Université Nice Sophia Antipolis

Dream IT
Innovation & Technologies
http://www.dreamit-fr.org/

U-Health

Document :	*Préparation à la mise en place du démonstrateur*
Sous-tâche :	2.1
Numéro des Livrables :	D2.1
Date	13/12/2013
Rédacteurs :	*Rebecca Poustis (Polytech), Christian Brel (I3S), Stéphane Lavirotte (I3S-Polytech), Jean-Yves Tigli (I3S-Polytech)*
Coordinateurs :	Stéphane Lavirotte

Sommaire

1 Introduction

L'objectif de ce livrable est de définir l'application qui sera construite à partir des objets communicants développés dans les phases précédentes du projet. Les différents objets créés s'appuient sur WComp, un environnement de prototypage rapide orienté composants. Cet environnement nous a permis d'augmenter des objets de la vie quotidienne avec des capteurs permettant à ces objets de répondre non seulement à leur usage intrinsèque mais aussi de proposer la capture d'autres informations de l'environnement. Ces objets fonctionnent effectivement en totale autonomie, proposent un service associé à leur état d'objet, mais ne proposent pas en soi un service/une application de plus haut niveau. Pour cela, nous devons définir des règles permettant de lier *n* objets entre eux et permettant l'application de ces règles de manière dynamique en fonction de l'apparition et la disparition des objets dans l'environnement.

2 Scénario envisagé

Le scénario suivant a été établi afin de répondre à une problématique de prévention de trouble du sommeil, type apnée du sommeil. Ce scénario fait intervenir les objets et l'application mobile, développés dans les phases précédentes du projet, de manière incrémentale, afin de bien définir les différentes règles au fur et à mesure.

Le scénario envisagé est donc le suivant :

1. Une personne achète un matelas et l'installe dans sa chambre. Elle se couche et le matelas s'interconnecte à tous les services de stockage d'informations fournis (localement ou dans le cloud grâce à l'intégration des objets eux-mêmes dans le cloud comme décrit dans le livrable 1.3). On peut aussi capturer les informations d'utilisation du matelas (temps couché, assis, ...) et surveiller ses mouvements sur le lit durant son sommeil.
2. Quelques jours plus tard, la personne va chez Ikea et revient avec un nouveau dispositif : une lampe. On va pouvoir surveiller plus d'informations puisque la lampe inclut un capteur sonore. Ce dispositif est interconnecté aux services de stockage et de visualisation des informations mais les objets ne sont pas interconnectés entre eux.
3. Elle va ensuite à la Fnac et s'achète un téléphone. De retour à la maison, celui-ci s'interconnecte avec la lampe. Il se produit alors une adaptation entre son téléphone et son environnement. La personne allume l'application « Réveil UPnP » et met en route une alarme ce qui éteint la lampe après 30 secondes d'attente.
4. Puis elle retourne à la Fnac et ajoute un chargeur à son environnement. On peut surveiller alors plus de paramètres car le chargeur contient un capteur de qualité d'air. Pour utiliser ce nouveau dispositif dans son environnement il est interconnecté à l'infrastructure existante. Lorsqu'elle branche son téléphone sur le chargeur, la lampe alors allumée, s'éteint pour permettre au chargeur lui même de s'illuminer. Une fois l'activation de l'alarme, ce n'est plus la lampe qui est éteinte mais le chargeur après 30 secondes d'attente.

Afin de répondre au besoin de monitoring des différentes données, nous avons développé différentes applications que nous décrivons dans la section suivante.

3 Applications interactives

Différentes applications de monitoring de données, exposants elles-mêmes un service UPnP, ont été développées dans le cadre du projet. Ces applications permettent, suivant le service proposé, de prendre des données en entrée, puis de les traiter afin de proposer un retour visuel et pertinent à l'utilisateur. Ces applications peuvent, à la manière des objets, être exécutées en local ou dans le cloud. Elles utilisent parfois des services externes pour assurer certaines de leurs fonctionnalités, services eux même disponibles dans cette architecture dans le cloud.
Le code source des ces différentes applications peut être téléchargé à l'adresse suivante : https://uhealth.polytech.unice.fr/codes

3.1 Bed Monitoring

La première application nommé BedMonitoring (c.f. Figure 1) permet l'affichage de la présence et de l'immobilité de la personne et les données sont monitorées dans une fenêtre en temps réel. Ceci permet de montrer que l'on peut recevoir et traiter toutes les données reçues.

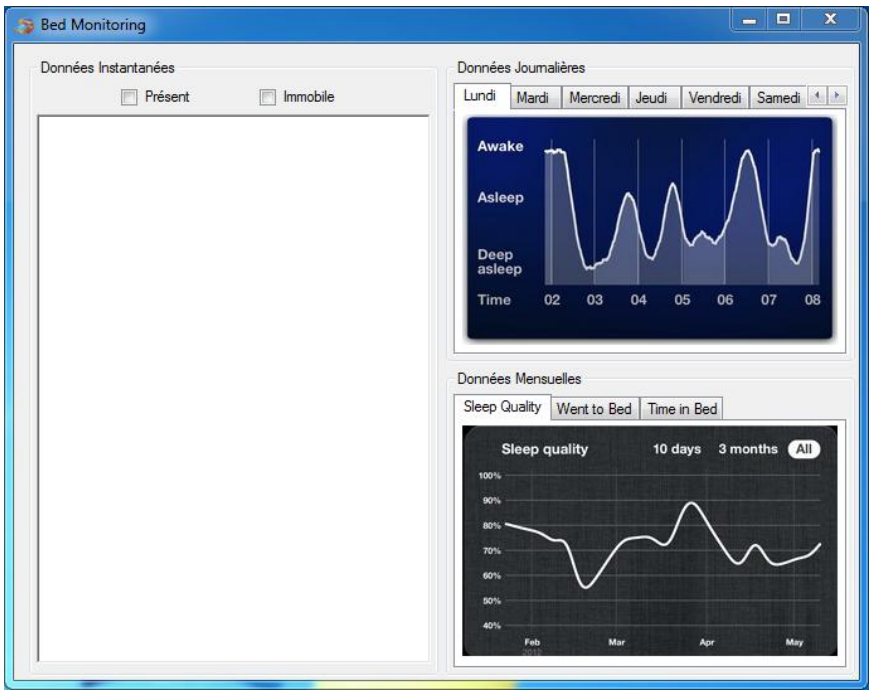


Figure 1 : Interface graphique de l'application BedMonitoring

3.2 Plotter

La seconde application, nommée Plotter (c.f. Figure 2) permet de dessiner une courbe par donnée reçue de manière générique pour ainsi pouvoir afficher toutes les courbes dont nous avons besoin.

