# Pyrame, un framework de prototypage rapide pour systèmes online

- Activités Techniques - Informatique -

Date de mise en ligne : lundi 10 mars 2014

**Laboratoire Leprince-Ringuet**

**Pyrame, un framework de prototypage rapide pour systèmes online**.

Frédéric Magniette, Miguel Rubio-Roy, Floris Thiant

Pyrame est un framework léger qui permet de prototyper très rapidement des systèmes online d'une complexité arbitraire. Il est constitué de deux parties distinctes : une chaine d'acquisition générique que l'on peut adapter facilement au media et au format des données, un module de commandes générique permettant d'implémenter des contrôles-commandes, facilement et de façon hiérarchique.

# Principe de fonctionnement du framework

Le framework Pyrame est basé sur une série de modules unitaires reliés entre eux par des sockets réseau (TCP) et une syntaxe de commande simple, basée sur XML. Chaque module gère un type de matériel spécifique. En temps normal, un module est en attente d'une commande. Celle-ci arrive via une connection TCP sur un port spécifique à ce module sous la forme d'un message XML :
set_clock_frequency">
 3
 50Hz

set_clock_frequency est le nom de la fonction. Le premier paramètre (3) est un entier désignant de manière unique le matériel concerné (device handler). Les paramètres suivants sont les valeurs des paramètres de la fonction.

Une fois la fonction exécutée, le résultat est renvoyé au moyen d'un message XML transmis sur la même socket que la demande.
0>Unknown device
La valeur de retour désigne le succès (valeur 1) ou l'échec (valeur 0) de la fonction. Un message est ajouté entre les balises précisant les raisons de l'échec.

Une fois cette communication terminée, la socket peut être fermée ou utilisée pour d'autres commandes.

Naturellement, les modules peuvent s'appeler mutuellement pour mener à bien leur tâche d'une façon hiérarchique. Pour cela, il est nécessaire que chaque module sache précisément sur quel hôte et à quel port il pourra faire effectuer ces fonctions.

# La chaine d'acquisition

La chaine d'acquisition permet d'acquérir des données venant de tout type de support série ou réseau. Il permet de décoder le protocole, de vérifier leur intégrité et de les répartir dans différents flux de données (paramétrable en fonction du protocole de transfert). Il permet ensuite de distribuer ces flux selon trois modalités : stockage sur fichiers (traitement offline), distribution par sockets TCP (traitement online déporté / monitoring) ou distribution par mémoires partagées (traitement online haut débit). Les données sont ensuite décodées pour produire des formats ouverts et facile à analyser.

Toute la partie de répartition de données est totalement générique. La partie acquisition propose différents types de bus (Ethernet RAW, UDP ou TCP, tout périphérique série possédant un driver pour le systême hôte : RS232, USB ou autre modem).
Dans le cas où les données sont mélangées avec les paquets de contrôle, la chaîne d'acquisition met à la disposition

 

des autres modules une fonction qui permet de faire une recherche dans les paquets reçus en fonction d'un ensemble d'identificateurs.

[/IMG/acq_chain.png]
Figure 1 : Chaîne d'acquisition générique.

Nous sommes en train de développer un constructeur d'évènements (event builder) générique online qui viendra completer la chaîne.

La chaîne est complètement écrite en C dans un souci de performance. Le code est optimisé pour favoriser le débit des données. Ainsi, le programme embarque son propre ordonnanceur, donnant la possibilité de limiter les calculs lorsqu'un burst de données arrive.

Afin d'assurer la généricité de cette chaîne d'aquisition, les parties spécifiques, notamment celle qui dépendent du format des paquets, ont été isolées afin que le travail du programmeur online soit réduit au maximum. Ces parties spécifiques sont insérées dans le logiciel au moyen d'une petite librairie qui doit contenir trois fonctions de contrôle et d'extraction de données à l'API fixée.

- int prefilter(char * packet,int size) : cette fonction permet simplement de discriminer les paquets en trois catégories : Data, Control ou Junk (à détruire). Elle doit être la plus légère possible car elle est appelée à chaque fois qu'un paquet est reçu, y compris lors des bursts de données.
- int uncap(char *packet,int packet_size,char ** result,int *result_size,unsigned char *loss,unsigned char *data,unsigned char *corrupted,int *beam) : C'est la fonction d'extraction proprement dite. Elle doit s'assurer que le paquet est non corrompu et reçu dans le bon ordre. Elle effectue enfin l'opération de « décapsulation », c'est à dire la suppression de toute la syntaxe purement liée au protocole pour ne transmettre que les données brutes. Elle n'est pas appelée lors des bursts de données : les paquets sont mis en attente pour traitement ultérieur.
- int select_packet(unsigned char *packet,int id1,int id2,int id3,int id4,int id5) : cette fonction est utilisée pour retrouver des paquets de contrôle. Pour cela le programmeur dispose de 5 identifiants entiers qu'il peut utiliser à sa guise. Elle est appelée sur sollicitation d'un autre module.

