IAm and the « COMPOSE » layer

Lecturer : Ass. Prof. Jean-Yves Tigli

http://www.tigli.fr

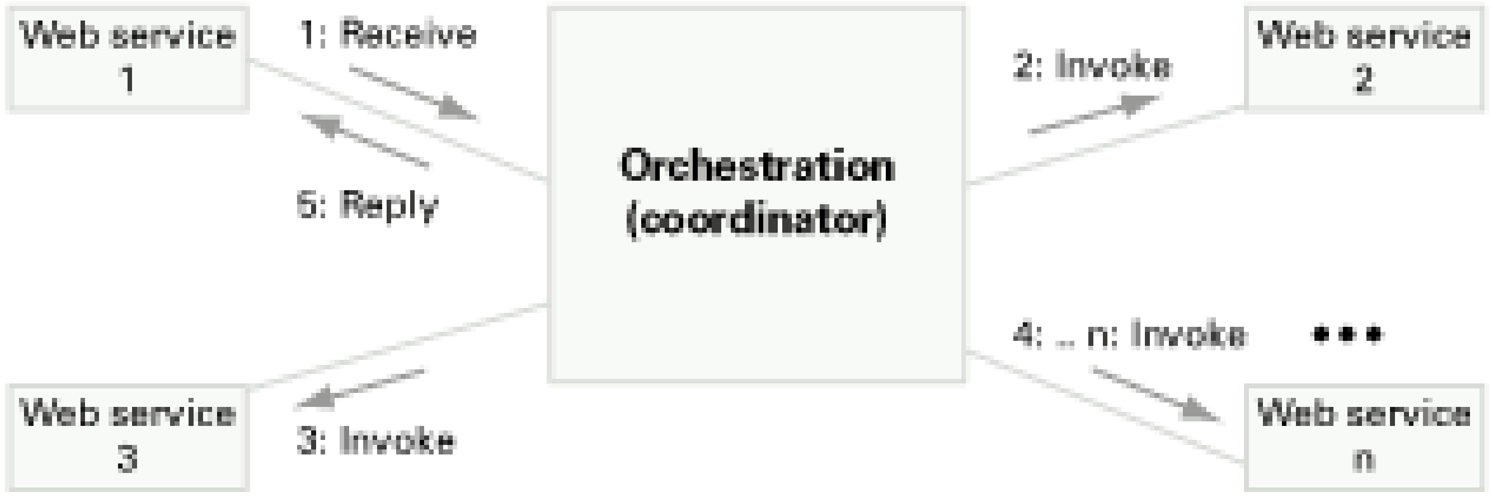
at Polytech of Nice - Sophia Antipolis University

