1 Introduction

Le projet Ironmines offre une interface de programmation pour faire évoluer le robot Kobuki. L'objectif est bien-sur de le faire concourir à la coupe de France de Robotique.

2 Installation

2.1 Pour le développement

Le programme ironmines peut être installer sur n'importe quelle machine linux compatible avec ROS groovy. On choisira Ubuntu 12.04 pour le développement. Cette partie expliquera comment installer une version entière du programme pour le développent. Les étapes sont les suivantes :

- ROS - Kobuki
- Arduino - Roserial
- PhaROS - Ironmines

2.1.1 ROS - Kobuki

Tout d'abord il faudra télé-charger et installer l'utilitaire PhaROS Tools, disponible à l'url :
`http://car.mines-douai.fr/wp-content/uploads/2014/01/pharos.deb`
Alors on pourra procéder à l'installation simplifiée de PhaROS :

```
pharos ros-install
```

On pourra ensuite installer le kobuki :

```
sudo apt-get install ros-groovy-kobuki-desktop
```

2.1.2 Arduino - Rosserial

On pourra installer l'IDE pour Arduino qui permet d'éditer les sous-procédure utilisé :

```
sudo apt-get arduino
cd ~/sketchbook
git clone https://github.com/mattonem/ironminesarduino.git
```

Cependant ce projet requière les librairies Rosserial :

```
cd ~/ros/workspace/src/
git clone https://github.com/ros-drivers/rosserial.git
cd ~/ros/workspace/
catkin_make
cd ~/sketchbook/libraries
rosrun rosserial_arduino make_libraries.py .
```

2.1.3 PhaROS - Ironmines

Il est temps de passer a l'installation de PhaROS et du projet Ironmines. Toute la procédure est très simplifiée avec l'outil PhaROS Tools bien-sur.

```
phaROS register-repository --url="http://smalltalkhub.com/mc/maxmattone/ironmines
    /main" --package=IronMinesDirectory
pharos install IronMines
```

2.2 Raspberry

L'objectif étant de faire fonctionner le kobuki sur Raspberry [1], voici la procédure que j'ai pu mettre sur pieds pour installer le projet. Je suppose qu'on dispose d'un Raspberry type B avec une distribution Raspbian fraîchement configurée ainsi qu'une connection internet viable.

2.2.1 ROS - Kobuki

Tout d'abord il faut installer ROS sur le Raspberry :

```
sudo sh -c 'echo "deb http://64.91.227.57/repos/rospbian wheezy main" > /etc/apt/
    sources.list.d/rospbian.list'
wget http://64.91.227.57/repos/rospbian.key -O - | sudo apt-key add -

sudo apt-get update

sudo apt-get install ros-groovy-ros-comm
pip install --upgrade rosdistro
sudo rosdep init
rosdep update

echo "source /opt/ros/groovy/setup.bash" >> ~/.bashrc
echo "export ROS_HOSTNAME=localhost" >> ~/.bashrc
source ~/.bashrc

 mkdir -p ~/ros/workspace/src
 cd ~/ros/workspace/src
 catkin_init_workspace
 cd ~/ros/workspace/
 catkin_make
 echo "source ~/ros/workspace/devel/setup.bash" >> ~/.bashrc
```

Puis installer les nœuds pour communiquer avec le Kobuki :

```
sudo apt-get install ros-groovy-diagnostics
sudo apt-get install ros-groovy-kobuki-node
sudo apt-get install ros-groovy-kobuki-msgs
sudo apt-get install ros-groovy-kobuki-ftdi
```

1. NB : Après test le Raspberry n'est pas assez puissant pour gérer le déplacement et l'odométrie ensemble.

```
rosrun kobuki_ftdi create_udev_rules
```

On pourra alors tester l'installation en exécutant dans un terminal :

```
roscore
```

Et dans un autre terminal :

```
roslaunch kobuki_node minimal.launch
```

Si tout ce lance correctement, on peut passer à la suite.

2.2.2 Rosserial

Comme on ne souhaite pas installer une version de développement, l'installation du paquet ros-groovy-rosserial suffira.

```
sudo apt-get install ros-groovy-rosserial-arduino
sudo apt-get install ros-groovy-rosserial
```

2.2.3 PhaROS - IronMines

On pourra dans un premier temps copier le dossier `/ros/workspace/src/IronMines` de la machine de développement sur le Raspberry au même endroit (il faudra corriger les chemins dans le script `bin/headless`). Ensuite il faudra télécharger la machine Pharo compatible Raspberry : `https://ci.inria.fr/pharo-contribution/view/RaspberryPi/job/RaspberryPi-Cross-Compilation/lastSuccessfulBuild/artifact/vmSources/results.tar.gz`. Puis remplacer, intelligemment la machine de projet qui se trouve dans `vm/vm-files/`.
À ce point, on teste notre programme ; On démarre le kobuki :

```
roslaunch kobuki_node minimal.launch
```

