Alma Mater Studiorum – University of Bologna
CdS Laurea Magistrale (MSc) in
Computer Science Engineering

Mobile Systems M course (8 ECTS)
II Term – Academic Year 2021/2022

06 – Discovery, Messaging, and Events

Paolo Bellavista
paolo.bellavista@unibo.it

http://lia.disi.unibo.it/Courses/sm2122-info/

Resource/Service Discovery

Sometimes the term is used in a broad sense to indicate both the “real” discovery and also the configuration operations needed to access resources/services, as well as the resource/service requests themselves

Key support features for any open, dynamic, loosely coupled, and peer-to-peer system:

- ***Automated configuration***
- ***Discovery of resources and services***
- ***Resource/service delivery***

Main discovery standards and solutions:

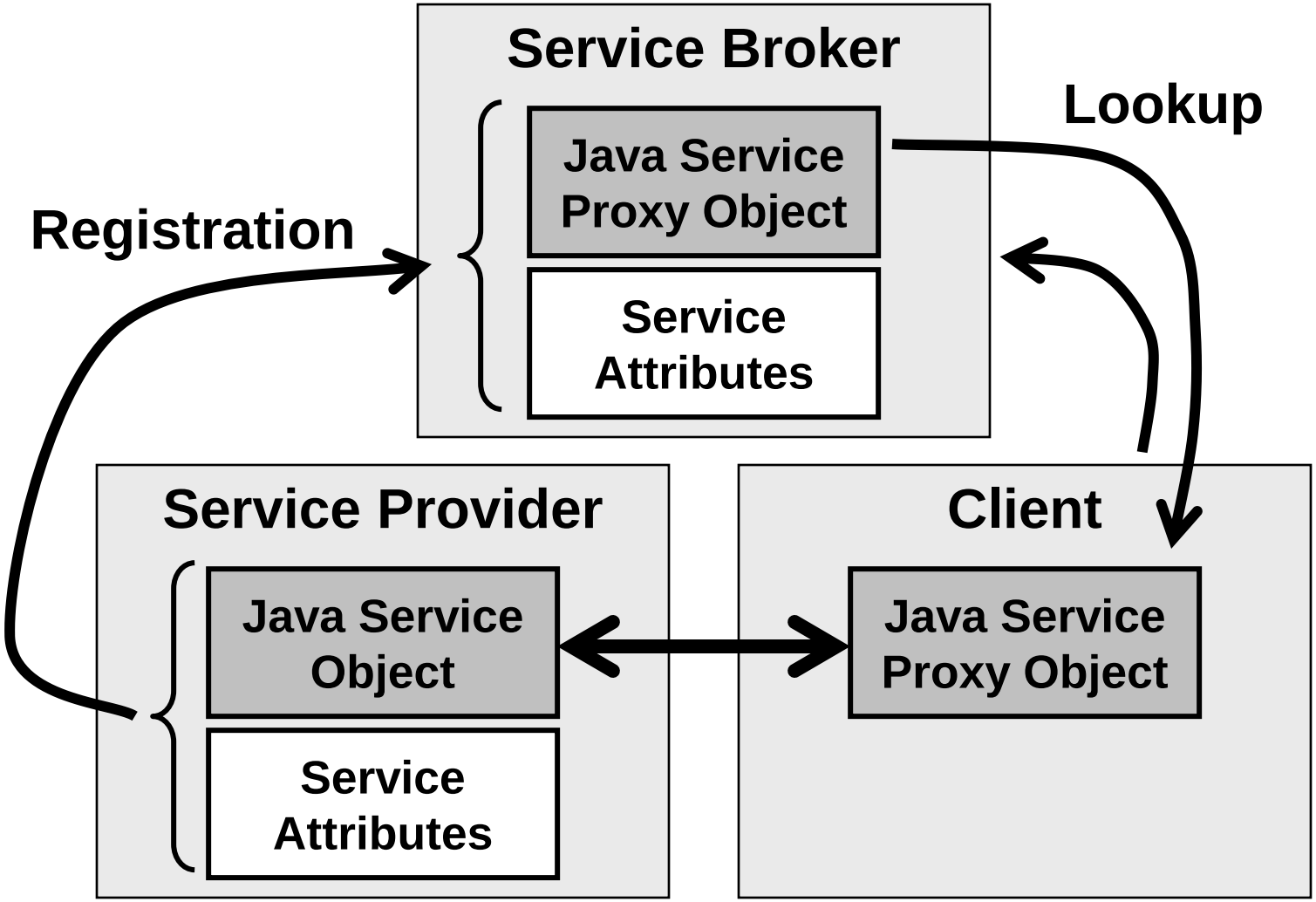
- Jini, Service Location Protocol (SLP), Universal Plug and Play (UPnP), …

Key Features

- ***Auto-configuration***
  - Device have to configure themselves to participate to offering/requesting resources and services
  - For instance, but not only, configuration of a temporary IP address in the current locality
- ***Discovery*** of available resources and services
  - Who offers resources and services (*service provider*) has to be able to make ***advertising of them***
    - Of which resources and services
    - Of how to make invocations, via which interfaces
  - Clients have to be able to ***find (local)*** resources and services
- ***Access*** to resources and services
  - Clients have to be able to communicate with service provider, invoke services, transfer input parameters, and possibly receive return results
  - Also support to authentication and authorization

- Relevance of achieving good ***reliability and scalability***

Jini
(Sun Microsystem, now Apache River)



Apache River

Most relevant discovery solution for the Java world

- Service provider dynamically discovers ***one or more lookup services (broker)***
- Service provider ***registers a resource/service object*** (modeled as Java object) and its attributes to the broker
- Client ***requests a service***, typically by specifying attributes of the looked-for service; ***one instance of object to simplify resource/service access*** moves to client at runtime
- Lookup service can ***notify registered clients*** when there is state change for their resources and services
- Client ***interacts with discovered resource/service via obtained Java object***

Reliability Management in River

- Possible failures (and not only, see versioning) are managed through ***lease mechanism***
  - River assigns resources to clients with ***lease of given time duration*** (less or equal to what requested by client)
  - Once terminated the lease interval, client has to ***re-fresh lease*** in order to continue accessing the resource/service
  - Also lookup registration is made with lease: therefore, all leases have an expiration deadline, for any user, in the case of “long” service provider fault
- River supports ***redundancy at the infrastructure level and resiliency against faults***
  - Possible to ***deploy several lookup services*** in the same network
  - Service providers can ***register their proxy objects in multiple lookup services***
  - Also usage within transactions and ***automated rollback when lease expires***

Scalability in Apache River

Scalability realized via ***dynamic organization in "communities" or "federations"***

- Groups of River resources/services can aggregate into a community, typically the one of ***local services*** registered at least in a local lookup service
- Different communities can be ***linked together in larger groups through lookup service***
  - One community registers itself at other communities by registering its own lookup service
  - How to manage the nesting of lookup services?

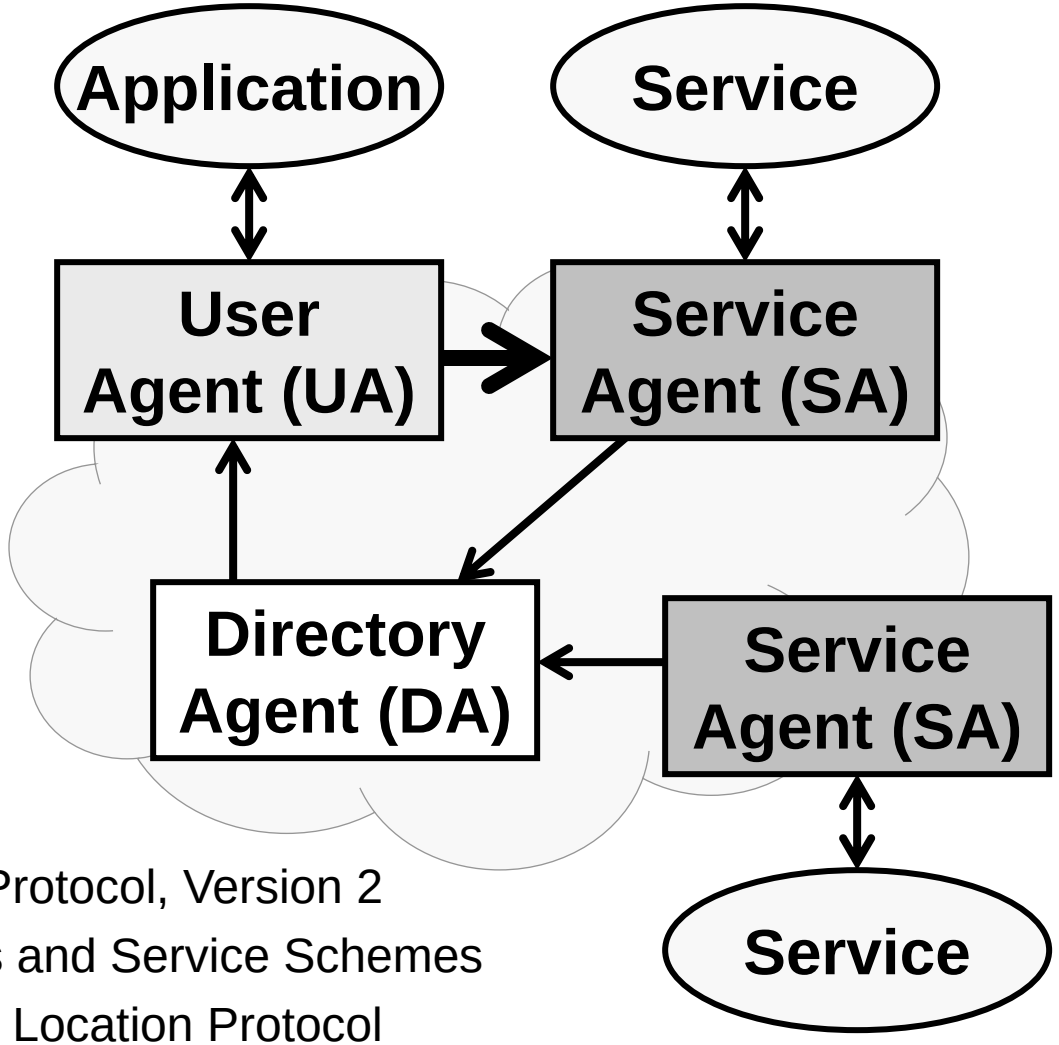
Service Proxy Object in River is Dynamic

If compared with more traditional solutions for resource/service access, anyway retrieved dynamically via intermediate stubs and skeletons:

- River overcomes ***limitations of stub/skeleton creation at compile-time***, which is for example typical of RPC
- River allow to clients ***to obtain service provider stubs at provisioning time***
- RPS stubs are similar to ***service proxy object that the River server dynamically loads in the lookup service***
- Service proxy object allows clients to use the discovered service with NO a priori knowledge of its implementation details

Service Location Protocol (SLP)

- SLP is the ***standard approach of Internet Engineering Task Force (IETF)*** for resource/service discovery
- Basic idea: SLP makes resources/services visible through URL registration at intermediate agents



- RFC 2608 - Service Location Protocol, Version 2
- RFC 2609 - Service Templates and Service Schemes
- RFC 2614 - An API for Service Location Protocol

SLP is based on 3 Types of Agents

Agents as entities capable of SLP message processing

- ***Service agent***
  - Performs broadcast (usually periodic) of advertisement messages about its resources/services (associations with URLs)
- ***Directory agent*** (optional)
  - Performs caching of advertisement messages (from service agents) as a centralized repository
  - Processes discovery queries received from user agents by returning back the URLs that match
- ***User agent***
  - To discover resources/services at the client side
  - Also efficient usage of multicast to service agent groups

Standard specification is relatively rich and flexible, but industry-mature implementations not so widespread (OpenSLP, Sun SLP, Xerox printers, …)

Universal Plug-and-Play (UPnP)

Standard specification started by Microsoft (at the beginning, an internal proprietary solution)

- Primary goal: to enable advertisement, discovery, and control of networked devices, services, ***consumer electronics in typically domestic ad hoc envs*** (see the current exploitation in media centers, tvs, hi-fi devices, …)
- UPnP uses, as underlying technologies:
  - UDP or TCP/IP
  - HTTP
  - XML/HTML and SOAP

Universal Plug-and-Play (UPnP)

Via UPnP a device can:

- Dynamically ***join a network***, by obtaining an ***IP address that is locally valid***
- Making visible ***its capabilities***, by need and on-demand
- ***Discover the presence and capabilities*** of other devices
- Dynamically leave a network

UPnP supports:

- Automated IP configuration
- Discovery of resources and services
- ***Description of resources/services based on XML***
- ***Service control based on SOAP***
- ***Event management*** (via Generic Eventing and Notification Architecture - GENA)
- Presentation in HTML/XML

