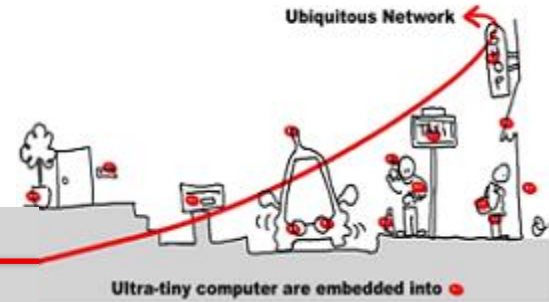


Middleware for Internet of Things



Lecture 3 : Web of Things and Services Composition

Lecturer : Ass. Prof. Jean-Yves Tigli

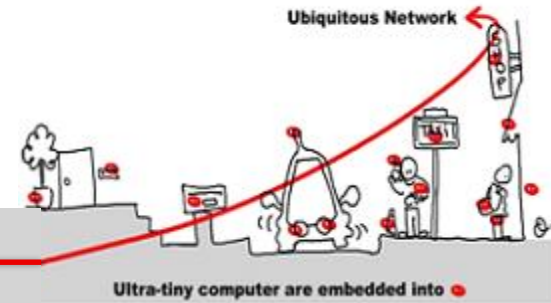
<http://www.tigli.fr>

at Polytech of Nice - Sophia Antipolis University

[Email : tigli@polytech.unice.fr](mailto:tigli@polytech.unice.fr)

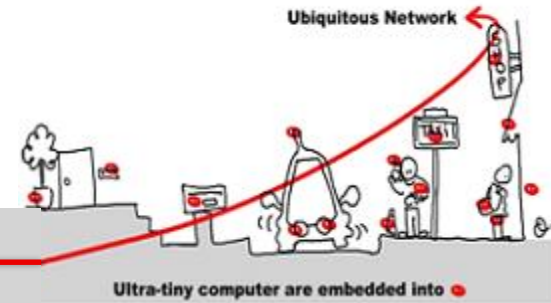


Service Composition



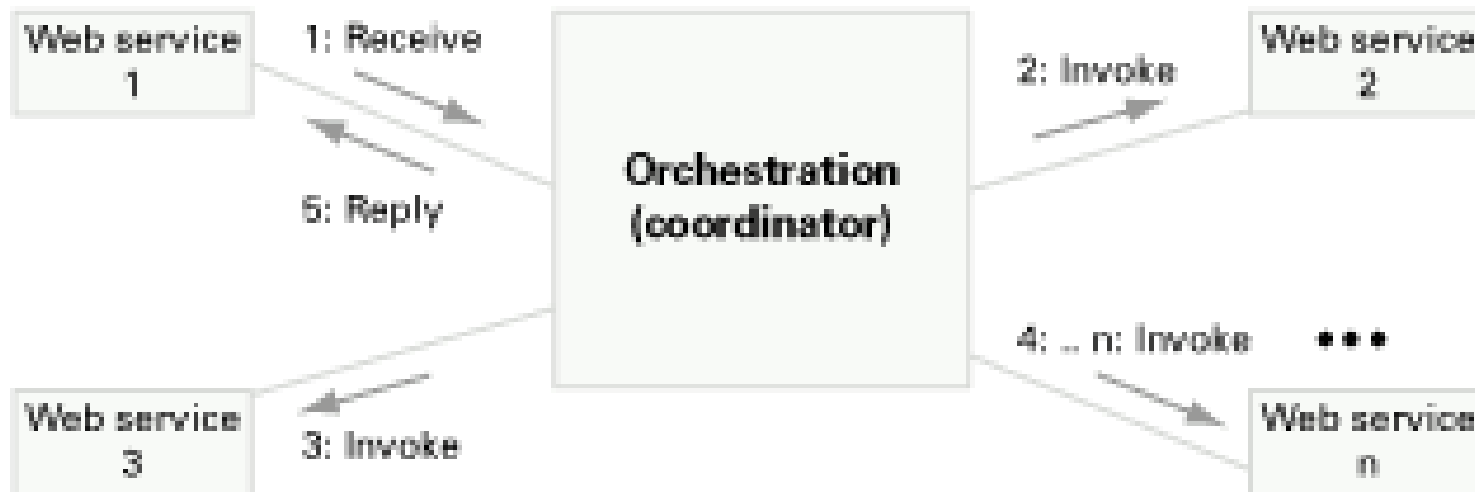
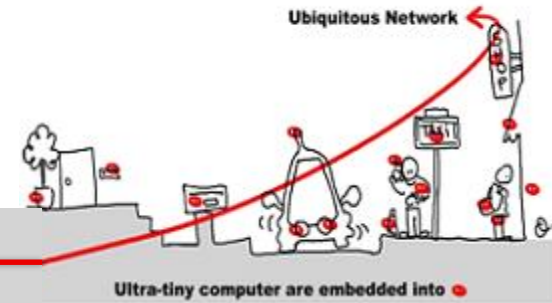
- Problem: more than one service might be needed to achieve a given objective
 - All such services need to interact seamlessly to achieve the objective
- Composite Web Services
 - Individual components implemented by different services and located at different locations
 - Execute in different contexts and containers
 - Need to interact to achieve an objective
- Benefits
 - Services can be reused
 - Access to high-level complex services

Services Composition

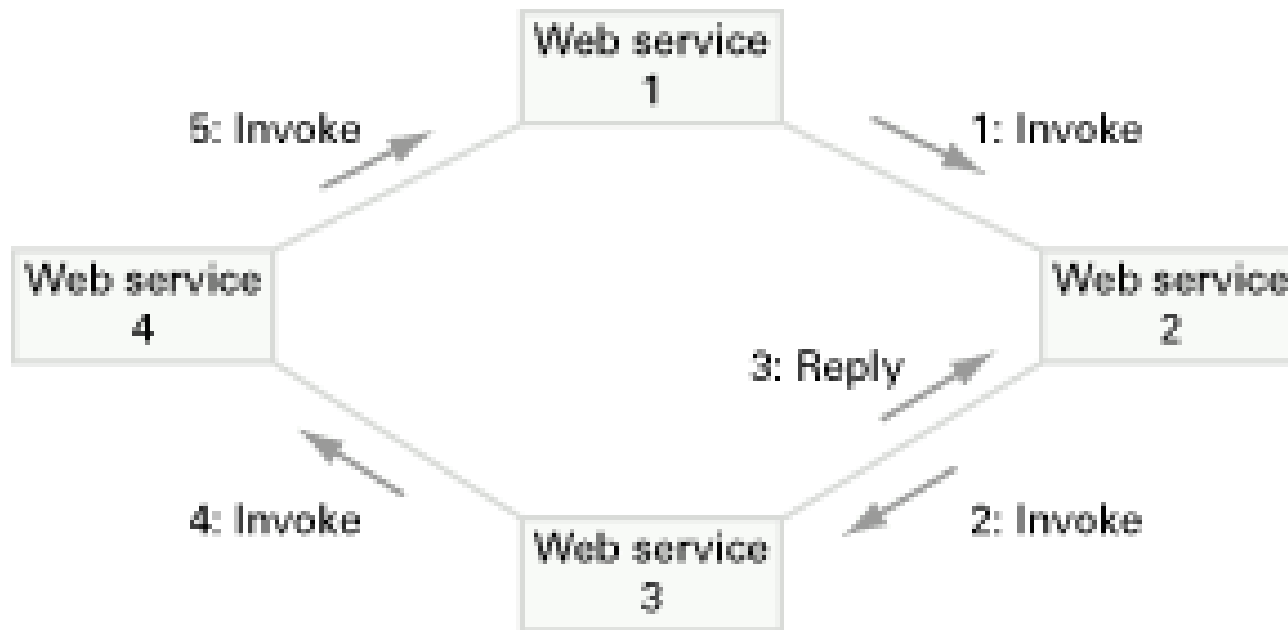
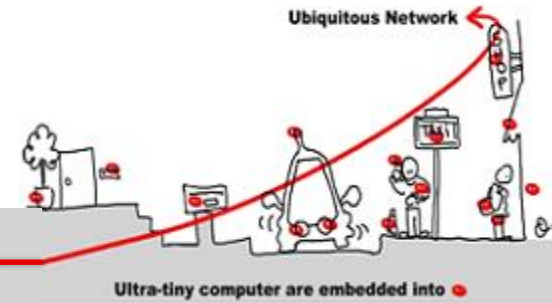


- Web services can be combined in two ways:
 - Orchestration
 - Choreography

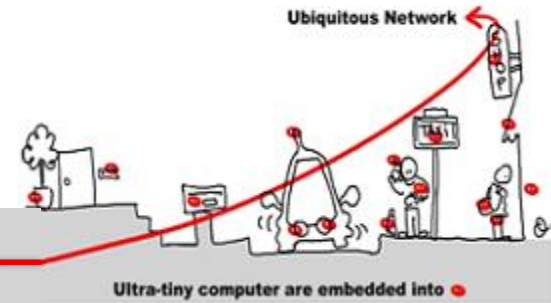
Orchestration (contd.)



