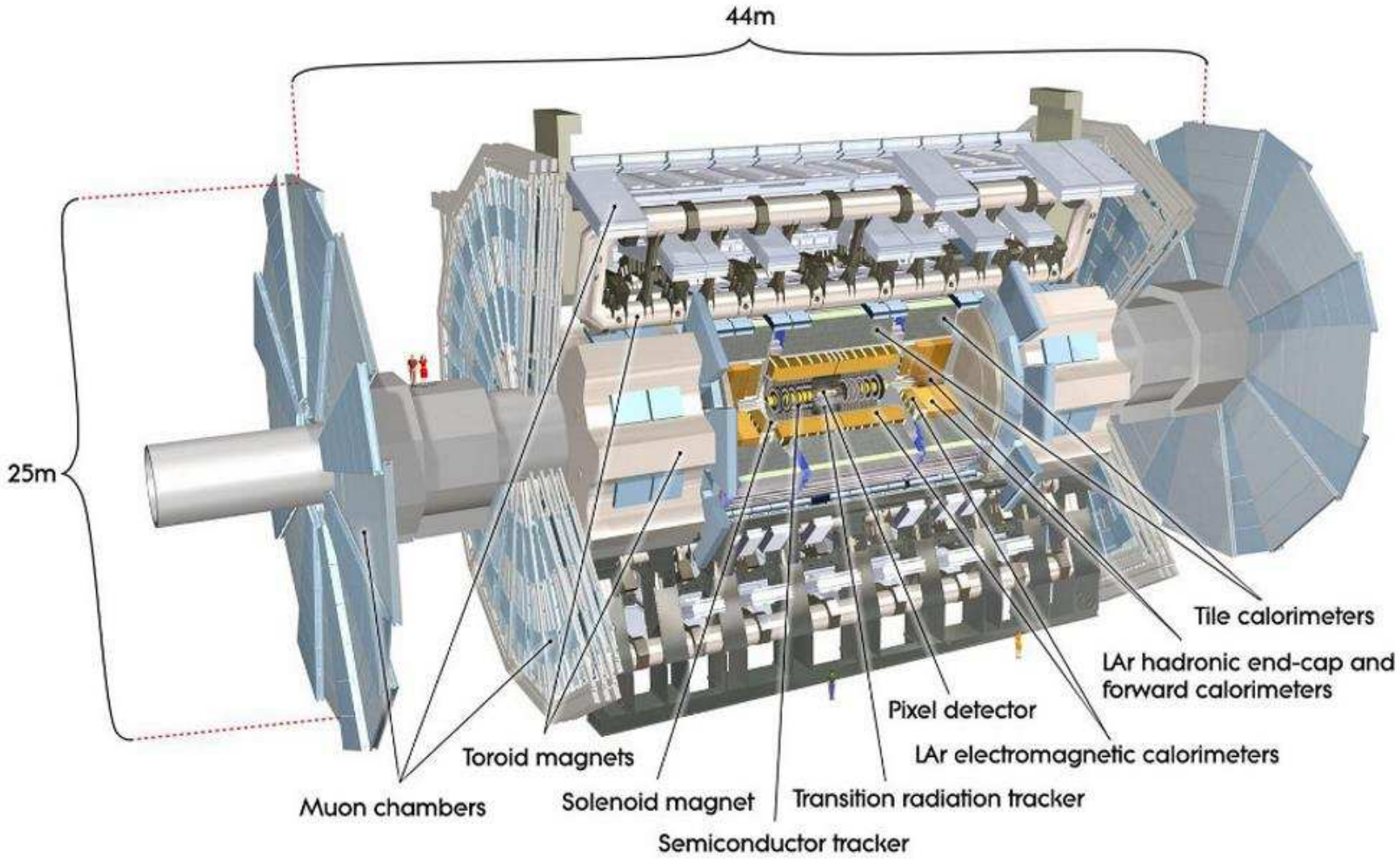
LAr Upgrade Démonstrateur et l'implémentation d'IPbus

FRAGNAUD Jasmin - LAPP - JI 2014 - 14/10/14

Détecteur ATLAS



Calorimètre a argon liquide

Evolution de l'électronique de lecture du calorimètre pour la montée en énergie du LHC.