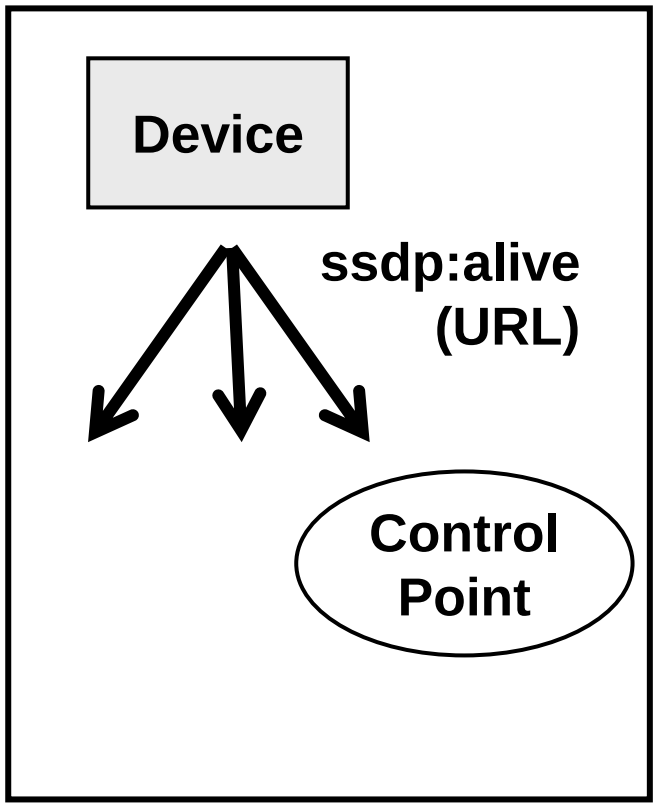
IP Addressing in UPnP

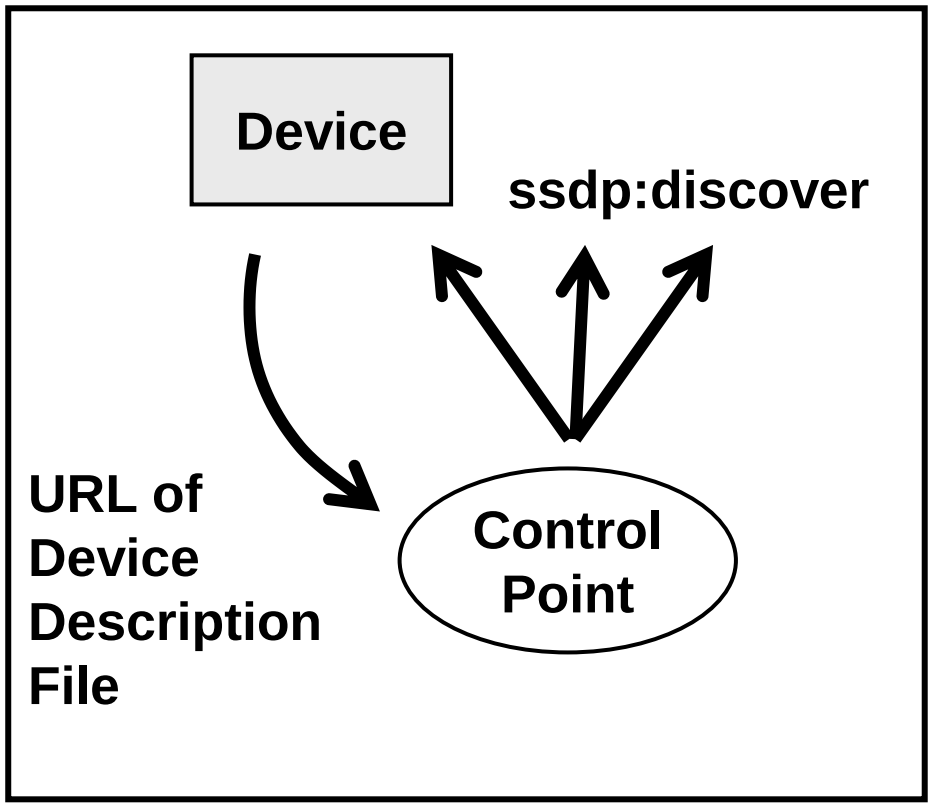
By delving into finer technical details:

- ***UPnP uses Auto IP*** (the real protocol part for auto-configuration) to enable devices to connect to the network with NO need of explicit administration
- When a device connects to a network, it tries to ***obtain an IP address*** from a DHCP server, if available
- If no DHCP server is available, an IP address is ***automatically claimed from a fixed reserved range for usage ONLY at local network***
  - An IP address from the link-local address range is ***selected randomly*** (169.254.0.0/16 for IPv4)
  - Request sent via Address Resolution Protocol (ARP) to check whether other devices have already picked up that address

Service Discovery in UPnP

UPnP exploits the Simple Service Discovery Protocol (SSDP) as discovery protocol based on ***usage of dedicated multicast address*** (239.255.255.250 on port 1900 via UDP)



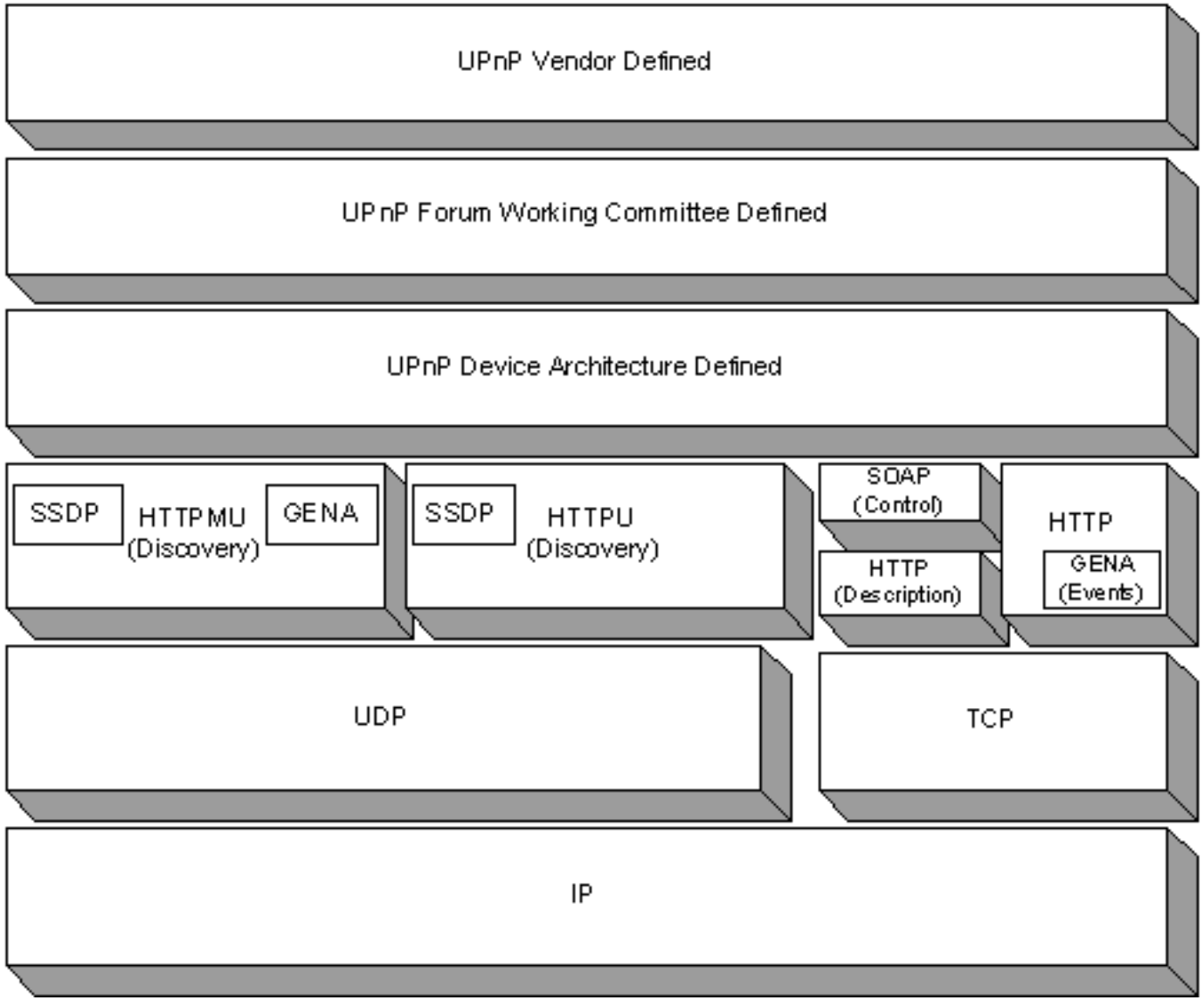


Service Discovery in UPnP

- Device (e.g., UPnP-compliant projector) ***multicasts*** an advertisement message ***(ssdp:alive)*** to ***exhibit its services to active control points*** (e.g., tablets or home gateways)
- Control point can perform ***multicast*** of ***search messages (ssdp:discover)***
  - Any device that receives a multicast message may reply with a ***unicast response message***
- The URL of XML Device Description File is returned back to the control point

UPnP Architecture and Protocol Stack

Usage of HTTP over UDP, either multicast (HTTPMU) or unicast (HTTPU)



Description of Device and its Services in UPnP

- UPnP uses ***XML to describe resources and services*** (standardization effort for interoperable representation)
- ***Advertisement message includes a URL*** related to the XML device description file
- Device description file describes the ***capabilities of the device*** for which advertisement is done
- Control point can dynamically obtain the device description file via HTTP and process it
- Any device can offer one or more resources/services
- **`<service>`** element includes
  - Service type and service ID
  - Service URL for ***invocation via SOAP***
  - URL for event subscription to enable ***notifications***
  - An additional file (Service Description File) for any offered service, with more specific and detailed descriptions

Device Description File for a Projector

projector-desc.xml

```
<?xml version="1.0" ?>
<root xmlns="urn:schemas-upnp-org:device-1-0">
<device>
  <deviceType>urn:schemas-upnp-
   org:device:projector:1</deviceType>
  <UDN>uuid:UPnP-Projector</UDN>
  <serviceList>
  <service>
  <serviceType>urn:schemas-upnp-
   org:service:control:1</serviceType>
  <serviceId>urn:upnp-org:serviceId:control</serviceId>
  <controlURL>isapictl.dll?control</controlURL>
  <eventSubURL>isapictl.dll?control</eventSubURL>
  <SCPDURL>projector-scpd.xml</SCPDURL>
  </service>
  </serviceList>
</device>
</root>
```

Service Description File for a possible "projector control" service

Service Control in UPnP

- The XML-based service description file (e.g., “projector control”) contains:
  - ***Action list with the operations*** that may be invoked on the service
  - ***Service state table*** including all exposed state variables (and their data type)
- To invoke a given service control for which a device has previously performed advertisement, ***control point sends a SOAP message to the URL specified*** for that service
  - ***Control point can access and update state variables in the table***
- Service performs the requested control action and ***returns the result via SOAP message***

Service Description File for a Service of Projector Control

projector-scpd.xml

```
<?xml version="1.0" ?>
<scpd xmlns="urn:schemas-upnp-org:service-1-0">
<actionList>
<action>
  <name>SetPower</name>
   <argumentList>
        <argument>
        <name>Power</name>
         <relatedStateVariable>Power</relatedStateVariable>
               <direction>in</direction>
        </argument>
      </argumentList>
</action>
 … Other actions …
</actionList>
```

Service Description File for a Service of Projector Control

```
…
<serviceStateTable>
    <stateVariable sendEvents="yes">
        <name>Power</name>
        <dataType>Boolean</dataType>
        <defaultValue>0</defaultValue>
     </stateVariable>
    <stateVariable sendEvents="yes">
        <name>File</name>
        <dataType>string</dataType>
        <defaultValue>default.ppt</defaultValue>
    </stateVariable>
… Other state variables …
</serviceStateTable>
</scpd>
```

Event Subscription in UPnP

- ***Control point may register itself for receiving notification events*** generated by advertised services when ***state variables are modified***
  - ***Subscription URL*** included in device description file
- Messages that implement the notified event are ***expressed in XML and formatted according to the General Event Notification Architecture (GENA) standard***; they include the modified state variables that caused event generation

For example, projector service can notify events towards control point when one of the following situations occur:

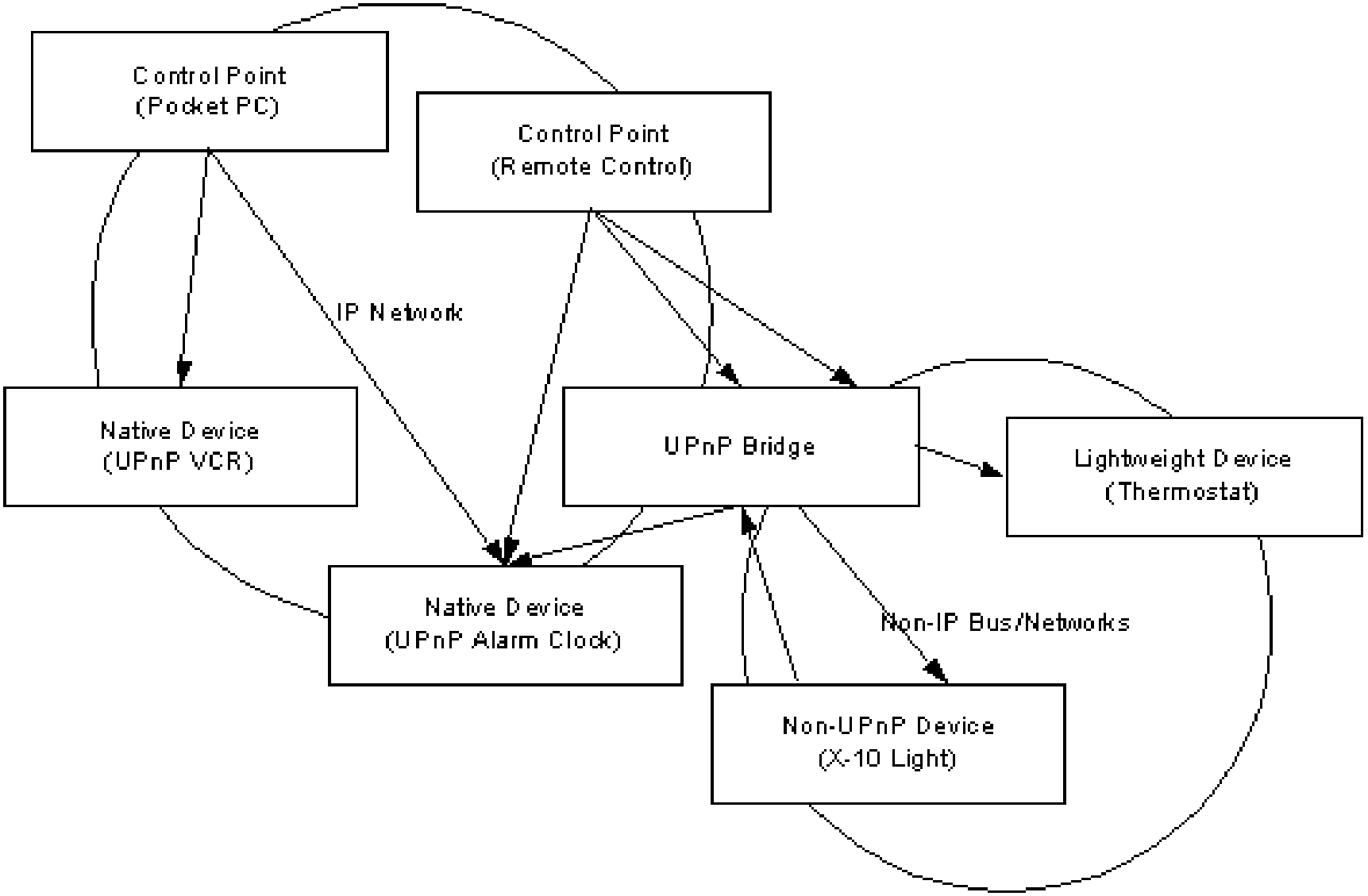
- Page up/down (change of pageNumber variable)
- Power on/off
- Files – change in ppt file list
- File – change in ppt file currently with ongoing presentation