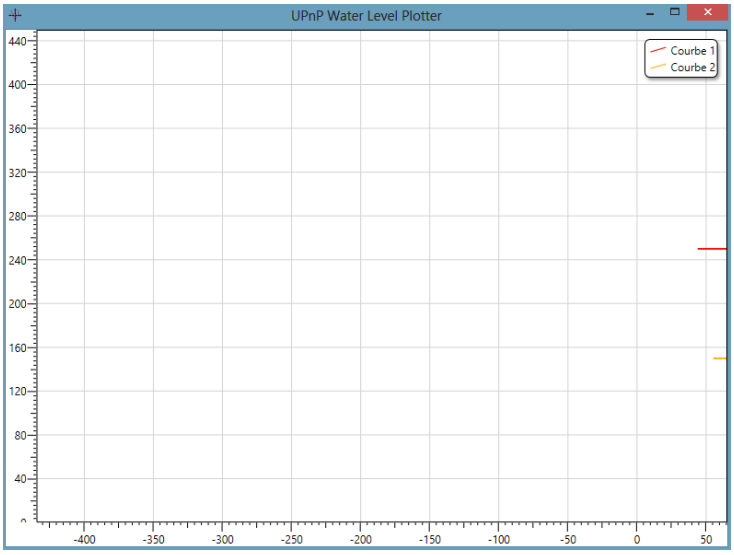


Figure 2 : Interface graphique de l'application Plotter

3.3 CVML

La troisième application, nommée CVML (c.f. Figure 3), a été développée à partir d'une étude de données faite par une autre équipe de recherche spécialisée en fouille de données. Grâce à un enregistrement des données renvoyées par les capteurs dans un fichier .csv, un apprentissage a pu être réalisé concernant la position du corps d'une personne sur le matelas. Cette étude a permis une meilleure disposition des capteurs afin d'obtenir le meilleur résultat concernant la position.

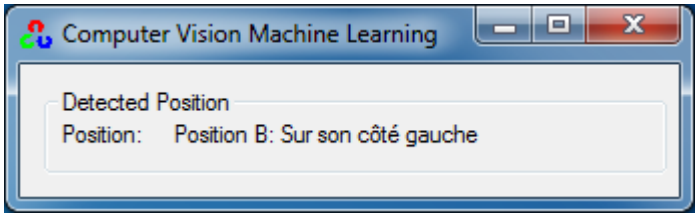


Figure 3 : Interface graphique de l'application CVML

3.4 AirControl

L'application Plotter nous permettait de visualiser les valeurs des différents capteurs mais cette solution n'était pas vraiment adaptée pour un utilisateur. Une application dédiée, nommée AirControl (c.f. Figure 4), a alors été développée afin d'afficher les différentes valeurs provenant de capteurs de température, d'humidité, de luminosité, et de qualité d'air. Dans ce service, en fonction de la valeur du taux de CO2, un smiley vert, orange ou rouge apparaît selon les normes de l'OMS. Des recherches sur les normes de la qualité de l'air ont donc été faites pour pouvoir introduire cet objet de la meilleure façon qu'il soit dans le scénario.

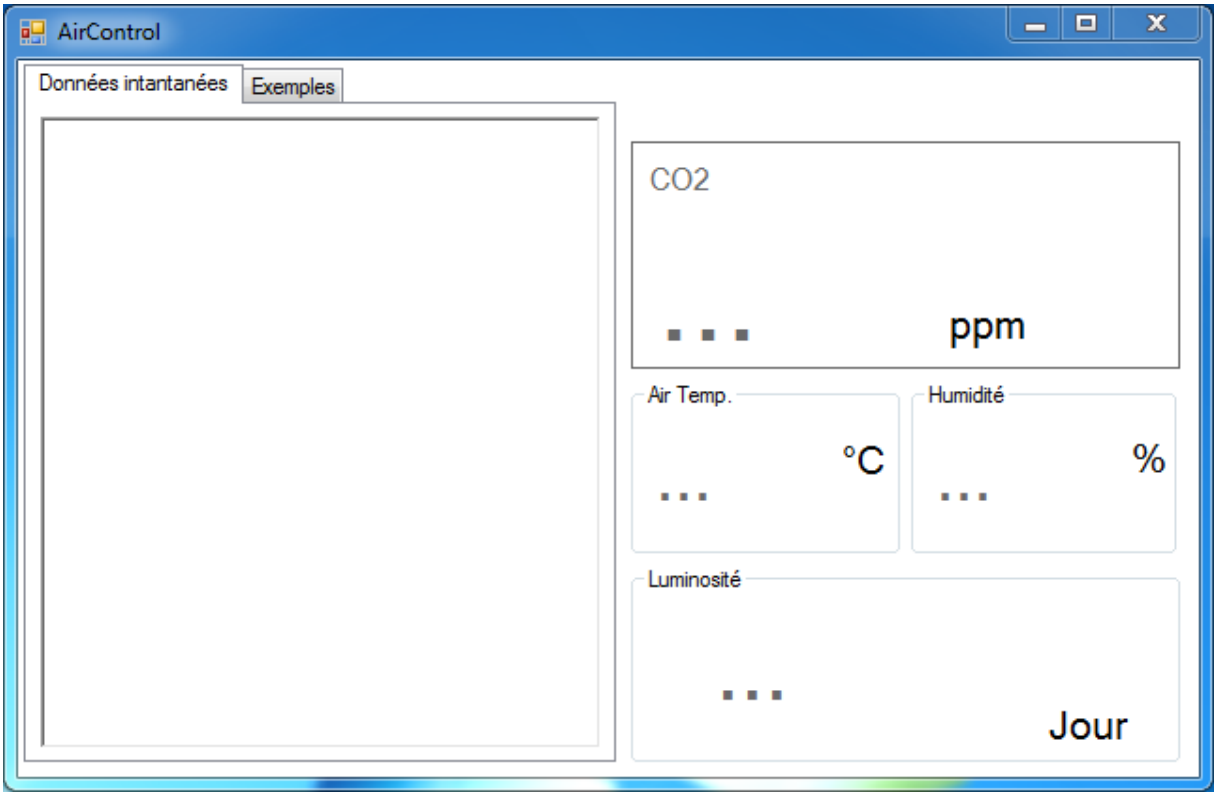


Figure 4 : Interface graphique de l'application AirControl

4 Architecture de l'application mise en place

Le scénario décrit précédemment permet de mettre en place une application intégrant les différents objets. En fonction de l'état des objets, l'application s'adapte afin de répondre aux différentes parties du scénario. Le scénario est traduit dans la plateforme WComp par des Aspects d'Assemblages, présentés dans la section suivante, permettant de tenir compte à la fois des services disponibles dans l'environnement et de l'apparition et disparition des différents objets.

L'architecture de l'application représentée sur la Figure 5 est donc composée de 3 parties :

- un ensemble d'applications interactives permettant d'effectuer entre autre du monitoring. Ces applications exposent un service UPnP donnant la possibilité d'être connecté à d'autres services par le biais d'un assemblage. Nous retrouvons dans cet ensemble les applications dédiées décrites précédemment. Ces applications peuvent être exécutées dans un cloud ;
- un ensemble de services dont certains utilisés directement par les applications interactives qui permettent un certain support de calculs comme l'utilisation d'OpenCV permettant la fouille de données pour obtenir les positions de la personne sur le matelas. Ces différents services peuvent eux-mêmes être disponibles dans un cloud ;
- un ensemble d'objets communicants qui exposent aussi un service UPnP sur lesquels va s'appuyer l'application afin de répondre au scénario décrit précédemment, objets pouvant être intégrés à un cloud comme décrit dans le livrable D1.3.

Ces 3 ensembles sont disponibles dans l'environnement formant un cloud plus important regroupant des objets physiques, des services externes et des applications dédiées.

