2009

Démonstration : Les Aspects d'Assemblage dans WComp

S. Lavirotte, J.-Y. Tigli, N. Ferry , G. Rey

Université de Nice – Sophia Antipolis

01/01/2009

TD

Les Aspects d'Assemblages dans WComp

1 Les Aspects d'Assemblage

Le concept d'Aspect d'Assemblage (AA) a été introduit dans la thèse Daniel Cheung, soutenue en mars 2009. Vous trouverez les détails des notions présentées ici dans le mémoire de cette thèse de Daniel Cheung.

> **Daniel Cheung-Foo-Wo** « *Adaptation Dynamique par Tissage d'Aspects d'Assemblage* », Phd Thesis, Université de Nice - Sophia Antipolis, 223 pages, mars 2009.

Le concept d'Aspect d'Assemblage se définit en trois volets :

1. Le point de coupe ou l'expression des règles de recherche des points de jonction dans l'assemblage (ici des composants) où l'aspect d'assemblage peut s'appliquer,
2. Le greffon ou la fabrique des assemblages qu'on introduit,
3. Le tisseur ou la méthode pour composer les assemblages introduits dans l'assemblage existant.

L'outil utilisé pour le tissage d'Aspects d' Assemblage est un Designer au sens WComp, indépendant donc des Conteneurs des Assemblages WComp.

2 Mon tout premier Aspect d'Assemblage

2.1 Définition d'un Aspect d'Assemblage

Un Aspect d'Assemblage est donc composé d'un point de coupe et d'un greffon.

Pour le point de coupe, il s'agit de règles basées sur des expressions régulières et plus particulièrement des expressions en AWK.

```
(<composant>|<événement>|<méthode>) = <expression régulière>
```

Pour le Greffon il s'agit d'un ensemble de règles de la forme :

```
<point de jonction> -> <description du comportement>
```

où le comportement est décrit à l'aide d'un langage, comme par exemple ISL4WComp qui sera utilisé dans la suite de cette présentation. La correspondance entre opérateurs du langage et composants prédéfinies de sémantique connue permet donc une transformation rapide entre la description du comportement et le sous-assemblage correspondant.

Les opérateurs utilisés dans nos exemples sont les suivants :

Opérateurs	Description
;	exprime une séquence
+	exprime un ordre indifférent entre les comportements
only	exprime l'unicité de l'appel à une méthode
if ... else ...	exprime une condition
call	désigne l'événement ou l'appel du point de jonction
delegate	désigne l'émission d'un événement ou un appel
^	exprime l'émission d'un événement

Pour écrire un AA il faut respecter certaines conditions syntaxiques.

- **La première lettre du nom d'une méthode doit toujours être en majuscule.**
- **Le nom des composants instanciés doit être en minuscule**

- **L'ordre de déclaration des points de coupes doit être le même que celui de déclaration des points de jonction**

2.2 Ecriture du premier Aspect d'Assemblage

Etant donné ces règles, écrivez un aspect d'assemblage vous permettant d'allumer une lampe avec un interrupteur.