UPnP ***does NOT allow subscription of control points to single state variables***; control point has to dynamically determine which state variable has been changed and has generated notification event

UPnP Presentation is based on HTTP

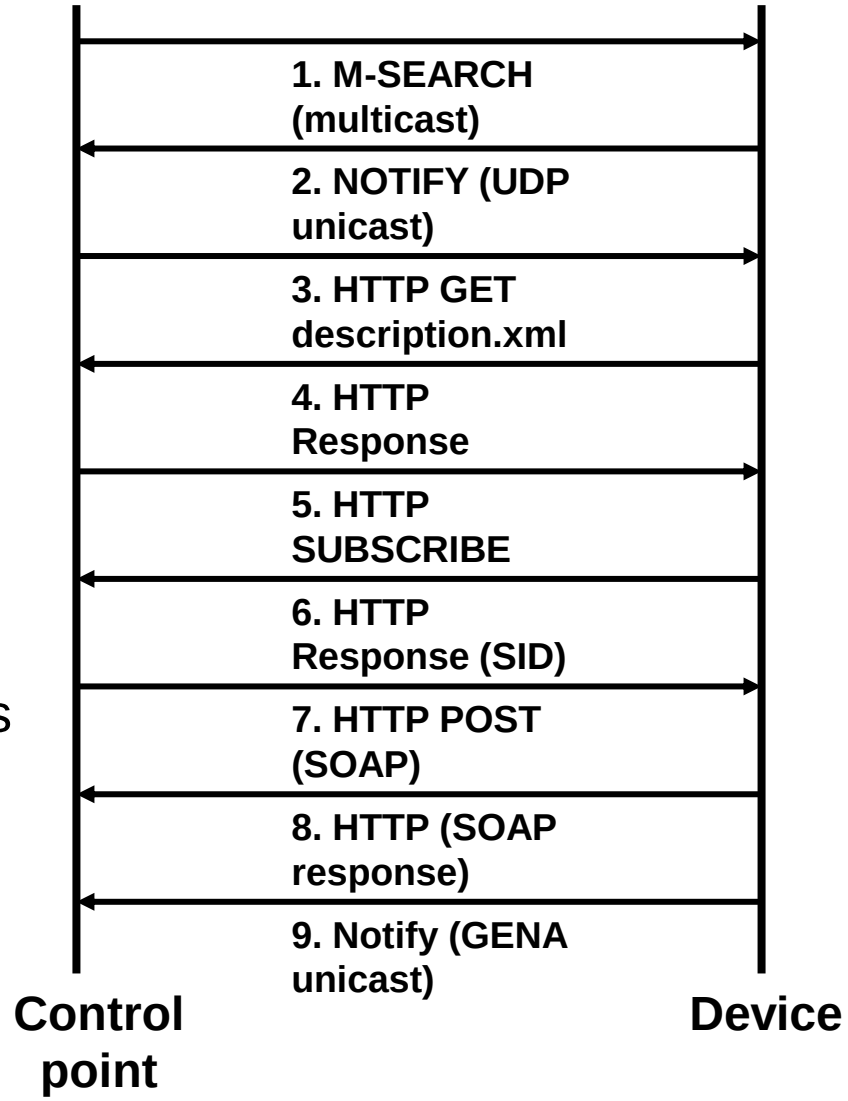
- Device may ***advertise a "presentation" URL for user interface describing service access via Web***:
  - Download Web page from URL and visualization at browser
  - Possibility for users to control the device
  - Possibility to visualize device state
- ***Given the usage of XML*** for data definition and exchange, ***UPnP potentially enables*** employment by a ***large set of resource-limited devices***: also ***automated XML transformations (reductions) based on XSLT***
  - Most UPnP implementations support only presentation based on HTML, slow transition towards XML

UPnP Architectue and Bridging Possibilities



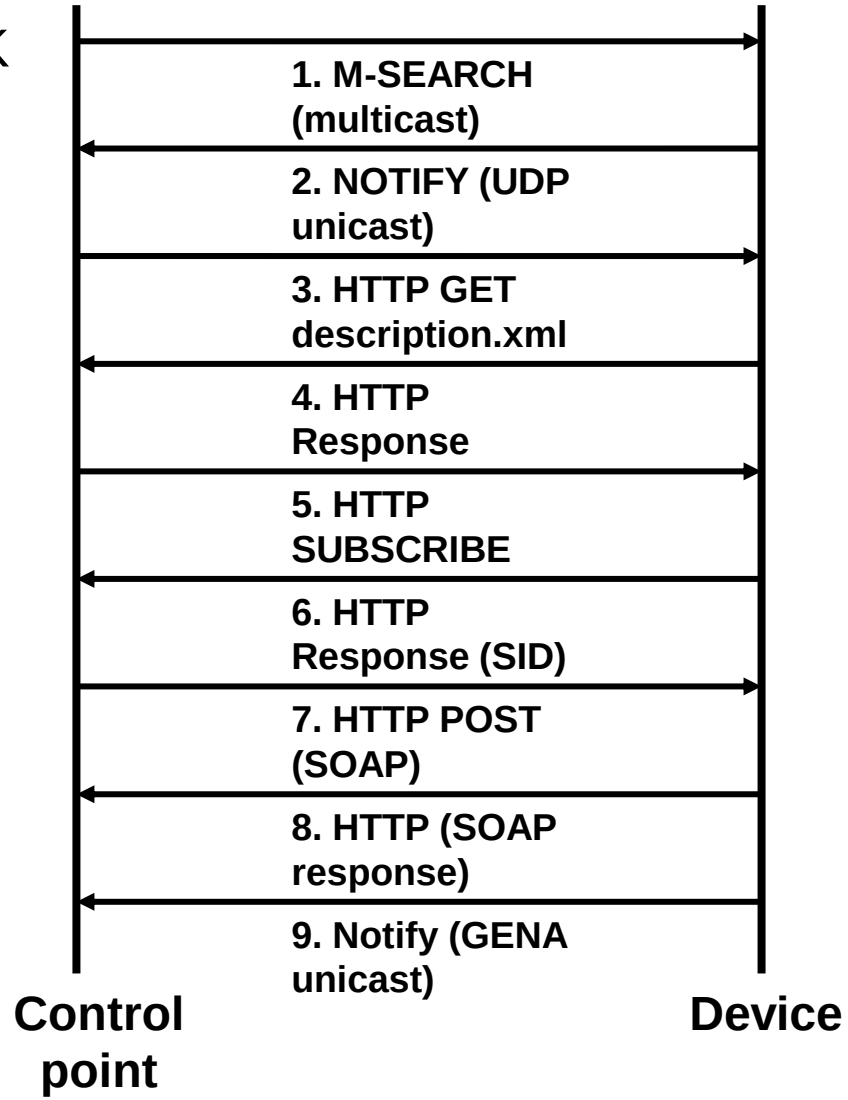
Details on Service-Control Point Interaction

1. Control point sends SSDP search request
2. Device replies with unicast UDP NOTIFY, which contains the URL of XML file with device description
3. Control point requests XML description document via HTTP
4. ***Web server included in the device*** replies to request and returns XML document
5. To automatically receive notifications of changes at device, control point can register itself to the services of interest via HTTP



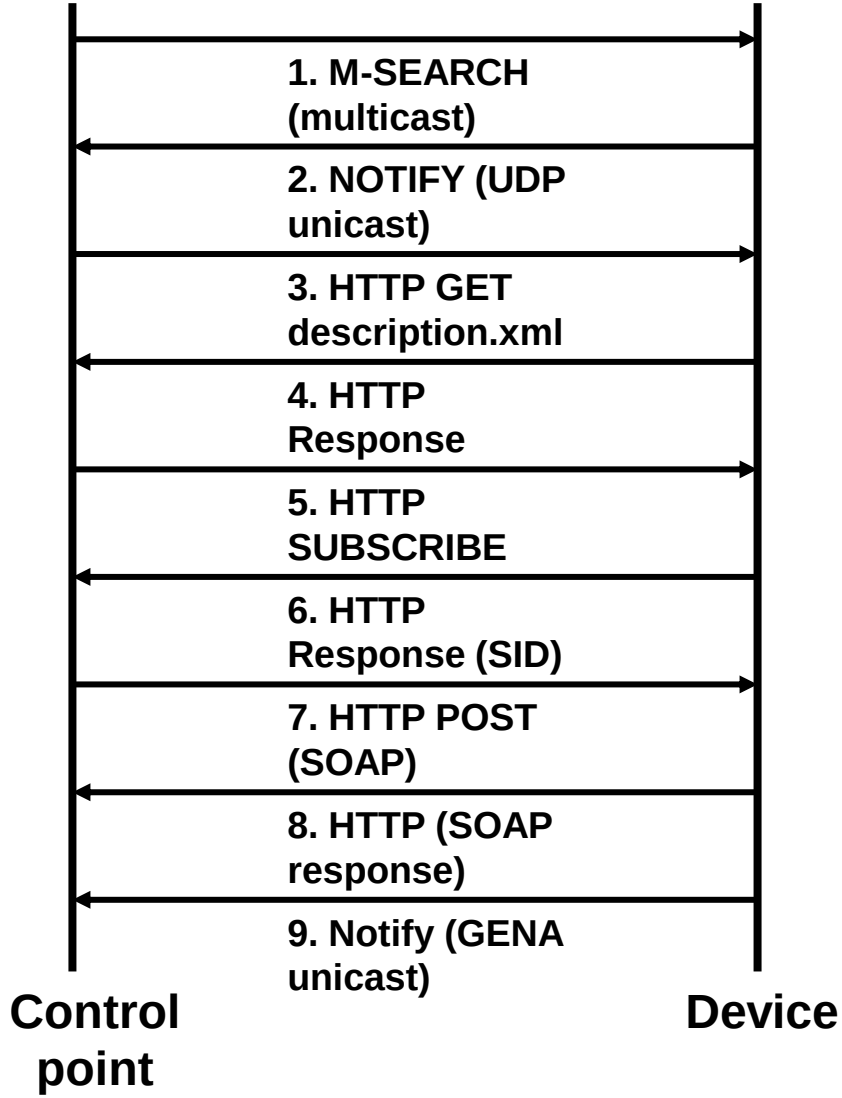
Details on Service-Control Point Interaction

6. Device replies with registration ACK and returns unique Subscription Identifier (SID)
7. Control point can command the execution of operations at device, with possible modification of state variables
   - URL to send control requests included in the XML document with device description
   - Control point sends SOAP request over HTTP



Details on Service-Control Point Interaction

8. Device possibly changes state variables and returns response as a SOAP message
9. Device can notify clients of the occurred state change, either stemming from invoked actions (as in the case of 8) or generated by implicit modifications at device
   - Device performs notifications via unicast NOTIFY messages over HTTP



By Summarizing… UPnP Middleware Features

- Service Discovery
  - Support designed for peer-to-peer environments without hierarchical structuring
- Adaptability
  - IP addresses are dynamically allocated
  - State modifications are made available via event notification
  - No support (yet) for service routing/selection based on client location
- Transparent support to communication
  - Exploitation of Internet standards
  - No support to multi-hop ad-hoc communications
- Data Transformation
  - Possible data transformation from standard XML format to proprietary formats that may be control-point-specific (e.g., proprietary Microsoft ones)

Additional Useful Links about UPnP

- S. Helal, “Standards for Service Discovery and Delivery,” *IEEE Pervasive Computing*, Vol. 1, No. 3, pp. 95-100, July-Sept. 2002
- Jini Forum, at http://www.jini.org/
- Service Location Protocol Working Group (svrloc), at http://www.ietf.org/html.charters/svrloc-charter.html
- UPnP Forum, at http://www.upnp.org/
- UPnP Forum, “Universal Plug and Play Device Architecture,” at http://www.upnp.org/resources/documents.asp
- Intel, “UPnP Technology,” at http://www.intel.com/technology/UPnP/