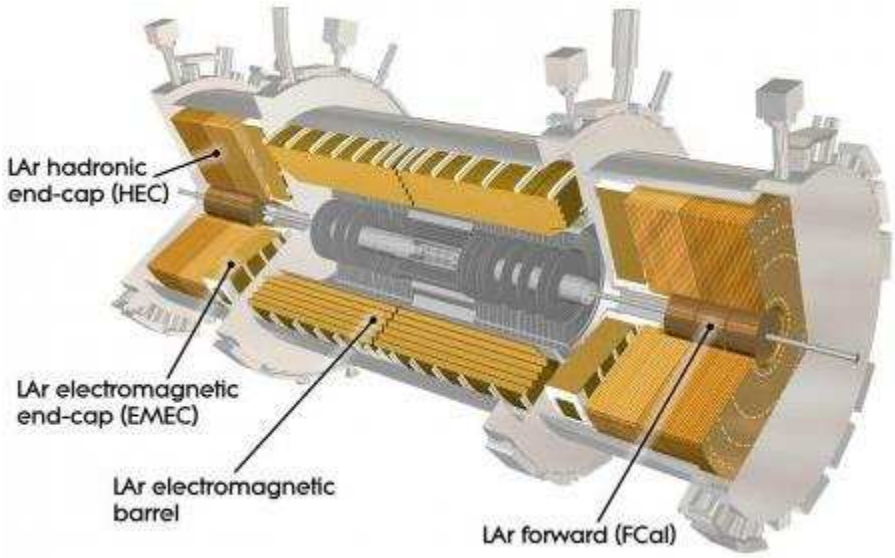
Nouvelle chaine d'acquisition.

Connexion en 10GbE, avec le protocole IPbus.

Signaux avec une meilleure granularité spatiale numérisés et traités.

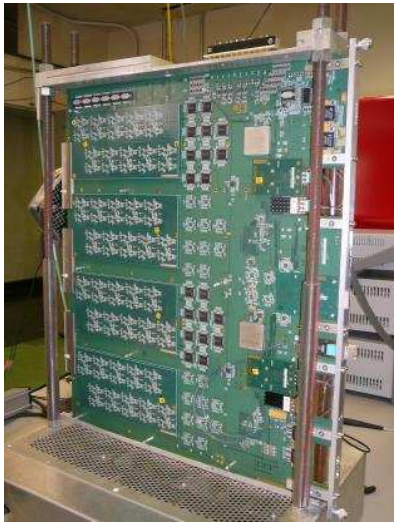
Démonstrateur installé sur le détecteur.

Installation du système complet en 2018-2019.

