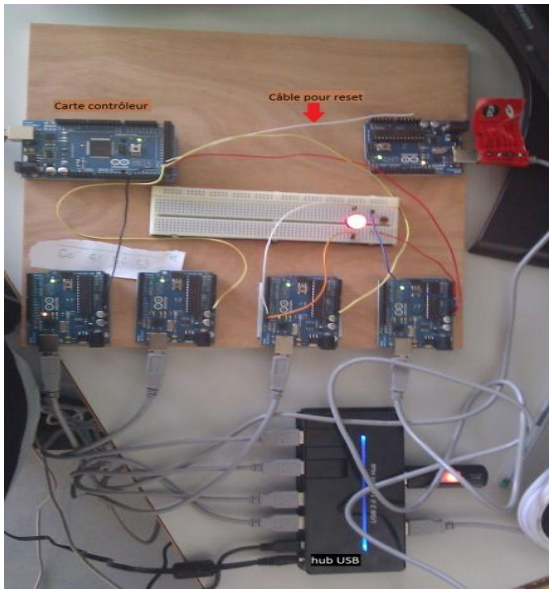
POLYTECH GRENOBLE

Gestionnaire de grille d'Arduino

ARDUI'GRID



Carpaye Willy
Dewulf Mathieu
El Bakkouri Nysrine
Jurado Leduc Thibault
Malkas Benjamin
Mraihi Haythem

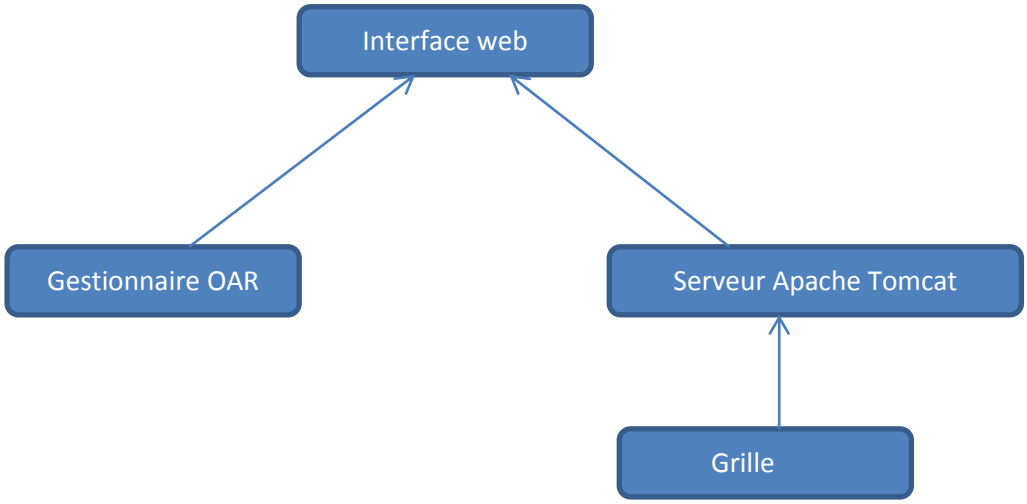
2010/2011

Introduction

Ce manuel donne des informations de développement aux futurs développeurs du projet Ardui'Grid. Ces informations sont en partie utiles pour la mise en place des conventions de codages utilisées dans ce projet.

Le logiciel Ardui'Grid a pour but d'identifier des cartes Arduino connectées entre elles à partir d'un serveur. On dispose d'un gestionnaire de réservations de type OAR permettant de réserver des cartes Arduino. Les résultats des expériences seront affichés dans une interface web.

L'architecture globale de notre application peut être décrite comme suit :



I. Communication avec les cartes

1/Implémentation physique

Une carte contrôleur a une connexion par carte présente dans la grille. Cette connexion sert à réinitialiser les cartes (notamment lors du chargement d'un programme). 2 ports USB sont utilisés sur le serveur : le premier pour la carte contrôleur, le deuxième pour le hub USB sur lequel toutes les cartes sont connectées. Il est important de connaitre pour chaque carte de la grille le lien entre le pignon de la carte contrôleur (reset) et le nom du port série sur lequel est branchée la carte. Ces liens servent à créer le fichier de configuration XML. S'il a lieu d'être le capteur est aussi ajouté dans le fichier de configuration.