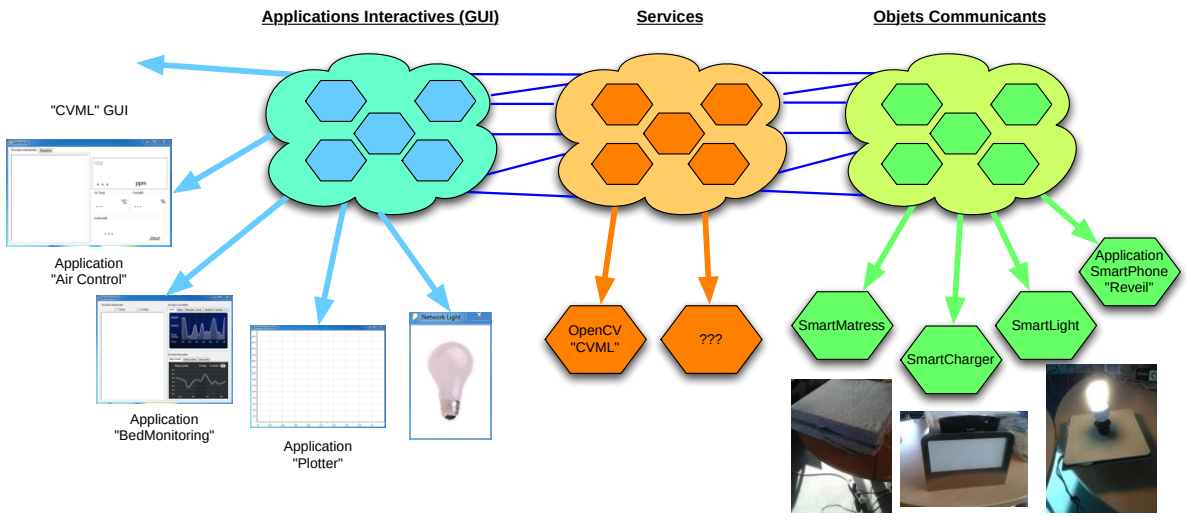


Figure 5 : Schéma de l'architecture de l'application sous la forme d'un cloud d'objets, d'un cloud de services et d'un cloud d'applications interactives

5 Réalisation du scénario

L'architecture de l'application étant définis par les 3 ensembles décrits précédemment. Cette étape est finale pour mettre en œuvre le scénario décrit.

5.1 Les Aspect d'Assemblage - AAs

Les Aspects d'Assemblage (AAs) permettent de reconfigurer de manière indépendante, dynamique des applications définies à partir de composants logiciels. Une application s'auto-adapte alors par tissage des aspects d'assemblage présélectionnés et qui sont applicables en fonction des services présents dans le contexte.

L'outil utilisé pour le tissage d'Aspects d'Assemblage est un designer graphique d'architecture de WComp comme expliqué dans le livrable D1.3.

Un aspect d'assemblage est composé d'un point de coupe et d'un greffon. Un point de coupe est l'expression des règles de recherche des points de jonction dans l'assemblage créé là où l'aspect d'assemblage peut s'appliquer. Un greffon est la fabrique des assemblages qu'on introduit.

Un aspect d'assemblage a une syntaxe particulière. Pour le point de coupe, il s'agit de règles basées sur des expressions régulières :

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

Pour le greffon, il s'agit d'un ensemble de règles de cette forme :

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

Une fois le modèle d'AAs écrit dans ce langage, plusieurs outils sont mis en jeu. L'UPnP designer crée les composants proxy dans le container WComp correspondant aux services UPnP associés aux dispositifs visibles sur le réseau et détruit ceux qui disparaissent de ce même réseau. L'AA designer permet de sélectionner les AAs et tisse les AAs qui peuvent l'être en fonction du contexte (des objets et services présents). Ce concept de tissage est donc très important dans la création des AAs, ci-après quelques explications supplémentaires.

5.2 Le tissage

Le mécanisme de tissage d'aspects d'assemblage est déclenché à chaque apparition ou disparition de services (services, applications dédiées ou services pour les dispositifs dans le cas de notre scénario). En conséquence les AAs peuvent être mis en œuvre selon deux modes.

Le premier mode est dit « a posteriori », l'AA sélectionné pourra s'appliquer sur les points de jonctions alors identifiés. Le deuxième mode est « 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 le point de jonction nécessaire à l'apparition de l'AA.

5.3 Les AAs du scénario

Pour chaque partie du scénario, nous avons décrit des aspects d'assemblages que nos détaillons dans les sections suivantes. A l'aide du designer d'AAs et d'un container WComp, nous avons effectué le scénario tel qu'il a été décrit précédemment. Au fur et mesure de la découverte des services, le tissage des AAs s'est fait dans le container mettant en relation les différentes parties de l'architecture de notre application (c.f. Figure 6).