Exercise on UPnP (1)

To design and implement a small application that uses ***UPnP to discover the availability of file multimedia files*** offered in a locality

To try to respect, as much as possible, the design architectural choice of ***out-of-band multimedia communication (direct between end points) wrt discovery***

Situation close to realistic scenario where UPnP is used as solution for configuration, discovery, and service access in home-oriented networks, ***in particular for media servers, rendering devices, data sources, control points, …*** (see the approach supported by Digital Living Network Alliance – DLNA - http://www.dlna.org/)

Exercise on UPnP (2)

Please refer to docs and dev tools, largely available in the community, such as:

- Microsoft, “Using the UPnP Control Point API”, http://msdn.microsoft.com/en-us/library/ms898948.aspx
- https://macchina.io/docs/00400-UPnPControlPointImplementationGuide.html

UPnP for Android:

- https://play.google.com/store/apps/details?id=com.bubblesoft.android.bubbleupnp&hl=it&gl=US

As usual, the exercise could be a starting seed for a possible further project activity…

Alternatively, Solution based on Apache River

To design and implement a small application that uses ***Apache River to discover the availability of multimedia files*** offered in a locality

To try to respect, as much as possible, the design architectural choice of **out-of-band multimedia communication** (direct between end points) wrt discovery

In this case, please refer to docs and development tools available at:

- River home page - https://attic.apache.org/projects/river.html

Decoupled Communications: Messaging

As already stated, ***relevance of decoupling*** in communication and interaction among mobile distributed components

Sometimes ***message exchange*** is even used as the general term to indicate the primary type of ***mobile communication middleware*** (see *S. Tarkoma, "Mobile Middleware"*) to highlight the importance of decoupling

Any mobile messaging solution must define:

- Principles and architecture
- Message ***syntax***
- ***Protocol*** for message exchange
- ***Locator***

Sometimes protocol term is used to include also syntax and locator…

Messaging: Principles and Architecture

Primary principle: ***loose coupling*** (via ***standard and open protocols/formats***)

In real systems, also ***extensibility***. How to?

- ***Version identifiers included in messages*** (non-recognized versions are considered as errors; back-compatibility?)
- ***Formats with extension points***
- ***Forward compatibility with possibility to ignore*** message parts that are not recognized (example of application of *robustness principle*)

Usually ***middleware APIs*** to allow applications to use communication facilities; sometimes ***middleware with visibility*** of requirements for data exchange and their semantics

Messaging: Message Syntax

Marshalling/unmarshalling management:

- ***Implemented at the application level***
- ***Code may be automatically generated*** (typically based on approaches like Interface Description Language – IDL – which is considered at development time)
- ***Introspection*** (higher expressive power for developers but typically more expensive at runtime)

How to agree on data format?

- ***Specification*** (usual approach of Internet protocols with messages in binary format)
- ***Negotiation***
- ***Receiver-makes-right*** (sender uses its native formats and specifies metadata to indicate which formats are used)

Primary types of message formats:

- Binary (ASN.1, …) or text-based (XML, JSON, …)

Messaging: Protocols

In addition to the usual protocol properties for communication in distributed systems (headers with metadata and payload, also application-layer metadata, message types and with which exchange patterns, …), special accent on ***connection management***

- Protocols that "mimic" transport layer, with application-level connection in 1:1 relationship with transport-level connection
- ***More often protocols that decouple the two aspects*** (***persistent session feature over multiple and temporary transport connections***; see TCP and change of dynamic IP address, or SIP…)

«Pure» end-to-end perspective or usage of mediators up to the application level?

Wide usage of ***store-and-forward architectures and protocols*** (decoupling in time, optimization of implementations for reliability, violation of end-to-end principle)

Messaging: Protocols

Classical schemes for message exchange: one-way, two-way, request-response, subscribe-notify, conversation, …

Relevant:

- Role at transport layer (***initiator-listener***), not necessarily the same as for the application/messaging levels
- Usual distinction ***blocking vs. non-blocking***
  - ***Polling method***, usually with object (***promise or future***) given to the sender; possible to make either inspect or blocking claim
  - ***Callback***

Which of the two schemes facilitates more the development of mobile applications and/or for mobile systems?

Messaging: Locator

We are used to ***locators strongly tightened to network addresses***

But also locators more articulated and complex, e.g., which include port numbers (transport) or paths (URLs)

In mobile systems ***many types of locators***, also non IP-based, in particular in the past when IP was not so dominant

Anyway, even nowadays, possibility of:

- ***"Transparent" locators***, typically implemented as URIs and codified in XML (it increases the level of abstraction + decoupling)
- ***"Opaque" locators***, as in CORBA. Need of middleware to generate and use opaque locators

Is mobility management a network-layer issue? Of course, given what we have already widely discussed in the course, NOT ONLY…

Often it is written that *mobile hosts are managed as second-class citizens*; ***towards locators independent from network layer***…

Messaging: Design for Mobility

Usual general considerations on:

- ***Valuable role of proxies,*** e.g., to split transport connections in two parts (breaking the end-to-end principle)
- ***Problems of Network Address Translation*** (NAT) when mobile nodes are willing to offer services (see also FTP and IRC…)

In addition:

- ***Ability to complete the scheme of message exchange*** even if communicating entities move
- Exploitation of classic transport-level connections is usually preferred
- ***Simple syntax and reduced message content***, also considering compression facilities
- ***Security at message level***: with SSL-like approaches (connection-level security), which kind of issues if end-to-end principle is broken? ***Message-level security with security applied to (parts of) the payload, not to headers***; of course, also combination of connection and message

Messaging: Reliability

As usual:

- Distinction between ***end-to-end and hop-by-hop***
- Basic technique with ***acknowledgement and re-transmissions*** (also at the application level)

ACK types:

- Regular
- Cumulative
- Negative
- Piggy-backed

In-order delivery? Sometimes it can be sacrificed for efficiency motivations

Indeed, reliability reduction due to performance motivations is a well-known concept (DNS, NTP, SIP, … typically use UDP)

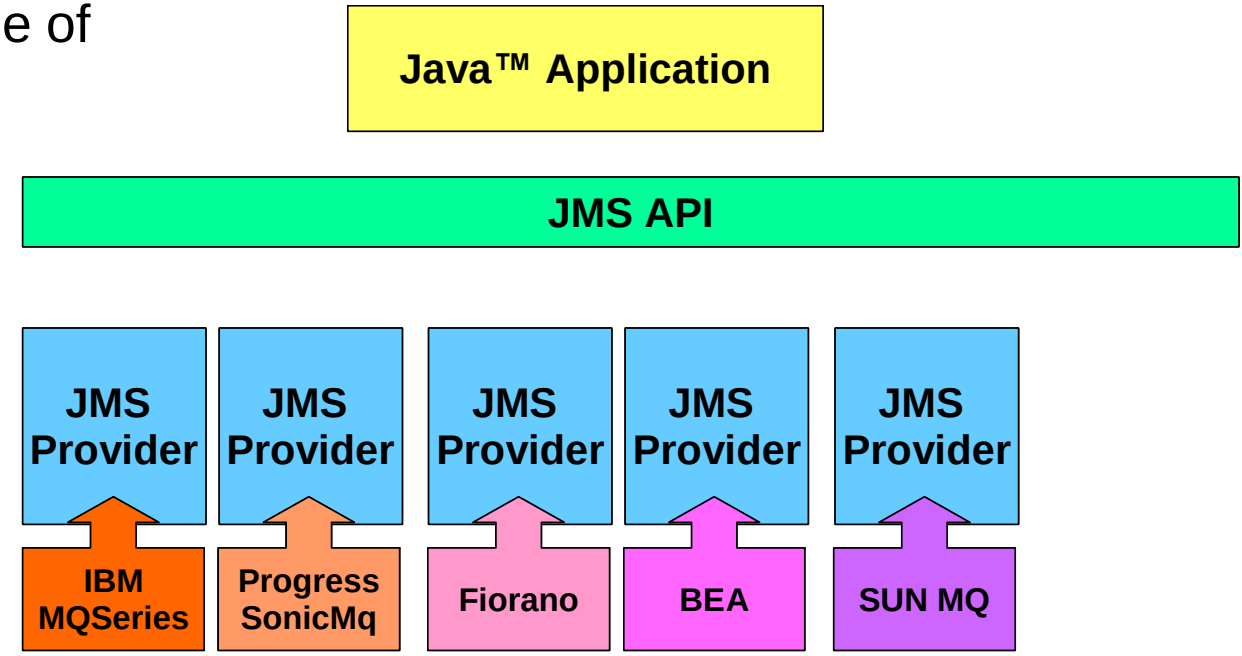
Messaging Examples: Java Message Service (JMS)

- Possibility to ask for ***only-once semantics for message delivery*** (more precisely once-and-only-once for persistent usage, at-most-once for non-persistent usage)
- ***Decoupling in time thanks to durable destinations***
- Partial time coupling for topics: it can be reduced via ***durable subscriptions***
- Possibility of ***non-blocking reception via listeners***
- Usage of ***ConnectionFactory***
- Messages sent within a session (via a given Session object) towards the same destination ***benefit from in-order delivery property***
- ***Three types of ACK*** (lazy, automatic, and client-side)

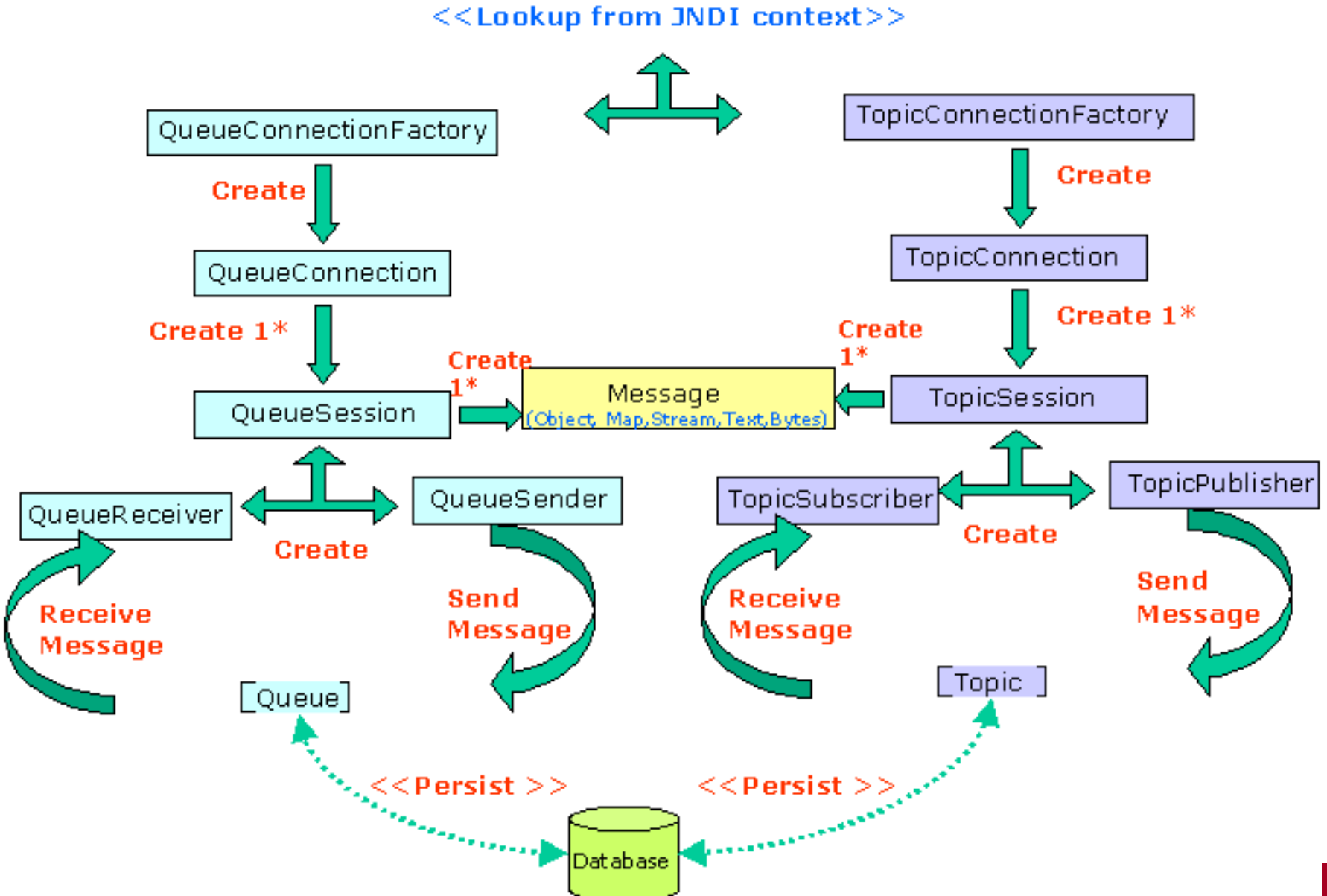
Design Goals in JMS

JMS is part of the J2EE platform. Goals:

