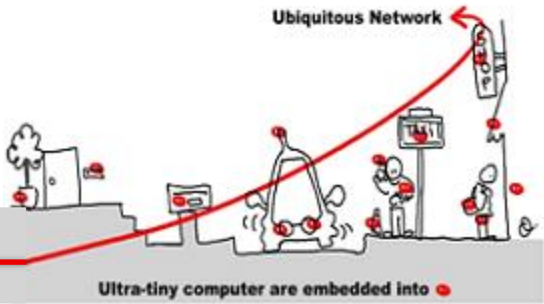
Middleware for Ubiquitous Computing



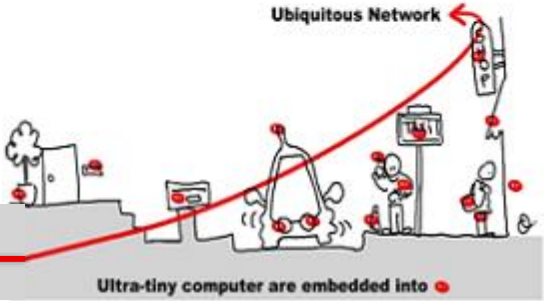
- Middleware … from distributed systems to network of things
- UbiComp Middleware :

http://www.tigli.fr/doku.php?id=cours:muc_2014_2015

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

Email : tigli@polytech.unice.fr

Middleware for Internet of Things : reference



- **Middleware Solutions for the Internet of Things, Delicato**, Flávia C., **Pires**, Paulo F., **Batista**, Thais, Sep 2013, Springer, ISBN 978-1-4471-5481-5

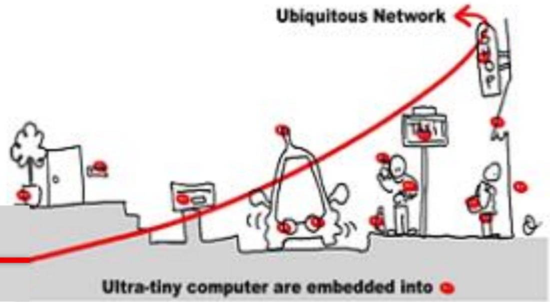
SPRINGER BRIEFS IN COMPUTER SCIENCE

Flávia C. Delicato
Paulo F. Pires
Thais Batista

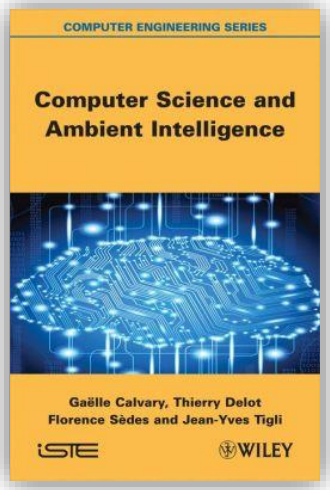
Middleware Solutions for the Internet of Things

Springer

New challenges in ubiquitous computing : reference



[2013] Gaëlle Calvary, Thierry Delot, Florence Sèdes, **Jean-Yves Tigli**, editors. “Computer Science and Ambient Intelligence” 335 pages, ISTE Ltd and Wiley & Sons Inc, March 2013, ISBN 978-1-84821-437-8

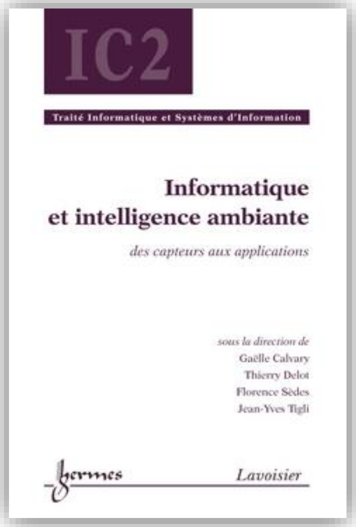

COMPUTER ENGINEERING SERIES

Computer Science and Ambient Intelligence

Gaëlle Calvary, Thierry Delot
Florence Sèdes and Jean-Yves Tigli

ISTE WILEY

[2012] Gaëlle Calvary, Thierry Delot, Florence Sèdes, **Jean-Yves Tigli**. “Informatique et Intelligence Ambiante : des Capteurs aux Applications (Traité Informatique et Systèmes d'Information, IC2)” Hermes Science, July 2012, ISBN 2-7462-2981-1


IC2

Traité Informatique et Systèmes d'Information

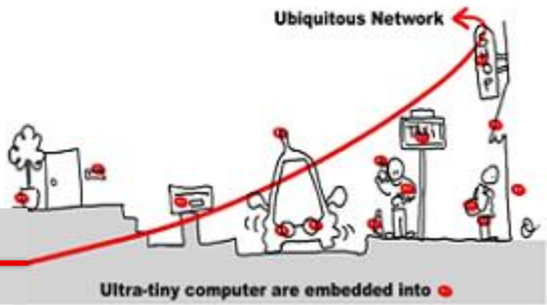
Informatique
et intelligence ambiante

des capteurs aux applications

sous la direction de
Gaëlle Calvary
Thierry Delot
Florence Sèdes
Jean-Yves Tigli

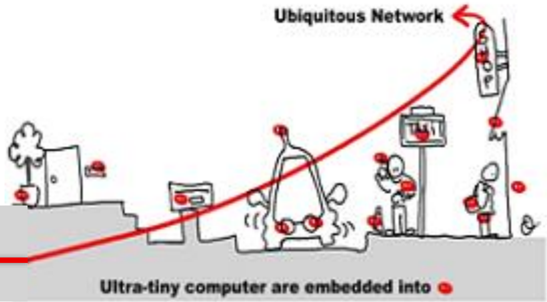
hermes Lavoisier

Course Outline



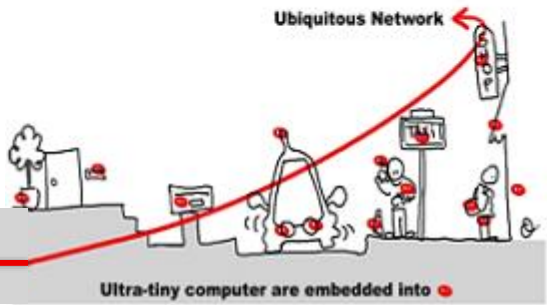
- Middleware for Internet of Things and Web of Things
- Web Service for Device – UPnP Tutorial
- Services for Device Composition – Lightweight Component based Composition Tutorial
- Validation and Model checking – Synchronous Language to model software component and composition -Tutorial
- Adaptation and Dynamic composition of Services for Device
- Tutorial : Aspect of Assembly

Examination



- Evaluated Tutorial on Synchronous Language and Model checking
- A final paper exam and/or or computer exam

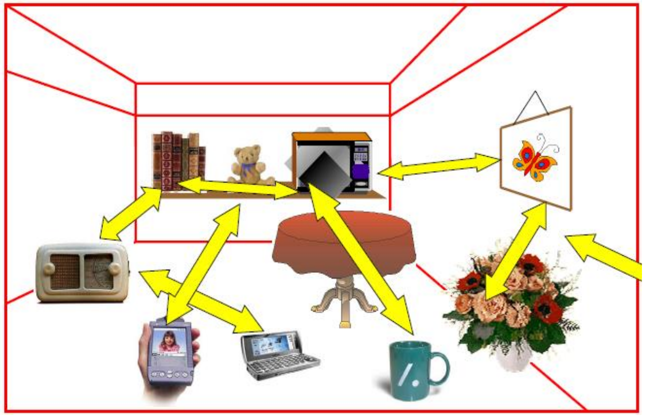
Introdution :
Ubiquitous Computing since 1991

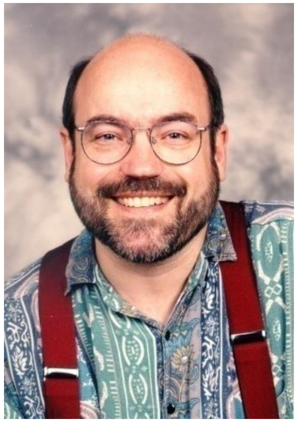


- Pervasive, Ubiquitaire, Disappearing, Ambient Computing …

- [Weiser 1991]

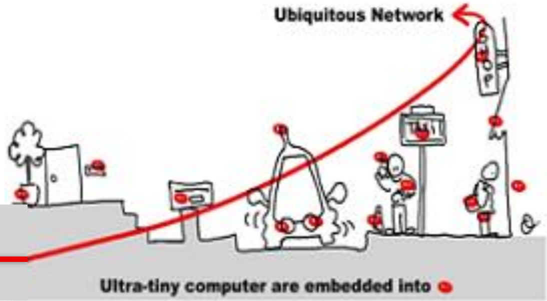
« Silicon-based information technology, is far from having become part of the environment »





They are Things …

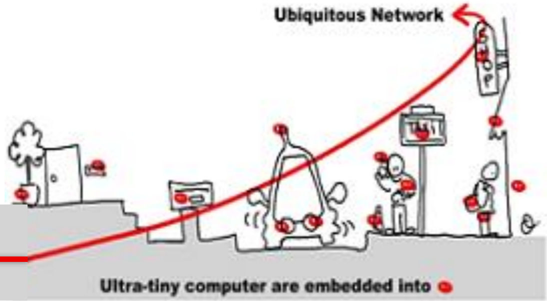
Introduction : Internet of Things



- It is predicted that there will be 31 billion things connected to the internet

- Environments, buildings, vehicles, clothing, portable devices and other objects will have the ability :
  - to sense,
  - to act
  - analyse,
  - communicate,
  - network and
  - produce new information

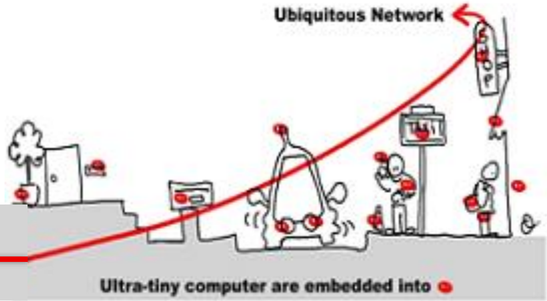
Video



- What are « Things », Internet of Things, Adaptation and Dynamic composition of Services for Things and Device, in our everyday life.

- http://www.tigli.fr/doku.php?id=cours:muc_2013_2014

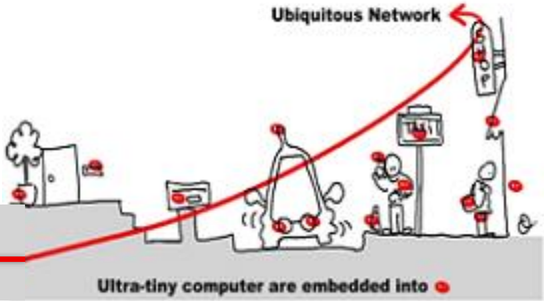
Integrateur-GBHD.mov (Ligne de commande)

