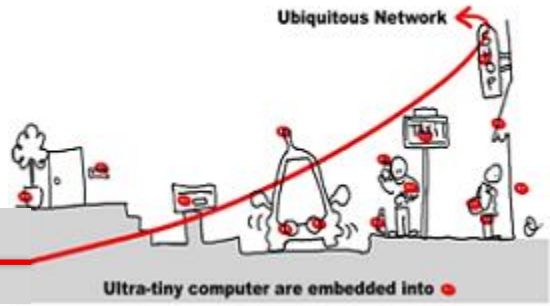
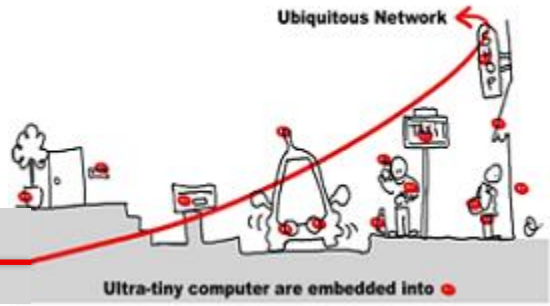


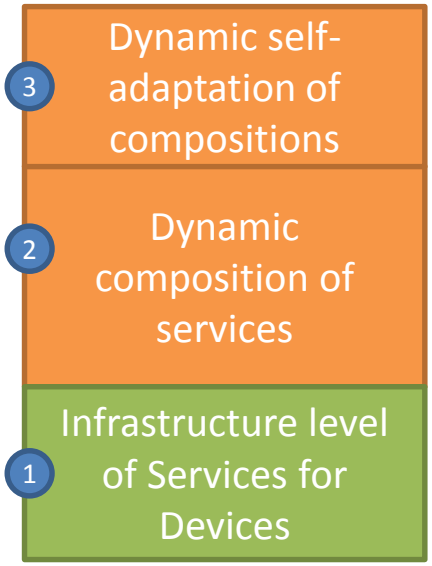
Lecture 6 : Introduction to self-adaptation

« When the infrastructure changes at runtime, the composition must dynamically adapt itself »

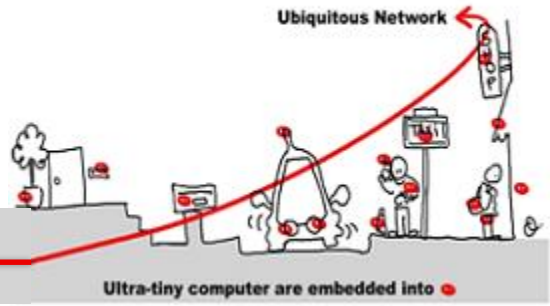


Self adaptation : How and Where ?

