Software Architecture in Practice

Components and Connectors

Literature

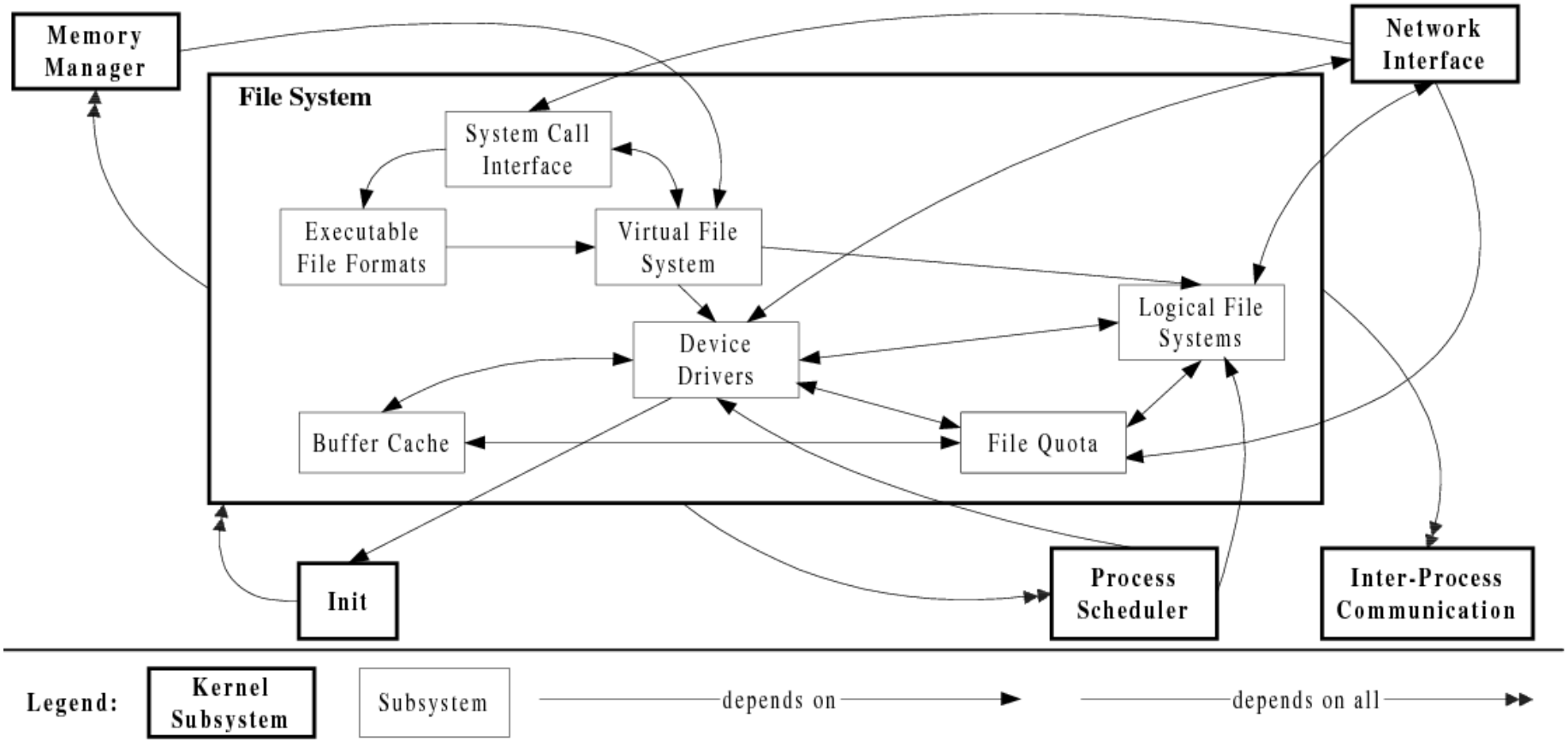
Main

- [Mehta et al., 2000]
  - Mehta, N., Medvidovic, N., and Phadke, S. (2000). Towards a taxonomy of software connectors. In Proceedings of ICSE'2000, pages 178–187.
- [Aldrich et al., 2003]
  - Aldrich, J., Sazawal, V., Chambers, C., and Notkin, D. (2003). Language support for connector abstraction. In Proceedings of ECOOP'2003, pages 74–102
- [Bass et al., 2003], chapter 18

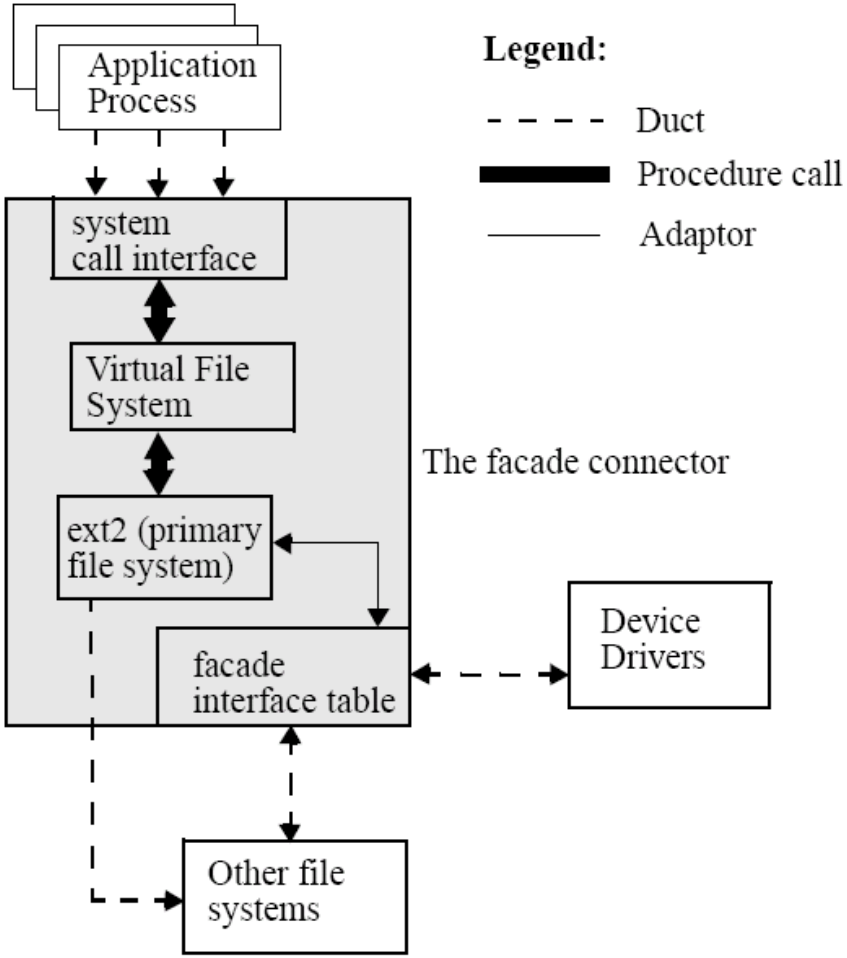
Background

- [Shaw, 1993]
  - Shaw, M. (1993). Procedure calls are the assembly language of software interconnection: connectors deserve first-class status. In Proceedings of Workshop on Studies of Software Design

What is this?


Memory Manager
Network Interface
File System
System Call Interface
Executable File Formats
Virtual File System
Logical File Systems
Device Drivers
Buffer Cache
File Quota
Init
Process Scheduler
Inter-Process Communication
Legend:
Kernel Subsystem
Subsystem
depends on
depends on all

Linux File Façade Connector



Components – What are they?

[Szyperski, 1998]

- A software component is
  - "… a unit of composition with contractually specified interfaces and explicit context dependencies only. A software component can be deployed independently and is subject to composition by third parties"
- Technical part
  - Unit of independent deployment
  - Contractually specified interfaces – provided and required
  - Subject to composition
- Market-related part
  - Third party involved

What in Java would correspond to a component?

[Bass et al., 2003] and others

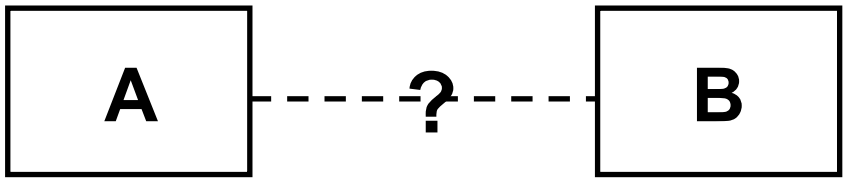
- Components as in "Component & Connectors"
- Components as principal units of computation

Connectors – What are they?

