Technische Universität Darmstadt

Telecooperation

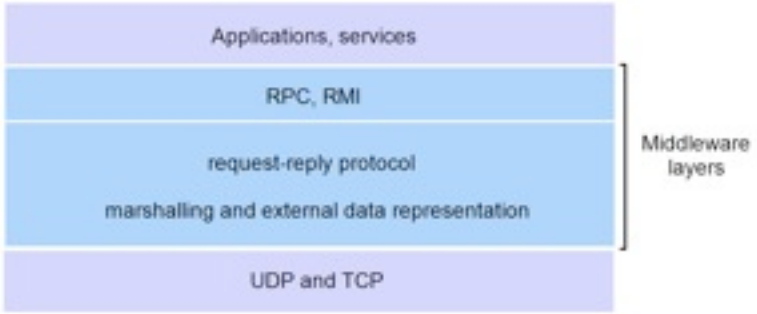
Ubiquitous & Mobile Computing

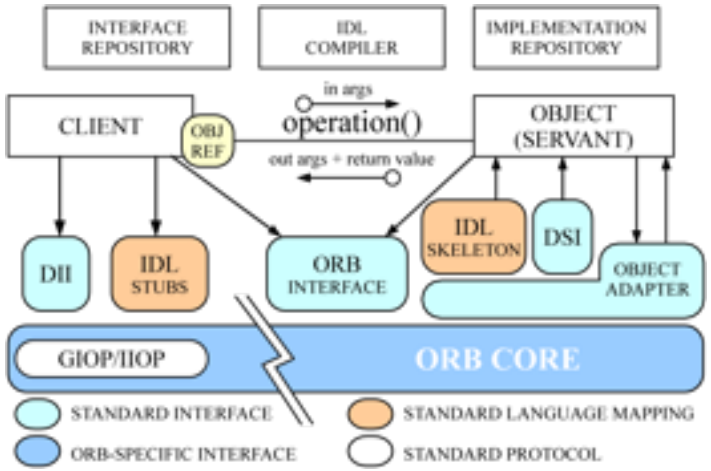
Connectivity: Ubiquitous Communications

Dr. Erwin Aitenbichler

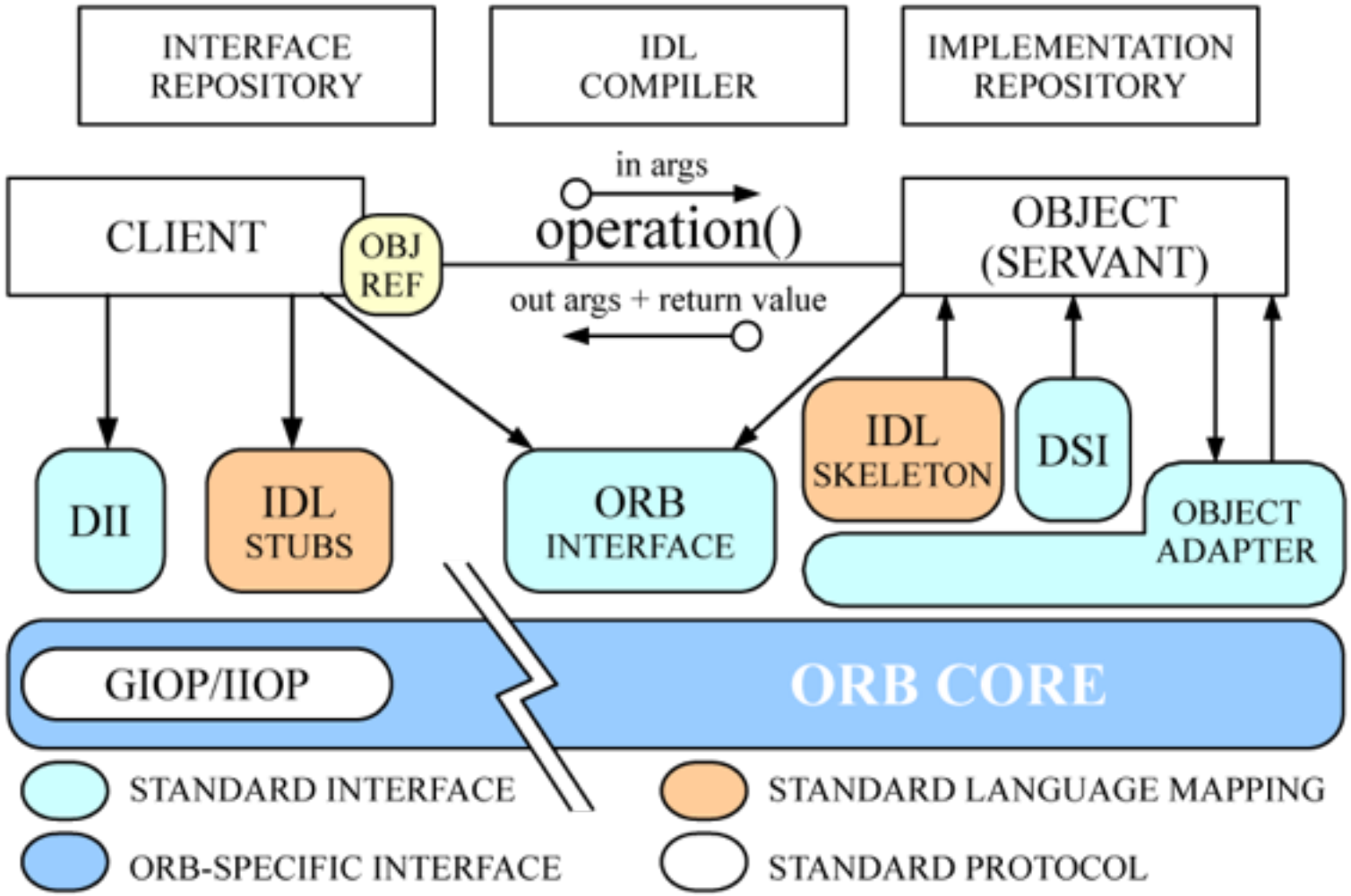
Distributed Systems

- Distributed Systems
  - Additional abstraction layer on top of communication networks
  - Extension of familiar prog. models to distributed systems: e.g., IPC, RPC, DOC
  - Advanced models: e.g., Pub/Sub, Tuple Spaces, DSM, PVM, MPI
- Problems with classical systems in UbiComp context are many
  - Connections assumed to be static
  - Strict (protocol) layering principle
  - Middleware not modular; computers just became more powerful
  - Often not open (CORBA interface IDs)
  - Often simple abstraction (WS -> RPC, REST -> IPC)
  - …





Remote Method Call/Invocation (here: CORBA)



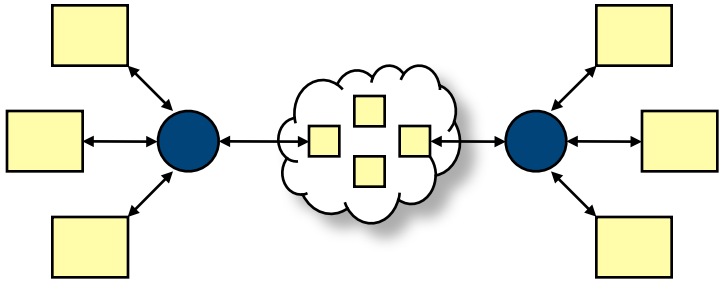
New in UbiComm

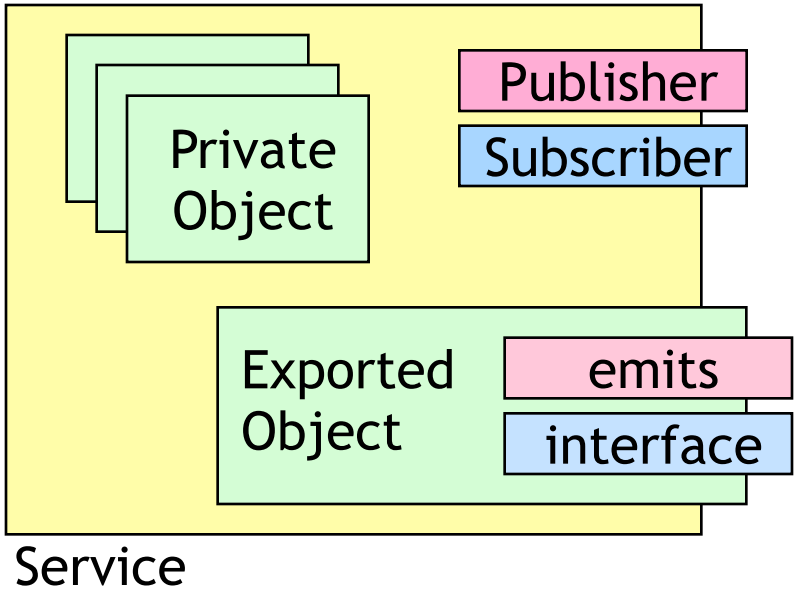
- Some important aspects of ubiquitous communications:

