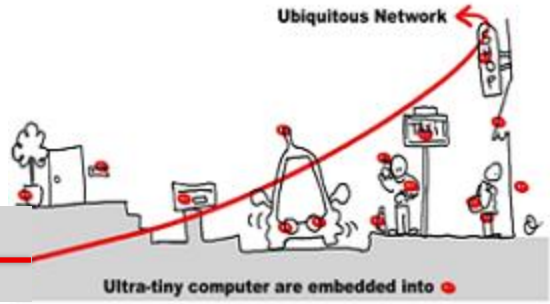
Middleware for Ubiquituous Computing

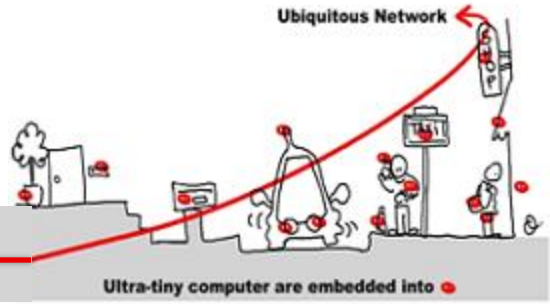


- Middleware … from distributed systems to network of things.

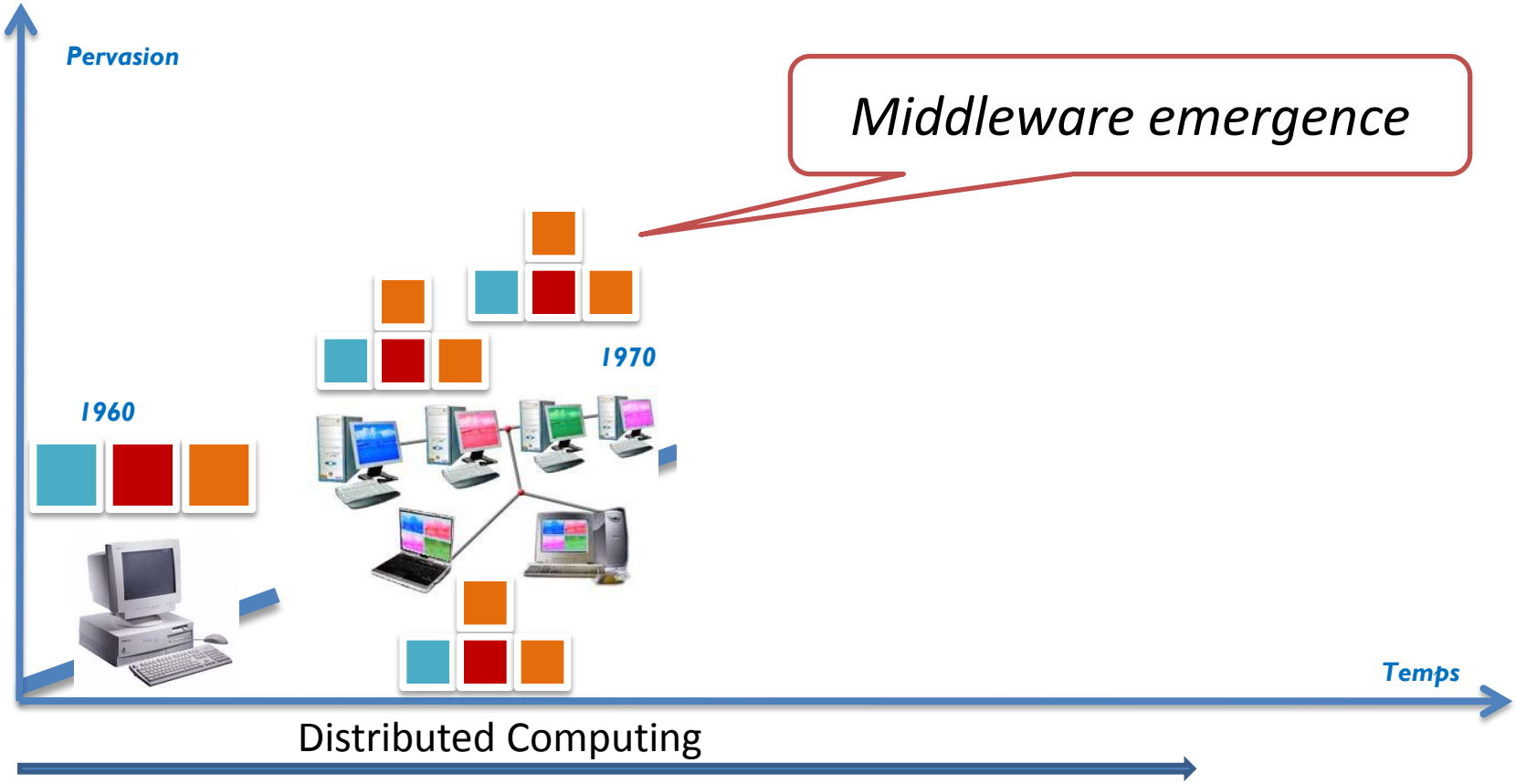
Main Instructor : Ass. Prof. Jean-Yves Tigli
http://www.tigli.fr
at Polytech of Nice - Sophia Antipolis University

