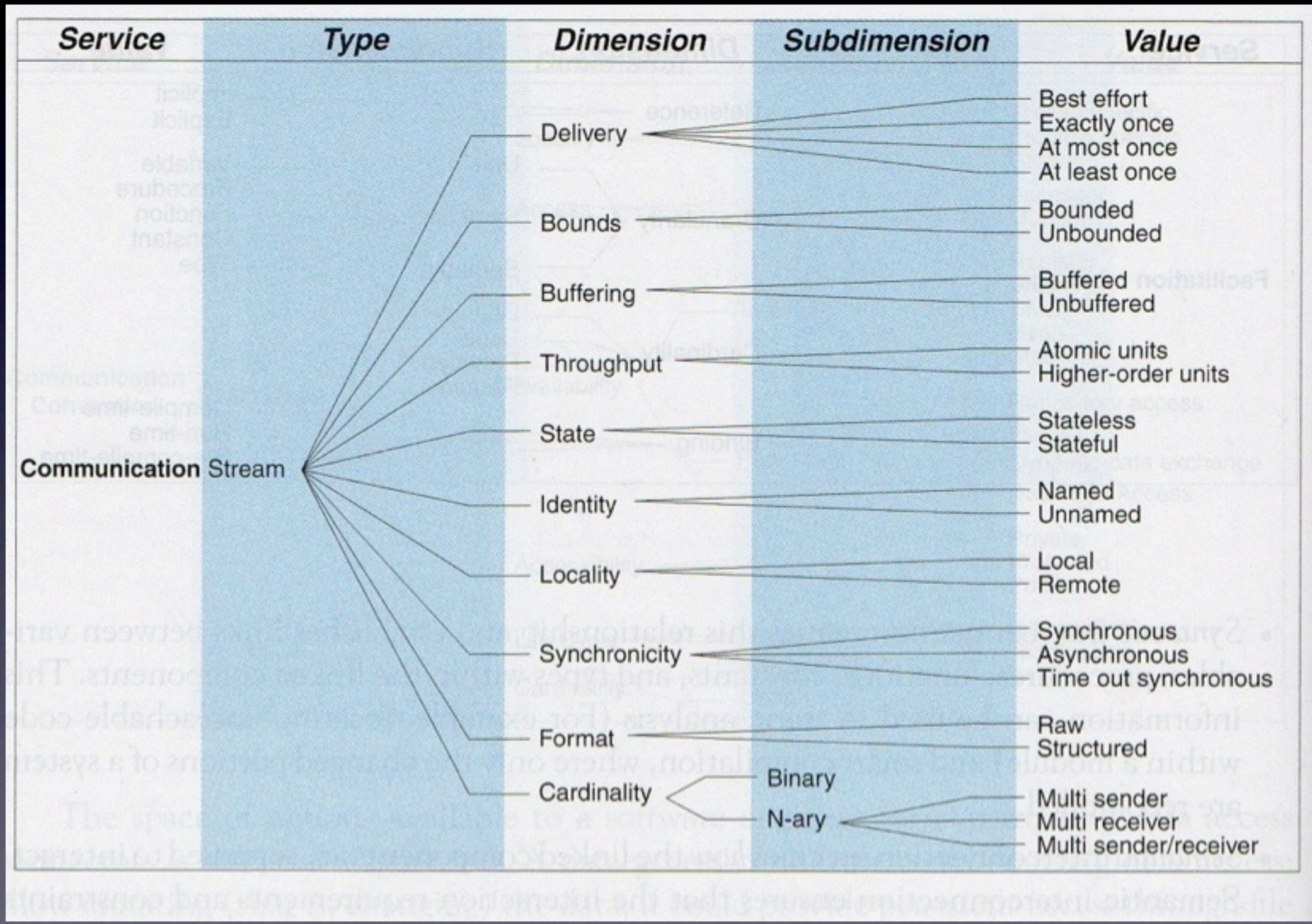
Conectores


Service
Type
Dimension
Subdimension
Value
Facilitation
Linkage
Reference
Implicit
Explicit
Granularity
Unit
Syntactic
Variable
Procedure
Function
Constant
Type
Semantic
Cardinality
Defines
Uses
Provides
Requires
Binding
Compile-time
Run-time
Pre-compile-time

Conectores

- Stream Connectors
  - Utilizados para transmitir grandes cantidades de datos entre procesos autónomos
  - Utilizados con otros conectores para acceso a BD y archivos

