PEPt
Service Oriented Architecture

Harold Carr
Sun Microsystems

harold.carr@sun.com

http://javaweb.sfbay.sun.com/~hcarr/pept/

Main Point

- **A SOA needs to be able to support multiple:**
  - **Programming models (presentation)**
  - **Encodings**
  - **Protocols**
  - **Transports**
- **PEPt**
  - **P(resenation) E(ncoding) P(rotocol) t(ransport)**
  - **Gives this ability**

Outline of Talk

- **Part 1 – WS / SOA / ESB**
  - **How .h, RPC, RMI WS leads to SOA**
  - **How SOA leads to ESB**
  - **How PEPt fits into ESB/SOA**
- **Part 2 - PEPt architecture**
  - **PEPt “requesting” side**
  - **PEPt “responding” side**
  - **Enterprise features enabled by PEPt**

Who Am I?

- **Degree in Distributed Computing – U/U 1994**
  - **Distributed C++**
    - **IEEE Parallel and Distributed Processing 1993**
  - **Distributed Scheme**
    - **Journal of LISP and Symbolic Computation 1992**

- **Joined SUN in 1994 – Project DOE/NEO**
- **OMG doing language-independent version of what I had done with DC++**
- **1st proprietary presentation, encoding, protocol and transport**
- **Then CDR/GIOP/IIOP**
- **Then Java version of NEO - "JOE"**
- **1st proprietary stubs/ties – then portable**
- **Then JAX-RPC – new programming model for different encoding/protocol/transport**
- **JAX-RPC changing: RPC to Doc-Literal and enabling JAX-FAST**

PEPt Papers & Patents

- **ACM Conference on Service Oriented Computing – 2004**
- **Conference on Communications in Computing – 2004**
- **OTM/Distributed Objects and Applications Conference – 2003**
- **ACM Middleware – 2003**
- **3 patents filed – more in process**

Point of History, Papers, Patents

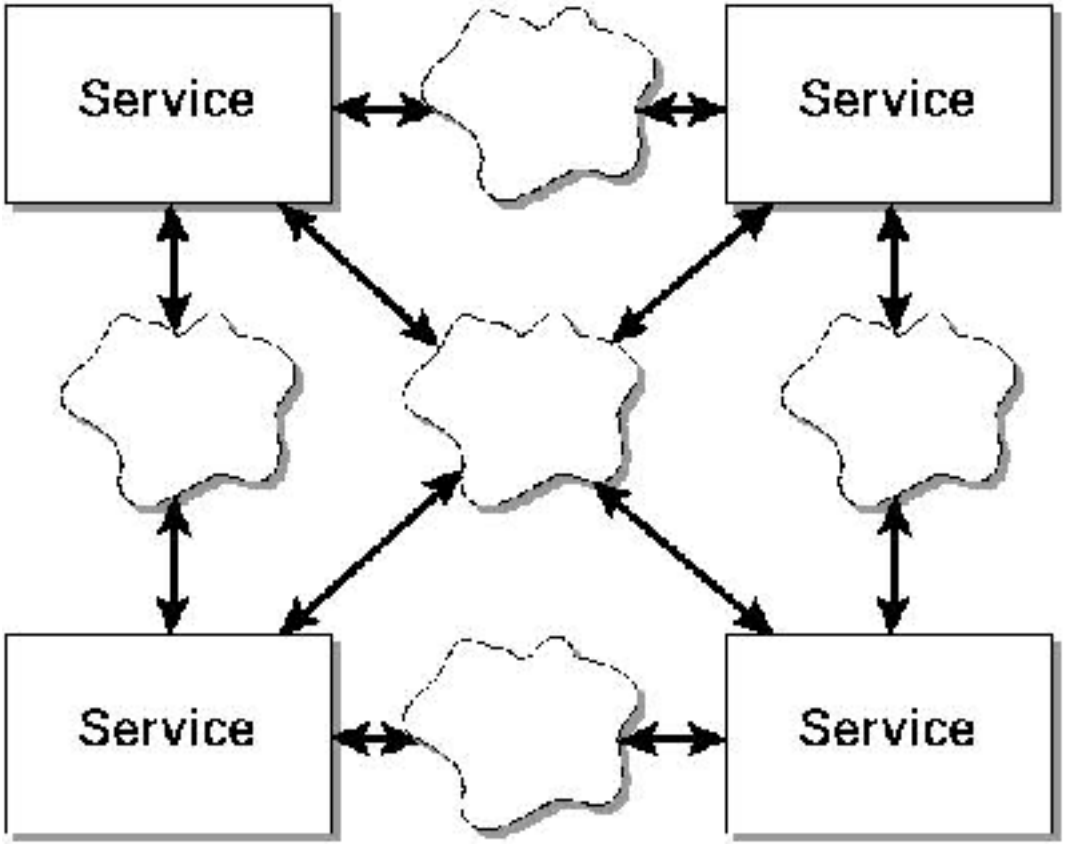
- **This is not an overnight design**
- **Result of years of experience**
  - **Design**
  - **Implement**
  - **Writing (papers, patents, doc)**
- **We've explored many of the deadends already**
  - **but not all!**

Outline of Talk

- **Part 1 – WS / SOA / ESB**
  - **How .h, RPC, RMI WS leads to SOA**
  - **How SOA leads to ESB**
  - **How PEPt fits into ESB/SOA**
- **Part 2 - PEPt architecture**
  - **PEPt "requesting" side**
  - **PEPt "responding" side**
  - **Enterprise features enabled by PEPt**

SOA Demystified

- **How RPC/RMI leads to WS**
- **How WS leads to SOA**



SOA History/(d)Evolution

- **SOA – abstract view of app or component**
- **App used as a service**
- **Not written by user – available elsewhere**
- **KEY: abstract service description**

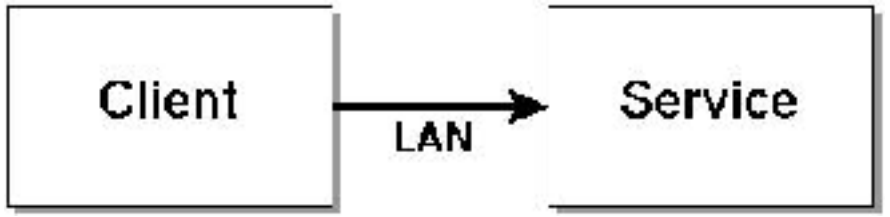
Libraries

- **When we wrote in C/C++**
- **Services called libraries**
- **Described with *.h files**
  - **and man pages**
- **Accessed via LD**
- **Mostly language-dependent**

```
extern FILE     *fdopen(int, const char *);
extern int   getw(FILE *);
extern int   putw(int, FILE *);
```

RPC

- **Services available on local network**
- **Described in IDL**
- **XDR encoding**
- **Language-independent**




```
program NFS_PROGRAM {
  version NFS_VERSION {
    void
    NFSPROC_NULL(void) = 0;

    attrstat
    NFSPROC_GETATTR(fhandle) = 1;
  } = 2;
} = 100003;
```

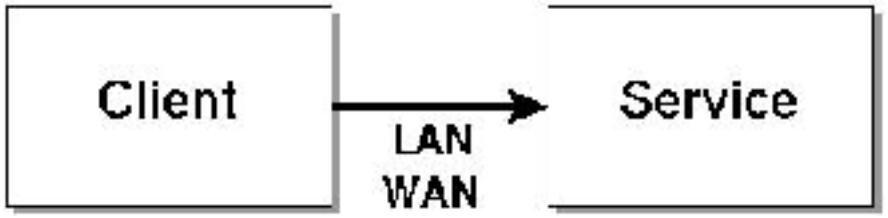
RPC++ : DCE, DCOM, CORBA

- **Services described by IDL**
- **Advertised (e.g., registered in NameService)**
- **DCE - “old-fashioned RPC”**
- **DCOM – OO version of DCE for windows**
- **CORBA- OO version of DCE - multiplatform**

```
module foo {
  struct SocketInfo {
    string host; long port;
  };
  interface I {
    boolean unregister(in SocketInfo si);
    boolean register(in SocketInfo si);
  };
};
```

RMI, RMI-IIOP

- **RMI – Java only**
- **RMI-IIOP – RMI but using GIOP/IIOP/CDR**
  - **enables non-Java clients**
  - **transactions**




```
package foo.bar;

import java.rmi.Remote;
import java.rmi.RemoteException;

public interface X extends Remote
{
    int sendBytes (byte[] x)
        throws RemoteException;
}
```

SOAP/HTTP

- **Brilliant marketing move by MS**
  - **win back turf from Java**
- **Language-independent "executable" XML on the wire**
- **HTTP : loose coupling – message exchange**
- **Note on IIOP/HTTP tunneling**

Web Services

- **Services described in WSDL**
- **Advertised (e.g., UDDI)**
- **SOAP/HTTP protocol/transport**
- **KEY POINT: Service Abstraction**
  - **(not SOAP or HTTP)**

```
<message name="GetLastTradePriceInput">
  <part name="body" element="xsd1:TradePriceRequest"/></message>

<message name="GetLastTradePriceOutput">
  <part name="body" element="xsd1:TradePrice"/></message>

<portType name="StockQuotePortType">
  <operation name="GetLastTradePrice">
    <input message="tns:GetLastTradePriceInput"/>
    <output message="tns:GetLastTradePriceOutput"/></operation></portType>

<binding ... />
```

SOA – Remoting systems

- **Description of service**
  - **WSDL, IDL, Remote**
- **Advertise/find services**
  - **UDDI, NameService**
- **Use the service**
  - **RPC/RMI or Messaging**
- **Provide service – map to business logic**
  - **POA, deployment descriptors, queues**
- **Implement a platform**
  - **XDR/RPC, CDR/IIOP, JS/JRMP, XML/SOAP/HTTP**

WS : Internet RPC

- **Current status of WS as manifestation of SOA**
- **Used in Client/Server fashion**
- **Service endpoints contacted directly**
- **Reliability (e.g., off/online) wired into service**

Note on Messaging

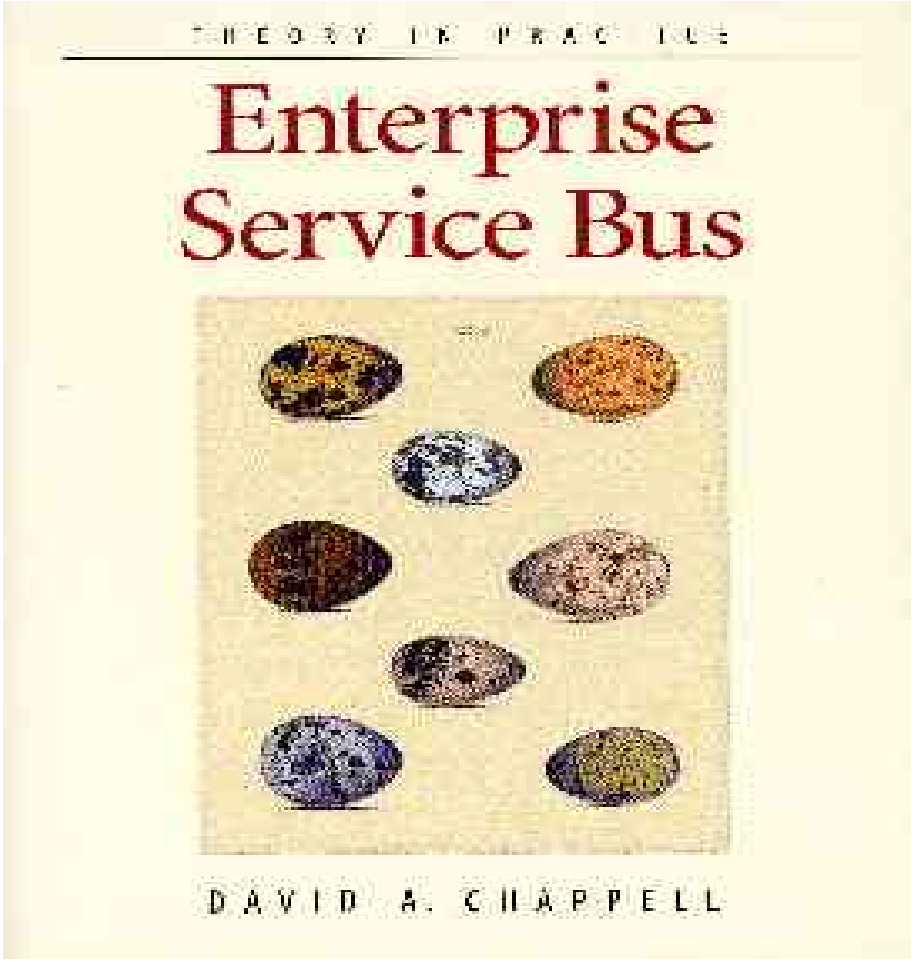
- **Async send & receive**
- **senders/receivers decoupled**
- **Talk to message system – not each other directly**
- **Reliability (e.g., store/forward) in MSG system – not apps**
- **Different programming model**
- **But same infrastructure**
  - **Encodings, protocols, transports**

Outline of Talk

- **Part 1 – WS / SOA / ESB**
  - **How .h, RPC, RMI WS leads to SOA**
  - **How SOA leads to ESB**
  - **How PEPt fits into ESB/SOA**
- **Part 2 - PEPt architecture**
  - **PEPt “requesting” side**
  - **PEPt “responding” side**
  - **Enterprise features enabled by PEPt**