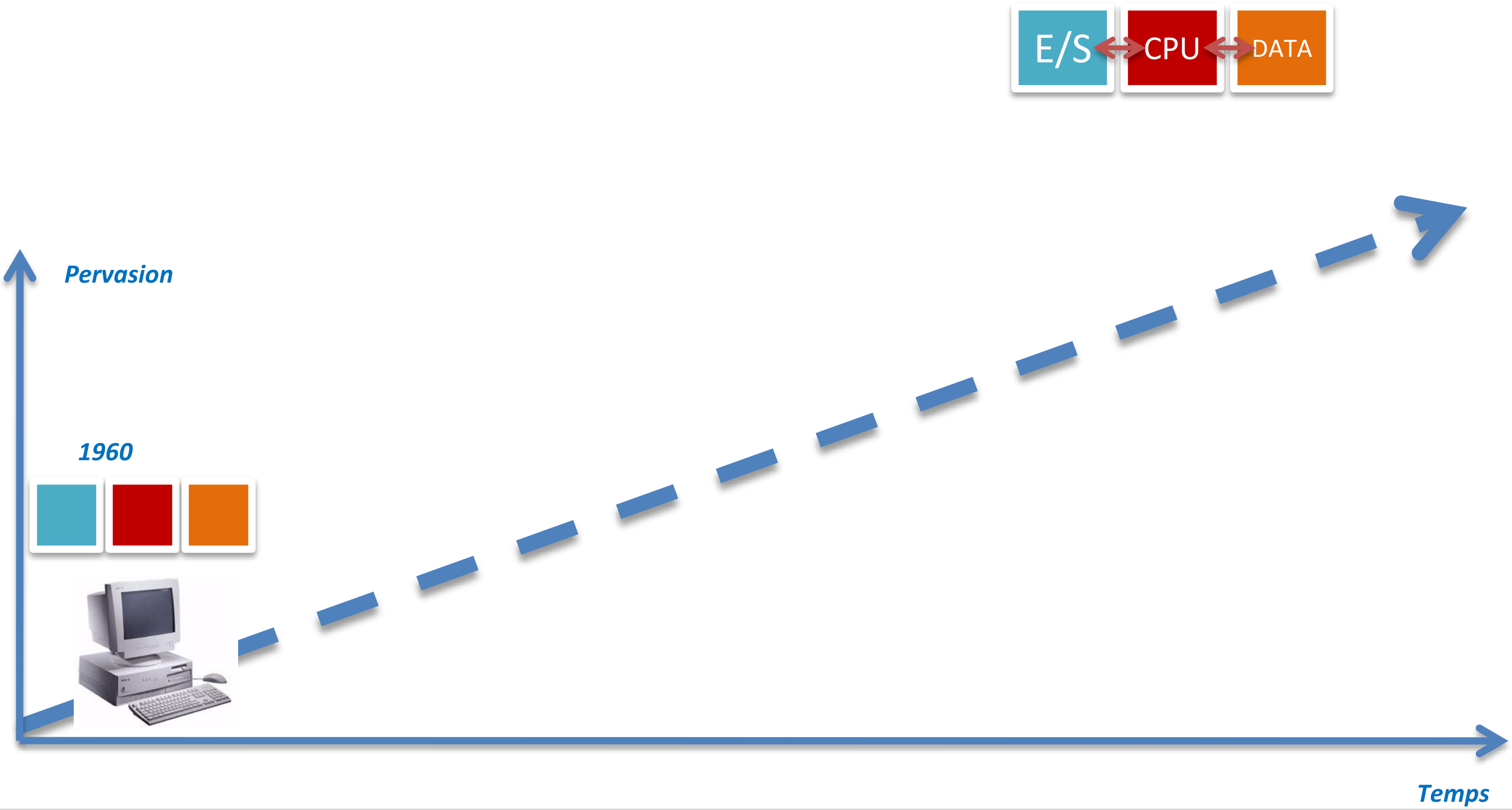


Middleware for distributed Computers

From Distributed Computing to Ubiquituous Computing

From Von Neumann Computer :





E/S ↔ CPU ↔ DATA

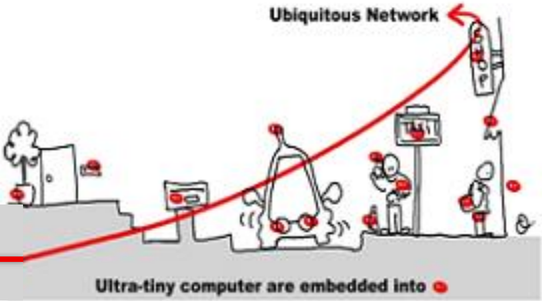
Pervasion

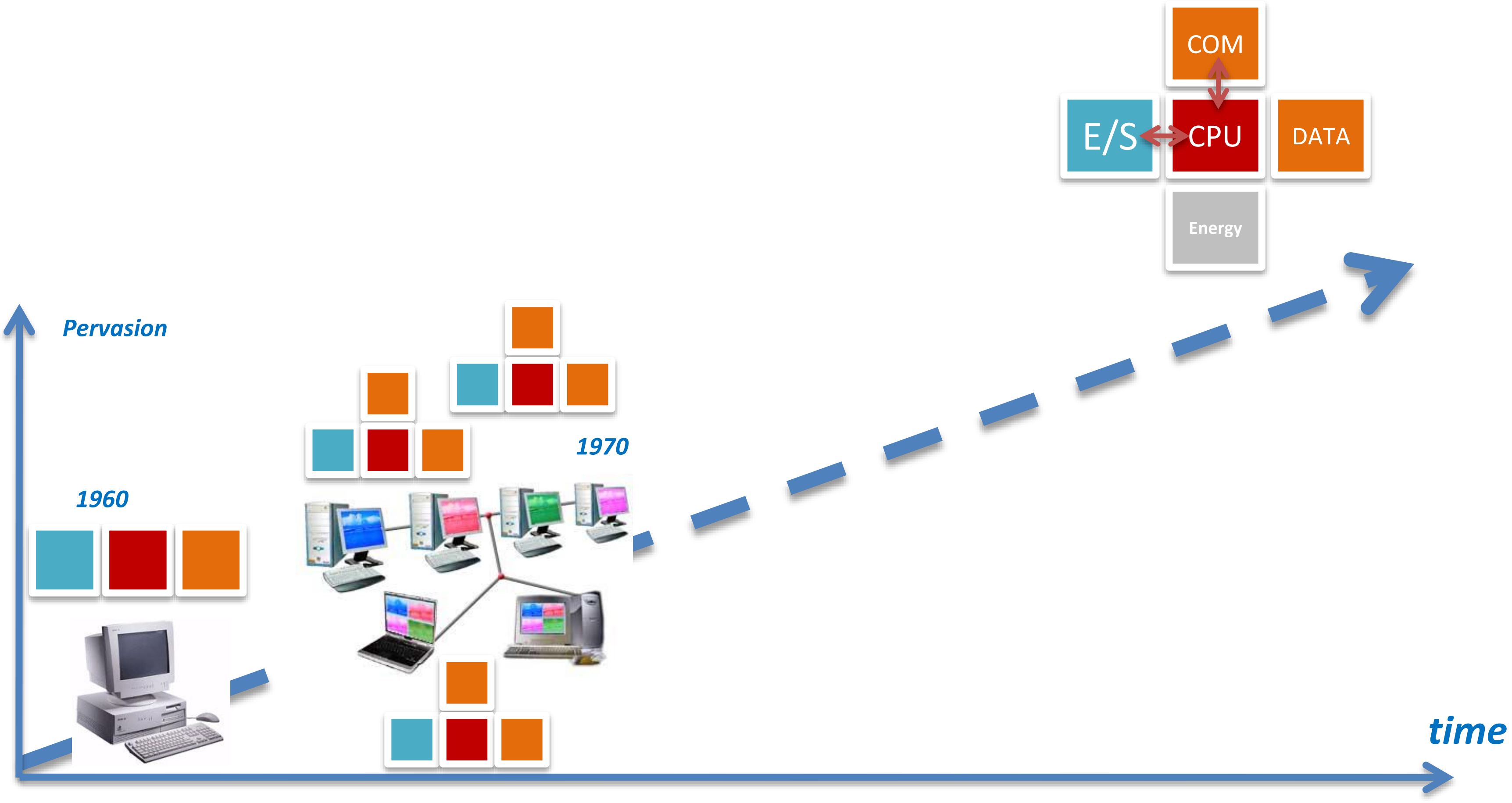
1960

Temps

Adapté de « Nano-informatique et intelligence ambiante », JB Waldner, Hermes Science Publishing, 2007

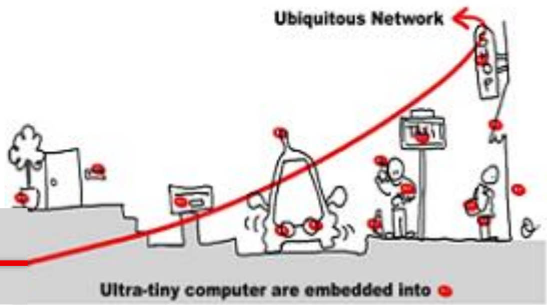
To Distributed Computing …



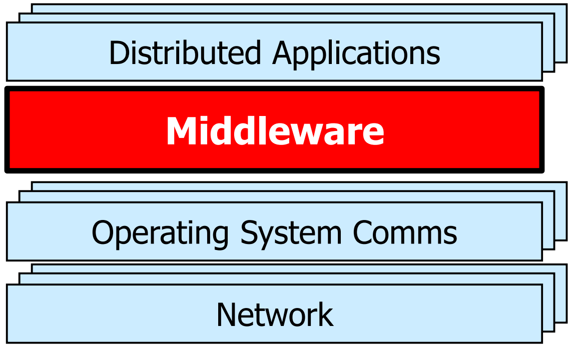


Adapté de « Nano-informatique et intelligence ambiante », JB Waldner, Hermes Science Publishing, 2007

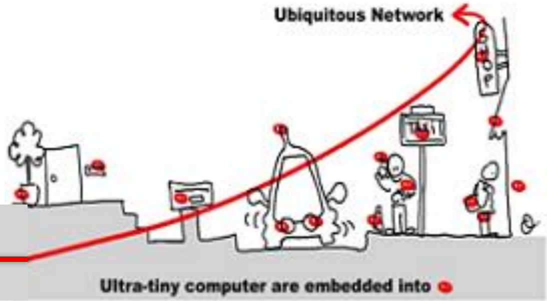
What is Middleware ?



- 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.



Generic definition



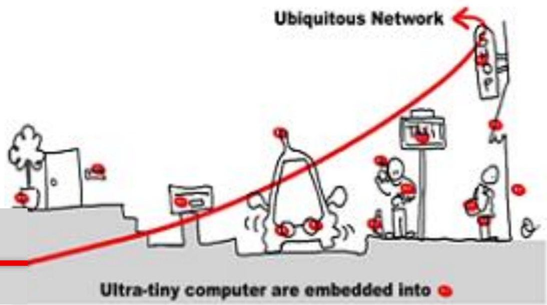
- *“The intersection of the stuff that network engineers don’t want to do with the stuff that applications developers don’t want to do.”*

  [Kenneth J. Klingenstein ('99)]

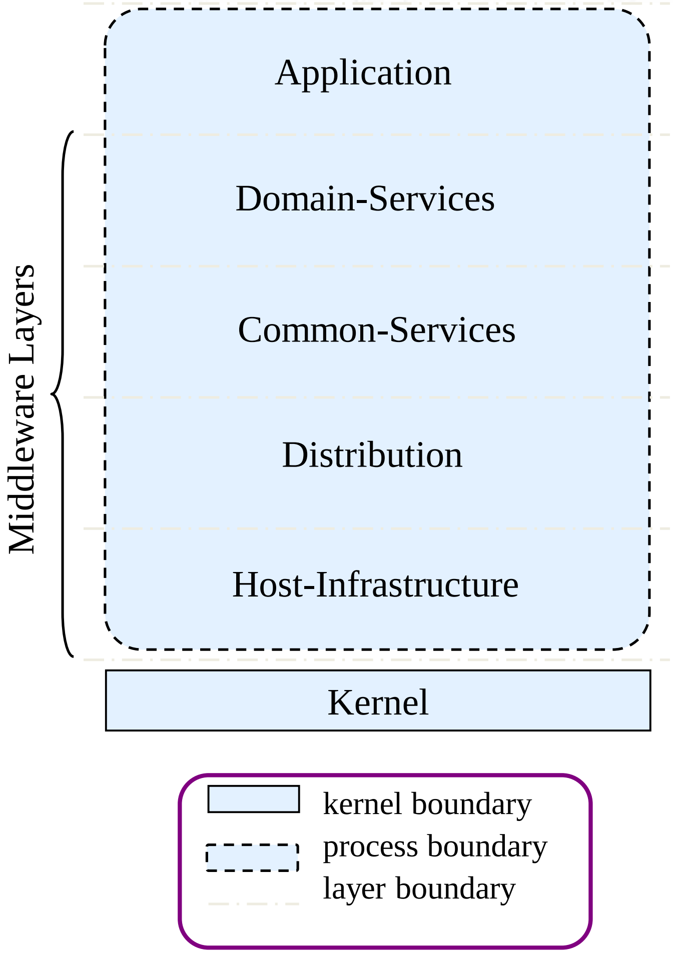
- But mainly to deal with distribution of software applications at the beginning

Middleware can be a Stack of Middlewares :

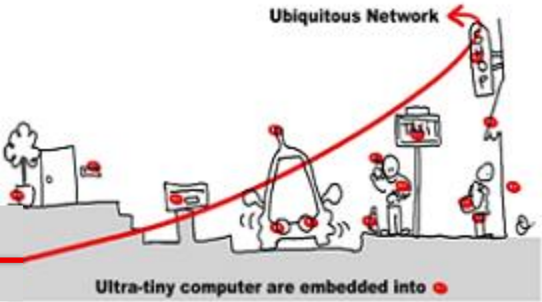
E. Schantz and D. C. Schmidt Taxonomy (2002)



- Schantz and Schmidt decomposed middleware into four layers:
  - Domain-Services
    - Tailored to a specific class of distributed applications
  - Common-Services
    - Functionality such as fault tolerance, security, load balancing and transactions
  - Distribution
    - Programming-language abstraction
  - Host-Infrastructure
    - Platform-abstraction



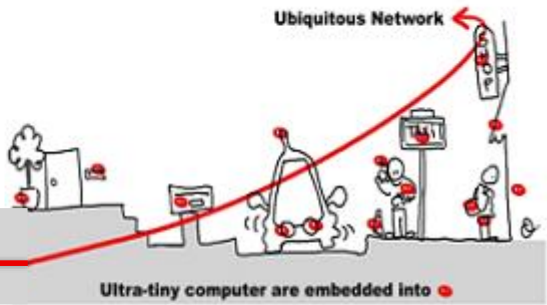
Middleware and software distribution



- 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.