Connectors mediate interactions among components

- *they establish the rules that govern component interaction*
- *they specify any auxiliary mechanisms required*

[Shaw & Garlan, 1996]



Too vague…

Viewpoints on connectors

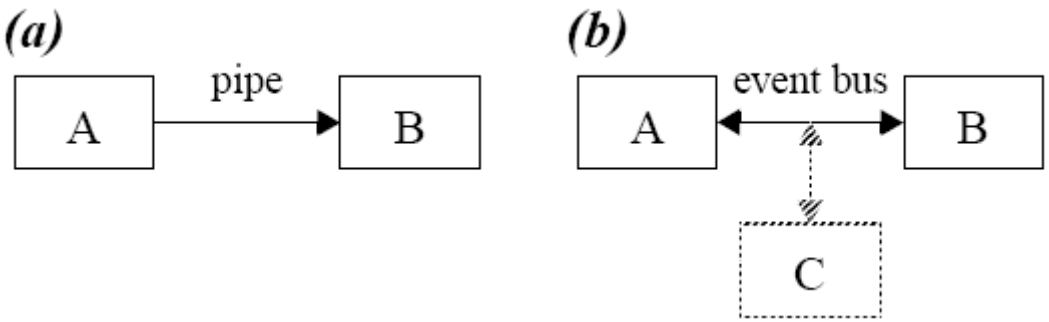
Two “views”

- [Bass et al., 2003] and others
  - Connectors as in “Component & Connectors” viewpoint
  - Communication vehicles among components
- [Mehta et al., 2000] and others
  - Components and connectors central in software architecture

[Mehta et al., 2000] – Motivation (1)

Connectors are important

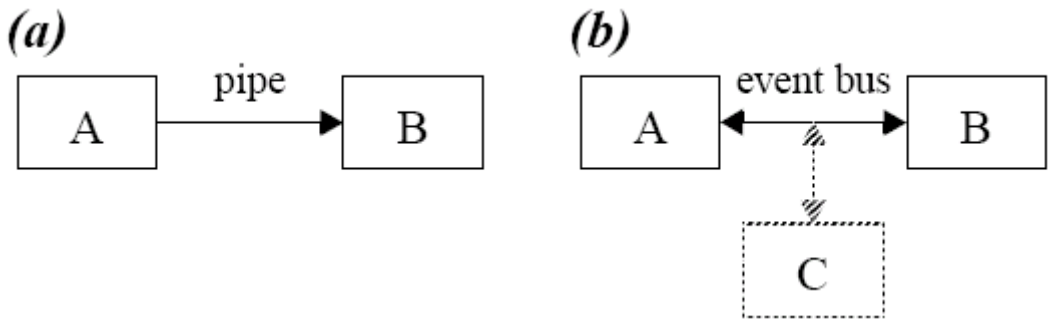
- Architecture embody a large number of interaction mechanisms/ connectors
  - E.g., shared variable access, buffers, linker instructions, procedure calls, networking protocols, pipes, SQL links
- Connectors key determinants of many system properties/quality attributes
  - E.g., performance, resource utilization, global rates of flow, scalability, reliability, security, evolvability
- Connectors may abstract away heterogeneous interaction mechanisms



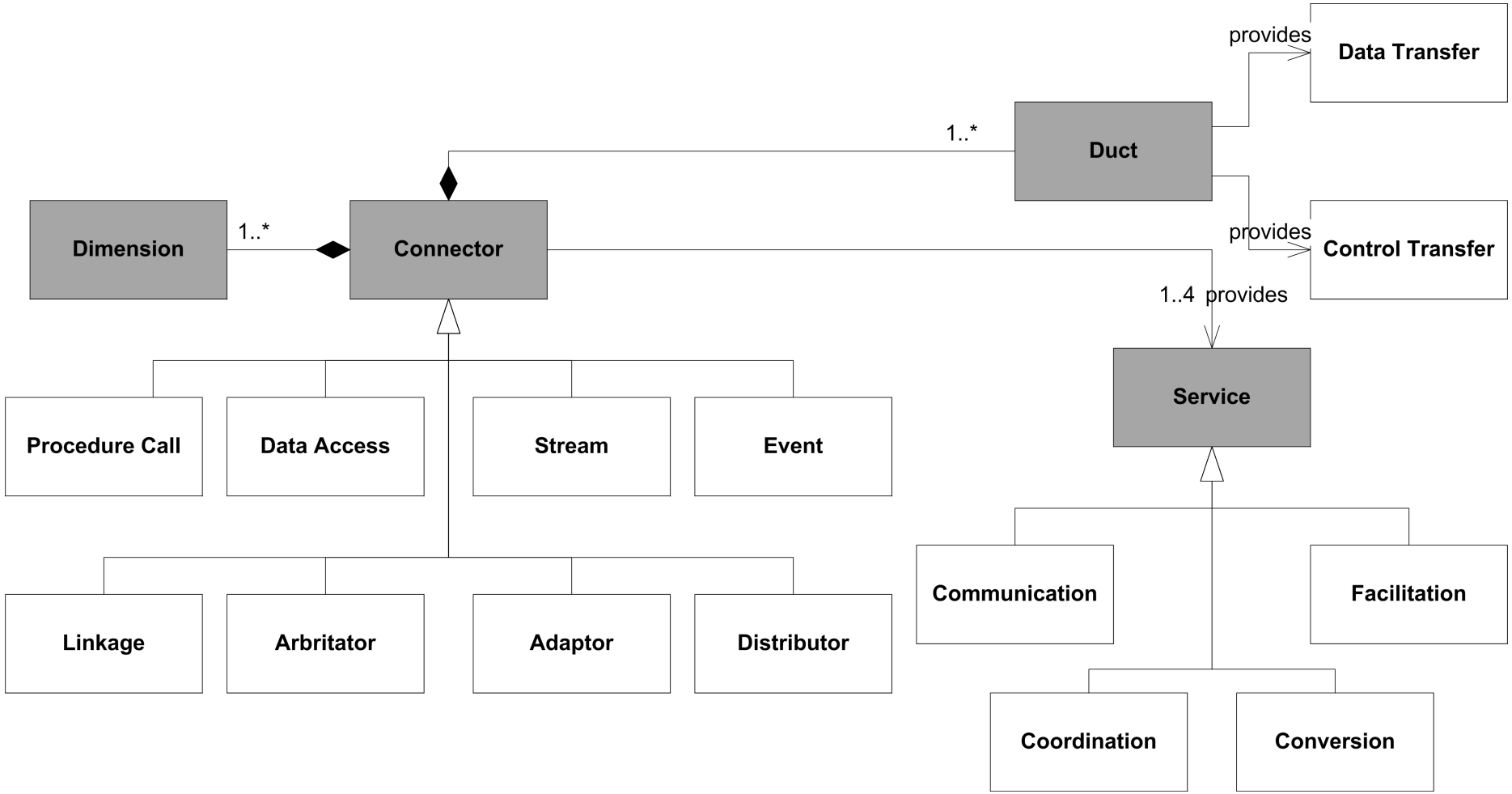
[Mehta et al., 2000] – Motivation (2)

Connectors are insufficiently treated in research and practice

- Composition of systems from components is commonplace
  - Heterogeneous, complex functionality, complex interaction
- Component research focuses on functionality
  - Interaction details hidden inside components
- Lack of understanding of fundamental building blocks of interaction



