Università della Svizzera italiana

Faculty of Informatics

Software Connectors

Prof. Cesare Pautasso
http://www.pautasso.info
cesare.pautasso@usi.ch
@pautasso

Connector

- Generic enabler of composition

Plug into matching component ports

Transfer data, control signals between ports

- Enable architects to assemble heterogeneous functionality, developed at different times, in different locations, by different organizations.

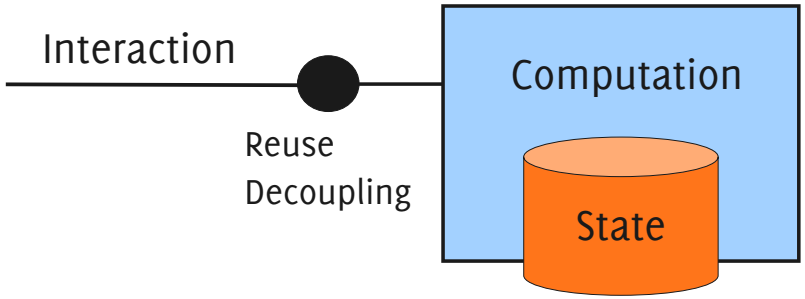
Software Connector

- Perform and regulate interactions between components

- A connector couples two or more components to perform transfers of data and control

Components vs. Connectors

- Connectors are a model of static and dynamic aspects of the interaction between component interfaces



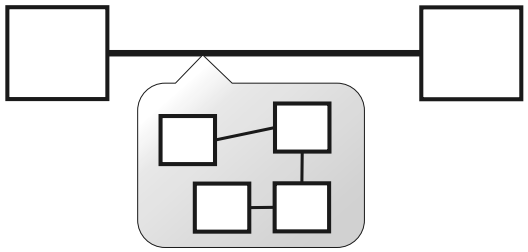
- Treat the design of interactions as orthogonal concern

Components vs. Connectors

- Components can be mapped to specific code artifacts (e.g., compilation units, deployment packages)
- Connectors are not usually directly visible in the code (e.g., linkage between modules, connections across the network, configurations of server addresses)
- Components can be both application-specific or application-independent (infrastructure)
- Connectors are mostly application-independent

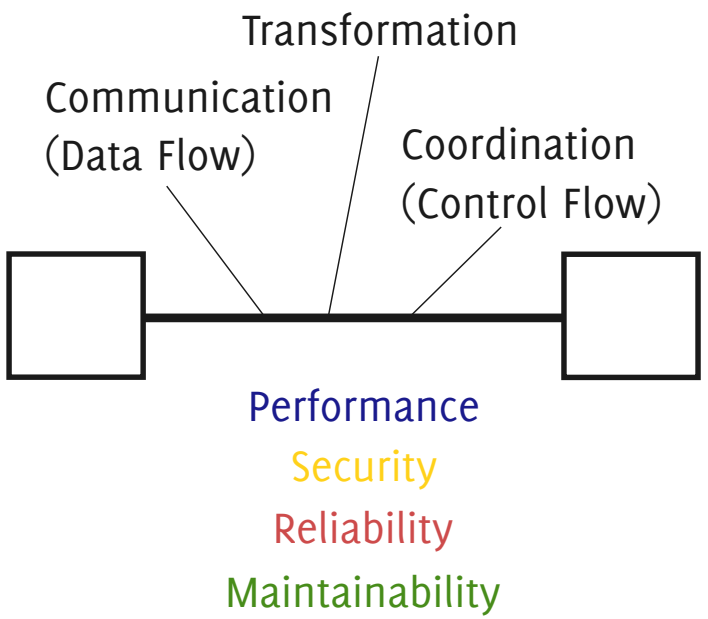
Connectors are Abstractions

- Connectors model interactions between components
- Connectors are built with (very complex) components

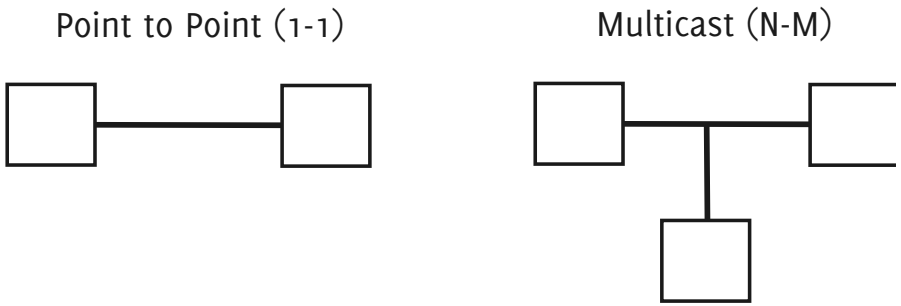


- Design Decision: when to hide away components inside a connector?

Connector Roles and Qualities



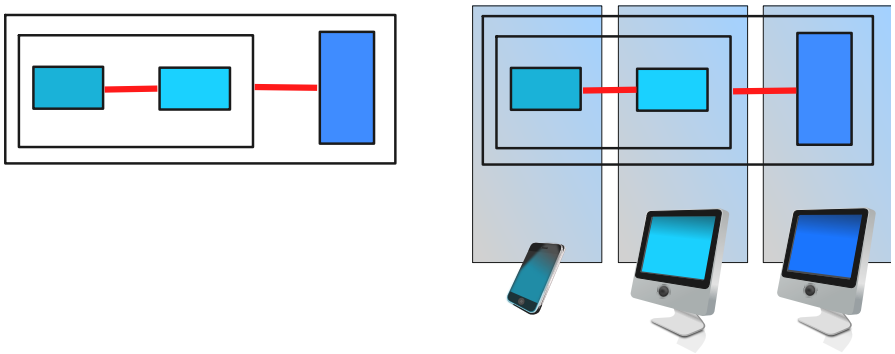
Connector Cardinality



Connectors and Bindings

- Depending on the connector, the topology of the graph of connected components may be defined at:
  - design-time (static binding)
  - deployment-time
  - or even modified at run-time (dynamic late binding).
  - very late binding