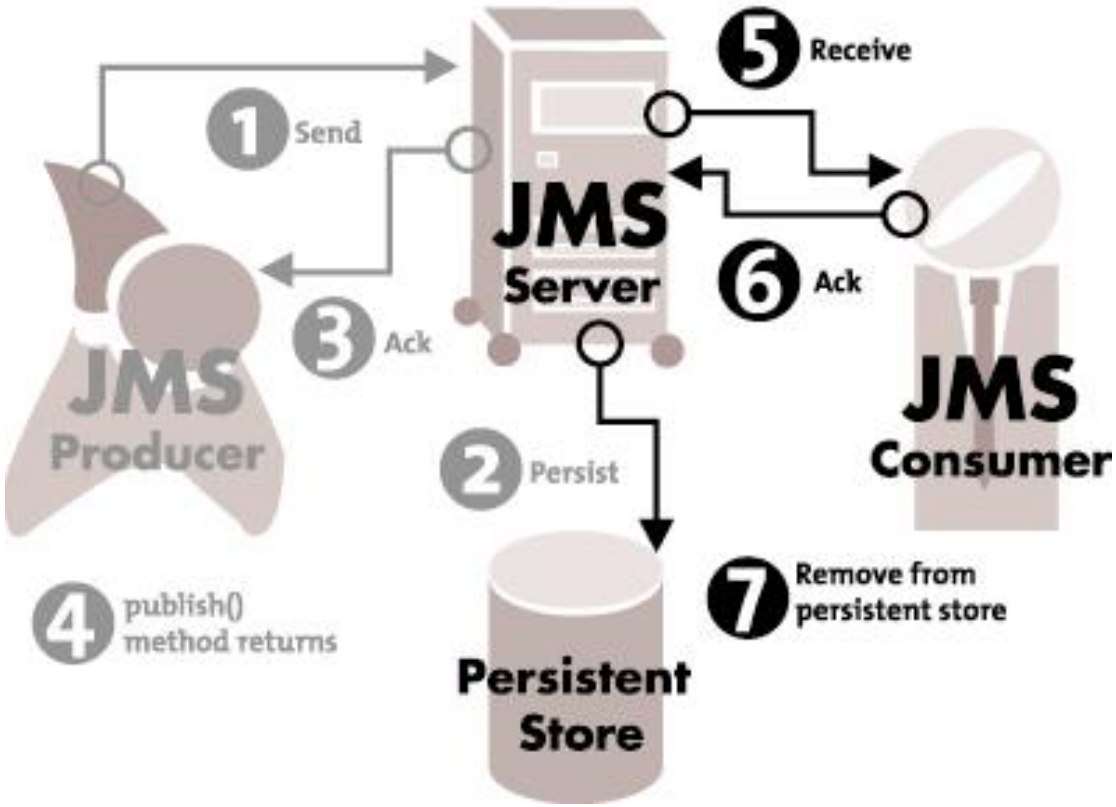
- ***Compliance/similarity*** with APIs of ***existing messaging systems***
- ***Independency*** from vendors of messaging systems
- Coverage of most common facilities that are offered in messaging systems
- It promotes the usage of Java technology



"Graphical Summary" of JMS APIs



Reliability through ACKs: e.g., Automatic ACKs



- ***Producer-side and consumer-side*** perspectives
- Differences between ***persistent and non-persistent cases***
- When is it possible to have ***duplicated messages***?
- When is it possible to have ***message losses***?
- In addition, ***three differentiated types of ack***

Persistency: Two Delivery Modes

- ***PERSISTENT***
  - Default
  - It specifies to ***JMS provider to guarantee*** that the message ***is not lost when in transit***, e.g., because of a failure at the JMS provider
- ***NON_PERSISTENT***
  - ***It does NOT request storing messages*** at the JMS provider side
  - Better performance results

SetDeliveryMode() method in the MessageProducer interface

- `producer.setDeliveryMode(DeliveryMode.NON_PERSISTENT);`
- ***Extended form***: `producer.send(message, DeliveryMode.NON_PERSISTENT, 3,10000);`

Priority and Expiration in Message Delivery

- 10 priority levels
  - from 0 (lowest) to 9 (highest)
  - default = 4

Usage of setPriority() method of MessageProducer interface, e.g., `producer.setPriority(7);`

or the extended form `producer.send(message, DeliveryMode. NON_PERSISTENT, 7, 10000);`

- Expiration: possibility to ***configure TTL*** via setTimeToLive() of the MessageProducer interface
  - `producer.setTimeToLive(60000);`
  - Or extended form, `producer.send(message, DeliveryMode.NON_PERSISTENT, 3, 60000);`

Messaging Examples: CORBA Messaging

CORBA Messaging specification includes:

- Asynchronous Messaging Interface (AMI)
  - Possibility of both ***polling and callback*** (callback is passed as CORBA object, therefore even not in the same addressing space of client)
- Time Independent Invocation (TII) to specify which CORBA objects play the ***role of router*** for the message
  - Rationale: sender and recipient may be temporarily disconnected
  - They compose a ***store-and-forward network***

CORBA locator = Interoperable Object Reference (IOR), with different profiles depending on binding protocol

Messages in binary format = Common Data Representation (CDR)

Extreme flexibility in the choice of the protocol

CORBA AMI: Callback Mode

Callback: client provides callback method to be invoked by the support after service completion via a given ***fire-and-forget (automatically invoked)***

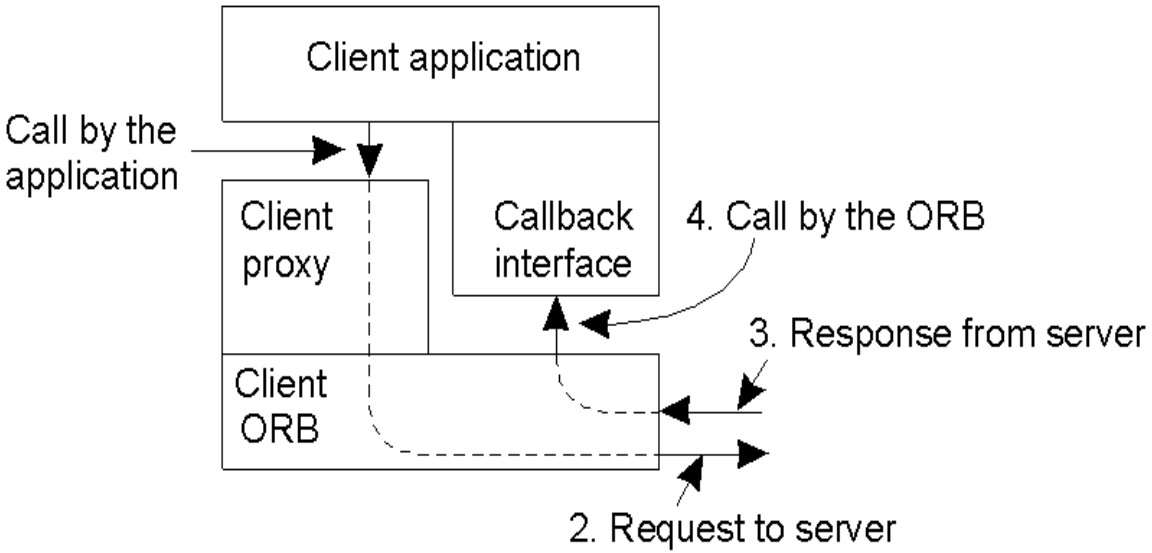
In place of: `int somma (in int i, in int j, out int somma)`

```
void  sendcallback_somma (in int i, in int j)
void  callback_somma (in int success, in int somma)
```

Usage of two methods by changing ***only client implementation*** *and* ***NOT any service part***

Client invokes **sendcallb…**

ORB invokes **callback_som..**



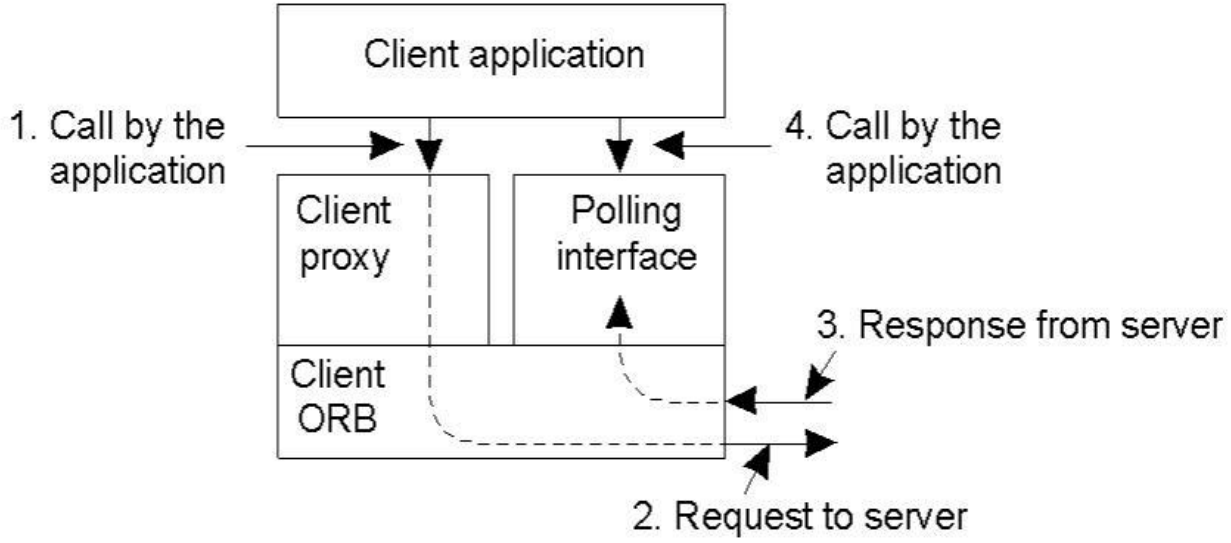
CORBA AMI: Polling Mode

Asynchronous polling: client decides ***when and whether*** to interrogate a method to check completion of remote operation (by collecting results); this method is ***created by the messaging support***

In place of: `int somma (in int i, in int j, out int somma)`

```
void  sendpoll_somma (in int i, in int j)
void  pollsomma (out int success, out int somma)
```

Result is collected on request by invoking ***pollsomma operation that is autom. generated by CORBA support***



Messaging Examples: Extensible Messaging and Presence Protocol (XMPP)

Essentially designed for instant messaging

RFC 3920 is oriented and similar to the existing implementation of the Jabber protocol; good popularity and widespread utilization thanks to the adoption by *Google, Twitter, Facebook*, …

It includes ***publish/subscribe mechanisms*** (see the following slides…) ***to update presence and state, and for service discovery***

Client-server model: client sends an XMPP dataflow to a server, after parameter negotiation

Peer-to-peer model: servers coordinate together for delivery to recipients

Usage of so-called ***stanzas***, of three types:

- *Message stanza* – one-to-one communication, similar to emails
- *Presence stanza* – simple pub/sub mechanism, communication is transferred to all subscribers
- *Info/Query stanza* –request-response mechanism

Messaging Examples: Extensible Messaging and Presence Protocol (XMPP)

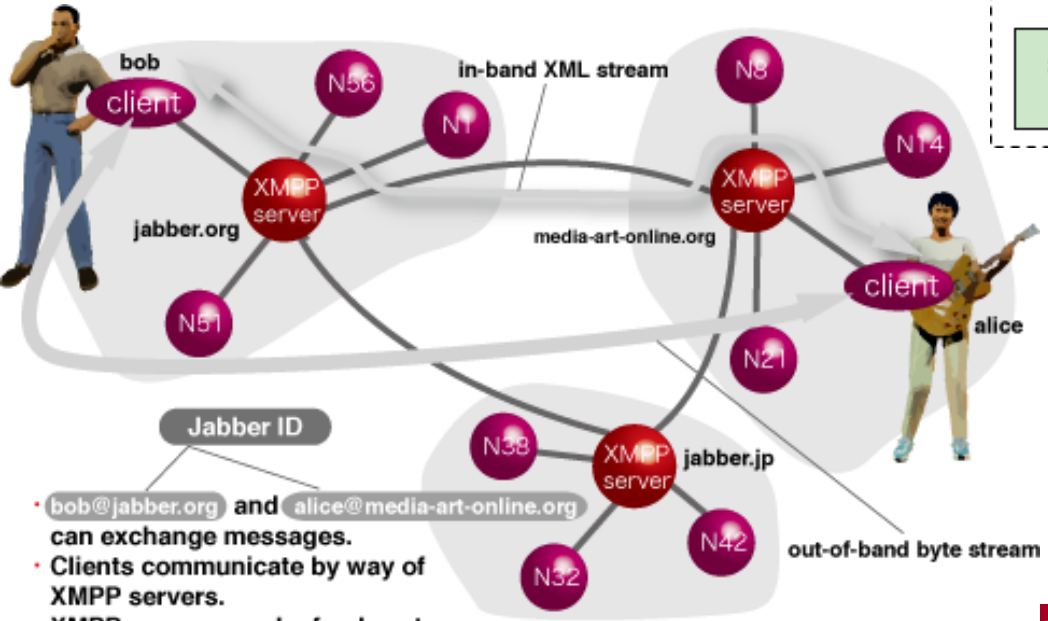
XMPP messages are streams codified in XML