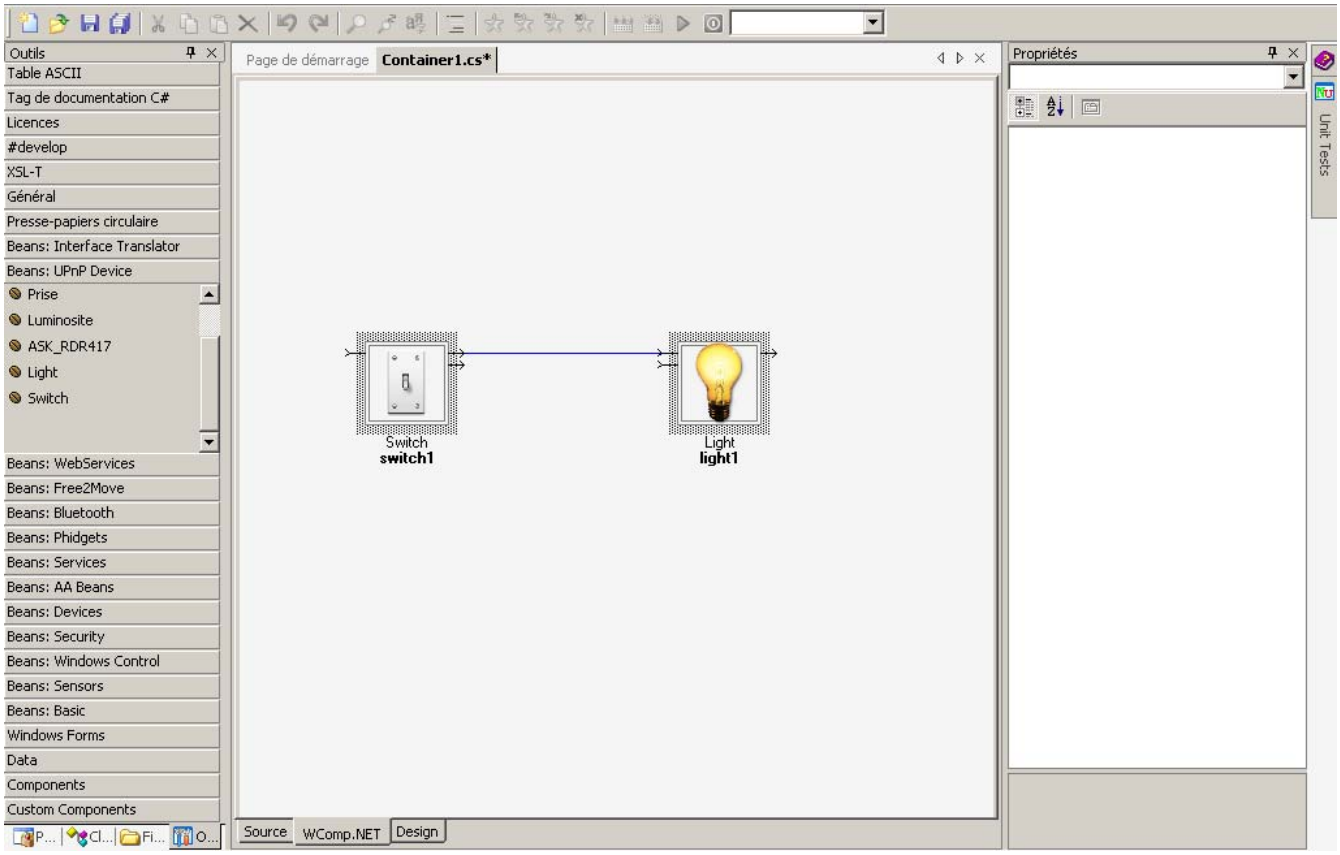
Point de Coupe:

```
emetter:=/switch*/
receiver:=/light*/
```

Greffon:

```
schema switch_wcomp(emetter, receiver):

emetter.^Status_Evented_NewValue -> (
      receiver.SetStatus
)
```

Vérifiez son application en présence des points de jonctions `switch1` et `light1`



3 Effet du Point de Coupe de l'Aspect d'Assemblage

Etudions maintenant plus en détails les variantes possibles sur le point de coupe. En jouant sur les expressions régulières nous pouvons sélectionner plus finement les multiples points de jonction résultants des règles de point de coupe et ainsi gérer au mieux les multiples applications d'un AA.

TD
Les Aspects d'Assemblages dans WComp

Ecrivez un aspect d'assemblage qui permettra de reconnaître alternativement les valeurs `1,2,3,4` et ainsi de sélectionner 4 couples de point de jonction `(button1,checkBox1)`, `(button2, checkBox2)`, `(button3, checkBox3)`, `(button4, checkBox4)` à l'aide du filtre [[:digit:]]. L'appui sur un de ces boutons cochera la checkBox

