OBJETS COMMUNICANTS

FILIÈRE INFORMATIQUE AMBIANTE ET MOBILE SI5 – MASTER IFI

"RENDEZ VOS OBJETS COMMUNICANTS ET INTERACTIFS AVEC DES PHIDGETS"

Jean-Yves Tigli, http://www.tigli.fr

Email : tigli@polytech.unice.fr

Tel : 0492081676

Bureau : 408

POLYTECH NICE-SOPHIA

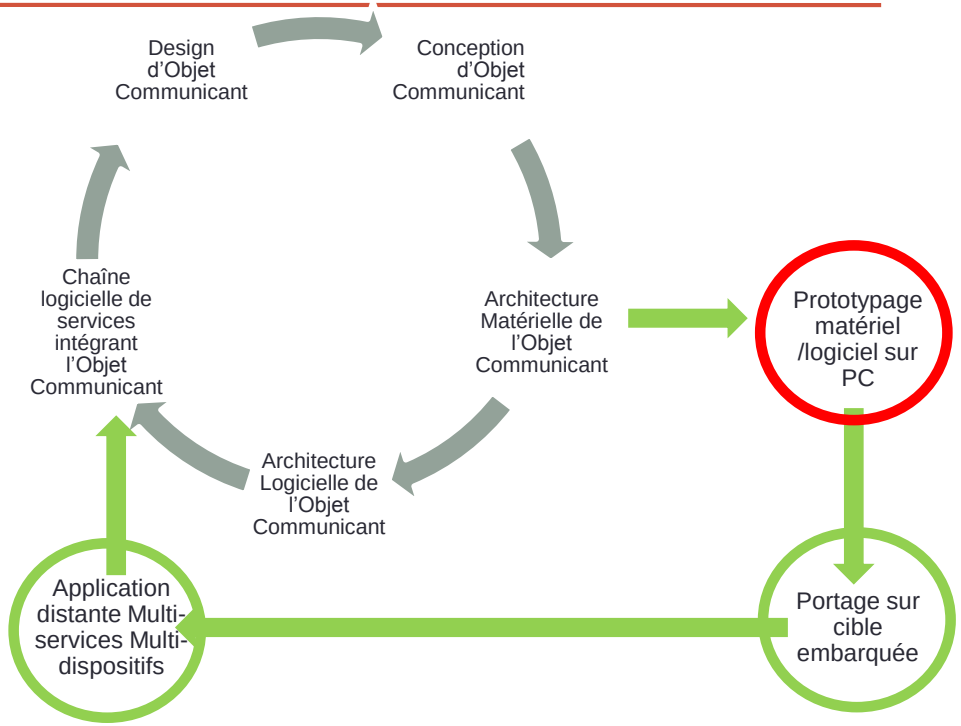
Université Nice Sophia Antipolis

i3s sophia antipolis

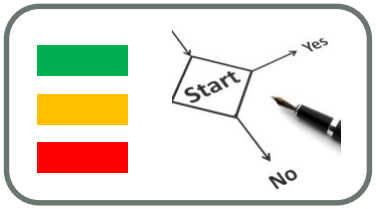
cnrs

RAINBOW

PLATEFORME LOGICIELLE WCOMP DE PROTOTYPAGE

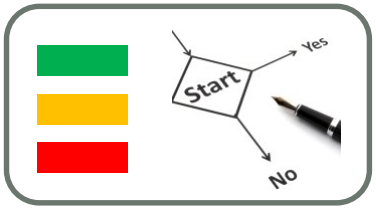


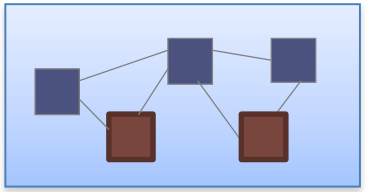
Avec le middleware WComp



- Avantage :
  - Architecture orientée composant
  - Composition dynamique au runtime
- Modèle du Middleware (LCA)
  - Modèle de Bean WComp (Propriété / Méthodes /Evénements)
  - Connecteurs WComp
- Créer / Gérer votre application par assemblage de composants