2/Implémentation logiciel

Pour communiquer avec les cartes nous avons utilisé une librairie RXRT. On communique donc par le biais des ports série USB.
Chaque carte est représentée par une instance de la classe « Carte ». La carte contrôleur dispose de sa propre classe. Une classe Grille représente la grille. Cette classe qui à priori n'est instanciée une seule fois, est le seul lien (interface) vers la partie serveur. Afin de créer cet objet, le constructeur de grille fait appel à l'objet Chargeur qui s'occupe de parser le fichier XML de configuration. Ensuite le constructeur se connecte à toutes les cartes et au contrôleur. Une fois que le constructeur de grille finit son exécution, la grille considère qu'elle *est déjà connectée,* il est donc important de physiquement connecter les cartes au préalable.

3/Précisions sur la compilation (classe Compiler)

La compilation et le chargement dans une carte se passe en 4 étapes :

1) On copie le fichier à compiler dans le répertoire courant
2) On crée le Makefile à partir du fichier « makebase » auquel on ajoute toutes les variables. Pour l'instant les variables ne sont que le port où uploader, mais si la grille comporte différents types de carte il faut ajouter quel est le type de carte.
3) On compile le fichier à l'aide du nouveau Makefile.
4) On upload avec le Makefile.

II. Installation d'OAR

L'installation du gestionnaire de nœud OAR se fait via les paquets debian se trouvant sur le site http://oar.imag.fr/debian. La version d'OAR utilisée est la version 2.5 dans le but de pouvoir utiliser l'api d'OAR.

Les paquets oar-admin, oar-api, oar-common, oar-server, oar-user, oar-web-status sont installés.

Le paquet oar-admin pour pouvoir administrer le gestionnaire de nœud et pouvoir gérer les différentes ressources. Oar-server permet l'utilisation du coté serveur de OAR, oar-user permet lui, l'utilisation du coté client de OAR, on peut donc avec ceci gérer des jobs sur les différents nœuds géré par OAR. Les paquets oar-api et oar-web-status permettent l'utilisation de l'api d'OAR.

I/Base de données

Une base de données a été mise en place dans le but de gérer les différentes ressources et jobs liés à OAR. MySQL doit donc être installer pour pouvoir disposer d'une base de données propre à OAR. Cette base de données doit être initialisée avec les commandes oar_mysql_db_init ou oar_psql_db_init suivant si on utilise MySQL ou PostgreSQL. Cette commande permet de rajouter la base de données OAR sur le serveur. Grâce à cette base de données, il est possible de consulter l'état des différents composants du système OAR (ressources, jobs...).

2/Fichiers de configuration

- Fichier de configuration de l'API OAR

Le fichier de configuration apache-api.conf (/etc/oar/apache-api.conf) doit être modifié pour pouvoir permettre l'accès à l'api directement sur un navigateur. Les droits d'authentification sont modifiés, plusieurs champs sont rajoutés :

Deny from all

Allow from localhost.localdomain

Allow from localhost

Allow from <name server>

Cela permet de donner l'accès à l'api pour l'utilisateur courant et d'interdire l'accès à des utilisateurs malveillants.

- Fichier de configuration d'OAR

Le fichier de configuration oar.conf (/etc/oar/oar.conf) doit être modifié comme suit :

// Par défaut on souhaite un nœud

OARSUB_DEFAULT_RESOURCES="/network_address=1"

// Utilisation du serveur ssh standard

#OPENSSH_CMD="/usr/bin/ssh -p 6667"

// Pas de test sur les noeuds (Pingchecker)

FINAUD_FREQUENCY="0"