1. Infrastructure : based on Web services for Device
2. Composition : based on CBSE
3. Self-Adaptation



Infrastructural Level : Web Services for Devices

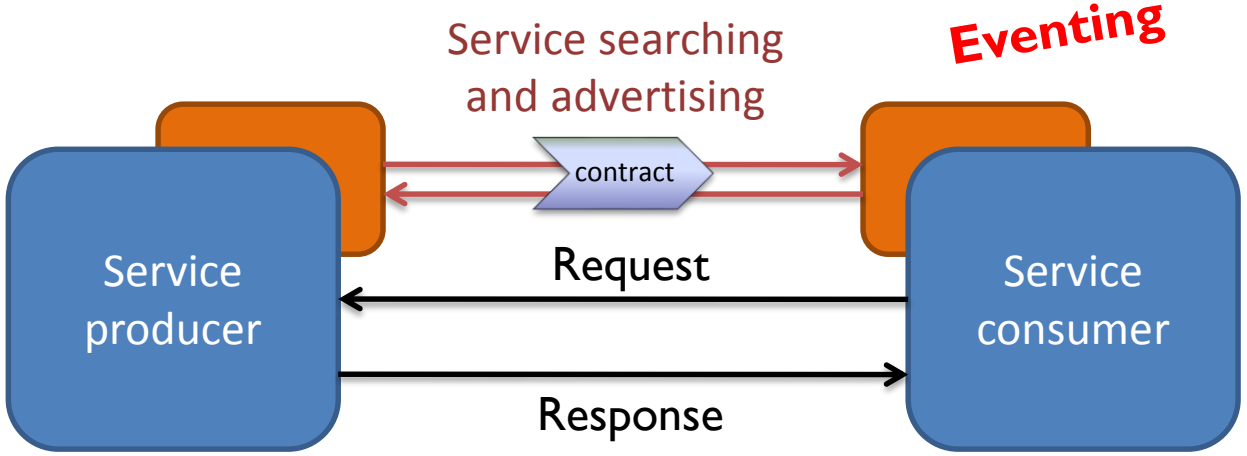


- Requirements for WComp Infrastructure
  - New ways of interacting: Eventing

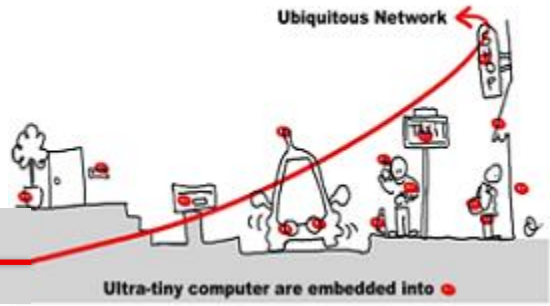
Decentralized and Contextual discovery

Managing Apperance and Disapperance

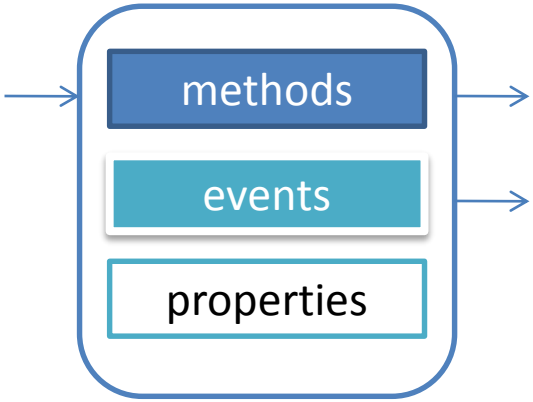
Eventing

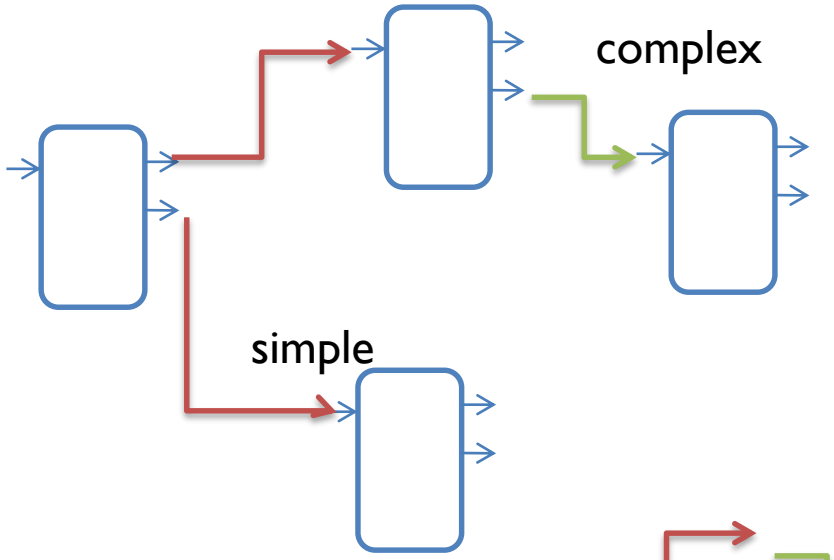


Dynamic Composition level : LCA model for Orchestration

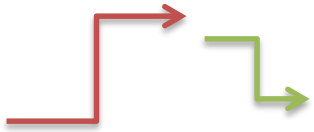


- LCA components

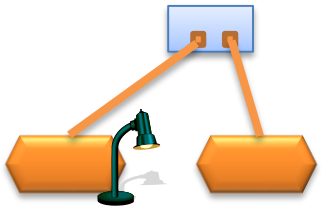




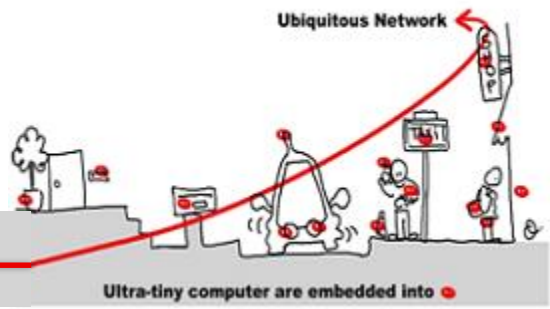
- Simple and complex Event based Connector