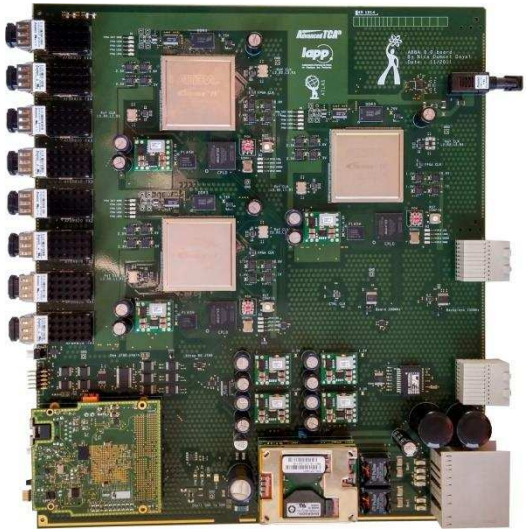


Chaine de lecture

LTDB



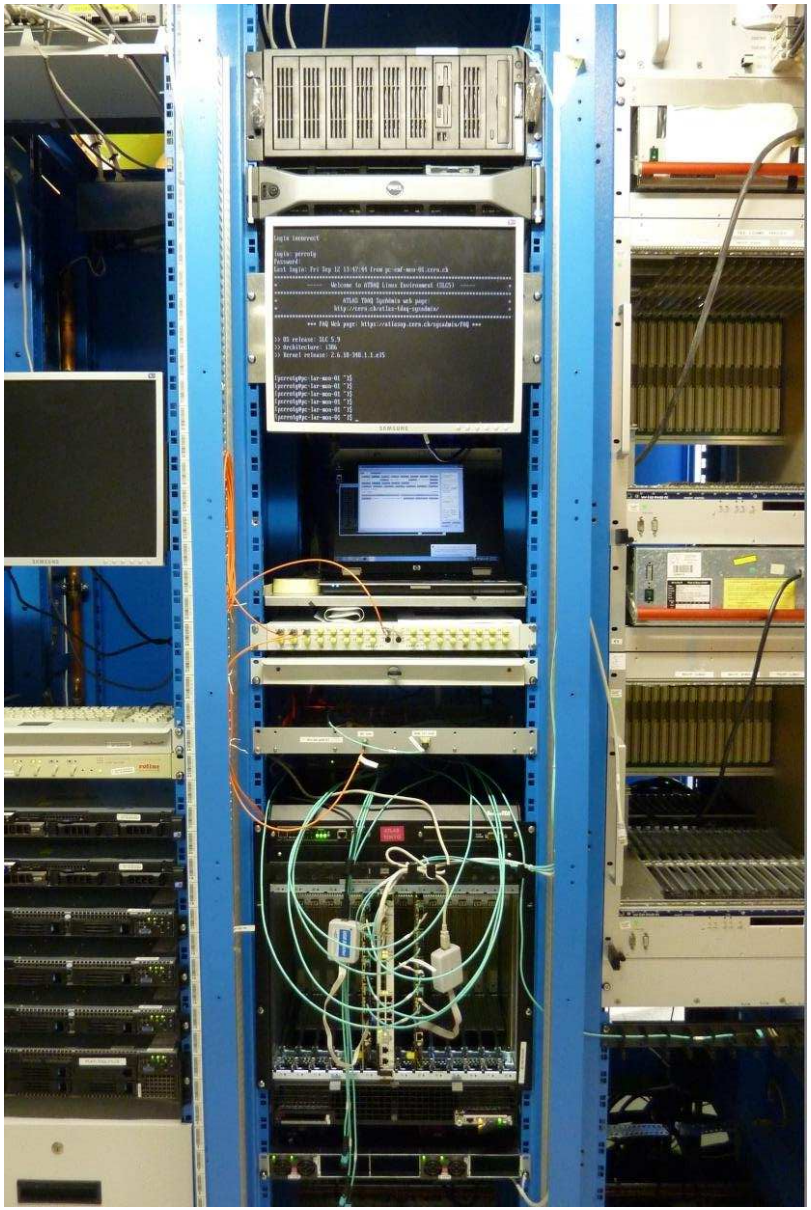
ABBA



PC

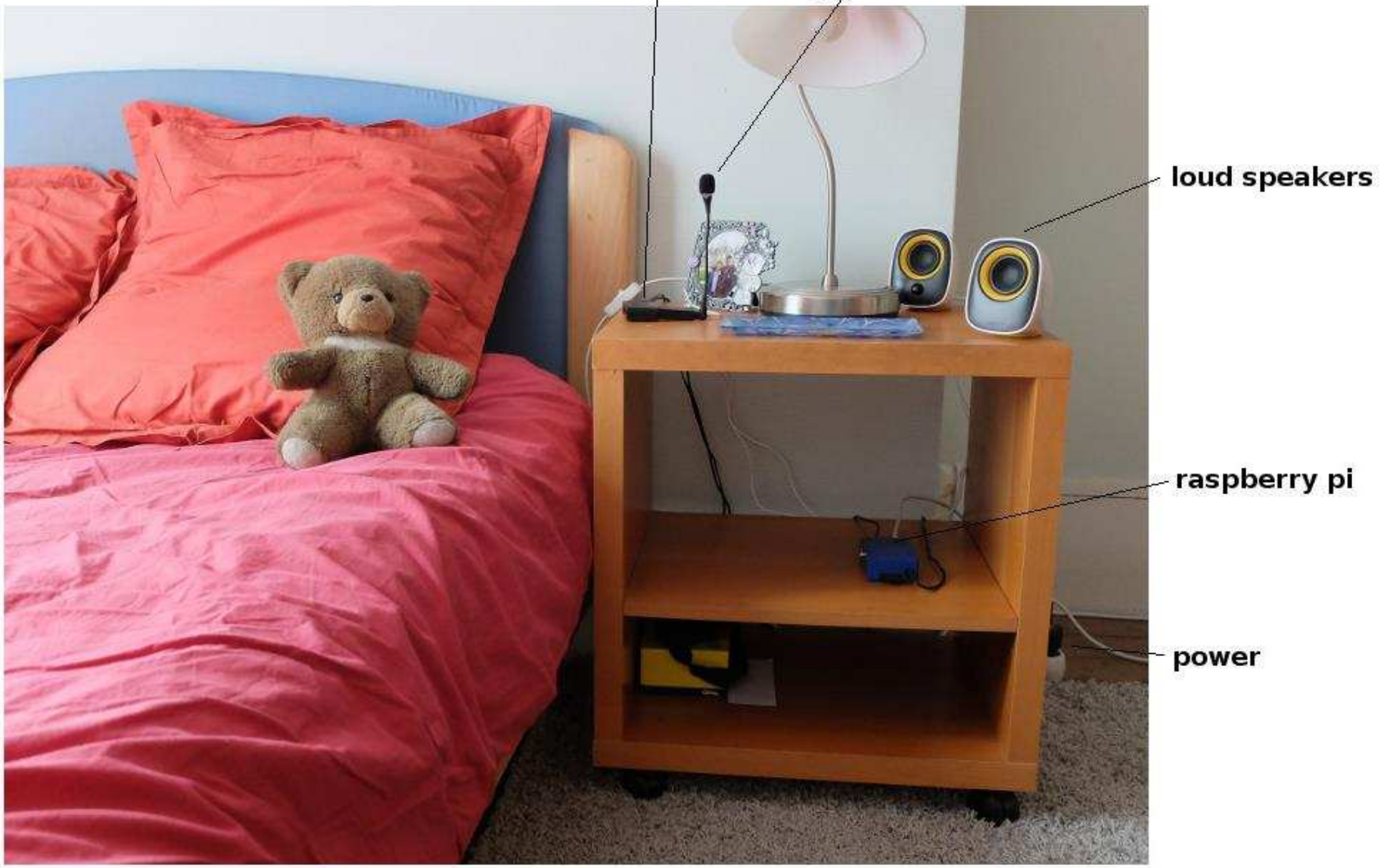
48 x 4,8 Gb → **10 GbE**

- ATLAS et L'Argon liquide
- Structure et éléments du système
- IPbus
- TDAQ
- Intégration et tests du système
- Structure mise en place



15/10/14 - 15h10

MA CHAMBRE



RADIOK WEB SITE

Fonteny Home Radio Kontroller

RadioK

Radiok est un radio-réveil internet construit avec une Raspberry Pi que j'ai installée dans ma chambre.
Il peut être contrôlé par la voix.