Email : tigli@polytech.unice.fr

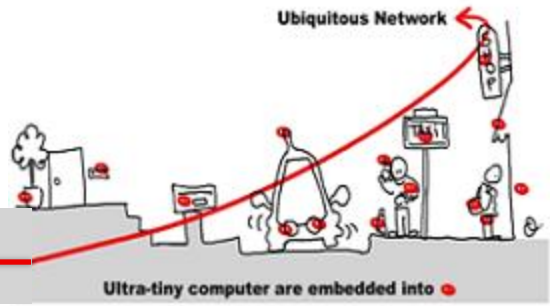
Distributing Computing



- From distributed systems …

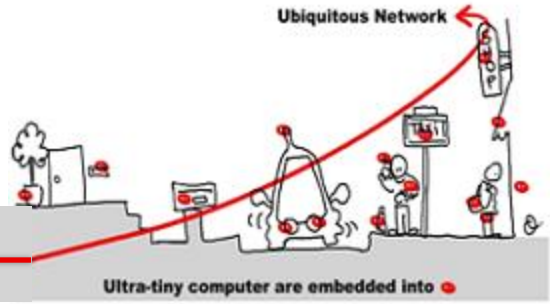


Middleware and software solutions

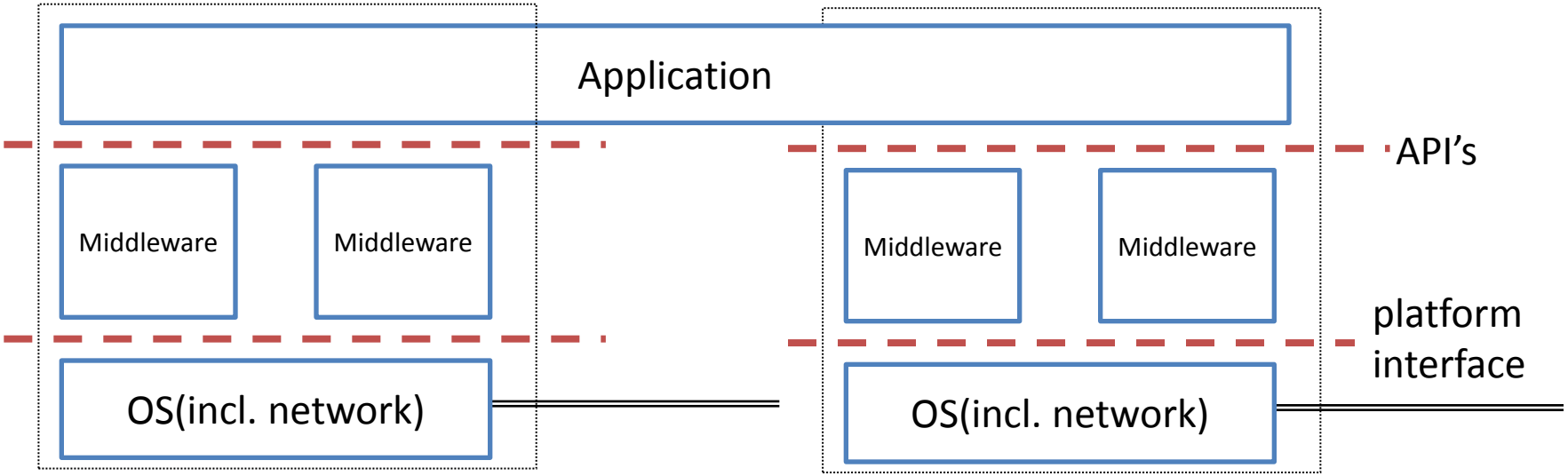


- The term middleware first appeared in the late 1980s to describe network connection management software
- It did not come into widespread use until the mid 1990s, when network technology had achieved sufficient penetration and visibility.

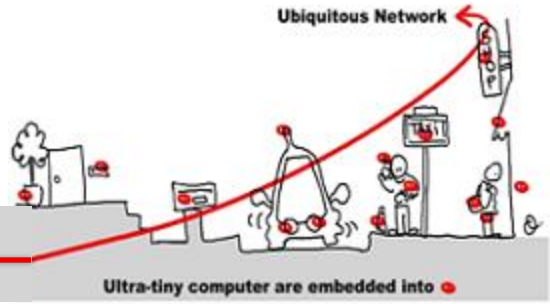
What is Middleware ?



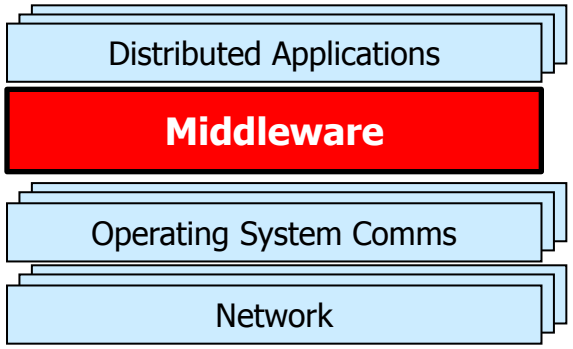
- Applications use intermediate software (middleware) that resides on top of the operating systems and communication protocols



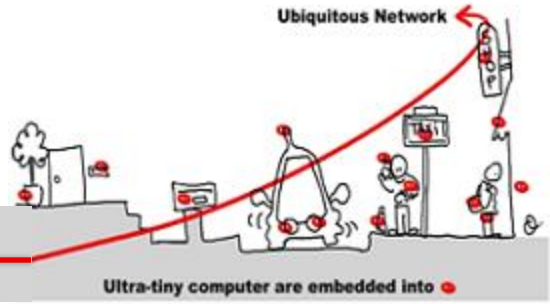
What is Middleware ?



- What is Middleware?
  - Layer between OS and distributed applications
  - Provides common programming abstraction and infrastructure for distributed applications
  - Hiding distribution, i.e. the fact that an application is usually made up of many interconnected parts running in distributed locations.
  - Hiding the heterogeneity of the various hardware components, operating systems and communication protocols that are used by the different parts of an application.
  - Supplying a set of common services to perform various general purpose functions, in order to avoid duplicating efforts and to facilitate collaboration between applications.



First definition

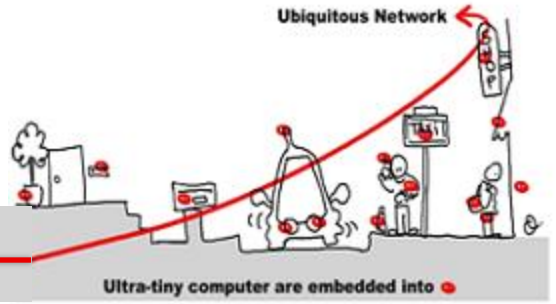


- *"The intersection of the stuff that network engineers don't want to do with the stuff that applications developers don't want to do."*

  [Kenneth J. Klingenstein ('99)]

- Mainly to deal with distribution of software applications

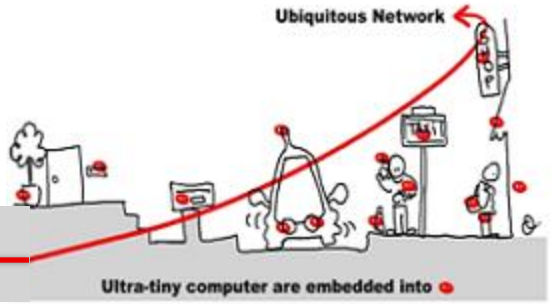
Middleware features



- Middleware provides support for (some of):
  - Naming, Location, Service discovery, Replication
  - Protocol handling, Communication faults, QoS
  - Synchronisation, Concurrency, Transactions, Storage
  - Access control, Authentication

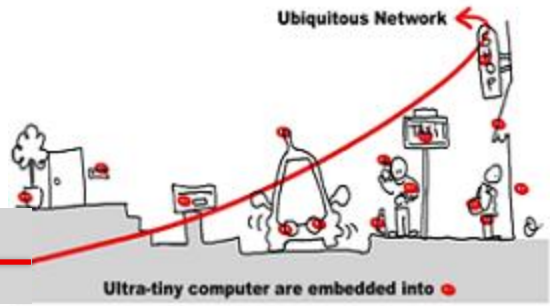
- Middleware dimensions:
  - Request/Reply vs. Asynchronous Messaging
  - Language-specific vs. Language-independent
  - Proprietary vs. Standards-based
  - Small-scale vs. Large-scale
  - Tightly-coupled vs. Loosely-coupled components