Puis on lance notre programme

```
rosrun IronMines headless kobuki
```

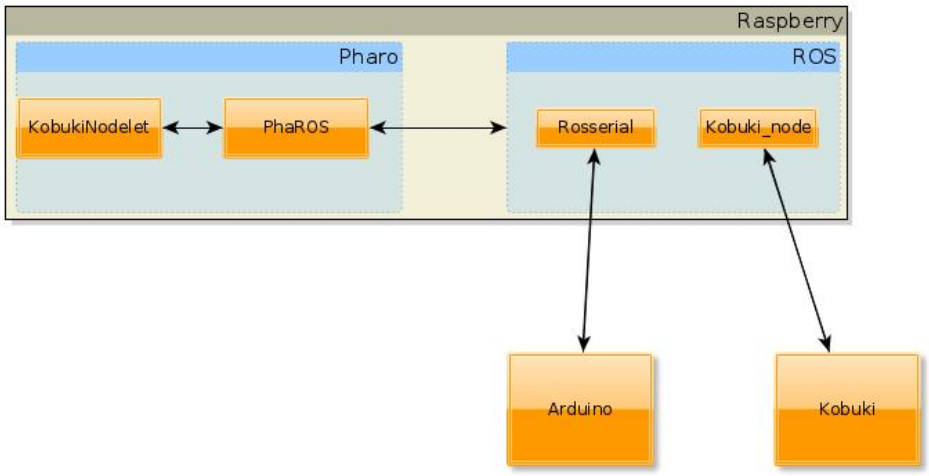
3 Démarrer le kobuki

Pour démarrer l'ensemble des fonctionnalités du kobuki et de l'Arduino, on procédera comme suit :

- brancher Arduino, Raspberry (ou ordinateur de développement)
- démarrer le kobuki
- démarrer kobuki_node sur l'ordinateur
  `roslaunch kobuki_node minimal.launch`
- démarrer Rosserial
  `rosrun rosserial_python serial_node.py /dev/ttyACM0`
- démarrer le programme
  `rosrun IronMines headless kobuki`

4 Architecture

D'un façon générale, Pharo fait office d'intelligence. ROS ne fait que transmettre les messages provenant de l'Arduino et du Kobuki au travers des nœuds Rosserial et Kobuki_node.



Architecture générale

4.1 Arduino

Le code pour l'Arduino se trouve a l'url :
`https://github.com/mattonem/ironminesarduino.git`.

Basiquement, on utilise la librairie fournie par Rosserial pour pouvoir publier différents topiques sur le port série ainsi que souscrire à d'autres. On publie les données relatives au contacteur de démarrage et au capteurs ultra-sons. On souscrit aux différents topiques concernant les actions à effectuer (activer le canon, bouger le bras rétractable).

4.1.1 Subscribed Topics

- `/canonTrigger (std_msgs/Bool)`
  Si msg.data == true : déclenche l'électrovanne pour faire tirer le canon.

4.1.2 Published Topics

- `/sonar/<i> (std_msgs/UInt16)`
  Donne la distance au mur en face du sonar `i`.
  Aujourd'hui, `i` est compris entre 2 et 7
- `/startTrigger (std_msgs/Bool)`
  Donne les fronts montant et descendant pour le cordon de démarrage.

