Master Web Intelligence 2A 2012-2013
Ubiquitous Computing
TP Context-Awareness

Christophe Gravier
christophe.gravier@telecom-st-etienne.fr
@chgravier

Table des matières

Préambule

Ce travail pratique fait suite aux deux séances précédentes. Nous nous plaçons toujours dans le cas d'usage des capteurs de température et de luminosité pour piloter des volets. Vous avez du mettre en œuvre la partie hardware Arduino (avec X. Serpaggi) et communication (avec J.P. Jamont). Nous allons mettre en place le serveur de contexte, et vous mettrez en œuvre l'aspect mobilité (avec J. Stan).

> Vous devez rendre un compte rendu de TP au format PDF. Ce compte-rendu comportera vos réponses aux questions posées dans ce document, ainsi que les portions de code que vous jugez importante en réponse à ces questions. Ce document sera envoyé à l'adresse : **wiuclab@gmail.com** avant **le 31 janvier 2013 23 :59**, en indiquant dans le nom de l'archive ainsi que dans le corps du mail les noms et prénoms du binôme. Vous pouvez également adjoindre dans le corps du message, ou envoyer *a posteriori* vos remarques/commentaires et suggestions d'amélioration de ce TP à la même adresse (tout commentaire constructif bienvenu !)

1 Description du TP

Concernant la partie mise en œuvre pour le "serveur de contexte", il nous faut faire le lien entre les Arduino d'un côté, et les clients Android de l'autre. Les contraintes technologiques sont donc :

- côté Arduino : les données sont exposées via un serveur HTTP qu'il faut "moissoner" (les Arduino ne poussent pas l'information, cf. fonctionnement TP Arduino).
- côté Android : le client doit être écrit en Java, afin de recevoir les notifications de changement de contexte.

Étant donné que :

1. vous n'avez plus un accès aisé aux Arduino
2. tout le monde n'a pas terminé les TPs précédents (?)

Nous avons le besoin de nous munir d'un simulateur de ces données de contexte (un serveur Web qui simulera celui embarqué dans les Arduinos). Plusieurs logiciels s'offrent à nous (Amigo [1], Siafu [2], etc.). Le problème de ces logiciels pour notre TP est qu'ils sont complexes et long à prendre en main (nous n'avons que 4h !).

1. `http://www.hitech-projects.com/euprojects/amigo/`
2. `http://siafusimulator.sourceforge.net/`

Je vous ai donc préparé un serveur en javascript utilisant node.js [3] et faye [4] qui :

- génère toutes les 3 secondes des nouvelles valeurs (aléatoirement) de température et de luminosité
- stocke la valeur opened/closed pour les volets dans un fichier
- sert au client, en réponse à une requête HTTP GET, les valeurs temperature, luminosité et état des volets en JSON (consultation du contexte)
- sert de serveur de publish/subscribe pour des clients compatibles avec le protocole bayeux [5]
- fait des publications toutes les 3 secondes du triple (temperature, luminosité, volets) comme les deux premières sont susceptibles d'avoir changées sur un topic dédié.

La première étape sera d'installer mon serveur sur votre machine (c.f. liste des tâches plus loin).

Tout le code et documentation dont vous disposez est accessible sur `https://github.com/cgravier/WI-UCLab`**.**

2 Compréhension de l'architecture

Nous allons implémenter l'architecture présentée à la figure . La partie entourée correspond à ce qui est traité durant ce TP, avec de part et d'autre ce que vous avez déjà réalisé et ce que vous réaslierez dans le prochain TP.

Traval demandé : se document sur le Web et répondre aux questions suivantes :

1. Qu'est-ce que le protocole Bayeux ?
2. Qu'est-ce que node.js ?
3. Qu'est-ce que le module Faye pour node.js ?

3 Installation et configuration du serveur de contexte

Dans cette partie nous nous intéressons à l'installation et à la configuration du serveur de contexte.