Le code est disponible sur GitHub.

http://www.fonteny.org/radiok

lefevre@fonteny.org

RADIOK GORY DETAILS

Raspberry Pi

Slackware

Linux

Git Hub

bash

cron

http

curl

NodeJS

javascript

C language

AngularJS

bootstrap

Carnegie Mellon University

CMU Sphinx

Jasper

sox

Audacity

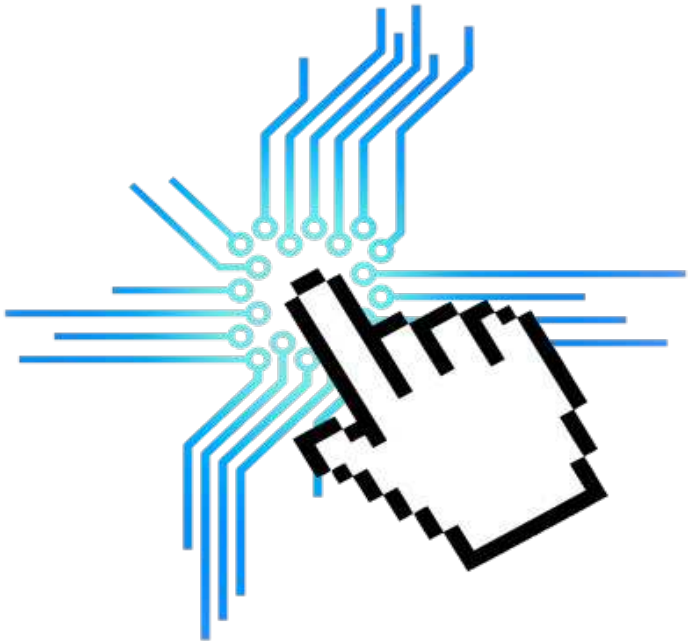
alsa

Google API

text to speech

speech recognition

json



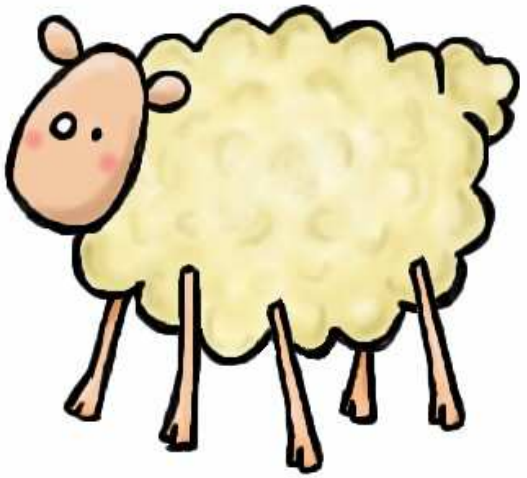
Pyrame, un framework de prototypage rapide pour systèmes online

Frédéric Magniette - LLR

Pourquoi un nouveau framework ?