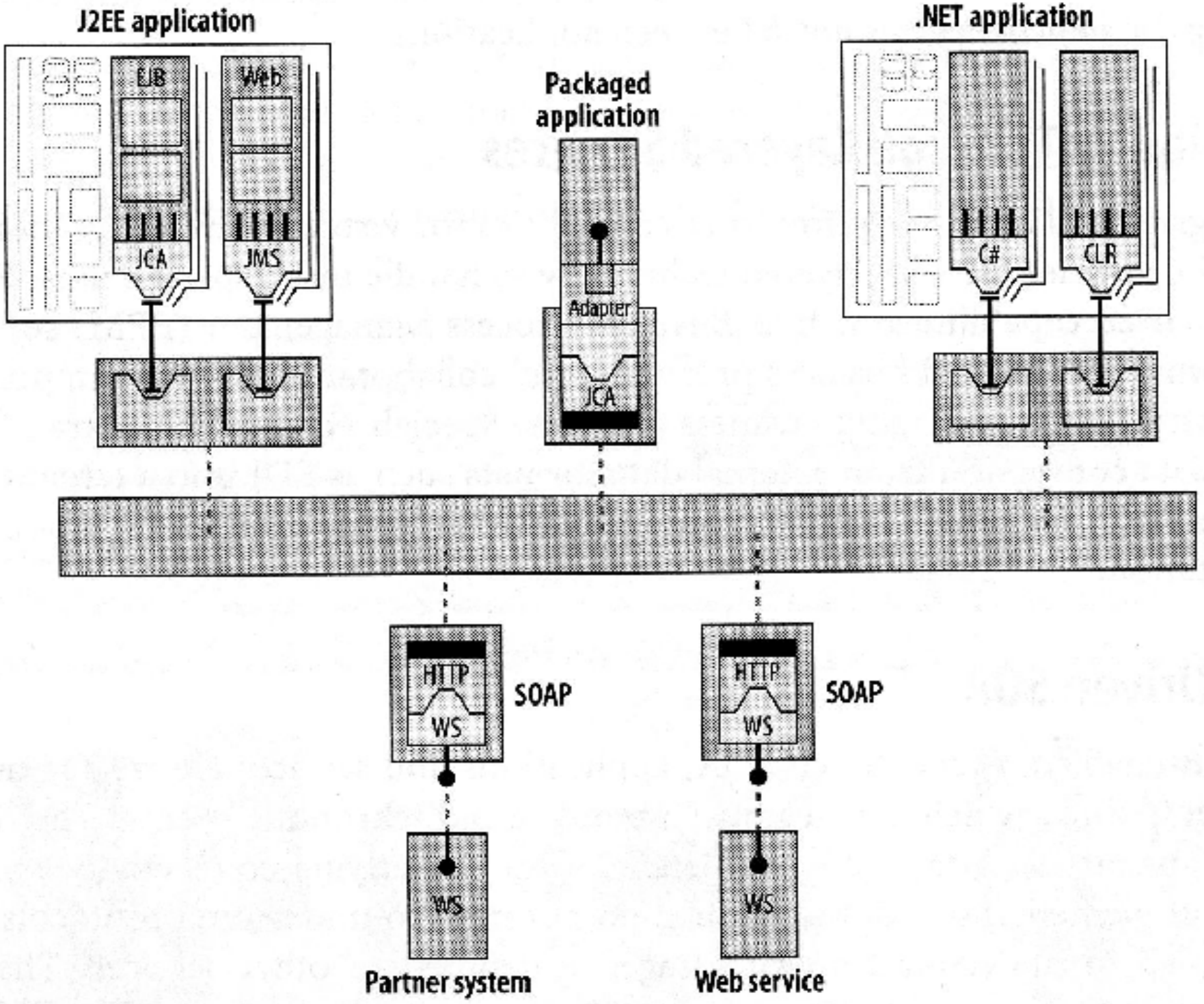
ESB Explained

- **Enterprise Service Bus**
- **How SOA leads to ESB**



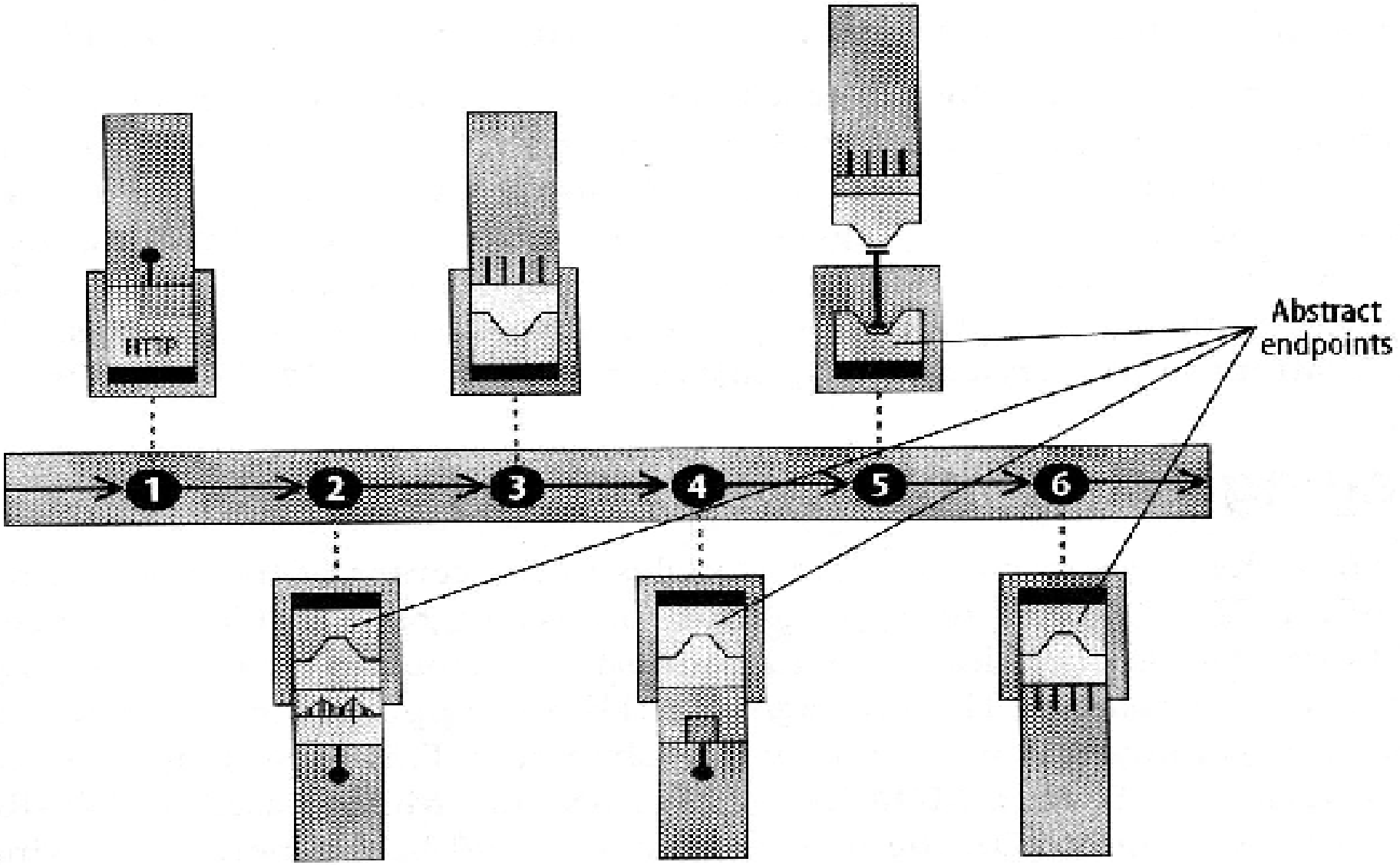
Enterprise Service Bus

- **Event-driven abstract business services**
  - **SOA – e.g., Web Services**
- **Message Oriented Middleware**
  - **multi-protocol**
- **Routing**
  - **Context-based: examine and direct appropriately**
  - **Rules-based: BPEL, WS-Choreography, ebXML**
- **XML data and transformations**

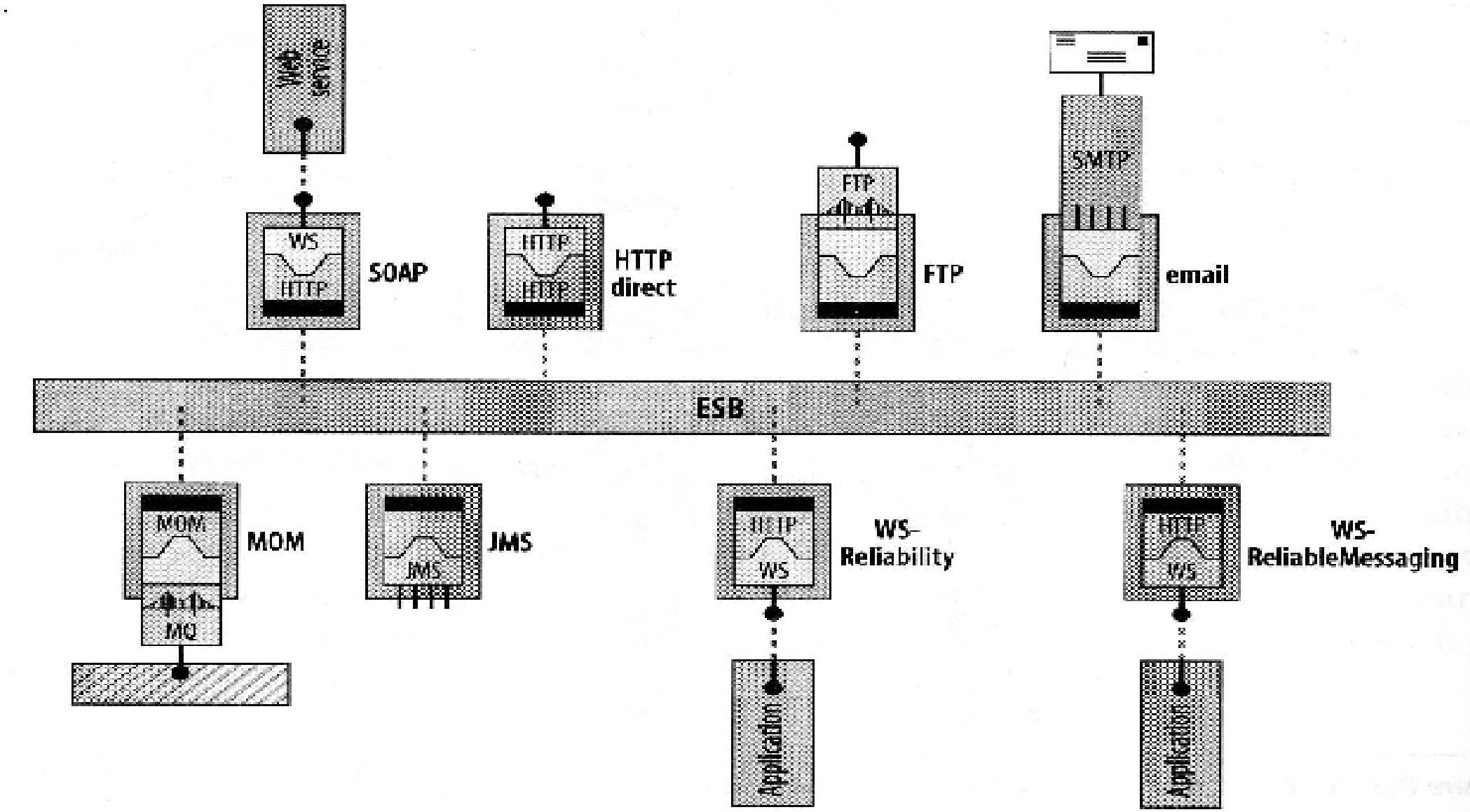

J2EE application
EJB
Web
JCA
JMS
Packaged application
Adapter
JCA
.NET application
C#
CLR
HTTP
WS
SOAP
HTTP
WS
SOAP
WS
WS
Partner system
Web service

ESB – backbone for SOA

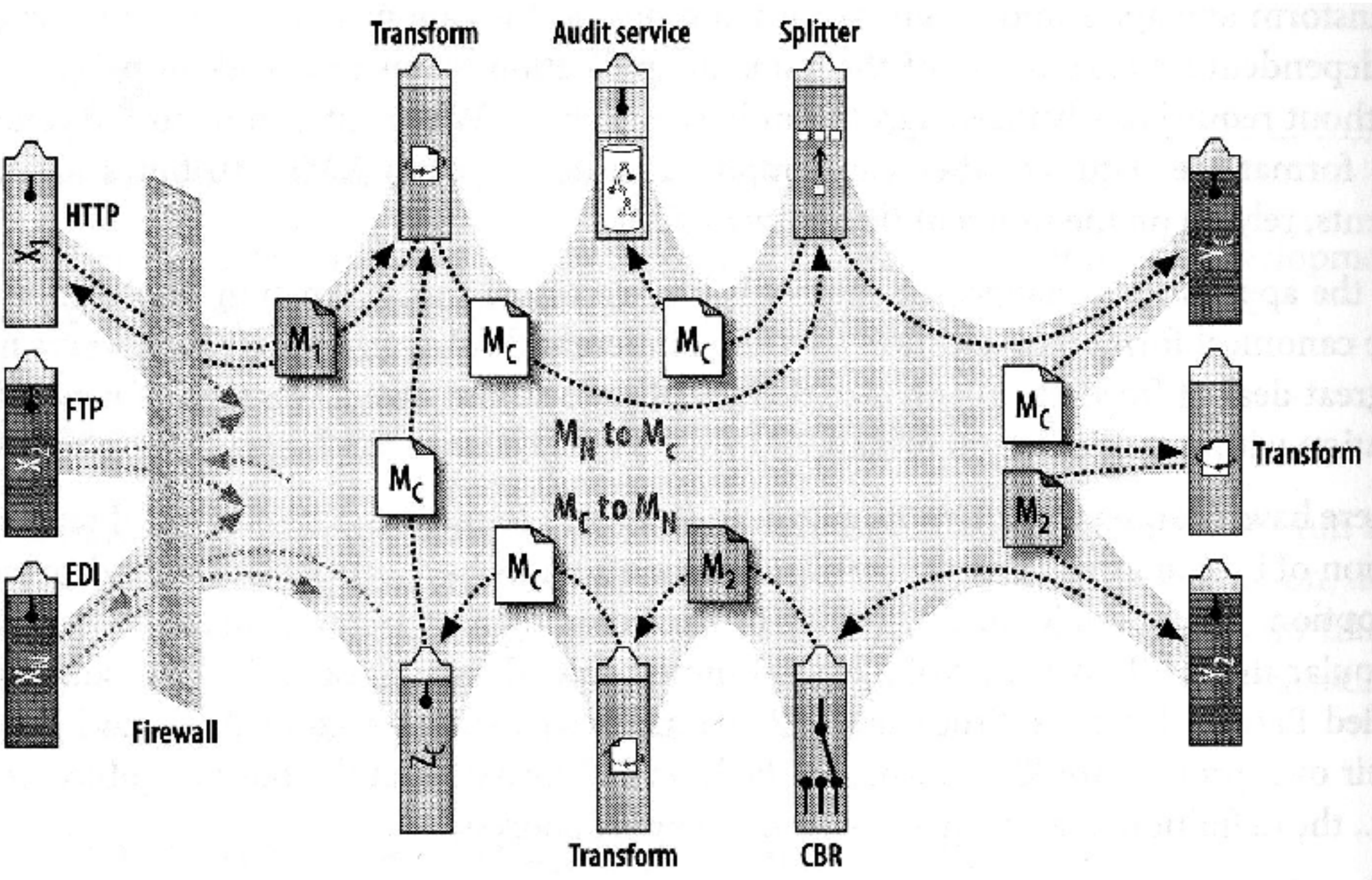
Applications as Services



Services Connected to Multiprotocol Bus



Orchestration/Routing between Services



Why not Web Services?