Avec le middleware WComp : Modèle LCA



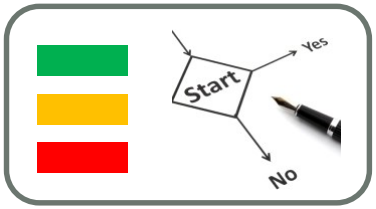


- LCA : Lightweight Components Architecture

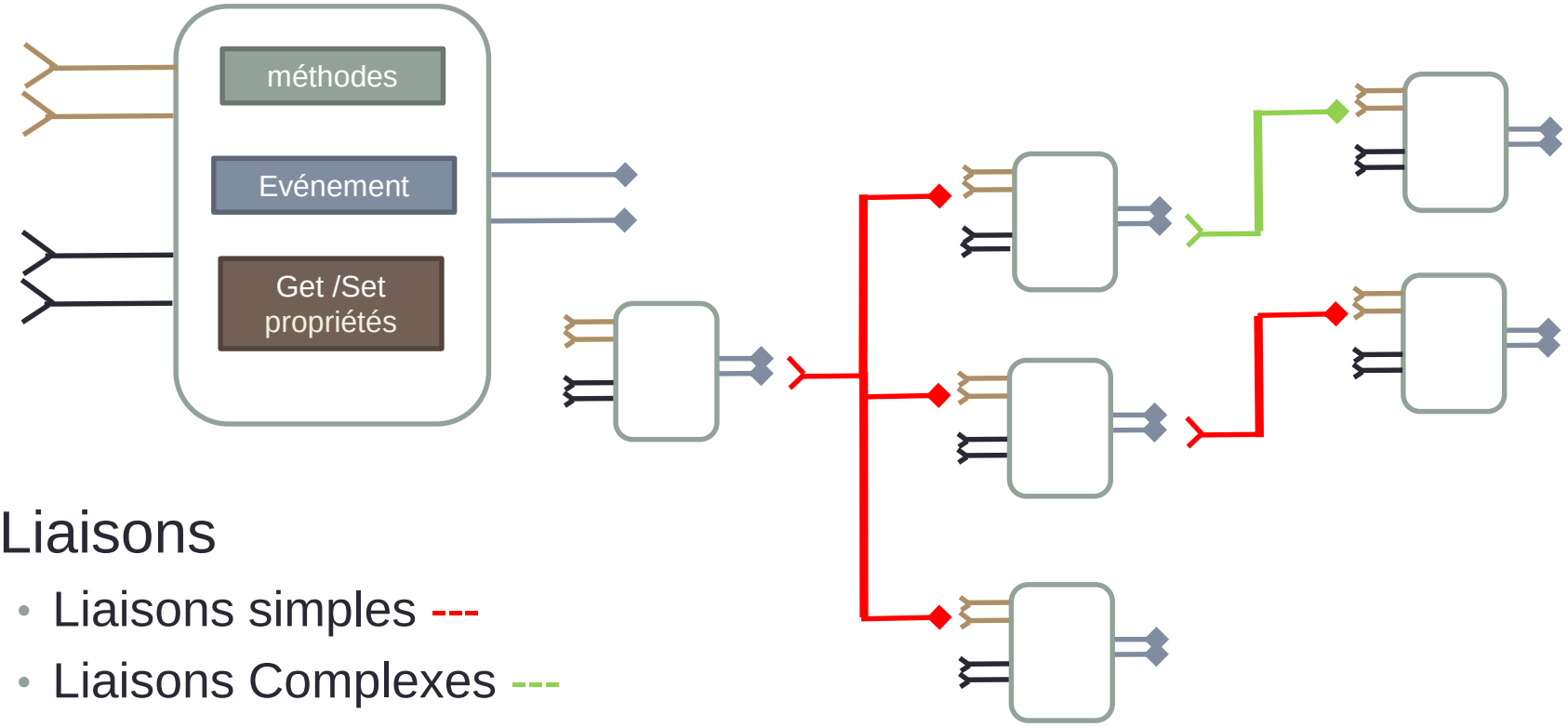
1. L'Application est un Assemblage de Composants légers
2. Composition par flots d'événements
3. Nœud d'exécution et distribution explicite

- Proche de la notion d'orchestration

Avec le middleware WComp : Des composants légers BeanWComp

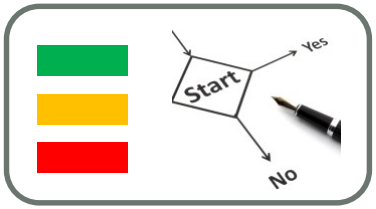


- Composant BeanWComp

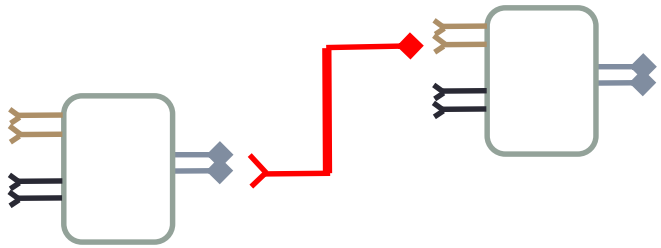


- Liaisons
  - Liaisons simples ---
  - Liaisons Complexes ---

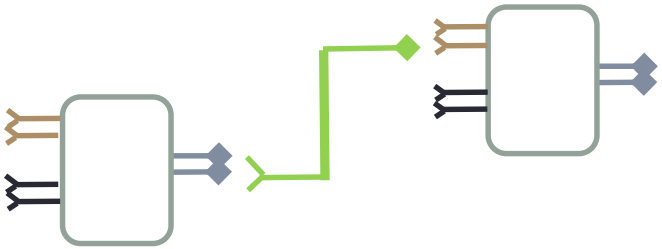
Avec le middleware WComp : Modèles de connecteurs : Event et Event Complexe