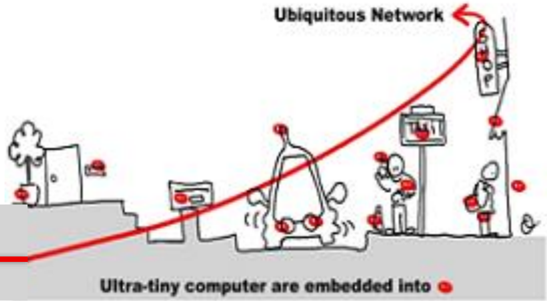
MAIN REFERENCE:
http://sardes.inrialpes.fr/~krakowia/MW-Book/Chapters/Preface/preface.html

Middleware and Communication Patterns

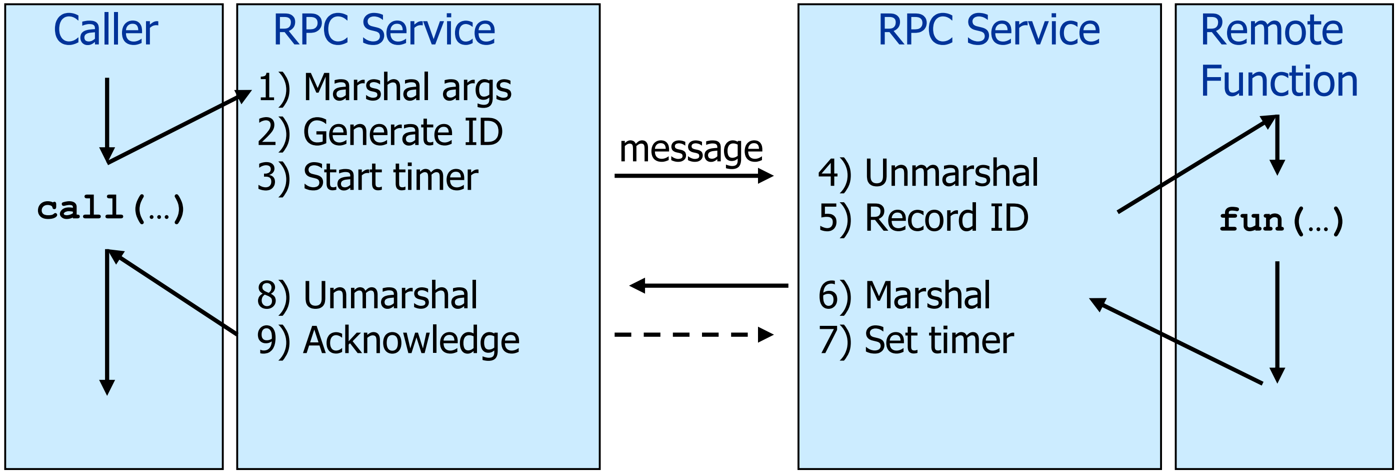


- I: Remote Procedure Call (RPC)
  - Historic interest
- II: Object-Oriented Middleware (OOM)
  - Ex. Java RMI
  - Ex. CORBA
- III: Message-Oriented Middleware (MOM)
  - Ex. Java Message Service
- IV: Event-Based Middleware
  - Cambridge Event Architecture
- They are prerequisites !
- **If you need practice, feel free to ask some tutorials and references**

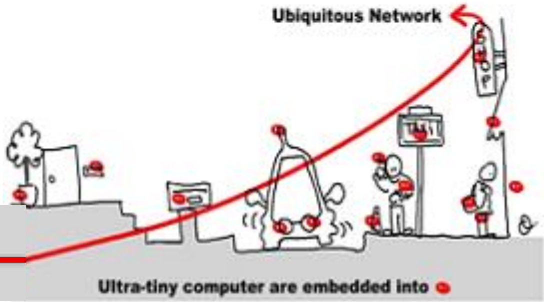
I: Remote Procedure Call (RPC)



- 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

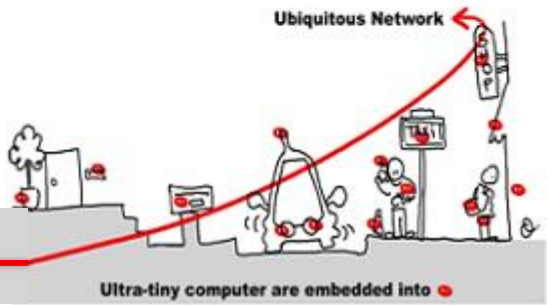


Properties of RPC

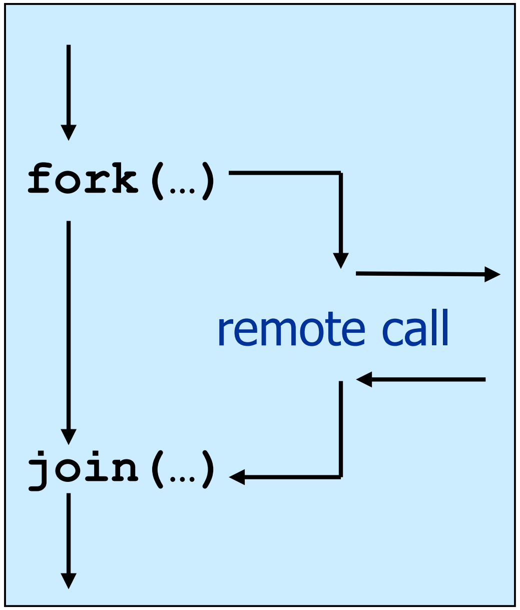


- Language-level pattern of function call
  - easy to understand for programmer
- Synchronous request/reply interaction
  - natural from a programming language point-of-view
  - matches replies to requests
  - built in synchronisation of requests and replies
- Distribution transparency (in the no-failure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

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

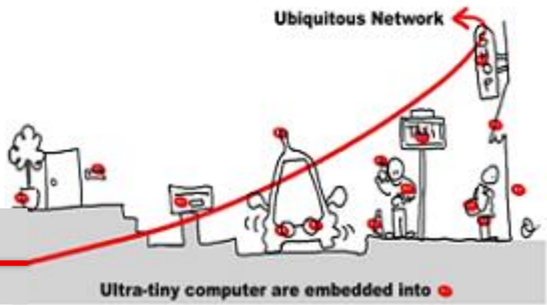


fork(…)

remote call

join(…)

Do you know ?

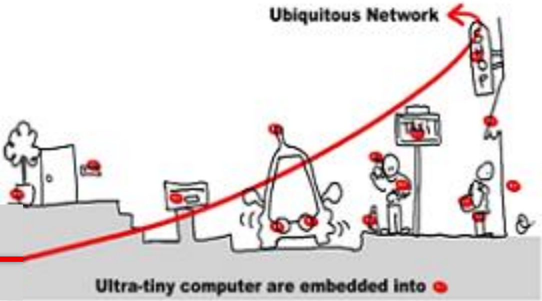


- Any example for RPC based Middleware ?

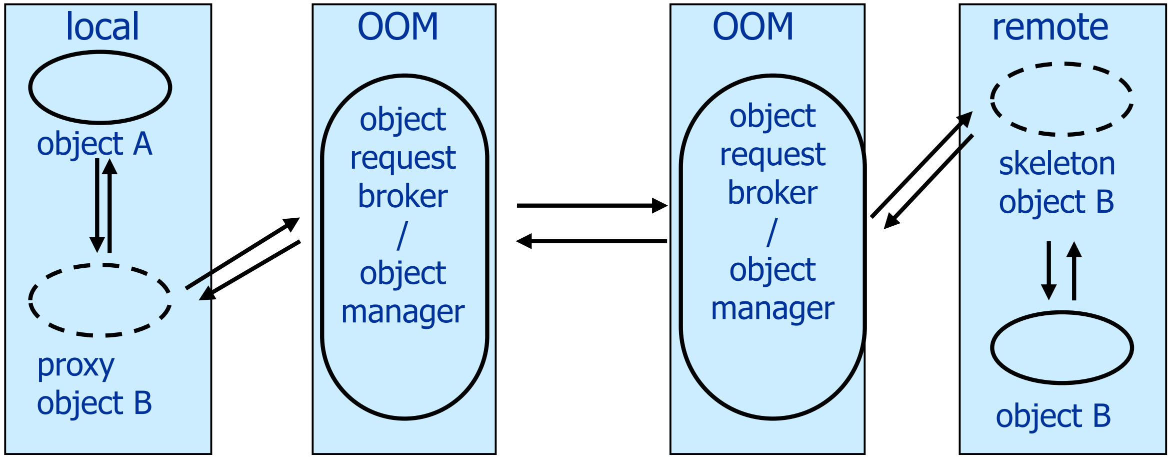
- in your background …

- Example :
  - See XML-RPC : http://www.tutorialspoint.com/xml-rpc/
  - One kind of Web Service Middleware Communication paradigm is RPC
    - See W3C consortium : http://www.w3schools.com/webservices/

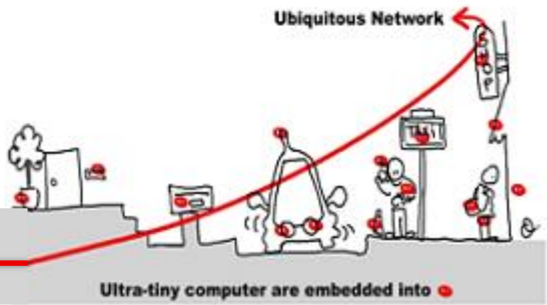
II: Object-Oriented Middleware (OOM)



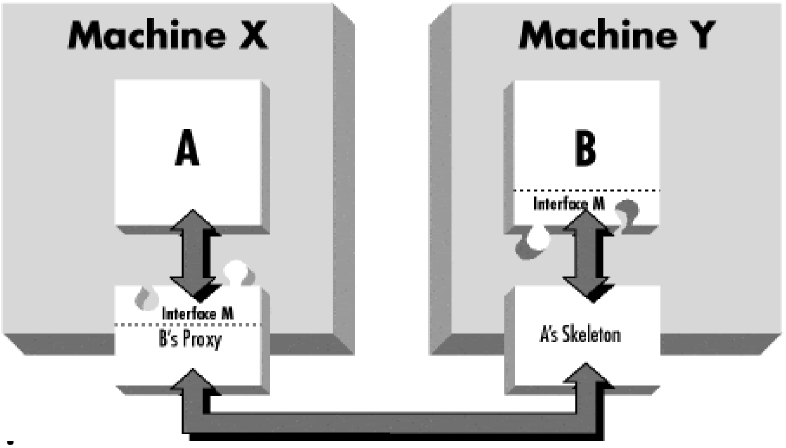
- Objects can be local or remote
- Object references can be local or remote
- Remote objects have visible remote interfaces
- Masks remote objects as being local using proxy objects
- Remote method invocation


local
object A
proxy object B
OOM
object request broker / object manager
OOM
object request broker / object manager
remote
skeleton object B
object B

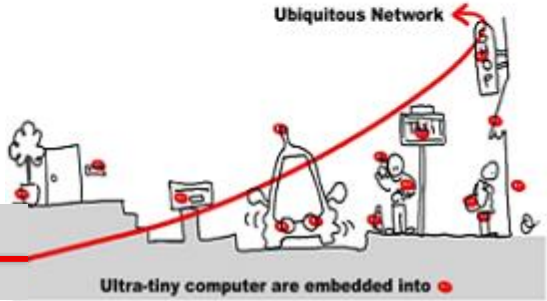
Properties of OOM



- 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

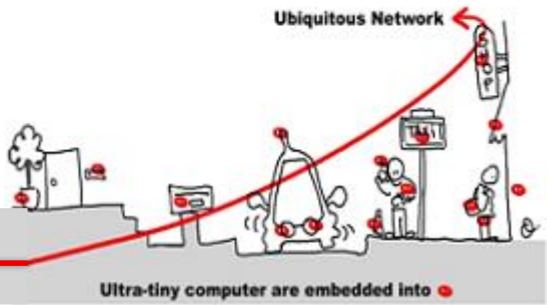


Do you know ?