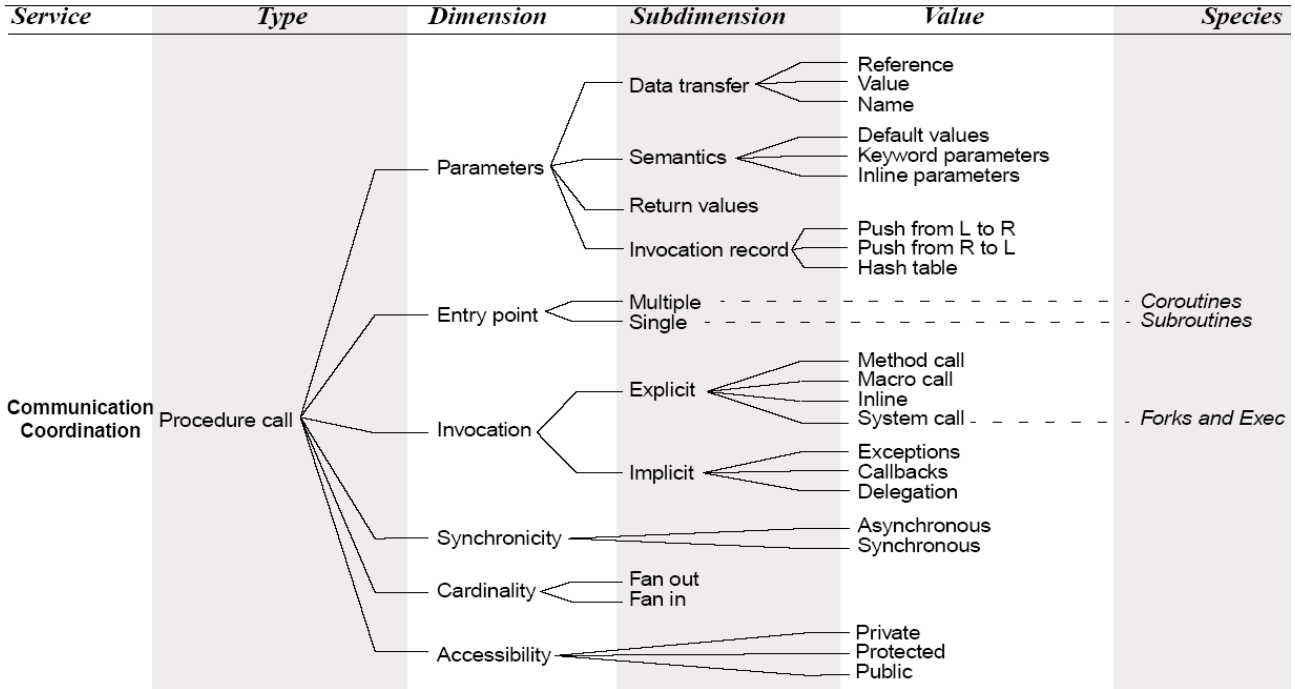
Connector Classification Ontology



Procedure Calls

Model the flow of control among components through invocation techniques

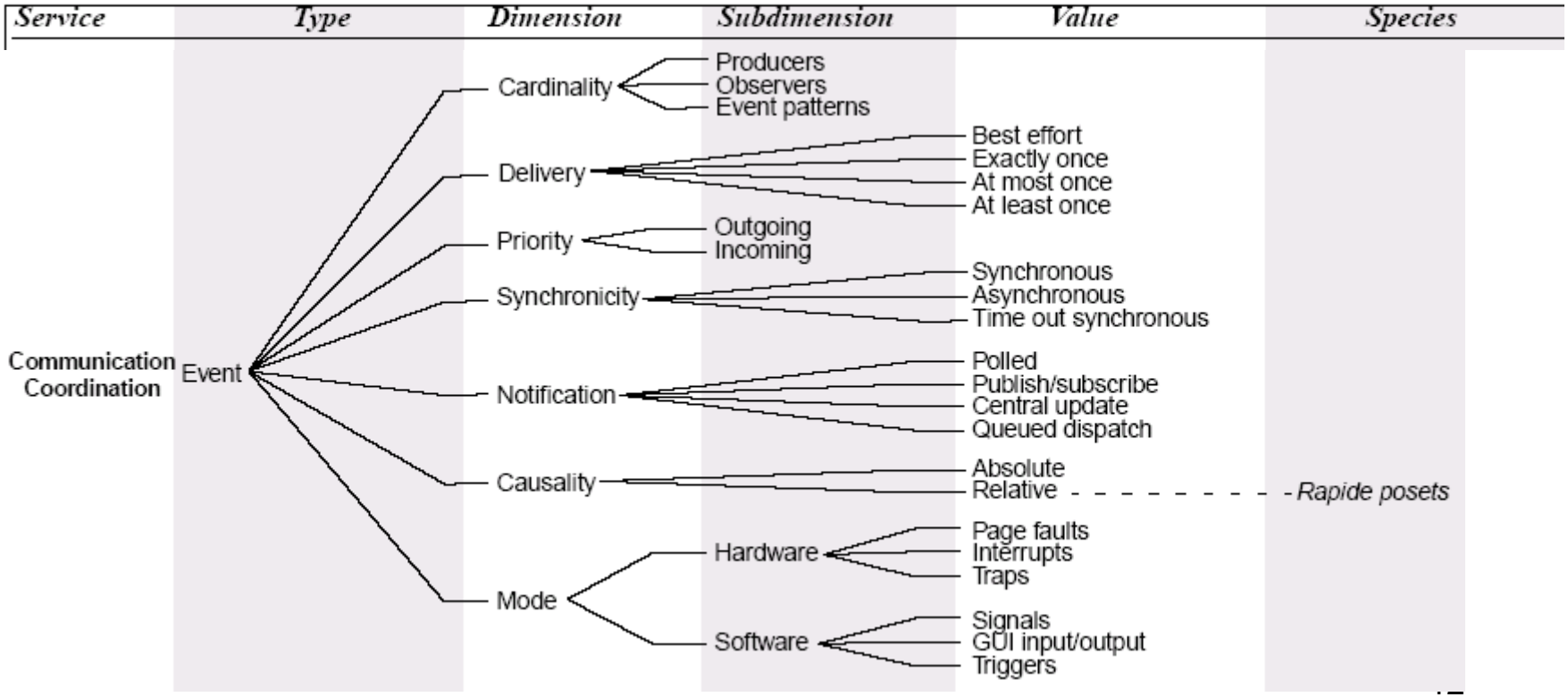
- Data transfer using parameters
- “Assembly language of software interconnection” [Shaw, 1993]



Events

"The instantaneous effect of the … termination of an operation on an object

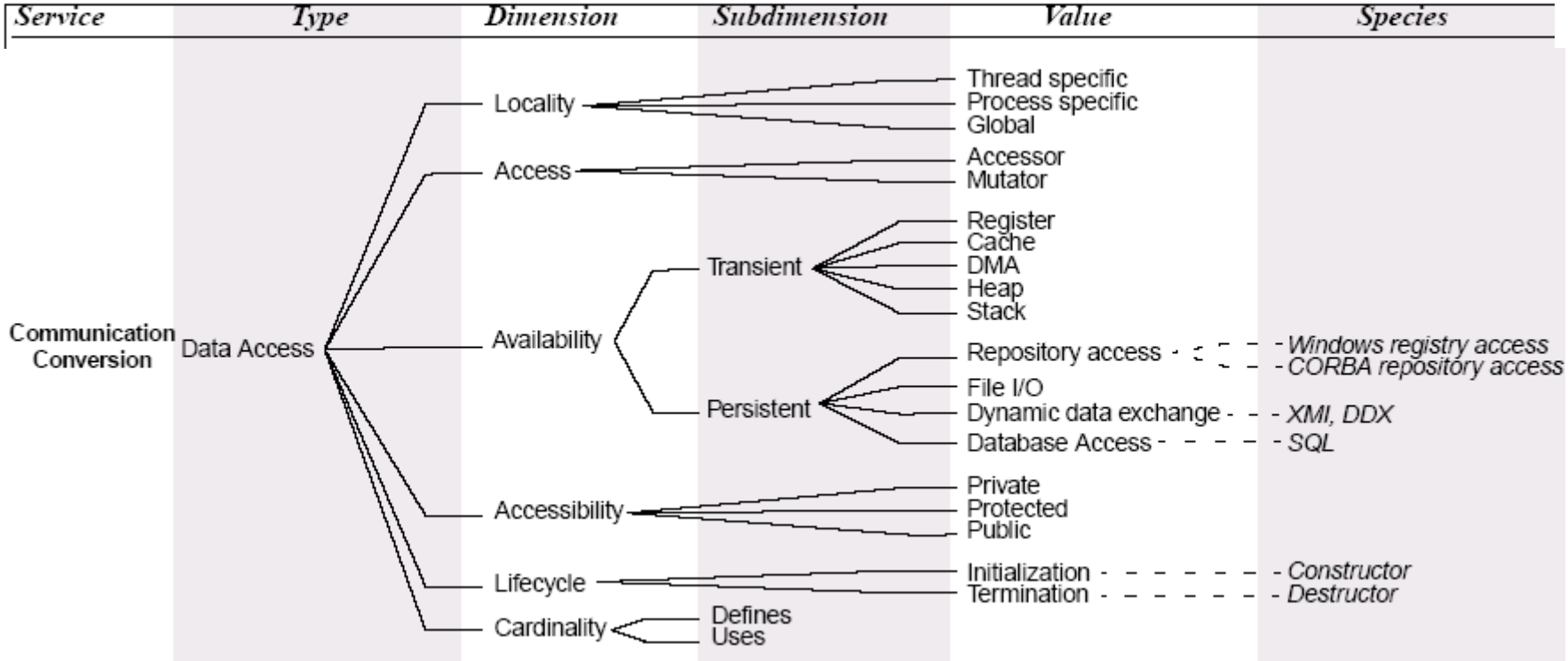
– *and it occurs at that object's location"* [Rosenblum & Wolf, 1997]