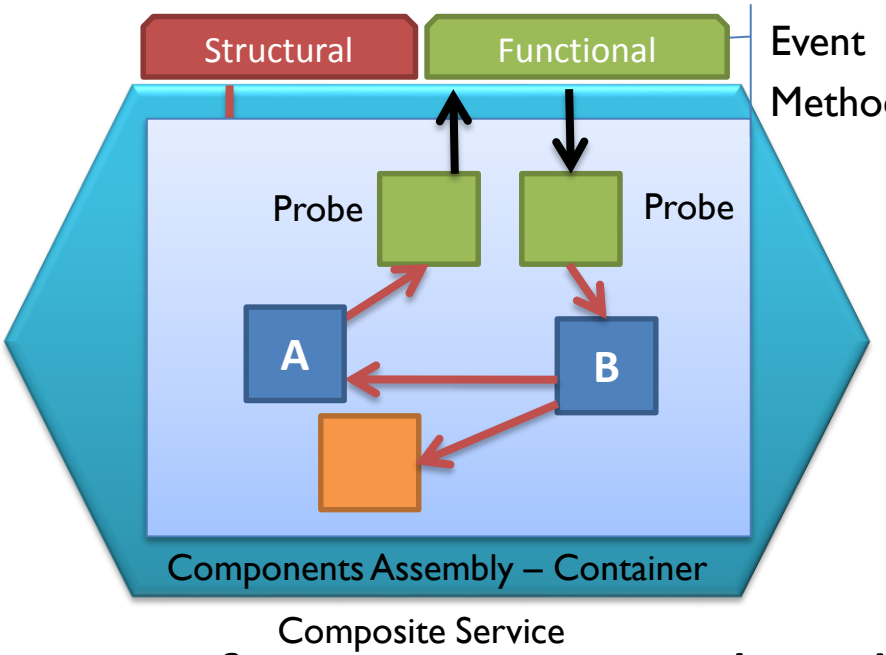
- Proxy Components



Dynamic Composition level: SLCA Model for Choregraphy



- New components as Probe components

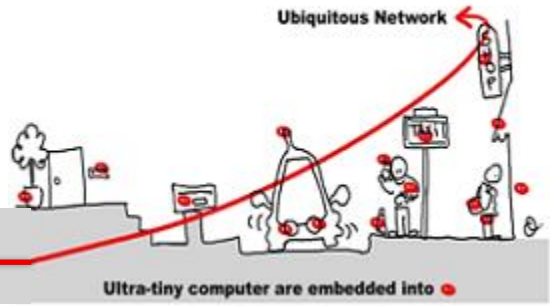


Probe emitter :
Local event → Service Event

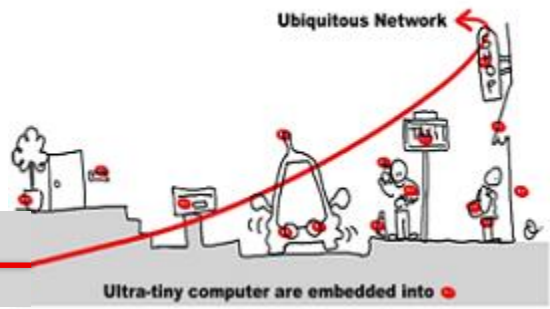
Probe receiver :
Service Method Invocation →
Local Event (Request)

- Two interfaces : structural and functional

Tutorial



- How to create a Composite Web Service for Device (CWSD)
- i.e. a composition between Web Services for Devices is not only a Service oriented Application but an other Web Service for Device.
- Thus
  - Functional interface of the CWSD can be create and modify at runtime using probe components
  - Structural interface of the CWSD allows to modify the internal assembly from the outside.