- Any example for OOM ?
- in your background …
- Examples …

Java Remote Method Invocation (RMI)

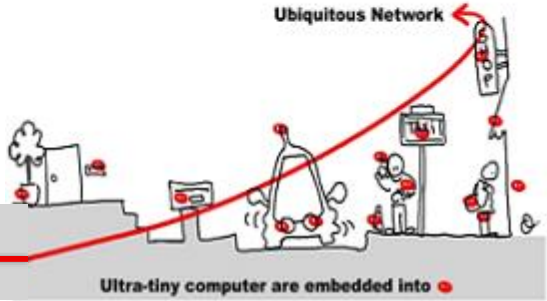


- Covered in Java programming
- Distributed objects in Java

```
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)

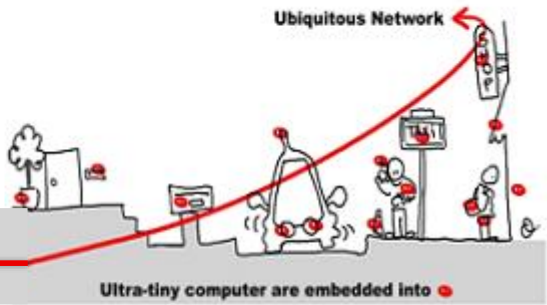
CORBA



- 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

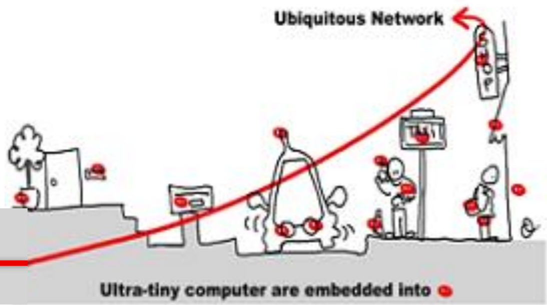


- Definition of language-independent remote interfaces
  - Language mappings to C++, Java, Smalltalk, …
  - Translation by IDL compiler
- Type system
  - basic types: long (32 bit), long long (64 bit), short, float, char, boolean, octet, any, …
  - constructed types: struct, union, sequence, array, enum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

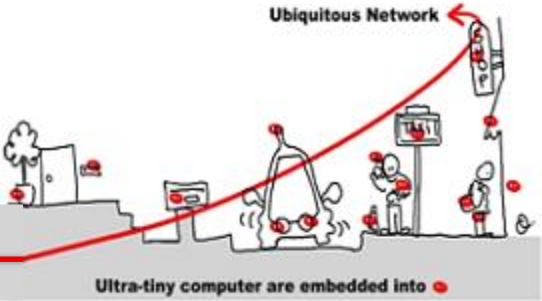
Advantages and Disadvantages of OOM



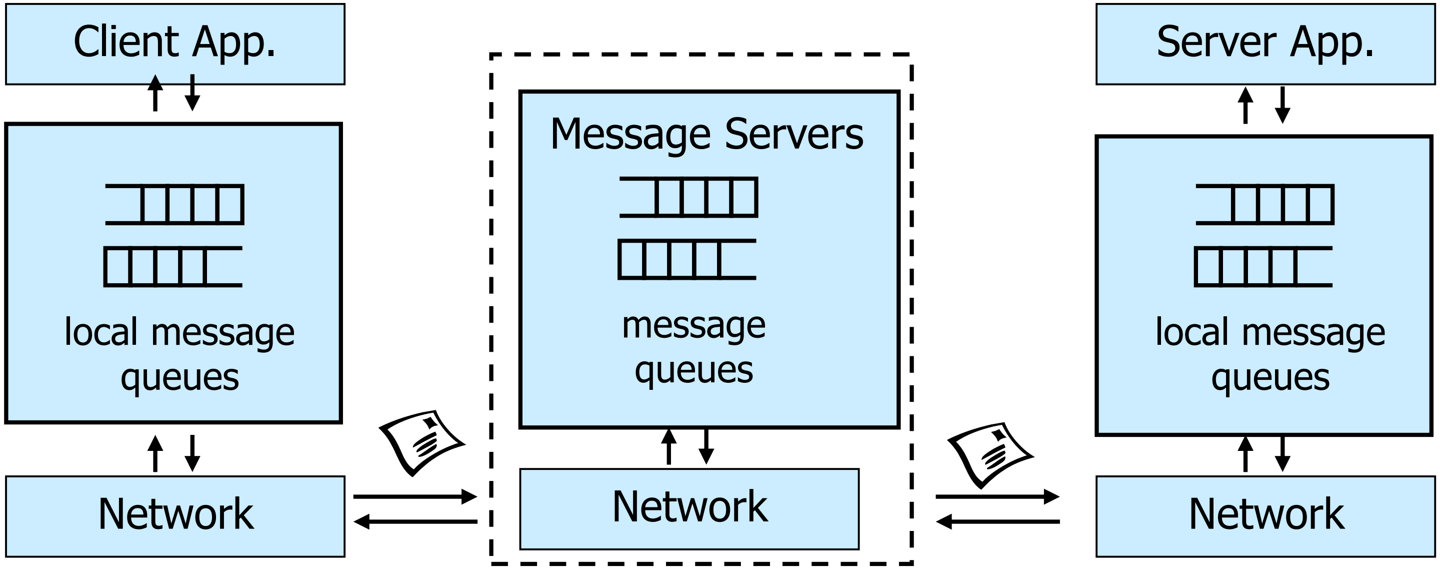
- Totally transparent distributed programming

- 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
  - Unadapted for ubiquitous systems and embedded devices

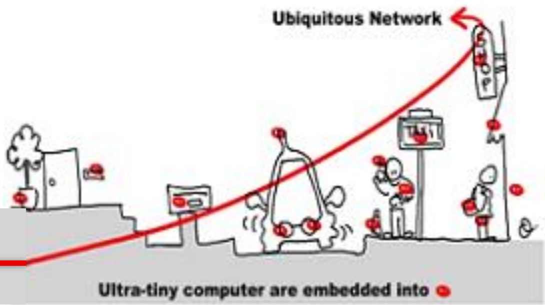
III: Message-Oriented Middleware (MOM)



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

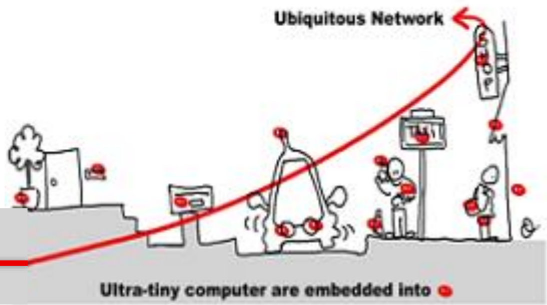


Properties of MOM



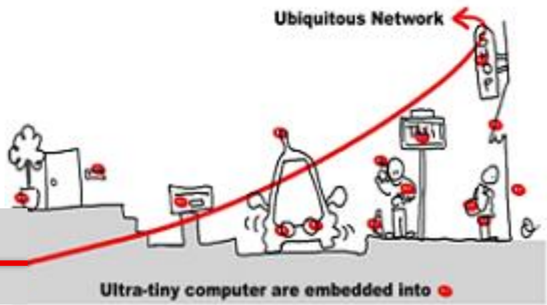
- Asynchronous interaction
  - Client and server are only loosely coupled
  - Messages are queued
  - Good for application integration
- Processing of messages by intermediate message server(s)
  - May do filtering, transforming, logging, …
  - Networks of message servers

Java Message Service (JMS)



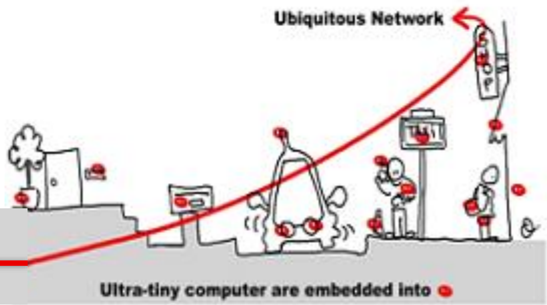
- API specification to access MOM implementations
- Two modes of operation *specified*:
  - Point-to-point
    - one-to-one communication using queues
  - Publish/Subscribe
    - cf. One pattern for Event-Based Middleware (ex . Java)
- JMS Server implements JMS API
- JMS Clients connect to JMS servers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

Disadvantages of MOM



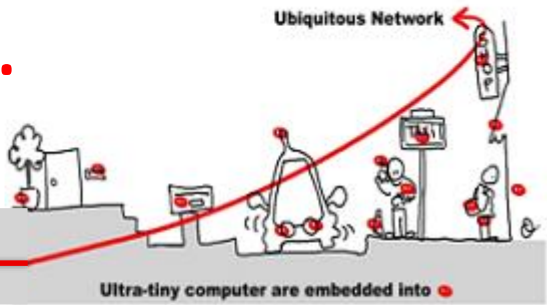
- Poor programming abstraction (but has evolved)
  - Rather low-level (cf. Packets)
  - Request/reply more difficult to achieve, but can be done
- Message formats originally unknown to middleware
  - No type checking (JMS addresses this – implementation?)
- Queue abstraction only gives one-to-one communication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

IV: Event-Based Middleware

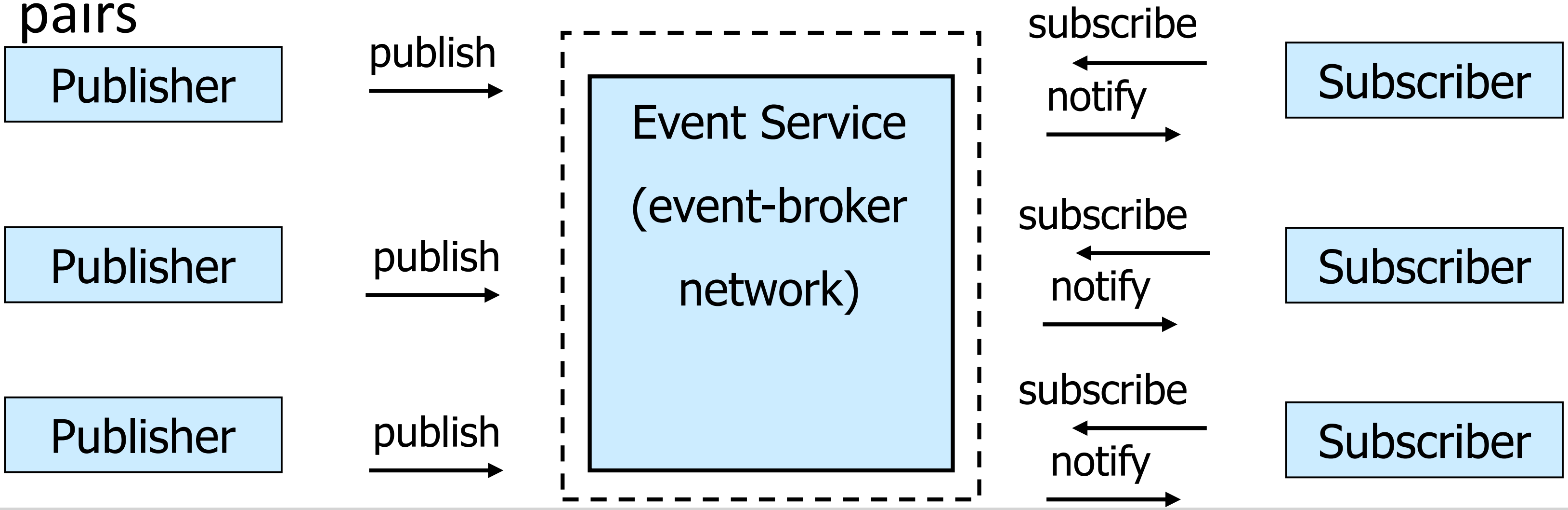


- 1 emitter – N receiver
- With broadcast communications (ex. UDP)
- With unicast communications or peer to peer (ex. TCP), multiple communications are required

IV: Event-Based Middleware, ex. Publish/Subscribe Pattern



- 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


Publisher

Publisher

Publisher

publish

publish

publish

Event Service (event-broker network)

subscribe

notify

subscribe

notify

subscribe

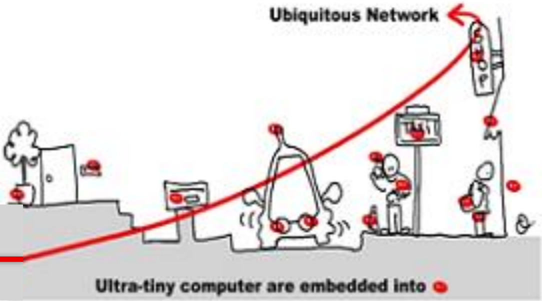
notify

Subscriber

Subscriber

Subscriber

Properties of Publish/Subscribe

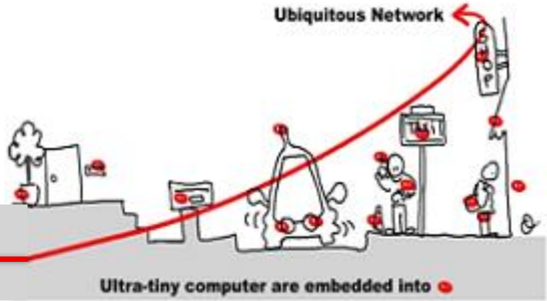