Data Access

Allow components to access data maintained by a data store component

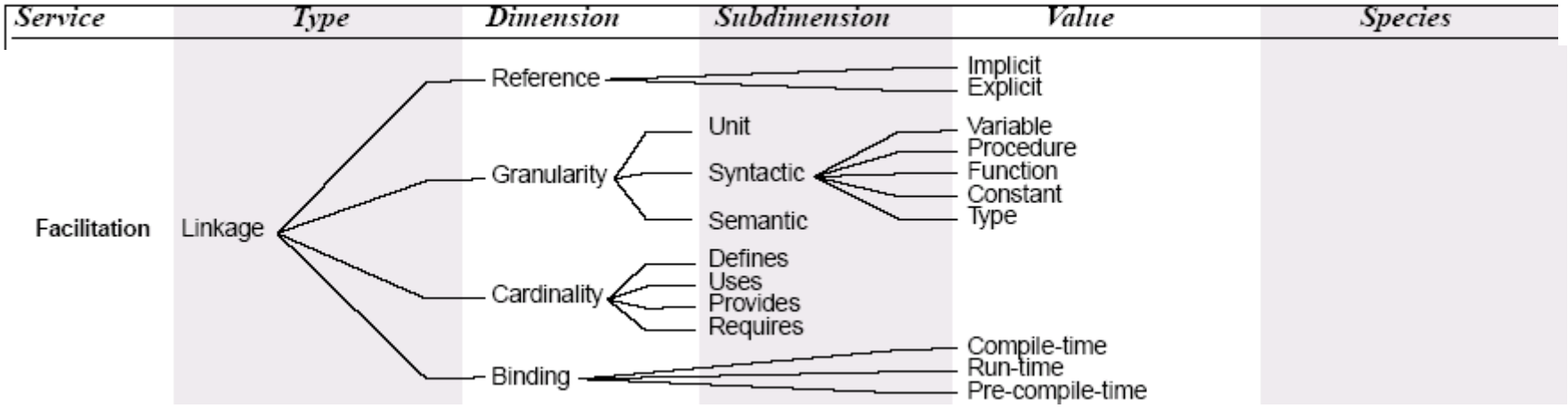
- May perform translation of information to and from components accessing



Linkage

Used to tie system components together during their operation

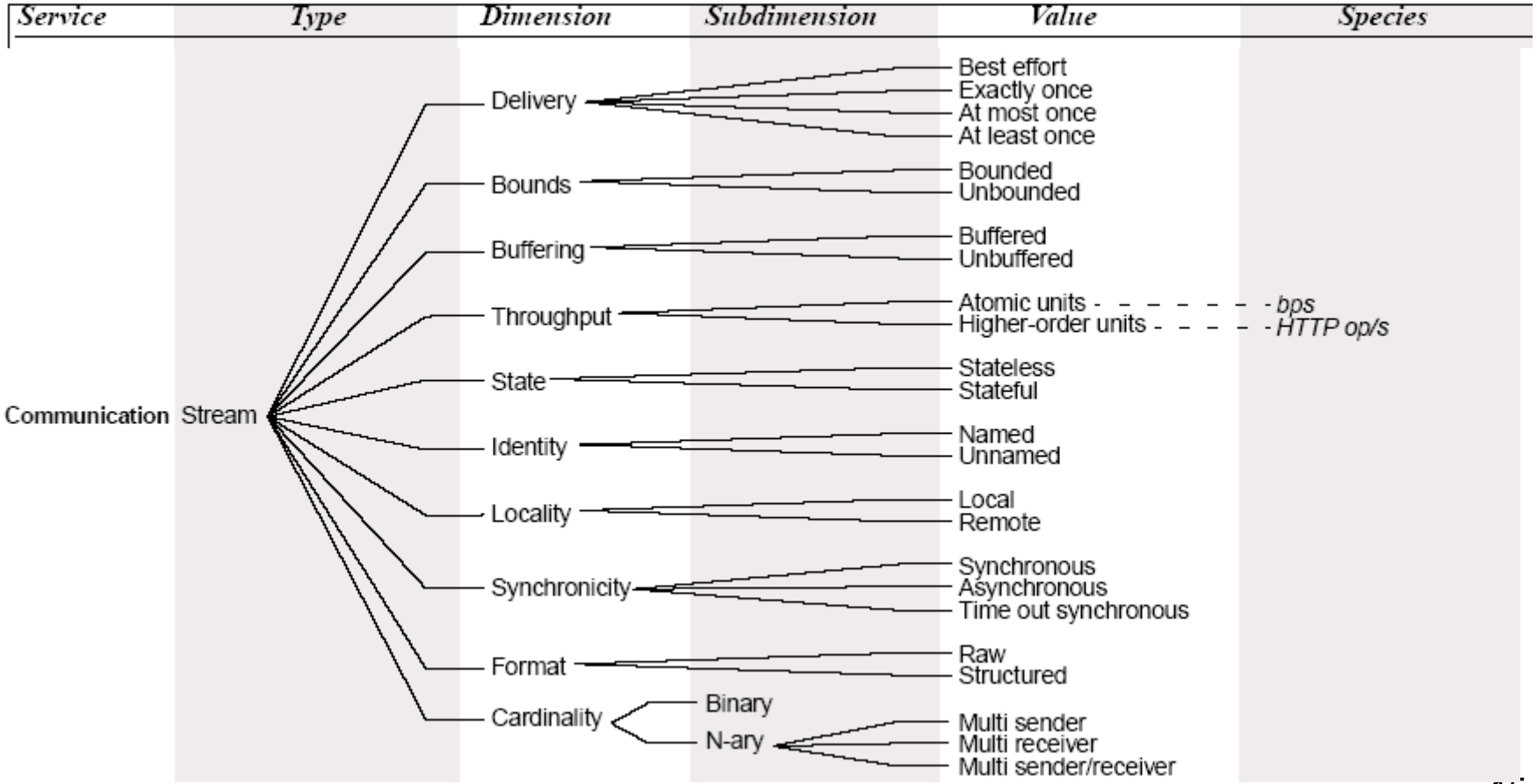
– Often disappear after being bound



Streams

Perform data transfer between autonomous processes

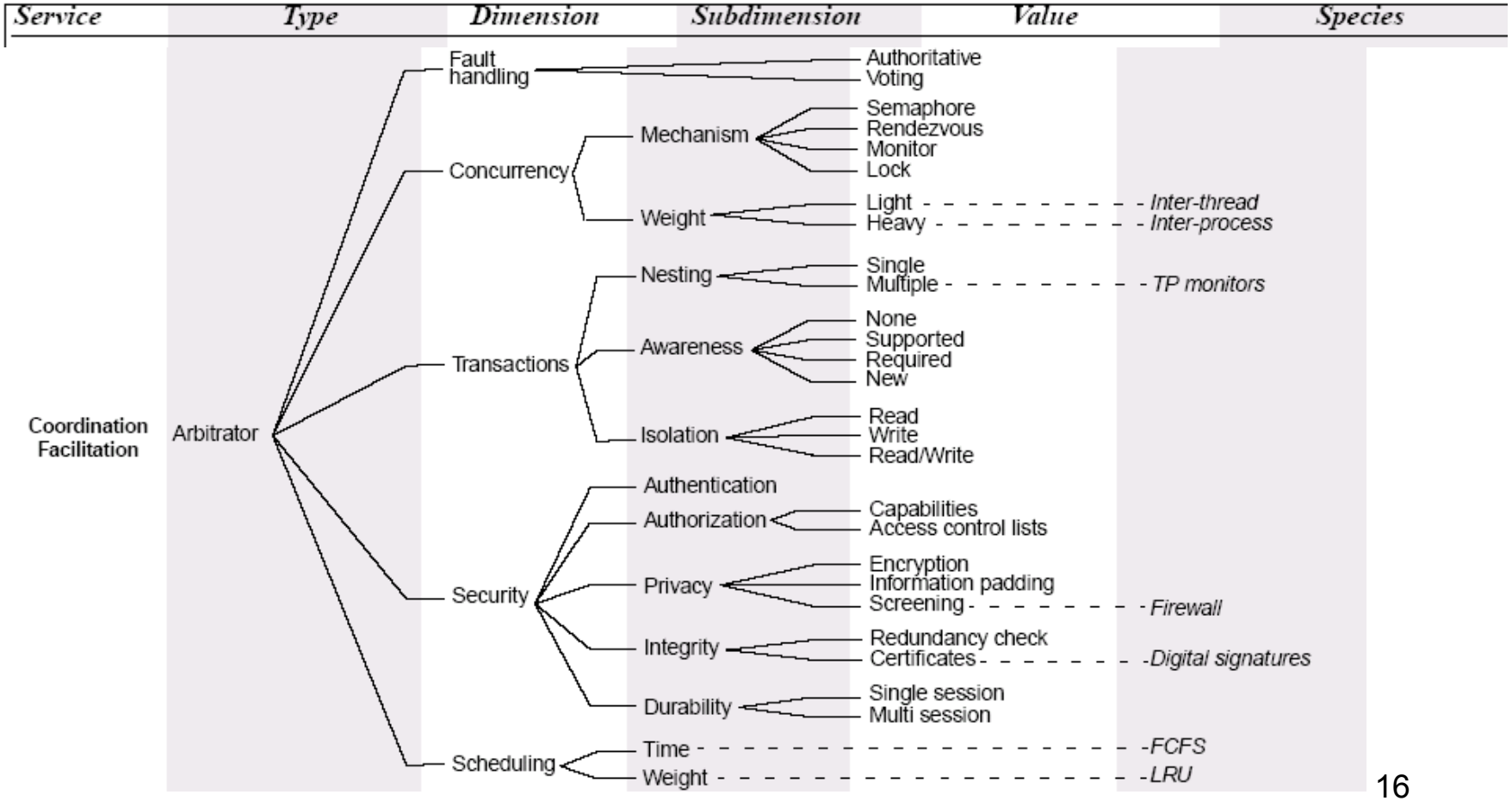
– E.g., UNIX pipes, TCP connectors, client/server protocols