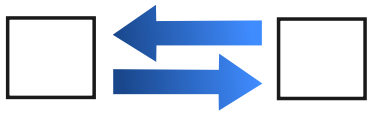
Connectors and Distribution



- Connectors define the points of distribution at design-time so that components can be deployed over multiple physical hosts at run-time

Connector Examples

RPC: Remote Procedure Call



Call

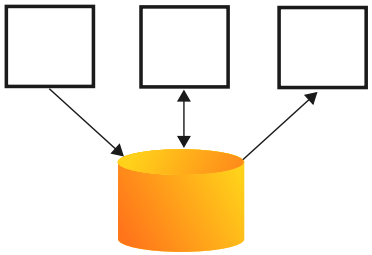
- Procedure/Function Calls are the easiest to program with.
- They take a basic programming language construct and make it available across the network (Remote Procedure Call) to connect distributed components.
- Remote calls are often used within the client/server architectural style, but call-backs are also used in event-oriented styles for notifications.

Stream

Send Receive

- Data streams let a pair of components exchange an infinite sequence of messages (discrete streams) or data (continuous streams, like video/audio)
- A streaming connector may buffer data in transit and provide certain reliability properties
- Streams are mostly used to set up pipelines in the pipe/filter architectural style

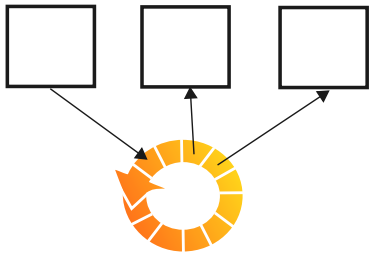
Shared Database



Create Read Update Delete

- Sharing a common database does not require to modify components, if they all can support the same schema
- Components can communicate by creating, updating and reading entries in the database, which can safely handle the concurrency

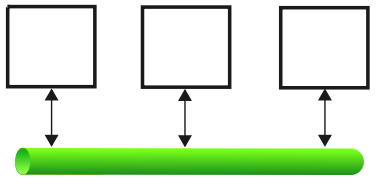
Disruptor



Next Publish WaitFor Get

- Lock Free: Single Writer, Multiple Consumers
- Cache-friendly Memory Ring-buffer
- High Throughput (Batched Reads) and Low Latency

Message Bus



Publish Subscribe Notify

- A message bus connects a variable number of components, which are decoupled from one another.
- Components act as message sources by publishing messages into the bus; Components act as message sinks by subscribing to message types (or properties based on the actual content)
- The bus can route, queue, buffer, transform and deliver messages to one or more recipients
- The “enterprise” service bus is used to implement the SOA style

File Transfer

Write Copy Watch Read

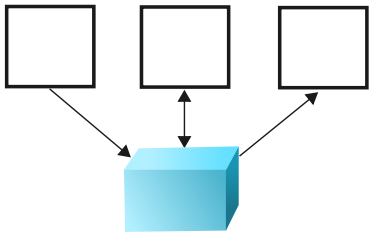
- A component writes a file, which is then copied on a different host, and fed as input into a different component.
- The transfers can be batched with a certain frequency
- Transferring files does not require to modify components

Linkage

Load Unload Call Read/Write

- Statically Linking components enables them to call each other, but also to share data in the same address space
- Dynamic linking also enables the components to be loaded and unloaded without stopping the whole system

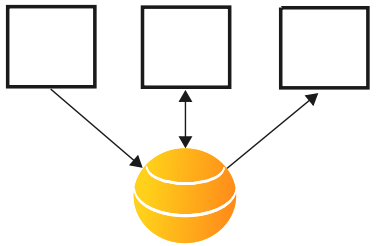
Tuple Space



In Out Rd

- A tuple space acts like a shared database, but offers a much simpler set of primitives and persistence guarantees
- Components can write tuples into the space (Out) or read tuples from it (Rd).
- Components that read tuples can also atomically take them out of the space (In)
- Extensions for the connector are available that support different kinds of synchronization (blocking or non-blocking reads), in addition to the basic data flow primitives
- A tuple space fits with the constraints of the Blackboard style and the Master/Worker pattern

Web



Get Put Delete Post

- Components reliably transfer state among themselves using the GET, PUT, DELETE primitives. POST is used for unsafe interactions.
- Components refer to each other with global addresses (URI)
- The Web is the connector used in the REST (Representational State Transfer) architectural style

References

- Richard N. Taylor, Nenad Medvidovic, Eric M. Dashofy, Software Architecture: Foundations, Theory and Practice, John-Wiley, January 2009, ISBN 978047016774
- Nikunj R Mehta, Nenad Medvidovic, Sandeep Phadke, Towards a taxonomy of software connectors, Proc. of the 22nd international conference on Software engineering (ICSE 2000), Pages 178-187.
- Andrew D. Birrell and Bruce Jay Nelson. Implementing remote procedure calls. ACM Trans. Comput. Syst. Volume 2, Number 1, Pages 39-59February 1984.
- Martin Thompson, Dave Farley, Michael Barker, Patricia Gee, Andrew Stewart, Disruptor (http://lmax-exchange.github.io/disruptor) , May 2011
- Gelernter, David. "Generative communication in Linda". ACM Transactions on Programming Languages and Systems, Volume 7, Number 1, Pages 80-112, January 1985
- D. Chappell, Enterprise Service Bus, O'Reilly, June 2004