- Asynchronous communication
  - Publishers and subscribers are loosely coupled

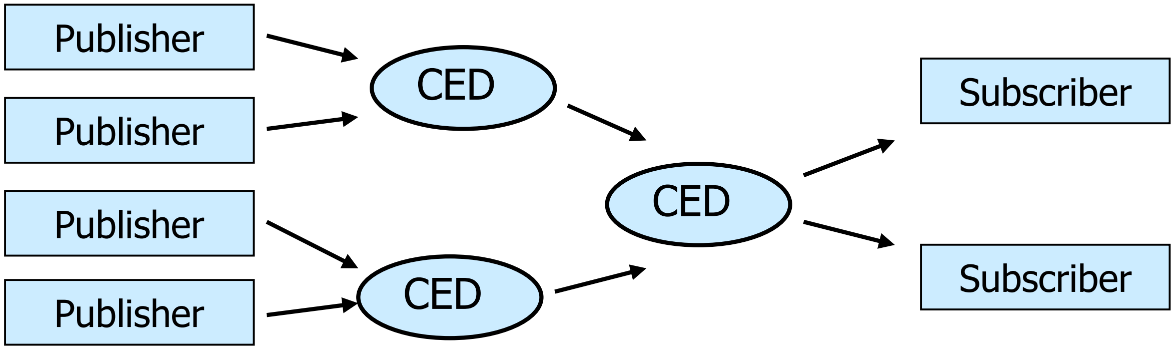
- Many-to-many interaction between pubs. and subs.
  - Scalable scheme for large-scale systems
  - Publishers do not need to know subscribers, and vice-versa
  - Dynamic join and leave of pubs, subs

- (Topic and) Content-based pub/sub very expressive
  - Filtered information delivered only to interested parties

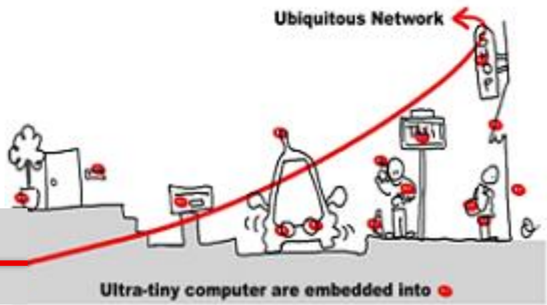
Composite Event Detection (CED)



- 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
- Alternative Implementation … (need multicast communications)



Summary

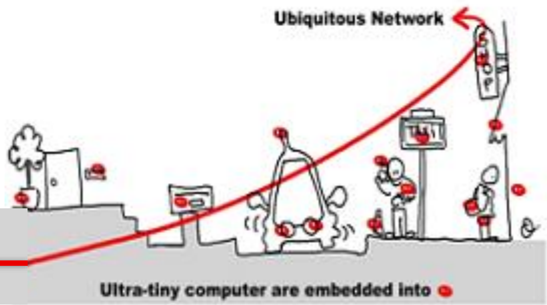


- Middleware is an important abstraction for building distributed systems

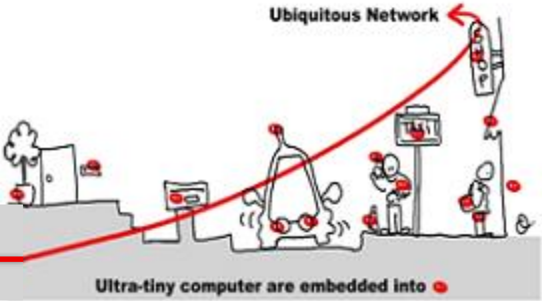
1. Remote Procedure Call
2. Object-Oriented Middleware
3. Message-Oriented Middleware
4. Event-Based Middleware

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

New Trends for Distributed Computing : SOM



- 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.

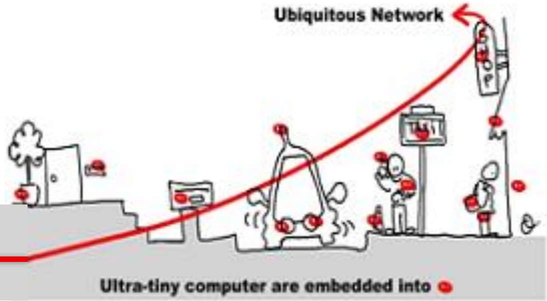


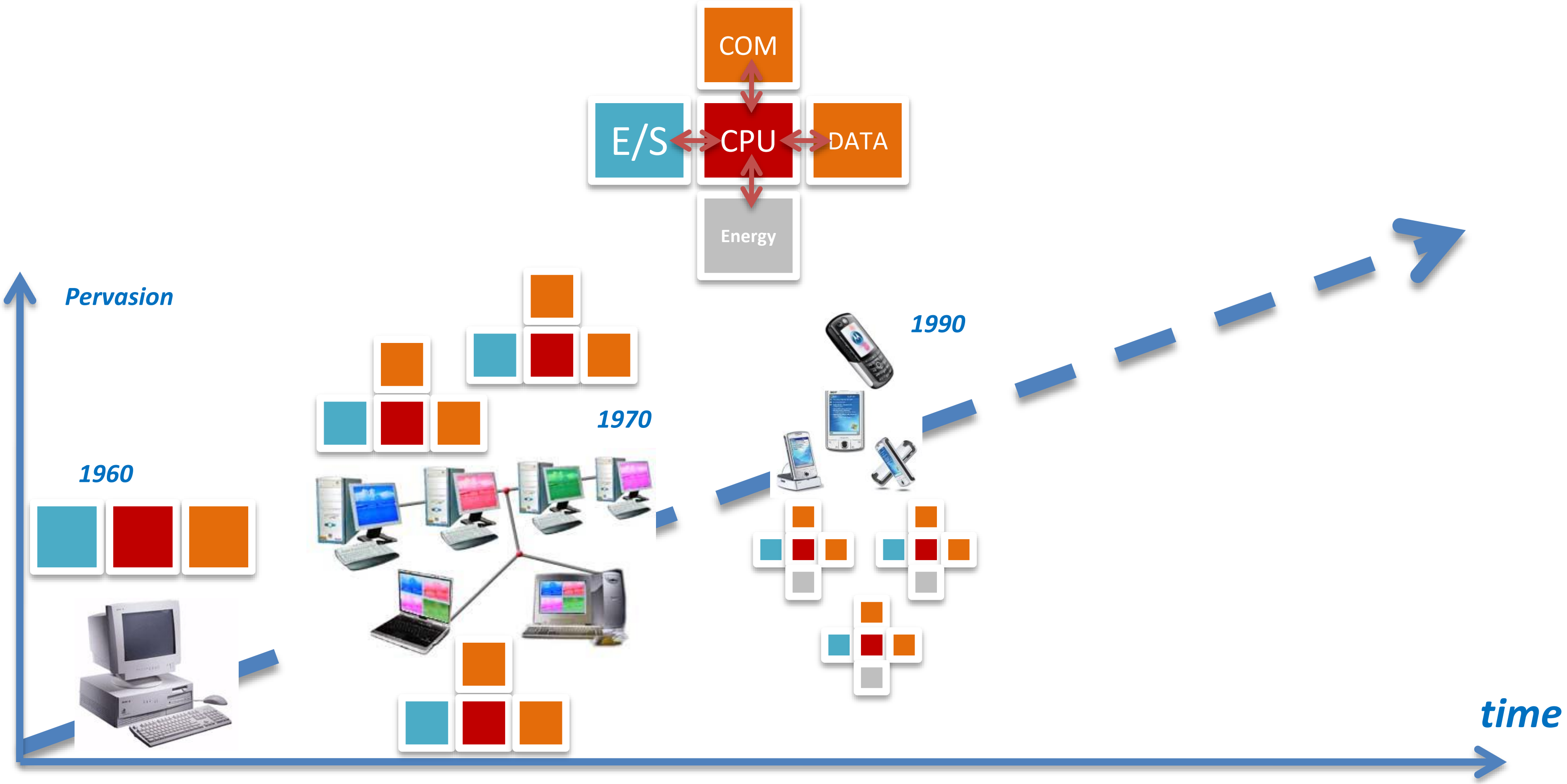
New Networked Systems and Ubiquitous/Pervasive Computing

Mobile Computers and then …

Devices, Things, Objects

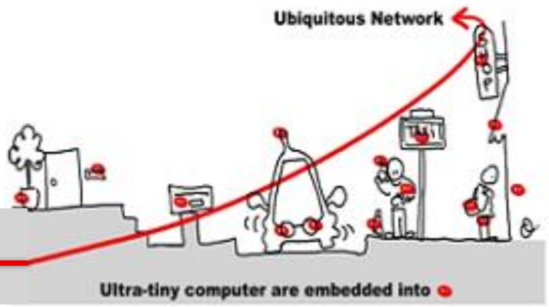
Mobile Computing

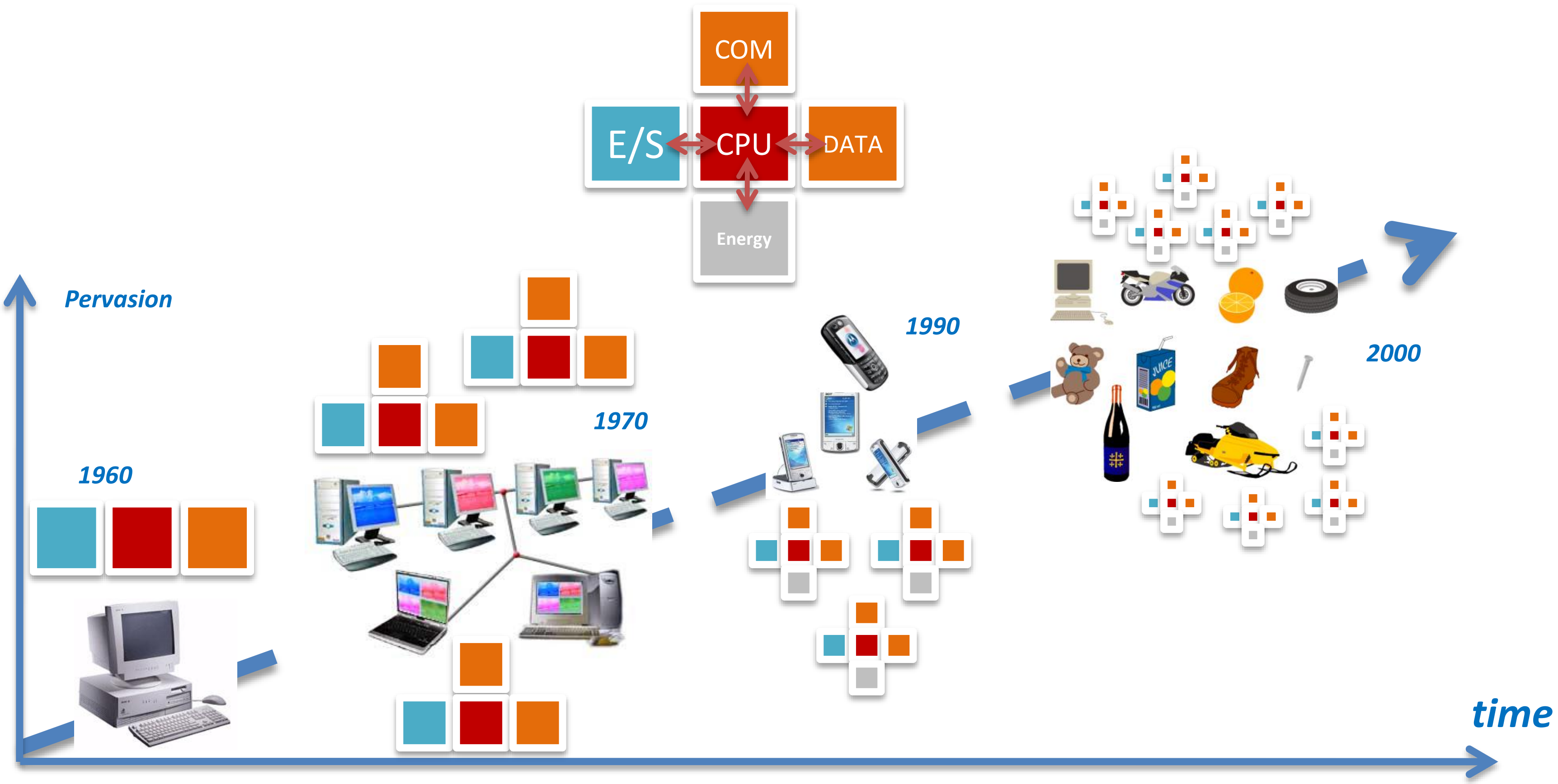




Adapté de « Nano-informatique et intelligence ambiante », JB Waldner, Hermes Science Publishing, 2007

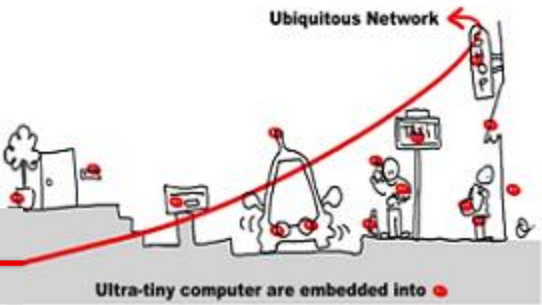
Ubiquitous Computing





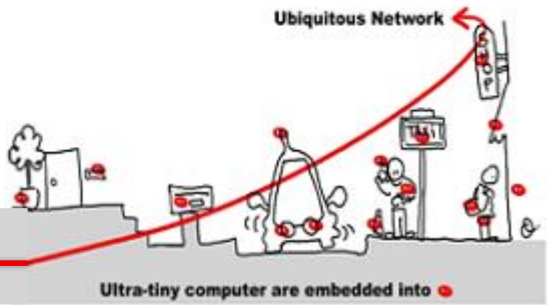
Adapté de « Nano-informatique et intelligence ambiante », JB Waldner, Hermes Science Publishing, 2007

Things are mainly based on Inputs / Outputs evolution …



- Things can be sensors
- Things can be actuators
- Things can be physically coupled sensors and actuators (heater : electrical resistance and temperature sensor)
- Things can be logically coupled sensors and actuators (electrical switch and controlled light)

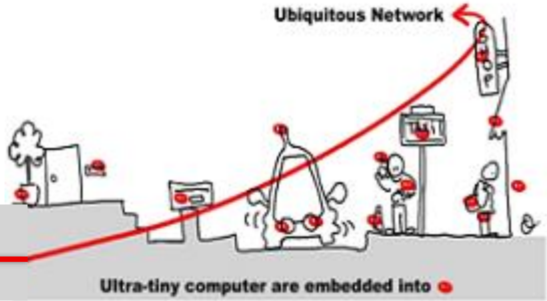
Trends *Web of Things*



- Two kind of Approches

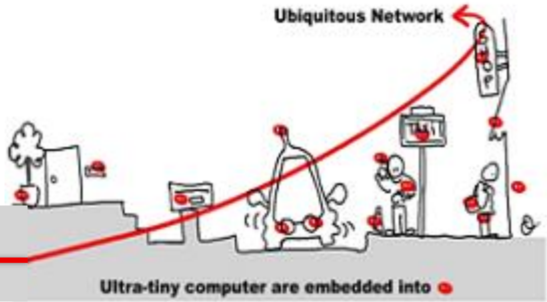
- Service oriented Architectures :
  - ROA (DAO) : Ressource or data oriented
  - SOA : Sevice oriented

- REST is an example for the first approach



Ressource Oriented Architecture

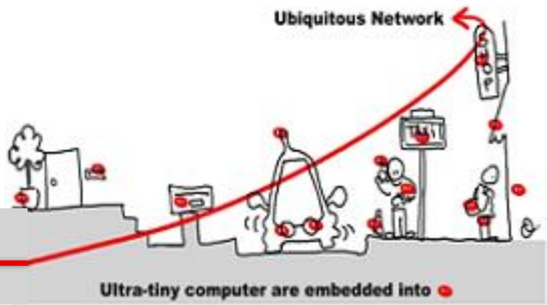
RESTful Web Services



- REpresentational State Transfer
  - Architecture inherent in all web based system since 1994, not explicitly described as an architecture until later
  - An architecture - not a set of standard
    - Web Services is both an architecture and a set of standards

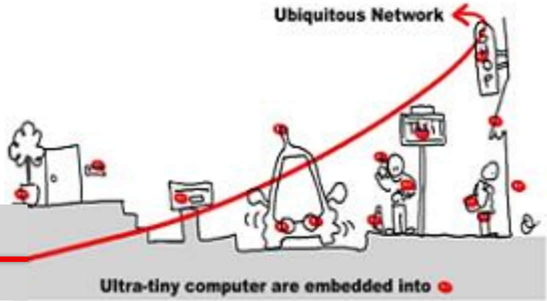
- Goal: To leverage web based standards to allow inter-application communication as simply as possible
  - Matches the 'standard' web interaction model

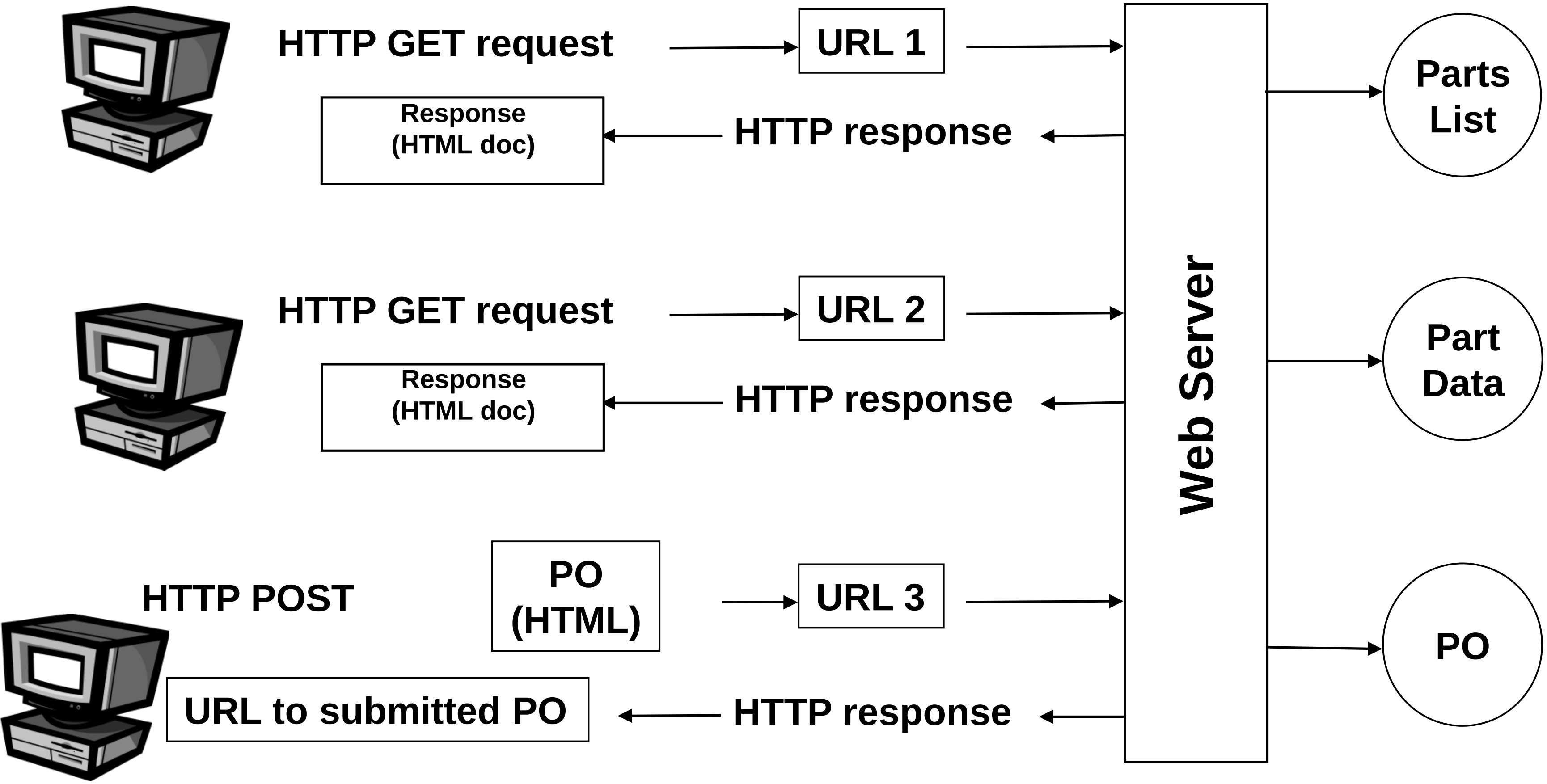
REST architecture



- Uses HTTP operations:
  - GET = "give me some info" (Retrieve)
  - POST = "here's some update info" (Update)
  - PUT = "here's some new info" (Create)