Email : tigli@polytech.unice.fr

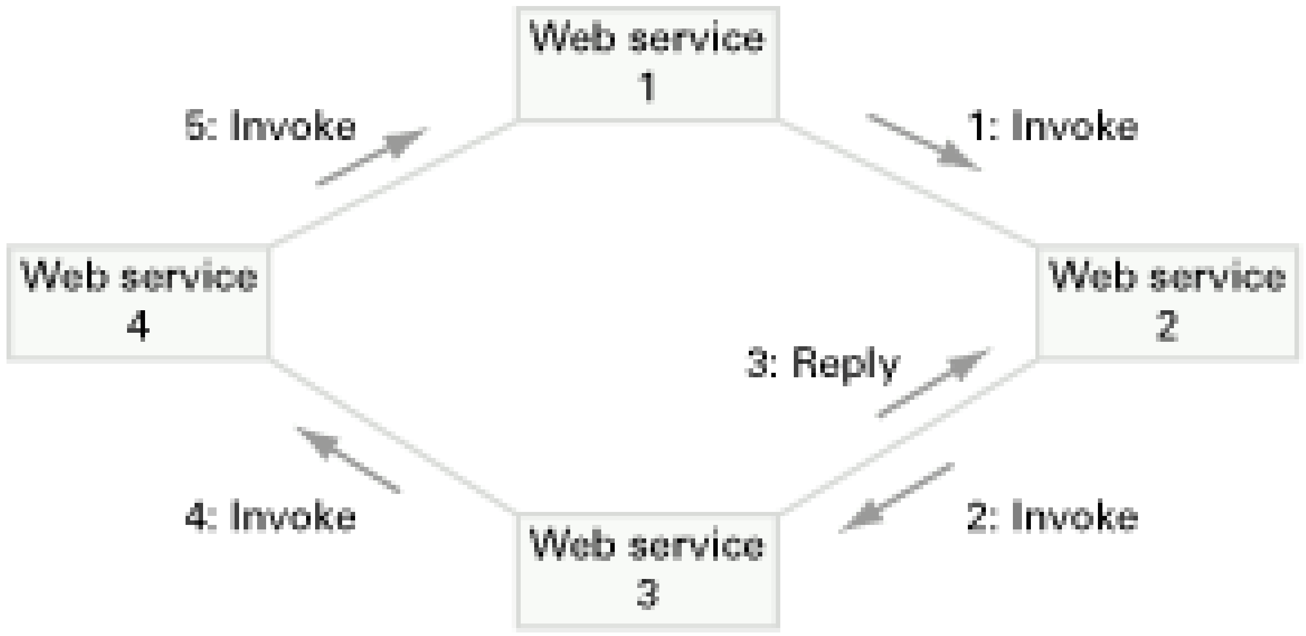
Service Composition

- 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

Service Composition an Orchestration (contd.)



Service Composition as a 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)

Another Example : 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

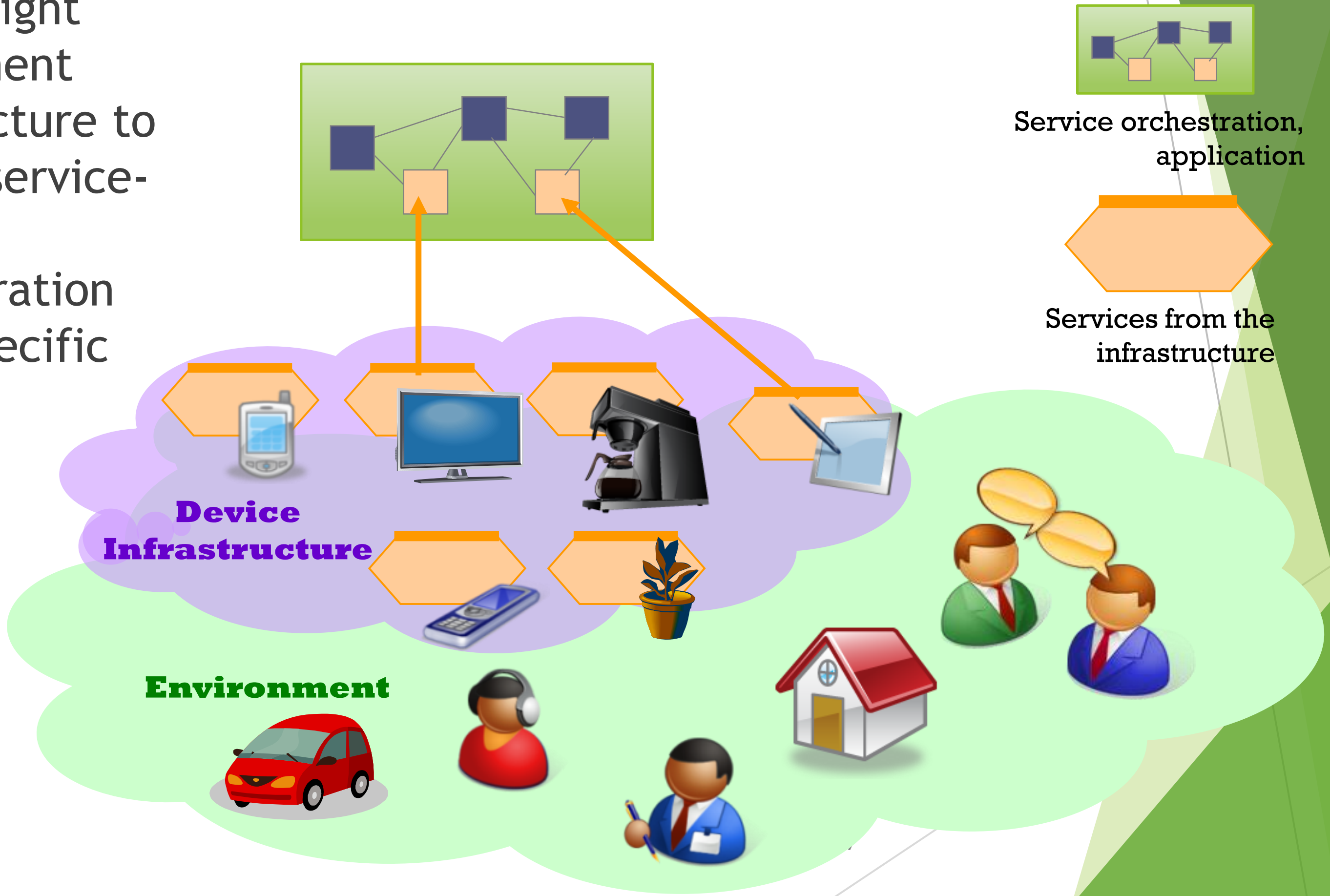
- => Formal Validation and Model Checking is a way to do that (SCADE and synchronous programming)

