Component based middleware and service composition for ubiquitous computing

Ass. Prof. Jean-Yves Tigli

tigli@polytech.unice.fr

www.tigli.fr

Ref : Component-based Software Engineering

Ivica Crnkovic
Next Step ... 

1. **Infrastructure**: based on Web services for Device
2. **Composition**: based on CBSE
3. **Self-Adaptation**: using Aspects of Assembly (AA)
Overview

• Introduction
• ACME Architectural Description Language
• Java Bean 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)
Disadvantages of CBSE

- Development of Components
- Lack of Components
- Component Maintenance Costs
- Sensitivity to changes
- Trust
General Model of CBSE

ADL - ACME
Architecture Definition

Languages

• ADLs primarily address the issues related to the early phases of software engineering
  – Design
  – Analysis

• They identify a number of concepts, such as:
  – Architecture, configurations, connectors, bindings, properties, hierarchical models, style, static analysis and behavior.
ACME Architectural Description Language

- Components and Ports
- Connectors and Roles
- Systems and Attachments
- Representations and Bindings
Components and Ports

- **Components**
  - Represent the computational elements and data stores of a system.

- **Ports**
  - Are the points of interaction between a component and its environment.
Connectors and Roles

• Connectors
  – Represent interactions between components such as method calls or an SQL connection between a client and a database server.

• The interface of a connector is defined as a set of roles
Systems and Attachments

• The structure of a system is specified by a set of components, a set of connectors, and a set of attachments.

• Attachment
  – Links a component port to a connector role.

![Diagram of systems and attachments]
Representations and Bindings

- Component
- Connector
  - Port
  - Role
- Attachment
- Binding
Fine grained Component

Or local Component
Fine-grained Component Ex. JavaBean Model and Key Features

• "A Java Bean is a reusable software component that can be manipulated visually in a builder tool ".
• The Java Bean was designed for the construction of graphical user interface (GUI).
• Explicitly tailored to interact in two different contexts:
  – At composition time, within the builder tool.
  – At execution time, with the runtime environment.
• Any Java class that adheres to certain conventions regarding property and event interface definitions can be a JavaBean.
• Beans are Java classes that can be manipulated in a visual builder tool and composed into applications.
Interface of a Component

- This model defines four types of port:
  - methods,
  - properties,
  - event sources (generate an event)
  - event sinks called listeners (they receive event)
Implementation of a Component

• Most bean components are implemented by a simple object and naming convention
• A component factory is a class
Components Assembly

- Assembly is one of the key features of Bean.
  - Composition tools (Bean Box)

- Different ways of assembling components are supplied.

Component-based assembly

Heterogeneous assembly
A component (assembly) is made of modules, which are traditional executable files (DLL).

Modules cannot be assemblies, thus the .NET model is not hierarchical.
Framework: The Container Approach

- Framework – a set of containers. Containers contain components and provide a set of standard services (security, events, persistence, life-cycle support)
A way to dynamically compose services

SLCA 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 based
  – 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, Bean WComp and ports

• Demo

![Diagram showing LCA, Bean WComp and ports](image-url)
**BeanWComp .Net template**

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

```csharp
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)
    }
}
```
• Propriétés

```csharp
... // 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

```csharp
// méthodes

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

- Demo
- (Generated source code)

Connectors

**Simple Event based Connector**

\[ C1.\text{Event} \ (\text{param}) \rightarrow C2.\text{Method} \ (\text{param}) \]

**Complex Event based Connector**

\[ C1.\text{Event} \ (\text{param}) \rightarrow C2.\text{Method} \ (C1.\text{GetAProperty}()) \]
LCA Proxy components to access to Services for Devices

- **Demo**

![Diagram of LCA Proxy components and their relationships with UPnP Devices and Service for Device](image)

- **Proxy Component**
  - `GetLoadLevelStatus` (LoadLevelStatus, LoadLevelTarget, MinLevel)
  - `GetMinLevel`
  - `SetLoadLevelTarget`
  - `GetStatus`
  - `SetTarget`

- **UPnP Devices**
  - Lumière (SOFTY)
  - `urn:schemas-upnp-org:service:DimmingService:1`
    - State variables
      - LoadLevelStatus
      - LoadLevelTarget
      - MinLevel
      - GetLoadLevelStatus
      - GetMinLevel
      - SetLoadLevelTarget
    - `urn:schemas-upnp-org:service:SwitchPower:1`
      - State variables
        - Status
        - Target
        - GetStatus
        - SetTarget

- **Service for Device**

---

2012-2013

MASTER_INFO_CORTE_IAM_2013 component_based Middleware Jean-Yves Tigli - tigli@polytech.unice.fr - www.tigli.fr
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

Delegate Host Class (Publisher)

Exposed Delegate

Delegate User Class (Subscriber)

Subscribing Method

Knows when the event happens but doesn’t know what to do about it

Knows what to do when an event happens but doesn’t know when

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 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

```csharp
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

```csharp
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
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; }
    }
}

(See Event Demo)
Library Delegates

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