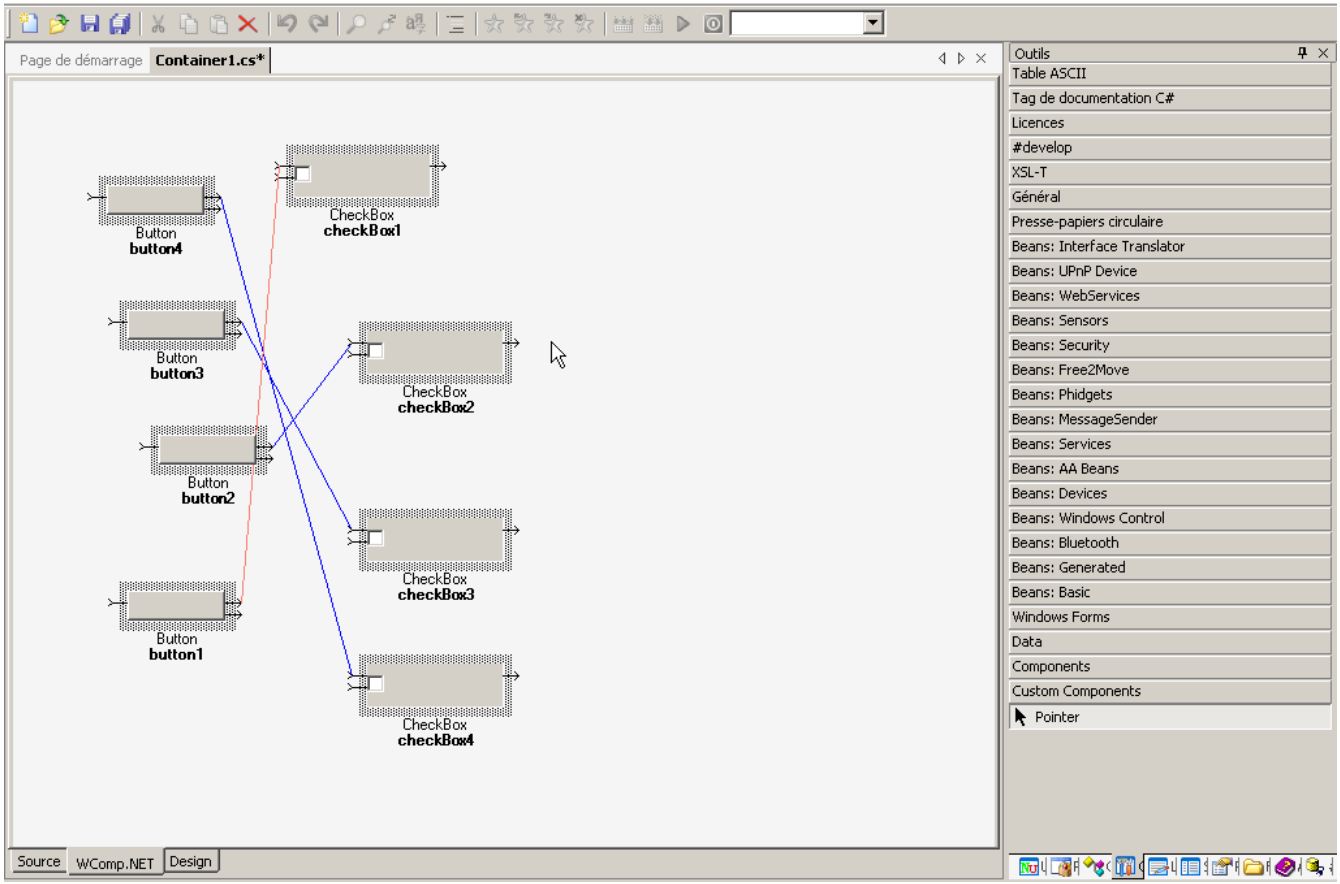
Point de Coupe :

```
but:=/button[[:digit:]]//
box:=/checkBox[[:digit:]]/
```

Greffon:

```
schema switch_wcomp(but,box):

box.^Click__get_Enabled -> (
      but.set_Checked
)
```

Vérifiez l'application multiple de cet AA.



TD

Les Aspects d'Assemblages dans WComp

4 Effet du Greffon de l'AA

Etudions maintenant plus en détails des greffons plus élaborés. En jouant sur le nombre de règles et des descriptions de comportement, les AAs sont un moyen de préparer un grand nombre de modifications ou d'adaptations d'assemblages WComp avec un haut niveau d'expressivité.

4.1 Syntaxe ISL4WComp

- ***Démo 5 : Etudions un exemple composé de 3 règles et instanciant de nouveaux composants.***

```
schema switch_wcomp2(switch,lum,rfid):

ident : 'WComp.Beans.RFiDId' ;
t1 : 'System.Windows.Forms.CheckBox' ;
t2 : 'System.Windows.Forms.TextBox' ;
activation : 'System.Windows.Forms.Button' ;

lum.SetStatus -> (
      if(ident.IsAccessGranted) {call} else {t1.set_Checked}
)

activation.^Click -> (
      rfid.EnterHuntPhase
)

rfid.^TagInfo_Evented_NewValue -> (
      (t2.set_Text ; ident.set_ident)
)
```