Four types of Middleware

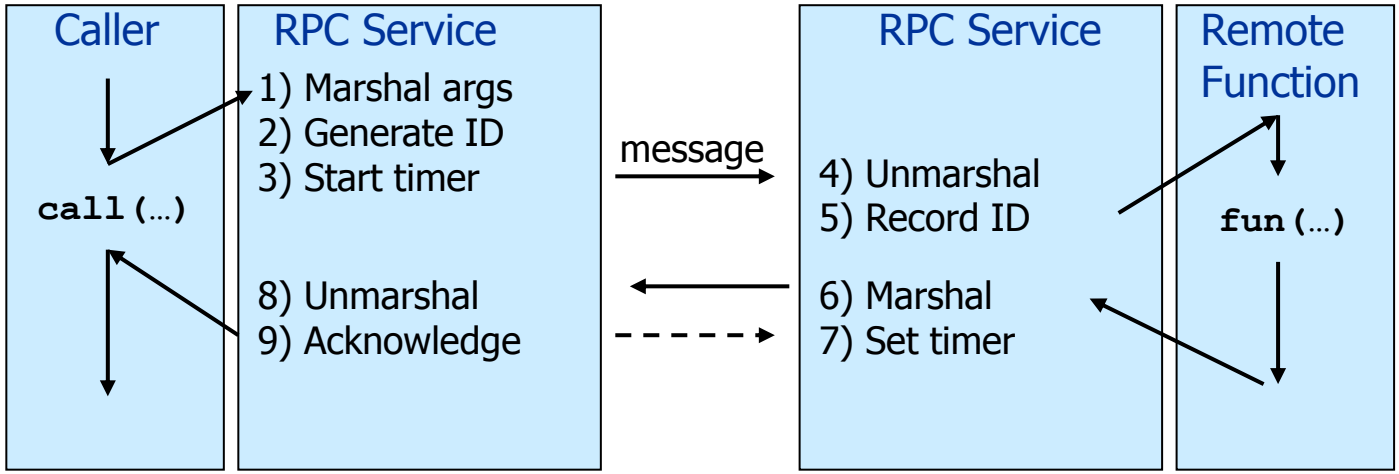


- Part I: Remote Procedure Call (RPC)
  - Historic interest
- Part II: Object-Oriented Middleware (OOM)
  - Ex. Java RMI
  - Ex. CORBA
- Part III: Message-Oriented Middleware (MOM)
  - Ex. Java Message Service
- Part IV: Event-Based Middleware
  - Cambridge Event Architecture

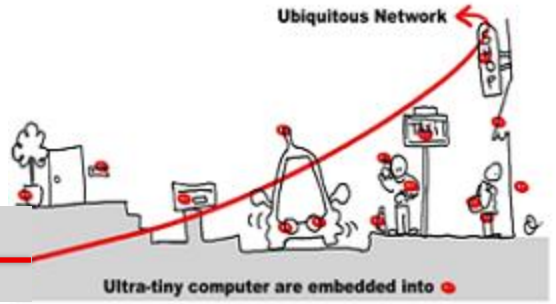
Part I: Remote Procedure Call (RPC)



- Masks remote function calls as being local
- Client/server model
- Request/reply paradigm usually implemented with message passing in RPC service
- Marshalling of function parameters and return value

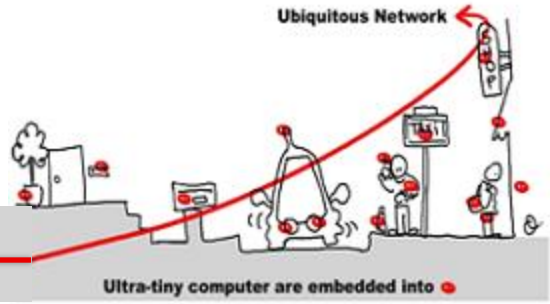


Properties of RPC



- Language-level pattern of function call
  - easy to understand for programmer
- Synchronous request/reply interaction
  - natural from a programming language point-of-view
  - matches replies to requests
  - built in synchronisation of requests and replies
- Distribution transparency (in the no-failure case)
  - hides the complexity of a distributed system

Failure Modes of RPC



- Invocation semantics supported by RPC in the light of:
  - network and/or server congestion,
  - client, network and/or server failure

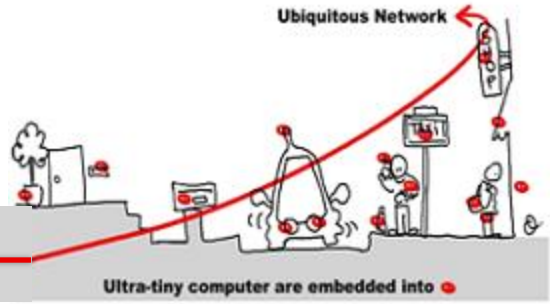
- RPC systems differ, many examples, local was Mayflower

Maybe or at most once (RPC system tries once)
- Error return – programmer may retry

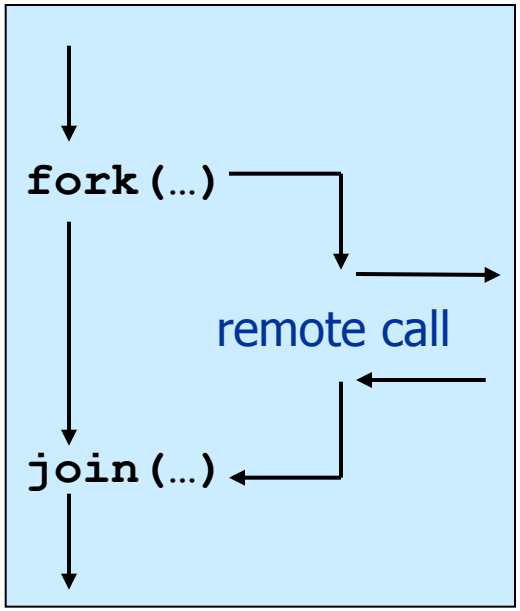
Exactly once (RPC system retries a few times)
- Hard error return – some failure most likely

note that “exactly once” cannot be guaranteed

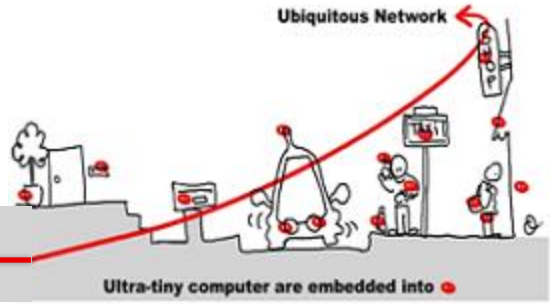
Disadvantages and limitations of RPC



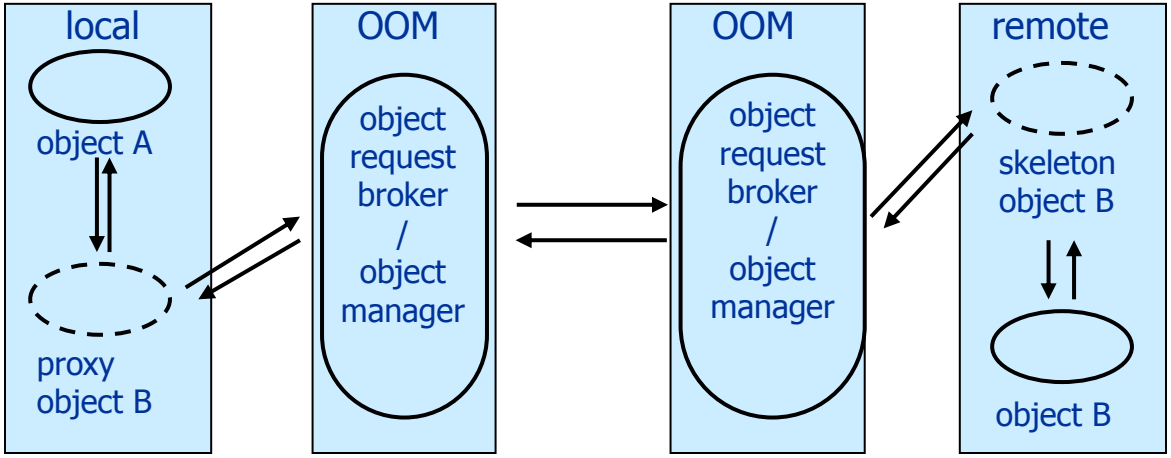
- Synchronous request/reply interaction
  - tight coupling between client and server
  - client may block for a long time if server loaded
  - leads to multi-threaded programming at client
  - slow/failed clients may delay servers when replying
  - multi-threading essential at servers
- Distribution Transparency
  - Not possible to mask all problems
- RPC paradigm is not object-oriented
  - invoke functions on servers as opposed to methods on objects