Arbitrators

Streamline system operation

- Resolve any conflicts
- Redirect the flow of control



Adaptors

Support interaction between components that have not been designed to interoperate

- Matching communication policies and interaction protocols

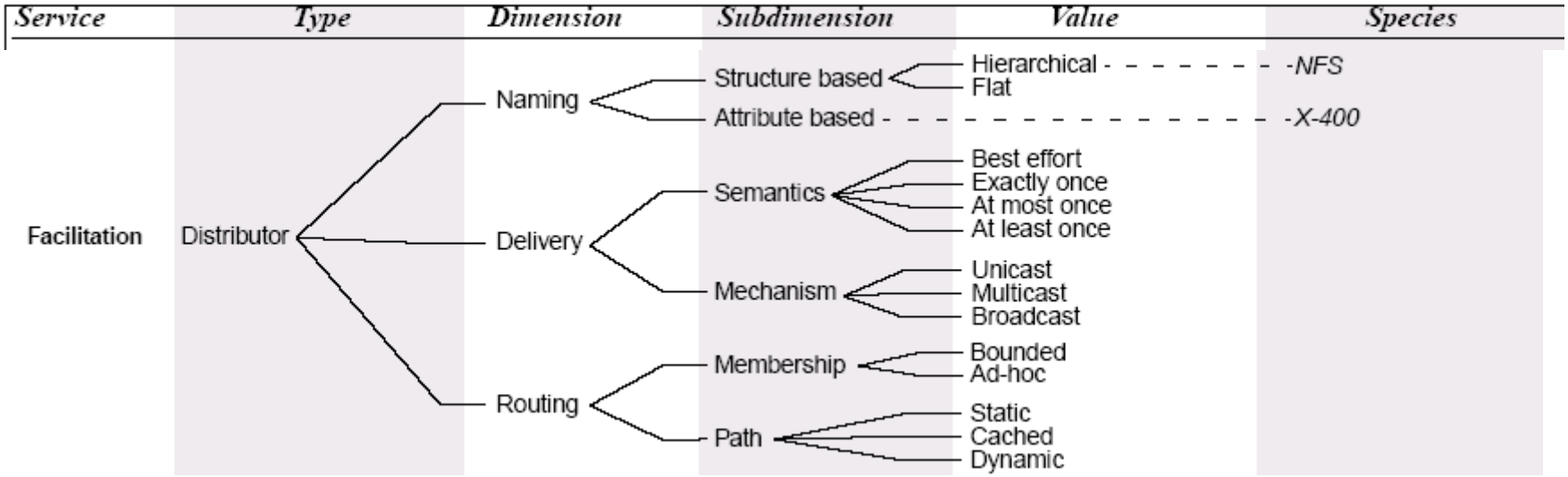
Service	*Type*	*Dimension*	*Subdimension*	*Value*	*Species*
Conversion	Adaptor	Invocation conversion	Address mapping		*V tables*
			Marshalling		*LRPC*
			Translation		*Interpreters*
		Packaging conversion		Wrappers	
				Packagers	*DeLine packaging*
		Protocol conversion			*Yellin&Strom adaptors*
					C2 domain translators
		Presentation conversion			*Internationalization*
					Clipboard access

Distributors

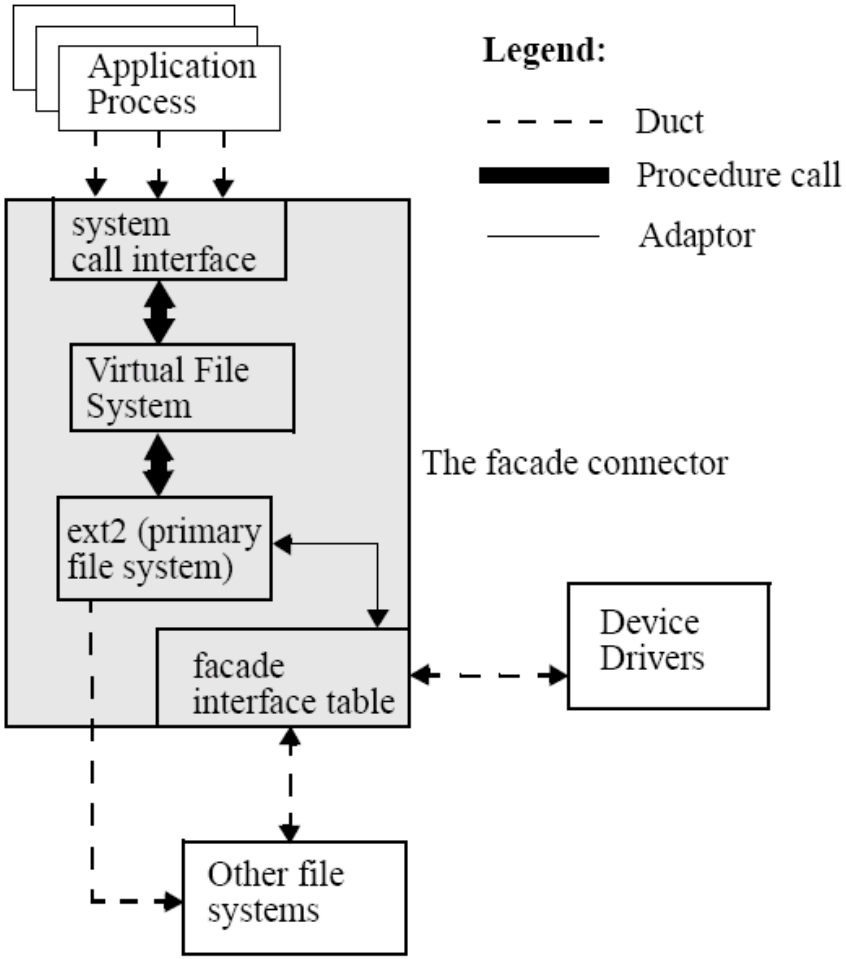
AARHUS UNIVERSITET

Perform identification of interaction paths

Route communication and coordination information among components along paths



Linux File Façade Connector



Connector Type	Dimension	Value
Arbitrator	Authorization	Access Control Lists
Arbitrator	Isolation	Read/Write
Adaptor	Invocation conversion	Translation

Why First-Class Connectors?

"First-class" XYZ
- XYZ may be used as a value
- Passed to methods, assigned to a variable, ...

[Shaw, 1993]
- Connectors may be complex
  - Elaborate definition, specifications
  - No component natural home for this
- Definition should be localized
  - Good design requires single place for definition of interaction
- Some information about the system does not belong in components
- Connectors are potentially abstract
  - Connectors may be application-independent/reusable
  - Parameterizable
- Connectors may require distributed system support
- Components should be independent
  - Component definitions should not describe how component is actually used
- Relations among components are not fixed
  - Components may be used differently by different types of connectors

[Aldrich et al., 2003]

Connections are important…

- Cf., e.g., [Mehta et al., 2000] and [Shaw, 1993]…

Languages (Java) have no explicit connections

- Components ~ classes
  - But more support needed according to ArchJava
- No support for connectors as such
  - E.g., caches, events, streams, RMI, method calls, shared variables
- Support is embedded
  - E.g., in object references, design patterns

Support for User-Defined Connectors?

Options

- Integrate connector code into components
  - Tight coupling
- Write connectors as reusable libraries
  - Often need to write generic interface
    - No type-checking
  - Many dependencies on connector code in component code
    - Hard to understand component code
- Proxy generation for remote objects
  - E.g., CORBA
  - Fixes particular semantics on distributed communication
    - Often based on synchronous method calls
- Provide explicit language support for user-defined connectors
  - *ArchJava*