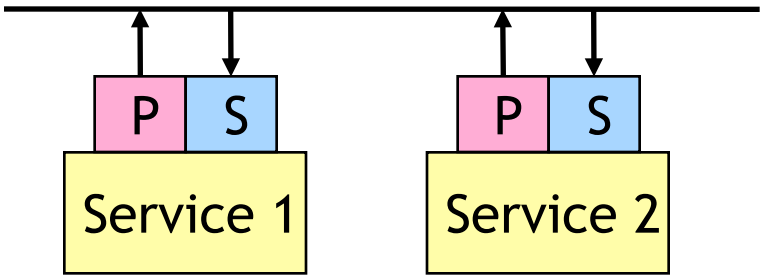
required	solutions
Machine-to-machine communication	Decoupling of communication partners Publish/Subscribe Protocol Heaps
Heterogeneity support	Modular operating systems, middleware, applications
Openness	Peer discovery (node discovery) Decoupling of software components Service discovery Service deployment
Adaptivity	Flexible comm. architectures Enhanced service discovery Connection-aware distributed objects Decentralized control

MundoCore: Architecture

- Communication Microkernel
  - Basic abstraction: Channel-based Publish/Subscribe
- Service
  - Aggregates objects
  - In- and **Out-Ports**
- Channels
  - unicast or multicast

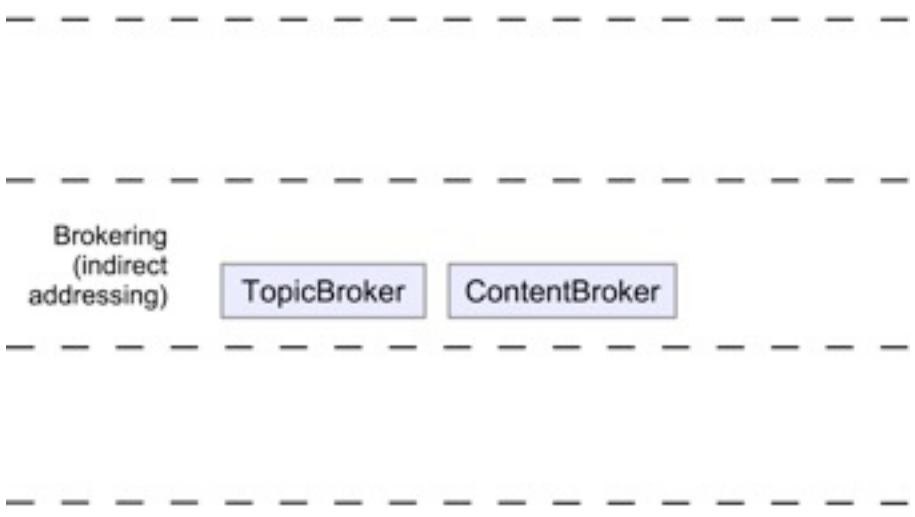






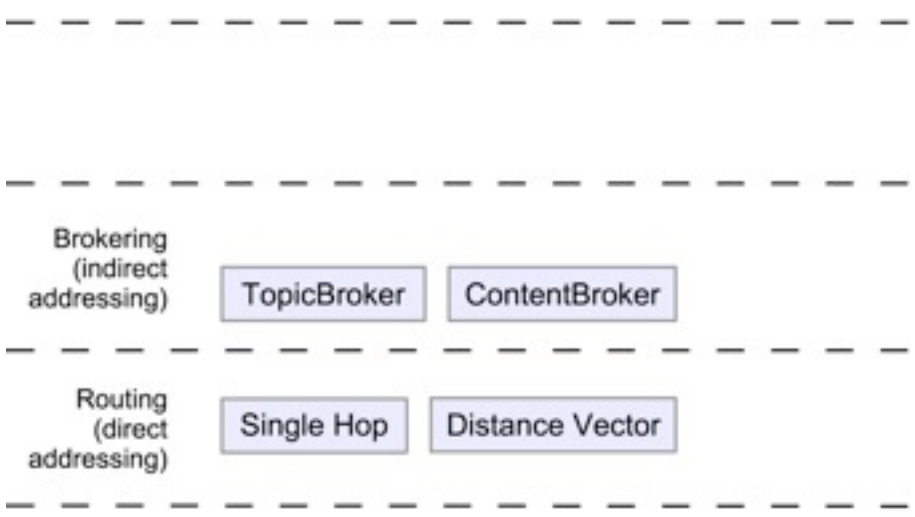
MundoCore: Layers

- Modular
  - Microkernel architecture
  - Services and Components
- Layer Model
  - Groups communication services by function
  - Core: Publish/Subscribe



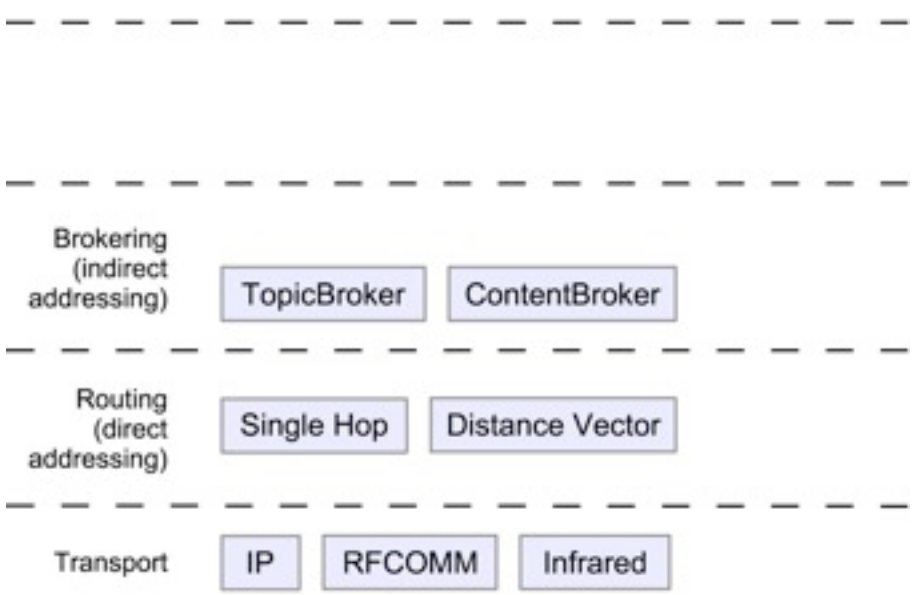
MundoCore: Layers

- Modular
  - Microkernel architecture
  - Services and Components
- Layer Model
  - Groups communication services by function
  - Core: Publish/Subscribe
- Routing: 3 network types
  - Single hop
  - Unstructured P2P
  - Structured P2P