- **WS is SOA based on internet protocols**
- **But do not want to wire SOAP/HTTP into service**
- **WSDL as description is good**
- **But used in client/server style (e.g., invoke methods) – not good**
- **Need to separate process routing and programming model from interface description and implementation**
- **Need to change underlying protocol without affecting higher-level service functions**

WS Leads to SOA

- **Apps become abstract service endpoints**
- **Respond to asynchronous messages (“events”)**
- **Abstract away from underlying connectivity and remoting infrastructure**
- **Just receive message, process, send message**
- **Messages given to ESB – not directly to target**

Abstract Connectivity

- **Main point of SOA leading to ESB**
- **1st services required separation of interface and implementation**
- **Now require separation of connectivity**

SOA Needs ESB

- **ESB does (XML) data transforms**
- **ESB does routing**
  - **Series of steps in flow between service endpoints**
  - **Can insert/delete steps without modifying apps**
- **ESB handles multi-protocol appropriate to endpoints**

ESB/SOA Requirements

- **Diverse connectivity**
  - **Support multiple protocols**
- **Durable architecture**
  - **Support evolution in protocols**
- **Need open-ended pluggable architecture**

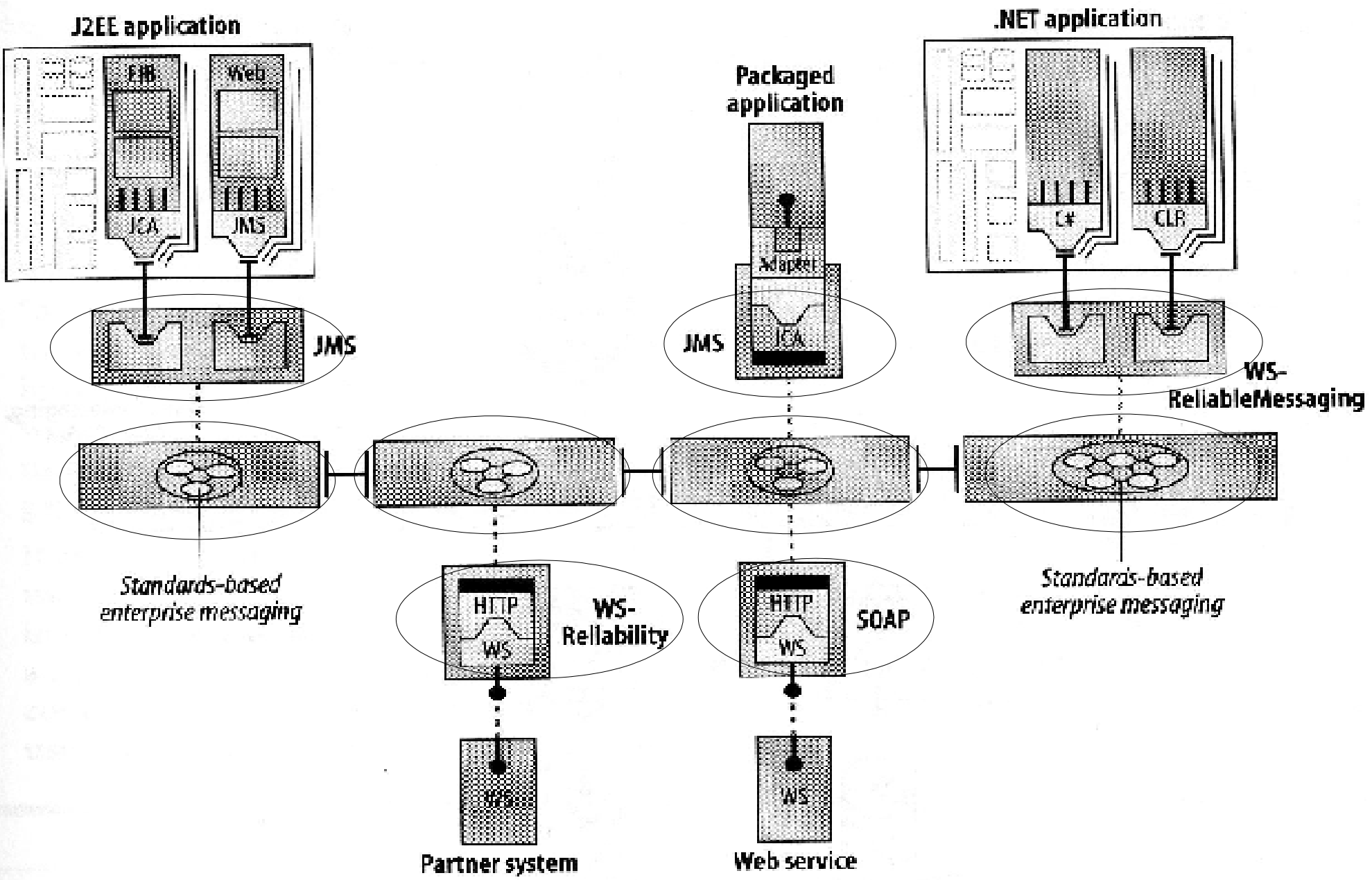
Outline of Talk

- **Part 1 – WS / SOA / ESB**
  - **How .h, RPC, RMI WS leads to SOA**
  - **How SOA leads to ESB**
  - **How PEPt fits into ESB/SOA**
- **Part 2 - PEPt architecture**
  - **PEPt "requesting" side**
  - **PEPt "responding" side**
  - **Enterprise features enabled by PEPt**

PEPt Connectivity for ESB/SOA

- **This (d)evolution will not stop**
  - **CORBA (JavaIDL, RMI-IIOP), RMI, DCOM, ActiveX, Sun-RPC, JAX-RPC, XML-RPC, SOAP, JMS, JAXM, ...**
- **Let customers concentrate on their data and the exchange of that data.**
- **Let us provide an adaptive platform supporting changing**
  - **Presentation, Encodings, Protocols, transports**

PEPt for Multiprotocol ESB Messaging Core

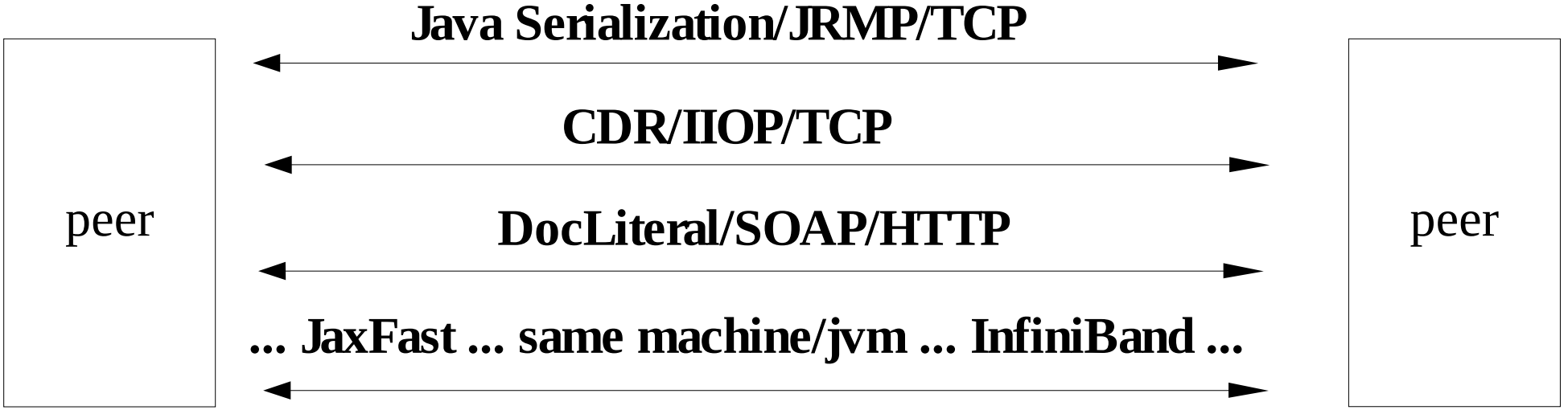


Outline of Talk

- **Part 1 – WS / SOA / ESB**
  - **How .h, RPC, RMI WS leads to SOA**
  - **How SOA leads to ESB**
  - **How PEPt fits into ESB/SOA**
- **Part 2 - PEPt architecture**
  - **PEPt "requesting" side**
  - **PEPt "responding" side**
  - **Enterprise features enabled by PEPt**

Problem Addressed

```
public interface Example extends Remote {
    void sendData(Data x) throws RemoteException;
    void Data getData() throws RemoteException;
}
```



Current partial and non-optimal solution in Java:

- **RMI : ... JRMP ...**
- **RMI-IIOP : ... IIOP ...**
- **JAXRPC : ... SOAP ...**

Limitations of Java Remoting

- Different programming model for each protocol.

- No SPI for plugging in new protocols.

Looking to the Future

- AVOID: new programming model for next protocol.

- Provide an adaptable remoting platform

PEPt Solution

- Provide uniform internal programming model.

- Support multiple external programming models.

- Identify platform interfaces that need to change to support a new “protocol.”

- Provide SPI and factories for those interfaces.

PEPt Benefits

Enables *remoting* system to *adaptively* support *multiple*:

presentations,

encodings,

protocols &

transports.

Remoting: Fundamental Building Blocks

P – Presentation

data types/APIs used by programmer

E – Encoding

wire representation of data types

P – Protocol

meta data to frame & convey intent of message

t – transport

moves data/protocol bits between locations

PEPt Benefits

- Shows responsibility, relationships and interactions between fundamental *remoting* (RPC and Messaging) building blocks.

- Simple but comprehensive framework to: *understand*, *implement*, *reuse*, *evolve*, *maintain* finer-grained details of *remoting* systems.

PEPt Benefits

Enables *remoting* system to *adaptively* support *multiple*:

presentations,

encodings,

protocols &

transports.

Related Work

Jini/Davis: http://davis.jini.org

Iona/Artix: http://www.iona.com/devcenter/artix

WSIF: Apache Web Services Invocation Framework
http://ws.apache.org/wsif/

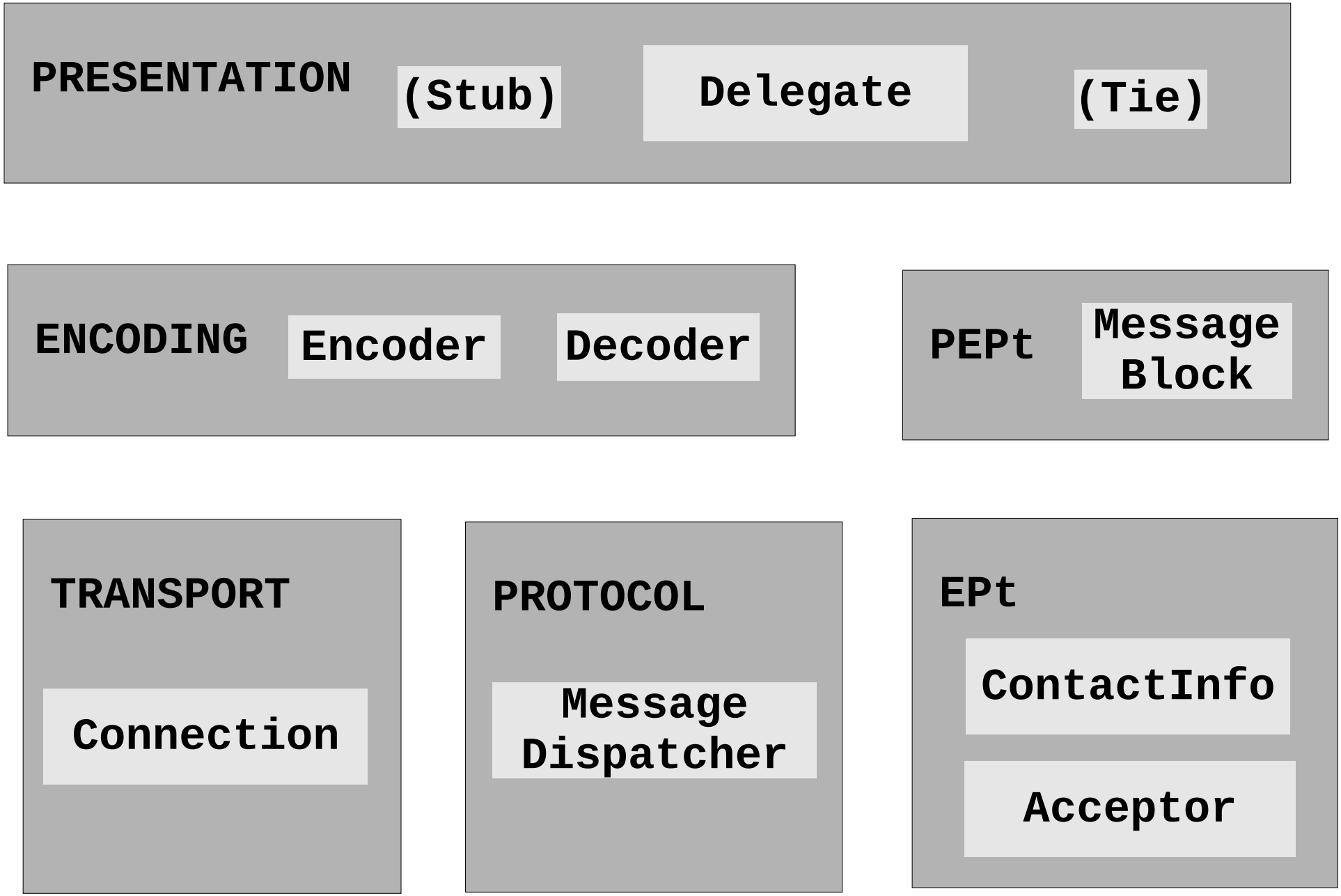
ACE: Reactor, Task, Acceptor-Connector, Proactor, Streams