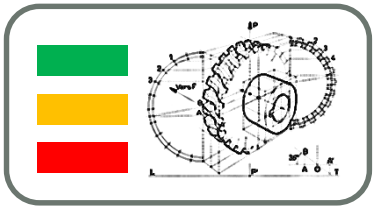
- Connecteur Event Simple
  - C1.Event (param) => C2.Methode (param)



- Connecteur Event Complexe
  - C1.Event (param) => C2.Methode (C1.GetProp())



Exemple de BeanWComp .Net



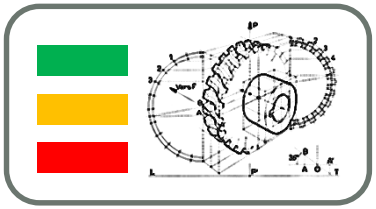
- Evénements sous C# basés sur le modèle du delegate

Attribut personnalisé

Evénement

```
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)
```

Exemple de BeanWComp .Net

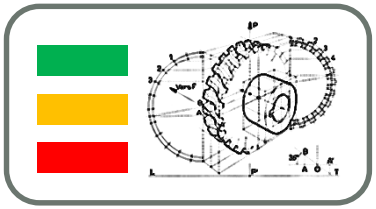


- Propriétés

Propriété

```
…

// 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;
                  }
            }
…
```