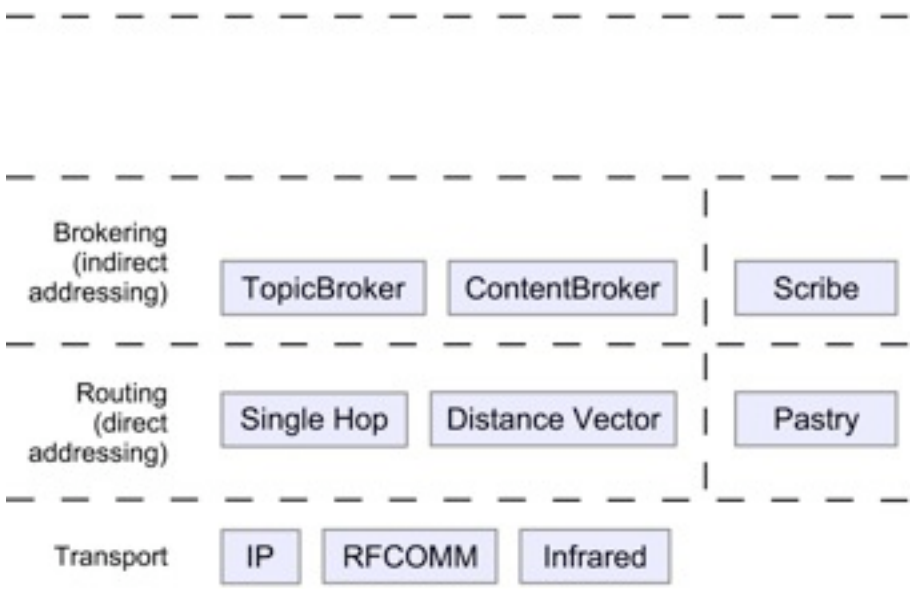
MundoCore: Layers

- Modular
  - Microkernel architecture
  - Services and Components
- Layer Model
  - Groups communication services by function
  - Core: Publish/Subscribe
- Routing: 3 network types
  - Single hop
  - Unstructured P2P
  - Structured P2P
- Transport
  - Discovery of nearby nodes



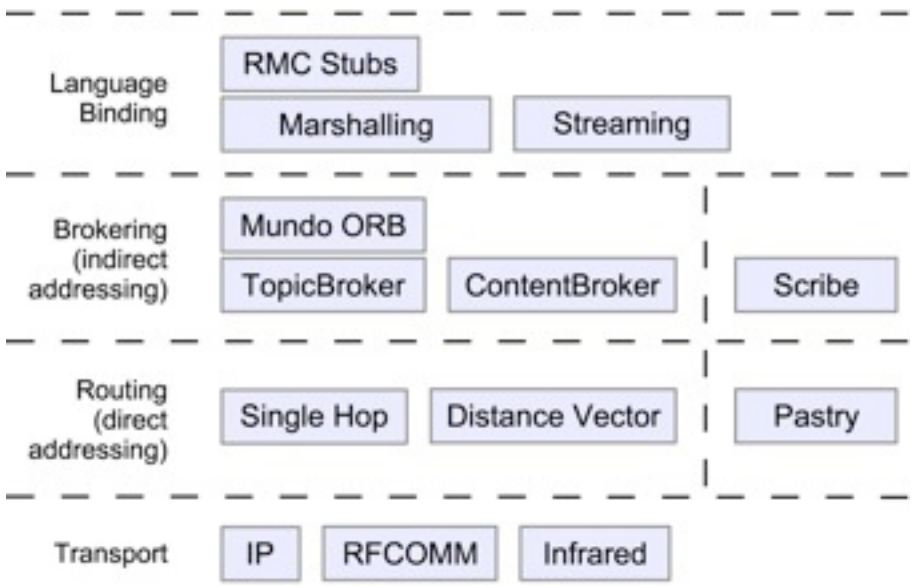
MundoCore: Layers

- Modular
  - Microkernel architecture
  - Services and Components
- Layer Model
  - Groups communication services by function
  - Core: Publish/Subscribe
- Routing: 3 network types
  - Single hop
  - Unstructured P2P
  - Structured P2P
- Transport
  - Discovery of nearby nodes



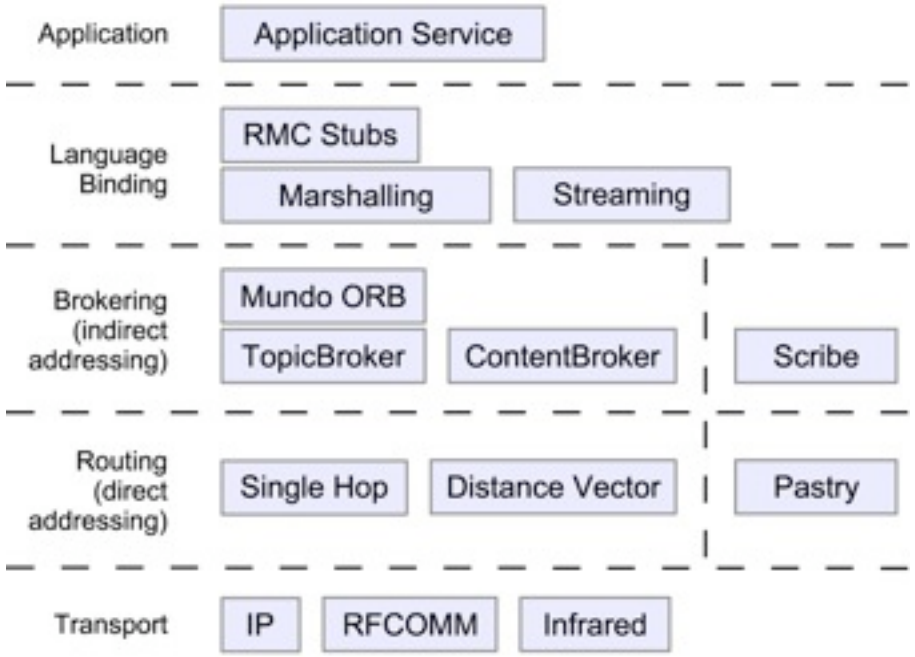
MundoCore: Layers

- Modular
  - Microkernel architecture
  - Services and Components
- Layer Model
  - Groups communication services by function
  - Core: Publish/Subscribe
- Routing: 3 network types
  - Single hop
  - Unstructured P2P
  - Structured P2P
- Transport
  - Discovery of nearby nodes
- Paradigms
  - Publish/Subscribe
  - Distributed OO-Programming
  - Streaming



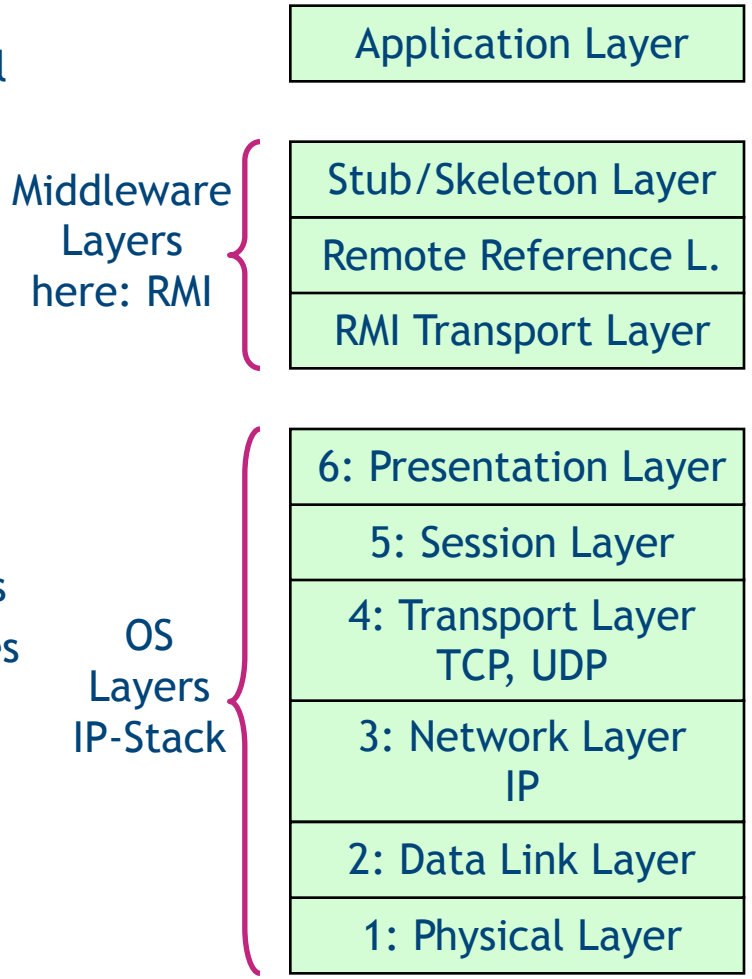
MundoCore: Layers

- Modular
  - Microkernel architecture
  - Services and Components
- Layer Model
  - Groups communication services by function
  - Core: Publish/Subscribe
- Routing: 3 network types
  - Single hop
  - Unstructured P2P
  - Structured P2P
- Transport
  - Discovery of nearby nodes
- Paradigms
  - Publish/Subscribe
  - Distributed OO-Programming
  - Streaming