Choreography (contd.)

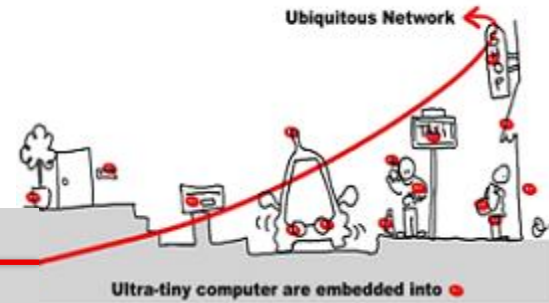


Service Composition



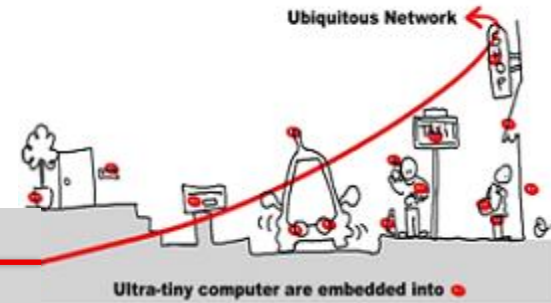
Different Approaches

- Ad-Hoc : Mashup Static composition
 - By hand
 - BPEL4WS
- Language based (control flow) :
 - Ex : BPEL4WS
- Others for Web Service for Device :
 - Event Driven (close to Data Flow but react to event appearance)
 - Ex. : Event Driven Component based Model : LCA and SLCA (Wcomp)



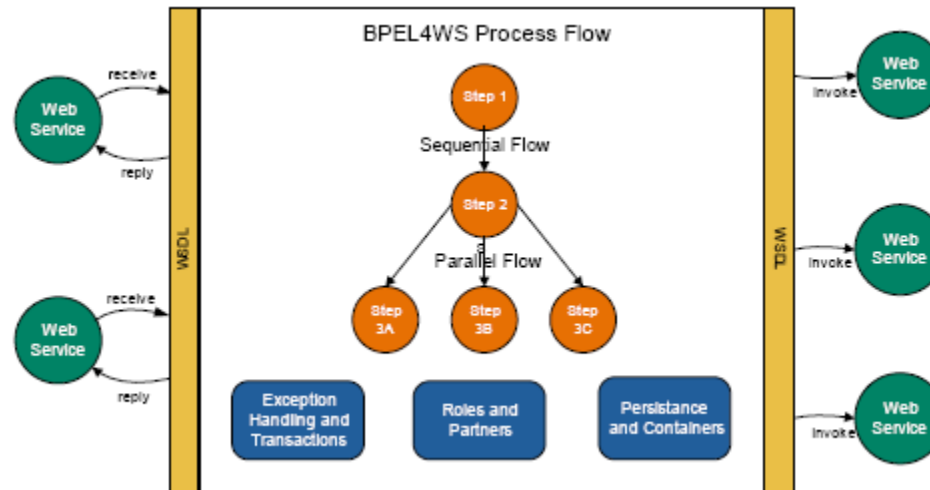
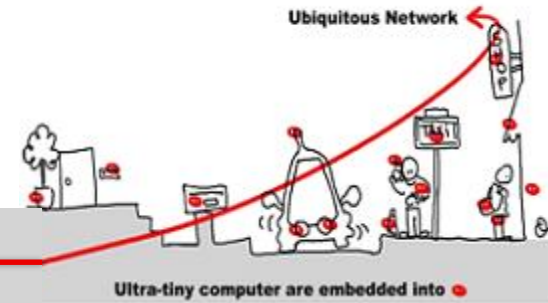
Example : a language for orchestration, BPEL

BPEL - Overview

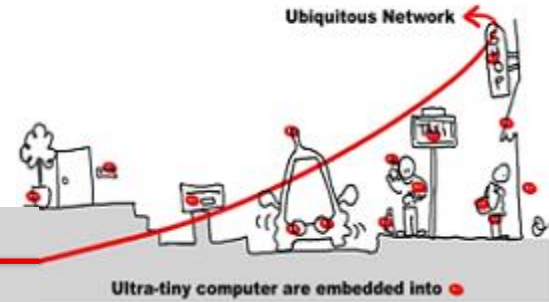


- Use Web Services Standard as a base
 - Every BPEL is exposed as a web service using WSDL. And the WSDL describes the public entry and exit points of the process
 - Interacts through WSDL interfaces with external web services
 - WSDL data types are used to describe information flow within the BPEL process

BPEL - Process Overview

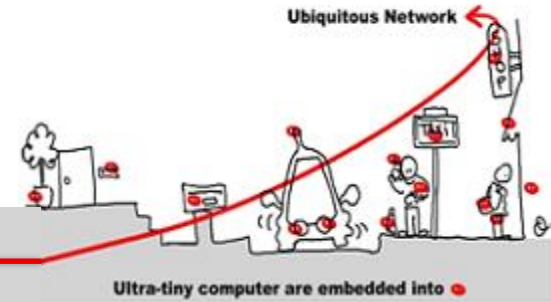


BPEL - Activities



- Basic Activities:
 - Interacts with external services
 - <invoke>, <receive>, and <reply>
- Structured Activities:
 - Internal process control flow
 - sequential flow, conditional branching, looping, and etc.
-

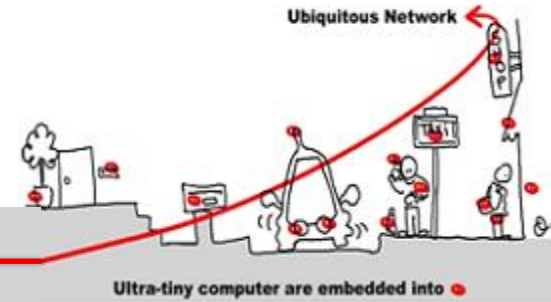
BPEL - Containers and Partners



- Containers
 - Data exchanges in the message flow
 - e.g. WSDL messageType
- Partners
 - Any services that the process invokes OR any services that the invokes the process

```
<partners>
  <partner name="buyer" ... myRole="agent"/>
  <partner name="supplier" ... myRole="requestor" partnerRole="supplier"/>
</partners>
<containers>
  <container name="request" messageType="tns:orderRequest"/>
  <container name="response" messageType="tns:orderResponse"/>
</containers>
```

BPEL - Code



- A sequence

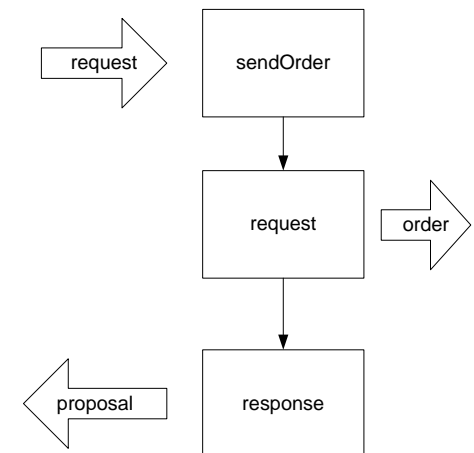
```
<sequence>
```

```
  <receive partner="buyer" ...  
  operation="sendOrder" container="request"/>
```

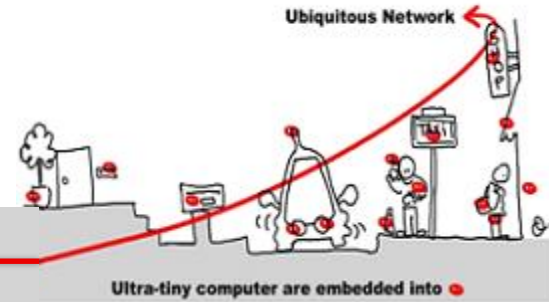
```
  <invoke partner="supplier" ...  
  operation="request" container="order"/>
```

```
  <reply partner="buyer" ... operation="response"  
  container="proposal"/>
```

```
</sequence>
```