4.2 Membre gauche type event

- ***Démo 5 : Etudions plus particulièrement le cas d'un membre gauche de type event***

```
activation.^Click -> (
      rfid.EnterHuntPhase
)
```

4.3 Membre gauche type appel de méthode

- ***Démo 5 : Etudions plus particulièrement le cas d'un membre gauche de type méthode***

```
lum.SetStatus -> (
      if(ident.IsAccessGranted) {call} else {t1.set_Checked}
)
```

4.4 Quelques opérateurs (composants instanciables de sémantique connue)

- ***Démo 5 : Etudions plus particulièrement le cas de l'utilisation d'opérateurs if … else, call et ;***

```
lum.SetStatus -> (
      if(ident.IsAccessGranted) {call} else {t1.set_Checked}
)

rfid.^TagInfo_Evented_NewValue -> (
```

```
      (t2.set_Text ; ident.set_ident)
)
```

4.5 Des instances locales de composants (composants instanciables blackbox, sur étagère)

- ***Démo 5 : Etudions plus particulièrement le cas de l'instanciation de composant sur étagère (blackbox)***

```
ident : 'WComp.Beans.RFiDId' ;
t1    : 'System.Windows.Forms.CheckBox' ;
t2    : 'System.Windows.Forms.TextBox' ;
```

5 Déclenchement du tissage d'AA

Le tissage des aspects dans les approches classiques AOP est souvent provoqué par le développeur pour provoquer des modifications transverses de son application logicielle. Le mécanisme de tissage d'Aspects d'Assemblage est lui déclenché à chaque modification des assemblages cibles (dans les conteneurs WComp concernés). En conséquence les AA peuvent être mis en œuvre selon deux modes.

- Dans un premier mode dit « a posteriori », l'AA sélectionné pourra s'appliqué sur les point de jonctions alors identifiés. Il s'agit d'un mode proche des approches classiques et « user–driven ».
- A contrario, dans un mode dit « a priori », un AA peut-être sélectionné mais non applicable en l'absence de point de jonction correspondant. Dans ce cas l'apparition d'un service pour dispositif dans l'infrastructure et du composant proxy correspondant, peut fournir a posteriori le point de jonction nécessaire à l'application de l'AA. Il s'agit alors d'un mode réactif et « context–driven ».

- ***Démo 6 : Exemple d'application d'un AA « a posteriori »***
- ***Démo 7 : Exemple d'application d'un AA « a priori »***

6 Gestion d'application des greffons sur des points de jonction communs

Nous allons présenter enfin le problème de l'application simultanée et multiple de greffons sur des points de jonction communs.