Protocol Layering (1)

- State of the art: **Protocol Layering**
  - Served well for strict end-to-end model of original Internet
  - Today: many „middle boxes“: firewalls, NAT, proxies, caches
  - Inserted functionality
    - MPLS @layer 2.5
    - IPsec @3.5
    - TLS @4.5
  - Reasons for inserting functionality:
    - Re-use of working implementations
    - Long-standing inter-layer interfaces
    - Coarse granularity of protocol functionality
- Drawbacks of Layering
  - Message flow strictly defined, i.e. a message can only be passed from layer n to layer n+1 or n-1
  - Cannot add functionality in or between layers



Protocol Layering (2)

- Communication in the Internet
  - Mostly file transfers, so TCP/IP is fine
- Communication in Ubiquitous Computing / Smart Environments
  - More machine-to-machine communication ⇨ message-based traffic
  - Sensor data ⇨ low latency required
    - in most cases no retransmissions required
    - sometimes Forward Error Correction (FEC) required for better reliability
  - Media appliances ⇨ streaming with rate control

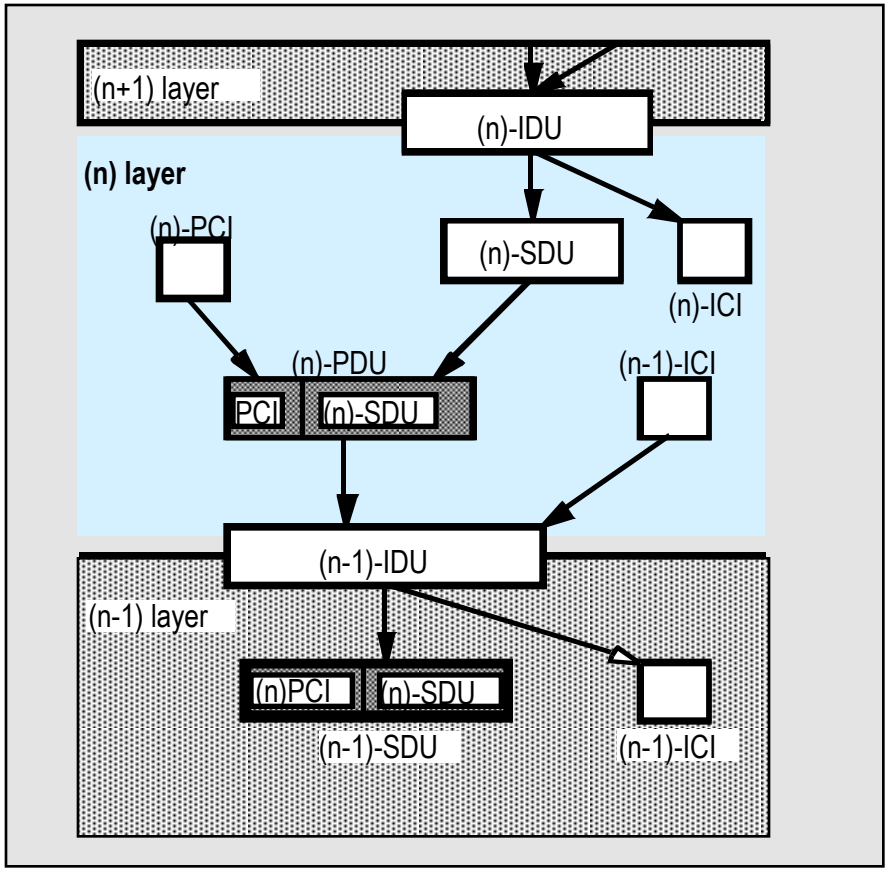
  ⇨ Totally different traffic
- More problems with TCP/IP
  - Poor performance in wireless networks, because of unsuitable congestion control
  - Stream-based, not message-based

Protocol Layering (3)

OSI Model: Data Units

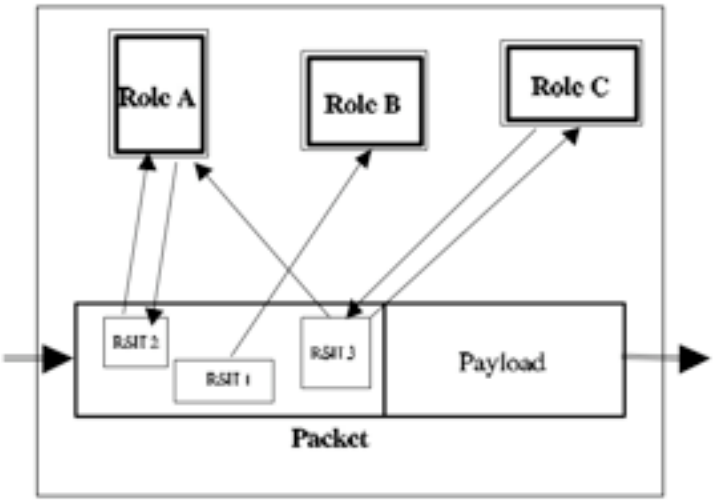
- I = interface
- P = protocol
- S = service

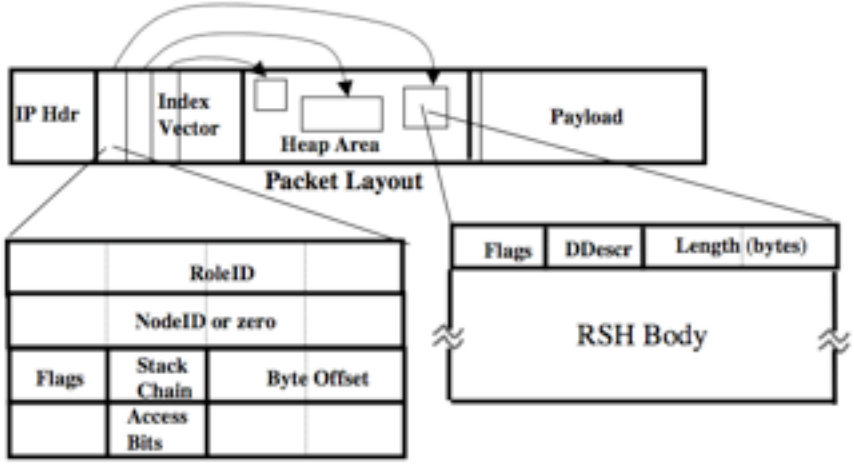
- DU = data unit
- CI = control information