A way to dynamicaly 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


Service orchestration, application
Services from the infrastructure
Device Infrastructure
Environment

WComp and Local Composition (LCA)

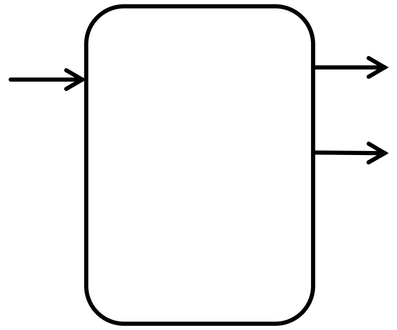
- Main requirements for ubiquituous computing :
  - Composition must be event driven
  - At runtime ….
- Solution :
  - Event based Local Composition : LCA (Lightweight Component Model) for each application execution node.

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

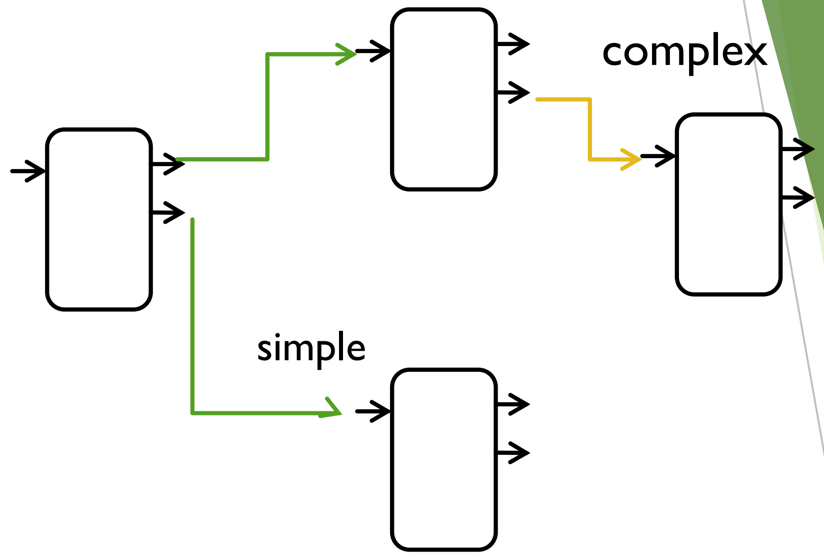
LCA Component Model

- 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

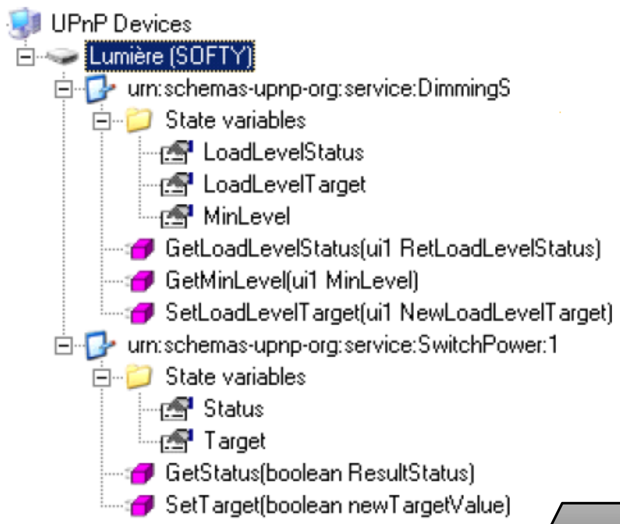
C1.Event (param) → C2.Method (param)