RM-ODP: Engineering Interface Channel Model: Stub, Binder, Protocol, Interceptor

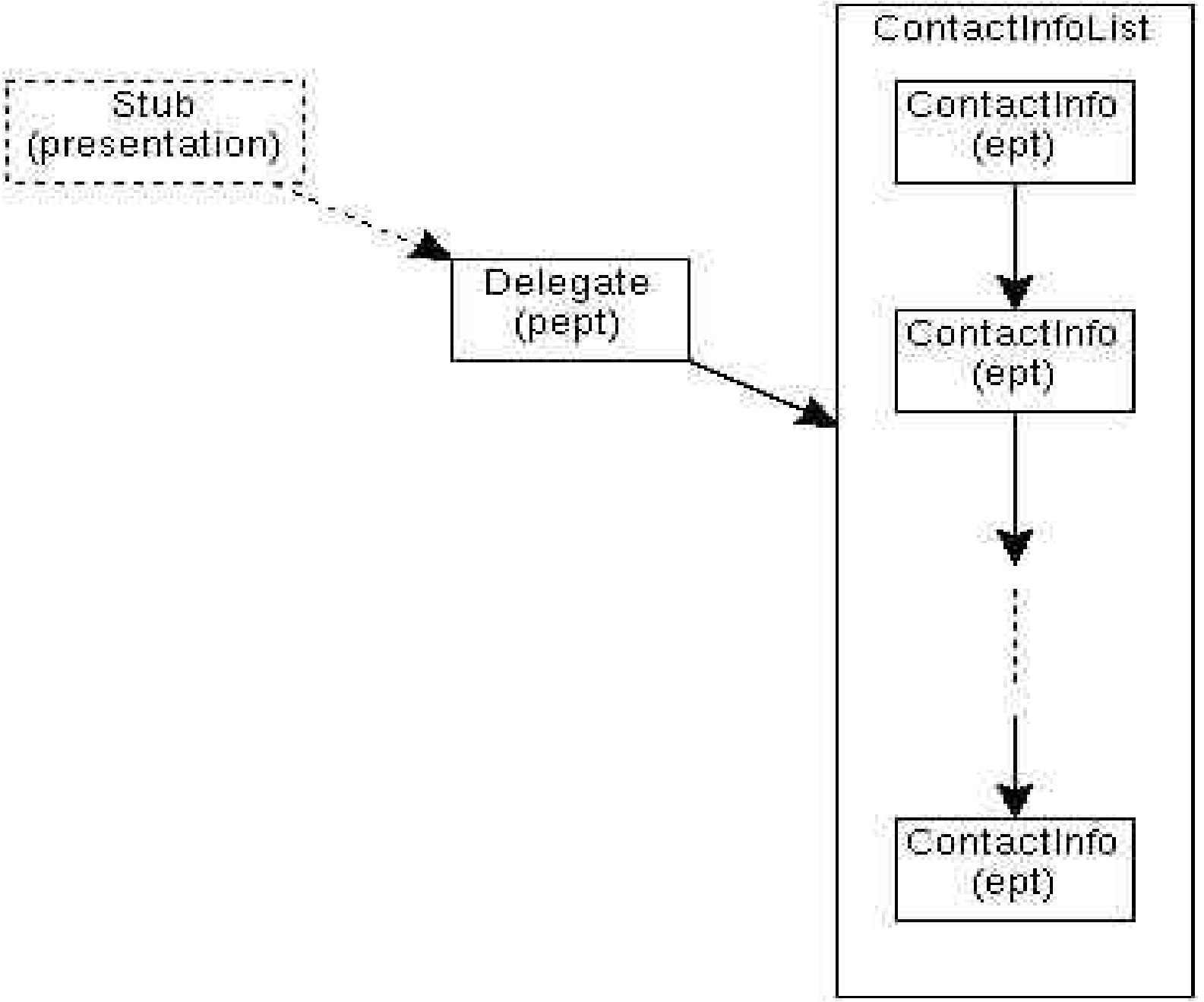
Outline of Talk

- **Part 1 – WS / SOA / ESB**
  - **How .h, RPC, RMI WS leads to SOA**
  - **How SOA leads to ESB**
  - **How PEPt fits into ESB/SOA**
- **Part 2 - PEPt architecture**
  - **PEPt "requesting" side**
  - **PEPt "responding" side**
  - **Enterprise features enabled by PEPt**