L'autre partie spécifique est le décodeur des données brutes. Il peut être écrit dans un langage quelconque mais doit répondre à trois contraintes :

- Il doit recevoir ses données sur des sockets TCP ou des mémoires partagées (pour le online)
- Il doit générer un format ASCII qui soit compatible avec l'event builder (naturellement il peut générer d'autres formats en même temps si nécessaire). Sans rentrer dans les détails, chaque événement doit être décrit par un tag de temps, un tag d'espace et une valeur de mesure.
- Il doit être suffisamment rapide pour ne pas saturer le système online. En effêt, il est nécessaire que les lectures se fassent à la vitesse de production des données sans quoi la chaîne d'acquisition se sature et finit par crasher.

# Le module de commande

Le module de contrôle-commande est une machine virtuelle Python qui exécute des commandes à la demande. Pour cela, il écoute un port TCP sur lequel il attend les commandes formatées en XML.

[/IMG/cmdmod.png]
Figure 2 : Module de commande générique.

Comme on le voit sur la figure 2, le module de contrôle se compose de trois parties :

- un serveur TCP muni d'un parseur XML. C'est lui qui reçoit les commandes, les décode et les transmet à la machine virtuelle Python. Il est configuré par un fichier XML qui lui indique la correspondance entre les noms de fonctions Pyrame et les noms de fonction Python.
- La deuxième partie est la machine virtuelle Python proprement dite. Elle fournit un environnement Python standard enrichi d'un module spécifique (submod) permettant de renvoyer un résultat XML (send_res) ou d'appeler une fonction externe (exec_cmd).
- Pour appeler une fonction externe, la machine Python fait appel au troisième composant qui va formater la demande Python en XML Pyrame, envoyer la requète et transférer le résultat à la machine Python.

Outre le fichier d'implémentation Python, il est configuré par un fichier XML qui décrit quelles sont les fonctions auquel il peut répondre mais également les fonctions sur lesquelles il peut s'appuyer dans les autres modules. Pour cela, à chaque fonction est associée un hôte et un port sur lequel le module devra adresser sa requete. L'avantage d'un tel système est que la répartition du code entre les différentes machines d'un banc-test se fait naturellement. On voit ici un exemple d'un tel fichier.

```
/opt/pyrame/gclock.py
<listen_port>20002</listen_port>
write_gpib" type="host">
10.220.0.25
20001

set_clock_frequency" type="script">
set_clock_frequency
```

Ici la fonction write_gpib (de type host) est exécutée sur la machine d'adresse IP 10.220.0.25 sur le port 20001. Au contraire, la fonction set_clock_frequency (de type script) est exécutée sur la machine virtuelle locale au moyen de la fonction Python du même nom.

Voici l'implémentation de cette fonction locale qui permet de fixer la fréquence d'une horloge adressable en GPIB :

```
def set_clock_frequency(clockid,newfreq) :
 if (clockid>maxclockid) :
 submod.setres(0,"Unknown device")
 return
 command="SET FREQ %d"%(newfreq)
 retcode,res=submod.execcmd("write_gpib",link[clockid]
 ,command)
 if retcode==0 :
 submod.setres(0,"Error writing to GPIB : %s" % (res))
 return
 else :
 submod.setres(1,"ok")
```

Les paramètres en brun sont ceux qui sont envoyés dans la requète XML Pyrame :

```
set_clock_frequency">
 3
```

50Hz

3 est le numéro unique désignant cette horloge (device handler) et 50Hz la nouvelle fréquence.
ou qui sont reçus dans la réponse XML :
0>Unknown device
0 désigne l'échec de la fonction et « Unknown device » précise la raison de cet échec : l'horloge numéro 3 n'existe pas.

Les fonctions en vert submod.execcmd et submod.setres sont fournies par la machine virtuelle pour les opérations Pyrame.

- submod.execmd sert à demander l'exécution d'une fonction à un autre module. Le résultat XML de cette requète sera parsé et viendra remplir les variables retcode et res.
- submod.setres sert à renvoyer le résultat de la fonction au module Pyrame qui en a fait la demande.

Enfin les variables en bleus font partie de l'environnement de l'objet lui-même (variables globales ou attribut d'objet Python. Maxclockid est le plus grand device handler du système et link est un tableau contenant les device handlers GPIB.

# Les modules de bus et de matériels usuels

Afin de permettre au développeur de modules de se concentrer sur la partie spécifique à sa problématique, Pyrame implémente un certain nombre de modules de bas niveau permettant d'utiliser les bus usuels pour piloter les périphériques.