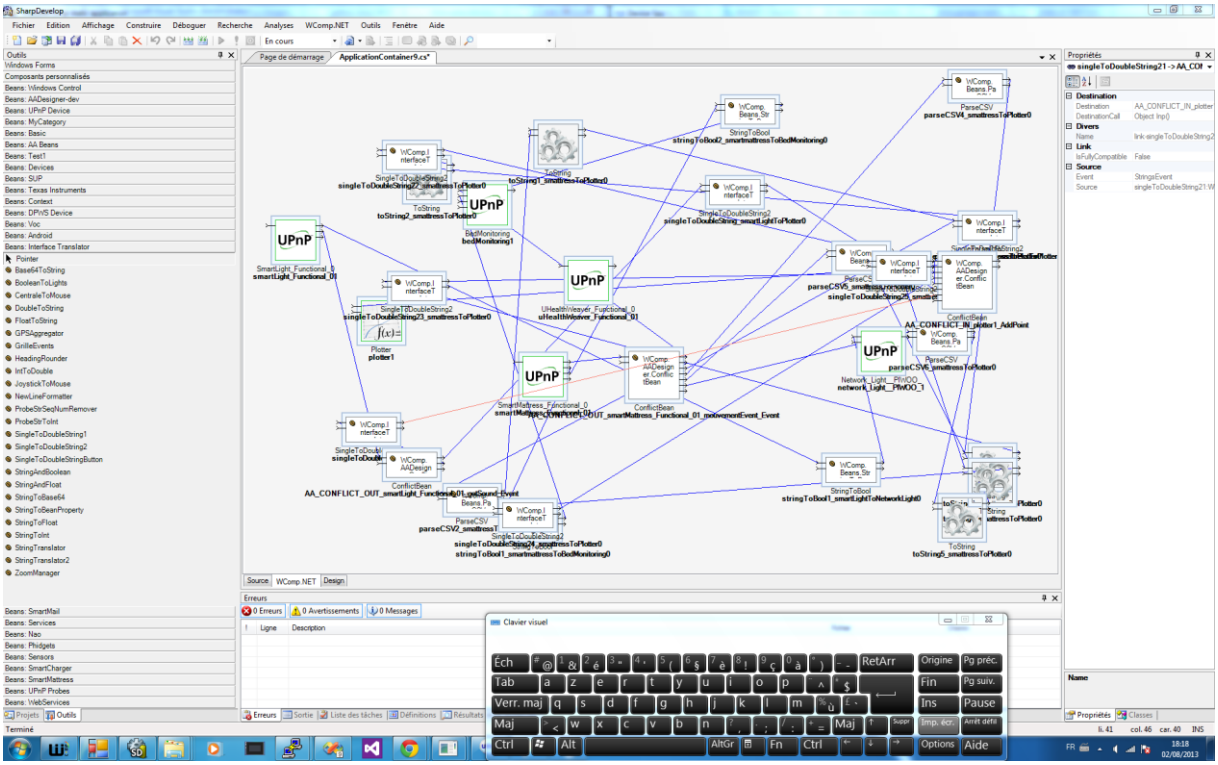
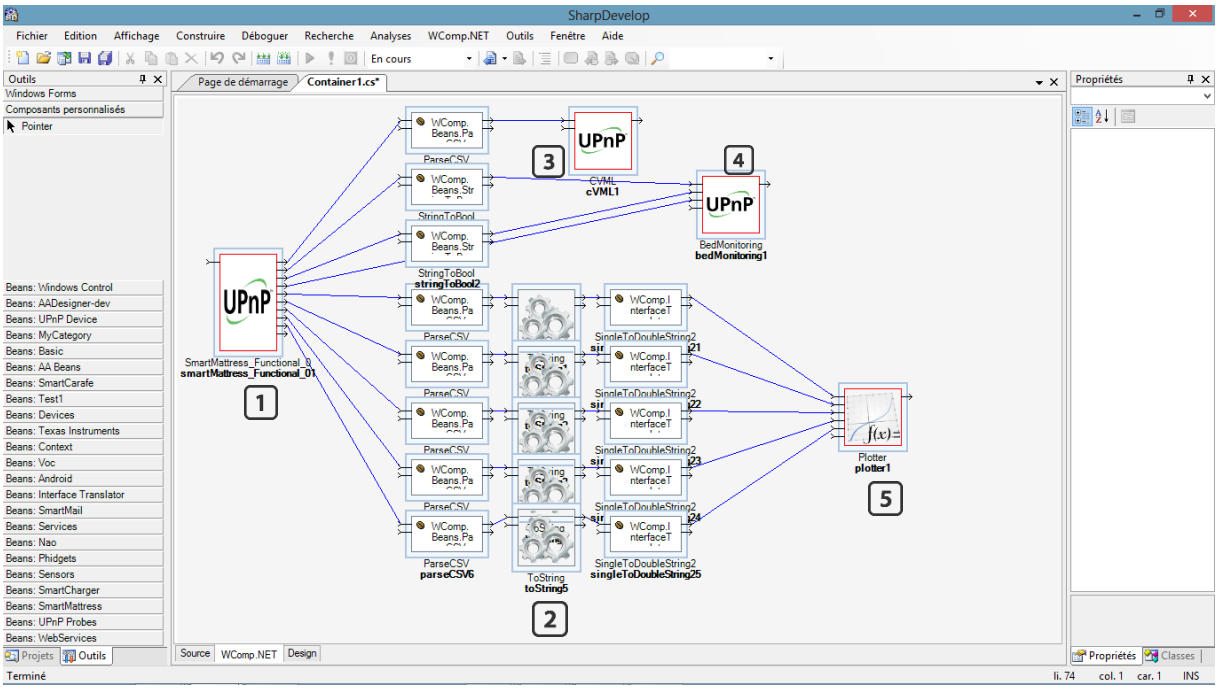


Figure 6 : Assemblage par tissage des Aspects d'Assemblage

5.3.1 Assemblage pour l'étape 1 du scénario mettant en jeu le SmartMattress ainsi que les applications CVML, BedMonitoring, Plotter



1 Représente le composant du proxy UPnP du SmartMattress

2 Ces trois colonnes sont tous les beans dont on a besoin pour trier et récupérer les données qui nous intéressent

3 Représente le composant graphique du proxy UPnP du logiciel CMVL (logiciel qui donne la position de la personne sur le matelas)

4 Représente le composant graphique du proxy UPnP du logiciel BedMonitoring

5 Représente le composant graphique du proxy UPnP du logiciel Plotter

Dans cet exemple, le matelas est interconnecté aux différents services.

5.3.2 Exemple de la syntaxe d'un AA

Afin de mettre en œuvre l'assemblage pour l'étape 1 du scénario, nous l'avons découpé en plusieurs parties dont chacune est traduite par un Aspect d'Assemblage. Un premier découpage de l'étape 1 est la connexion entre le SmartMattress et l'application BedMoniroting (c.f.Figure 7)

*smartmattress = smartMattress**
*bedMonitoring = bedMonitoring**
advice smartmattressToBedMonitoring(smartmattress,bedMonitoring) :
stringToBool1 : WComp.Beans.StringToBool
stringToBool2 : WComp.Beans.StringToBool
smartmattress.^estImmobileEvent_Event -> (stringToBool1.set_MyProperty)
stringToBool1.^PropertyChanged -> (bedMonitoring.SetImmobile)

smartmattress.^estPresentEvent_Event -> (stringToBool2.set_MyProperty)
stringToBool2.^PropertyChanged -> (bedMonitoring.SetPresent)
smartmattress.^mouvementEvent_Event -> (bedMonitoring.SetText)

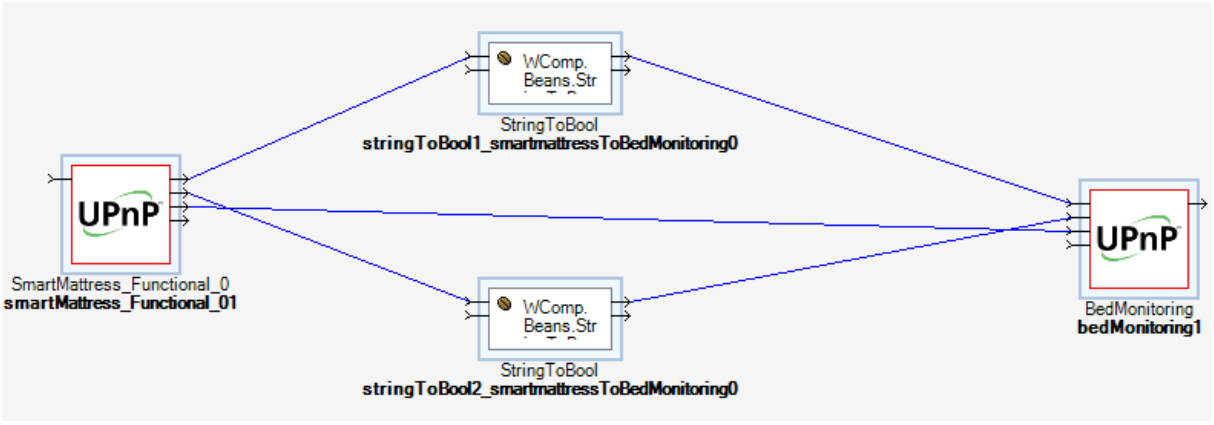


Figure 7 : Résultat de l'Aspect d'Assemblage entre le SmartMattress et l'application BedMonitoring

Cet Aspect d'Assemblage décrit l'assemblage du SmartMattress avec le service BedMonitoring.
Cet aspect d'assemblage se décompose en deux parties : une première partie correspondant au point de coupe (les deux premières lignes dans cet exemple) et une fabrique de greffon (après le mot clé advice)
Les deux premières lignes permettent d'identifier les composants (proxy de service UPnP par exemple) qui apparaitront dans le container. Ensuite, on créé les beans dont on a besoin (ici, deux stringToBool).
Une fois ces deux proxys reconnus et les beans créés, on crée les liens pour mettre en place la logique souhaitée. Les lignes du dernier paragraphe représentent ces liens, pour la première ligne par exemple, le proxy smartMattress est lié au bean stringBool et à la réception de l'événement estImmobileEvent la méthode set_MyProperty est modifiée.