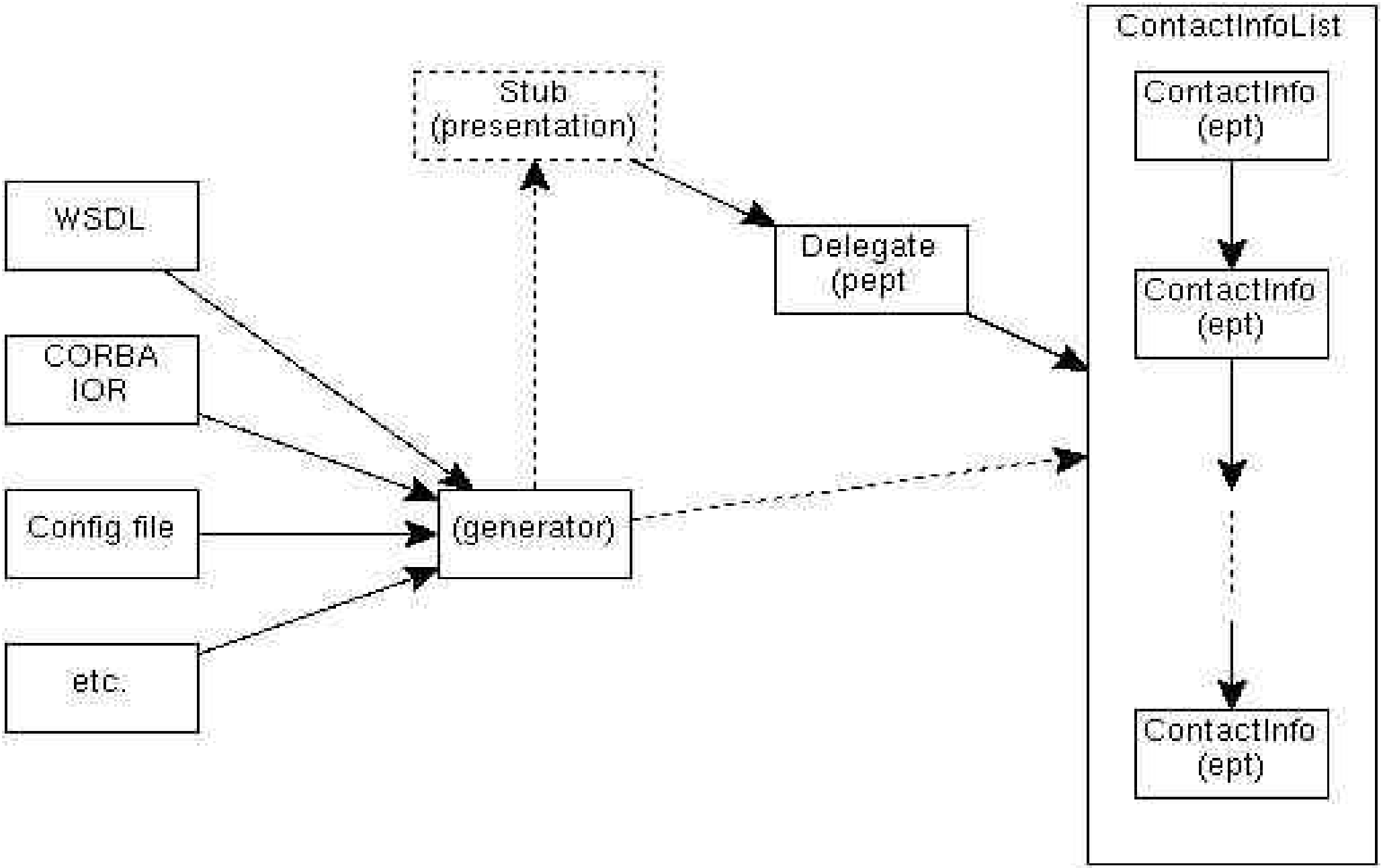
PEPt Architecture



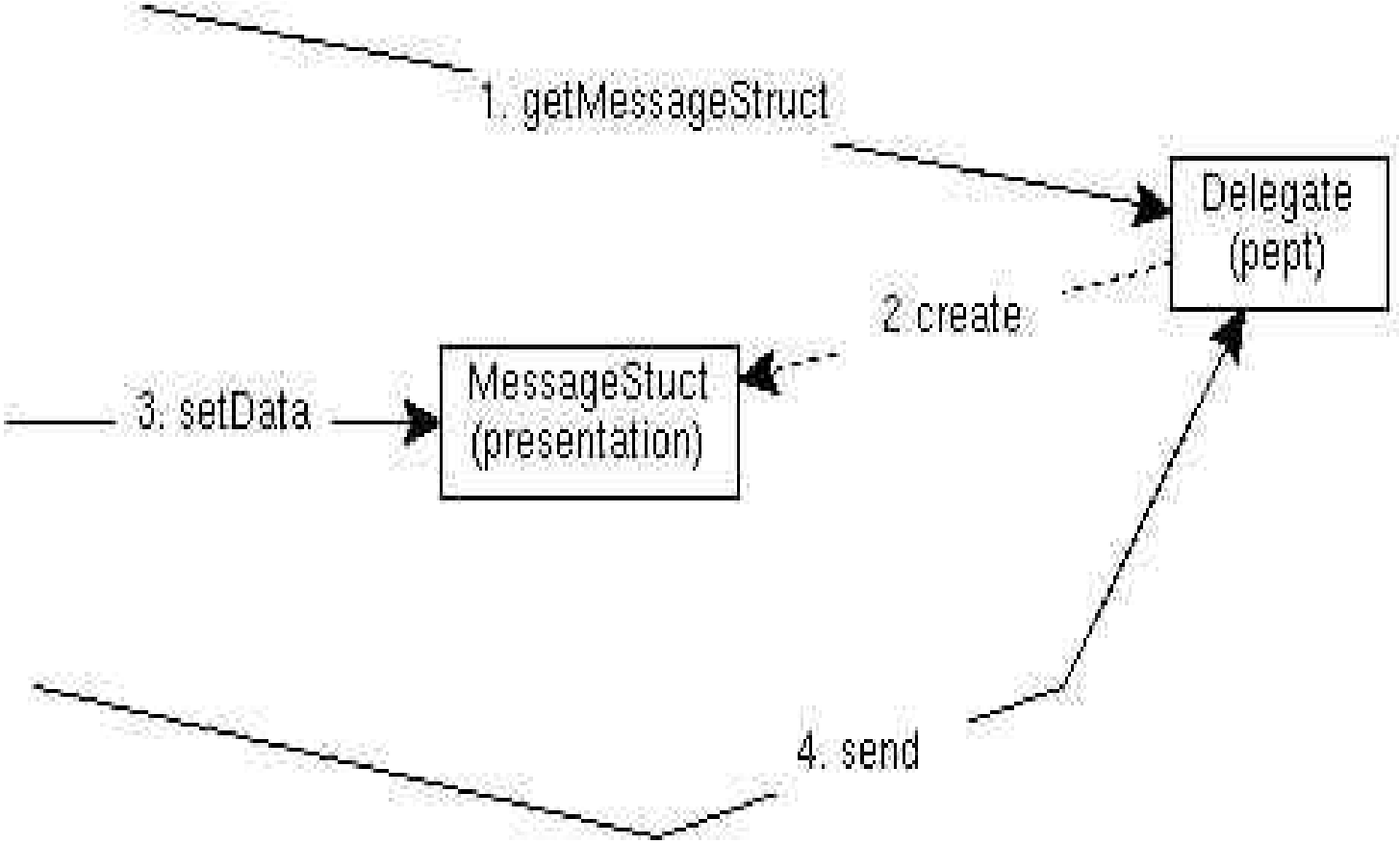
Client Static Structure



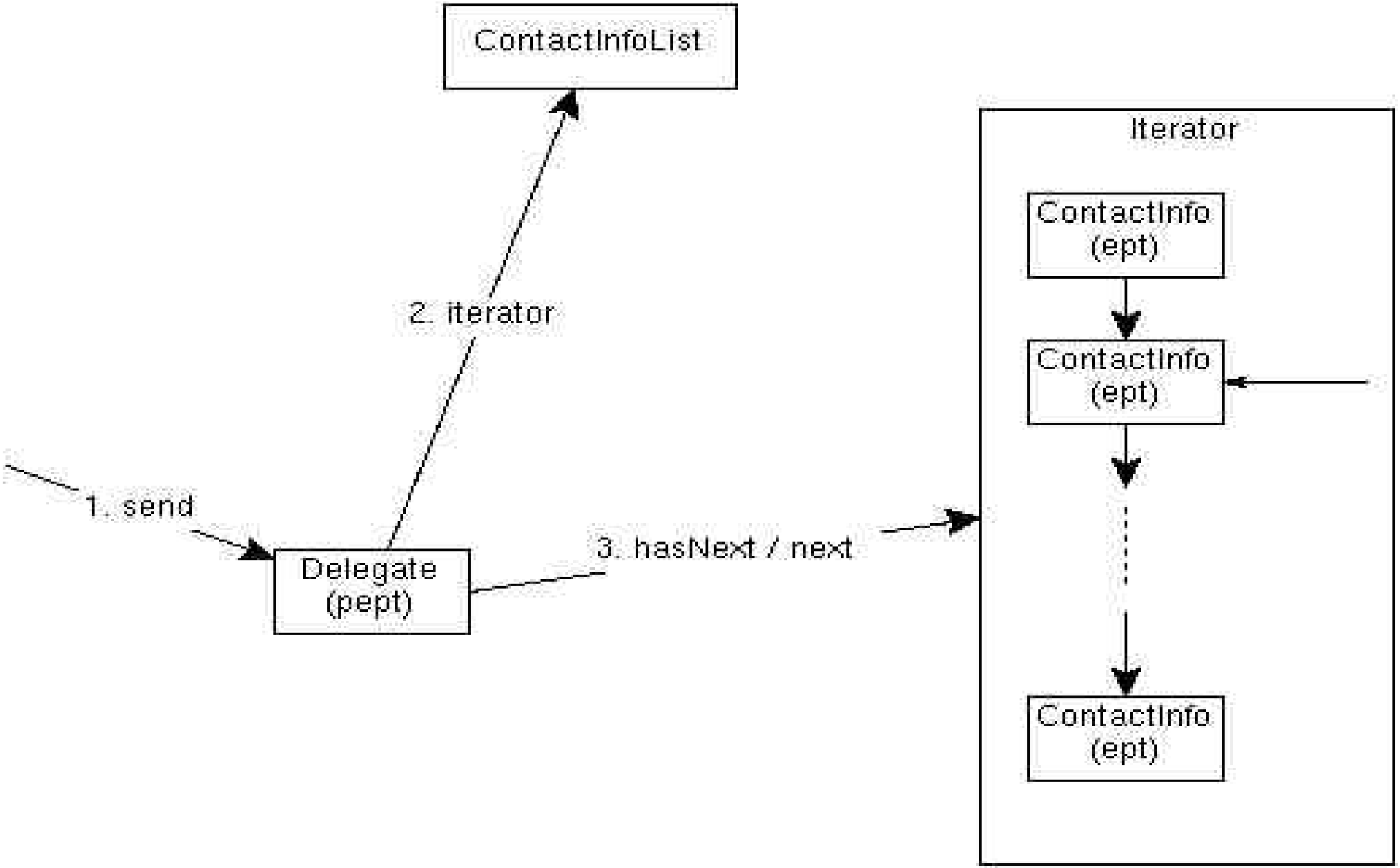
Generate Static Structure



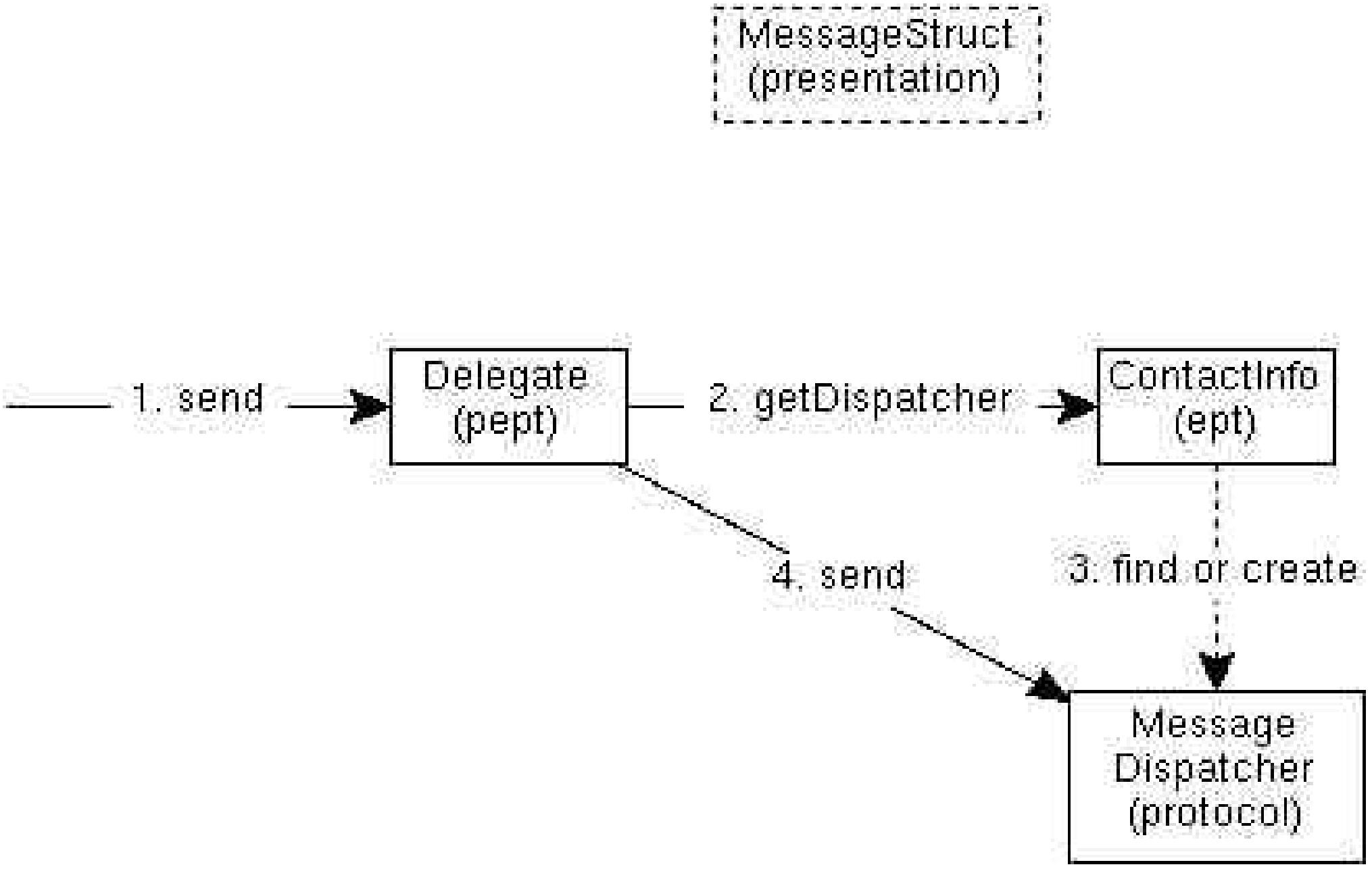
Message Struct



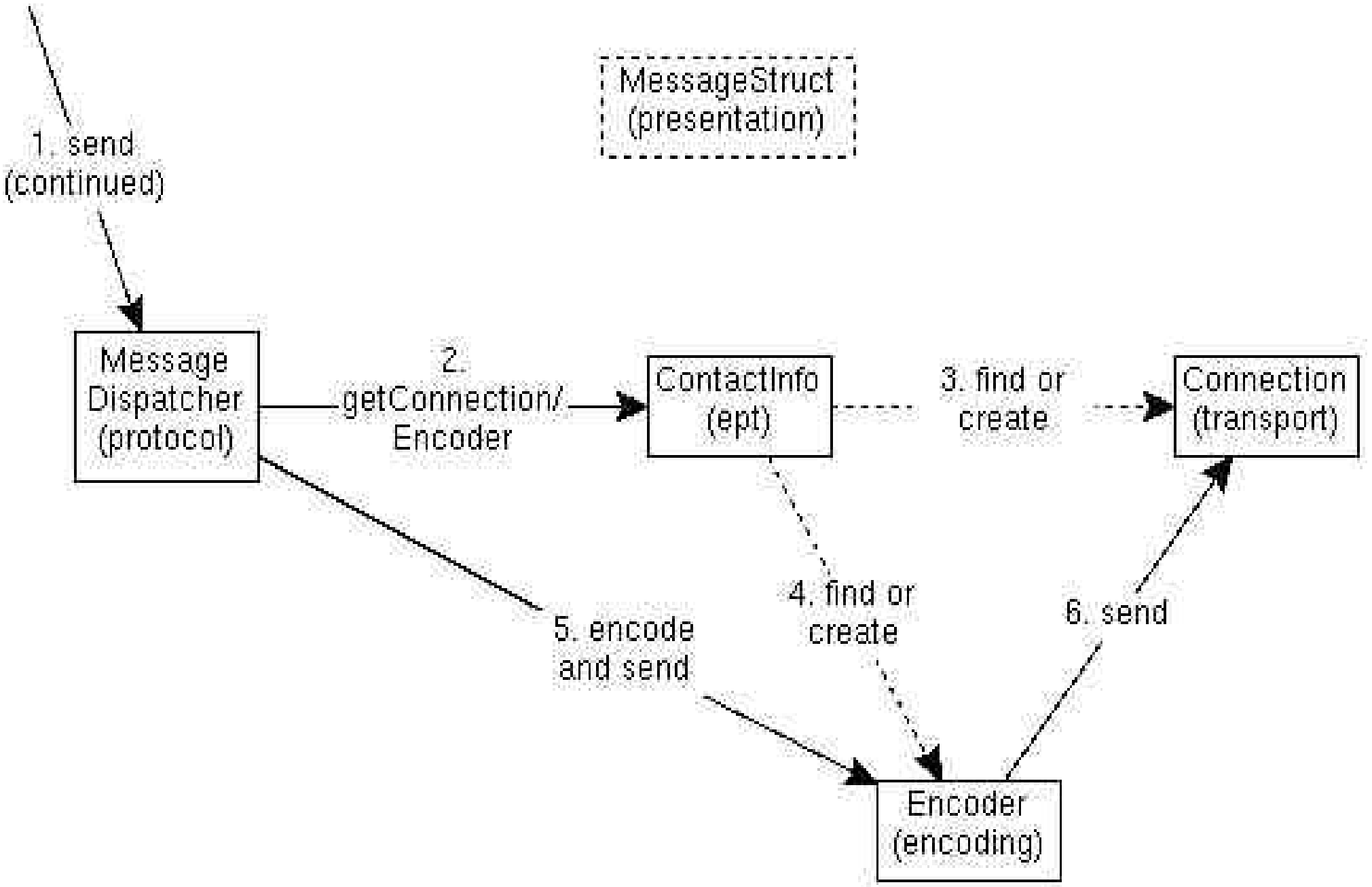
Choose ContactInfo



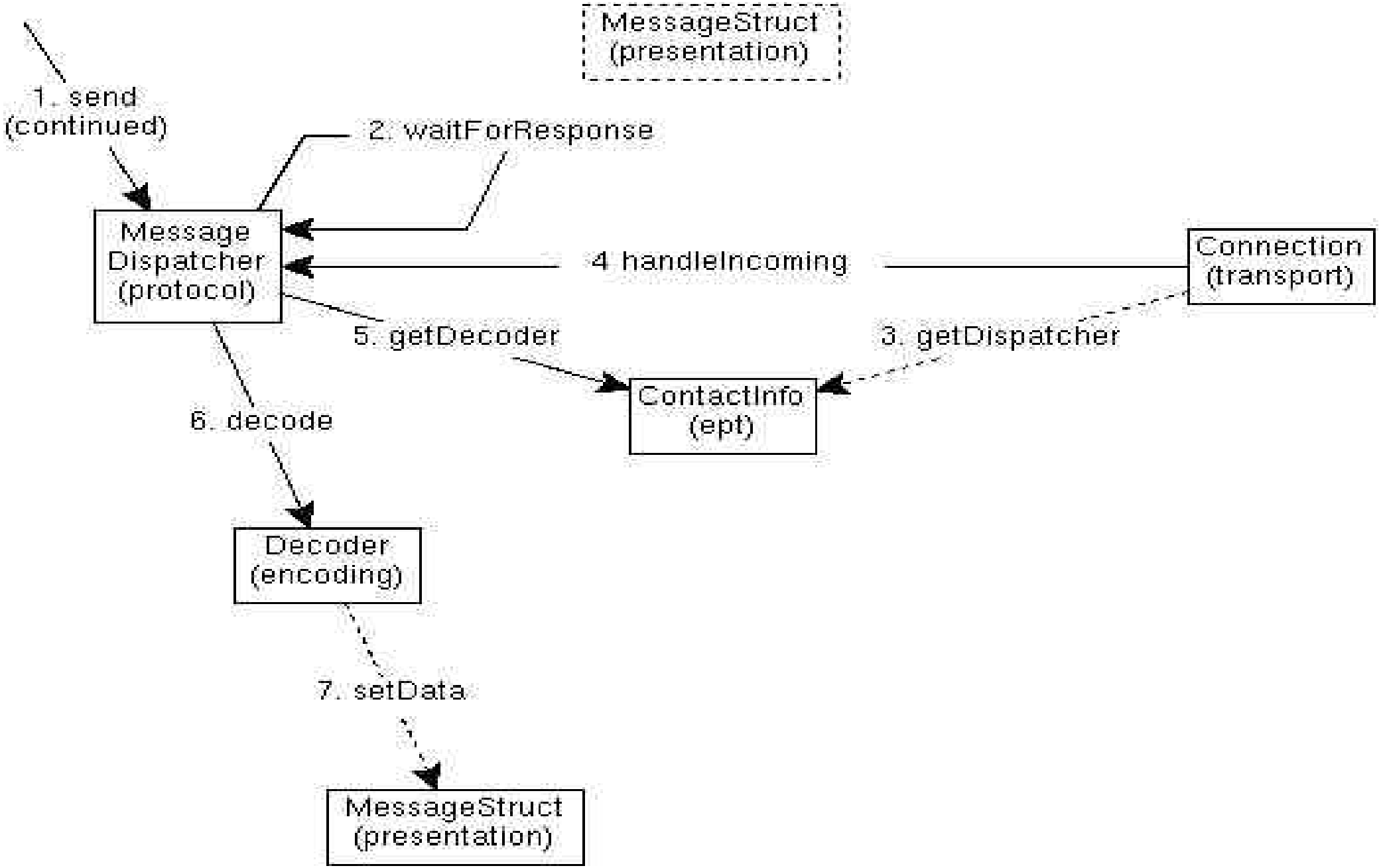
Message Dispatcher



Encoder – Connection – Send



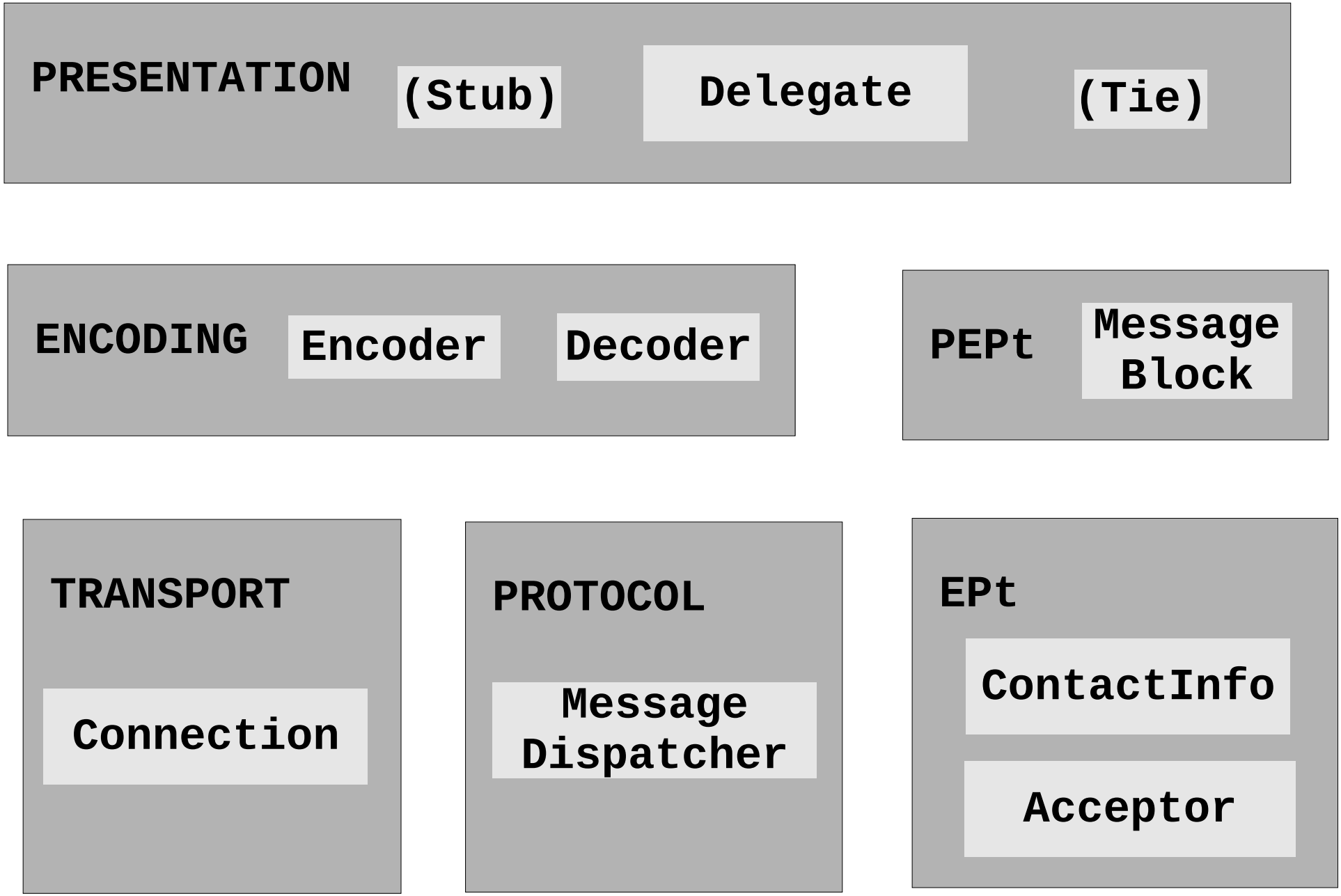
Receive


MessageStruct
(presentation)
1. send
(continued)
2. waitForResponse