Protocol Heaps

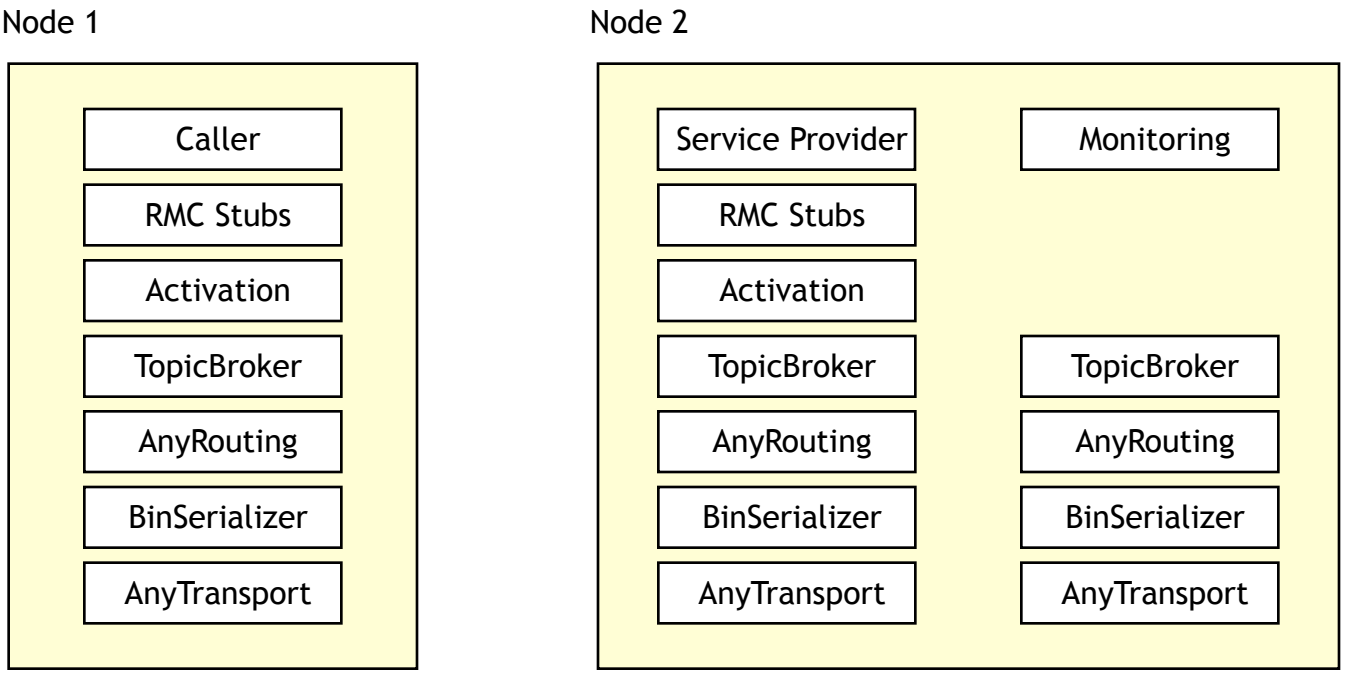
- Role-based Architecture (RBA)
- Instead of last-on/first-off model: random access to RSHs (role-specific headers)
- Problem: Nondeterminism – Which role handler should be invoked next?
- A simple solution is:
  - Pass messages down like in layered architecture
  - Introduce „address“ RSH with MIME-Type and use this info when passing messages up





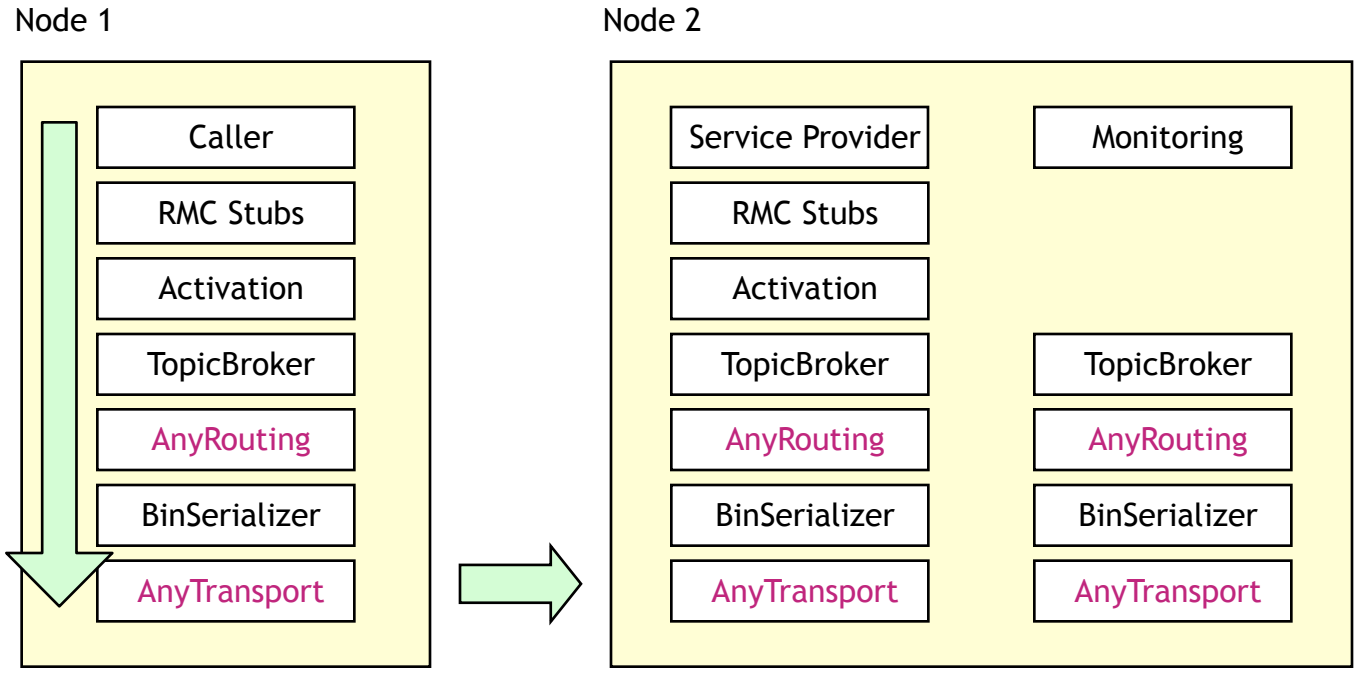
Protocol Heap

- MundoCore
  - Combination of protocol heap and configurable protocol stack



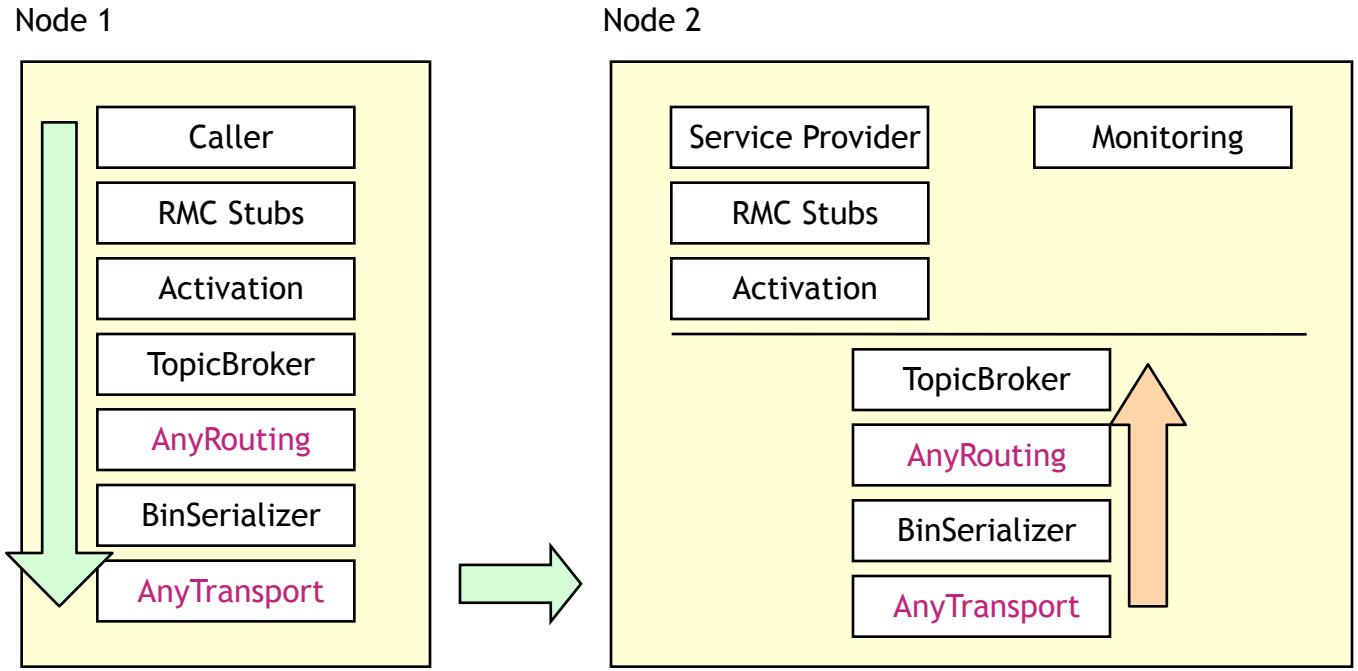
Protocol Heap

- MundoCore
  - Combination of protocol heap and configurable protocol stack
  - Downward processing: defined by stack + **Any*** proxies