Pendant les phases de R&D sur banc-test :

- Besoin d'un système très flexible
- Besoin de rapidité dans le montage
- Besoin de stabilité
- Besoin de facilité de codage
- Besoin de réutilisabilité
- Besoin de distribution sur le réseau

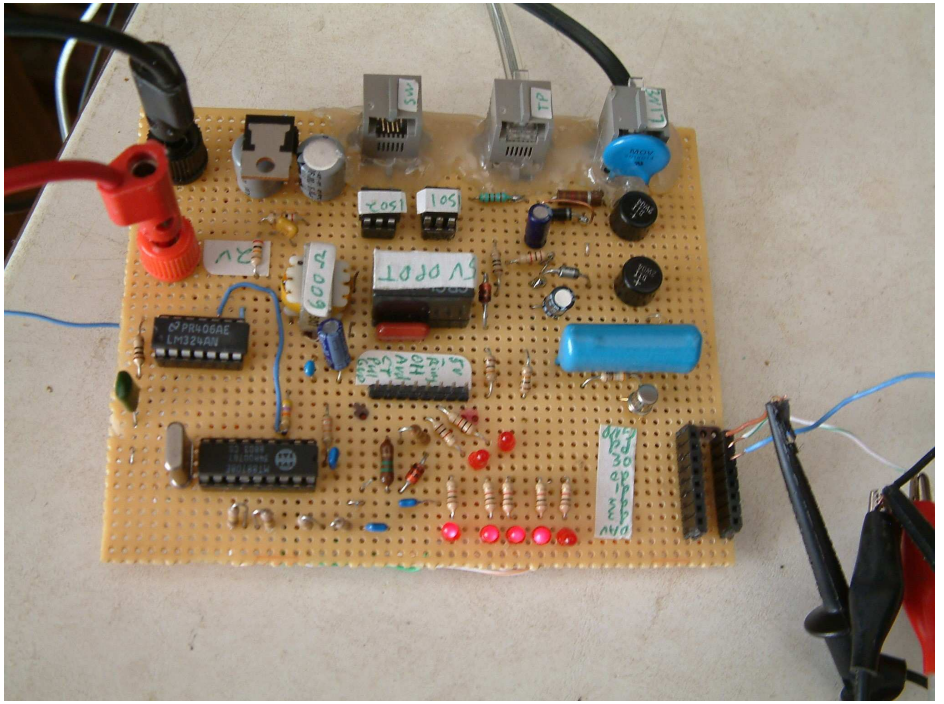


Aucun outil n'a le bon compromis : soit facile et instable, soit usine à gaz !

L'idée

S'inspirer du prototypage rapide des électroniciens :

- Un framework facile à lancer
- Des composants standards
- Des blocs sur étagères
- Des « fils »
- Des interfaces vers l'extérieur
- Un monitoring facile



La réalisation

- Un protocole réseau ouvert (TCP+XML) orienté online
- Un framework très simple à mettre en œuvre
  - Daemons en mode texte dans une console
  - Compatible avec Scientific Linux 6
- Un module d'aide à la programmation en Python
- Une quinzaine de modules matériels standards : alimentations, générateurs de patterns, motion controllers, oscilloscopes digitaux…
- Modules de services : Chaîne d'acquisition, gestion des configurations
- Interfaçage facile avec les langages et les SCADAS

Problématique

- Grande diversité de matériels et de logiciels

Problématique

- Grande diversité de matériels et de logiciels
- Soutenir les efforts du groupe DAQ IN2P3 (standardisation xTCA)