ArchJava

Extension to Java

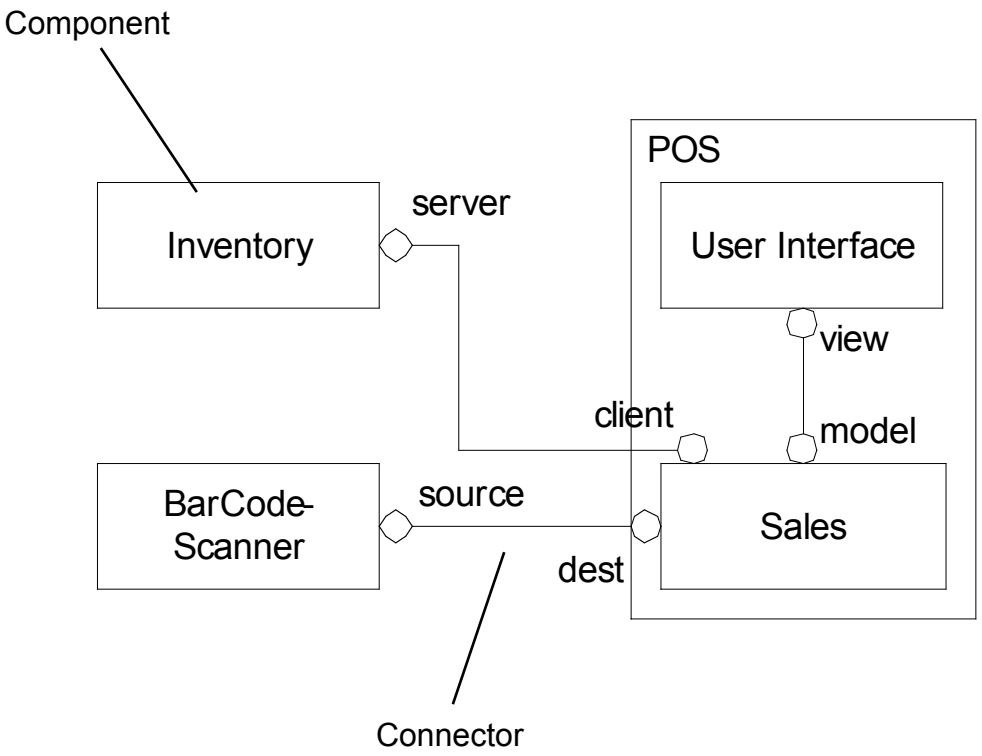
- Added keywords and libraries
- Communication integrity
  - Components communicate as specified by connections in architecture
  - Enforced by type system
- Architecture updated as code evolves
  - Architecture-as-implemented = architecture-as-designed (!)

Open source

- http://www.archjava.org/
- Integrated with AcmeStudio
- Also stand-alone Eclipse “plug-in”

Example…

POS in ArchJava



The User Interface Component (1)

User Interface

```
public component class UserInterface {
```

Component class

- Defines architectural object
- Must obey architectural constraints

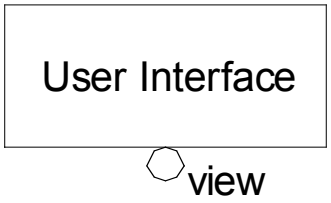
The User Interface Component (2)


```
public component class UserInterface {
  public port view {
    requires String getData();
    requires void setData(String data);
    provides void updated();
  }
```

Components communicate through *ports*

- A two-way interface
- Define *provided* and *required* methods

The User Interface Component (3)

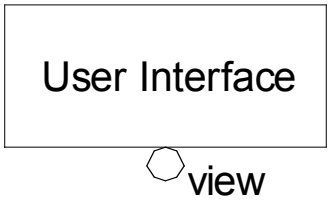



```
public component class UserInterface {
  public port view {
    requires String getData();
    requires void setData(String data);
    provides void updated();
  }
```

Ordinary (non-component) objects

- Passed between components
- Sharing is permitted
- Can use just as in Java

The User Interface Component (4)

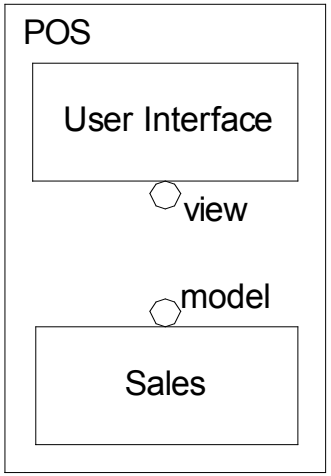



```
public component class UserInterface {
  public port view {
    requires String getData();
    requires void setData(String data);
    provides void updated();
  }

  public void updated() {
    String current = view.getData();
    System.out.println("Current data: " + current);
  }
}
```

Can fill in architecture with ordinary Java code

Hierarchical Composition (1)


```
public component class POS {
  private final Sales sales = new Sales();
  private final UserInterface userInterface
    = new UserInterface();
```



Subcomponents

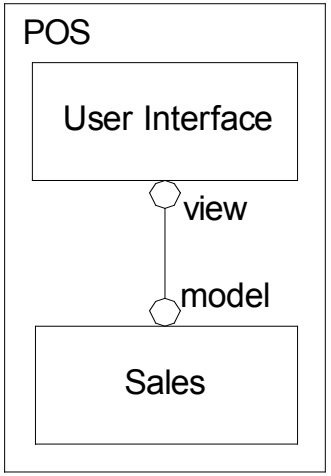
- Component instances inside another component
- Communicate through connected ports

Hierarchical Composition (2)


```
public component class POS {
  private final Sales sales = new Sales();
  private final UserInterface userInterface
    = new UserInterface();
  connect pattern Sales.model,UserInterface.view;
```

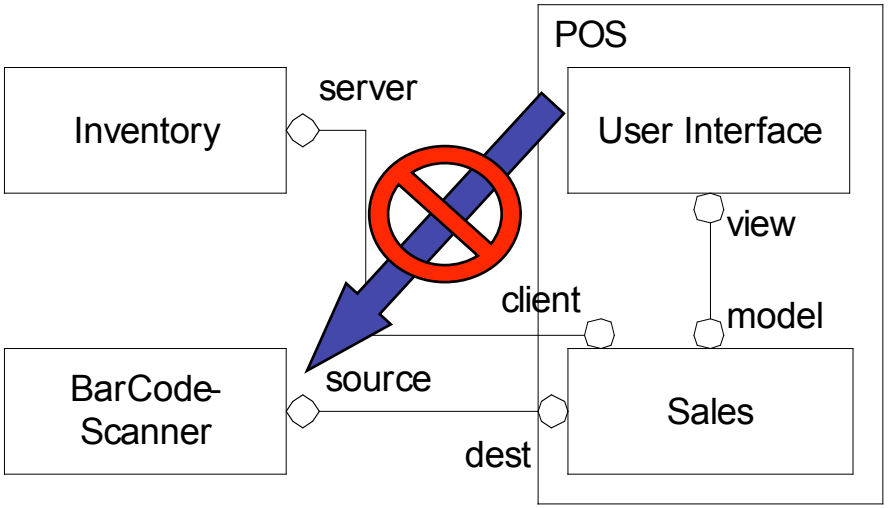
Connections

- Bind required methods to provided methods
- Required methods must match exactly provided methods



Communication Integrity

A component may only communicate with the components it is connected to in the architecture



ArchJava enforces communication integrity

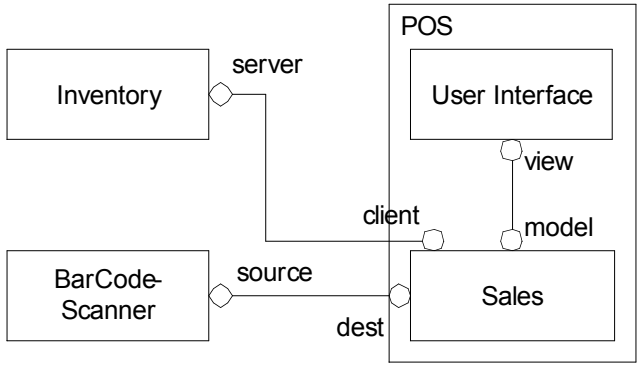
No method calls permitted from one component to another except

- From a parent to its nested subcomponents
- Through connections in the architecture

Components are not allowed as arguments

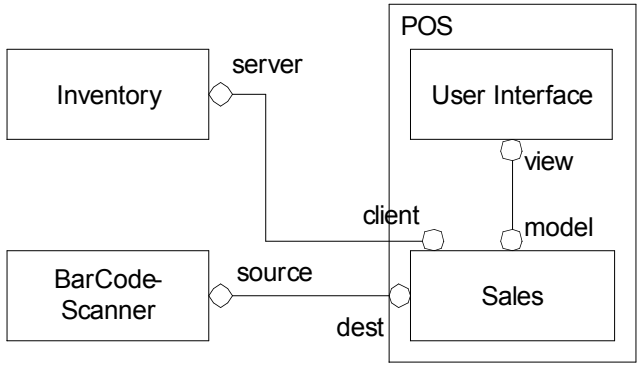
POS.archj


```
package pos.simple;

public component class POS {
    private final Sales sales = new Sales();
    private final UserInterface userInterface = new UserInterface();

    connect pattern Sales.model, UserInterface.view;

    public POS() {
      connect(sales.model, userInterface.view);
    }

    public void run() {
      sales.setData("Software Architecture in Practice, 2nd Edition");
    }

    public static void main (String[] args) {
      (new POS()).run();
    }
}
```



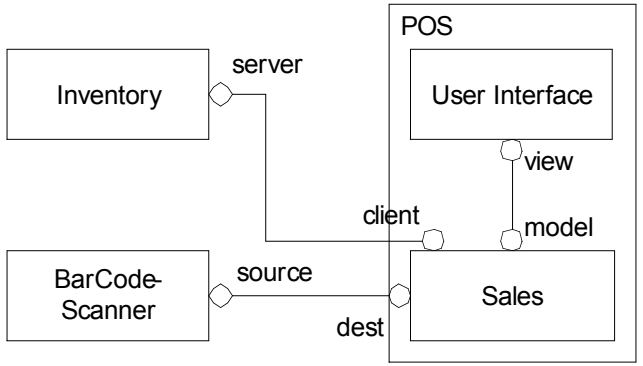
Sales.archj


```
package pos.simple;

public component class Sales {
    private String data;

    public port model {
      provides String getData();
      provides void setData(String data);
      requires void updated();
    }

    public String getData() {
      return data;
    }

    public void setData(String data) {
      this.data = data;
      model.updated();
    }
}
```