  - DELETE = "delete some info" (Delete)
- Typically exchanges XML documents
  - But supports a wide range of other internet media types
- Example of client side REST request: GET /shoppingcart/5873
  - Server must be able to correctly interpret the client request as there is no explicitly defined equivalent to an interface definition

The standard Web architecture





HTTP GET request

URL 1

Response
(HTML doc)

HTTP response

Parts
List

HTTP GET request

URL 2

Response
(HTML doc)

HTTP response

Part
Data

Web Server

HTTP POST

PO
(HTML)

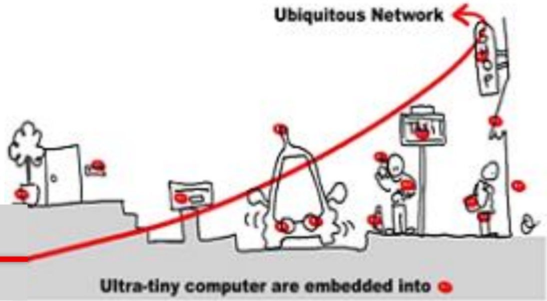
URL 3

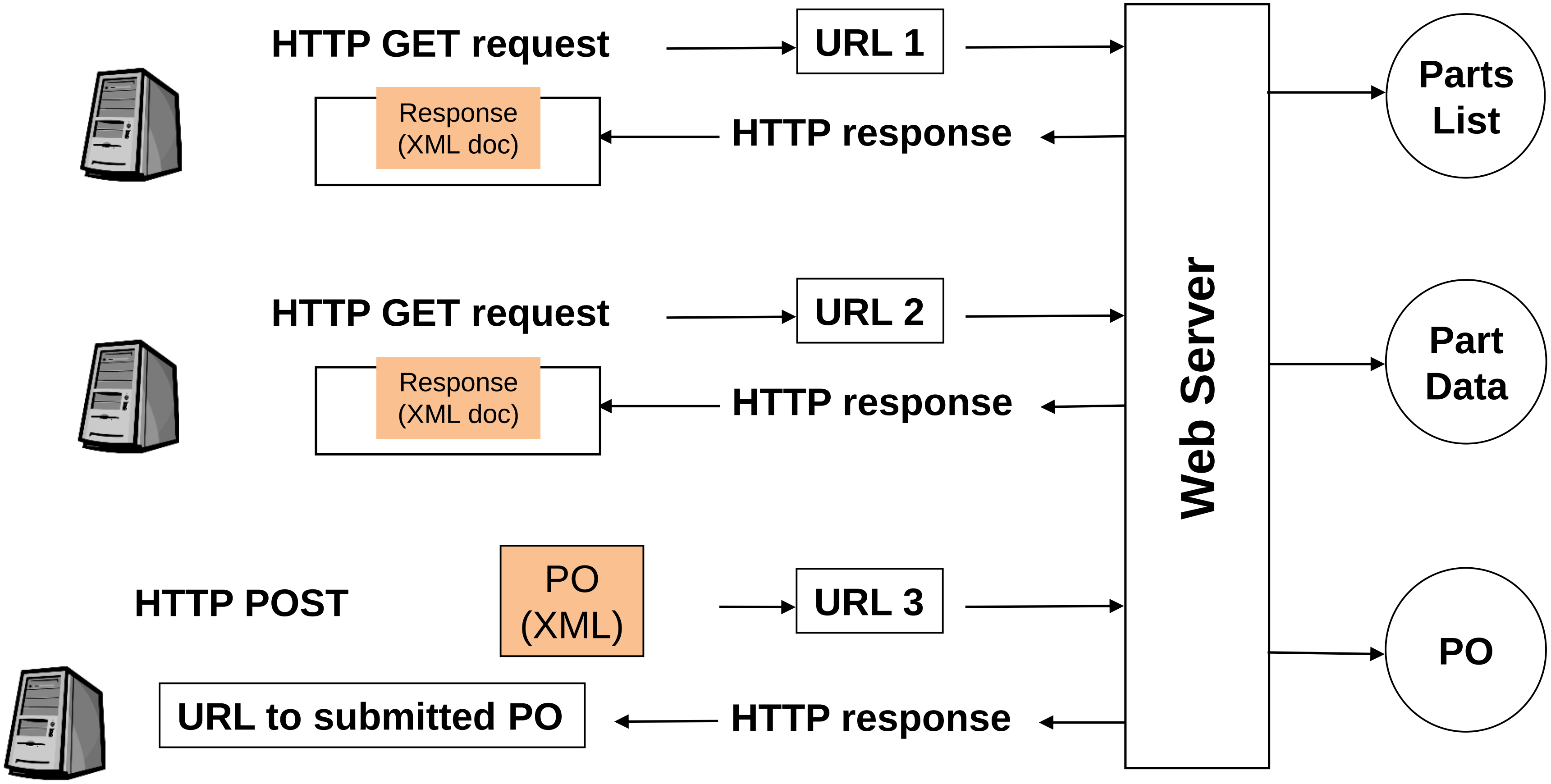
PO

URL to submitted PO

HTTP response

The RESTful architecture





HTTP GET request

URL 1

Response
(XML doc)

HTTP response

HTTP GET request

URL 2

Response
(XML doc)

HTTP response

HTTP POST

PO
(XML)

URL 3

URL to submitted PO

HTTP response

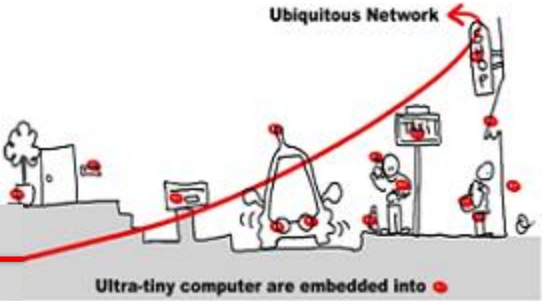
Web Server

Parts
List

Part
Data

PO

REST Architecture

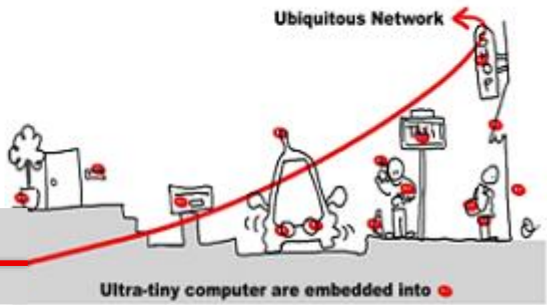


- Servers are stateless and messages can be interpreted without examining history
  - Messages are self-contained

- There is no such thing as a “service”.
  - There are just resources which are accessed through URI
    - URI = generalisation of URL

- Clients navigate through a series of steps towards a goal by following hypertext links (GET) and submitting representations (POST).

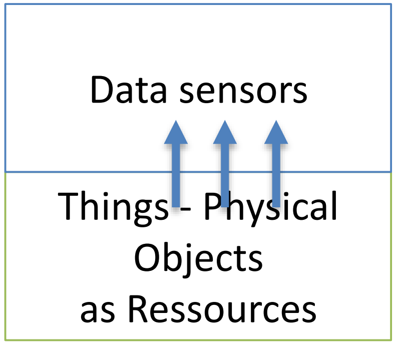
ROA and Mashup



- Mashups is “A way to create new Web applications by combining existing Web resources utilizing data and Web APIs” [Benslimane et al., 2008]

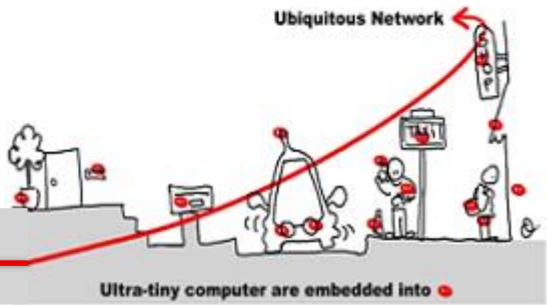
- ROA is Well-adapted for Mashups (Composite Web Applications)
- Well-adapted for Web Sensors Network (WSN)

- But lacks for non sensor device … like actuators …



Data sensors

Things - Physical Objects as Ressources

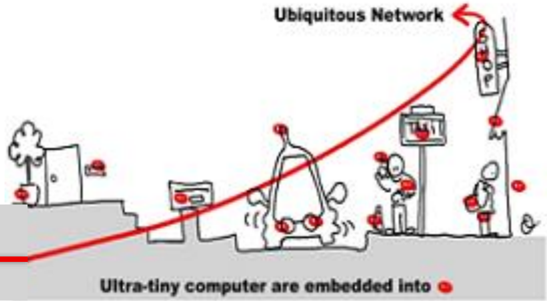


VIDEO

- Difference between Ambient Applications and Mashups for IoT

continuum4_scenario_fontainier_mi-parcours.mp4 (Ligne de commande)

REST – strong versus weak



- Pure REST should use ‘pure’ URI only
  - E.g. GET /shoppingcart/5873

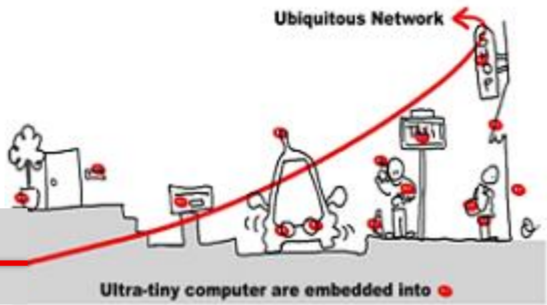
- Many REST implementations also allow parameter passing
  - E.g. GET /shoppingcart/5873?sessionID=123

- Allowing parameter passing makes REST a lot more usable but blurs the architectural principle of statelessness

- Indeed Data can be specific command like instruction code …
  - But is it the purpose ?
  - Is this not another way to rebuild a SOA stack ?

Service oriented architecture (SOAP-WS)

SOA : Service oriented Architecture

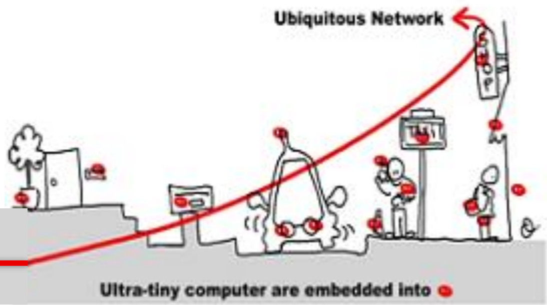


- A service provides business functions to its consumer and in ISO 19119 [ISO/TC-211] it is defined as

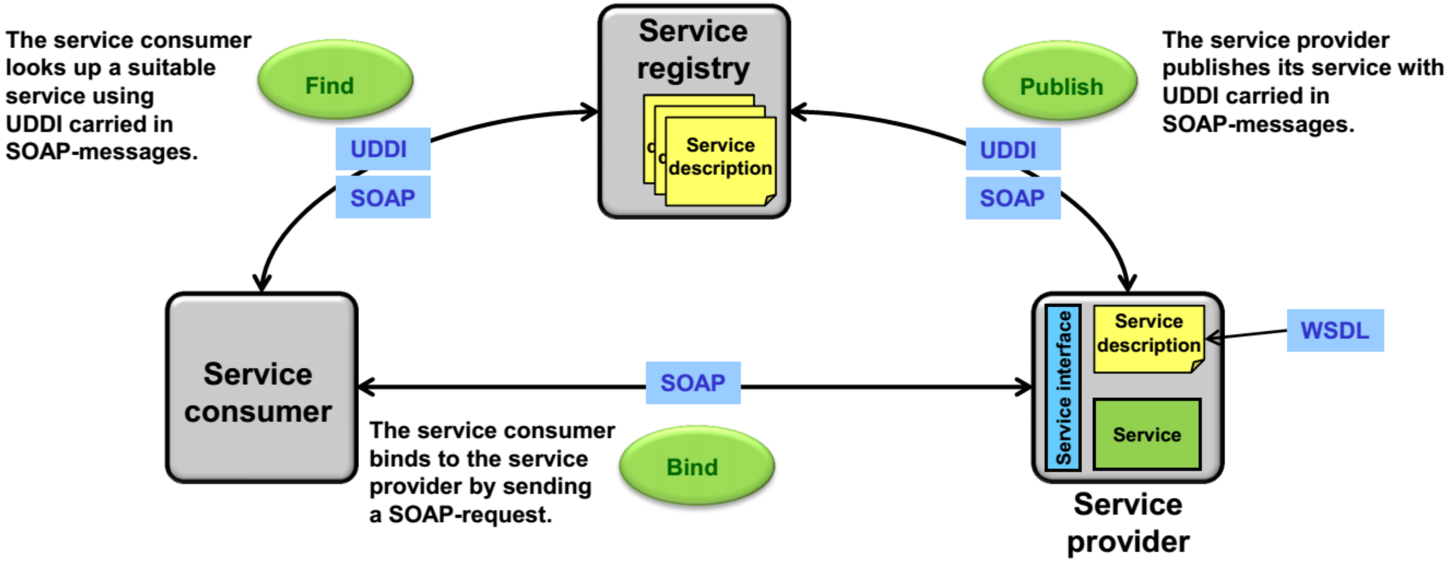
“ Distinct part of the functionality that is provided by an entity through interfaces ”.

- SOAP based Web Service, the alternative
- Also called WS-* (for * recommendations, Cf. http://www.w3.org/)

WS-*architecture more than ROA