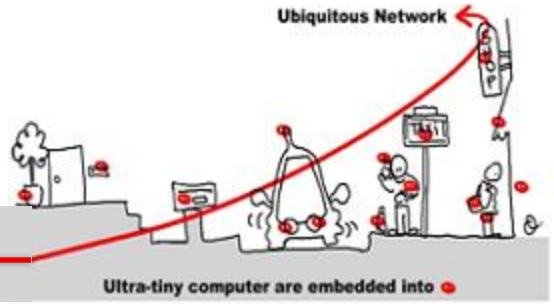
Part II: Object-Oriented Middleware (OOM)



- Objects can be local or remote
- Object references can be local or remote
- Remote objects have visible remote interfaces
- Masks remote objects as being local using proxy objects
- Remote method invocation

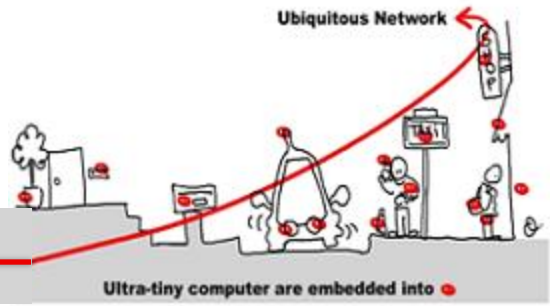


Properties of OOM



- Support for object-oriented programming model
  - objects, methods, interfaces, encapsulation, …
  - exceptions (were also in some RPC systems)
- Synchronous request/reply interaction
  - same as RPC
- Location Transparency
  - system (ORB) maps object references to locations

Java Remote Method Invocation (RMI)

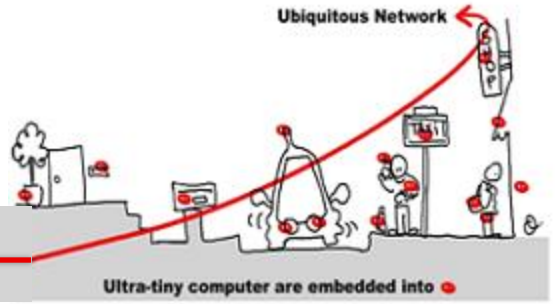


- Covered in Java programming
- Distributed objects in Java

```
public interface PrintService extends Remote {
  int print(Vector printJob) throws RemoteException;
}
```

  - RMI compiler creates proxies and skeletons
  - RMI registry used for interface lookup
  - Entire system written in Java (single-language system)

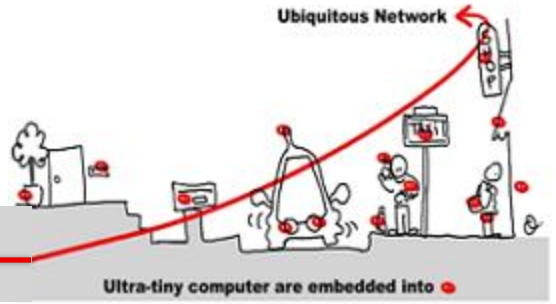
CORBA



- Common Object Request Broker Architecture
  - Open standard by the OMG (Version 3.0)
  - Language- and platform independent

- **Object Request Broker** (ORB)
  - General Inter-ORB Protocol (GIOP) for communication
  - Interoperable Object References (IOR) contain object location
  - CORBA **Interface Definition Language** (IDL)
    - Stubs (proxies) and skeletons created by IDL compiler

CORBA IDL

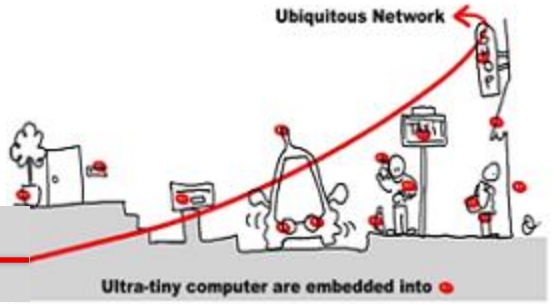


- Definition of language-independent remote interfaces
  - Language mappings to C++, Java, Smalltalk, …
  - Translation by IDL compiler
- Type system
  - basic types: long (32 bit), long long (64 bit), short, float, char, boolean, octet, any, …
  - constructed types: struct, union, sequence, array, enum
  - objects (common super type Object)

```
typedef sequence<string> Files;
interface PrintService : Server {
  void print(in Files printJob);
};
```

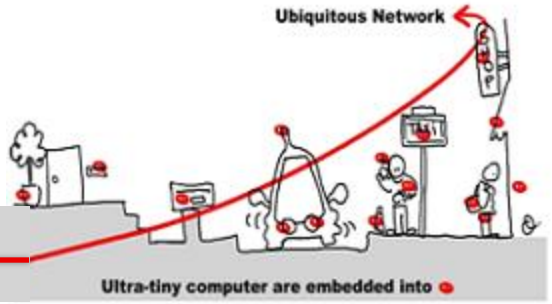
- Parameter passing
  - in, out, inout
  - basic & constructed types passed by value
  - objects passed by reference

CORBA Services (selection)



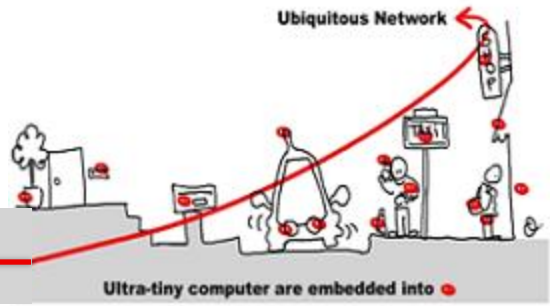
- Naming Service
  - Names ➔ remote object references
- Trading Service
  - Attributes (properties) ➔ remote object references
- Persistent Object Service
  - Implementation of persistent CORBA objects
- Transaction Service
  - Making object invocation part of transactions
- Event Service and Notification Service
  - In response to applications‘ need for asynchronous communication
  - built above synchronous communication with push or pull options