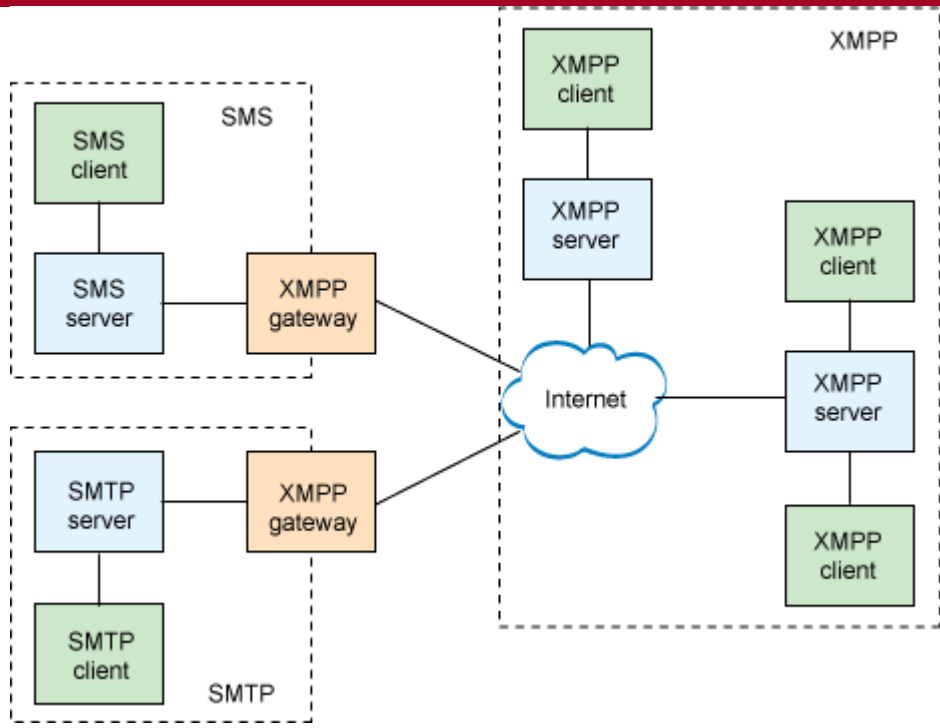
Given the widespread adoption, good candidate to support messaging in mobile systems, EVEN IF:

- Not specifically designed for mobile systems
- Expensive XML processing, expensive connection management in particular in terms of energy
- ***Expensive re-connections to XMPP server*** (need to re-establish a **new session of interaction per any new transport connection**, transmission of XML data that is non-negligible at each session start)

Android implements a specialized and proprietary variant of it, with non-XML-based protocol and NO creation of a new session per any new connection

Messaging Examples: Extensible Messaging and Presence Protocol (XMPP)





Messaging Examples: Web Services

SOAP is built on top of interaction model based on message exchange

- Architecture based on senders, receivers, and intermediate nodes
- Locator = HTTP URI
- ***Document-style SOAP***: ***messages as XML-based documents*** that have to be processed
- Possibility of ***different protocol bindings***, but definitely the most used one is HTTP, utilization of POST method (employed more as transport protocol, while ignoring its application semantics)
- In mobile environments, where HTTP is sometimes the only protocol practically usable because of firewalls and NAT, this use/misuse of HTTP could be considered as legitimate and become largely adopted…
- ***Also specification for binding to email and XMPP***

Messaging Examples: Representational State Transfer (REST)

REST is substantially a ***solution architectural style***, *Resource Oriented Architecture* (Roy Fielding, UCI PhD Thesis, 2000)

To promote ***client-server and stateless interaction, oriented to the usage of caching opportunities***, also with possibility of code-on-demand to clients

Any resource has a persistent identifier; idea to transfer NOT resources but their representations via HTTP protocol

Constraint: exchange of self-descriptive messages (languages for representation, negotiation of supported modes, …)

Messaging Examples: Representational State Transfer (REST)

Locator = HTTP URI

Three types of metadata included in HTTP headers:

- ***Resource metadata*** – about resources, e.g., timestamp about last modification
- ***Representation metadata*** – about transferred representation, e.g., its media type
- ***Control metadata*** – about message, e.g., its length and caching possibility

Notable example: ***RESTful Web services***

RESTful Web service as a simple Web service implemented by using HTTP and REST principles, thus resource collection with 3 well-defined aspects:

- ***URI base for service***, e.g., http://example.com/resources/
- ***Internet media type*** for data used in the service (usually JSON or XML)
- Set of service operations supported ***via HTTP*** method invocations (e.g., via POST, GET, PUT or DELETE)

Messaging Examples: Representational State Transfer (REST)

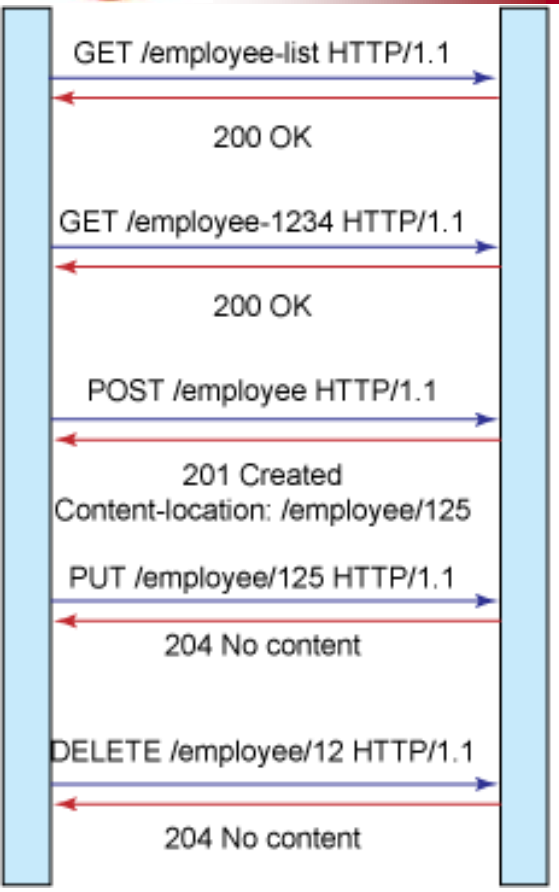
Notable example: ***RESTful Web services***

Resource	GET	PUT	POST	DELETE
URI for resource collection, e.g., http://example.com/resources/	To list all collection members	To replace the whole collection	To create a new element to be inserted in the collection	To remove the whole collection
URI for single element, e.g., http://example.com/resources/ef7d-xj36p	To obtain the representation of the targeted element, espressed in the appropriate Internet media type	To replace or create an element of the collection	To consider the element as a collection and to create a new element internally to it	To remove an element from the collection

Examples of today's REST usage:

- Majority of ***Web blogs*** (download of XML files in *RSS/Atom format*, which contain links to other resources)
- ***Simple Storage Service (S3)*** by Amazon.com
- ***OpenStreetMap*** (REST interface)… and many many others

Messaging Examples: Representational State Transfer (REST)



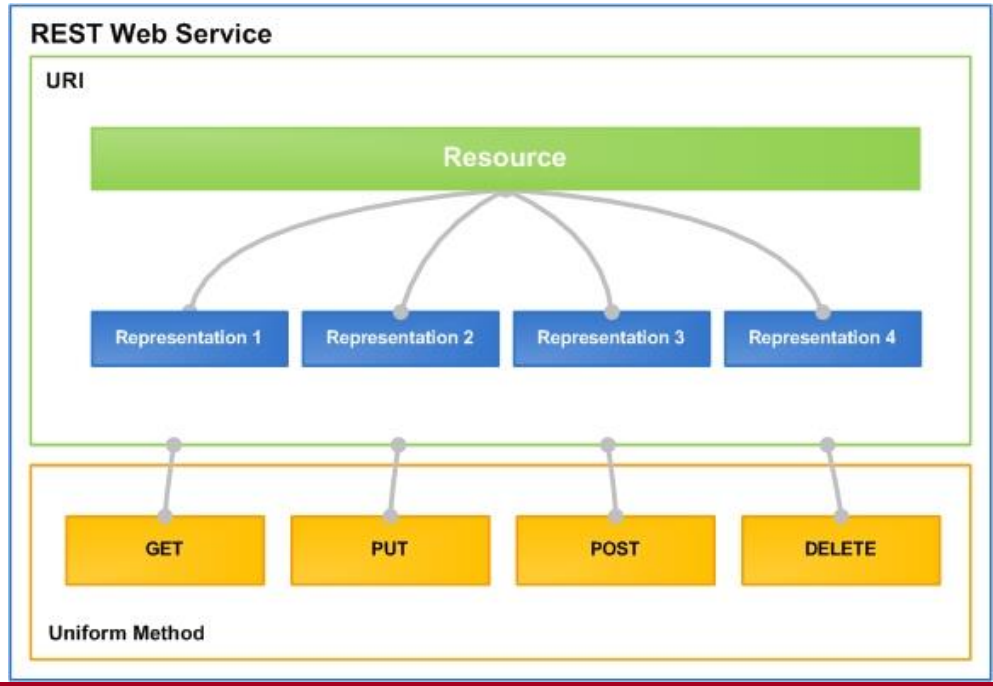
Retrieve a list of all employees

Retrieve a particular employee

Create a new employee and retrieve new ticket URI from HTTP headers

Update an employee

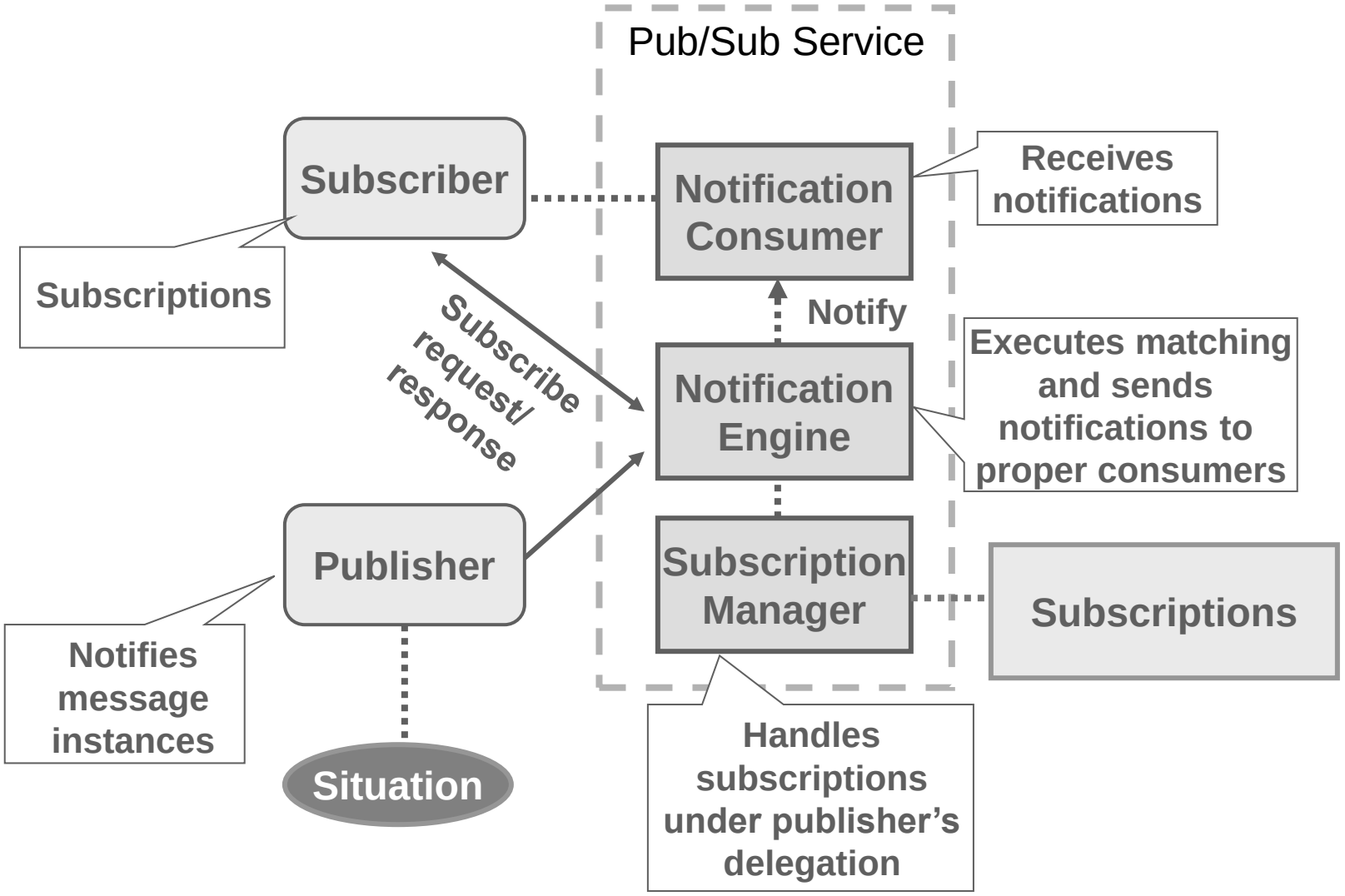
Delete an employee



Event Management and Publish/Subscribe Systems

- ***Event delivery from publishers to subscribers***
  - Events as messages with content
  - ***One-to-many, many-to-many*** (traditional message systems are queue-based and one-to-one)
  - Often implemented based on ***messaging systems and on store-and-forward solutions***
- Comm. paradigm of frequent usage, in particular in mobile systems
  - ***Decoupling in space and time***
- Event system as ***logically centralized system***
  - Anonymous communication
  - Possibility to ***use filters*** (on headers or entire messages)
  - Basic primitives: subscribe, unsubscribe, publish, also with filters
- ***Different topologies for routing and different semantics*** associated to event sending/notification
- Associated operations are typically ***non-blocking*** (polling, callback)

General Architecture for Publish/Subscribe Systems



Event Router and Topologies

Event router or broker

- Works as mediator (decoupling) between publishers and subscribers
- Usage of ***routing table (also with filters)*** for local event dispatching or to indicate to which «near» router to forward in the case of distributed brokers (to achieve scalability, reliability, and high availability)
- ***Filters*** may be also based on ***content => content-based routing***
- Other non-functional requirements: notification within time deadlines (***bounded delivery time***), QoS, fault-tolerance, ordering (***causal order, total order***)

Possible router topologies:

- ***Centralized***
- ***Hierarchical*** (notifications always sent to master, i.e., root of the distribution tree)
- ***Cyclic, acyclic*** (peer-to-peer, cyclic allows redundancy but need of *minimum spanning tree techniques* to prevent from cycles)
- Based on ***rendez-vous point*** (special router that works as rendez-vous, typically for pre-determined types of events)

  Partially related: have you ever heard of ***Distributed Hash Tables (DHTs)***?