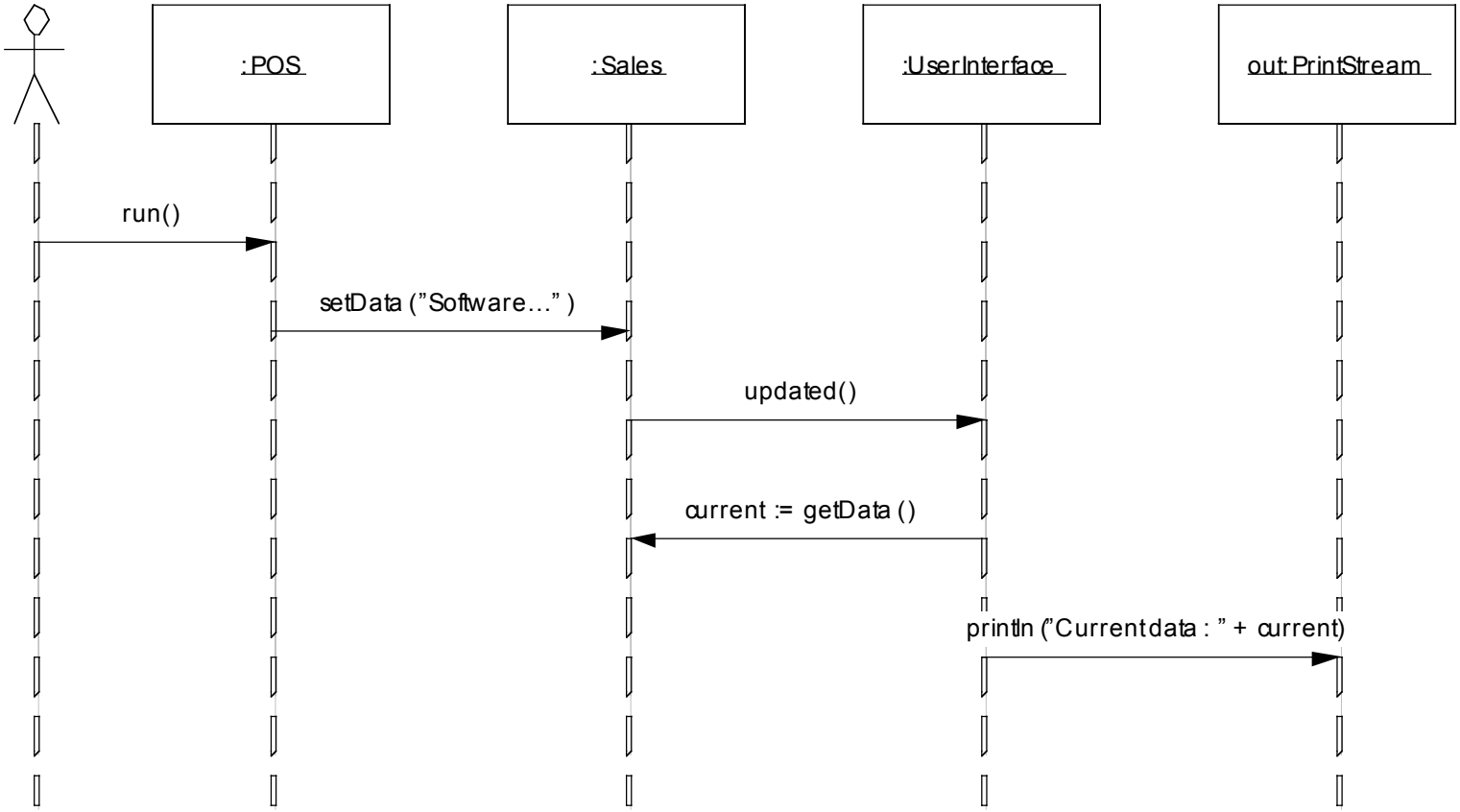


UserInterface.archj


```
package pos;

public component class UserInterface {
    public port view {
      requires String getData();
      requires void setData(String data);
      provides void updated();
    }

    public void updated() {
      String current = view.getData();
      System.out.println("Current data: " + current);
    }
}
```



Interaction (1)



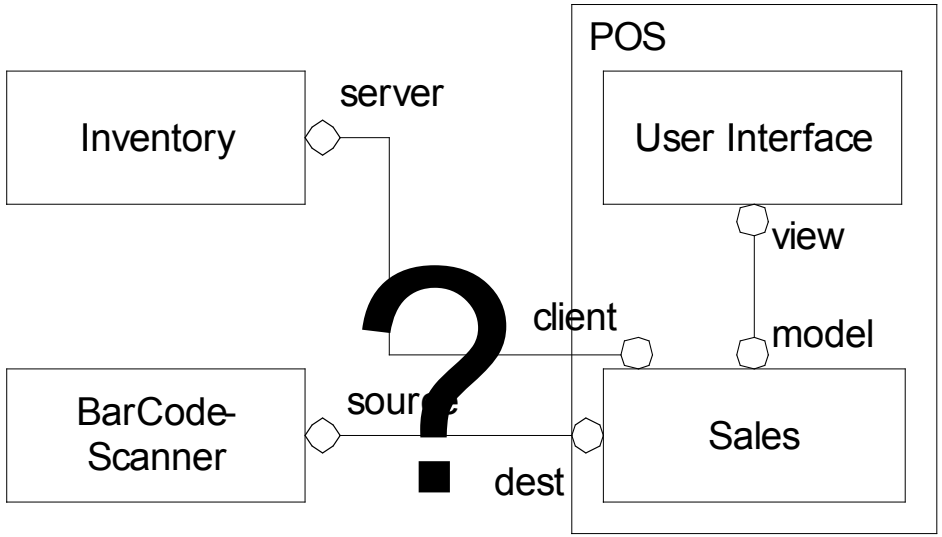
Interaction (2)

Terminal — bash — 92×18

```
odin:ArchJavaPos klausmariushansen$ ./archj pos/simple/*.archj
odin:ArchJavaPos klausmariushansen$ java -cp ../ArchJava/lib/archj.jar:. pos.simple.POS
Current data: Software Architecture in Practice, 2nd Edition
odin:ArchJavaPos klausmariushansen$
```

But…

What about the connectors among distributed components?



Let's assume that the protocols for communicating with Inventory and BarCode-Scanner run over TCP/IP…

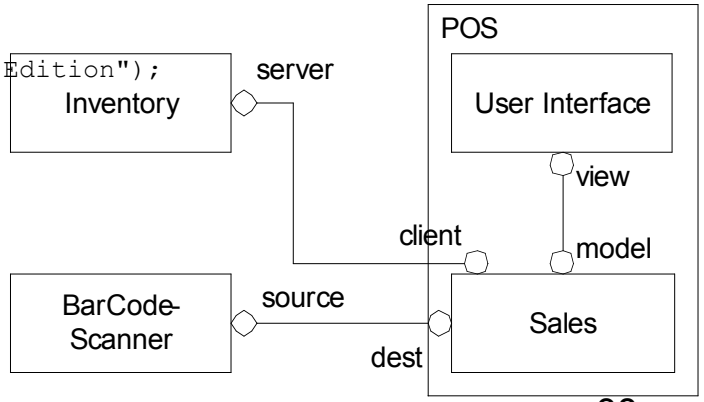
- Since there is a predefined TCP/IP connector in ArchJava ;-)

New POS.archj


```
package pos;
…
public component class POS {
    …
    private final Sales sales = new Sales();
    private final UserInterface userInterface = new UserInterface();

    connect pattern Sales.model, UserInterface.view;
    connect pattern Sales.client, Inventory.server
        with TCPConnector {
        connect(Sales sender) throws Exception {
            return connect(sender.client, Inventory.server)
                    with new TCPConnector(connection, InetAddress.getByName(JDBC_SERVER_ADDRESS),
                                          JDBC_SERVER_PORT, JDBC_SERVER_NAME);
        }
    };

    public POS() {
        connect(sales.model, userInterface.view);
    }

    public void run() {
        sales.setData("Software Architecture in Practice, 2nd Edition");
    }

    public static void main (String[] args) {
        (new POS()).run();
    }
}
```