Message
Dispatcher
(protocol)
4 handleIncoming
Connection
(transport)
5. getDecoder
3. getDispatcher
ContactInfo
(ept)
6. decode
Decoder
(encoding)
7. setData
MessageStruct
(presentation)

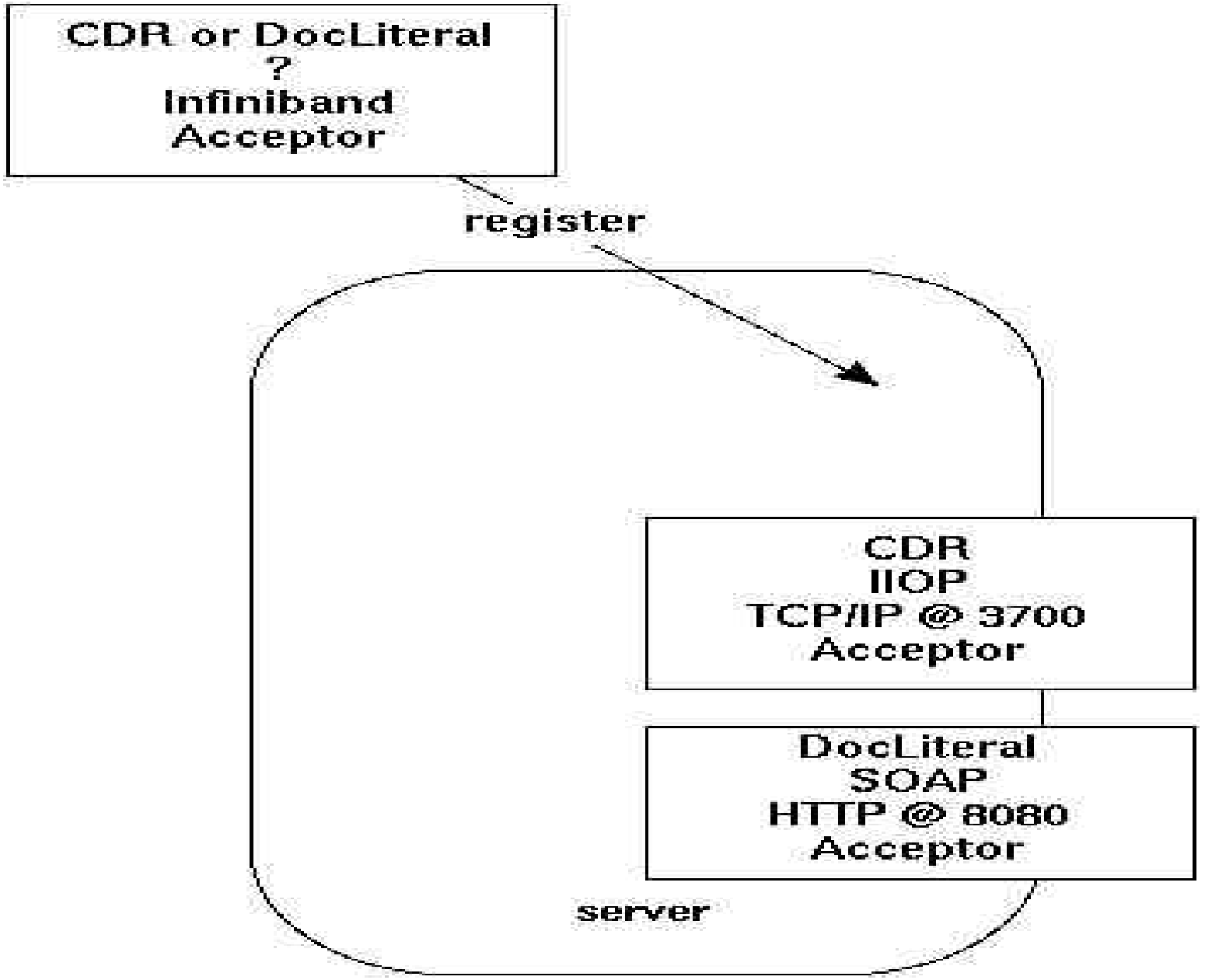
Outline of Talk

- **Part 1 – WS / SOA / ESB**
  - **How .h, RPC, RMI WS leads to SOA**
  - **How SOA leads to ESB**
  - **How PEPt fits into ESB/SOA**
- **Part 2 - PEPt architecture**
  - **PEPt "requesting" side**
  - **PEPt "responding" side**
  - **Enterprise features enabled by PEPt**