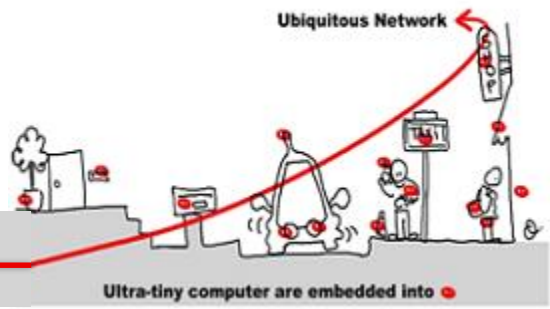


Lecture 6 : Introduction to Self Adaptive Middleware for Web of Things

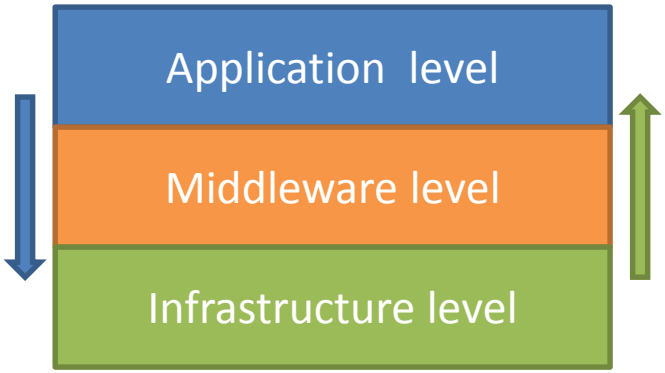
Jean-Yves Tigli

University of Nice – Sophia Antipolis / Polytech'Nice - Sophia
Members of the RAINBOW research team (I3S)

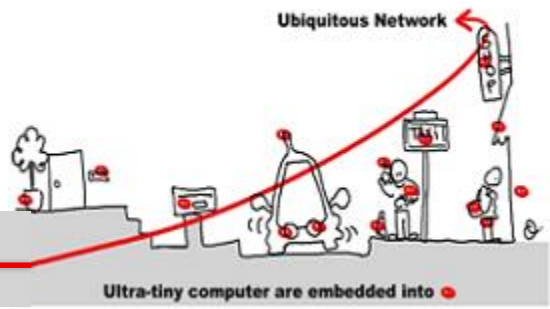
Challenge 1 : Real world interaction



- Ubiquitous Computing applications are continuously interacting with a real world, partly unknown at design time and, always changing at runtime in uncountable manner
- We witness to a kind of inversion in the classical software methodology where the software applications levels are much more stable and stationary than the software infrastructure level.

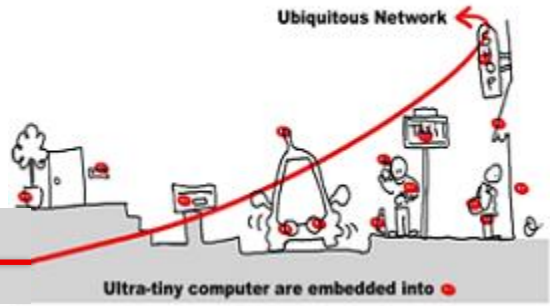


Challenge 2 : Multi-Domain Adaptation



- Ubiquitous Middleware must continuously adapt at runtime, application requirements to changing computing environment (due to mobility) in multiple domains :
  - HMI,
  - Power,
  - Network bandwidth,
  - Devices availability, …