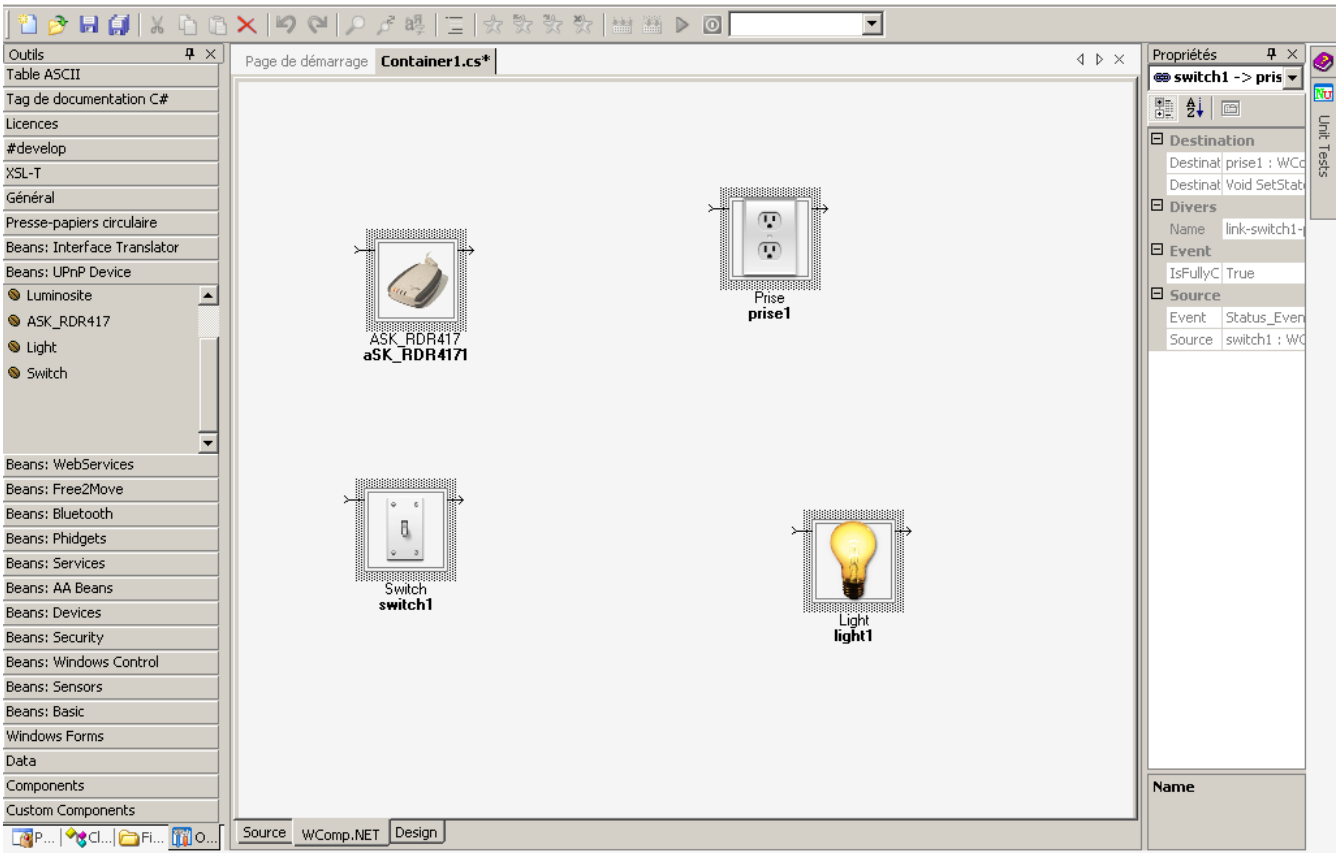
De multiples stratégies de composition sont alors possibles. A l'instar des approches AOP classiques, des ordonnancements entre greffons « en conflit » seraient envisageables basés sur des mécanismes de type « before », « after », …etc.

L'approche utilisée ici s'appuie plutôt sur un mécanisme de fusion qui vise à définir un greffon unique comme le résultat de la composition des greffons « en conflit ». Un mécanisme de règles de composition symétrique, (vérifiant donc les propriétés comme la commutativité, l'associativité et l'idempotence) permet alors d'obtenir un greffon résultant, qui sera finalement appliqué et qui ne dépend pas de l'ordre des greffons « en conflit ».

- ***Démo 8 : La suite des exemples repose sur ces composants proxy et donc sur la présence des services pour dispositifs correspondants dans l'infrastructure WComp.***

L'assemblage de départ :

TD
Les Aspects d'Assemblages dans WComp



6.1 Exemple Simple

- ***Démo 9 : Mettons les deux AA suivants en conflit et vérifions le résultat de la fusion dans l'apparition du composant « PAR ».***

Aspect d'assemblage 1 :

Point de coupe :

```
switch:=/switch.*/
lum:=/light1/
```

Greffon :

```
schema switch_wcomp(switch,lum):

switch.^Status_Evented_NewValue -> (
      lum.SetStatus
)
```