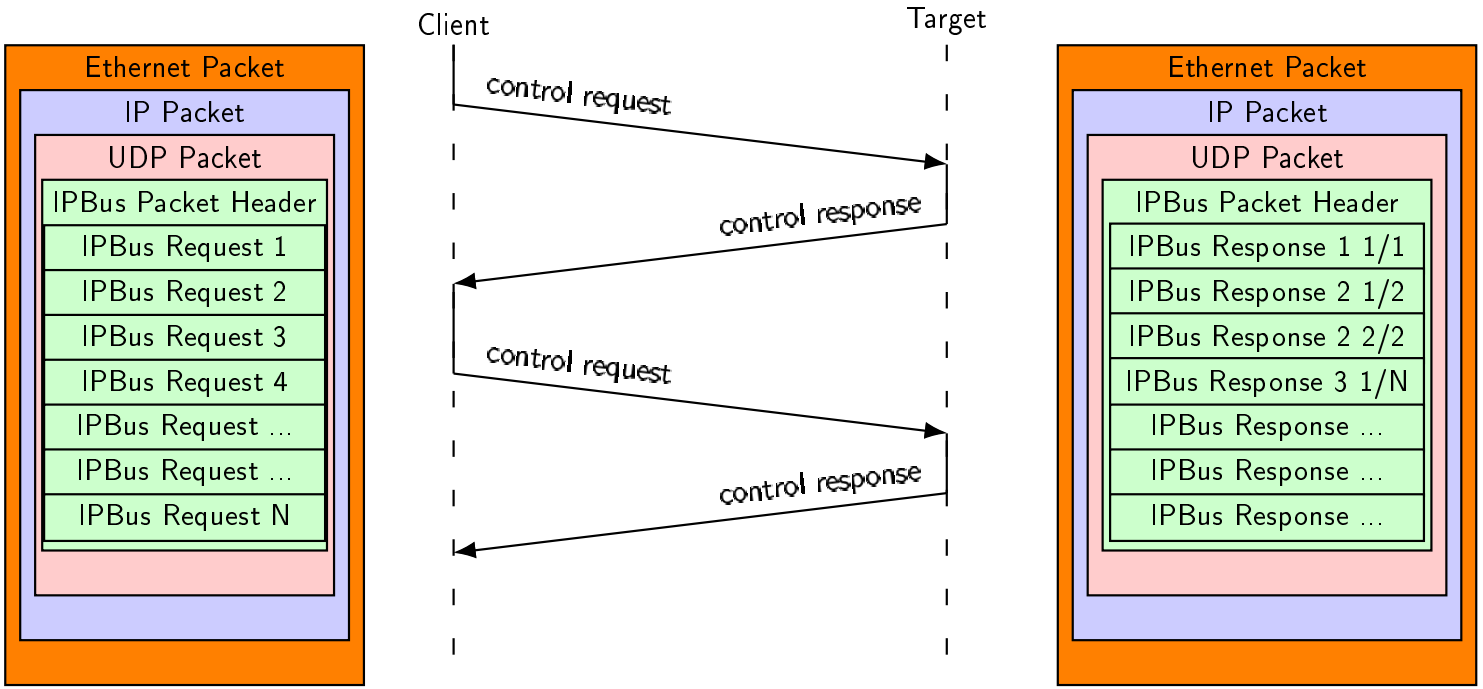
Problématique

- Grande diversité de matériels et de logiciels
- Soutenir les efforts du groupe DAQ IN2P3 (standardisation xTCA)
- Assurer un support logiciel permettant de :
  - réduire le passage du banc de test à la mise en exploitation
  - outil commun pour toutes les composantes physiques de l'IN2P3 (Astroparticules - particules - nucléaire)

Problématique

- Grande diversité de matériels et de logiciels
- Soutenir les efforts du groupe DAQ IN2P3 (standardisation xTCA)
- Assurer un support logiciel permettant de :
  - réduire le passage du banc de test à la mise en exploitation
  - outil commun pour toutes les composantes physiques de l'IN2P3 (Astroparticules - particules - nucléaire)
- Solution locale (CSNSM)
  - $\boldsymbol{ENX + NARVAL \Rightarrow DCOD}$

Principe de fonctionnement



Réalisation maquette

- Équipe électronique + logicielle
- un ingénieur de «chaque monde»

Réalisation maquette

- Équipe électronique + logicielle
- un ingénieur de «chaque monde»
- une semaine de travail
  - prise en main IP open hardware
  - ajout d'un esclave pour gérer le LCD
  - développement bibliothèque IPbus
  - développement plugin ENX

Points clefs

- Protocole Open Hardware

Points clefs

- Protocole Open Hardware
- UDP (qualité de service renforcée)

Points clefs

- Protocole Open Hardware
- UDP (qualité de service renforcée)
  - Dispose d'une suite logicielle éprouvée : uHAL
  - Portage ENX, DCOD en cours