En annexes de ce document, nous pouvons retrouver l'ensemble des Aspects d'Assemblage développés dans le cadre du scénario.

6 Conclusion

Nous avons défini dans ce document un scénario impliquant les différents objets communicants développés précédemment dans le cadre du projet. A partir de ce scénario et de l'architecture décrivant les différents objets, services et applications dédiées disponibles dans l'environnement, nous avons utilisé les Aspects d'Assemblages nous permettant de gérer l'adaptation de l'application en fonction de la disponibilité des différents services (et notamment l'apparition/disparition des objets communicants). Chaque aspect permet de décrire des sous-parties des différentes étapes du scénario.

Ainsi, les différents éléments ont été développés afin de permettre la mise en place d'une démonstration mettant en œuvre le scénario décrit dans ce document, scénario incluant les différents objets de la vie quotidienne augmentés de capteurs permettant la prévention ou la surveillance de maladies telles que celles concernant les troubles du sommeil.

7 Annexes

7.1 Etape 1 du scénario

Une personne achète un matelas et l'installe dans sa chambre. Elle se couche et le matelas s'interconnecte à tous les services de stockage d'informations fournis. On peut aussi surveiller ses mouvements sur le lit pendant son sommeil.

7.1.1 Aspect d'Assemblage entre le SmartMattress et l'application BedMonitoring

*smartmattress = smartMattress**
*bedMonitoring = bedMonitoring**
advice smartmattressToBedMonitoring(smartmattress,bedMonitoring) :
stringToBool1 : WComp.Beans.StringToBool
stringToBool2 : WComp.Beans.StringToBool
smartmattress.^estImmobileEvent_Event -> (stringToBool1.set_MyProperty)
stringToBool1.^PropertyChanged -> (bedMonitoring.SetImmobile)
smartmattress.^estPresentEvent_Event -> (stringToBool2.set_MyProperty)
stringToBool2.^PropertyChanged -> (bedMonitoring.SetPresent)
smartmattress.^mouvementEvent_Event -> (bedMonitoring.SetText)

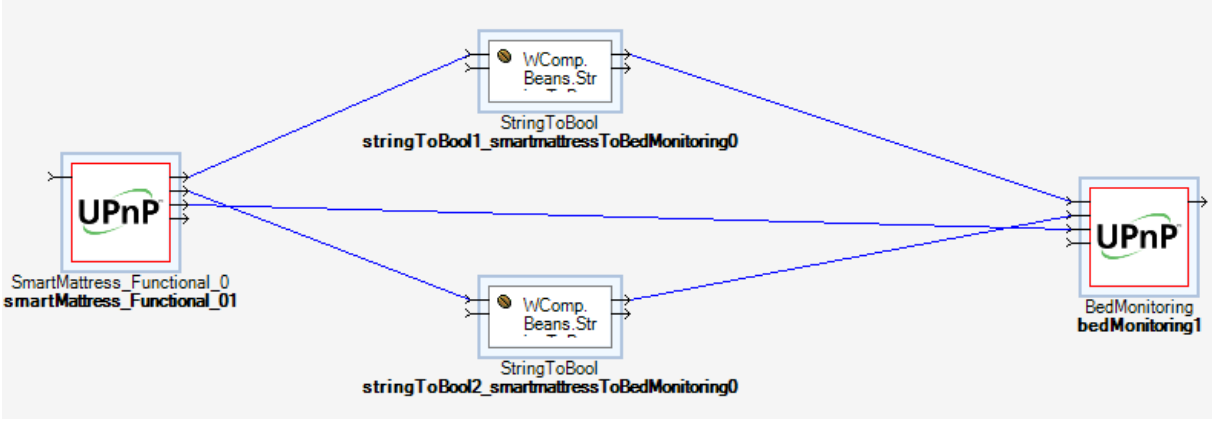


Figure 8 : Résultat de l'Aspect d'Assemblage entre le SmartMattress et l'application BedMonitoring

7.1.2 Aspect d'Assemblage entre le SmartMattress et l'application CVML

*smartmattress = smartMattress**
*cvml = cVML**
advice smartmattressToCVML(smartmattress,cvml) :
parseCSV1 : WComp.Beans.ParseCSV
smartmattress.^mouvementEvent_Event -> (parseCSV1.Parse)
parseCSV1.^Values -> (cvml.Predict)

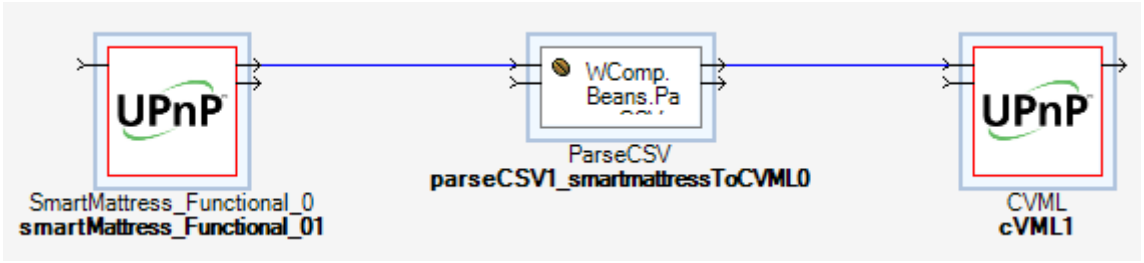


Figure 9 : Résulat de l'Aspect d'Assemblage entre le SmartMattress et l'application CVML

7.1.3 Aspect d'Assemblage entre le SmartMattress et l'application Plotter

*smartmattress = smartMattress**
*plotter = plotter**
advice smattressToPlotter(smartmattress,plotter) :
parseCSV2 : WComp.Beans.ParseCSV
parseCSV3 : WComp.Beans.ParseCSV
parseCSV4 : WComp.Beans.ParseCSV
parseCSV5 : WComp.Beans.ParseCSV
parseCSV6 : WComp.Beans.ParseCSV
toString1 : WComp.BasicBeans.ToString
toString2 : WComp.BasicBeans.ToString
toString3 : WComp.BasicBeans.ToString
toString4 : WComp.BasicBeans.ToString
toString5 : WComp.BasicBeans.ToString
singleToDoubleString21 : WComp.InterfaceTranslator.SingleToDoubleString2 (Value1:="capteur1")
singleToDoubleString22 : WComp.InterfaceTranslator.SingleToDoubleString2 (Value1:="capteur2")
singleToDoubleString23 : WComp.InterfaceTranslator.SingleToDoubleString2 (Value1:="capteur3")
singleToDoubleString24 : WComp.InterfaceTranslator.SingleToDoubleString2 (Value1:="capteur4")
singleToDoubleString25 : WComp.InterfaceTranslator.SingleToDoubleString2 (Value1:="capteur5")
smartmattress.^mouvementEvent_Event -> (parseCSV2.Parse)
smartmattress.^mouvementEvent_Event -> (parseCSV3.Parse)
smartmattress.^mouvementEvent_Event -> (parseCSV4.Parse)
smartmattress.^mouvementEvent_Event -> (parseCSV5.Parse)
smartmattress.^mouvementEvent_Event -> (parseCSV6.Parse)
parseCSV2.^Mouvement __ GetValueSensor1 -> (toString1.Input)
parseCSV3.^Mouvement __ GetValueSensor2 -> (toString2.Input)
parseCSV4.^Mouvement __ GetValueSensor3 -> (toString3.Input)
parseCSV5.^Mouvement __ GetValueSensor4 -> (toString4.Input)
parseCSV6.^Mouvement __ GetValueSensor5 -> (toString5.Input)
toString1.^Output ->(singleToDoubleString21.set_Value2SendEvent)
toString2.^Output ->(singleToDoubleString22.set_Value2SendEvent)
toString3.^Output ->(singleToDoubleString23.set_Value2SendEvent)
toString4.^Output ->(singleToDoubleString24.set_Value2SendEvent)
toString5.^Output ->(singleToDoubleString25.set_Value2SendEvent)
singleToDoubleString21.^StringsEvent -> (plotter.AddPoint)