Disadvantages of OOM

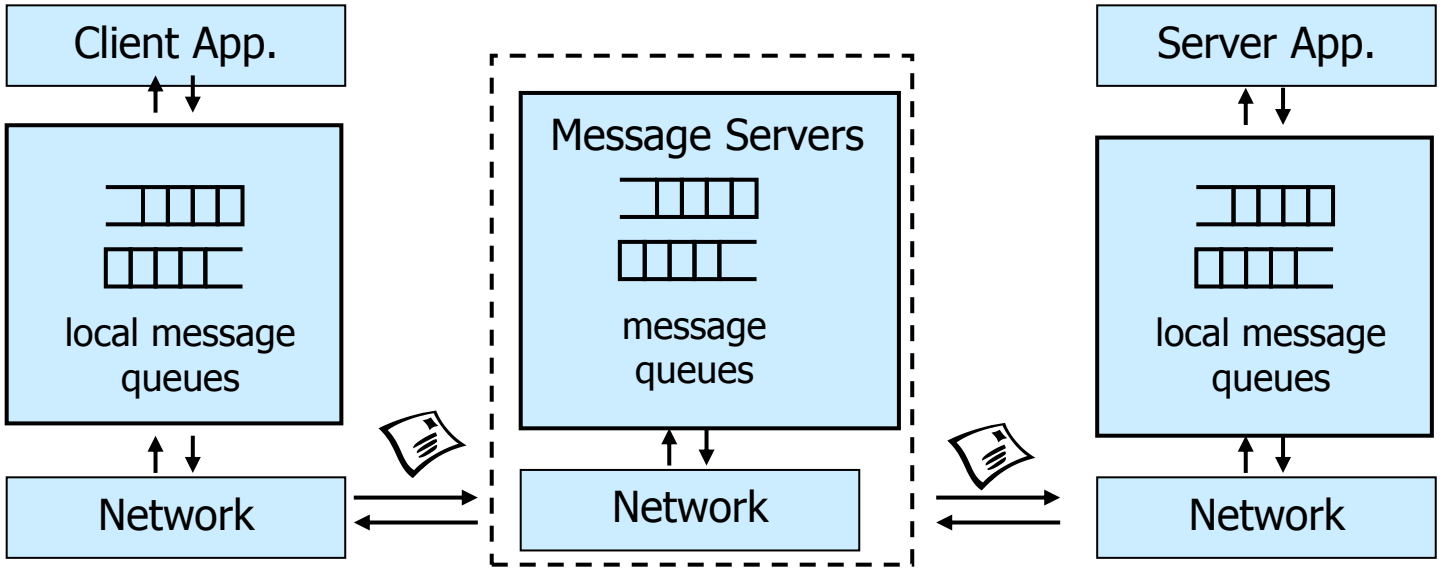


- Synchronous request/reply interaction only
  - So CORBA oneway semantics added Asynchronous Method Invocation (AMI)
  - But implementations may not be loosely coupled
- Distributed garbage collection
  - Releasing memory for unused remote objects
- OOM rather static and heavy-weight
  - Bad for ubiquitous systems and embedded devices

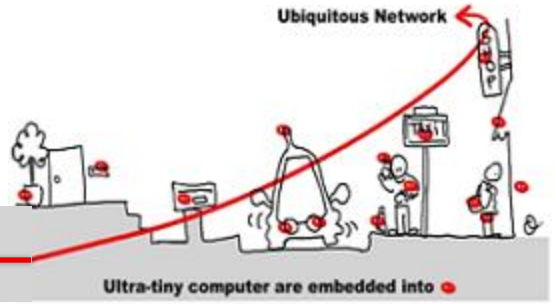
Part III: Message-Oriented Middleware (MOM)



- Communication using messages
- Messages stored in message queues
- message servers decouple client and server
- Various assumptions about message content

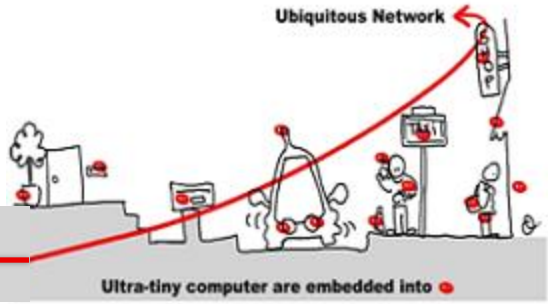


Properties of MOM



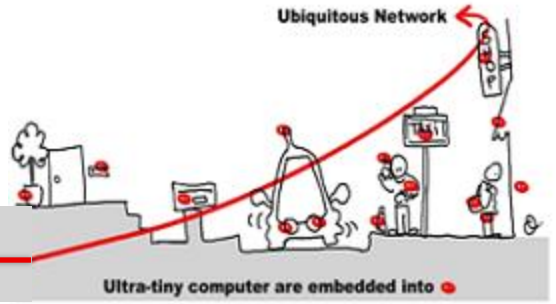
- Asynchronous interaction
  - Client and server are only loosely coupled
  - Messages are queued
  - Good for application integration
- Processing of messages by intermediate message server(s)
  - May do filtering, transforming, logging, …
  - Networks of message servers

Java Message Service (JMS)



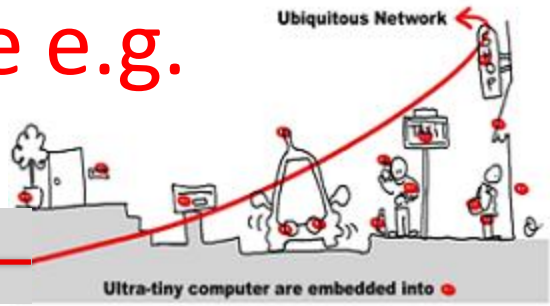
- API specification to access MOM implementations
- Two modes of operation *specified*:
  - Point-to-point
    - one-to-one communication using queues
  - Publish/Subscribe
    - cf. Event-Based Middleware
- JMS Server implements JMS API
- JMS Clients connect to JMS servers
- Java objects can be serialised to JMS messages