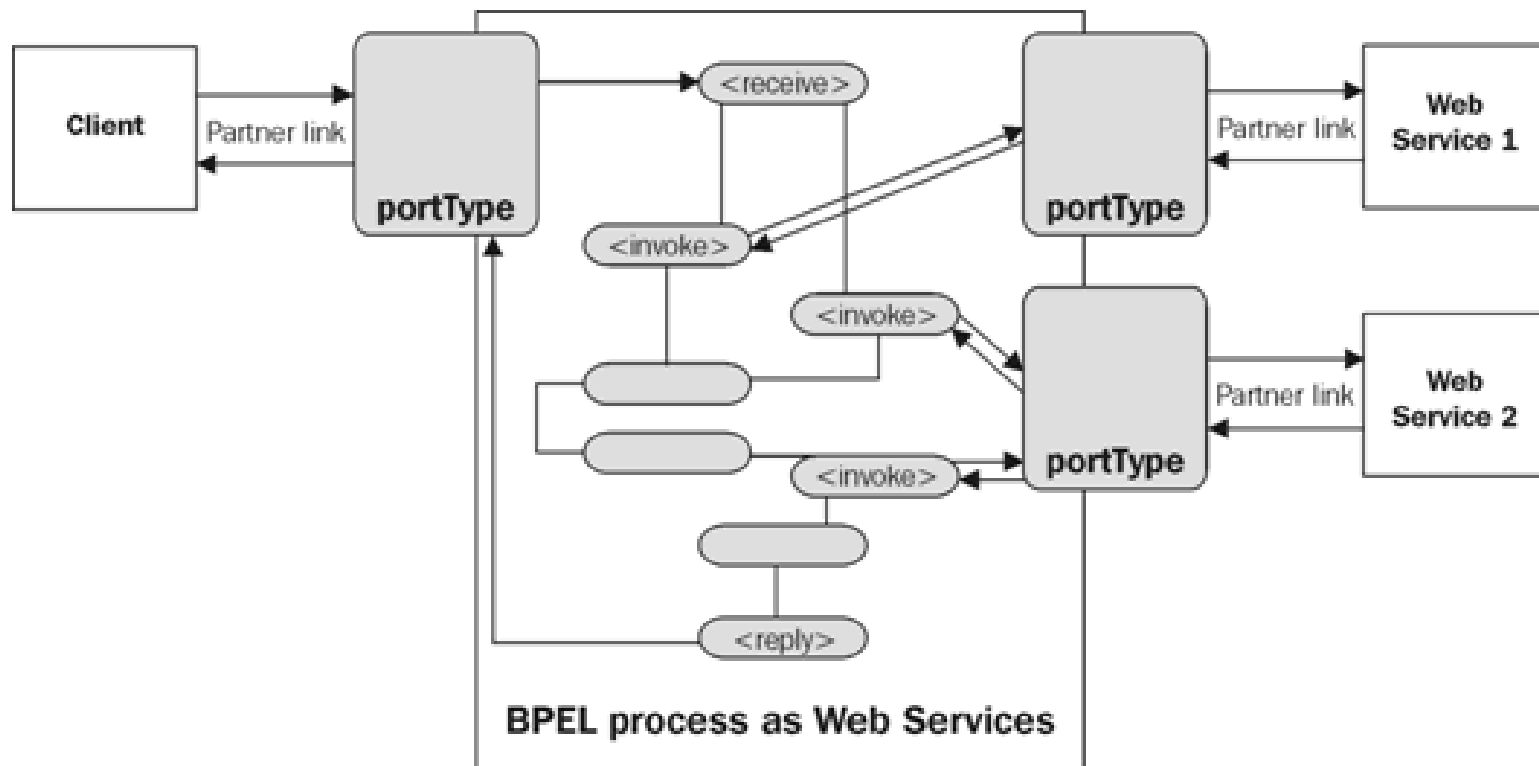
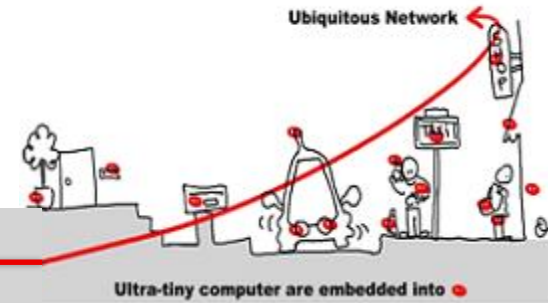


BPEL - Others

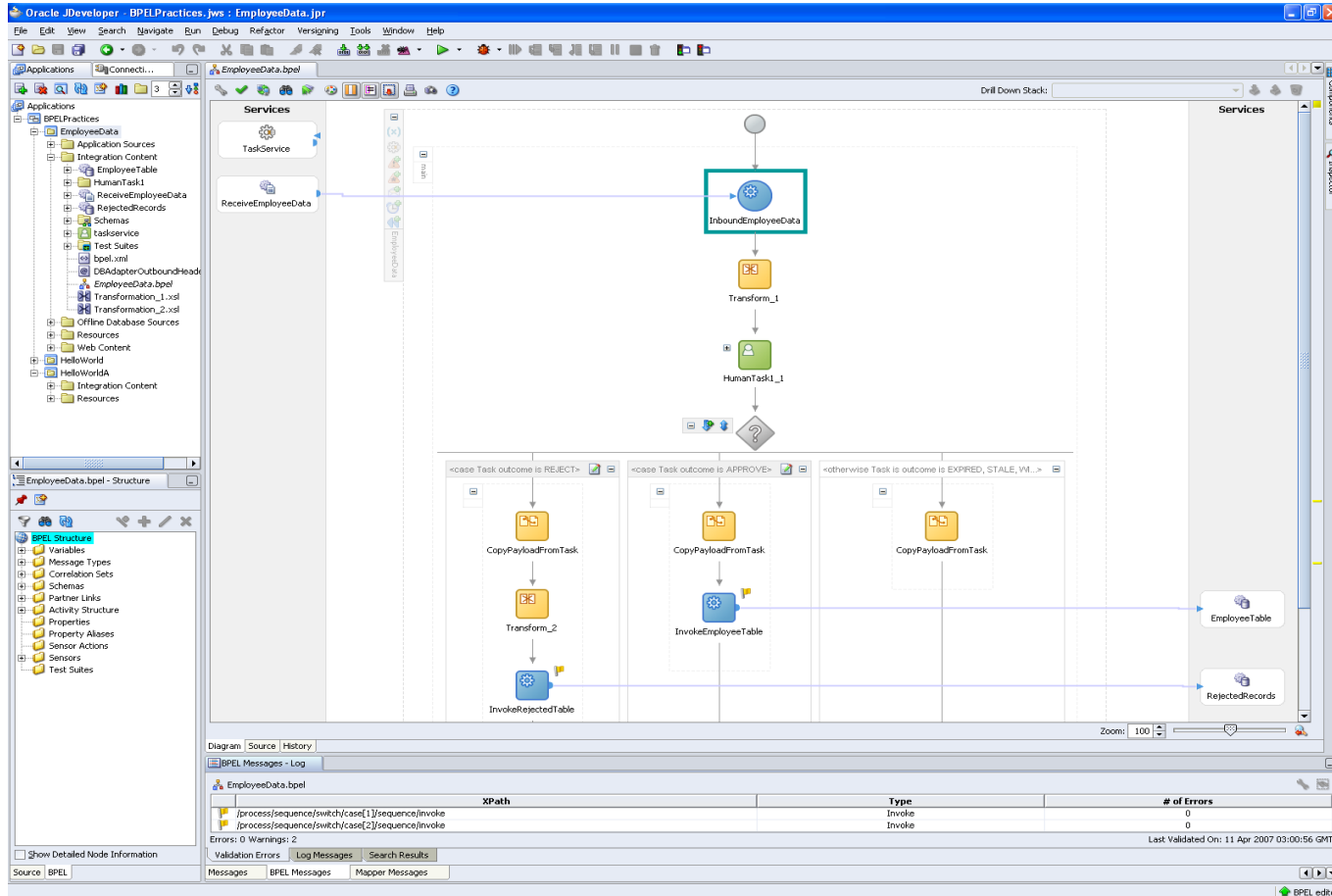
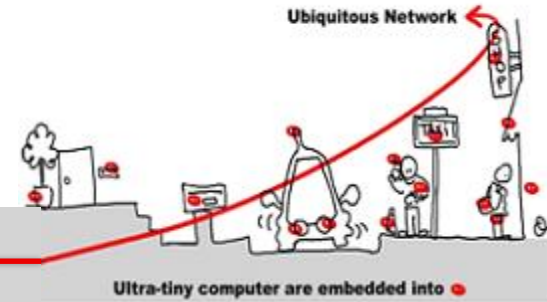