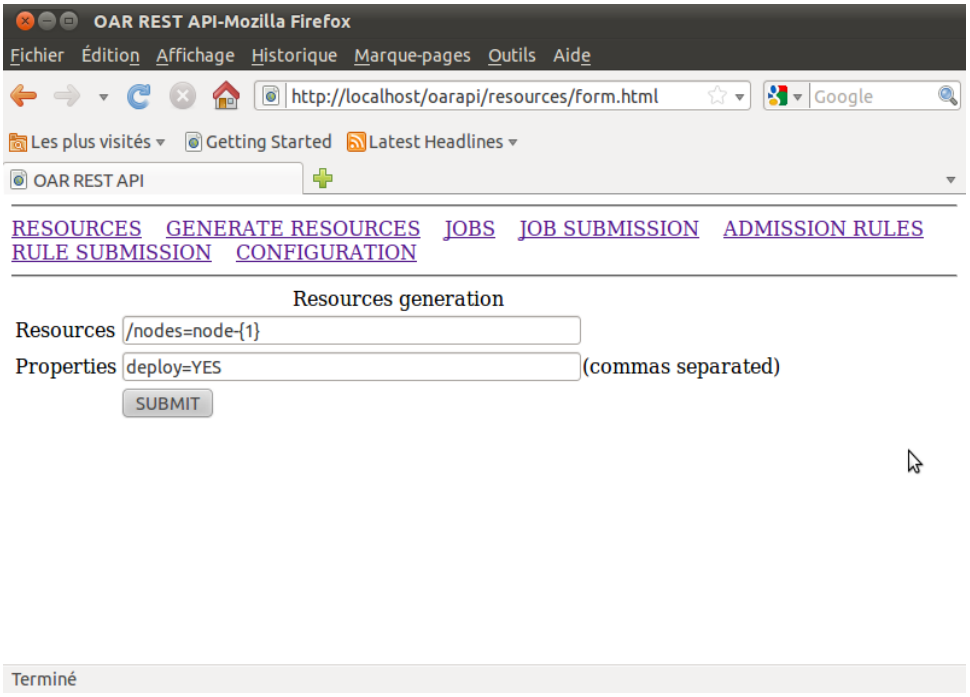
3/ Quelques fonctions de l'api OAR

L'API OAR permet de faire des appels à OAR par le web, ceci permet de créer aisément des interfaces web de soumission, administration, visualisation...

Dans notre projet elle se trouve à l'adresse http://localhost/oarapi/index.html sur le serveur OAR.

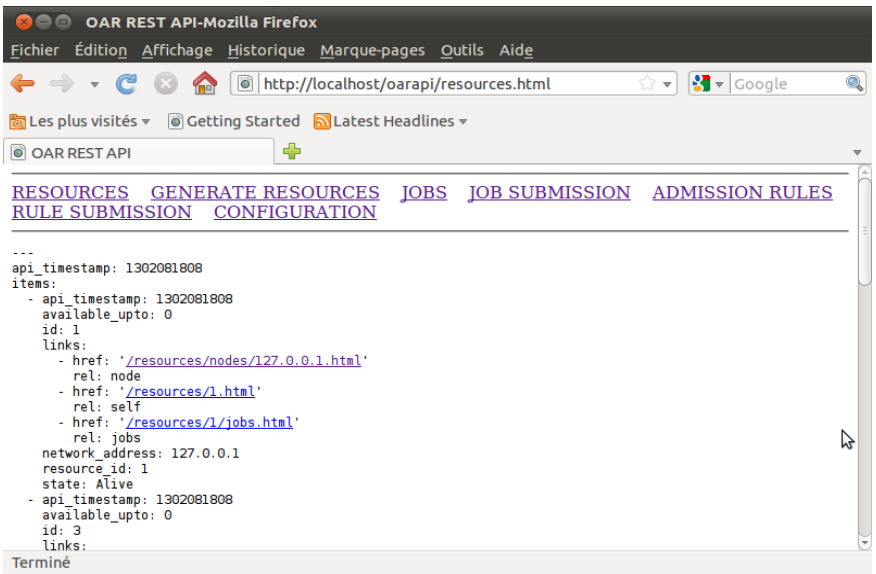
L'utilisation de l'api est très simple, voici quelques captures qui résument bien les différentes fonctions.