Points clefs

- Protocole Open Hardware
- UDP (qualité de service renforcée)
  - Dispose d'une suite logicielle éprouvée : uHAL
  - Portage ENX, DCOD en cours
- **Démontre l'importance du binôme électronicien/développeur**

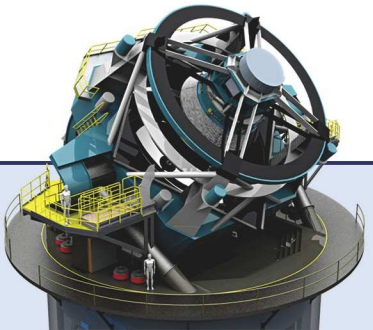
Références

- http://ohm.bu.edu/~chill90/ipbus/ipbus_protocol_v2_0.pdf
- https://svnweb.cern.ch/trac/cactus

Développement d'un framework en Java pour le contrôle-commande de la caméra du LSST

Large Synoptic Survey Telescope

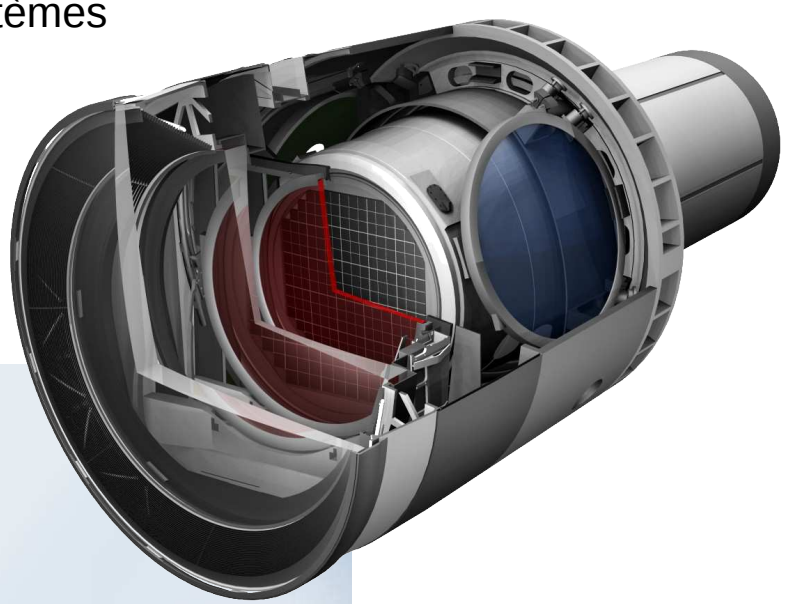
14 Octobre 2014 – JI 2014 Montpellier
Etienne Marin-Matholaz, IN2P3-APC, Paris, France