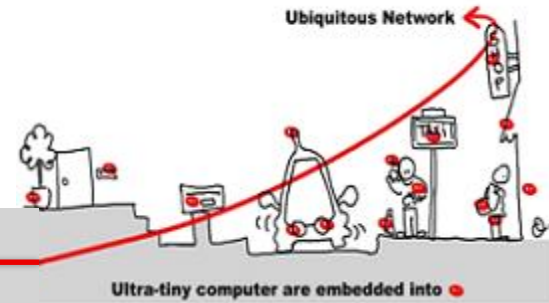
- Transactions and Exceptions
 - Building on top of WS-Coordination and WS-Transaction specifications
 - Transaction
 - A set of activities can be grouped in a single transaction through the `<scope>` tag
 - Can specify compensation handlers (rollback) if there is an error
 - Exception Handling
 - Through the use of `throw` and `catch` (similar to Java)

BPEL – Example Process



BPEL Process in JDeveloper

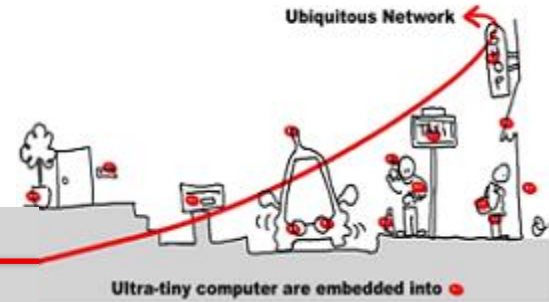




Event-driven Composition

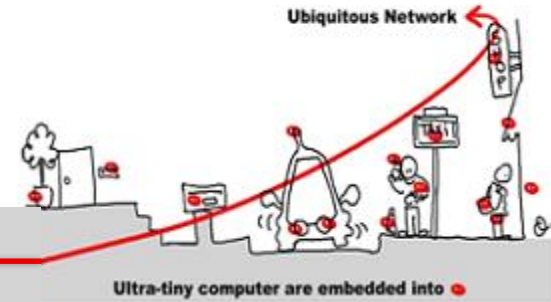
Through Components Assemblies

Overview



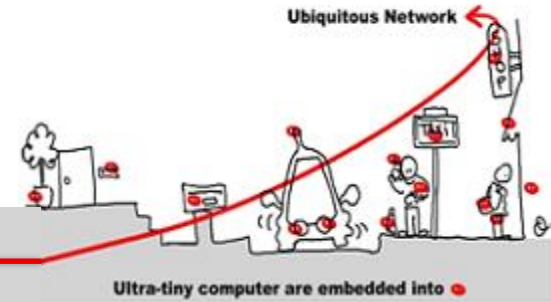
- Introduction
- LightWeight Component Model
- LCA (Wcomp) Component Model, for ubiquitous computing

What is a Component?



- *“A software component is a software element that conforms to a component model, and can be independently deployed and composed without modification according to a composition standard.”*
- Component Model
 - Interaction Standards
 - Clearly Defined Interface
 - Composition Standards
 - Describe how components can be composed into larger structures
 - Substitutions

CBSE Definition

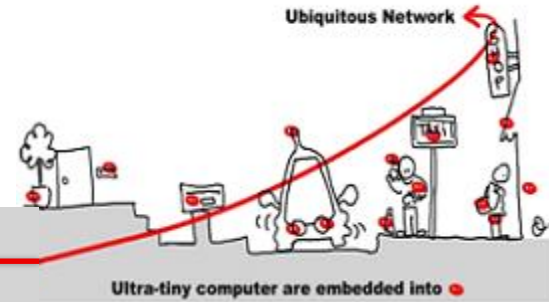


- Developing new software from pre-built components.
- Attempt to make an association between SE and other engineering disciplines.

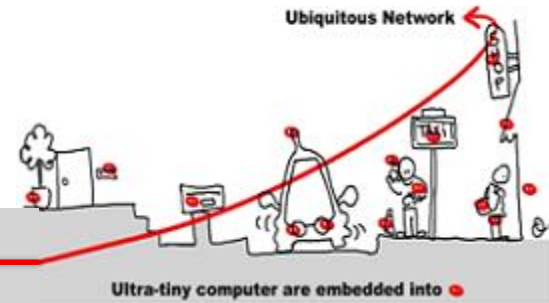
Advantages of CBSE

- Management of Complexity
- Reduce Development Time
- Increased Productivity
- Improved Quality

More on Trust



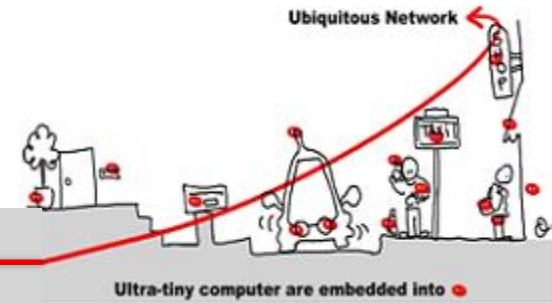
- Components come in several forms
 - Binary
 - Source Code
- Need a Certification Standard
 - Tests
 - Environments
- => Formal Validation and Model Checking is a way to do that (SCADE and synchronous programming)



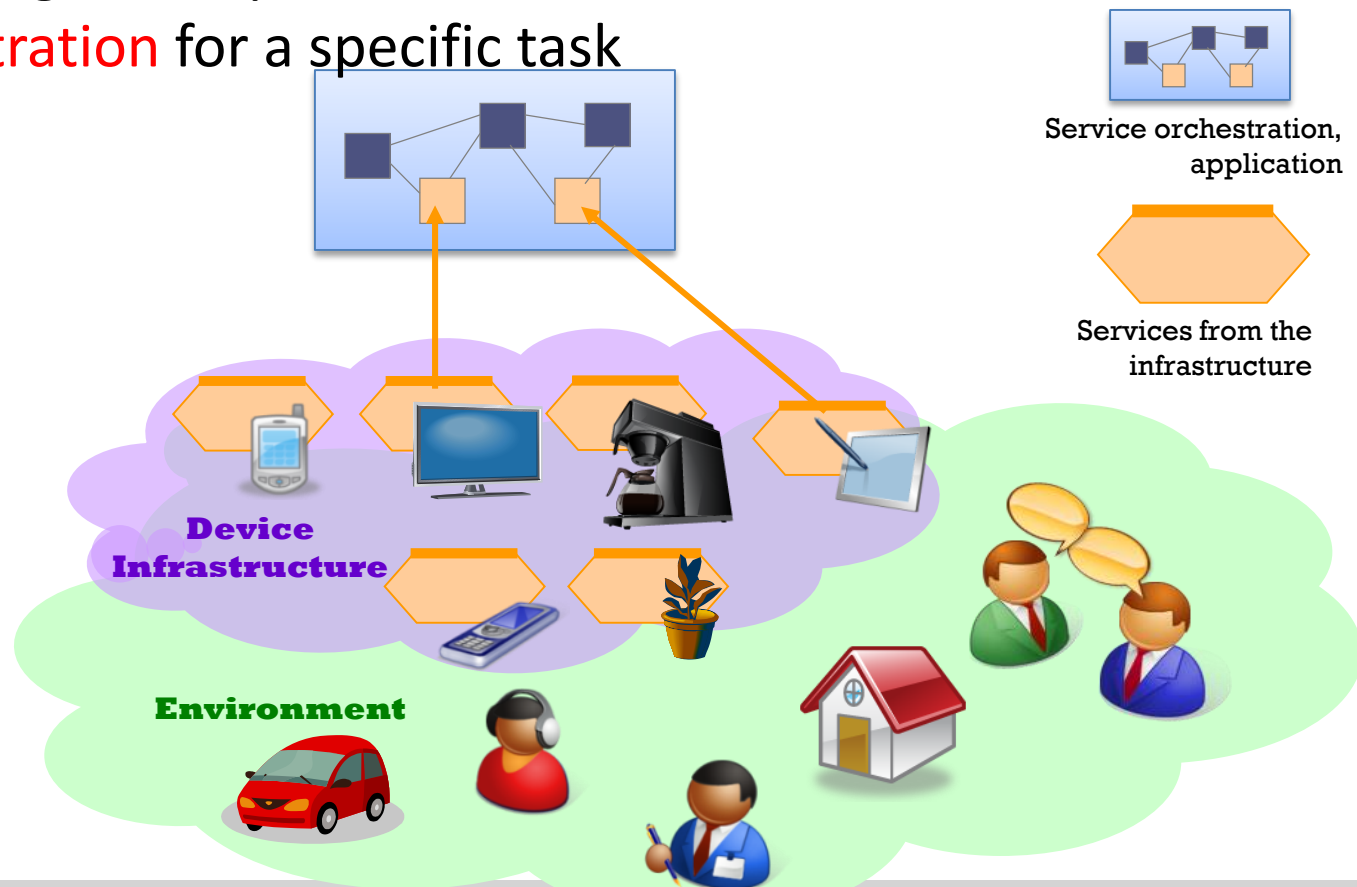
A way to dynamically compose services with an event driven approach