Disadvantages of MOM

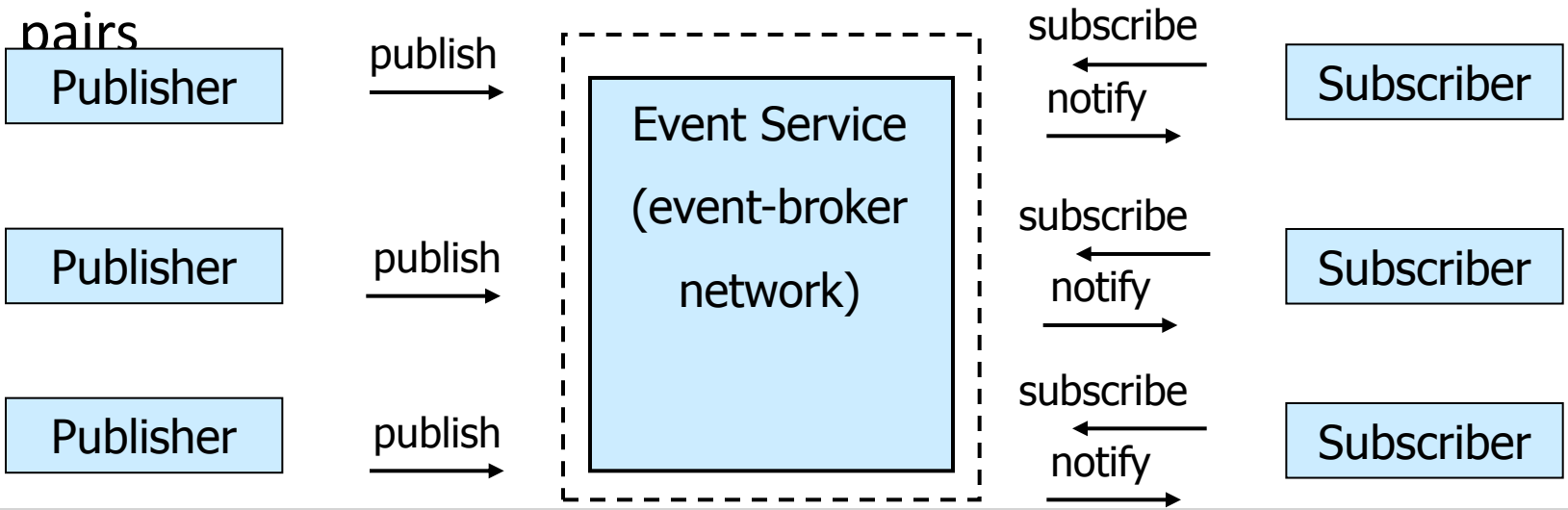


- Poor programming abstraction (but has evolved)
  - Rather low-level (cf. Packets)
  - Request/reply more difficult to achieve, but can be done
- Message formats originally unknown to middleware
  - No type checking (JMS addresses this – implementation?)
- Queue abstraction only gives one-to-one communication
  - Limits scalability (JMS pub/sub – implementation?)

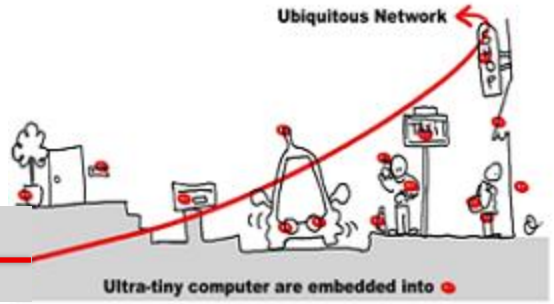
Part IV: Event-Based Middleware e.g. Publish/Subscribe



- Publishers (advertise and) publish events (messages)
- Subscribers express interest in events with subscriptions
- Event Service notifies interested subscribers of published events
- Events can have arbitrary content (typed) and name/value pairs

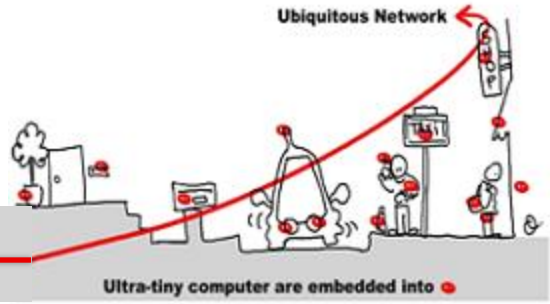


Properties of Publish/Subscribe

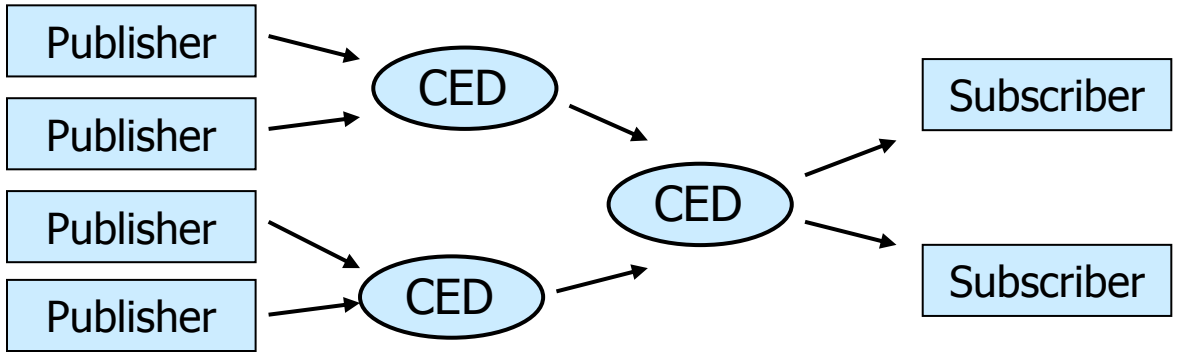


- Asynchronous communication
  - Publishers and subscribers are loosely coupled
- Many-to-many interaction between pubs. and subs.
  - Scalable scheme for large-scale systems
  - Publishers do not need to know subscribers, and vice-versa
  - Dynamic join and leave of pubs, subs
- (Topic and) Content-based pub/sub very expressive
  - Filtered information delivered only to interested parties

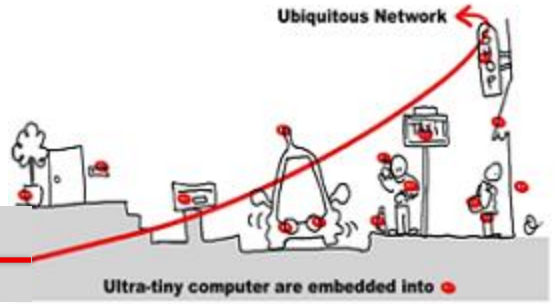
Composite Event Detection (CED)



- Content-based pub/sub may not be expressive enough
  - Potentially thousands of event types (primitive events)
  - Subscribers interest: event patterns
- Composite Event Detectors (CED)
  - Subscribe to primitive events and publish composite events



Summary

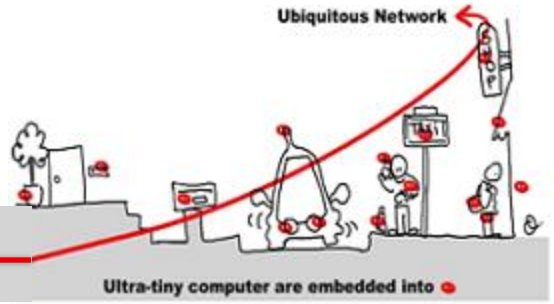


- Middleware is an important abstraction for building distributed systems

1. Remote Procedure Call
2. Object-Oriented Middleware
3. Message-Oriented Middleware
4. Event-Based Middleware

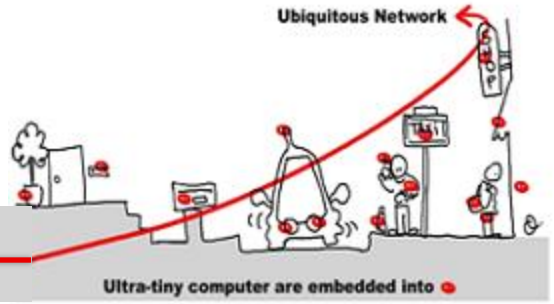
- Synchronous vs. asynchronous communication
- Scalability, many-to-many communication
- Language integration
- Ubiquitous systems, mobile systems

Classification: Where to use them ?



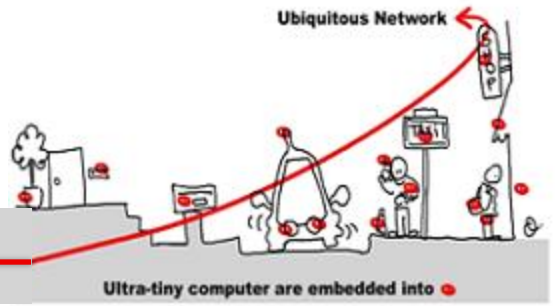
- *Middleware: Past and Present a Comparison*, [Hennadiy Pinus 2004]
- MOM can be used in the applications,where the network or all-components availability is not warranted
- RPCs could be used in small, simple applications with primarily point-to-point communication. "RPCs are not a good choice to use as the building blocks for enterprise-wide applications where high performance and high reliability are needed.“

Classification: Where to use them ?