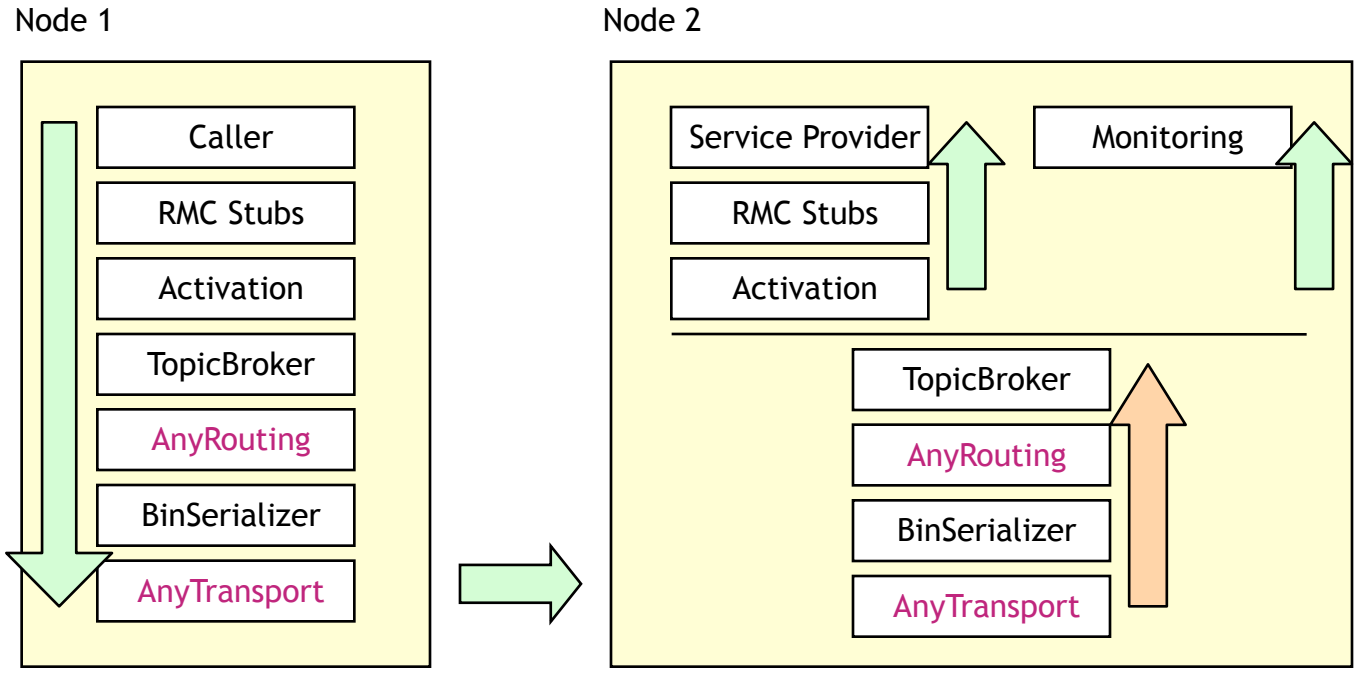
Protocol Heap

- MundoCore
  - Combination of protocol heap and configurable protocol stack
  - Downward processing: defined by stack + **Any*** proxies
  - Upward processing
    - Phase 1: Use MIME-Type to determine next handler



Protocol Heap

- MundoCore
  - Combination of protocol heap and configurable protocol stack
  - Downward processing: defined by stack + **Any*** proxies
  - Upward processing
    - Phase 1: Use MIME-Type to determine next handler
    - Phase 2: Kernel knows service interconnections ⇨ Use stacks

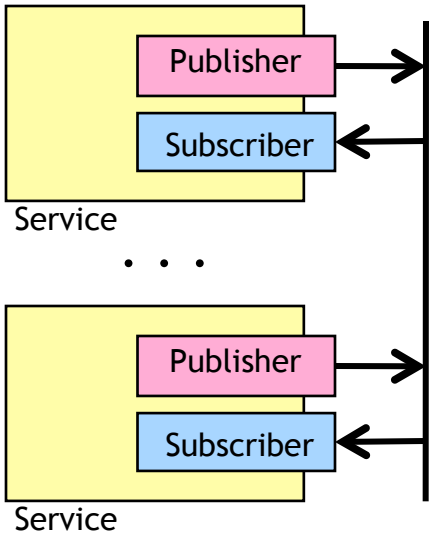


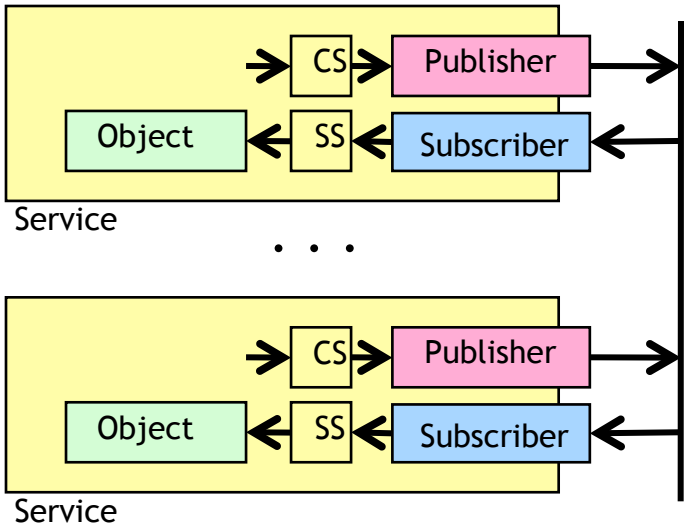
Combination of Pub/Sub and RMC

- Externalization and Serialization
  - are separate concepts in MundoCore
  - **Externalization**: transforms a message containing an active object graph into a message containing a passive data structure.
    A passive message contains typed name-value pairs. Only base types (int, float, String, etc.), arrays, and maps are allowed as types.
  - **Serialization**: transforms a passive message into an XML/SOAP document or into some binary representation
  - Benefits are, e.g.,
    - Send notification using oneway RMC and define pub/sub filter on passive form
- Remote Method Calls
  - Automatic generation of client and server stubs by precompiler
  - Naming service implicitly provided by Pub/Sub
- Connector-Abstraction
  - Automatic generation of connectors by precompiler
  - Java Language Extension with **emits** keyword

Messages vs. RMC

- Note: two different levels of abstraction!





- Send

```
pub = session.publish(zone, channel);
pub.send(message);
```

- Receive

```
sub = session.subscribe(zone, channel,
    new IReceiver() {
        public void received(…
})
```

- Send

```
stub = new DoX();
Signal.connect(stub, …publish(…))
stub.doSomething();
```

- Receive

```
Signal.connect(…subscribe(…),
    targetObject)
```

RMC Client

- Importing an Object

```
DoChatService stub = new DoChatService();
Signal.connect(
    stub,
    getSession().publish("lan", "chat_rmc")
);
```

- Remote call
  - Synchronous

```
stub.chatMessage(ln);
```

  - One-way („fire and forget“)

```
stub.chatMessage(ln, stub.ONEWAY);
```

  - Asynchronous

```
stub.chatMessage(ln, stub.ASYNC);
```

Asynchronous Calls

- Example:

```
class Calculator {
    @mcMethod
    int add(int x, int y);
}
```

- Asynchronous call:

```
AsyncCall callObject = stub.add(1, 2, stub.ASYNC);
…
int result=((Integer)callObject.getObj()).intValue();
```

- Asynchronous call with Callback:

```
AsyncCall callObject = stub.add(1, 2, stub.CREATEONLY);
callObject.setResultListener(
    new AsyncCall.IResultListener() {
    public void resultReceived(AsyncCall callObject) {
        int rslt = ((Integer)callObject.getObj()).intValue();
    }
});
callObject.invoke();
```