Custom connector

Type reference
to remote component



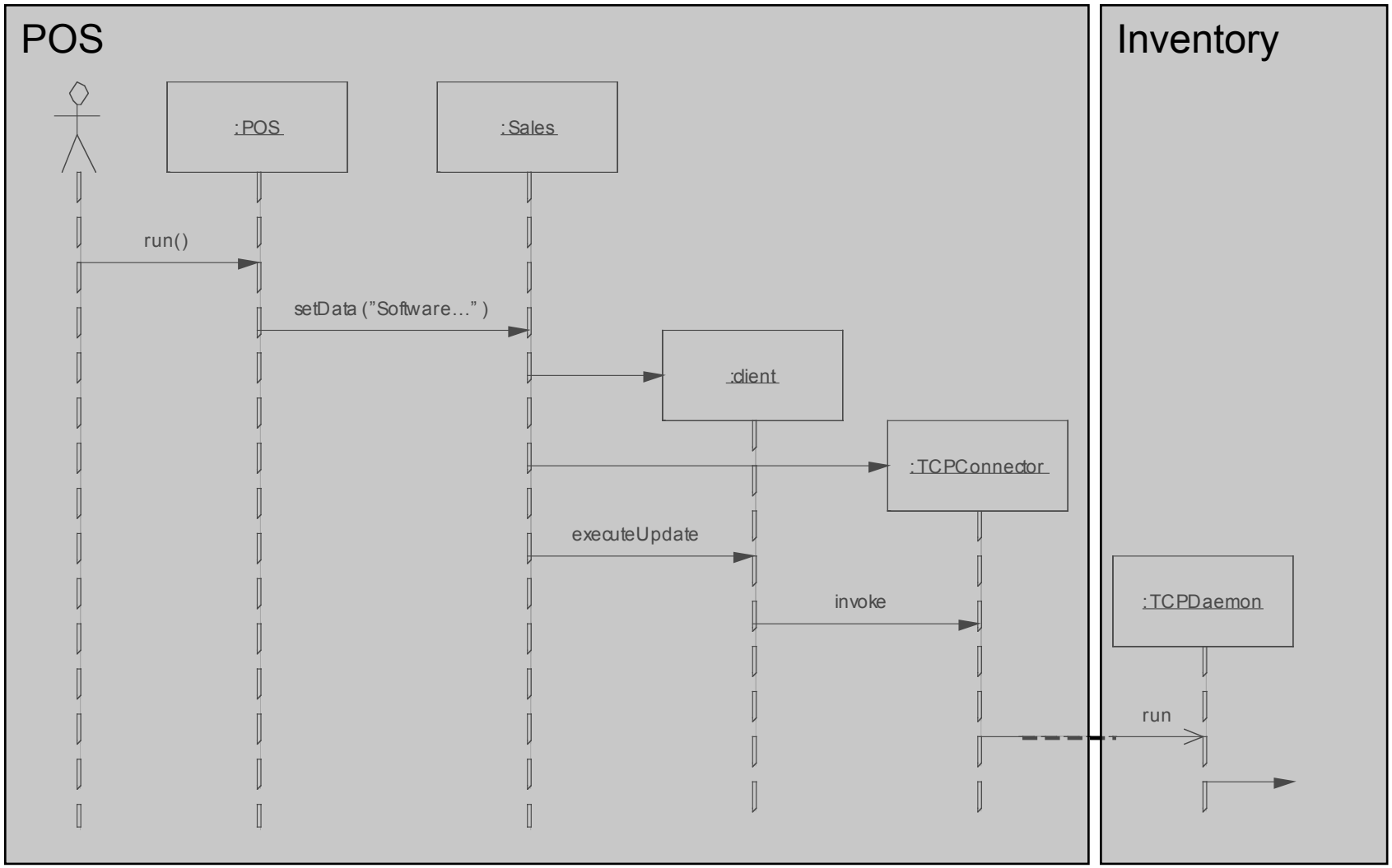
New Sales.archj


```
package pos;
…
public component class Sales {
    private String data;

    public port model {
        provides String getData();
        provides void setData(String data);
        requires void updated();
    }

    public port interface client {
        requires connect() throws Exception;
        requires String executeUpdate(String statement);
    }

    public String getData() {
        return data;
    }

    public void setData(String data) {
        try {
            this.data = (new client()).executeUpdate(data);
        } catch (Exception e) {
            e. printStackTrace();
        }
        model.updated();
    }
}
```

Inventory.archj


```
package pos;
…
public component class Inventory {
    public port interface server {
      provides String executeUpdate(String statement);
    }

    public String executeUpdate(String statement) {
      return statement + " (validated)";
    }

    public Inventory () {
      try {
          TCPConnector.registerObject(this, POS.JDBC_SERVER_PORT,
                                      POS.JDBC_SERVER_NAME);
      } catch (IOException e) {
          e.printStackTrace();
      }
    }

    public static void main(String[] args) {
      new Inventory();
    }
}
```

Interaction (1)



Interaction (2)

Terminal — java — 91×8

```
odin:ArchJavaPos klausmariushansen$ java -cp ../ArchJava/lib/archj.jar:. pos.Inventory
```

Terminal — java — 91×8

```
odin:ArchJavaPos klausmariushansen$ java -cp ../ArchJava/lib/archj.jar:. pos.POS
Current data: Software Architecture in Practice, 2nd Edition (validated)
```

A User-Defined Connector


```
public class TCPConnector extends Connector {
        // data members
        protected TCPEndpoint endpoint;
        // public interface
        public TCPConnector(Connection connection, InetAddress host, int prt, String objName) throws IOException {
                        …
        }

        public Object invoke(Call call) throws Throwable {
                        Method meth = call.getMethod();
                        return endpoint.sendMethod(meth.getName(), meth.getParameterTypes(), call.getArguments());
        }

        public static void registerObject(Object o, int prt, String objName) throws IOException {
                        TCPDaemon.createDaemon(prt).register(objName, o);
        }
        // interface used by TCPDaemon
        TCPConnector(TCPEndpoint endpoint, Object receiver, String portName) {
                        super(new Object[] { receiver }, new String[] { portName });
                        this.endpoint = endpoint;
                        endpoint.setConnector(this);
        }
        Object invokeLocalMethod(String name, Type parameterTypes[], Object arguments[]) throws Throwable {
                        // find method with parameters that match parameterTypes
                        Method meth = findMethod(name, parameterTypes);
                        return meth.invoke(arguments);
        }
        // typechecking semantics defined on next slide
}
```

Type-Checking


```
public class TCPConnector extends Connector {
        public static Error[] typecheck(Connection c) {
                // First invoke the default Java typechecker
                Error [] errors = Connector.typecheck(c);
                if (errors.length > 0)
                        return errors;
                // ensure all arguments and results are Serializable
                Type serializable = Type.forName("java.lang.Serializable");
                for (int pI = 0; pI < c.getPorts().length; ++pI) {
                        for (int mI = 0; mI < c.getPorts()[pI].getMethods().length; ++mI){
                                Method method = c.getPorts()[pI].getMethods()[mI];
                                Type returnType = method.getReturnType();
                                if (!serializable.isAssignableFrom(returnType))
                                        return new Error[] { new Error("type not serializable", c)
                                };
                                // similar check for method arguments
                        }
                }
        }
}
```

ArchJava: Defining Connectors


```
public class Connector {
        public static Error[] typecheck(Connection c);
        public Object invoke(Call c) throws Throwable;
        public Connector();
        protected Connector(Object components[], String portNames[]);
        public final Connection getConnection();
}

public final class Connection {
        public Port[] getPorts()
        public Connector getConnector()
}

public final class Port {
        public String getName();
        public Method[] getRequiredMethods();
        public Method[] getProvidedMethods();
        public Object getEnclosingObject();
}

public final class Method {
        public String getName();
        public Type[] getParameterTypes();
        public Object invoke(Object args[]) throws Throwable;
}

public final class Type {
        public String getName();
        public boolean isAssignableFrom(Type other);
        public static Type forName(String qualifiedName);
}

public final class Call {
        public Method getMethod();
        public Object[] getArguments();
}
```

ArchJava: Evaluation

Engineering benefits

- Based on case study of a.o. PlantCare ubiquitous computing system

Expressiveness

- Based on [Mehta et al., 2000]

[Mehta et al., 2000] Types in ArchJava

Mostly maps well to ArchJava connectors

Examples

- Procedure call
  - Issues: Parameter passing, async/sync, cardinality
  - E.g.
    - AsynchronousConnector
      - » Accept incoming required method calls
      - » Return to sender immediately
      - » Invoke corresponding method asynchronously in another thread
    - SummingBroadcastConnector
    - TCPConnector
- Event
  - Issues: cardinality of producers/consumers, event priority, synchronicity, event notification mechanism
  - E.g.,
    - EventDispatcherConnector
      - » Queue events
      - » Dispatch asynchronously
- Data access
  - Issues: Initialization and cleanup, conversion and presentation of data
  - E.g.,
    - CachingConnector
      - » Store calculated result
      - » Return stored result if called with identical parameters
- Linkage
  - Compile time: outside of scope of ArchJava connectors (?)

Discussion

Is it sufficient with one viewpoint on software architecture?

- I.e., Component & Connector
- Conversely: do we loose too much by having just one viewpoint?

Does C&C *really* map well to implementation

- Or is it something different from implementation?

Scalability and performance?

ArchJava vs architectural prototyping?

- Can we, e.g., inline measurements in connectors?