LCA Model

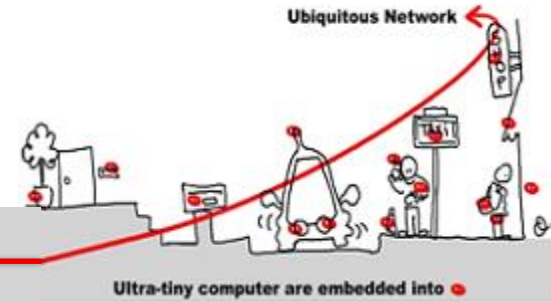
LCA to compose services for Devices



- Lightweight Component Architecture to create service-based **orchestration** for a specific task

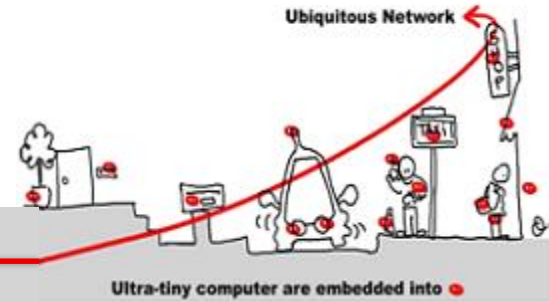


WComp and Local Composition (LCA)



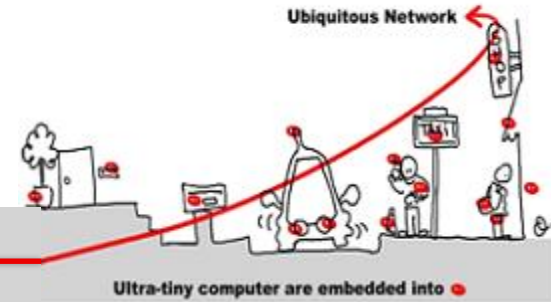
- Main requirements for ubiquitous computing :
 - Composition must be **event driven**
 - **At runtime**
- Solution :
 - **Event based Local Composition** : LCA (Lightweight Component Model) for each application execution node.

Main Features of LCA Model :

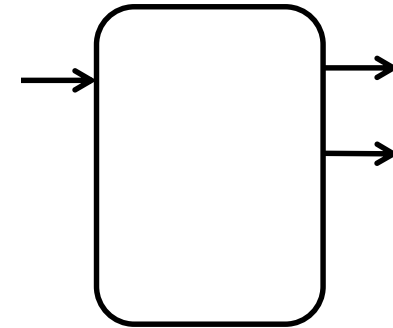


- Goal :
 - Allow to compose Services for Device between them towards a multiple devices ubiquitous application.
- Principles
 - LightWeight Components Approach :
 - Like OpenCom, JavaBeans, PicoContainer
 - On the same execution node
 - For each execution node, a container dynamically manage the assembly of components
 - Event-based interaction between components
 - Blackbox LightWeight Components

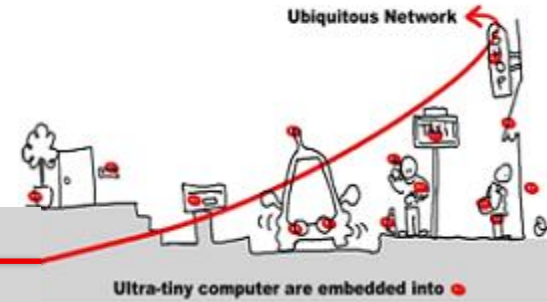
LCA Component Model



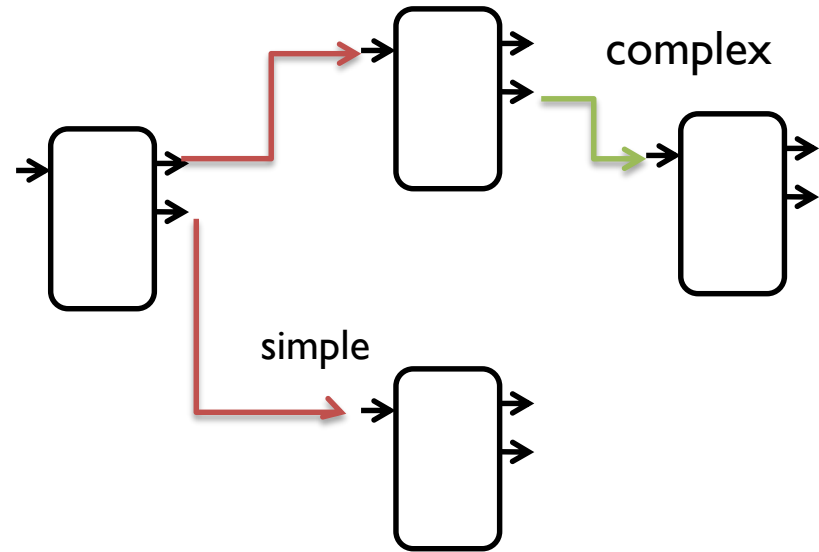
- Input : Methods
 - C2.Method (param)
- Output : Events
 - C1.Event (param)
- Internal Properties are associated with Getters and Setters
 - C2.Set<Name>(<type>)
 - <type> C2.Get<Name>()



LCA, connectors



- Demo
- (Generated source code)



Connectors

Simple Event based Connector

`C1.Event (param) → C2.Method (param)`

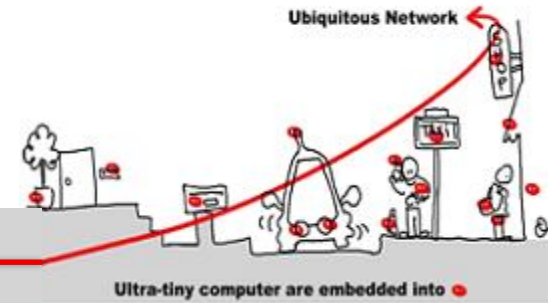


Complex Event based Connector

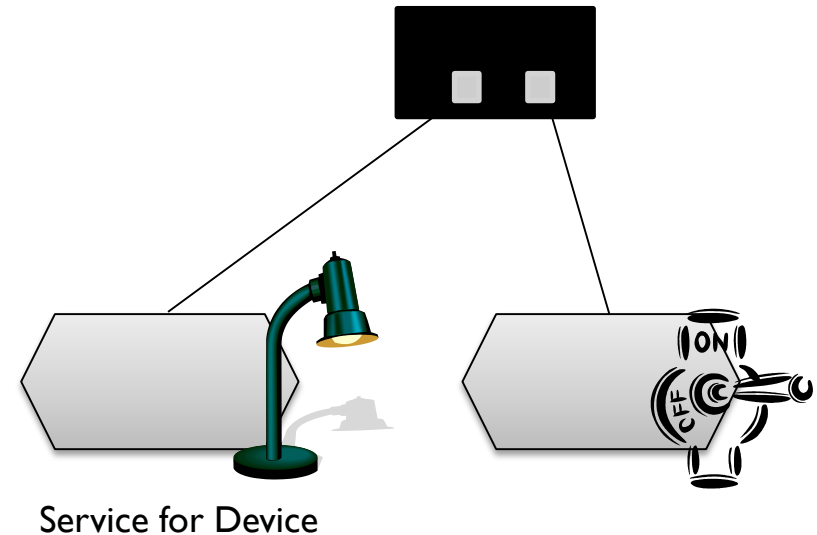
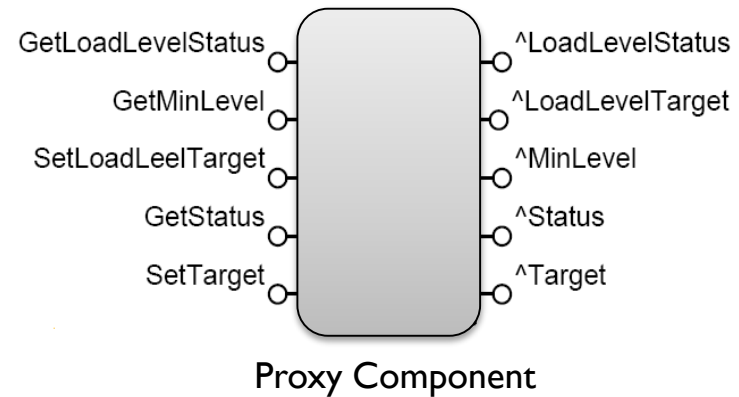
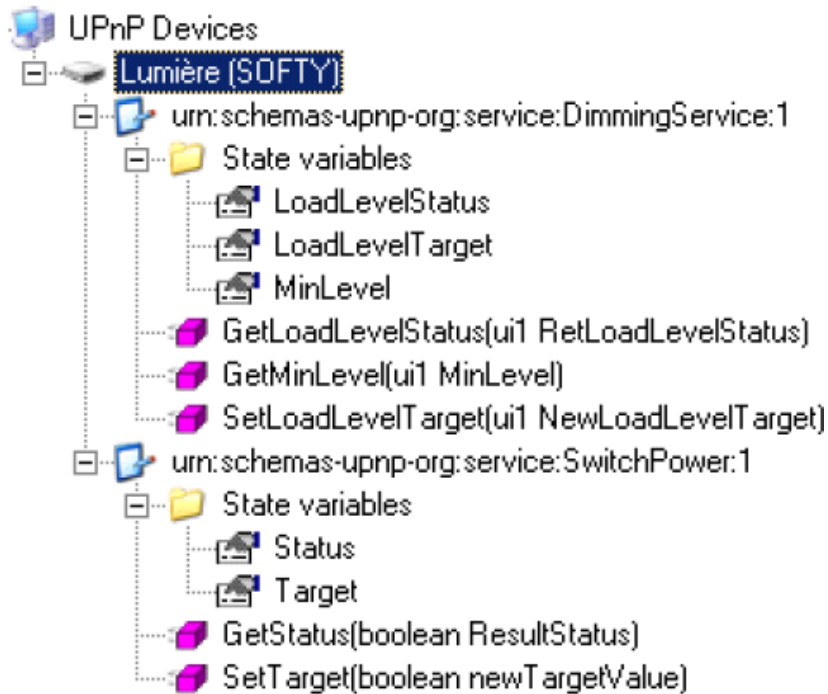
`C1.Event (param) → C2.Method (C1.GetAProperty())`