Serialization

- Example:

```
@mcSerialize
public class Message {
    public String text;
    public Message() {
    }
    public Message(String t) {
        text = t;
    }
}
```

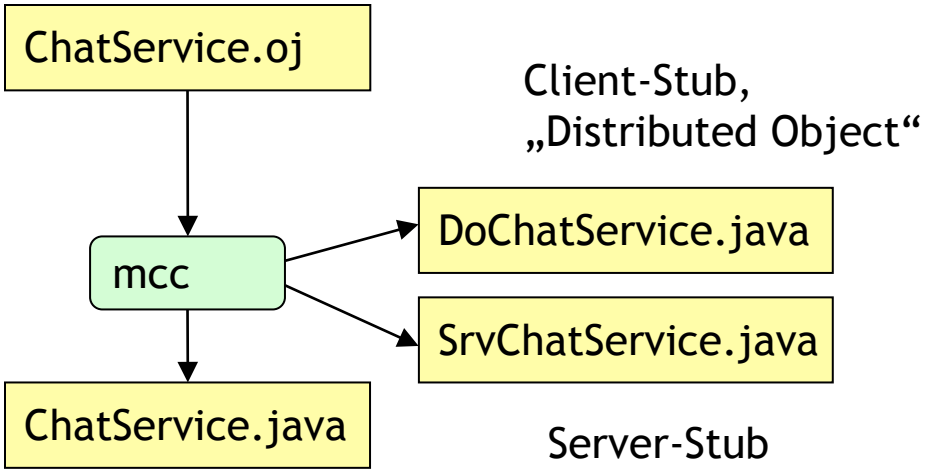
- Note
  - @mcSerialize: mcc generates serializers for all non-transient fields
  - Class must have public nullary constructor – otherwise it cannot be instantiated by reflection

Parameter Marshalling

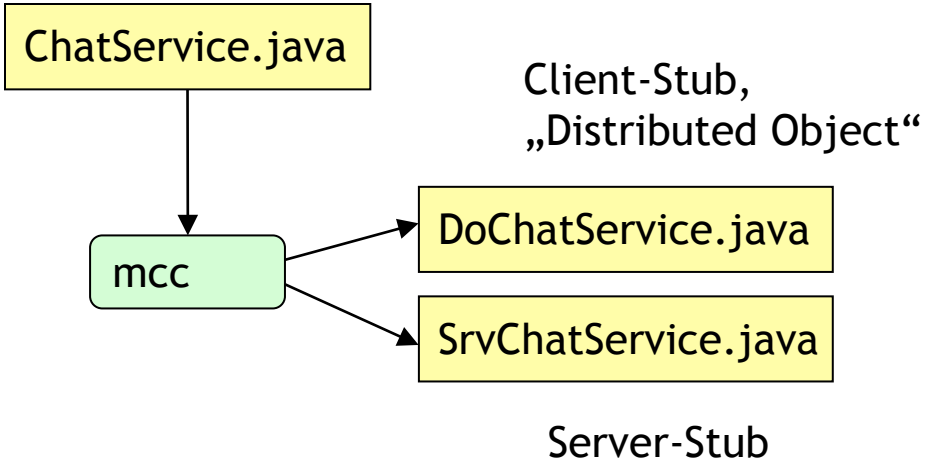
- Local:
  - Primitive types: by-value
  - Objects: by-reference
- RMC:
  - Primitive types: by-value
  - Objects: by-value
    Deep copy by serialization
  - Distributed Objects: by-reference
    (Classes with Do-Prefix)
- Client receives Reference to remote object
  - As return value of an RMC call
  - By using the Signal.connect method (cf. RMC Client example)

MundoCore: Build Process

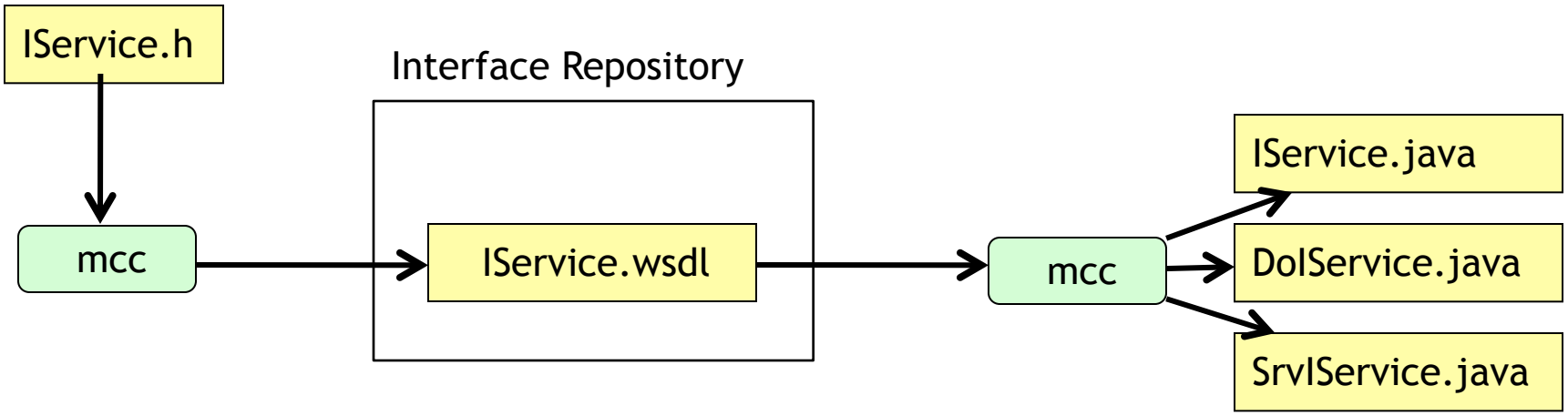
- Precompilation for Java 1.1-



- Precompilation for Java 1.5-

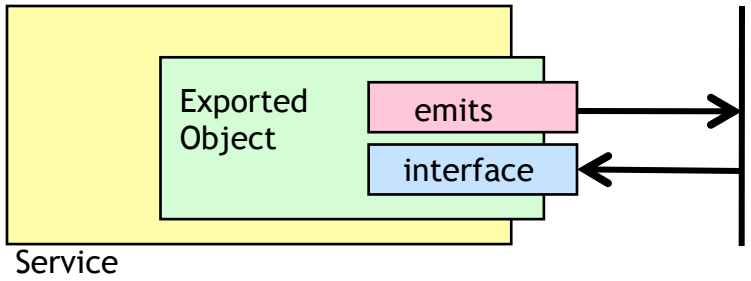


MundoCore: Interoperability



- MundoCore uses WSDL
  - common representation of interfaces
  - extension: data objects
    - description for object serialization
    - constants also supported

Decoupling of Services




```
Signal.connect(sourceObject,
   …publish(…))
```

```
Signal.connect(…subscribe(…),
    targetObject)
```

Will be possible in future releases:

```
Signal.connect(doSource, doTarget)
```

- Support for output interfaces in MundoCore
  - Java language extension, handled by mcc
- Services can be connected from “outside”
  - No explicit creation of distributed objects, connects, channel names in service code
- MundoCore is aware of all links between services
  - Allows dynamic service loading/unloading, persistence, migration
  - Allows separate evolution of connectors

MundoCore: Output interfaces

	inbound	outbound

oneway	**Procedure call**
	class C implements I {
	  void m() {}
	}
	
	i.m();
		Signal event
		class C emits I {
		  void e() {}
		}
		
		emit.e();

request/ reply	**Function call**
	class C implements I {
	  int m() { return 3; }
	}
	
	x=i.m();
		Anonymous request/reply
		class C emits I {
		  int m() {}
		}
		
		x=emit.m();

MundoCore: Peer Discovery

- Rendezvous via the primary port
  - All nodes in the same overlay network share the same primary port (default: 4242)
  - If at least one MundoCore-process is running on a host, then this port is bound by some process
- UDP Broadcast
  - Reaches all nodes configured with the same primary port in the same subnet
- UDP Multicast
  - Reaches all nodes in the same multicast group (configuration setting)
  - Multicast packets can also reach other subnets, if supported by router
- Neighbor-Messages
  - Uses R-Multicast provided by routing services
  - All-or-nothing-semantics: A new node can either use all services available or does not get access to them at all.