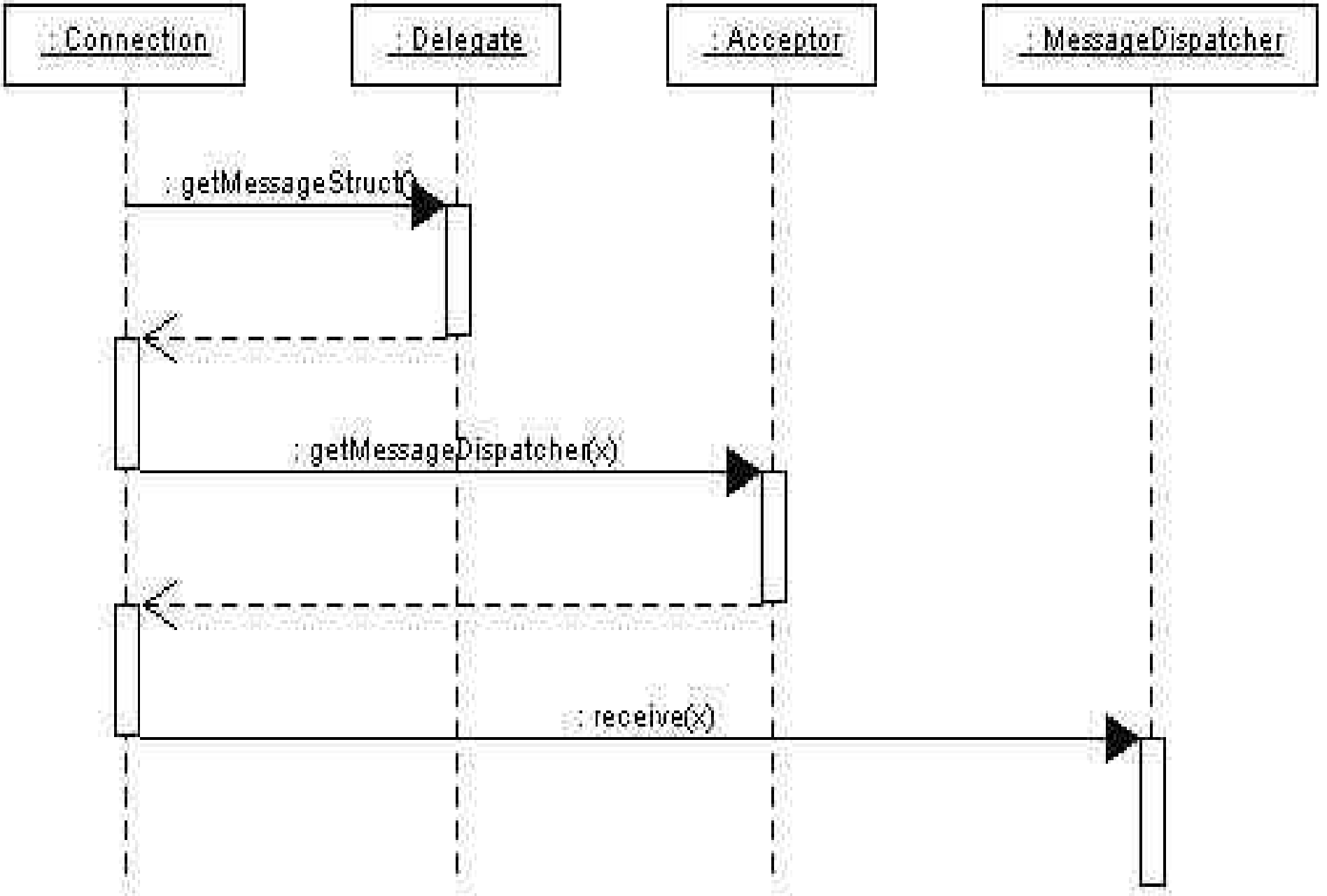
PEPt Architecture



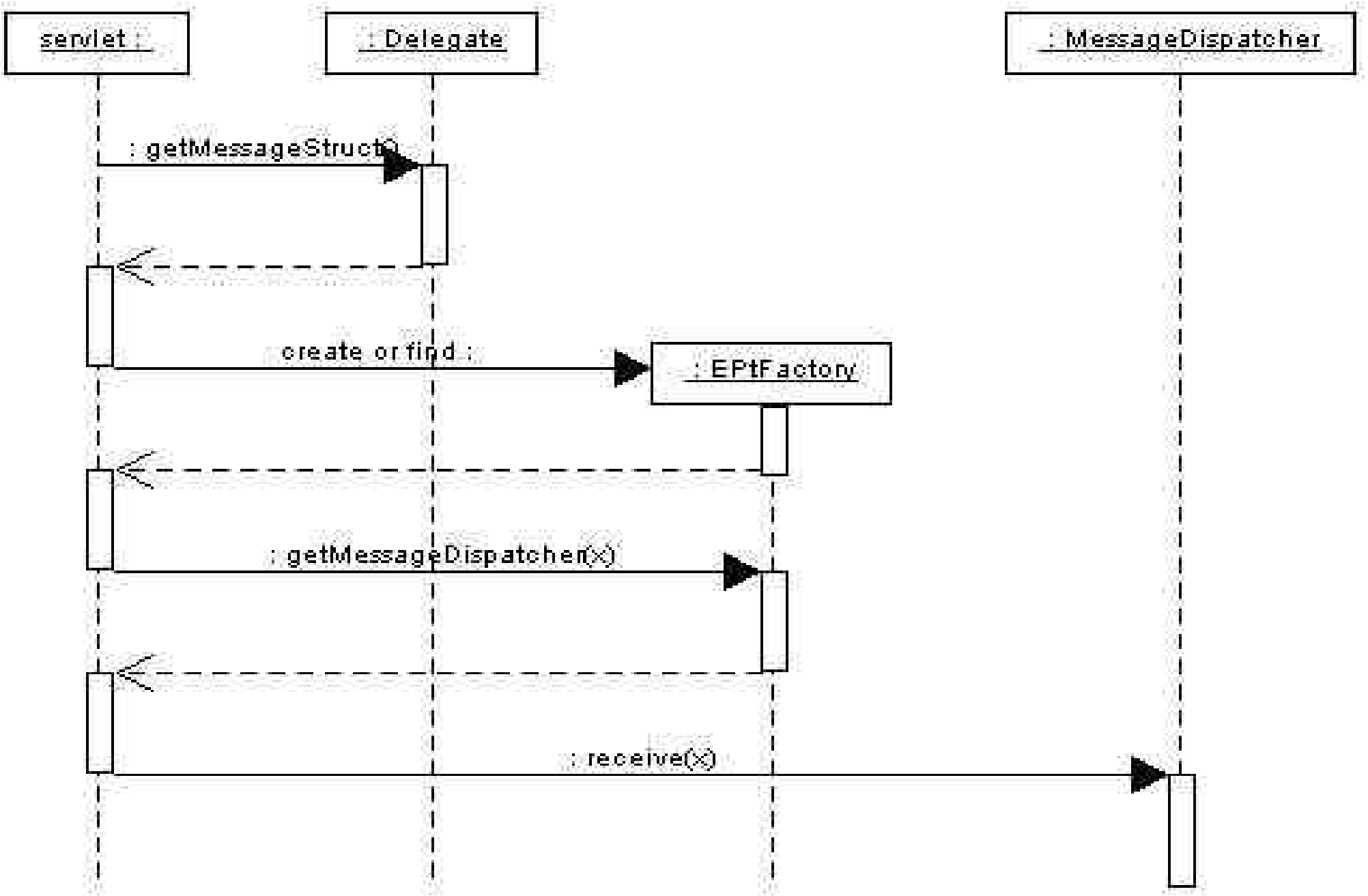
Register Acceptor



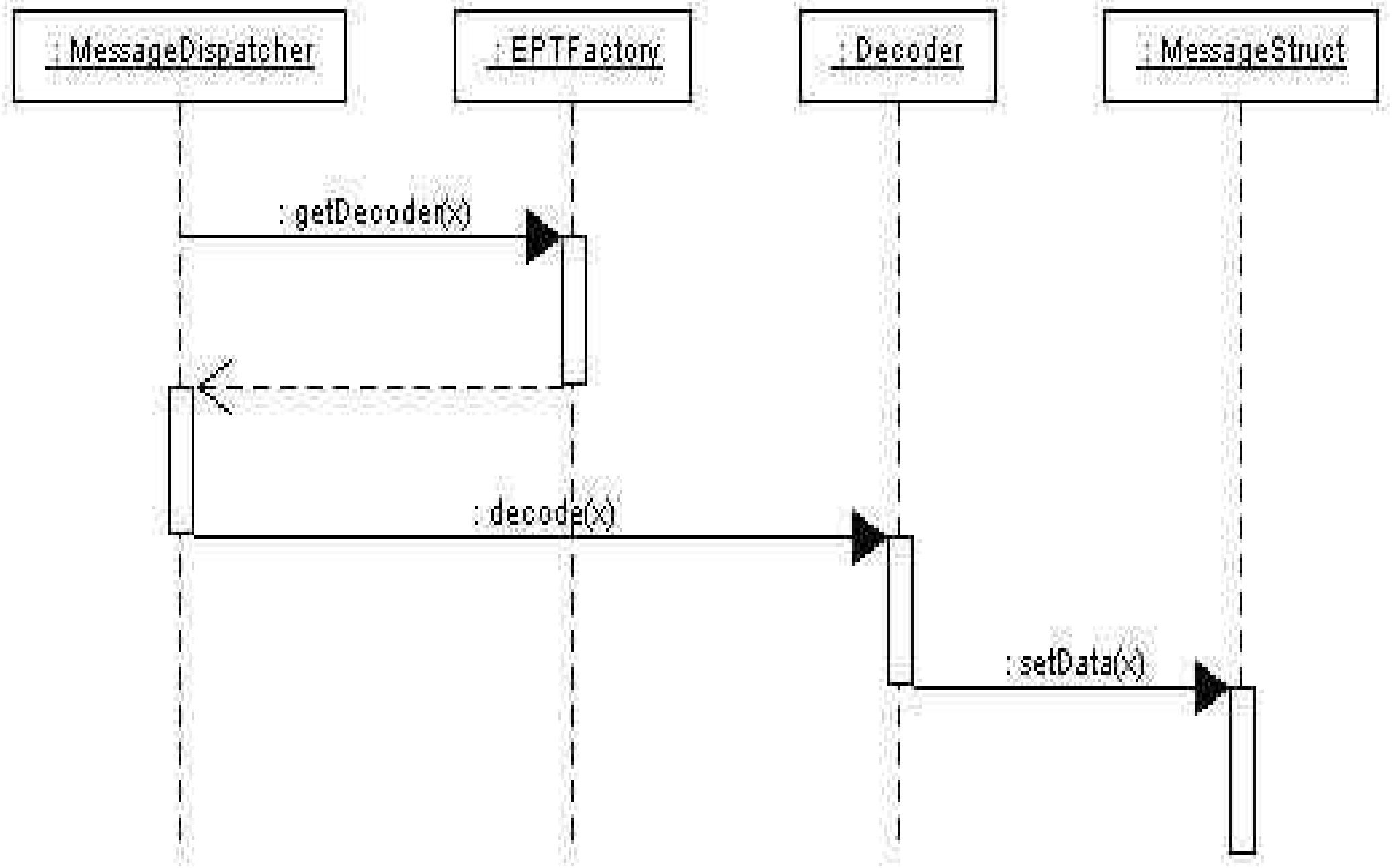
Get MessageDispatcher



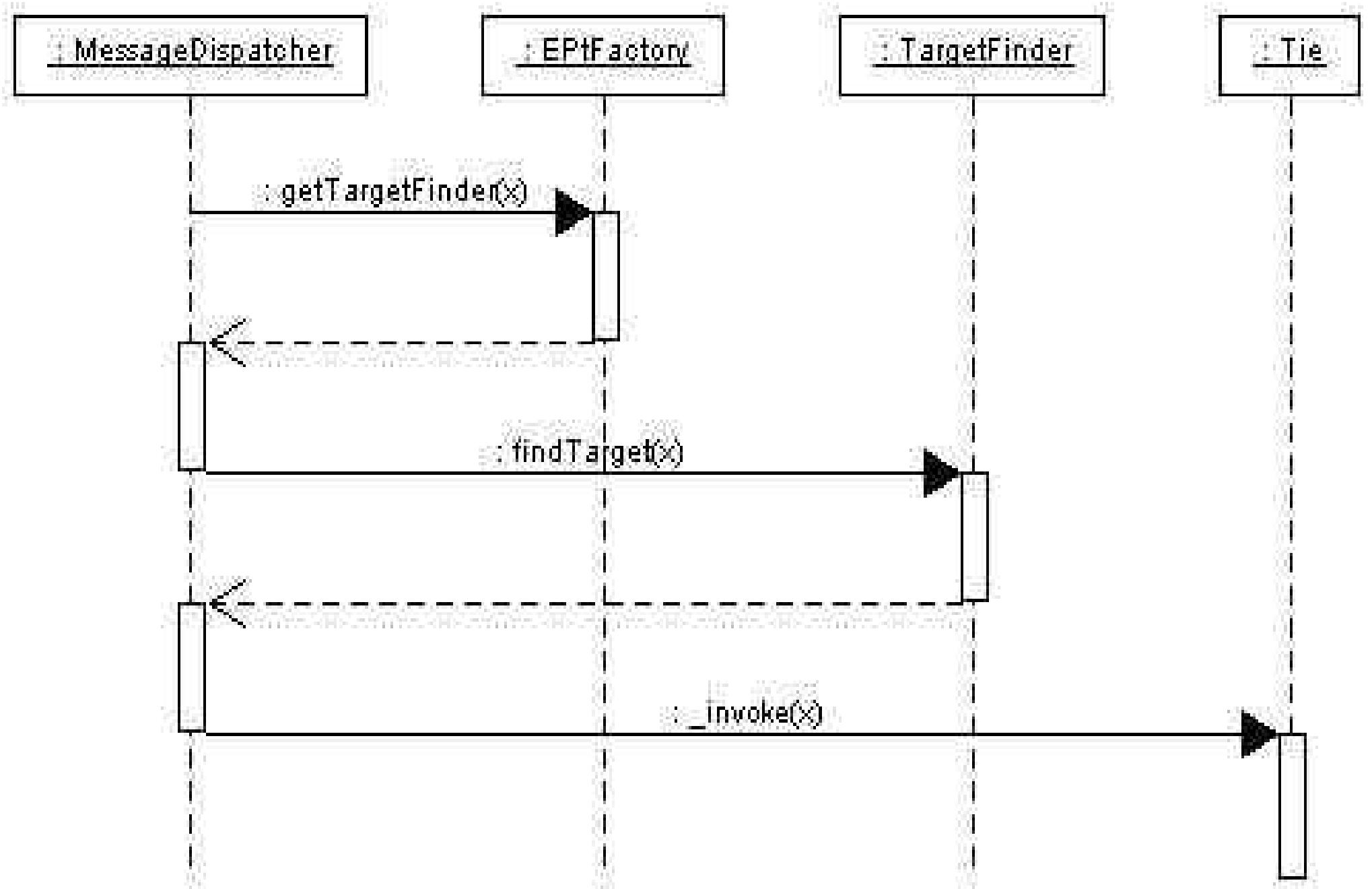
PEPt in a Web Container



Decode


: MessageDispatcher
: EPTFactory
: Decoder
: MessageStruct
: getDecoder(x)
: decode(x)
: setData(x)