- Ajouter une ressource



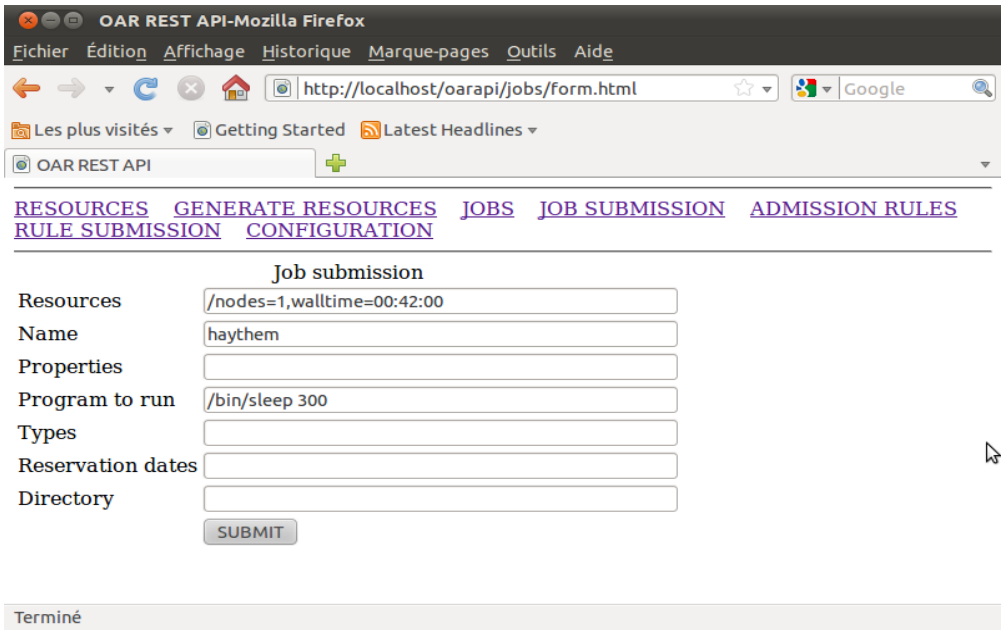
Pour ajouter une ressource, il ne faut pas oublier de préciser son type (ex : deploy) sinon le type « default » lui sera affecté.

- Récupérer la liste des ressources



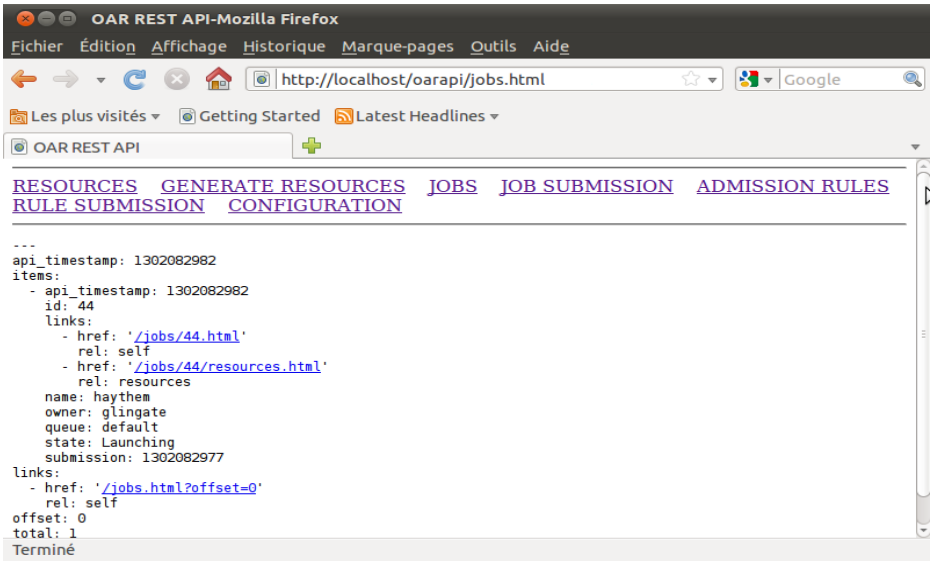
Ca permet de récupérer la liste des ressources avec leurs propriétés. Par exemple, l'état de la ressource (Alive, Dead, Absent, Suspected) ou encore son adresse IP.