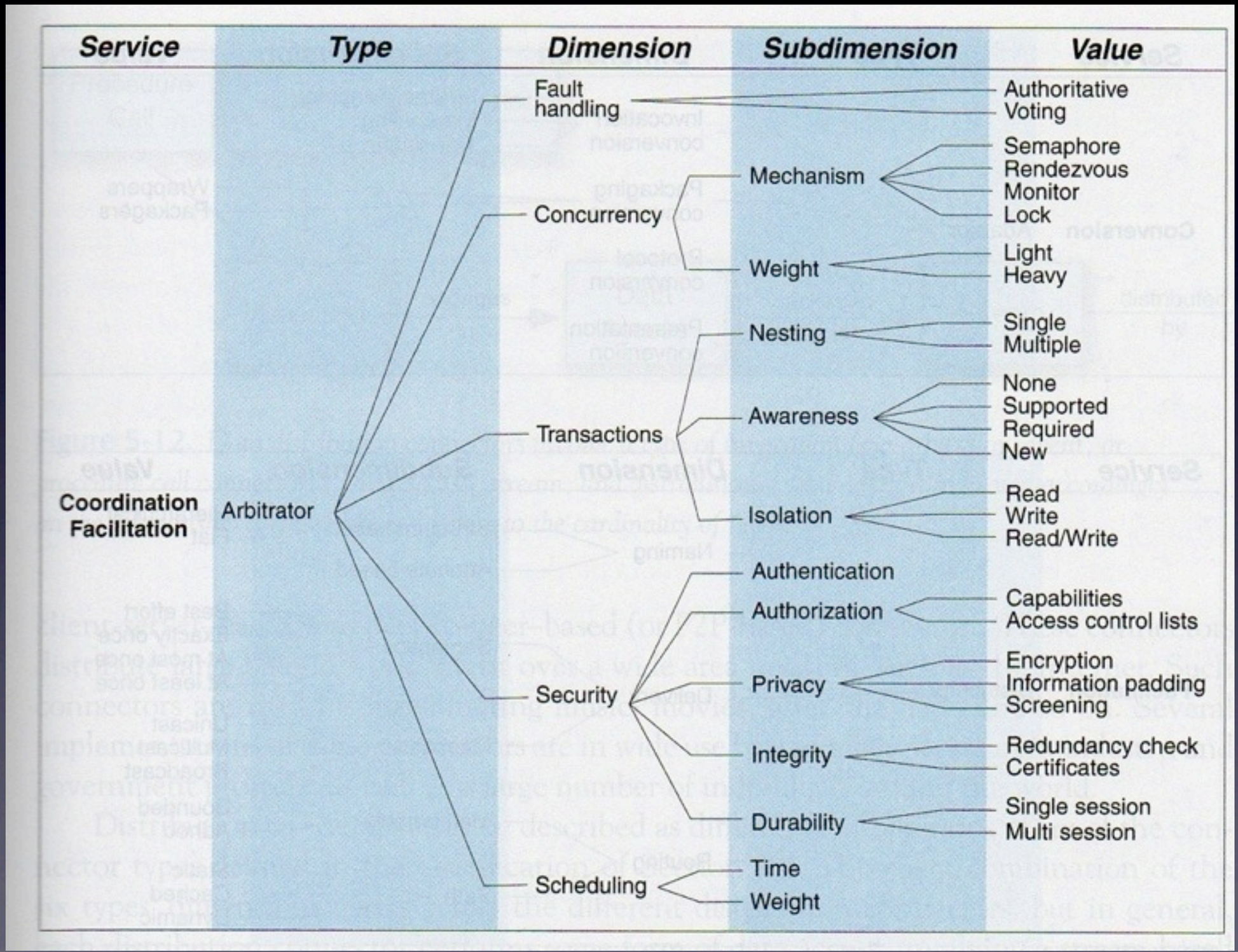
Conectores


Service
Type
Dimension
Subdimension
Value
Communication
Stream
Delivery
Best effort
Exactly once
At most once
At least once
Bounds
Bounded
Unbounded
Buffering
Buffered
Unbuffered
Throughput
Atomic units
Higher-order units
State
Stateless
Stateful
Identity
Named
Unnamed
Locality
Local
Remote
Synchronicity
Synchronous
Asynchronous
Time out synchronous
Format
Raw
Structured
Cardinality
Binary
N-ary
Multi sender
Multi receiver
Multi sender/receiver

Conectores

- Arbitrator Connectors
  - Ayudan en situaciones donde no se sabe el estado de los componentes
    - Ejemplo:Sincronización y concurrencia
  - Ayudan en la negociación de niveles de servicio
    - Ejemplo: Scheduling, Load Balancing

Conectores


Service
Type
Dimension
Subdimension
Value
Coordination Facilitation
Arbitrator
Fault handling
Authoritative
Voting
Concurrency
Mechanism
Semaphore
Rendezvous
Monitor
Lock
Weight
Light
Heavy
Nesting
Single
Multiple
Transactions
Awareness
None
Supported
Required
New
Isolation
Read
Write
Read/Write
Security
Authentication
Authorization
Capabilities
Access control lists
Privacy
Encryption
Information padding
Screening
Integrity
Redundancy check
Certificates
Durability
Single session
Multi session
Scheduling
Time
Weight

Conectores

- Adaptor Connectors
  - Proveen facilidades para soportar interacción entre componentes
    - No diseñados para interoperar
  - Proveen servicios de conversión

Conectores

Service	Type	Dimension	Subdimension	Value
Conversion	Adaptor	Invocation conversion	Address Mapping	
			Marshaling	
			Translation	
		Packaging conversion		Wrappers
				Packagers
		Protocol conversion		
		Presentation conversion		

Service	Type	Dimension	Subdimension	Value
Facilitation	Distributor	Naming	Structure based	Hierarchical
				Flat
			Attribute based	
		Delivery	Semantics	Best effort
				Exactly once
				At most once
				At least once
			Mechanism	Unicast
				Multicast
				Broadcast
		Routing	Membership	Bounded
				Adhoc
			Path	Static
				Cached
				Dynamic

Conectores

- Distributor Connectors
  - Realizan la identificación e interacción de caminos y rutas de comunicación y coordinación entre componentes
  - Ejemplo: DNS, Routing, Switching

Plan de Trabajo

- Lo que sigue
  - Documento Arquitectura
    - Escenarios Operacionales - Casos Uso
    - Punto de Vista Funcional
    - Diagrama de Componentes
    - Identificación / Documentación Servicios
  - Preparación para el desarrollo
    - Tutoriales Spring/OSGI
    - Ambiente de desarrollo y producción

Referencias

- Software Architecture: Foundations, Theory and Practice. Richard Taylor, Nenad Medvidovic, and Eric Dashofy