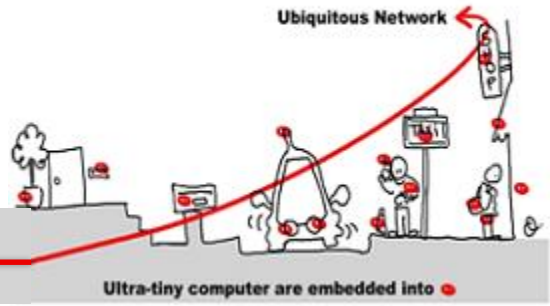
Challenge 3 : Reactive Adaptation



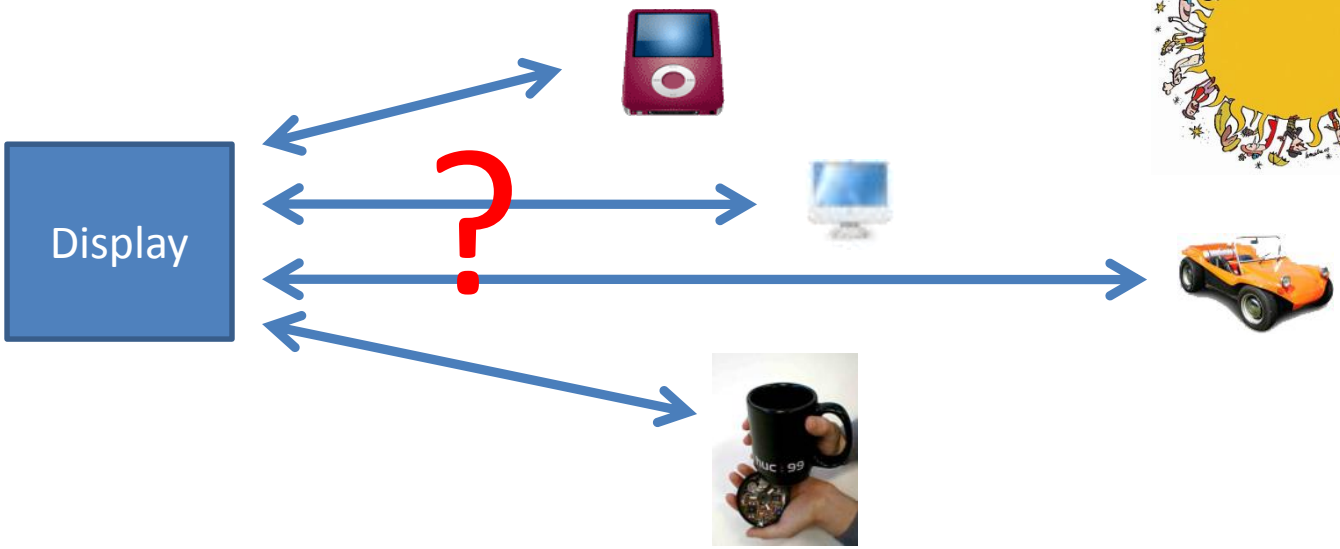
- Reactive adaptation is defined as the ability for the Ubiquitous applications to perceive the environment and adapt to changes in that environment in a timely fashion.

- Ubiquitous Middleware must provide reactive adaptation mecanism to changing operational environment.

Challenge 4 : Semantic Adaptation

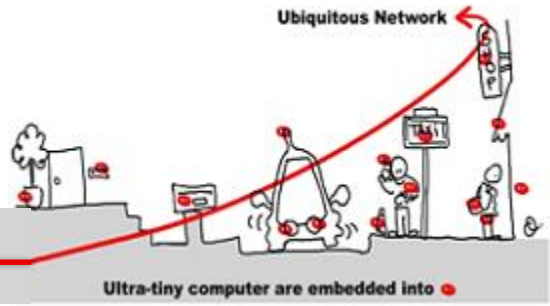


- Ubiquitous Middleware must match at run-time the current operational environment and application requirements.



Can match with ?

Autonomy is the Key



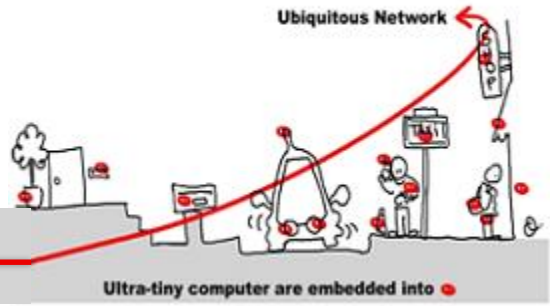
- Example :
  - Ambiant Service Continuity
  - Context aware Computing
  - Autonomic Computing

- Which manage adaptation
- End / Expert user can't react instead of the system
- From software adaptation to automatic or self-adaptation …

- Manual Adaptation
- Automatic Adaptation

David Garlan, Bradley Schmerl, and Shang-Wen Cheng, "Software Architecture-Based Self-Adaptation" in Autonomic Computing and Networking, M.K. Denko et al. (eds.),DOI 10.1007/978-0-387-89828-52,C Springer Science+Business Media, LLC 2009

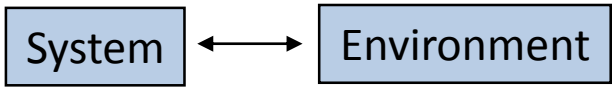
Self-adaptive System Definition

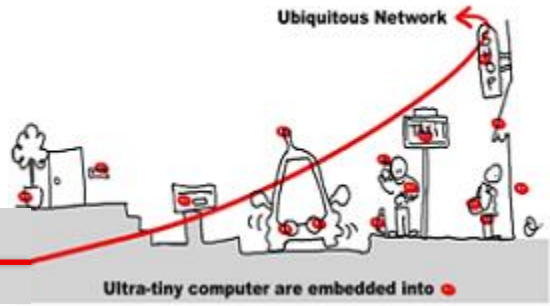


By self adaptive we mean systems and components that configure themselves and dynamically adapt to changing environments with minimal human participation.