- Soumettre un job



Pour soumettre un job, il faut préciser le nombre de ressource nécessaire au job ainsi que la durée de ce dernier « walltime ». Il faut aussi préciser l'emplacement du programme à exécuter. Le champ « Réservation dates » est optionnel, il permet juste de préciser la date à partir de laquelle le job doit commencer, s'il n'est pas remplit, le job débutera directement sans attendre.

- Récupérer la liste des jobs



Ca permet de récupérer la liste des jobs avec leurs propriétés. Parmi les propriétés d'un job on trouve :

-L'identifiant de celui qui a lancé le job

-L'état du job: launching, waiting, running…

-La date de soumission : dans notre cas 1302082977

Pour plus d'informations consultez :

- Le site officiel : http://oar.imag.fr/
- le wiki : http://wiki-oar.imag.fr/index.php/Main_Page

III. Interface graphique

1/Explications générales

Pour notre interface graphique, nous avons utilisé la librairie GWT (Google Web Toolkit) permettant de créer et maintenir des applications web dynamiques mettant en œuvre JavaScript, HTML et CSS dans la logique de la technologie web2.0 AJAX. Tout cela en utilisant le langage et les outils JAVA. Notre interface est basée sur le modèle MVP (Model View Presenter). C'est une architecture et une méthode de conception considérée comme un dérivé du modèle de conception MVC (Model View Controller).

Pour avoir une communication asynchrone entre le serveur et le client, on a utilisé la technologie GWT-RPC. En effet, le Framework GWT permet depuis le client JavaScript de faire des appels RPC au serveur GWT. L'objectif est de permettre au JavaScript client d'exécuter des méthodes sur le serveur et d'en récupérer le résultat, tout cela de façon totalement asynchrone (AJAX). Le schéma ci-dessous explique bien l'infrastructure RPC :

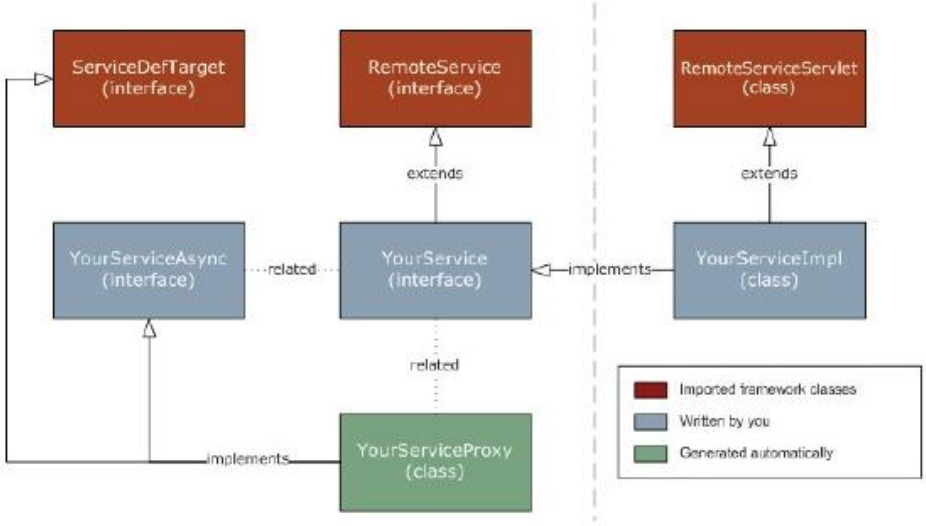


Figure 1: Google 2011

2/Organisation du code :

Dans cette partie, nous allons détailler la structure de notre code. Nous expliquerons le rôle des différents packages et classes dans le fonctionnement final de l'application.

Notre code est organisé dans plusieurs packages afin de rendre notre application la plus générique et maintenable possible.

-**glingate.account**: permet de gérer les comptes des utilisateurs.