singleToDoubleString22.^StringsEvent -> (plotter.AddPoint)
singleToDoubleString23.^StringsEvent -> (plotter.AddPoint)
singleToDoubleString24.^StringsEvent -> (plotter.AddPoint)
singleToDoubleString25.^StringsEvent -> (plotter.AddPoint)

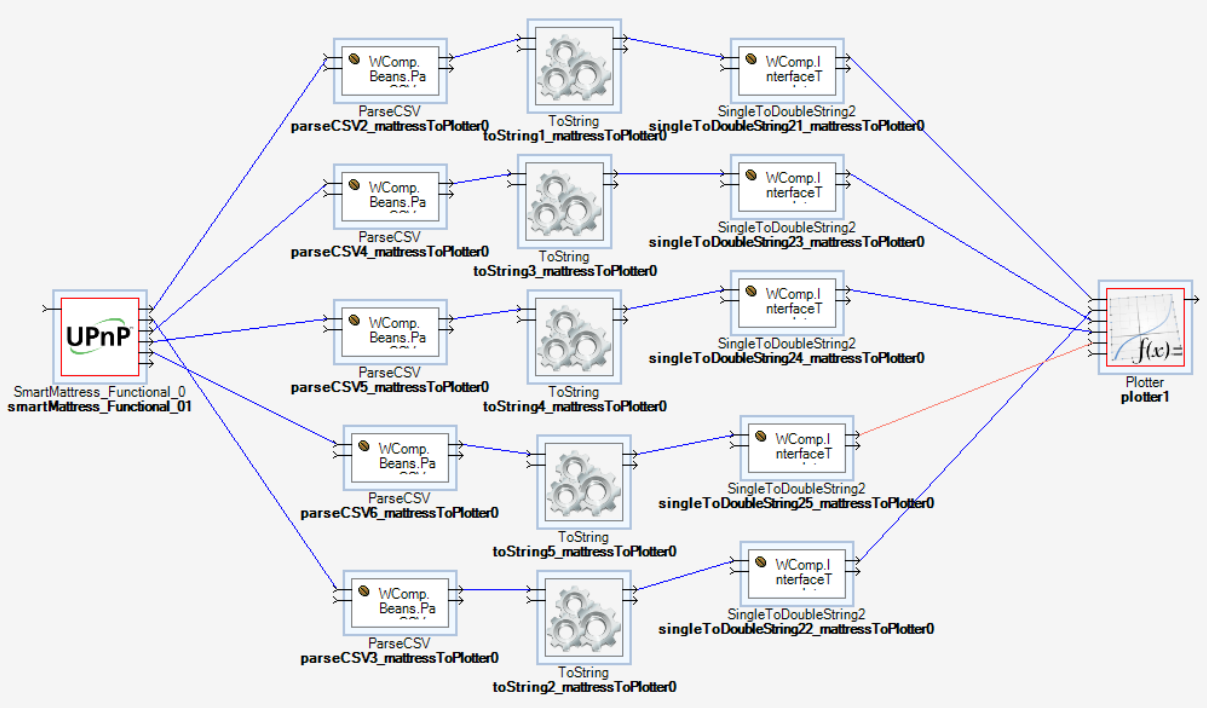


Figure 10 : Résulat de l'Aspect d'Assemblage entre le SmartMattress et l'application Plotter

7.2 Etape 2 du scénario

Puis la personne va chez Ikea et revient avec un nouveau dispositif : une lampe. On va pouvoir surveiller plus d'informations puisque la lampe inclut un capteur sonore. On interconnecte ce dispositif aux services de stockage et de visualisation des informations mais les objets ne sont pas interconnectés entre eux.

7.2.1 Aspect d'Assemblage entre le SmartLight et l'application NetworkLight

*smartLight = smartLight**
*networkLight = network_Light**
advice smartLightToNetworkLight(smartLight,networkLight) :
stringToBool1 : WComp.Beans.StringToBool
smartLight.^getLed_Event -> (stringToBool1.set_MyProperty)
stringToBool1.^PropertyChanged -> (networkLight.SetTarget)

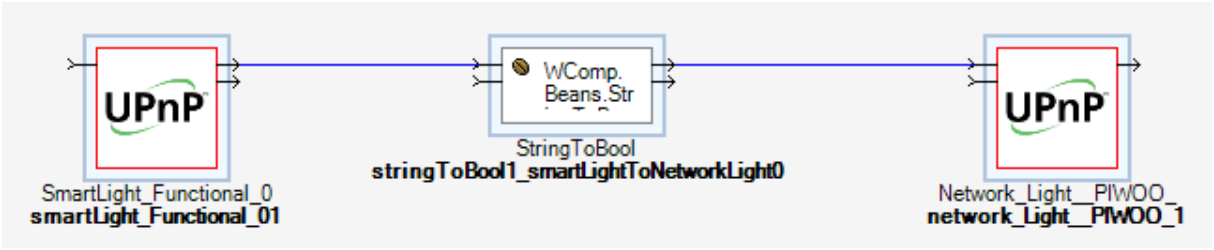


Figure 11 : Résulat de l'Aspect d'Assemblage entre le SmartLight et l'application NetworkLight

7.2.2 Aspect d'Assemblage entre le SmartLight et l'application Plotter

*smartLight = smartLight**
*plotter = plotter**
advice smartLightToPlotter(smartLight,plotter) :
singleToDoubleString : WComp.InterfaceTranslator.SingleToDoubleString2 (Value1:="Son")
smartLight.^getSound_Event ->(singleToDoubleString.set_Value2SendEvent)
singleToDoubleString.^StringsEvent -> (plotter.AddPoint)