Ainsi, le framework dispose actuellement des modules de bus suivants :

- Serial : permet de piloter tout périphérique sur port série (RS-232) ou USB s'il dispose d'un driver sur le système hôte.
- GPIB : permet de piloter les périphériques utilisant le bus GPIB (IEEE-488) via un adaptateur Prologix USB ou Ethernet.
- Serial-Cometh : permet de piloter des périphériques série (RS-232) via un réseau ethernet à travers un adaptateur Cometh.
- Raw-ethernet : permet d'envoyer des trames Ethernet au formatage arbitraire (autre que le format IP) .
- TCP : permet d'utiliser les périphériques qui utilisent un simple socket TCP pour recevoir leurs commandes comme par exemple les appareils qui utilisent SCPI sur TCP.

Il existe un grand nombre de matériels qui sont communs à la plupart des bancs-tests électroniques. Nous proposons un ensemble de modules déjà écrits pour ces matériels :

- Alimentation bas voltage (Agilent, Hameg, Lambda)
- Alimentation haut voltage (CAEN, Keithley)
- Générateur de fonctions / pulsations(Agilent)

Cette bibliothèque s'enrichit au fur et à mesure de nos développements.

# Composition et configuration des modules

Par la composition des modules, on peut faire des systèmes d'une complexité arbitraire. Ainsi, le framework est

utilisé pour piloter un calorimètre électromagnétique ainsi que tous les périphériques et cartes électroniques nécessaires à son fonctionnement. La figure 3 montre le schéma général du contrôle-commande de cette expérience. On voit les modules Pyrame en orange, fournis à l'expérience par le framework. En vert, on voit les modules de commandes spécifiques à l'expérience, pilotant les cartes électroniques spécifiques et le PC d'acquisition.
Au niveau du contrôle commande de l'expérience, la structure hiérarchique des modules de commande permet de le ramener à une simple machine d'état, provoquant les transitions fondamentales du système et vérifiant l'intégrité de chaque sous-systèmes.

[/IMG/ctrl_cmd_calicoes.png]
Figure 3 : Contrôle-commande du SiW-Ecal/ILD.

Naturellement, une telle architecture nécessite une configuration complexe. Afin de simplifier cette opération, nous avons implémenté un système qui permet de décrire l'ensemble du système, sa connectivité ainsi que les valeurs de configuration de ses composants dans un même fichier XML.

Ce fichier est parsé, déclenchant une série de commandes de configuration vers le module Acq.PC. Ce module vérifie la cohérence de ces paramètres puis les répartit entre tous les autres modules, configurant ainsi l'ensemble du matériel.

Le fichier XML permet de représenter des matériels liés par une relation hiérarchique. Par exemple, un PC d'acquisition va gérer plusieurs cartes électroniques qui lui sont reliés. Celles-ci sont liées à des composants de lecture qui leur sont associés. On peut également déclarer autant de périphériques indépendants qu'on le souhaite. Le format est générique au sens ou l'on peut déclarer n'importe quel matériel par un nom arbitraire. Tous les paramètres doivent avoir leur nom préfixé par le nom du matériel correspondant.

Le problème d'un tel format est que le fichier peut devenir très gros si le setup est important. Afin d'en limiter la taille, il existe deux dispositifs : l'un est un système de valeurs par défaut. On remplit un fichier qui contient une valeur par défaut pour chacun des paramètres utilisés. Ainsi tous les paramètres qui conservent la valeur par défaut n'ont pas besoin d'être précisés explicitement dans le fichier de configuration.

L'autre dispositif est la déclaration implicite de composants. On peut par exemple préciser qu'à un PC est rattaché 4 cartes électroniques, puis donner des paramètres génériques qui seront appliqués à toutes les cartes sans préciser davantage. Pour permettre l'expression de ces paramètres génériques, on utilise une syntaxe d'expression régulière.

[/IMG/calxml.png]
Figure 4 : Schéma de principe de CalXml.

# Plateformes supportées et modules embarqués

Le framework est écrit en C et ne dépend que de librairies open-source (gdbm,expat). Il a été codé avec la préocupation de la portabilité. Les contrôles-commandes sont écrits en Python.

Il est conçu au départ pour tourner sur une distribution Scientific Linux qui fait autorité dans la communauté des hautes énergies. Nous fournirons d'ailleurs très bientôt une distribution binaire en RPM pour Scientific Linux 6. Pyrame peut facilement être compilé sur les systèmes unix. Seuls quelques codes d'acquisition sont spécifiques à Linux.

Il est souvent utile de pouvoir déporter une partie du contrôle-commande sur une plateforme légère. La structure

fondamentalement décentralisée de Pyrame facilite cette opération. Ainsi, Pyrame a été portée sur deux plateformes embarquées : l'Arduino et le Raspberry Pi.