Complex Event based Connector

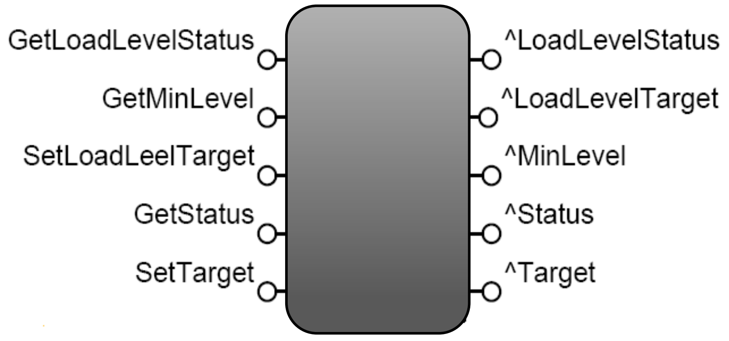
C1.Event (param) → C2.Method (C1.GetAProperty())

LCA Proxy components to access to Services for Devices

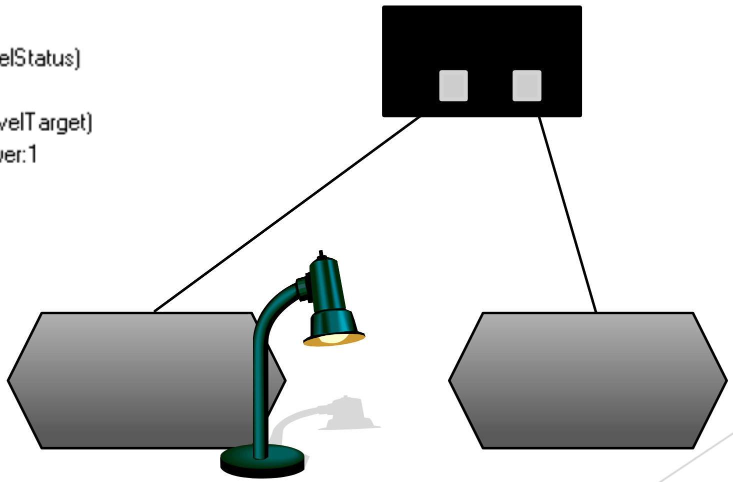
- Demo



- UPnP Devices
  - Lumière (SOFTY)
    - urn:schemas-upnp-org:service:DimmingS
      - State variables
        - LoadLevelStatus
        - LoadLevelTarget
        - MinLevel
      - GetLoadLevelStatus(ui1 RetLoadLevelStatus)
      - GetMinLevel(ui1 MinLevel)
      - SetLoadLevelTarget(ui1 NewLoadLevelTarget)
    - urn:schemas-upnp-org:service:SwitchPower:1
      - State variables
        - Status
        - Target
      - GetStatus(boolean ResultStatus)
      - SetTarget(boolean newTargetValue)