Finding & Invoking the Tie



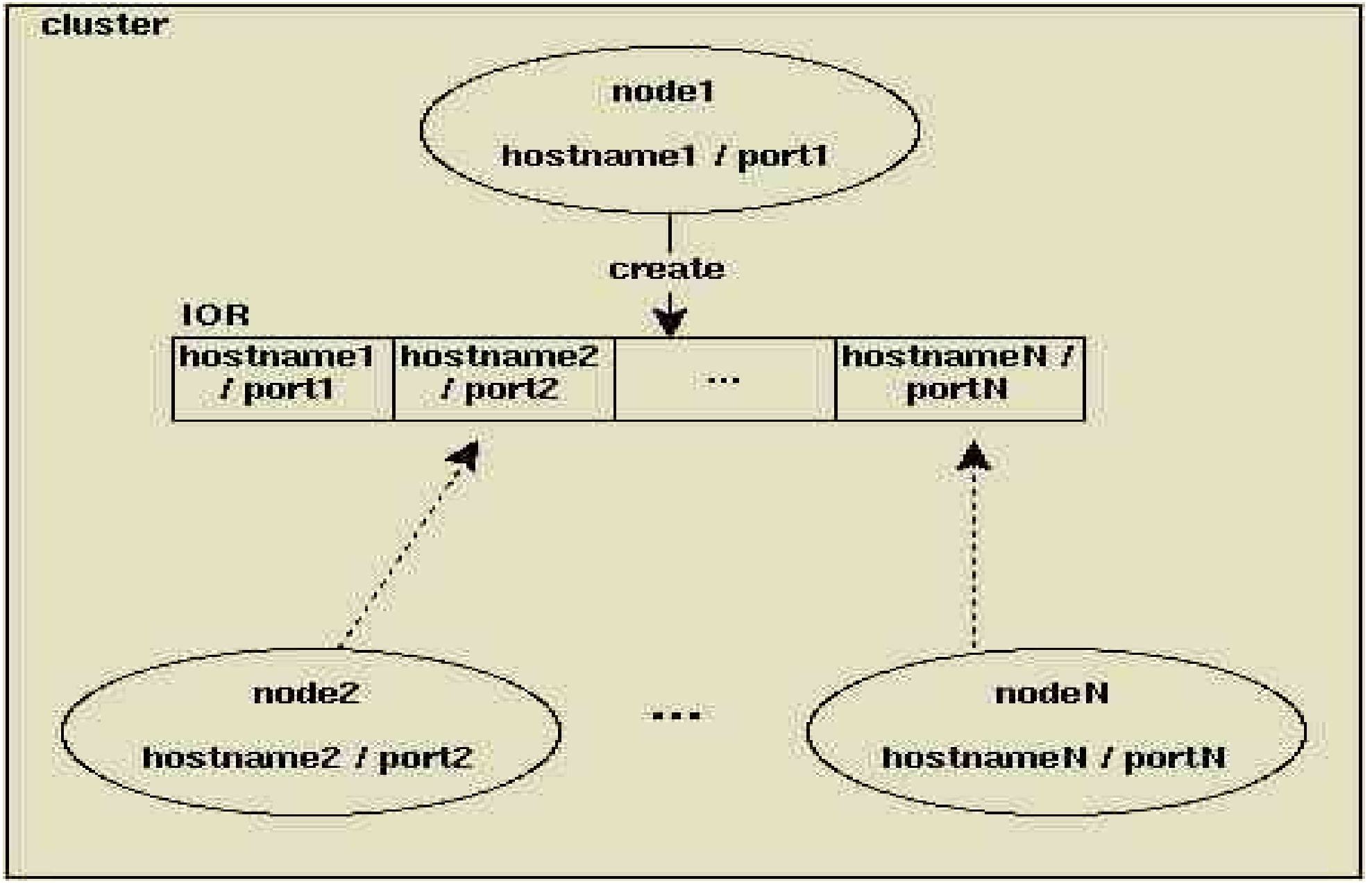
Send Response

- **Similar to Requesting side sending a request**

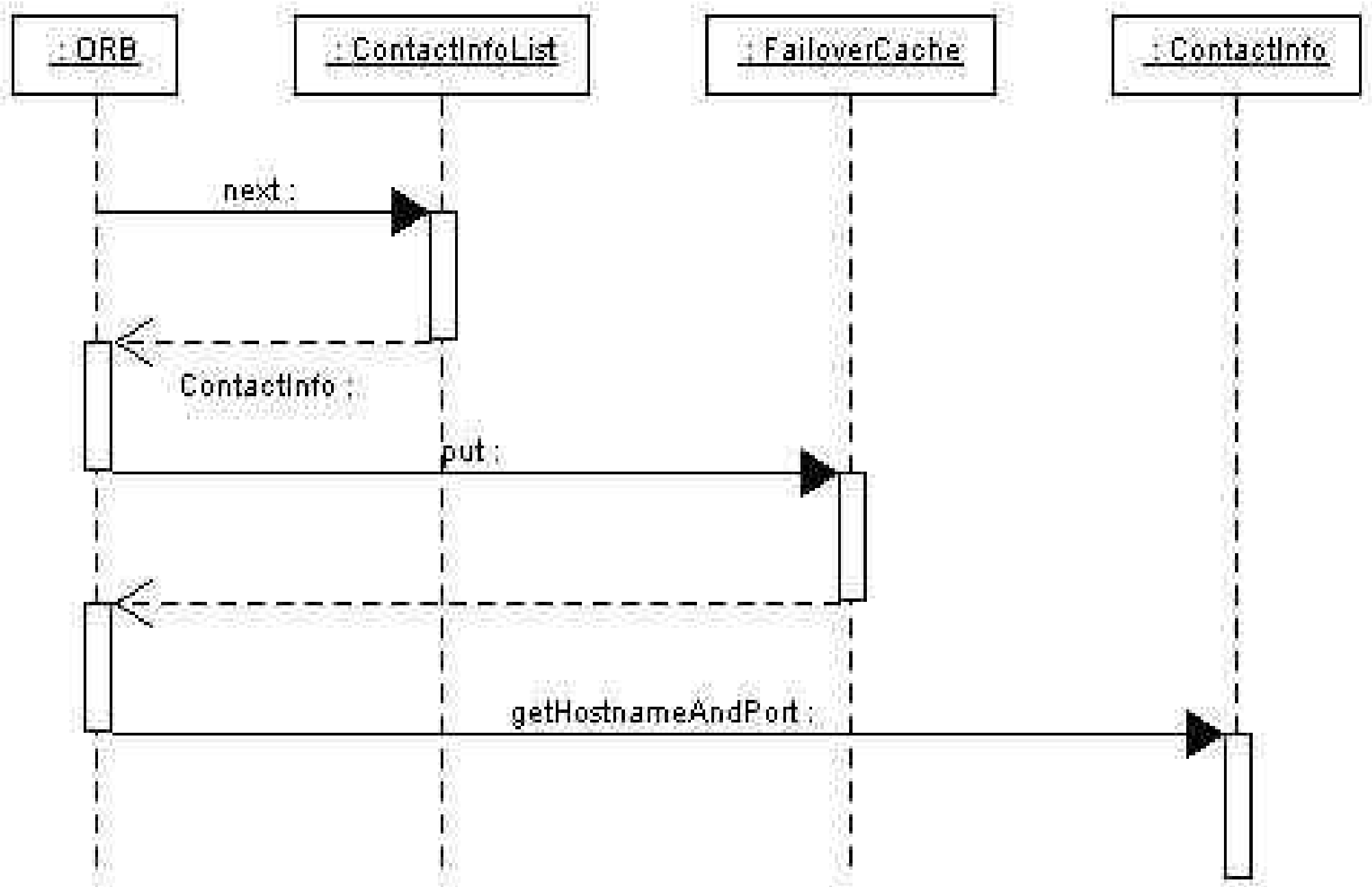
Outline of Talk

- **Part 1 – WS / SOA / ESB**
  - **How .h, RPC, RMI WS leads to SOA**
  - **How SOA leads to ESB**
  - **How PEPt fits into ESB/SOA**
- **Part 2 - PEPt architecture**
  - **PEPt "requesting" side**
  - **PEPt "responding" side**
  - **Enterprise features enabled by PEPt**

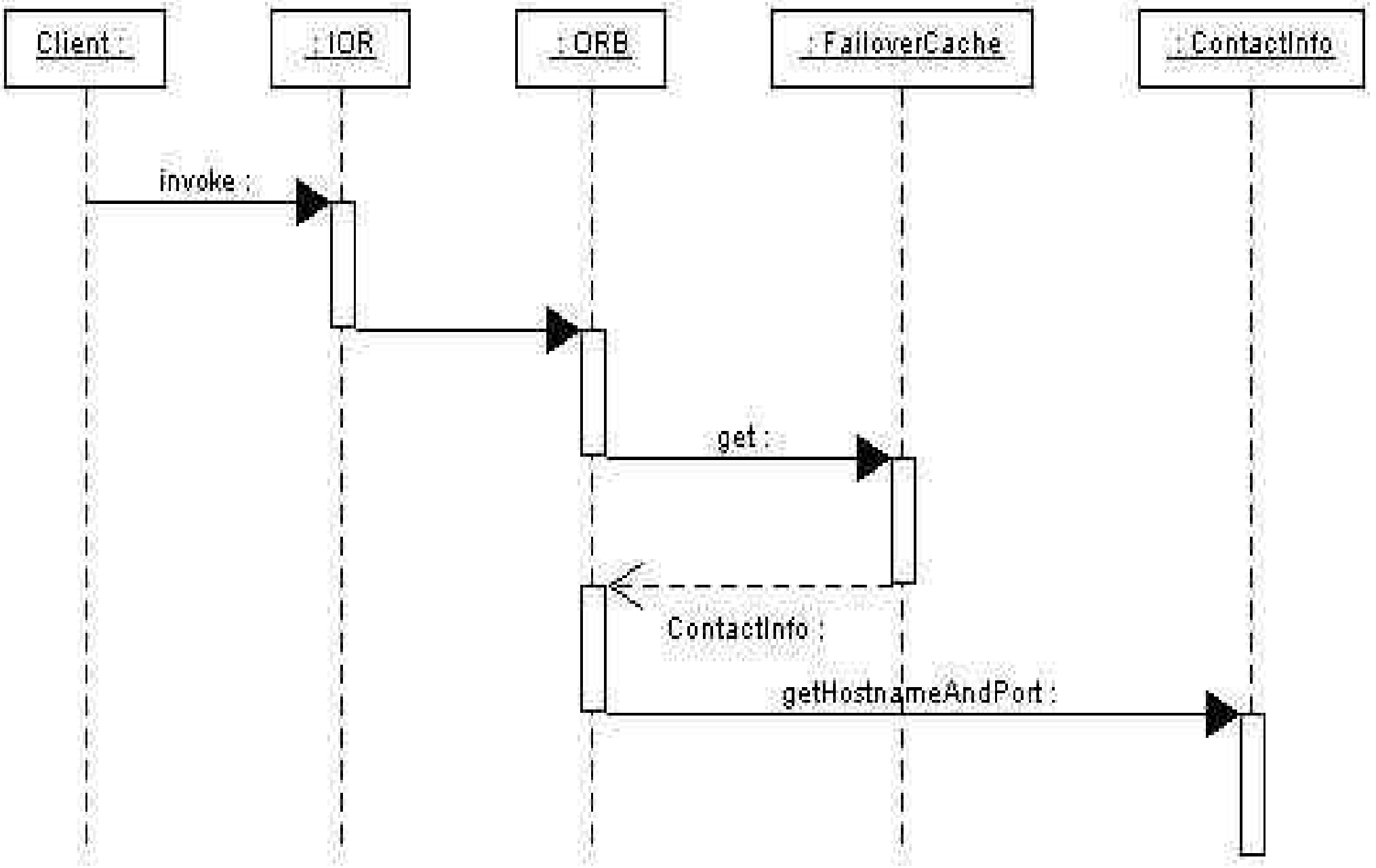
Failover


cluster
node1
hostname1 / port1
create
IOR
hostname1 / port1
hostname2 / port2
...
hostnameN / portN
node2
hostname2 / port2
...
nodeN
hostnameN / portN

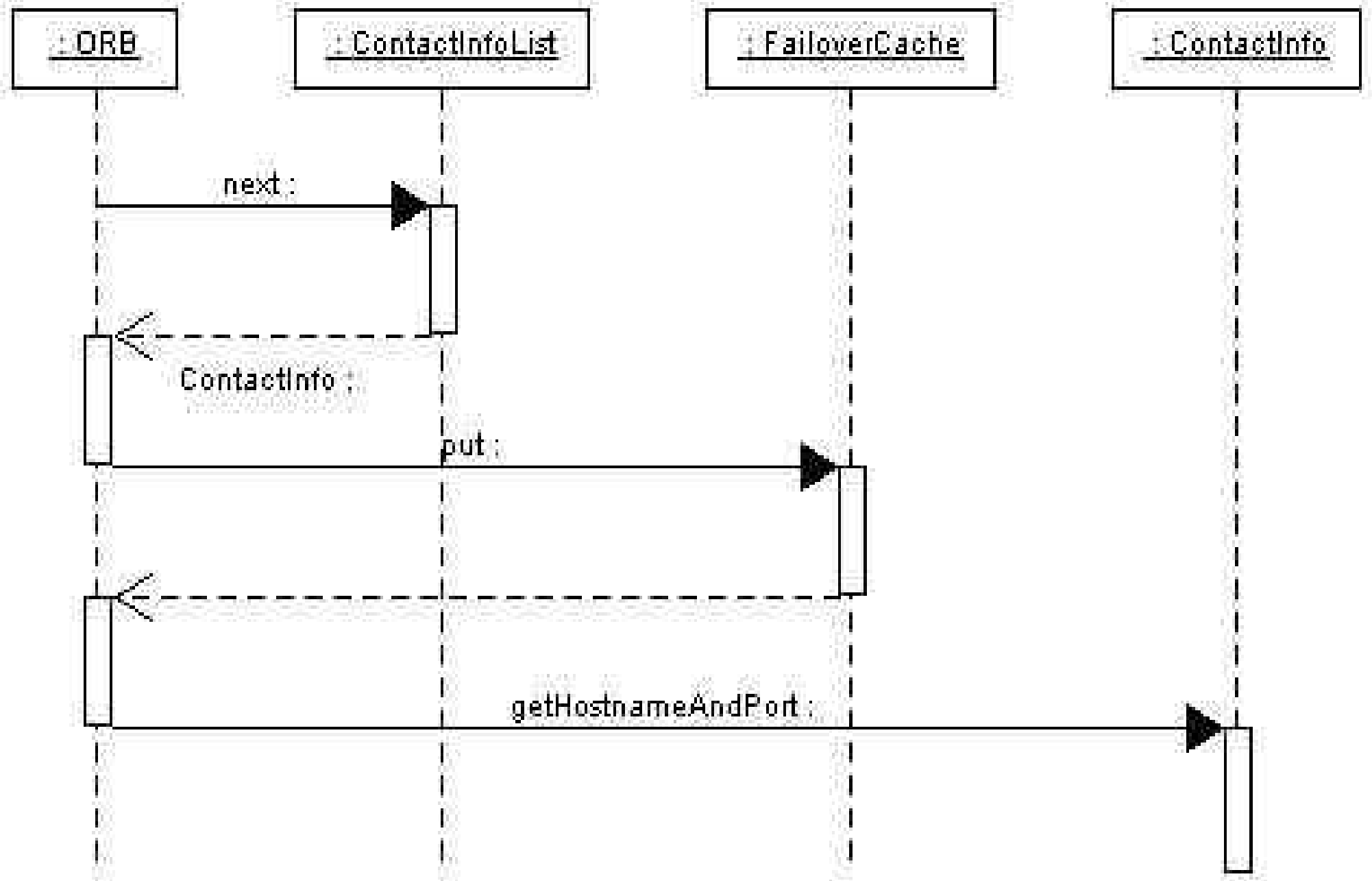
Initialize Cache & Try Invoke



Using a Cached Address



Failover to Next Address



Outline of Talk

- **Part 1 – WS / SOA / ESB**
  - **How .h, RPC, RMI WS leads to SOA**
  - **How SOA leads to ESB**
  - **How PEPt fits into ESB/SOA**
- **Part 2 - PEPt architecture**
  - **PEPt "requesting" side**
  - **PEPt "responding" side**
  - **Enterprise features enabled by PEPt**
- **Conclusion**

Conclusions

- PEPt: Adaptable Architecture for Remoting Systems.

- PEPt: Handles addressing, colocation, retries, errors, connection multiplexing, fragmentation, transactions, security, ...

- Used in:
  - ORB in Sun's J2SE 5
  - App Server 8.x
  - JAX-RPC 2.0

- Influencing JBI (JSR-208) Binding Components

PEPt Benefits

Enables *remoting* system to *adaptively* support *multiple*:

presentations,

encodings,

protocols &

transports.

More Info

http://javaweb.sfbay.sun.com/~hcarr/pept/

http://haroldcarr.net/

Extra Slides

- **The following slides are not part of the presentation.**

Original PEPt Motivation

- **Small CORBA (i.e., JavaIDL, RMI-IIOP) team**
- **Deliver into both J2SE and J2EE from same codebase**
- **Need pluggable architecture to deliver enterprise features to EE but not SE**
  - **Load-balancing**
  - **Failover**
  - **High-performance encodings/transports**

Reduce Implementation Space

- **For:**
- **N - programming models (presentation) &**
- **M - EPTs (encoding, protocol, transport)**
- **Current status: N * M**
- **With PEPt: N + M**

App Server viz-a-viz ESB

- **App Server to host business logic**
- **ESB for connectivity and routing**