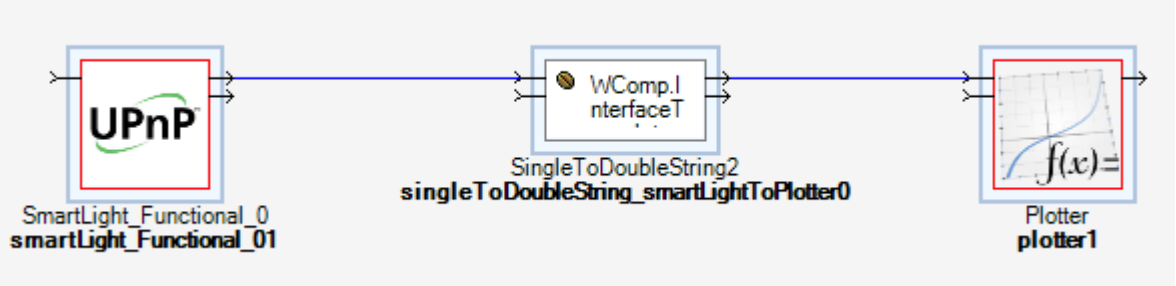


Figure 12 : Résulat de l'Aspect d'Assemblage entre le SmartLight et l'application Plotter

7.3 Etape 3 du scénario

Elle va ensuite à la Fnac et ramène un téléphone : celui-ci s'interconnecte avec la lampe. Il se produit alors une adaptation entre son téléphone et son environnement. La personne allume l'application « Réveil UPnP » et met en route une alarme ce qui éteint la lampe après 30 secondes d'attente.

7.3.1 Aspect d'Assemblage entre l'application Android Réveil et le SmartLight

*telephone = reveilUPnPDevice**
*smartLight = smartLight**
advice telephoneToSmartLight(telephone,smartLight) :
timer : WComp.BasicBeans.Timer (Period:=10000)
booleanFactory : WComp.BasicBeans.BooleanFactory (State:=False)
telephone.^SetAlarm_Event ->(timer.Start)
timer.^TimerTick -> (booleanFactory.FireBool)
booleanFactory.^BooleanCreated -> (smartLight.setLed)
timer.^TimerTick -> (timer.Stop)

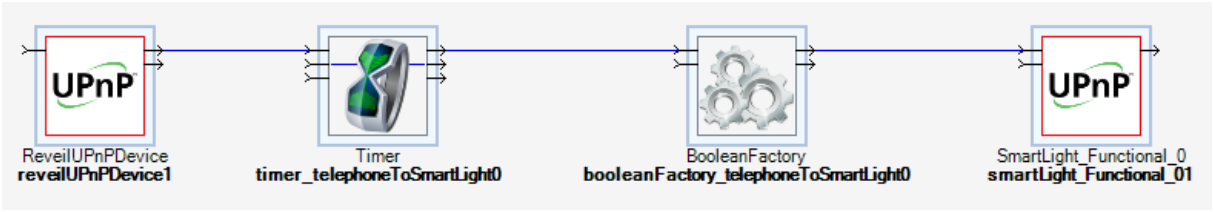


Figure 13 : Résulat de l'Aspect d'Assemblage entre l'application Android Réveil et le SmartLight

7.4 Etape 4 du scénario

Puis elle retourne à la Fnac et ajoute un chargeur à son environnement. On surveille plus d'informations et on interconnecte plus d'objets. Lorsqu'elle branche son téléphone sur le chargeur, la lampe alors allumée, s'éteint pour permettre au chargeur lui même de s'illuminer. Une fois l'activation de l'alarme, ce n'est plus la lampe qui éteinte mais le chargeur après 30 secondes d'attente.

7.4.1 Aspect d'Assemblage entre le SmartCharger et l'application AirControl

*smartcharger = smartCharger**
*aircontrol = airControl**
advice smartChargerToAirControl(smartcharger,aircontrol) :
smartcharger.^temperatureEvent_Event ->(aircontrol.SetTemp)
smartcharger.^humidityEvent_Event ->(aircontrol.SetHum)
smartcharger.^luminositeEvent_Event ->(aircontrol.SetLum)
smartcharger.^airQualityEvent_Event ->(aircontrol.SetQualite)
smartcharger.^valeursCaptEvent_Event -> (aircontrol.SetText)

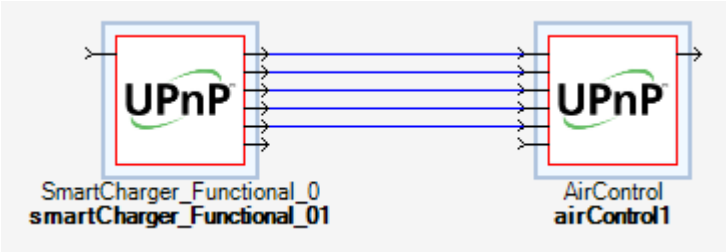


Figure 14 : Résulat de l'Aspect d'Assemblage entre le SmartCharger et l'application AirControl

7.4.2 Aspect d'Assemblage entre le SmartCharger et l'application Plotter

*smartcharger = smartCharger**
*plotter = plotter**
advice smartChargerToPlotter(smartcharger,plotter) :
parseCSV2 : WComp.Beans.ParseCSV
parseCSV3 : WComp.Beans.ParseCSV
parseCSV4 : WComp.Beans.ParseCSV
parseCSV5 : WComp.Beans.ParseCSV
toString1 : WComp.BasicBeans.ToString
toString2 : WComp.BasicBeans.ToString

toString3 : WComp.BasicBeans.ToString
toString4 : WComp.BasicBeans.ToString
singleToDoubleString21 : WComp.InterfaceTranslator.SingleToDoubleString2 (Value1:="temp")
singleToDoubleString22 : WComp.InterfaceTranslator.SingleToDoubleString2 (Value1:="hum")
singleToDoubleString23 : WComp.InterfaceTranslator.SingleToDoubleString2 (Value1:="lum")
singleToDoubleString24 : WComp.InterfaceTranslator.SingleToDoubleString2 (Value1:="air")
smartcharger.^valeursCaptEvent_Event -> (parseCSV2.Parse)
smartcharger.^valeursCaptEvent_Event -> (parseCSV3.Parse)
smartcharger.^valeursCaptEvent_Event -> (parseCSV4.Parse)
smartcharger.^valeursCaptEvent_Event -> (parseCSV5.Parse)
parseCSV2.^Mouvement __ GetValueSensor1 -> (toString1.Input)
parseCSV3.^Mouvement __ GetValueSensor2 -> (toString2.Input)
parseCSV4.^Mouvement __ GetValueSensor3 -> (toString3.Input)
parseCSV5.^Mouvement __ GetValueSensor4 -> (toString4.Input)
toString1.^Output ->(singleToDoubleString21.set_Value2SendEvent)
toString2.^Output ->(singleToDoubleString22.set_Value2SendEvent)
toString3.^Output ->(singleToDoubleString23.set_Value2SendEvent)
toString4.^Output ->(singleToDoubleString24.set_Value2SendEvent)
singleToDoubleString21.^StringsEvent -> (plotter.AddPoint)
singleToDoubleString22.^StringsEvent -> (plotter.AddPoint)
singleToDoubleString23.^StringsEvent -> (plotter.AddPoint)
singleToDoubleString24.^StringsEvent -> (plotter.AddPoint)