Exemple de BeanWComp .Net

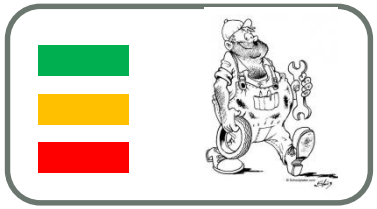


- Méthodes

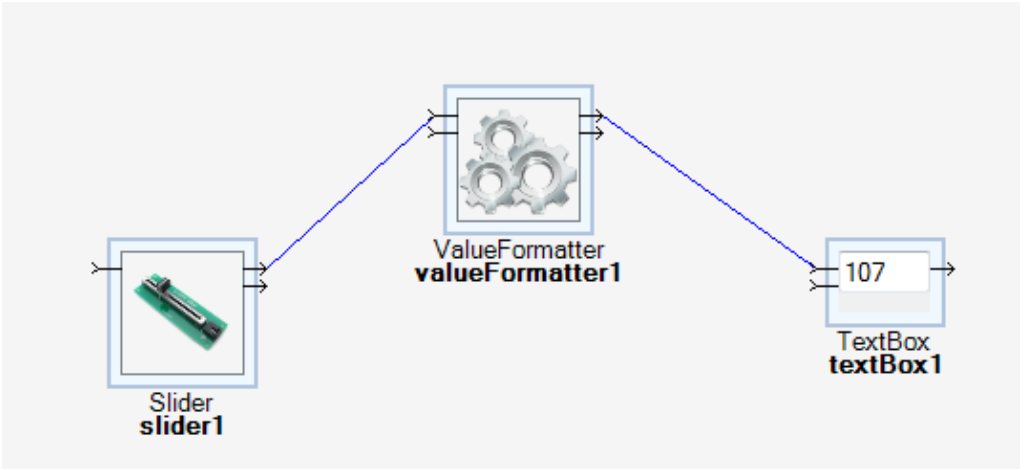
```
// méthodes

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

Méthode

Mise en oeuvre de WComp

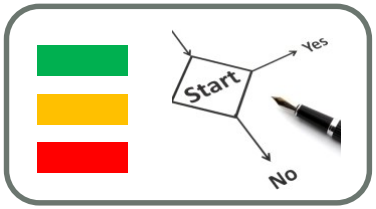


- PC (container SharpDevelop)
- Faire une application sous WComp (assemblage dynamique de composants BeanWComp)
- Créer un Bean WComp



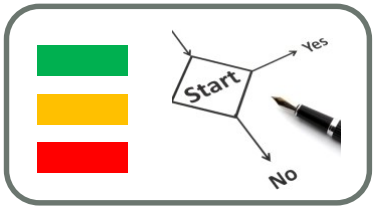
DEMO

Annexe 1 : Delegate et Event en C# / .Net



- Presque tout le monde connaît les événements.
- A chaque fois que vous cliquez sur un bouton dans une application Windows vous déclenchez un événement.
- Ici, le but est de créer des événements qui vont être propres à votre application. C'est-à-dire déclencher une action dans votre application lorsque quelque chose se produit et informer les objets abonnés à cet événement.

Annexe 1 : Delegate et Event en C# / .Net



- Notre événement va nous servir à afficher un message toutes les secondes. Nous allons déclarer la classe comme ceci :

- Langage C#
- using System;

- namespace TutoEvent
- {
- /// <summary>
- /// A la responsabilité de contenir le text de l'évenement et de le rendre accessible

- /// </summary>
- public class GenerateTextEventArgs : EventArgs
- {
- private string myEventText = null;

- public GenerateTextEventArgs(string theEventText)
- {
- if (theEventText == null) throw new NullReferenceException();
- myEventText = theEventText;
- }
-

- public string EventText
- {
- get { return this.myEventText; }
- }
- }
- }

Annexe 1 : Delegate et Event en C# / .Net

- http://freddyboy.developpez.com/dotnet/articles/events/
- Il n'est pas possible de parler des événements sans parler des delegate.
- Un delegate est un objet qui permet d'appeler une fonction ou une série de fonction.
- Un delegate est similaire aux pointeurs de fonctions du C/C++.
- Une variable delegate va permettre d'exécuter une fonction ou plusieurs fonctions.
- Pour cela le delegate va stocker des références sur des méthodes (que nous appellerons un gestionnaire d'évenéments ("Event handler")).

Annexe 1 : Delegate et Event en C# / .Net

- La signature des méthodes référencées devra respecter les règles suivantes :
  - retourner void
  - prendre comme premier paramètre un type object que nous appellerons généralement sender
  - prendre comme second paramètre un objet héritant de EventArgs, donc dans notre cas un objet GenerateTextEventArgs.

- Note : il est possible de voir des delegate déclarer autrement. Ici on montre simplement la méthode généralement utilisée pour les événements.

- Pour déclarer un delegate, nous utilisons la syntaxe suivante :

- public delegate void TextGeneratedEventHandler (object sender, GenerateTextEventArgs e);

Event

- Il nous faut ensuite déclarer un objet event du type du delegate déclaré plus haut.
- Le mot clé event vous permet de spécifier un délégué à appeler lors de l'occurrence d'un certain événement dans votre code.
- Pour déclarer un event, nous utilisons la syntaxe suivante:
- public event TextGeneratedEventHandler OnTextChanged;

Générer l'événement

- Pour générer un événement il suffit d'appeler son constructeur avec les paramètres éventuels comme ceci :

```
GenerateTextEventArgs e =
new GenerateTextEventArgs("Compteur = " + i.ToString());
```

- Puis il nous reste à envoyer cet événement à tout le monde :

- Langage C#
- if (e != null) OnTextChanged(this,e);

Classe complète

```
using System;
using System.Threading;

namespace TutoEvent
{
		/// <summary>
		/// A la responsabilité d'envoyer un evenement GenerateTextEvent toutes les secondes

		/// </summary>
		public class GenerateText
		{
				/// <summary>
				/// Declare un delegate
				/// </summary>
				public delegate void TextGeneratedEventHandler(object sender, GenerateTextEventArgs e);
				/// <summary>
				/// Declare un evenement qui va contenir les informations que nous souhaitons envoyer
				/// </summary>
				public event TextGeneratedEventHandler OnTextChanged;

				public GenerateText(){}

				public void Start(int theNumber)
				{
						int i = 0;
						while (i  < theNumber)
						{
								GenerateTextEventArgs e = new GenerateTextEventArgs("Compteur = " +
i.ToString());

								if (e != null) OnTextChanged(this,e);
								Thread.Sleep(1000);
								i++;
						}
				}
		}
}
```

Récupérer un événement dans un gestionnaire d'événements

- Un gestionnaire d'évenéments ("Event Handler") est la méthode qui va s'exécuter en réponse à l'événement.

- Un Event hanlder retourne normalement void et accepte 2 paramètres qui sont :
  - le sender : l'objet dans lequel l'événement s'est produit.
  - Un argument de type EventArgs qui contient les informations relatives à l'événement.

  Pour récupérer un événement, la première chose à faire est de se placer à l'écoute de cet événement. C'est là que le delegate que nous avons déclaré plus haut trouve toute son utilité.