Many systems have some degree of self-adaptiveness, but the abilities vary:

- static systems: parameter adaptation
- dynamic systems: compositional adaptation



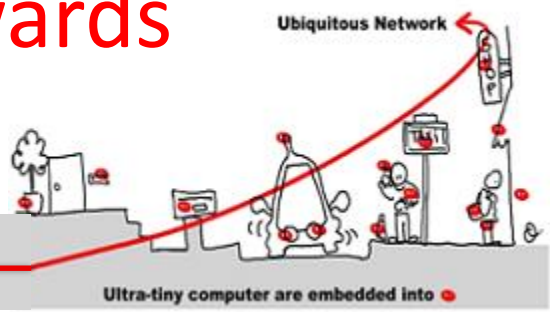


SOFTWARE SELF-ADAPTATION FOR AUTONOMOUS SYSTEMS

Different Points of view …

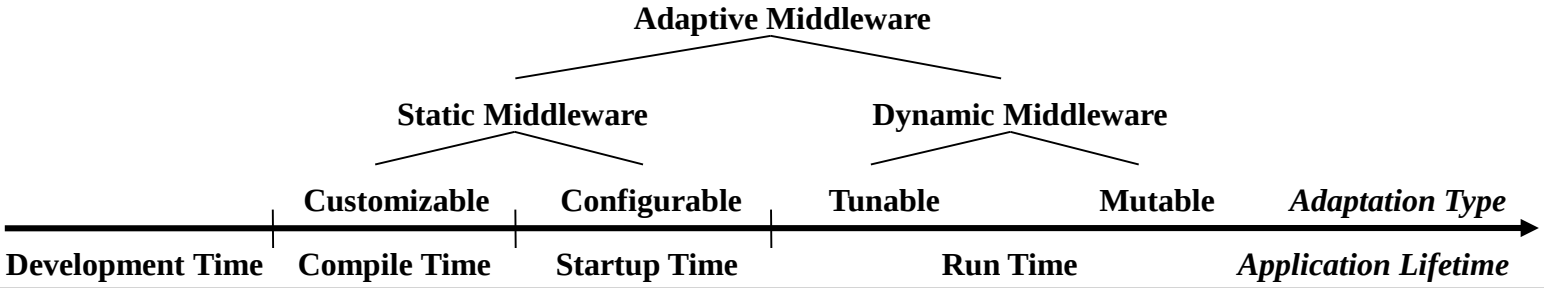
CANAL, Carlos, MURILLO, Juan Manuel, POIZAT, Pascal, *et al.* Software Adaptation. *L'objet*, 2006, vol. 12, no 1, p. 9-31.

Reconfiguration (Design Time) towards Dynamic Adaptation (Runtime)

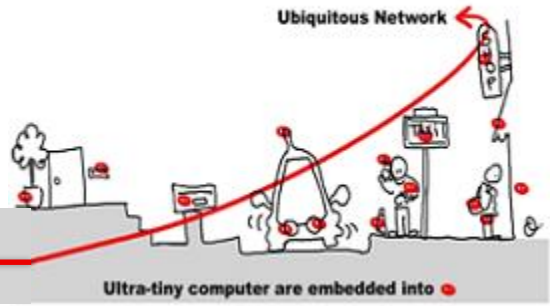


- Static Middleware
  - Customizable Middleware
    - Enables developers to compile (and link) customized versions of applications.
  - Configurable Middleware
    - Enables administrators to configure the middleware after compile time.

- Dynamic Middleware
  - Tunable Middleware
    - Enables administrators to fine-tune applications during run time.
  - Mutable Middleware
    - Enables administrators to dynamically adapt applications at run time.



General Description



Abstract service to maintain

Service Description Analysis (1)

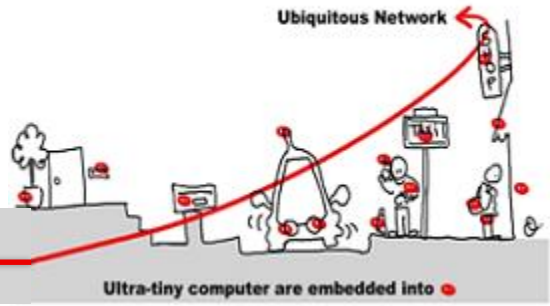
Reasoning / plan adaptation computing (3)

Adaptation / plan execution (2)