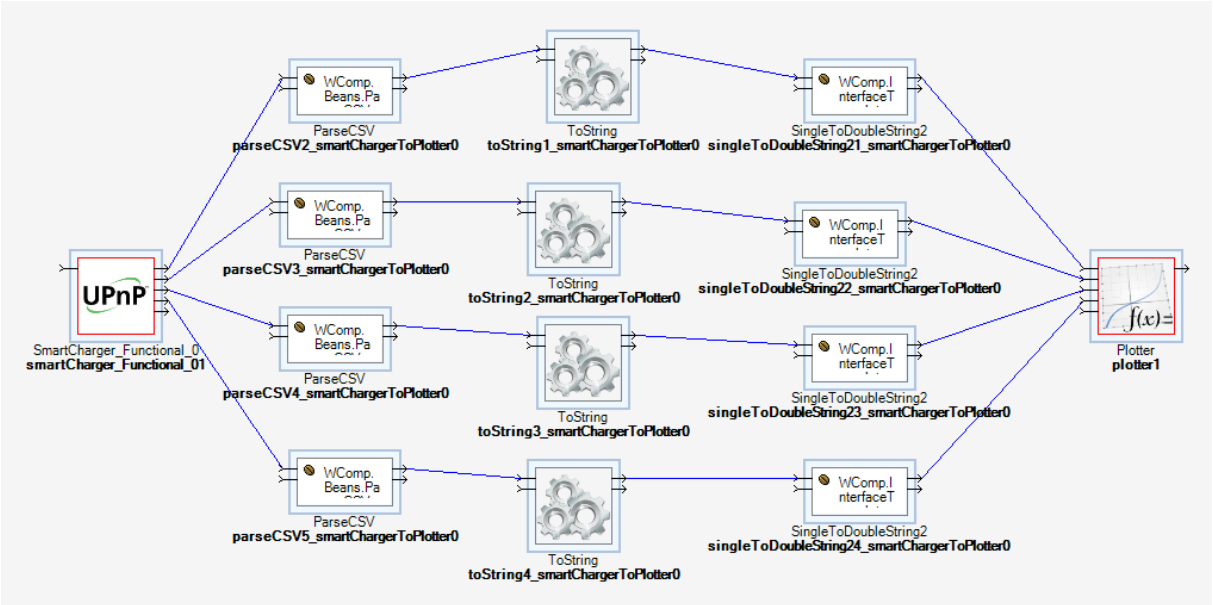


Figure 15 : Résulat de l'Aspect d'Assemblage entre le SmartCharger et l'application Plotter

7.4.3 Aspect d'Assemblage entre l'application Android Réveil et le SmartCharger

*telephone = reveilUPnPDevice**
*smartcharger = smartCharger**

advice telephoneToSmartCharger(telephone,smartcharger) :
timer : WComp.BasicBeans.Timer (Period:=10000)
booleanFactory : WComp.BasicBeans.BooleanFactory (State:=False)
telephone.^SetAlarm_Event ->(timer.Start)
timer.^TimerTick -> (booleanFactory.FireBool)
booleanFactory.^BooleanCreated -> (smartcharger.ledOff)
timer.^TimerTick -> (timer.Stop)

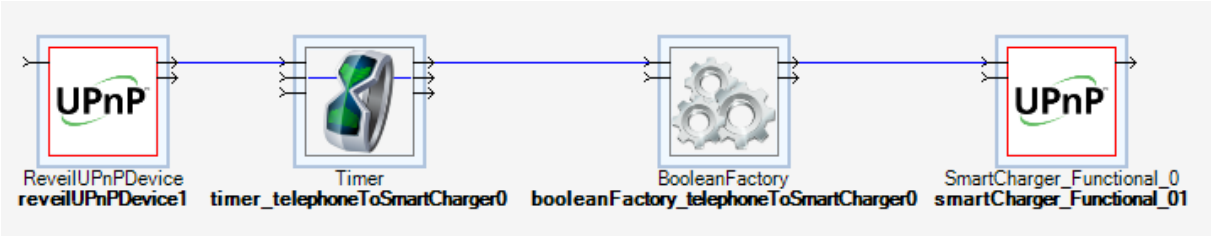


Figure 16 : Résulat de l'Aspect d'Assemblage entre l'application Android Réveil et le SmartCharger

7.4.4 Aspect d'Assemblage entre l'application Android Réveil, le SmartCharger et la SmartLight

*telephone = reveilUPnPDevice**
*smartcharger = smartCharger**
*smartlight = smartLight**
advice telephoneWithSmartChargerAndSmartLight(telephone,smartcharger,smartlight) :
ifThenElse : WComp.BasicBeans.IfThenElse
booleanFactoryTrue : WComp.BasicBeans.BooleanFactory (State:=True)
booleanFactoryFalse : WComp.BasicBeans.BooleanFactory (State:=False)
telephone.^PluggedIn_Event -> (ifThenElse.SetLevel)
ifThenElse.^ThenEvent -> (booleanFactoryFalse.FireBool)
booleanFactoryFalse.^BooleanCreated -> (smartlight.setLed)
ifThenElse.^ThenEvent -> (smartcharger.ledOn)
ifThenElse.^ElseEvent -> (booleanFactoryTrue.FireBool)
booleanFactoryTrue.^BooleanCreated -> (smartlight.setLed)
ifThenElse.^ElseEvent -> (smartcharger.ledOff)

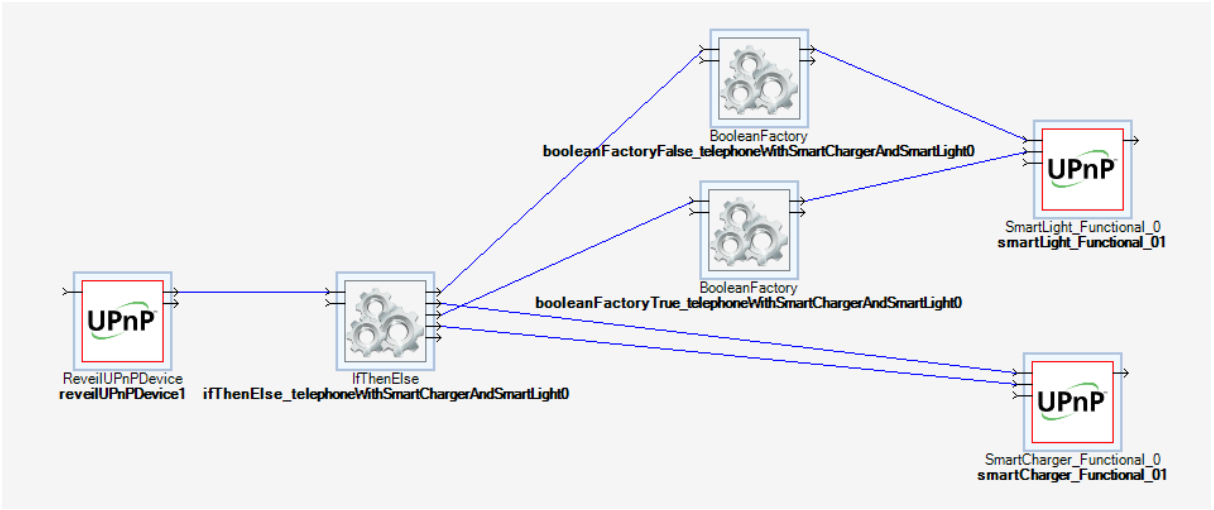


Figure 17 : Résulat de l'Aspect d'Assemblage entre l'application Android Réveil, le SmartCharger et le SmartLight