- SOAP+WSDL+UDDI defines a general model for a web service architecture.
  - SOAP: Simple Object Access Protocol
  - WSDL: Web Service Description Language
  - UDDI: Universal Description and Discovery Protocol
  - Service consumer: User of a service
  - Service provider: Entity that implements a service (=server)
  - Service registry : Central place where available services are listed and advertised for lookup

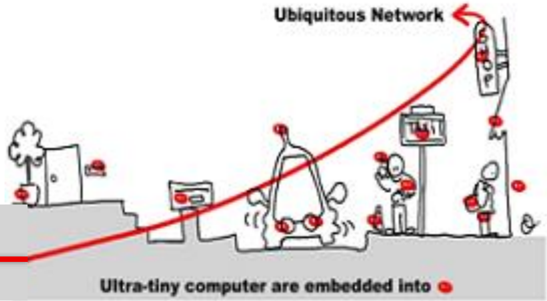


The service consumer looks up a suitable service using UDDI carried in SOAP-messages.

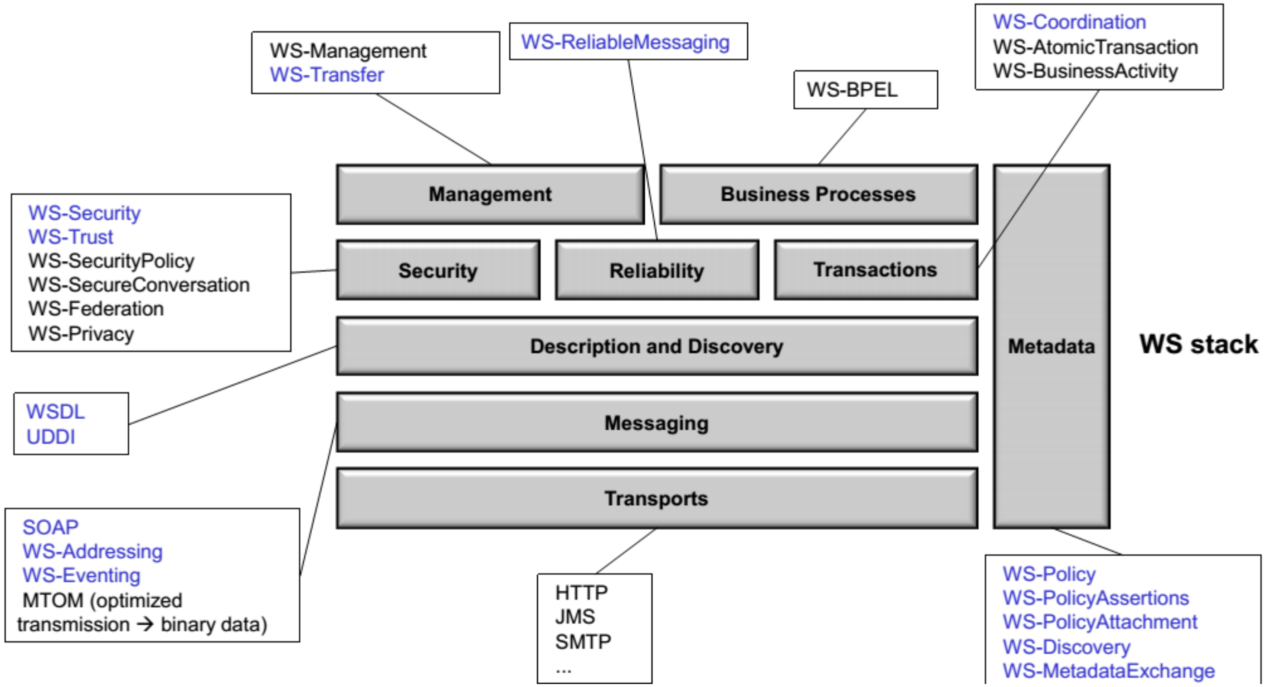
The service provider publishes its service with UDDI carried in SOAP-messages.

The service consumer binds to the service provider by sending a SOAP-request.

WS-* Models



- Stack of WS-standards
- The W3C and OASIS WS-stack provide a framework / toolbox for constructing web service architectures



WS-Management
WS-Transfer

WS-ReliableMessaging

WS-BPEL

WS-Coordination
WS-AtomicTransaction
WS-BusinessActivity

Management

Business Processes

Security

Reliability

Transactions

Description and Discovery

Messaging

Transports

Metadata

WS stack

WS-Security
WS-Trust
WS-SecurityPolicy
WS-SecureConversation
WS-Federation
WS-Privacy

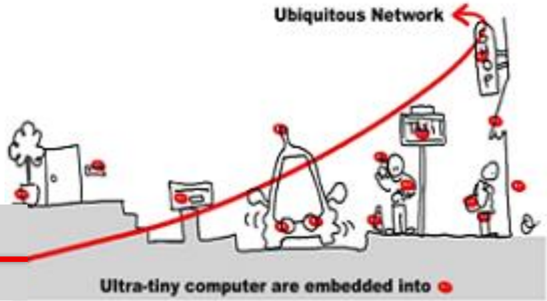
WSDL
UDDI

SOAP
WS-Addressing
WS-Eventing
MTOM (optimized transmission → binary data)

HTTP
JMS
SMTP
...

WS-Policy
WS-PolicyAssertions
WS-PolicyAttachment
WS-Discovery
WS-MetadataExchange

SOAP-WS versus REST Model



SOAP-WS:

☺ Rather complete (there is a WS-standard for almost every aspect / problem).
☺ Modular (take what you need and compose your web service architecture).
☹ Complex (too many different WS-standards with dependencies to each other, difficult to find a common base that is supported by all participants).
☹ Performance penalty due to chatty protocols with large overhead (SOAP).

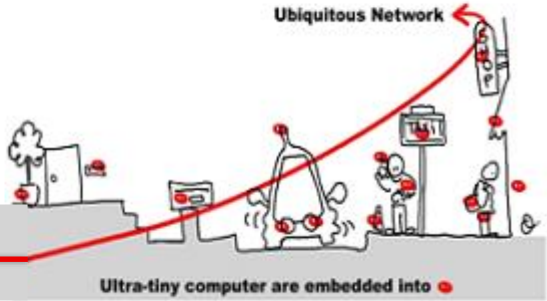
Applicability / suitability:
Enterprise SOA-architecture which requires security, orchestration, management etc.

REST-WS:

☺ Simple.
☺ Fits the bill for most applications?
☹ No standard, semantics of service mostly described in human readable form, not machine processable without description language (e.g. WADL or WSDL 2.0).
☹ Too simple (missing functionality for advanced services which require coordination etc.).

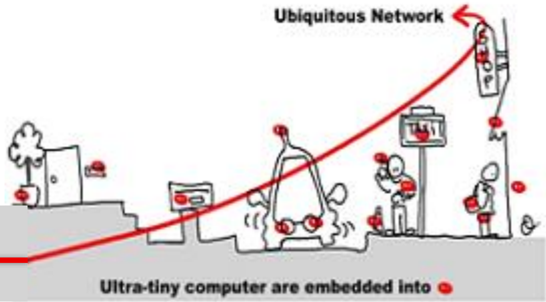
Applicability:
Simple and isolated access (read) to data / resources.

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

Interroperability… and Research Challenges

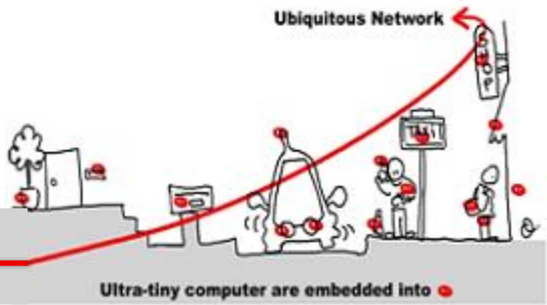


Interroperability between Technology :

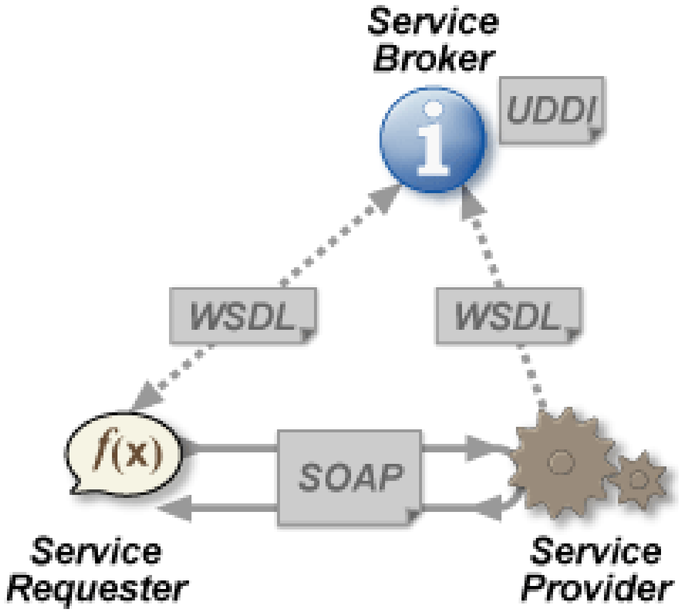
- Thanks to a common software platform
- Thanks to a common network protocol
- Research Challenge : Adpative Middleware like Emergent Middleware

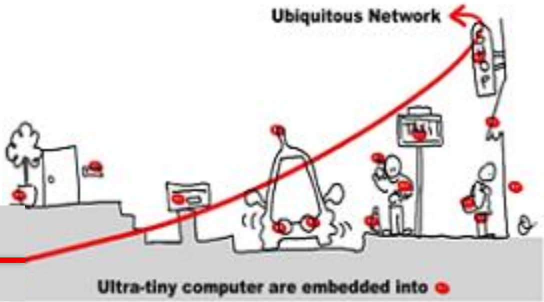
Emergent Middleware *by Paul Grace, Gordon S. Blair and Valerie Issarny , [5,6]*

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 (good or bad thing ?)
  - anonymity of communicating entities