Traval demandé : Les étapes du travail à réalier sont les suivantes :

3. `http://nodejs.org/`
4. `http://faye.jcoglan.com/node.html`
5. `http://svn.cometd.com/trunk/bayeux/bayeux.html`

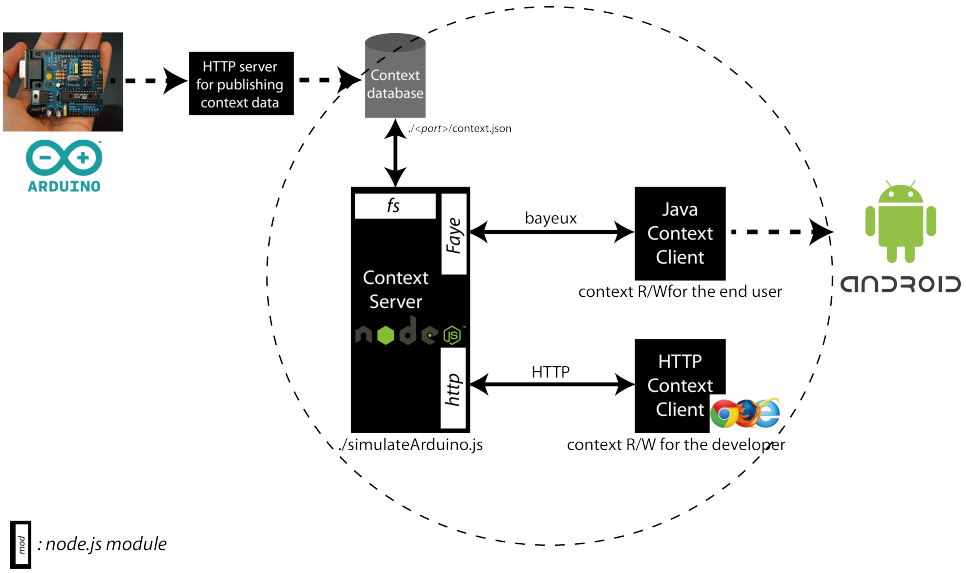

FIGURE 1 – Architecture déployée durant ce TP.

1. Installer node.js [6].
2. Télécharger le serveur écrit en javascript [7].
3. Installer les modules nécessaires à node.js pour exécuter le serveur (parcourir les dix premières lignes du code source peut se révéler intelligent).
4. Décider d'un numéro de port (> 1024, par exemple 8081) sur lequel votre serveur écoutera.
5. Dans le répertoire contenant le script du serveur, créer un sous répertoire portant comme nom votre numéro de port
6. Lancer le serveur (les instructions sont dans le fichier README).

Lorsque votre serveur est en cours d'exécution, il sait sait répondre à deux types de requêtes sur le port choisi :

- Requêtes HTTP GET : nous l'utiliserons pour consulter l'état du contexte dans le navigateur Web.
- Requêtes conforme au protocole bayeux : note client Java utilisera ce protocole.

Vous pouvez vérifier le bon fonctionnement de votre serveur en accédant dans votre navigateur à l'adresse : `http://<votreip>:<votreport>`. Tester l'accès au serveur en HTTP via un navigateur :

6. `http://nodejs.org/`
7. `https://github.com/cgravier/WI-UCLab/blob/master/context-server-nodejs/simulateArduino.js`

– Récupérer les valeurs température, luminosité et état des volets en JSON,
– Ouvrir et refermer les volets à l'aide de requêtes HTTP GET.

Les données sont publiées toutes les 3 secondes. Si vous refaite une requête après 3 secondes, vous pourrez constater que le contexte évolue au cours du temps, mais que les volets restent ouverts ou fermés tant que vous n'avez pas agit. Automatiser l'ouverture et la fermeture des volets fait l'objet de la suite de ce travail.

4 Client réactif aux modifications de contexte en Java

4.1 Coder un client standalone Java au serveur

Traval demandé :

1. À l'aide des bibliothèques listées et accessibles sur le compte git mentionné précédemment, écrire un client standalone Java au serveur.
2. La principale bilbiothèque à s'approprier est CometD, une bibliothèque permettant de se connecter en Java (entre autre) à un serveur Bayeux `http://cometd.org/`
3. Tester l'accès au serveur via le client Java (confronter que le client Java et votre navigateur récupèrent les mêmes valeurs).

Notez bien que lire les fichiers README est toujours utile.