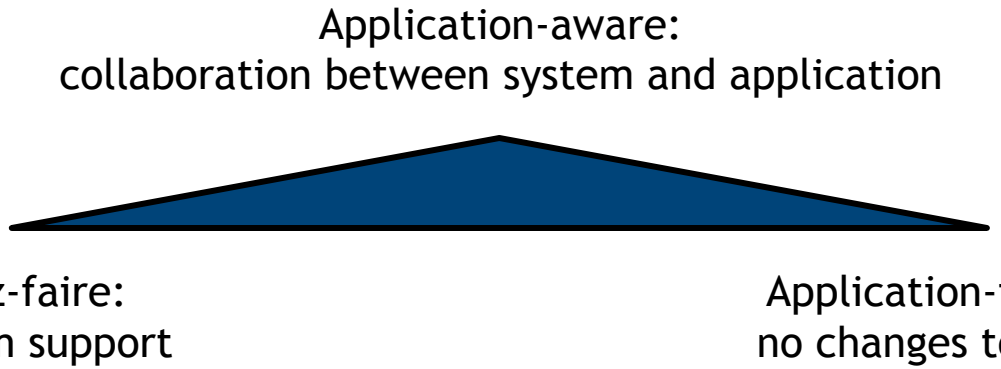
MundoCore: Service Discovery

- Builds internally on content-based publish/subscribe
  - Services offered by a node are published
    - when the node joins/leaves the network
    - when services are registered/unregistered
    - on explicit query
- In most cases: discovery based on interfaces
- Service Query:

```
filter = new ServiceInfoFilter();
filter.filterInterface("org.mundo.service.IIODevice");
filter.zone = "lan“;
filter._op_zone = IFilter.OP_EQUAL;
ResultSet rs = ServiceManager.getInstance().query(filter, null);
Thread.sleep(1000);
System.out.println(rs);
```

- Continuous Query with contQuery()
  - Listener receives inserted, removing, removed, propChanged events until result set is closed

MundoCore: Adaptivity



- Spectrum of adaptation strategies (Satyanarayanan,1996)
- Application-transparent:
  - Switching between different transport and routing services
  - Peer discovery: only discovery strategies are configured, no addresses
  - Service discovery, migration
  - Conditional protocol handlers
- Application-aware:
  - Process node join/leave events
  - Process service register/unregister events
  - Process objectConnected/objectDisconnected events

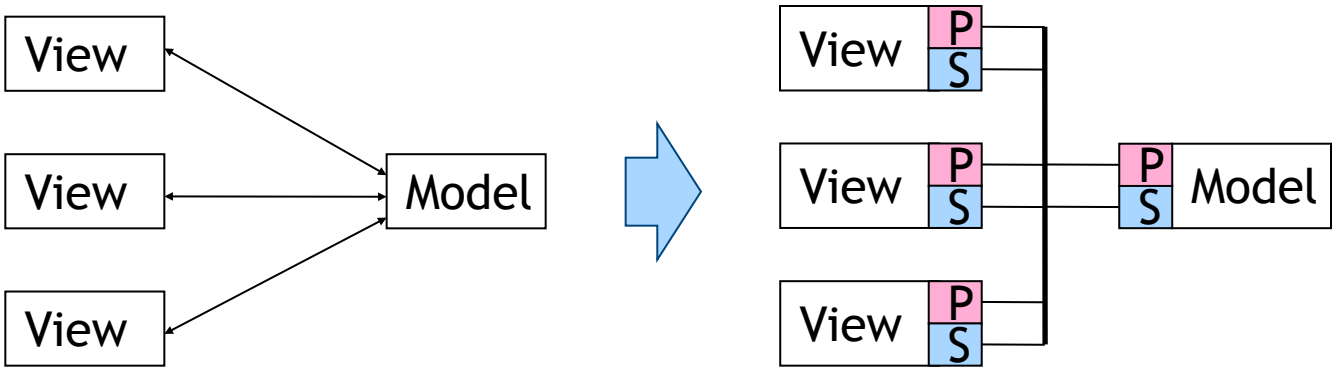
Conditional Protocol Handlers

- Example: (default stack defined in node.conf)

```
<default-stack xsi:type="array">
   <handler>org.mundo.net.ActivationService</handler>
   <handler>org.mundo.net.P2PTopicBroker</handler>
   <handler>org.mundo.net.RoutingService</handler>
   <if xsi:type="map">
      <condition>org.mundo.net.ip.IfUDP</condition>
      <then xsi:type="array">
         <handler>org.mundo.net.NAckHandler</handler>
         <handler>org.mundo.net.BinSerializationHandler</handler>
         <handler>org.mundo.net.BinFragHandler</handler>
      </then>
      <else xsi:type="array">
         <handler>org.mundo.net.BinSerializationHandler</handler>
      </else>
   </if>
   <handler>org.mundo.net.ip.IPTransportService</handler>
</default-stack>
```

Decentralized Control

- Example: MVC Pattern implemented with Publish/Subscribe
  - Model does not have to keep track of views
  - Easy to support multiple models -> decentralized control



Synchronous RMC

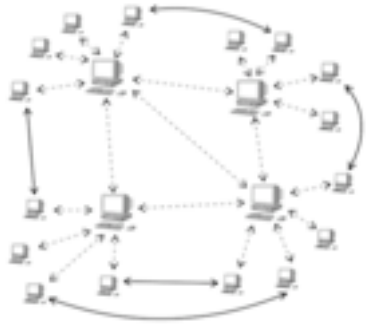
Publish/Subscribe

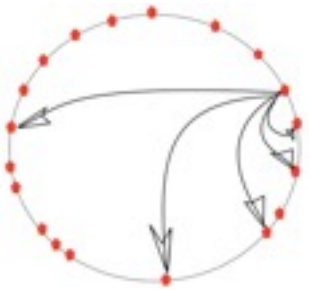
Network Types

- Overlay networks add an abstraction layer atop the phys. network
  - … to implement content-based addressing, caching/replication, better fault tolerance, etc.
  - In the following, we consider object location in P2P systems as example
- We can distinguish two main types
  - **Unstructured networks**
    - based on searching: networks find objects by searching with keywords that match objects' descriptions
      - Pro: No need to know unique names or hashes
      - Con: Hard to make efficient
    - Unstructured does **not** mean complete lack of structure: Network has structure, but peers are free to join anywhere and objects can be stored anywhere
  - **Structured networks**
    - based on addressing: networks find objects by addressing them with their unique name, hash value, etc.
      - Pro: Object location can be made efficient
      - Con: Unique name/hash must be known, no wildcard searches
    - Network structure determines where peers belong in the network and where objects are stored

MundoCore: Network Types

- Unstructured
  - Single-hop routing
    - Fully meshed network
    - Scales up to ~30 clients
    - Best reliability
    - MundoCore: P2PTopicBroker
  - Super-peer network
    - Only super-peers are fully meshed
    - Only super-peers run full pub/sub-broker
    - Leaf nodes are only pub/sub clients
    - MundoCore: DVEventRouter
- Structured
  - MundoCore: PastryERS
- Hybrid configurations possible





MundoCore: Summary

- Different traffic in UbiComp
  - Three kinds: event-based, request/reply, (media) streaming
- Flexible communication architectures
  - Modular middleware; case study: MundoCore
  - structured messages ⇨ event filtering (brokering layer)
  - typing ⇨ RMC and object marshalling (language binding layer)
- Adaptivity
  - Peer discovery, Service discovery
  - Application-transparent adaptation: “Any-Proxies”, conditional protocol handlers, peer and service discovery, anonymous request/reply
  - Application-aware adaptation: node join/leave, service register/ unregister, object connect/disconnect
- Decoupling
  - …of communication: publish/subscribe
  - …of services: Output interfaces, connector abstraction
    - ListenerList bookkeeping should not be part of event source
  - Rethink traditional implementations