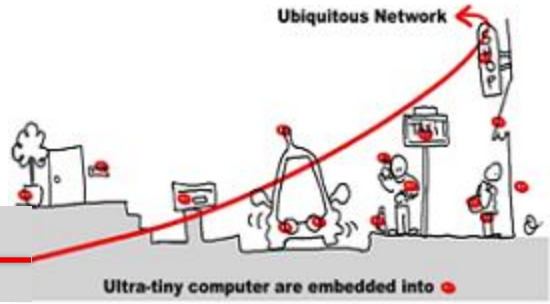
- *Middleware: Past and Present a Comparison*, [Hennadiy Pinus 2004]
- OOM "should be considered for applications where immediate scalability requirements are somewhat limited. These applications should be part of a long-term strategy towards object orientation.“
- Event-based “is a decoupled, many-to-many communication paradigm, well-adapted for building large-scale distributed systems” [Pietzuch 2002]

Recent trends : SoM, Service oriented Middleware



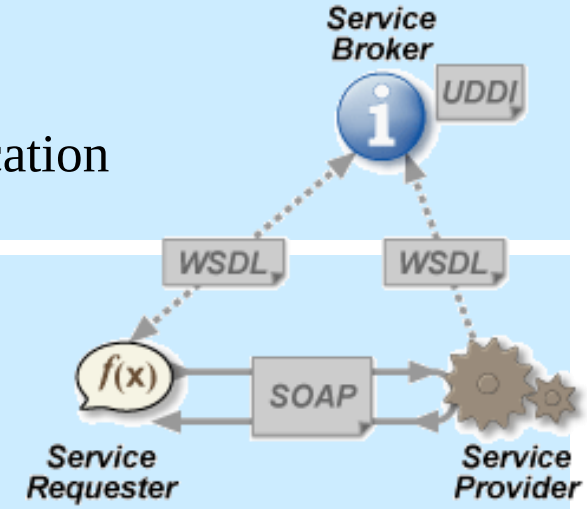
- The SOA style is structured around three key architectural components: (i) service provider, (ii) service consumer, and (iii) service registry.
- The SOA design is structured around four key functionalities : service description, discovery, access and composition in the Future Internet of services
- The Service-Oriented Middleware (SOM) is in charge of enabling the deployment of services and coordination among the four key conceptual elements and four key functionalities that characterize the SOA style and design.

Web Services



Communication

- Message content expressed in **XML**
- **S**imple **O**bject **A**ccess **P**rotocol (**SOAP**)
  - Lightweight protocol for sync/async communication



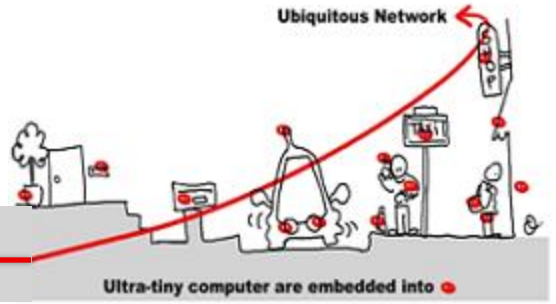
Service Description

- **W**eb **S**ervices **D**escription **L**anguage (**WSDL**)
  - Interface description for web services

Service Discovery

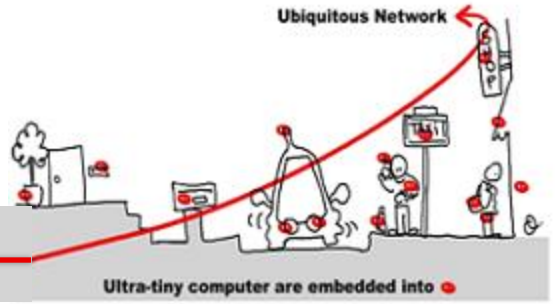
- **U**niversal **D**escription **D**iscovery and **I**ntegration (**UDDI**)
  - Directory with web service description in WSDL

XML

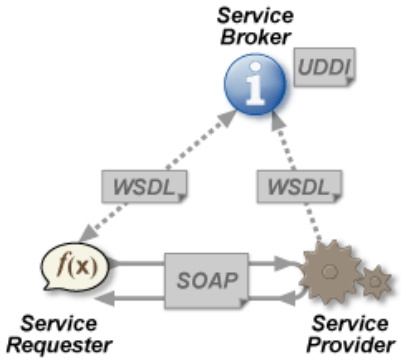


- Simple markup, data description language
- “Human-readable”
- Customizable schema can be used to define attributes and elements (meta-language)
- XML can be used to created custom XML-based languages and services (meta-language)
- XML-based messages transported over HTTP serve as the basic platform for Web Services

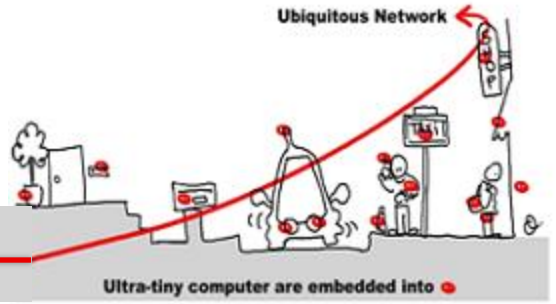
SOAP



- XML-based protocol defining message format
- One-way asynchronous technology
- Can use a variety of message passing styles: RPC, publish/subscribe
- Primary underlying protocol is HTTP, but others can be used (SMTP)