4.2 Mise en œuvre d'un début de context-awareness

Traval demandé :

1. Modifier le client Java pour prendre des décisions et réaliser l'action automatiquement d'ouvrir/fermer les volets suivant certains seuils de sensibliité au contexte :
   – fermer les volets si la température descend en dessous de 15 degrés (il fait trop froid !) ou la luminosité dépasse 1500 lux (trop de lumière !).
   – ouvrir les volets si la température dépasse 25 degrés (il fait trop chaud !) ou la luminosité descend en dessous de 800 lux (pas assez de lumière !).

   Dans le cas où il fait plus de 25 degrés mais avec une luminosité de plus de 1500 lux, ou encore que la température est inférieure à 15 degrés mais avec une luminosité de plus de moins de 800 lux, l'arbitrage est de laisser les volets fermés.
2. Compiler et vérifier que cela fonctionne !

4.3 Impliquer l'utilisateur

Dans les systèmes sensibles au contexte, il est quelques fois difficile de prendre des décisions à la place de l'utilisateur. Dans le cas des volets, bien que la température ou la luminosité franchissent certains seuils, il est possible que l'utilisateur souhaite conserver l'état ouvert ou fermé de ses volets.

- Modifier le client standalone Java en ajoutant une fenêtre modale pour permettre de confirmer ou d'infirmer le fait d'ouvrir ou fermer les volets (en cas de changement d'état).
- Il est difficile de demander à l'utilisateur systématiquement son avis (notamment lorsque les valeurs oscillent autour des seuils de manière permanente). Ainsi, lorsque le système aura sollicité l'utilisateur et quelque soit sa réponse, le système ne changera plus l'état des volets et ne sollicitera plus l'utilisateur pour la prochaine heure [8].

5 Client proactif aux modifications de contexte en Java

Jusqu'à maintenant, nous nous sommes bornés à réagir à des changements de contexte. De tels systèmes vont quelques fois jusqu'à anticiper le changement. C'est ce que nous allons étudier dans cette dernière partie du TP.
Nous nous plaçons dans la configuration où un utilisateur ouvre et ferme manuellement ses volets. Nous nous contentons de consigner toutes les deux heures : la valeur de la température, la valeur de la lumisoité, l'état des volets, et l'heure de la journée. Nous procédons à ces observations pendant 3 mois. Nous disposons donc de 1080 observations. Nous souhaitons maintenant mettre en oevure une approche qui permettra de déployer un système dédié à notre utilisateur pour anticiper l'ouverture et la fermeture de ses volets, suivant son habitude.

Traval demandé :

1. Proposer une approche qui permettrait d'adresser ce problème (1 page).

6 Optionnel : proposer une modification du serveur de simulation de données de contexte

Le serveur génère des données fictives, pris de manière aléatoire entre 0 et 40 degrés pour la température et 0 et 2000 lux pour la luminosité. Rien

8. Pour les besoins du tests, fixer ce temps d'attente à 30 secondes

n'empêche le simulateur de donner une température de 3 degrés, suivi 3 secondes plus tard d'une température de 37 degrés !

– Modifier le serveur pour générer des valeurs plus plausibles. Idéalement votre proposition est paramétrable (i.e. ne pas coder avec les pieds !).

7 Wrap-up

Dans ce TP nous avons essayé de détourer la notion de contexte. Nous avons vu que le contexte était :

– évolutif dans le temps,
– issu de données hétérogènes,
– quelque fois subit (température ou luminosité ici) ou piloté (volets ouverts/fermés).

Vous vous êtes heurtés à des difficultés de différents ordres pour mené à bien la gestion du contexte dans l'applicaiton qui nous intéresse ici. Une première partie de ces difficultés est en fait d'ordre scientifique pour les chercheurs dans ce domaine, à savoir :

– comment modéliser le contexte ?
– comment réagir à des changements de la manière la plus pertinente et transparente pour l'utilisateur ?
– comment gérer de manière individuelle le contexte ?
– comment prédire des changements de contexte significatifs pour que le système devienne ubiquitaire ?
– comment justifier les actions du système auprès de l'utilisateur ? (facile en cas de règle métier, quid lorsque l'on apprend un modèle ?)
– comment conduire des inférences sur les données de contexte ?

Bien entendu ces problèmes sont intimement connecté avec les considérations technologiques associées à la gestion de contexte, puisque le contexte est :

– centré sur le Web.
– dépendant du middleware sous-jacent.