L'Arduino est un petit module doté d'un micro-contrôleur avec un OS minimaliste. La version ethernet est dotée d'un contrôleur réseau et les librairies standards permettent les manipulations TCP/IP. Un module pyrame a ainsi été développé, permettant d'utiliser l'Arduino comme un DIO distant ou même un petit processeur temps-réel (avec des performances très limitées). Nous l'avons également utilisé comme une horloge basse fréquence à bas prix.

[/IMG/Arduino_Ethernet.jpg]

[/IMG/raspberry(1).jpg]
Figure 5 : Arduino Ethernet (à gauche) et Raspberry Pi (à droite)

Le Raspberry Pi est une petite plateforme ARM doté d'un système Linux. Pyrame a ainsi pu être compilé sur cette plateforme. Il peut être utilisé comme contrôleur distant pour un périphérique usb ou série ou encore un périphérique vidéo ou audio.

Afin de pouvoir interagir avec Pyrame, nous fournissons également des routines de connection dans les langages et les SCADA qui sont couramment utilisés sur de tels bancs. C'est pourquoi nous avons développé plusieurs bindings : langage C, C++, Python, langage R (statistiques).
Nous disposons également d'une commande (chkpyr.py) permettant d'accéder à un module Pyrame depuis un script shell.
Grâce à ces bindings, nous pouvons nous interfacer avec les SCADA usuels. Ainsi nous avons implémenté un binding pour Tango, OPC-UA (CTA) ainsi que XDAQ (CMS).
Vu l'ouverture complète des formats (TCP+XML), il est très facile d'implémenter un nouveau binding. Il suffit de disposer d'une librairie TCP et d'un parseur XML minimaliste.

# Les modules de service

Pour faciliter l'utilisation d'un système complexe, il est nécessaire de disposer de quelques services centraux (ou au moins régionaux). Nous avons implémenté trois de ces services :

- module de variables
- module de statistiques
- modules de signaux

Le module de variables permet de partager un espace de variables entre tous les modules Python. Le principe est simple : tous les modules peuvent demander la création d'une variable globale. Par la suite, tout autre module peut demander la valeur courante de la variable ou une mise à jour de cette valeur.

Le module de statistiques permet à tout module de mettre à jour les valeurs de variables de statistique par incrément ou par valeur fixe. Cette partie est tout à fait similaire au module de variables. Son intérêt réside dans la seconde partie du module qui implémente un système d'abonnement à certaines variables. Le module qui s'inscrit reçoit des mises à jour de la valeur des variables à temps fixe. Cela permet de décorréler le trafic de mise à jour avec le trafic

 

des clients de statistiques (typiquement des GUIs).

Le module de signaux permet d'identifier par un nom unique, des signaux qui sont déclenchés par des générateurs de pulsations répartis sur le banc test. Ainsi, tout module peut déclencher ce signal sans savoir où il se situe physiquement.

# Intérêt pour la communauté de la physique des particules

Sur les bancs tests pour l'électronique HEP, c'est généralement Labview qui est utilisé. Cet outil présente des avantages :

- son interface de programmation graphique permet de développer facilement et rapidement des contrôles-commande simples.
- Il possède une bibliothèque importante de composants déjà implémentés

et des inconvénients :

- La programmation graphique est difficilement maintenable car elle cache les structures de contrôle, surtout sur les projets importants.
- Labview est un outil assez instable qui n'est pas fait pour une utilisation prolongée.
- C'est un outil qui ne s'interface pas facilement dans des frameworks plus importants (SCADA).

Pyrame peut se positionner sur le même créneau en apportant des solutions à ces inconvénients :

- La programmation en Python est suffisamment simple pour une grande facilité d'utilisation mais également pour assurer une maintenabilité du programme.
- Pyrame est un outil très stable qui permet de faire tourner des bancs sur des périodes de temps très longues (mois).
- Naturellement, la bibliothèque de composants de Pyrame est encore petite mais elle s'étoffera si plusieurs projets l'utilisent.
- La grande simplicité du protocole Pyrame facilite grandement son interfaçage avec les SCADA.

# Conclusion

Pyrame est un système de contrôle-commande et d'acquisition de données très souple qui permet de développer très rapidement des suites logicielles pour les projets en phase de banc-test ou de prototype technique. Il est multi-plateforme incluant des plateformes embarquées, d'une nature profondément décentralisée facilitant la distribution sur les réseaux hétérogènes. Il facilite le travail du développeur online en fournissant une API très simple dans plusieurs langages usuels. Il fournit aussi de nombreux modules déjà implémentés.

Site web du projet : http://llr.in2p3.fr/sites/pyrame