-**glingate.booking**: permet de gérer les réservations de cartes.

-**glingate.client**: contient la logique de navigation entre les applications.

-**glingate.dashbord**: permet d'afficher le tableau de bord.

-**glingate.login**: permet de gérer les connexions des utilisateurs.

-**glingate.menuHeader**: définit notre menu.

-**glingate.node**: permet d'afficher les fonctionnalités des nœuds sur la grille.

-**glingate.software**: permet de compiler, créer, éditer et tester des programmes.

-**glingate.task**: permet à l'utilisateur de créer des jobs sur OAR.

-**glingate.lab** : permet de récupérer les données des expériences passées.

-**glingate.common** : contient toute la logique de l'architecture.

http://www.youtube.com/watch?v=g2XclEOJdIc

3/ Maintenance et développement de l'application

Outils requis : Eclipse, plugin GWT pour Eclipse, navigateur web avec un plugin GWT pour le Development Mode (de préférence Google Chrome).

Les sources fournit sont à placer dans un projet Web Application Project dans Eclipse. Glingate.html remplace votre page HTML présente dans le dossier /war de votre projet.

Mode de développement :

-Development Mode permet de tester votre application sans la compiler via un navigateur contenant le plugin GXT

-Hosted Mode permet de déployer votre application en la compilant. La compilation permet de créer des classes dans le dossier /war/WEB-INF/lib/ qui sont vos classes serveurs (servlet et logique serveur). Le dossier /war peut être déployé facilement avec Apache-Tomcat.

Servlet et Gwt-Rpc :

La création d'un servlet requiert de l'indiqué dans le fichier war/WEB-INF/web.xml avec :

```
<servlet>
  <servlet-name>myServiceImpl</servlet-name>
  <servlet-class>
    com.example.foo.server.MyServiceImpl
  </servlet-class>
</servlet>
<servlet-mapping>
  <servlet-name>myServiceImpl</servlet-name>
  <url-pattern>/com.example.foo.Foo/myService</url-pattern>
</servlet-mapping>
```

4/ Evolution de l'application

Base de données :

Le back-end server doit intégrer une base de données MySQL ou postgreSQL.

Cette base de donné contient les tables suivantes :

-utilisateur : contient les utilisateurs

-reservation : contient les reservations qui doivent correspondre à des réservations OAR

-programme : contient les programmes utilisateurs

-taches : contient les jobs OAR

Programmes et données :

Les programmes et les données seront stockés dans des fichiers XML dont le format reste à définir.

IV.Configuration Apache Tomcat

Fichier policy :

Le fichier policy de Tomcat est le fichier repertoiredetomcat/conf/catalina.policy.

Il est nécessaire de le modifier en ajoutant des permissions pour l'accès aux ports USB utilisés pour les cartes Arduino.

Exemple sous windows :

Grant codeBase « COM3 » {

Permission java.security.AllPermission ;

} ;

Des entrées sont aussi nécessaires pour exécuter RunTime.exec et accéder en lecture écriture à des fichiers disponibles sur le serveur.

5/ Logique du dialogue client/serveur

Cohérence des données :

Les données doivent conservées une cohérence entre leur représentation dans le code serveur et dans le code client. Pour cela il faut attribué un unique identifiant pour un objet qui doit circuler entre le serveur et le client. Par exemple associer un uid à un programme permet de dissocier la description du programme dans la base de donnée et son contenu dans un fichier XML stocké sur le serveur. Ceci est particulièrement utile lorsque l'on utilise des listes d'objets présentées aux clients.

Asynchrone :

Les Gwt-Rpc permettent d'obtenir une communication asynchrone (AJAX) par rapport à l'utilisation des requêtes http GET et POST. Ceci permet d'obtenir une fluidité dans l'application en ne bloquant pas le client à chaque action de sa part.

6/Références

Google Web Toolkit :

Tutoriaux et documentation : http://code.google.com/webtoolkit/

Vidéos :

architecture : http://www.youtube.com/watch?v=PDuhR18-EdM

user interface : http://www.youtube.com/watch?v=g2XclEOJdIc