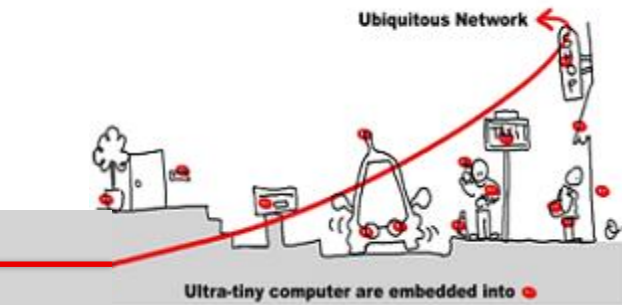
LCA Proxy components to access to Services for Devices



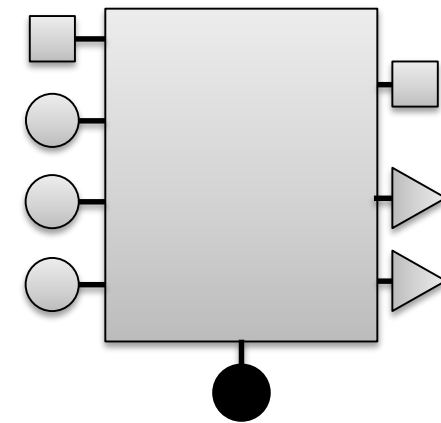
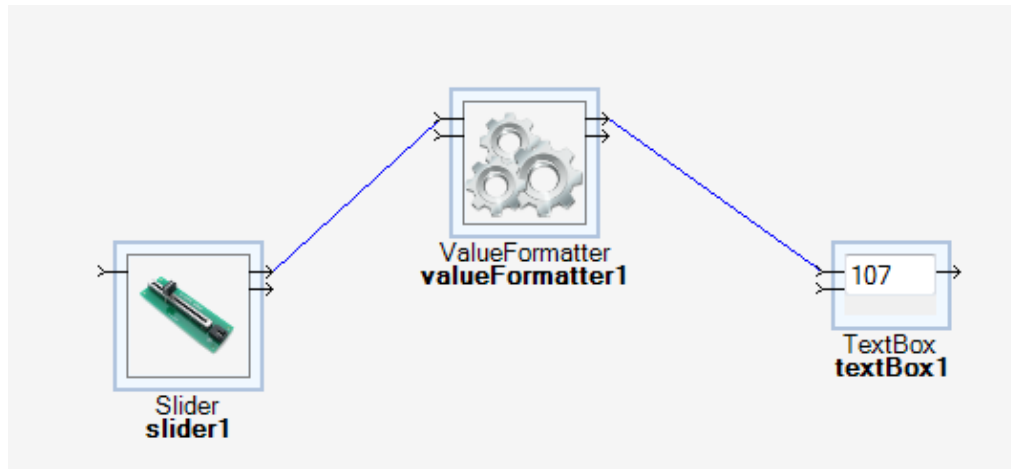
- Demo



Build your own orchestration set of operators / beans

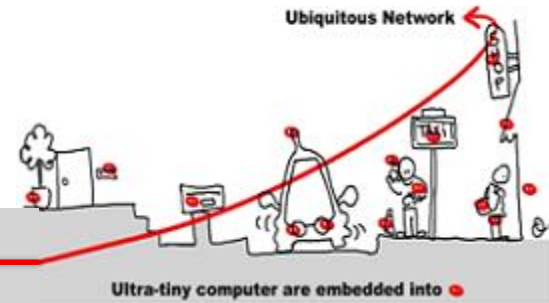


- Demo



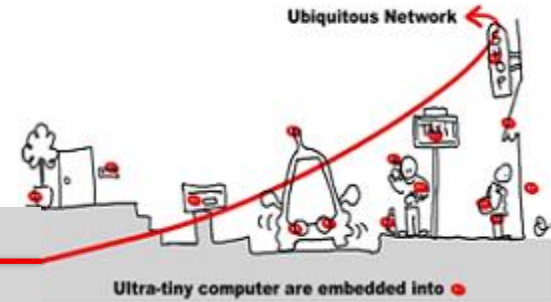
- Property
- Method
- ▶ Event source

- If you need If, filters, ... feel free ..



Build your own component with C#

BeanWComp .Net template



- Events are based on « delegate » model (in C#)

Category

Event

```
using System;
using System.ComponentModel;
using WComp.Beans;

namespace Bean4
{
    /// <summary>
    /// Description rsume de Class1.
    /// </summary>
    [Bean(Category="MyCategory")]

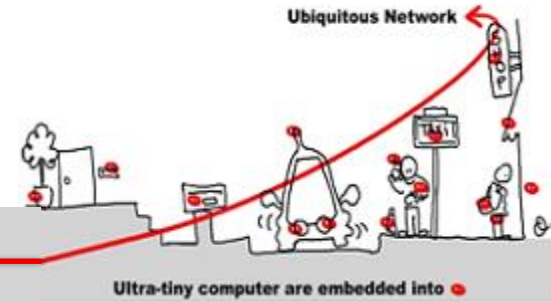
    public class Class1
    {

        // delegate implicite de void EventHandler(object sender, EventArgs e)

        public event EventHandler MyEvent;

        // graphiquement ce qui sera fait :
        // MyEvent += new EventHandler(func)
        // avec private void func(object sender, EventArgs e)
    }
}
```

BeanWComp .Net template

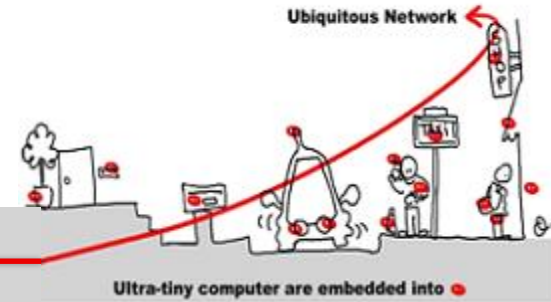


- Propriétés

```
...  
  
// Nom de la propriété avec minuscule  
// variable de sauvegarde propriété  
  
    protected int myprop = 1;  
  
        //meta donnée : valeur par défaut propriété  
        [DefaultValue(1)]  
  
// déclaration propriété : public <type> Nom  
public int Myprop  
{  
    get  
    {  
        return myprop;  
    }  
  
    set  
    {  
        if (myprop < 1)  
        {  
            throw new ArgumentException("positif !");  
        }  
        // mot clef value  
        myprop = value;  
    }  
}  
  
...
```

Property

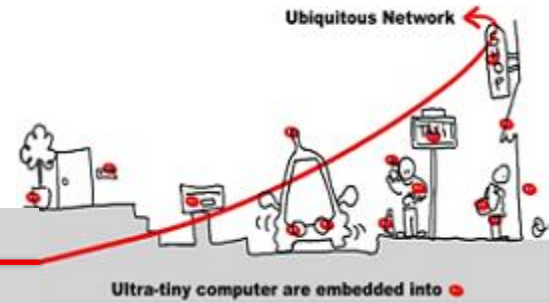
BeanWComp .Net template



- Méthodes

```
// méthodes  
  
public void MyStep(int val1, int val2)  
{  
    if (myprop >= max)  
    {  
        myprop=1;  
        MyEvent(this, null);  
    }  
    else  
        myprop++;  
}
```