- Support for Device
  - data values from sensors
  - lightweight software
- And other requirements for Ubiquitous Computing …

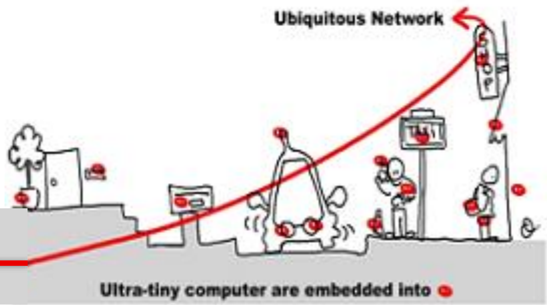

Service
Broker
UDDI
WSDL
WSDL
f(x)
SOAP
Service
Requester
Service
Provider



Web Service for Device …

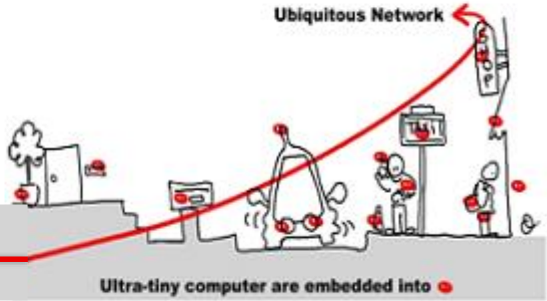
UPnP, DPWS ….

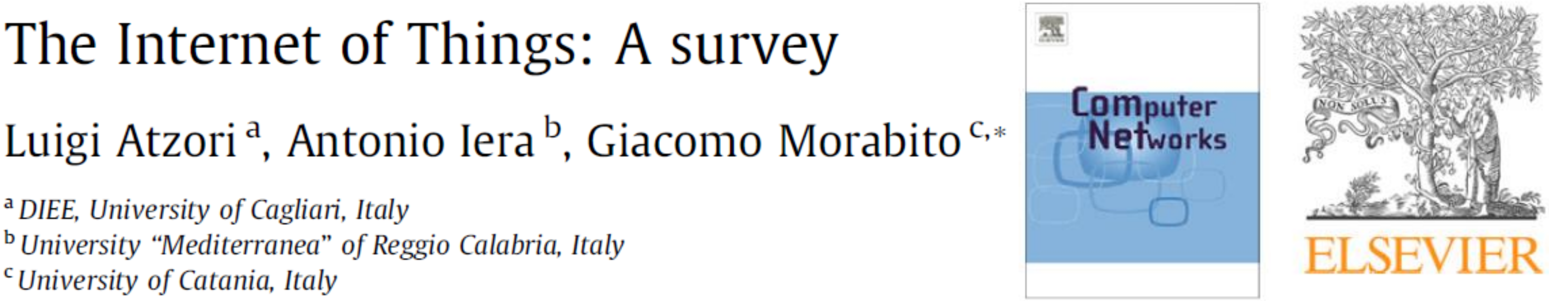
Related material



- Reading List:
  - http://aspire.surrey.ac.uk/lists/35640FC8-892D-E309-E66C-F07C3D9BCB28.html
- ETSI, Machine to Machine Communications
  - http://www.etsi.org/technologies-clusters/technologies/m2m
- Machine-to-Machine Communications, OECD Library,
  - http://www.oecd-ilibrary.org/science-and-technology/machine-to-machine-communications_5k9gsh2gp043-en
- W3C Semantic Sensor Networks
  - http://www.w3.org/2005/Incubator/ssn/XGR-ssn-20110628/

References




The Internet of Things: A survey

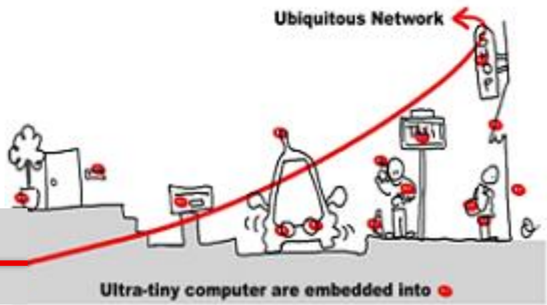
Luigi Atzori[a], Antonio Iera[b], Giacomo Morabito[c,*]

[a] DIEE, University of Cagliari, Italy
[b] University "Mediterranea" of Reggio Calabria, Italy
[c] University of Catania, Italy

Computer Networks
ELSEVIER

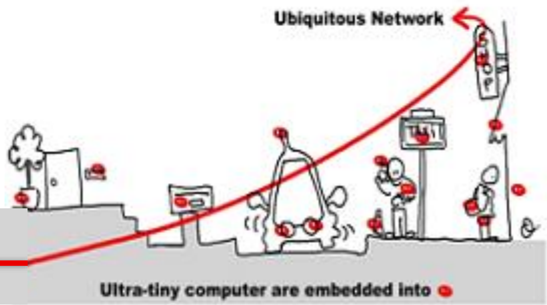
- From the Internet of Things to the Web of Things: Resource-oriented Architecture and Best Practices, Dominique Guinard, Vlad Trifa, Friedemann Mattern, Erik Wilde , Architecting the Internet of Things, 2011, pp 97-129, Editors: Dieter Uckelmann, Mark Harrison, Florian Michahelles, ISBN: 978-3-642-19156-5 (Print) 978-3-642-19157-2 (Online) - http://www.vs.inf.ethz.ch/publ/papers/dguinard-fromth-2010.pdf
- Services Mashups: The New Generation of Web Applications, Issue No.05 - IEEE Internet Computing - September/October (2008 vol.12), pp: 13-15, Djamal Benslimane , Lyon University, Schahram Dustdar , Vienna University of Technology, Amit Sheth , Wright State University http://www.infosys.tuwien.ac.at/staff/sd/papers/ServicesMashups_IC.pdf

References



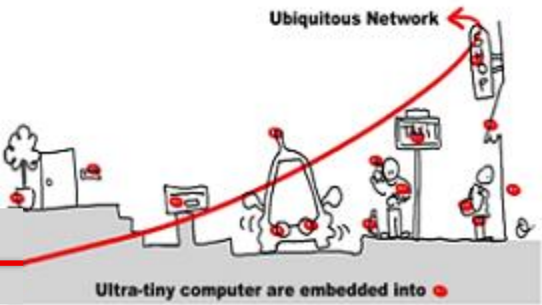
- M. Satyanarayanan. Pervasive computing : Vision and challenges. IEEE [see also IEEE Wireless Communications] Personal Communications, 8(4) :10–17, 2001. https://www.cs.cmu.edu/~aura/docdir/pcs01.pdf
- ISO/TC-211, ISO 19119 Geographic information - Services. 2005.
- Fielding, R.T., Architectural Styles and the Design of Network-based Software Architectures, Ph.D. dissertation, in University of California, Irvine, 2000. https://www.ics.uci.edu/~fielding/pubs/dissertation/fielding_dissertation.pdf
- 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
- 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

Main event : WoT , Web of Things (Ubicomp conference)…



- W3C
- Regular Workshop WoT (Web of Things)
  - During UbiComp Conference (also Pervasive and ISWC since 2013)
  - 4th in 2013
- http://www.w3.org/community/wot/
- The aim of the Web of Things Community Group (CG) is to accelerate the adoption of Web technologies as a basis for enabling services for the combination of the Internet of Things with rich descriptions of things and the context in which they are used.