Service oriented Application

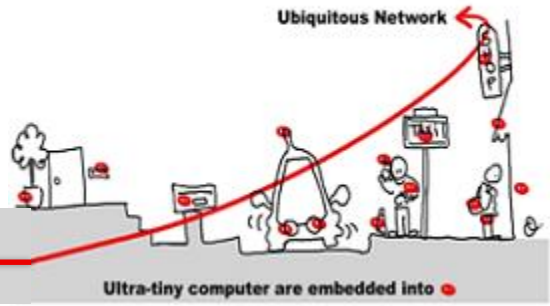
Service oriented Infrastructure

Sevices orientes Adaptation : Levels of Heterogeneity (1)

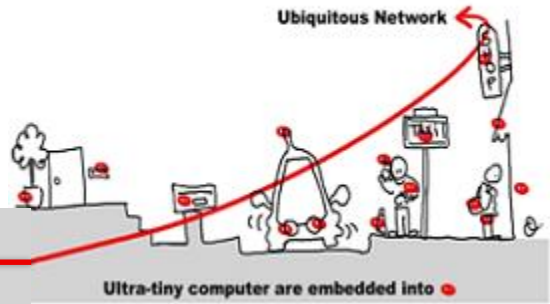


- We can distinguish between several levels of Heterogeneity, and accordingly of Service interface description :
  - Signature level
  - Behavioral level
  - Semantic level
  - Service level or QoS

Adaptivity classes (2)



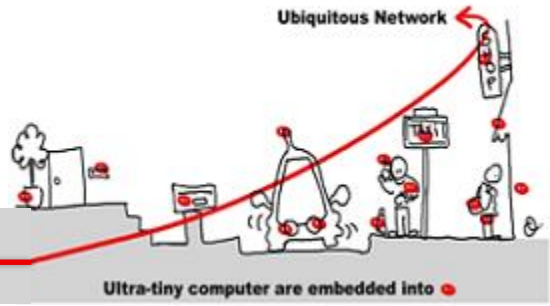
- Parameter adaptation: changing values without changing components or algorithms.
- Compositional adaptation:
  - Structural – changing parts and part structure
  - Behavioral – changing behavior/types and algorithms



Some Key Paradigms and Taxonomies for Adaptation (3)

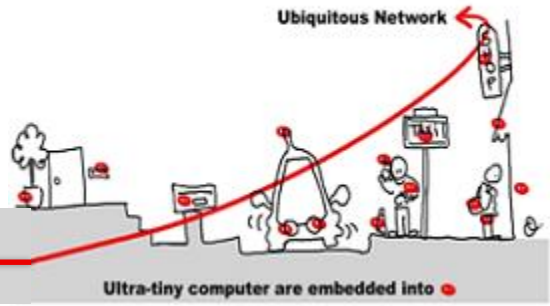
Computational Reflection
Component-Based Design
Aspect-Oriented Programming
Software Design Patterns

Some middleware Paradigms for Adaptation



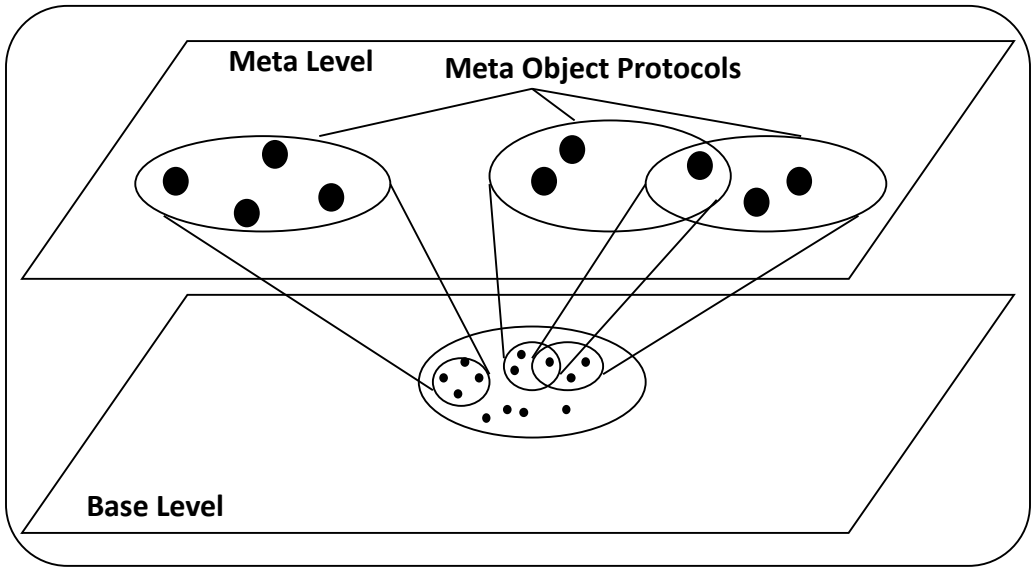
- Reconfiguration (Design Time) to Dynamic Adaptation (Runtime)
- Computational Reflection
- Policy-based adaptation
- Aspect-Oriented Programming