Method

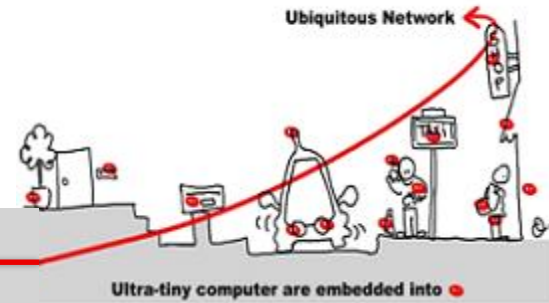


CNS 3260

C# .NET Software Development

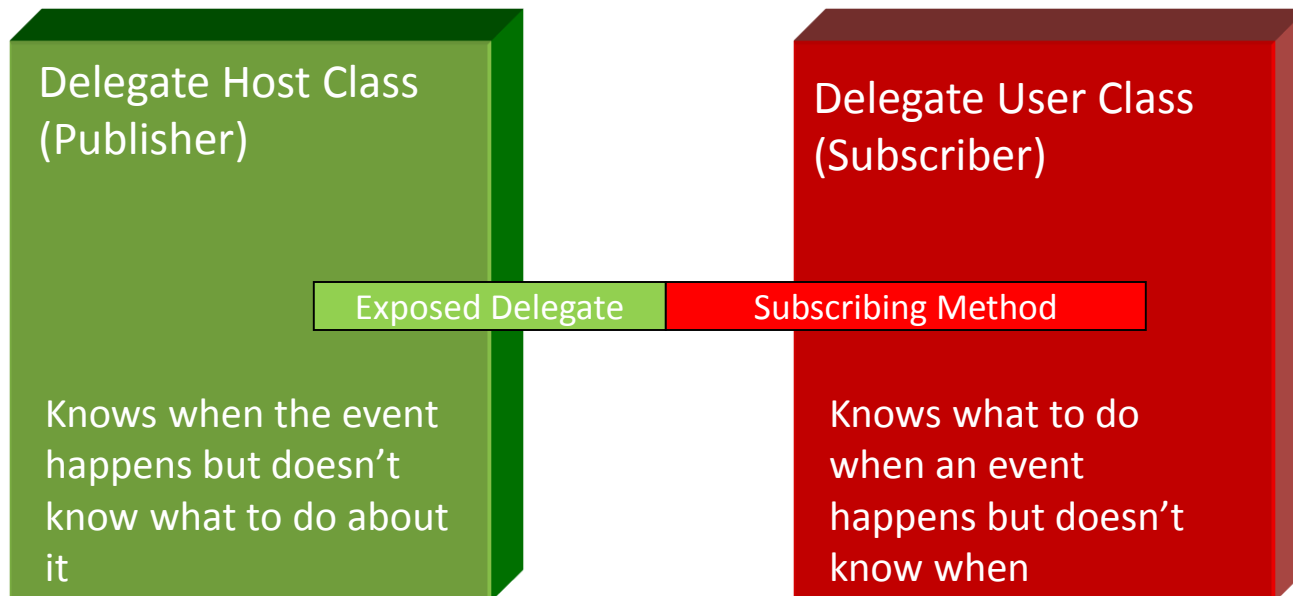
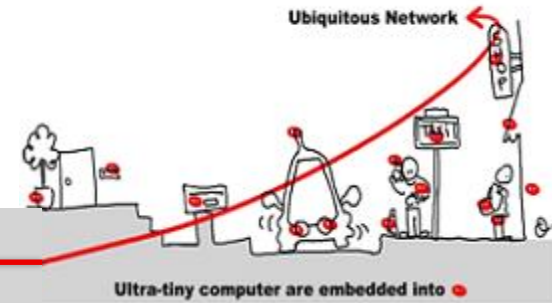
ANNEX DELEGATES AND EVENTS IN C#

Delegate types



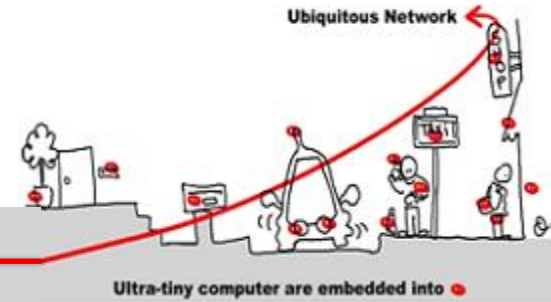
- A delegate declaration defines a new type
- Delegates are similar to function pointers
- Delegate types are derived from `System.MulticastDelegate`

Simple Delegate Command Pattern



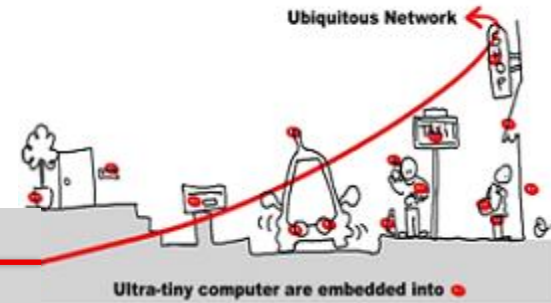
The Observer Pattern or .NET Event Model

Two reasons to use Delegates



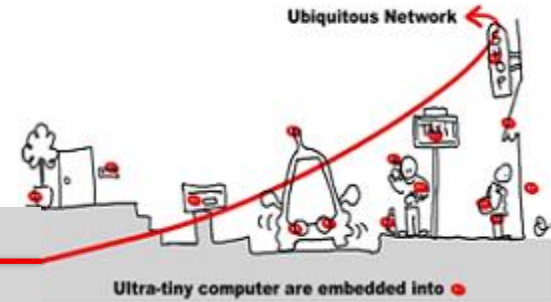
- When you're not sure what should happen when an event occurs
 - GUI events
 - Threading situations
 - Callbacks
 - Command Pattern
- To keep your interface clean
 - Looser coupling

Defining and using Delegates



- three steps:
 - Declaration
 - Instantiation
 - Invocation

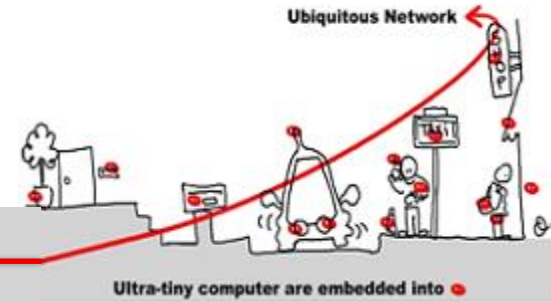
Delegate Declaration



- namespace some_namespace
- {
- delegate void MyDelegate(int x, int y);

Delegate Type Name

Delegate Instantiation



```
delegate void MyDelegate(int x, int y);
```

```
class MyClass
```

```
{  
    private MyDelegate myDelegate = new MyDelegate( SomeFun );
```

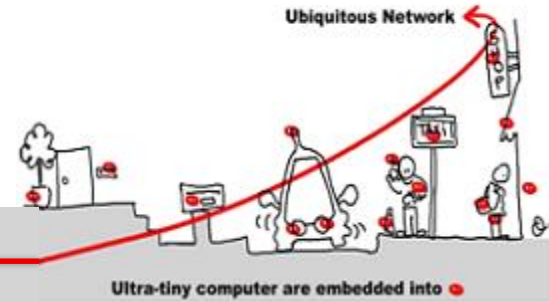
```
    public static void SomeFun(int dx, int dy)
```

```
{  
    }  
}
```

Invocation Method

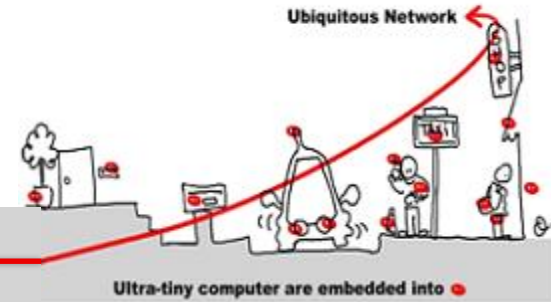
Invocation Method
name (no params
or perens)

Delegate-Method Compatibility



- A Method is compatible with a Delegate if
 - They have the same parameters
 - They have the same return type

Delegate Invocation



```
class MyClass
{
    private MyDelegate myDelegate;

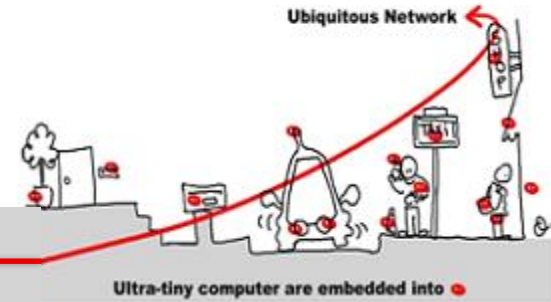
    public MyClass(MyDelegate myDelegate)
    {
        this.MyDelegate = myDelegate;
    }

    private void WorkerMethod()
    {
        int x = 500, y = 1450;

        if(myDelegate != null)
            myDelegate(x, y);
    }
}
```

Attempting to invoke a delegate instance whose value is null results in an exception of type *System.NullReferenceException*.