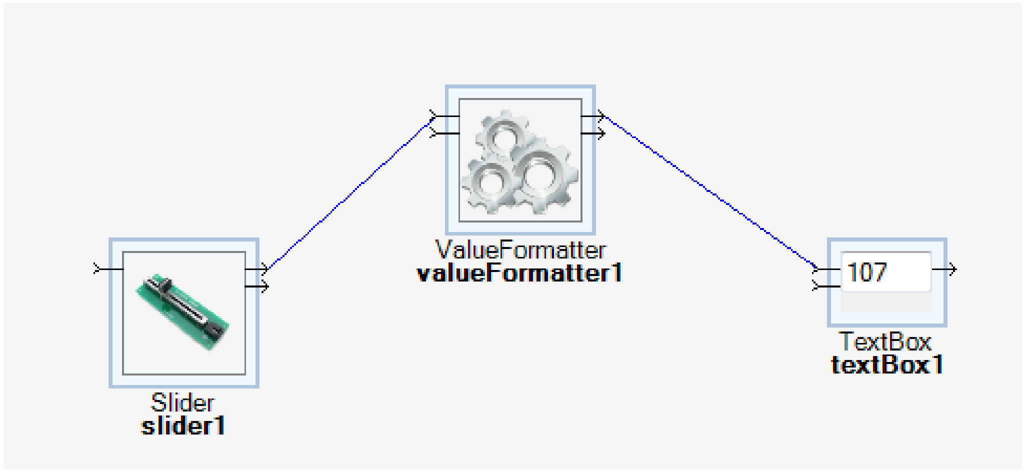
Proxy Component

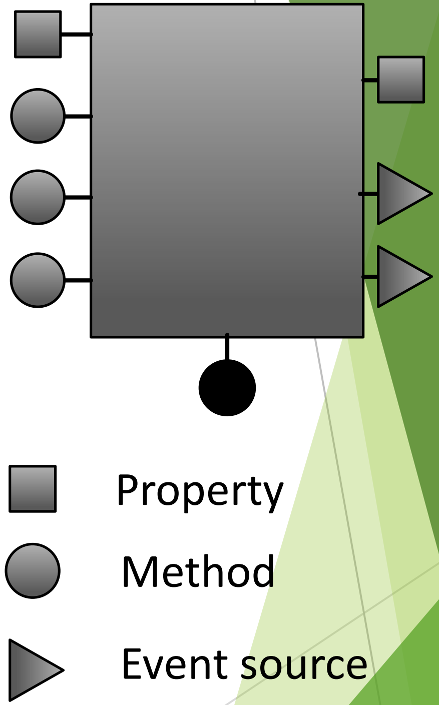


Service for Device

Build your own orchestration set of operators / beans

- Demo





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

Build your own component with C#

BeanWComp .Net template

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

```
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)
```

Category

Event

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

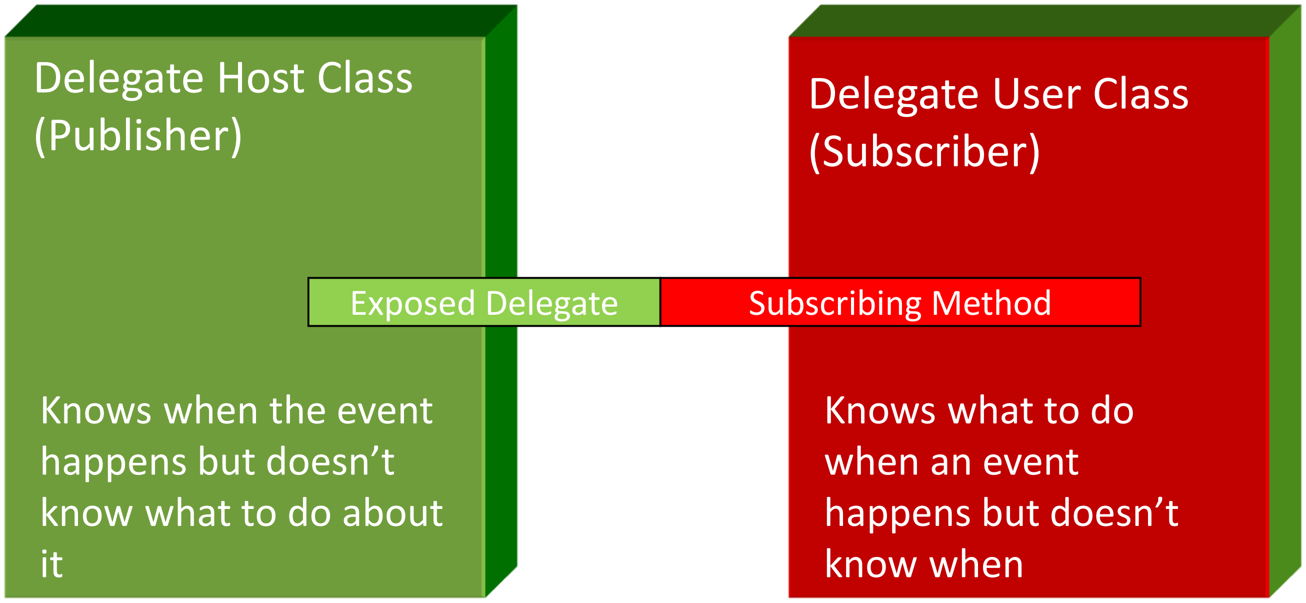
Annex Delegates and Events in C#

C# .NET Software Development

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
Subscribing Method
Delegate User Class (Subscriber)
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 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)