Computational Reflection



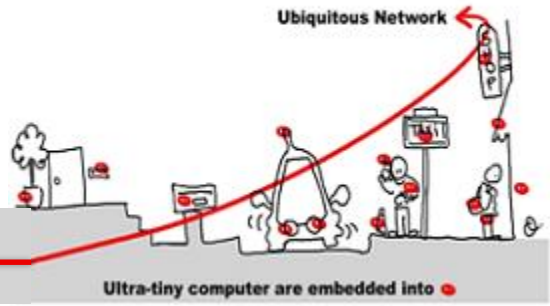
- The ability of a program to reason about, and possibly alter, its own behavior.
- Enables a system to “open up” its implementation details for such analysis without revealing the unnecessary parts or compromising portability.
- Terminology

- Base-level
- Meta-level
- MOP
- Casually connected
- Per-ORB, per-class, per-object, and per-interface reflection



Relationship between meta-level and base-level objects.

Policy-based adaptation



Policy Rules :

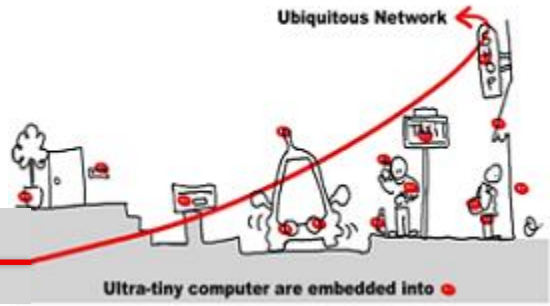
Particular conditions -> matching rules

Middleware :

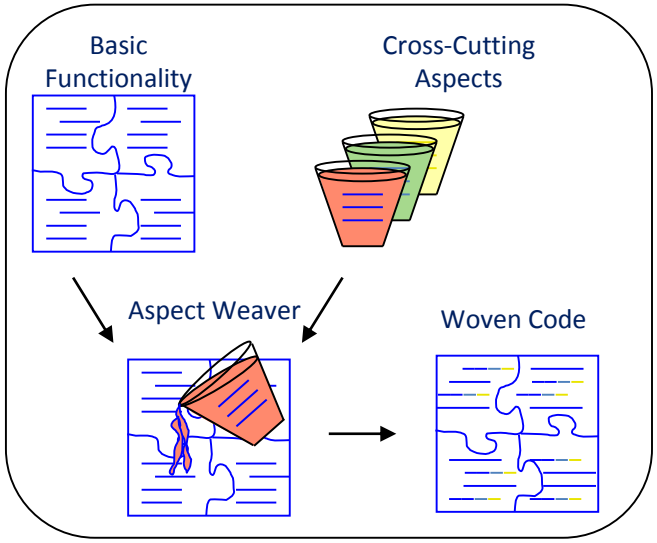
selected script -> behavior modification

- Policy rules are often context based
- Example : ECA (Event-Condition-Action) rules
  - The event part specifies the context change that triggers the invocation of the rules
  - The condition part tests if this context change is satisfied
  - Which causes the description of the adaptation (action) to be carried out

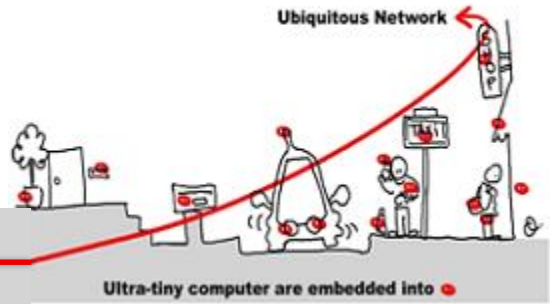
Aspect-Oriented Programming



- Complex programs are composed of different intervened cross-cutting concerns.
- Cross-cutting concerns:
  - Properties or areas of interest such as QoS, energy consumption, fault tolerance, and security.
- Terminology
  - Aspect
  - Basic Functionality
  - Aspect Language
  - Aspect Weaver
    - Static
    - Dynamic
  - Woven Code



AOP in action.



NEXT LECTURE :
Aspects of Assemblies Approach

For structural self-adaptation

Questions ?