Delegate's "Multicast" Nature

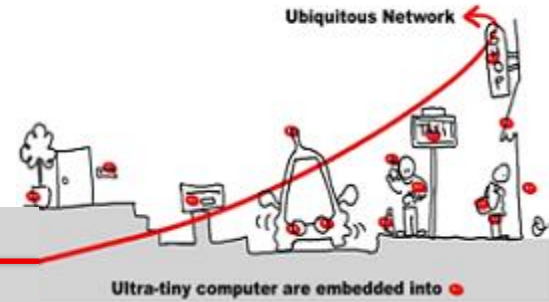


- Delegate is really an array of function pointers

```
mc.MyDelegate += new MyDelegate( mc.Method1 );  
mc.MyDelegate += new MyDelegate( mc.Method2 );  
mc.MyDelegate = mc.MyDelegate + new MyDelegate( mc.Method3 );
```

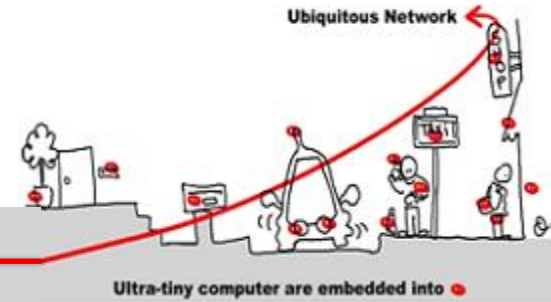
- Now when Invoked, mc.MyDelegate will execute all three Methods
- Notice that you don't have to instantiate the delegate before using +=
 - The compiler does it for you when calling +=

The Invocation List



- Methods are executed in the order they are added
- Add methods with + and +=
- Remove methods with - and -=
 - Attempting to remove a method that does not exist is not an error
- Return value is whatever the last method returns
- A delegate may be present in the invocation list more than once
 - The delegate is executed as many times as it appears (in the appropriate order)
 - Removing a delegate that is present more than once removes only the last occurrence

Multicast example



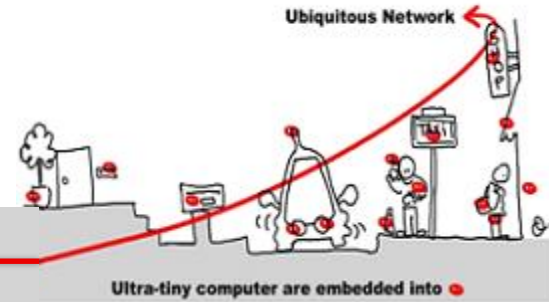
```
mc.MyDelegate = new MyDelegate( mc.Method1 );  
mc.MyDelegate += new MyDelegate( mc.Method2 );  
mc.MyDelegate = mc.MyDelegate + new MyDelegate( mc.Method3 );
```

```
// The call to:  
mc.MyDelegate(0, 0);  
// executes:
```

```
// mc.Method1  
// mc.Method2  
// mc.Method3
```

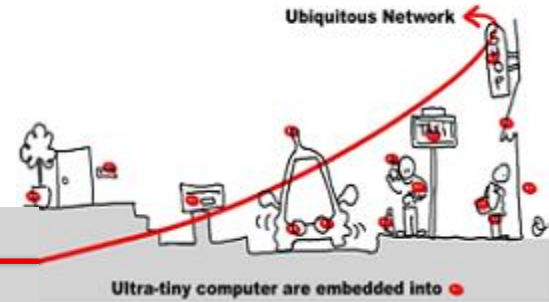
(See Delegates Demo)

Events



- Events are “safe” delegates
 - But they are delegates
- Restricts use of the delegate (event) to the target of a += or -= operation
 - No assignment
 - No invocation
 - No access of delegate members (like GetInvocation List)
- Allow for their own Exposure
 - Event Accessors

Event Accessors



```
public delegate void FireThisEvent();
class MyEventWrapper
{
    private event FireThisEvent fireThisEvent;

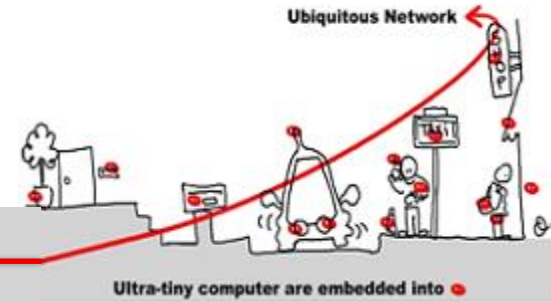
    public void OnSomethingHappens()
    {
        if(fireThisEvent != null)
            fireThisEvent();
    }

    public event FireThisEvent FireThisEvent
    {
        add { fireThisEvent += value; }
        remove { fireThisEvent -= value; }
    }
}
```

add and remove
keywords

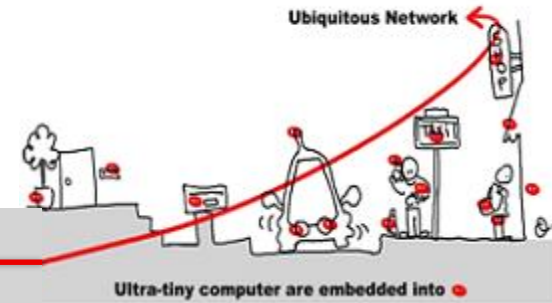
(See Event Demo)

Library Delegates



- ThreadStart
- TimerCallback
- ASyncCallback
- EventHandler
- KeyPressEventHandler
- KeyEventHandler
- etc.

References



The Internet of Things: A survey

Luigi Atzori ^a, Antonio Iera ^b, Giacomo Morabito ^{c,*}

^a DIEE, University of Cagliari, Italy

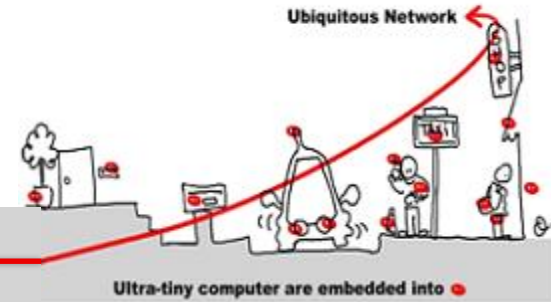
^b University "Mediterranea" of Reggio Calabria, Italy

^c University of Catania, Italy



- 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>

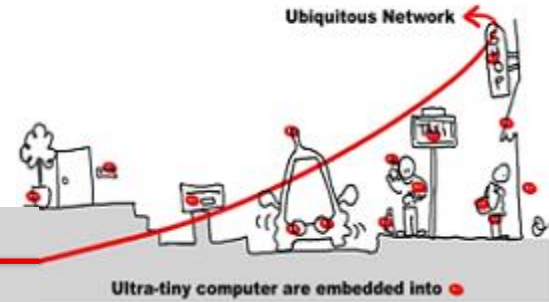
References



Mashups

- X. Liu et al., “Towards Service Composition Based on Mashup,” Proc. IEEE Congress on Services, 2007, pp. 332–339
<http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=4278815>
- Services Mashups: The New Generation of Web Applications, Issue No.05 - IEEE Internet Computing - September/October (2008 vol.12), pp: 13-15, Djamel 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



BPEL :

- Y. Zheng and P. Krause, "Analysis of bpele data dependencies," in Proc. of EUROMICRO, IEEE Computer Society, 2007.
<http://epubs.surrey.ac.uk/1985/1/fulltext.pdf>

Event Driven Composition :

- J.-Y. Tigli, S. Lavirotte, G. Rey, V. Hourdin, M. Riveill, "Lightweight Service Oriented Architecture for Pervasive Computing" IJCSI International Journal of Computer Science Issues, Vol. 4, No. 1, September 2009, ISSN (Online): 1694-0784, ISSN (Print): 1694-0814