Interest Propagation and Subscriptions

One of the primary functions of a router is to ***propagate notifications to near routers that are interested*** in that event. To this purpose, how to propagate interests and subscriptions?

Properties to be achieved: reduced forwarding overhead, high performance, fast support to variations

- ***Simple routing***: any router knows all subscriptions in the global systems (subscription flooding), possibly with optimization of NO forwarding if subscription message has been already circulated
- ***Covering-based routing***: forwarding of only the more general subscription filters (*which possible issues with unsubscription?*)
- ***Merging-based routing***: it allows to merge different entries in routing table for the sake of table size optimization (usually combined with covering, here also unsubscription issues)

Notifications are usually distributed over ***reverse paths (wrt subscription paths)***

Decision about Message Routing

Depending on what is used to take message routing decisions, classification into:

- ***Channel/topic-based***: depending on the channel (usually named channel) on which the event is published. Pub/sub agreement on the channel name, also possibility of associated multicast address
- ***Subject-based***: depending on event subject, single field of info
- ***Header-based***: depending on a set of fields. For example, SOAP supports header-based routing for its messages
- ***Content-based***: possibly depending on the whole message content. Higher expressive power, higher costs

Also ***context+content-based routing***, particularly suitable for mobile systems/services with ***event filters that are context-dependent***

Java Model for Distributed Events

Also Java has a built-in ***model for event distribution, based on RMI***, e.g., used in Jini/River

- Based on ***Remote Event Listener*** (consumers are registered to receive given types of events from given objects, **`notify()`** method)
- ***Remote Event object returned back during notification*** (data, reference to source object, handback object, unique identifier)
- ***Lease mechanism***
- The specification includes possibility to define ***Distributed Event Adaptors that implement filters and QoS policies***
  - Idea to exploit handback object, returned by the event source, to transfer state and behavior (e.g., to implement event filters)

Java Model for Distributed Events

```
package net.jini.core.event;

public class RemoteEvent {
    public long getID();
    public long getSequenceNumber();
    public java.rmi.MarshalledObject getRegistrationObject();
}
```

Events generated in local components may transfer even quite complex object state. ***NOT distributed events: only info on how state retrieval is possible at runtime***

- Remote event as serializable object that can be transferred between listeners
- Idea, "stolen" from Xt Intrinsics and Motif solutions: ***to register clients by including handback objects, returned back with any event***

For example, a Jini taxi driver subscribes to taxi bookings while passing through a city area (handback includes location); when it receives an event, it can be informed of old location (at the moment of registration)

Possibility to register other objects for notification delegation: in this case, handback can work as "reminder" with info of subscribers *(stock broker model)*

Java Model for Distributed Events

Event registration

Jini/River does NOT specify how to register listeners at event sources; only specification to use a class as return value from subscription:

```
package net.jini.core.event;
import net.jini.core.lease.Lease;

public class EventRegistration implements java.io.Serializable {
    public EventRegistration(long eventID, Object source,
                             Lease lease, long seqNum);
    public long getID();
    public Object getSource();
    public Lease getLease();
    public long getSequenceNumber();
}
```

Therefore, the developer of event source has to implement:

```
public EventRegistration
   addRemoteEventListener(RemoteEventListener listener);
```

Java Model for Distributed Events

Java model for local events work with objects that are all in the same addressing space

Jini as community of distributed objects that cooperate through ***proxies***

For remote events, **"*inversion of proxy direction*"**

- For example, ***Jini client uses its proxy for service access and through it registers itself as listener***
  - Need for a proxy method to ***add event listeners***
- Proxy will invoke the "real" method for listener adding over the discovered resource
- Invocation of ***registration of local event to proxy***; invocation of proxy for remote resource registration

As if real resource ***obtains a proxy for the client to use in the notification chain***

OMG Distributed Data Service (DDS)

- ***OMG specification*** (neither based on CORBA nor highly interoperable) for ***data distribution service designed for real-time systems***
- Specification defines ***APIs for so-called publish/subscribe data-centric communication***; in other terms, DDS middleware offers abstraction of global data space that is accessible to all interested applications
- Usage of ***combination of Topic objects and keys*** to univocally identify ***instances within a datastream*** of the same topic
- ***Support to content filtering and QoS negotiation***
- Suitable for distributed propagation of signals, data, and events

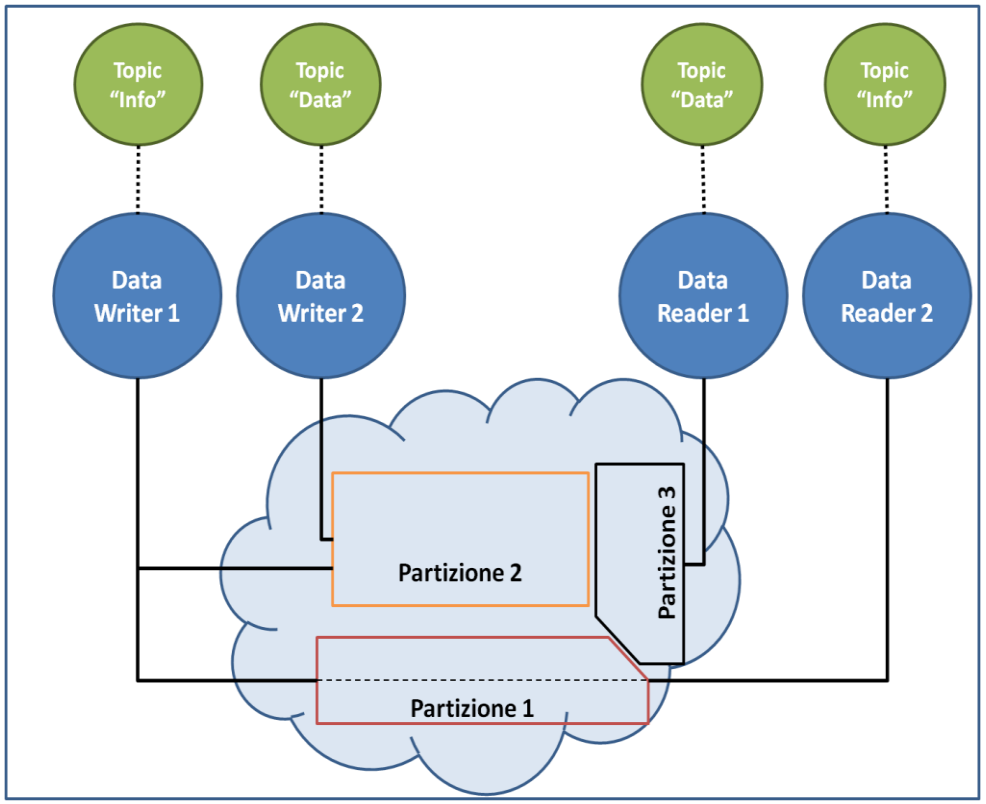
CORBA Event Service (NOT data-centric and with NO QoS support);

CORBA Notification Service (filters, QoS, but mandatory usage of CDR and IIOP)

Content Subscription: DDS Partitions

Partitions are namespaces to allow the ***logical splitting of a DDS domain***

Publisher/Subscribers can decide ***at runtime*** (and NOT at instantiation time as for JMS Topics) on which partitions to publish/subscribe data

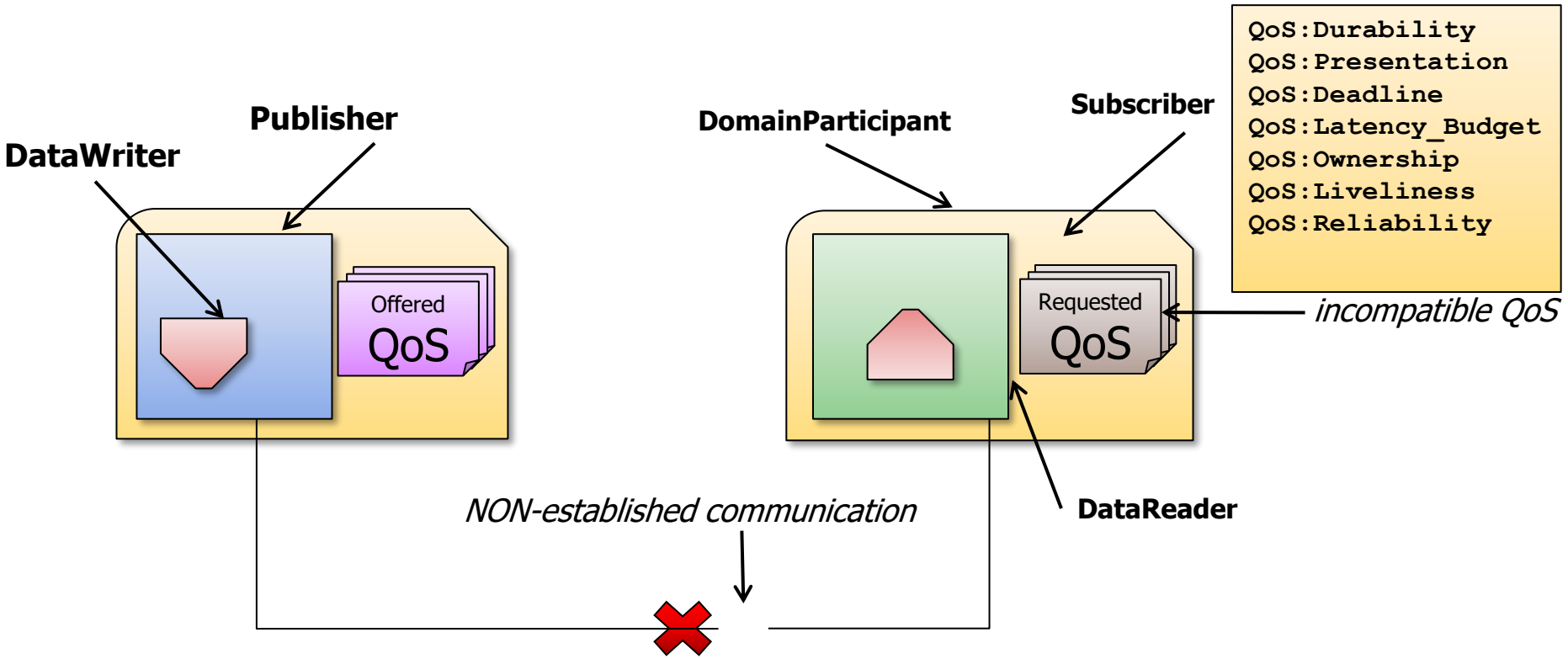


For a DataReader to receive messages from a DataWriter, there is the need to share ***both the same Topic and the same partition***

Partitions are considered to enforce a QoS policy

QoS Negotiation in DDS

- To allow a Subscriber receiving publications from a Publisher, ***QoS properties have to be compatible***
- Protocol of ***Request/Offer negotiation***



DDS supports different modes for message sending (e.g., best-effort, reliable) and personalized management of data persistence

Quality as Reliability

DDS identifies ***two QoS policies for message reliability***:

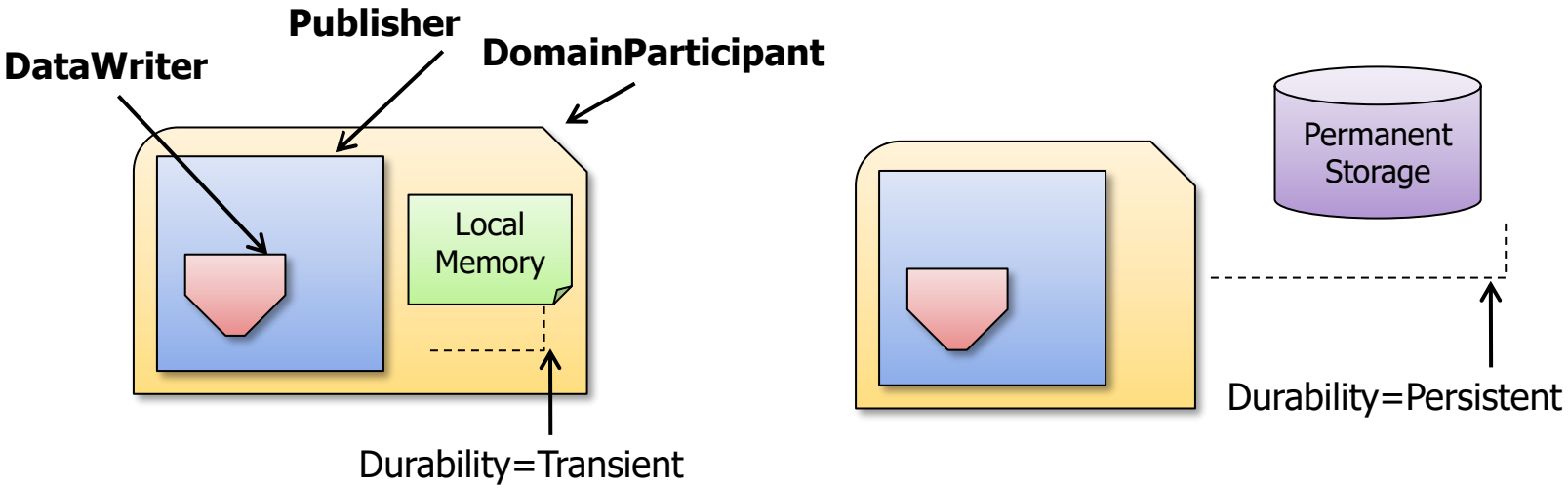
- ***BEST_EFFORT*** – ***NOT guaranteed*** that all messages are received, NOT guaranteed delivery order
- ***RELIABLE*** – guranteed that all messages are received and delivery order. Via Publishers that ***re-send*** data to Subscribers if needed and via Subscribers that send reception ***feedback (ack)***