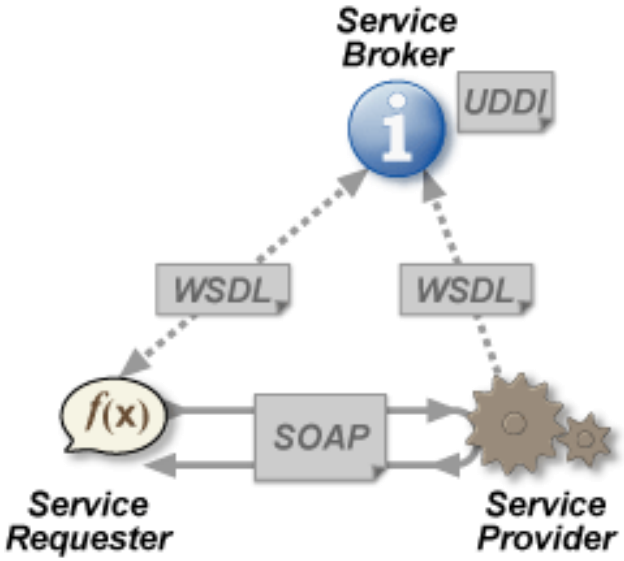


WSDL

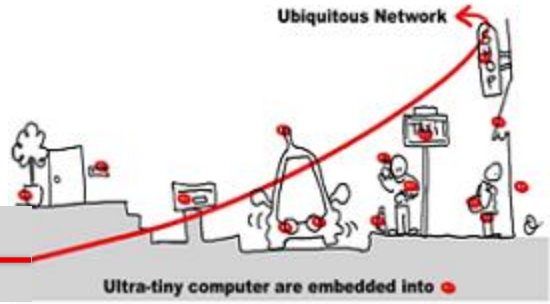


- XML-based language

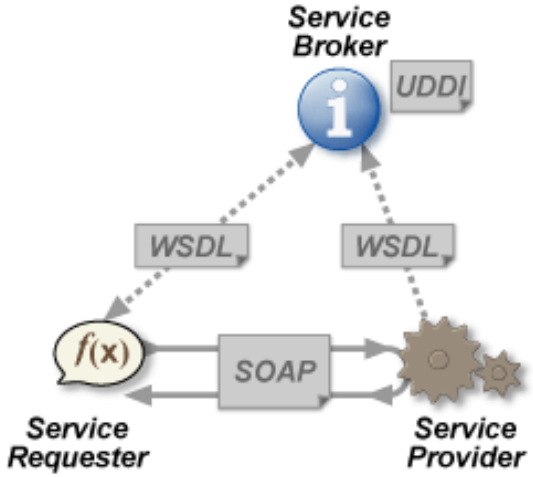
- Defines/describes Web services interfaces, data and message types, interaction patterns and protocol mappings



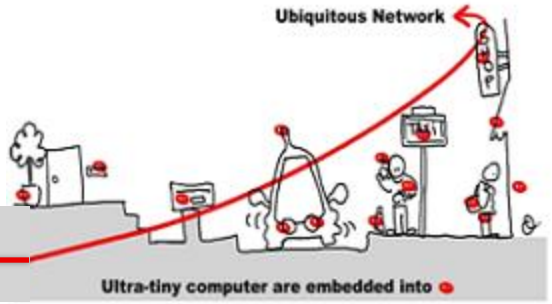
UDDI



- Web Services Registry (of WSDL documents)
- Protocol for discovering and publishing Web Services
- UDDI registry is accessed by XML-based SOAP messages

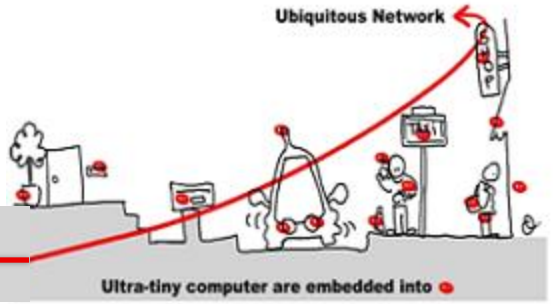


Properties of Web Services



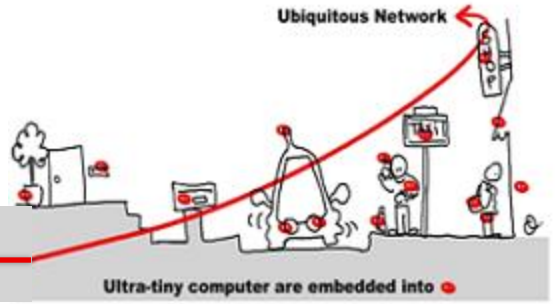
- Language-independent and open standard
- SOAP offers OOM and MOM-style communication:
  - Synchronous request/reply like OOM
  - Asynchronous messaging like MOM
  - Supports internet transports (http, smtp, ...)
  - Uses XML Schema for marshalling types to/from programming language types
- WSDL says how to use a web service
  - **http://api.google.com/GoogleSearch.wsdl**
- UDDI helps to find the right web service
  - Exports SOAP API for access

Disadvantages of Web Services



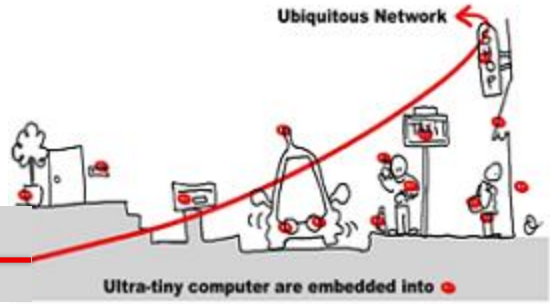
- Low-level abstraction
  - leaves a lot to be implemented
- Interaction patterns have to be built
  - one-to-one and request-reply provided
  - one-to-many?
  - still synchronous service invocation, rather than notification
- No location transparency

What we lack, so far



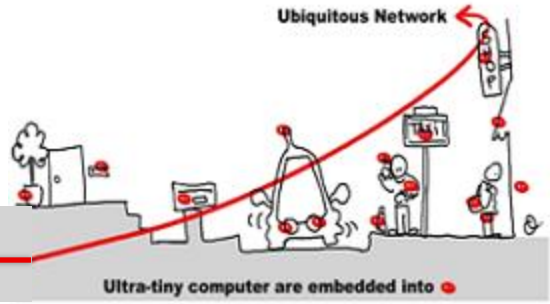
- General interaction patterns
  - we have one-to-one and request-reply
  - one-to-many? many to many?
  - notification?
  - dynamic joining and leaving?
- Location transparency
  - anonymity of communicating entities
- Support for Device
  - data values from sensors
  - lightweight software

References



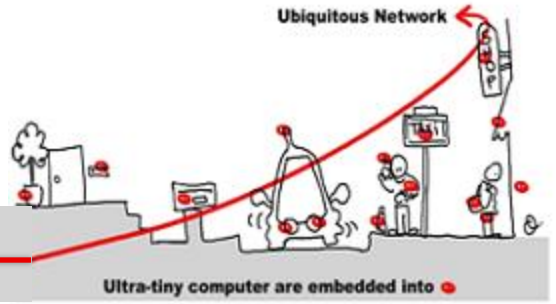
- Peter R. Pietzuch, Event-Based Middleware: A New Paradigm for Wide-Area Distributed Systems?, In 6th CaberNet Radicals Workshop, February 2002
- Hennadiy Pinus, Middleware: Past and Present a Comparison, June 2004
- Wolfgang Emmerich. Software engineering and middleware: a roadmap. In Proceedings of the Conference on The future of Software engineering, pages 117-129, 2000.
- Hurwitz, Judith "Sorting Out Middleware". DBMS magazine, January, 1998.

References



- D.E. Bakken, Middleware, in Encyclopedia of Distributed Computing, 2003, http://www.eecs.wsu.edu/~bakken/middleware-article-bakken.pdf.
- A Perspective on the Future of Middleware-based Software Engineering. V. Issarny, M. Caporuscio, N. Georgantas. In Future of Software Engineering 2007 (FOSE) at ICSE (International Conference on Software Engineering). L. Briand and A. Wolf editors, IEEE-CS Press. 2007.
- Sacha Krakowiak, Krakowiak, Sacha. "What's middleware? " , 2005, http://sardes.inrialpes.fr/~krakowia/MW-Book/Chapters/Intro/intro.html

References



- Service-Oriented Middleware for the Future Internet: State of the Art and Research Directions, Valérie Issarny, Nikolaos Georgantas, Sara Hachem, Apostolos Zarras, Panos Vassiliadis, Marco Autili, Marco Aurelio Gerosa, Amira Ben Hamida, Journal of Internet Services and Applications 2, 1 (2011) 23-45