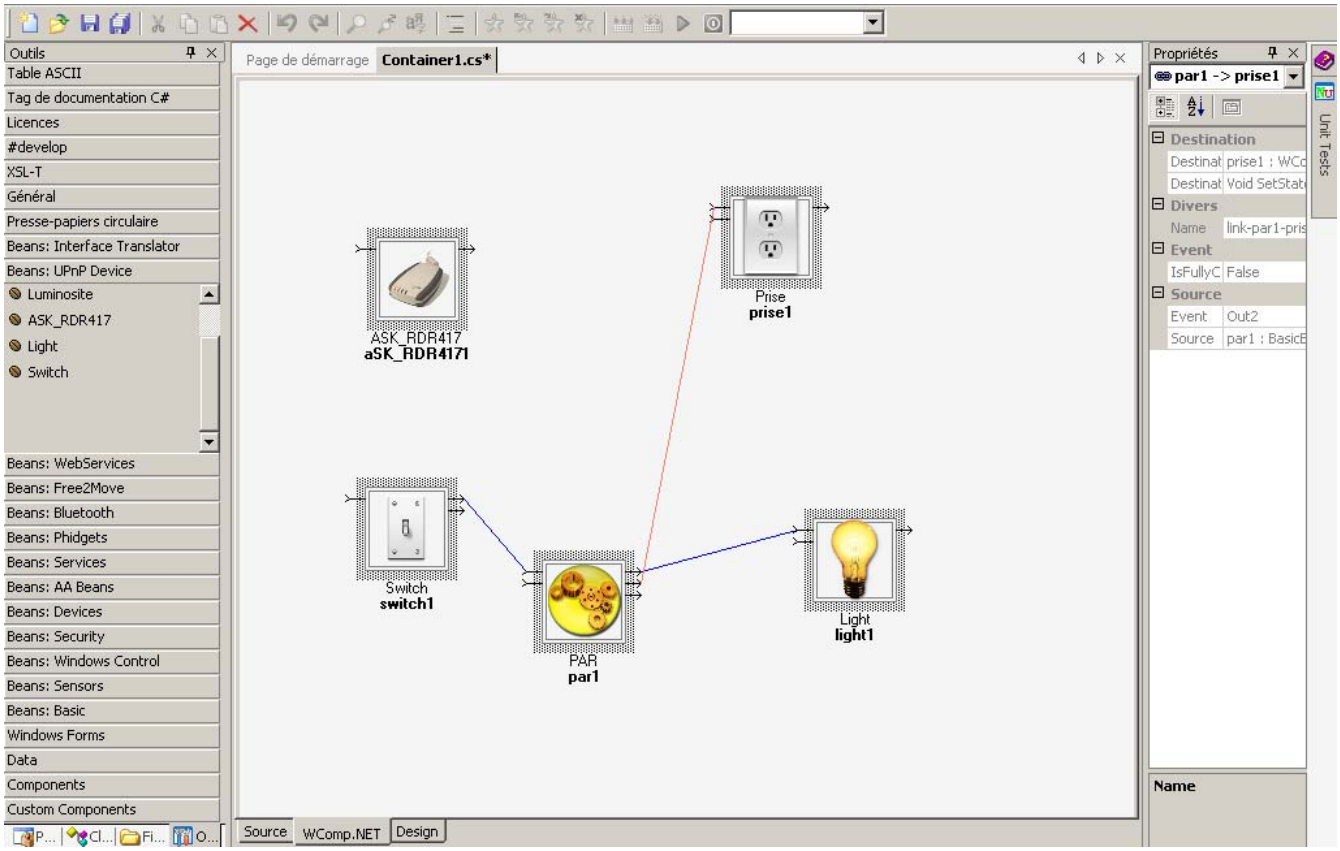
Aspect d'assemblage 2 :

Point de coupe :

```
switch:=/switch[[:digit:]]/
lum:=/light2/
```

Greffon :

```
schema switch_wcomp_2(switch,lum):

switch.^Status_Evented_NewValue -> (
      lum.SetStatus
)
```



6.2 Exemple Complet

- ***Démo 10 : Appliquons maintenant un exemple complet et vérifions le comportement résultant de l'application***

Point de coupe :