4.2 PhaROS - Ironmines

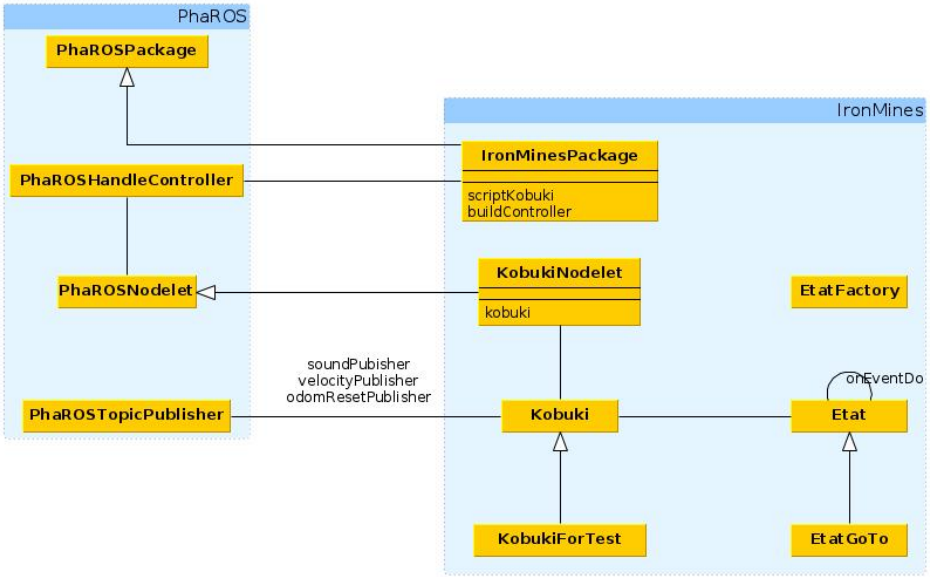


Diagramme de classe

Le point d'entrée dans le programme est `IronMinesPackage` et sa méthode `scriptKobuki`. C'est cette méthode qui va lancer l'instantiation de l'ensemble des entités nécessaires pour le programme. Va suivre l'instantiation du `Controller` et son association avec les nodelets utilisées (ici on utilise uniquement `PhaROSTransformationNodelet` et `KobukiNodelet`). `KobukiNodelet` s'occupe alors de la création d'un objet `Kobuki`, qui est une représentation logique du robot.

On peut alors affecter au Kobuki des états. Ceux-ci représentent les actions qu'on lui demandera de faire (ex : `EtatGoTo`). Un état hérite de la classe `Etat`, et peut définir dans la méthode `action` la (les) action(s) actions à effectuer. À la fin de l'action, si l'état doit finir et mener sur un autre état (ce n'est pas le cas de tous), on pourra alors appeler la méthode `onFinish`.

Il existe différentes façon de sortir d'un état. La première est déclenché lorsque l'état est terminé (`onFinish`). Les autres peuvent être personnalisées : `onObstacle`, `onBumperEvent`. Chacune de ces portes de sorties sont paramétrées à l'instantiation par la factory. Exemple :

```
newEtat := EtatFactory createGoTo: (0 @ 1) onObstable: unEtatdEvitement next:
    etatSuivant.
kobuki changeEtat: newEtat.
"NB : je ne prsente pas ici une interface finalisee, ces methodes sont susceptibles de
    changer"
```

Ici on crée un nouvel état qui demandera au kobuki de se rendre en `(0,1)`, puis de passer à l'état `etatSuivant`, mais en cas d'obstacle on passera a l'état `unEtatdEvitement`. Comme on peut le voir dans le graphique d'objet

suivant (représentant le kobuki au départ d'un script pour tirer sur le mammouth) les états peuvent former des graphes complexes.

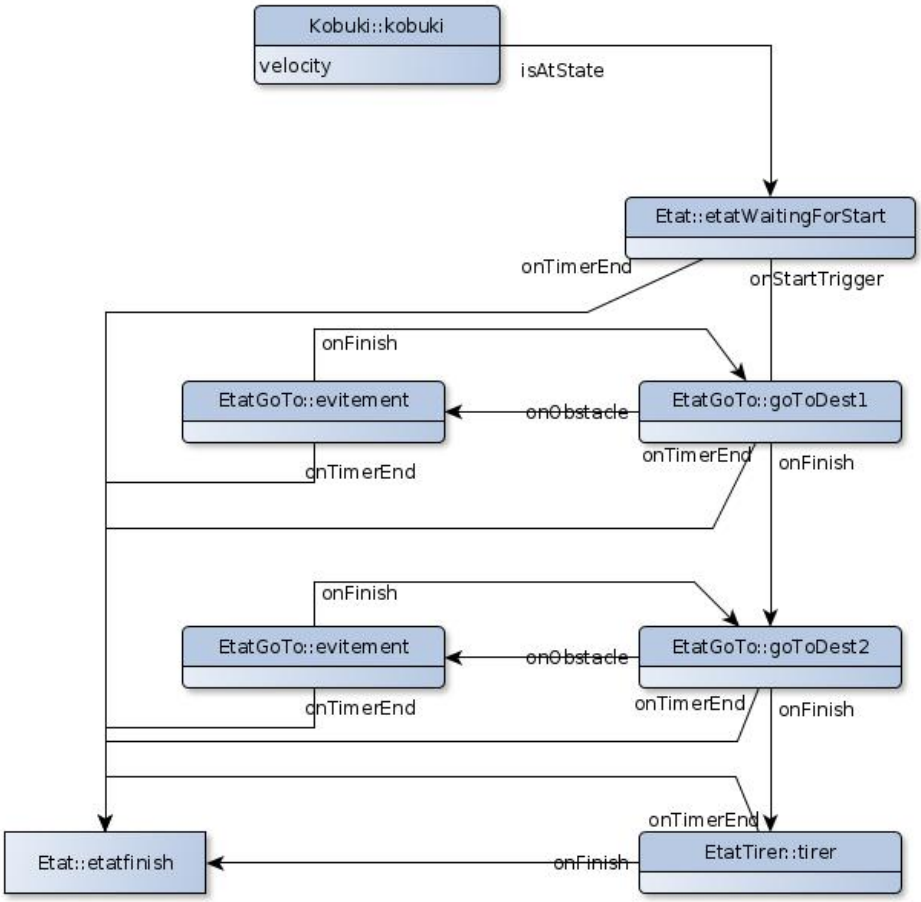


Diagramme d'objets