In reliable case, all sent messages are ***kept in a history queue*** while waiting for being confirmed (publisher side) and processed by application (subscriber side); queue size can be defined, through ***HISTORY policy***

It is also possible to define ***how many resources*** (e.g., memory, max instances) to use to maintain data, through ***RESOURCE_LIMITS policy***

Quality as Durability

- Through ***Durability policy*** it is possible to define whether and how many data to be maintained at publisher side in order to enable their future successive request
- DDS supports 3 persistency types:
  - ***VOLATILE*** – No Instance History Saved
  - ***TRANSIENT*** – History Saved in Local Memory
  - ***PERSISTENT*** – History Saved in Permanent storage



DDS Quality: Additional Policies

DDS supports a wide set of other policies to define:

- ***Ordering of received messages*** (DESTINATION_ORDER - BY_RECEPTION_TIMESTAMP, BY_SOURCE_TIMESTAMP – ***eventual consistency, …***)
- ***Message priority*** (LATENCY_BUDGET)
- Exclusiveness on some given data types (OWNERSHIP)
- Data authentication and security (USER_DATA)
- ***Time constraints on message sending/delivery rates*** (TIME_BASED_FILTER)
- Fault detection and heartbeat (LIVELINESS)

More detailed technical documents at :

- Getting Started Guide
  www.rti.com/eval/rtidds44d/RTI_DDS_GettingStarted.pdf
- RTI DDS User's Manual
  www.dre.vanderbilt.edu/~mxiong/tmp/backup/RTI_DDS_UsersManual.pdf

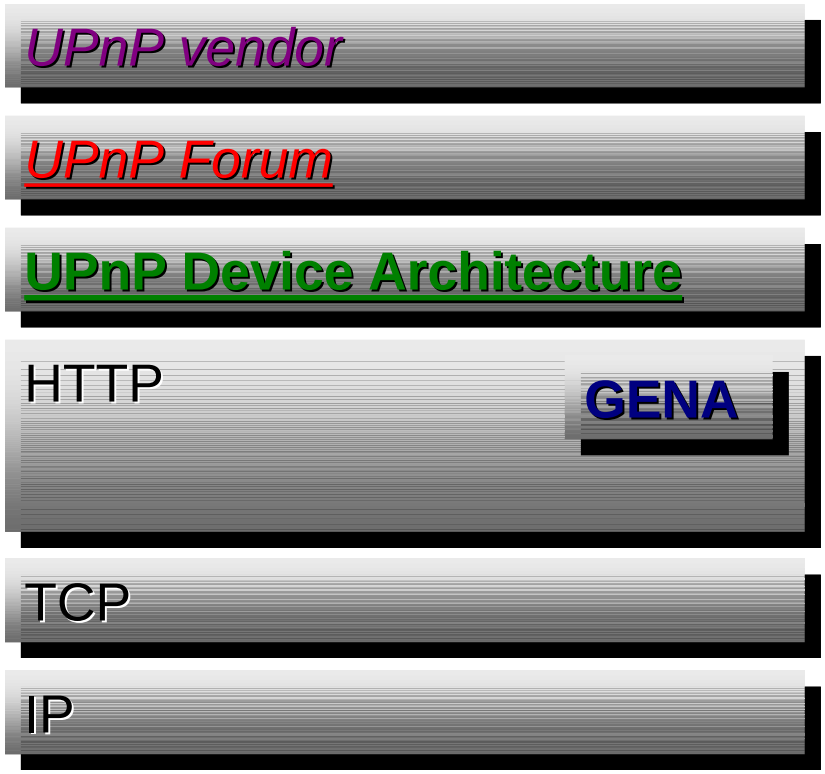
General Event Notification Architecture (GENA)

As already stated, used primarily in UPnP

- Control point is listener of modifications of device state
  - 0 obtains address
  - 1 discovers device
  - 2 determines XML descriptor
    - Obtains URL for eventing
  - 4 registers itself

Extreme simplicity:

Notification sending/reception via HTTP over TCP/IP or multicast UDP



GENA: Subscription

Control point has to register itself before being able to receive any event

SUBSCRIBE *publisher path* HTTP/1.1
HOST: *publisher host*:*publisher port*

CALLBACK: <*delivery URL*>
NT: **upnp:event**
TIMEOUT: Second-*requested subscription duration*

Device accepts subscription: it immediately sends a special event (initial) to control point with the value of all state variables

HTTP/1.1 200 OK
SID: uuid:*subscription-UUID*
TIMEOUT: Second-*actual subscription duration*

GENA: Notifications

When a state variable changes value at a device:

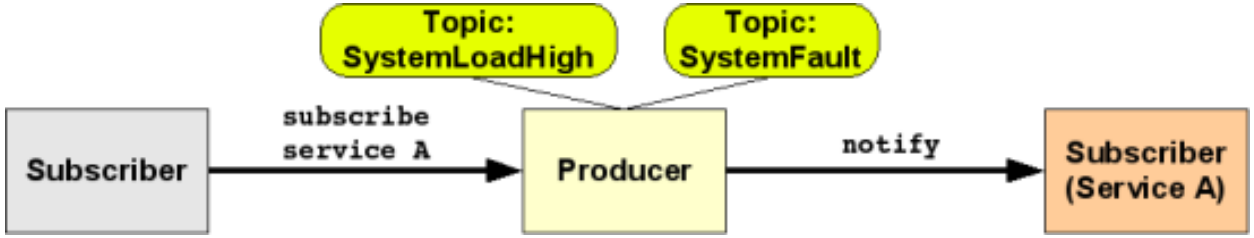
```
NOTIFY delivery path HTTP/1.1
HOST: delivery host:delivery port
CONTENT-TYPE: text/xml
 NT: upnp:event
 NTS: upnp:propchange
 SID: uuid:subscription-UUID
 SEQ: event key

 <e:propertyset xmlns:e="urn:schemas-upnp-org:event-1-0">
   <e:property>
     <variableName>new value</variableName>
   </e:property>
   Other (possible) names of variable and associated values
 </e:propertyset>
```

Web Services Event&Notification

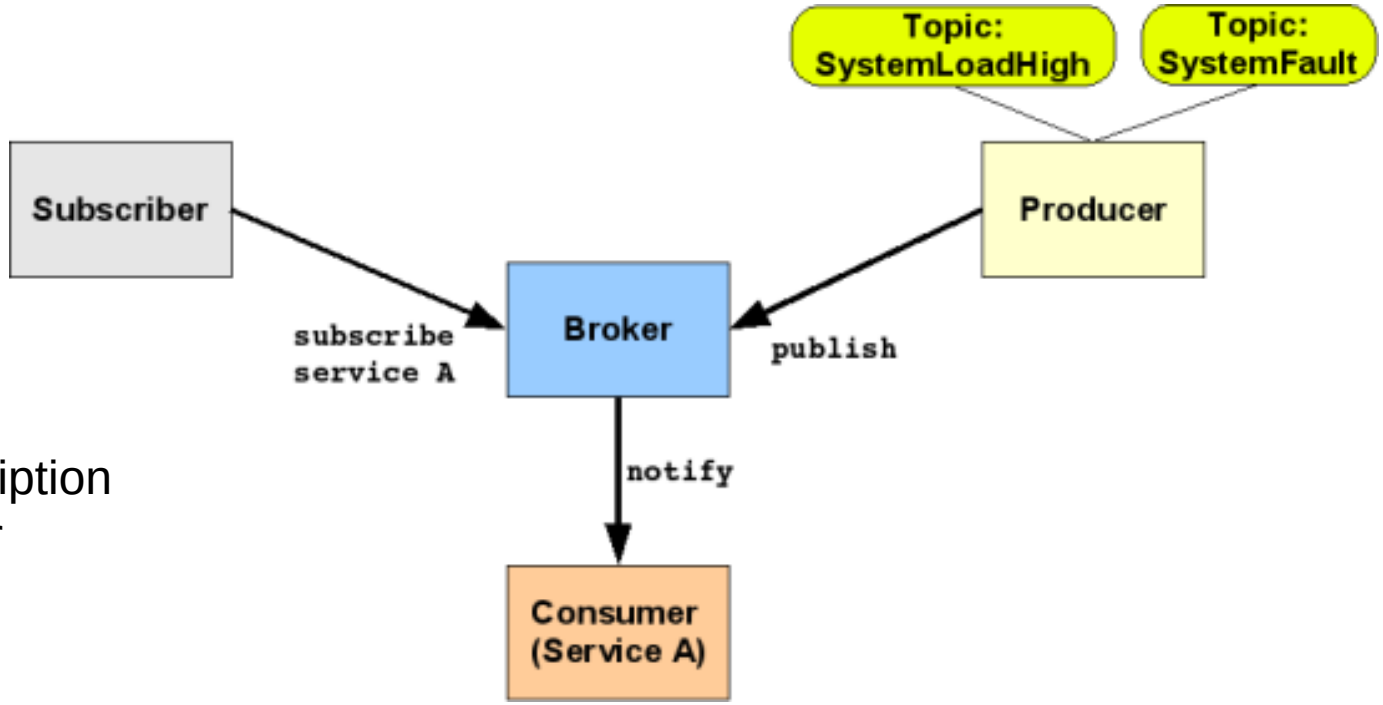
Two key mechanisms to implement pub/sub for Web services: ***WS-Eventing and WS-Notification*** (standardization in 2006)

- ***WS-Eventing*** is the specification of protocol with which Web services have to ***make/accept registrations for event notification***
  - Mechanisms to create/remove subscriptions
  - Mechanisms to define ***expiration time*** and to allow renewal
  - ***Support to filters*** (different languages for filter definition may be used)
- ***WS-Notification*** is the specification to allow Web services ***to disseminate data to other Web services***
  - Also possibility of organizations oriented to interests (called topics) and ***interest-based filtering***
  - ***Distributed topologies for notification brokers***

Web Services Event&Notification



Possible subscription from third parties (direct, with NO broker)



More usually, subscription through broker for better decoupling

Programming Example of WS-Event&Notification

For example, ***how to implement WS subscriber by using IBM WebSphere:***

- As usual, need to ***obtain WSDL file*** for notification broker and subscription manager services (resp. *NotificationBroker.wsdl* and *SubscriptionManager.wsdl*)
- If not yet available at client, need to execute wsimport to ***generate client stub***
- ***Look up at notification broker*** (need for reference to notification broker service)
- ***Instantiation of subscription request object and configuration of consumer reference***
- Instantiation of subscribe object to include subscription details, like reference to notification consumer

```
import org.oasis_open.docs.wsn.b_2.Subscribe;
import javax.xml.ws.wsaddressing.W3CEndpointReference;
import javax.xml.ws.wsaddressing.W3CEndpointReferenceBuilder;
// Crea oggetto subscription request. DEVE contenere
// ConsumerReference e PUO' includere filtro, InitialTerminationTime
// e SubscriptionPolicy
Subscribe subscribeRequest = new Subscribe();
W3CEndpointReference consumerReference = new
   W3CEndpointReferenceBuilder().address(consumerURI).build();
subscribeRequest.setConsumerReference(consumerReference);
```

Programming Example of WS-Event&Notification

Definition of ***topic expression as registration filter***

It is possible to associate a ***Filter object*** to registration request to indicate which events are relevant (***filter based on topic, message content, or both***). For example, topic-based filter (with IBM helper classes):

```
import com.ibm.websphere.sib.wsn.jaxb.base.FilterType;
import com.ibm.websphere.sib.wsn.jaxb.base.TopicExpressionType;
// To prepare the topic expression
topicExpression = topicNamespacePrefix + ":" + topicExpression;
TopicExpressionType topicExpressionType = new TopicExpressionType();
topicExpressionType.setExpression(topicExpression);
// To specify mapping from namespace prefix to topic namespace URI
topicExpressionType.addPrefixMapping(topicNamespacePrefix,
   topicNamespace);
// To specify dialect TopicExpression to use
topicExpressionType.setDialect(topicDialect);
// Filter instantiation
FilterType filter = new FilterType();
// To add expression to filter and needed configuration
// subscribe with filter
filter.addTopicExpression(topicExpressionType);
subscribeRequest.setFilter(filter);
```

Programming Example of WS-Event&Notification

Specification of registration duration and request sending

Two modes to specify expiration time for registration:

1) namespace URI and Qname objects

2) Helper factory = JAXB ObjectFactory

```
import javax.xml.bind.JAXBElement;
import javax.xml.datatype.DatatypeFactory;
import javax.xml.datatype.Duration;
// Option 1: Duration specification (one year from now)
DatatypeFactory factory = DatatypeFactory.newInstance();
Duration duration = factory.newDuration(”1Y”; JAXBElement<String>
   initialTerminationTime = new JAXBElement<String>(
     new QName("http://docs.oasis-open.org/wsn/b-2",
     "InitialTerminationTime"), String.class, duration.toString());
// Option 2:
org.oasis_open.docs.wsn.b_2.ObjectFactory objectFactory = new org.
   oasis_open.docs.wsn.b_2.ObjectFactory();
initialTerminationTime = objectFactory.createSubscribeInitial-
   TerminationTime(duration.toString());

subscribeRequest.setInitialTerminationTime(initialTerminationTime);
org.oasis_open.docs.wsn.b_2.SubscribeResponse
   subscribeResponse = port.subscribe(subscribeRequest);
```