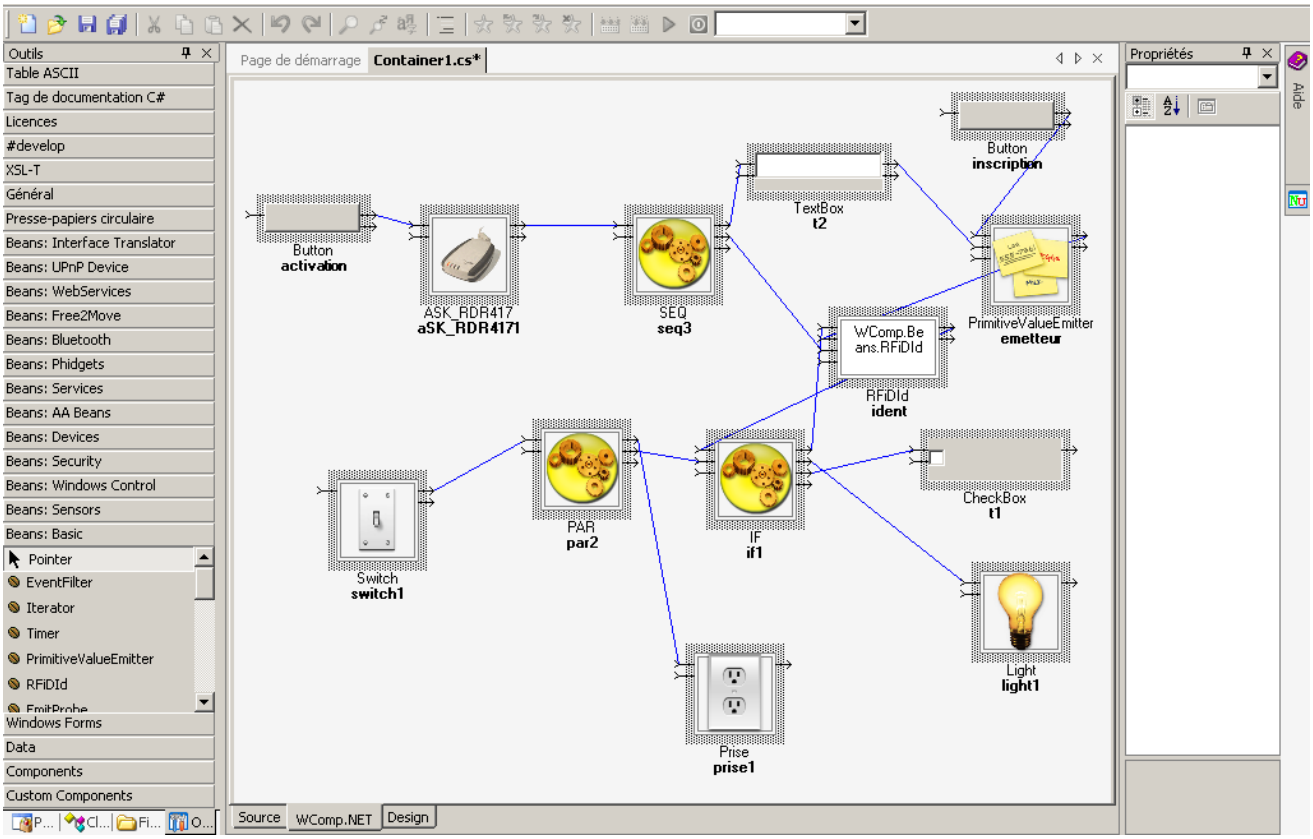
```
switch:=/switch[[:digit:]]/
lum:=/light1/
rfid:=/aSK.*/
```

Greffon :

```
schema switch_wcomp2(switch,lum,rfid):

ident : 'WComp.Beans.RFiDId' ;
t1 : 'System.Windows.Forms.CheckBox' ;
t2 : 'System.Windows.Forms.TextBox' ;
activation : 'System.Windows.Forms.Button' ;
emetteur : 'WComp.BasicBeans.PrimitiveValueEmitter' ;
inscription : 'System.Windows.Forms.Button' ;
```

TD
Les Aspects d'Assemblages dans WComp

```
lum.SetStatus -> (
      if(ident.IsAccessGranted) {call} else {t1.set_Checked}
)

activation.^Click -> (
      rfid.EnterHuntPhase
)

rfid.^TagInfo_Evented_NewValue -> (
      (t2.set_Text ; ident.set_ident)
)

t2.^TextChanged__get_Text -> (
      emetteur.set_StringValue
)

inscription.^Click -> (
      emetteur.FireValueEvent
)

emetteur.^EmitStringValue -> (
      ident.AddAuth
)
```



TD
Les Aspects d'Assemblages dans WComp

7 Un exemple plus complet

Vous allez maintenant écrire votre propre aspect d'assemblage qui réutilisera le composant Blink (développé dans le tutoriel sur la composition avec WComp). Il va s'agir en fait d'écrire un AA complémentaire à celui que vous avez écrit tout à l'heure mais qui ajouterai un comportement à notre application. Désormais ouvrir l'interrupteur fera clignoter la lampe et le fermer l'éteindra. Pour ce faire vous devrez utiliser l'opérateur `only`.

Greffon :

```
schema blink(switch,receiver):

blinker : 'Wcomp.Beans.Blink' ;

switch.^Status_Evented_NewValue -> (
      only(blinker.MethodBlinking)
)

blinker.^Blinking -> (
      receiver.SetStatus
)
```

Testez votre AA en le sélectionnant ainsi que celui vous permettant d'allumer et d'éteindre simplement votre lampe.