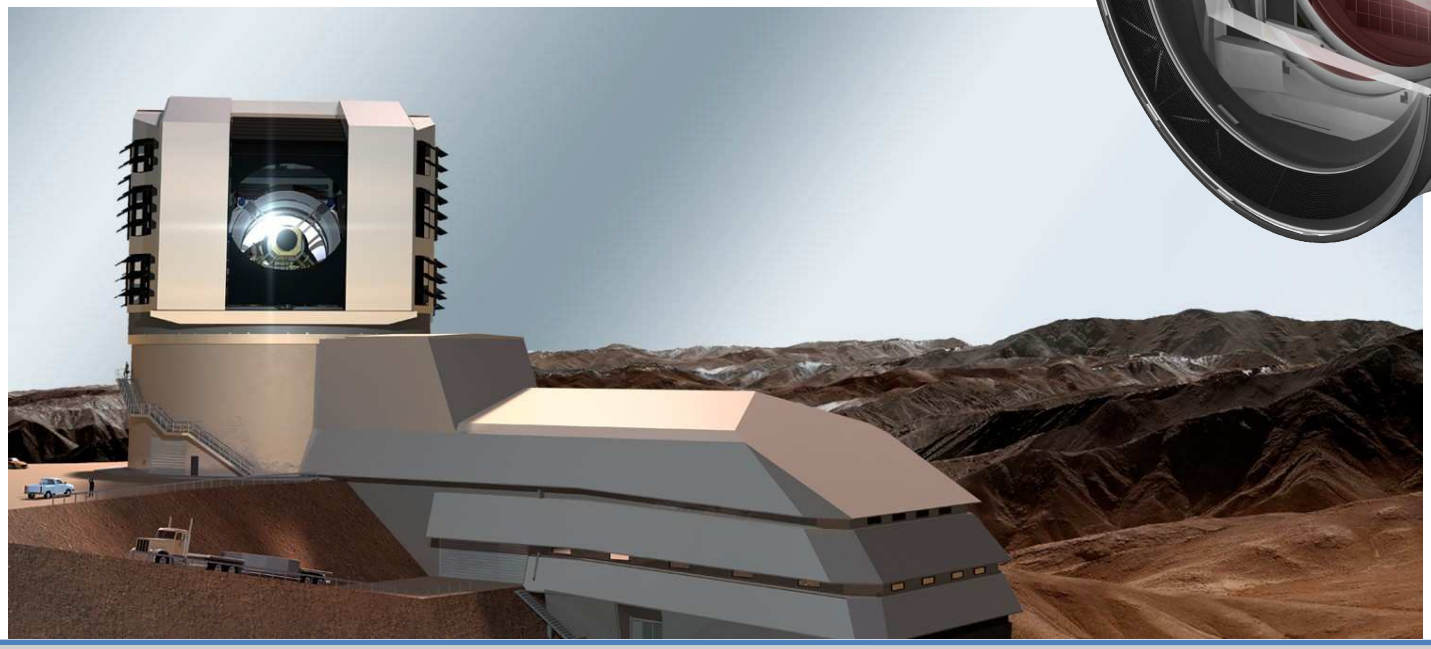


Large Synoptic Survey Telescope

- 800 photos par nuit, 2 photos de l'hémisphère sud 2 fois par semaine pendant 10 ans, premières images en 2020
- 12 To / nuit de données, ~ 30 Po de données sur 10 ans
- Caméra : assemblage d'une quinzaine de sous-systèmes

Le logiciel doit être durable et maintenable pour une durée de 30 ans !





Camera Control System Framework

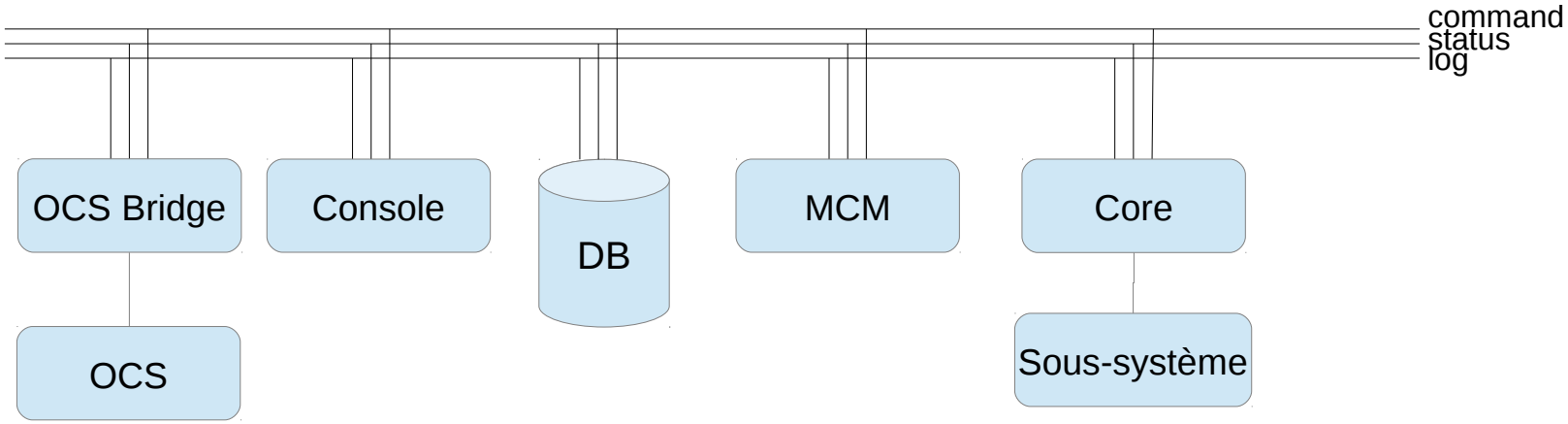
- Spécifications du CCS

→ Langage choisi : Java :
- Lisibilité et modularité
- Interfacage avec les librairies open-source
- Documentation pour l'utilisateur du télescope et le développeur de sous-systèmes (Javadoc)

→ Utilisation de librairies open-source répandues
- Communication : JMS et JGroups
- Description de sous-systèmes : Groovy
- Bases de données : Hibernate

Camera Control System Framework

- Objectifs

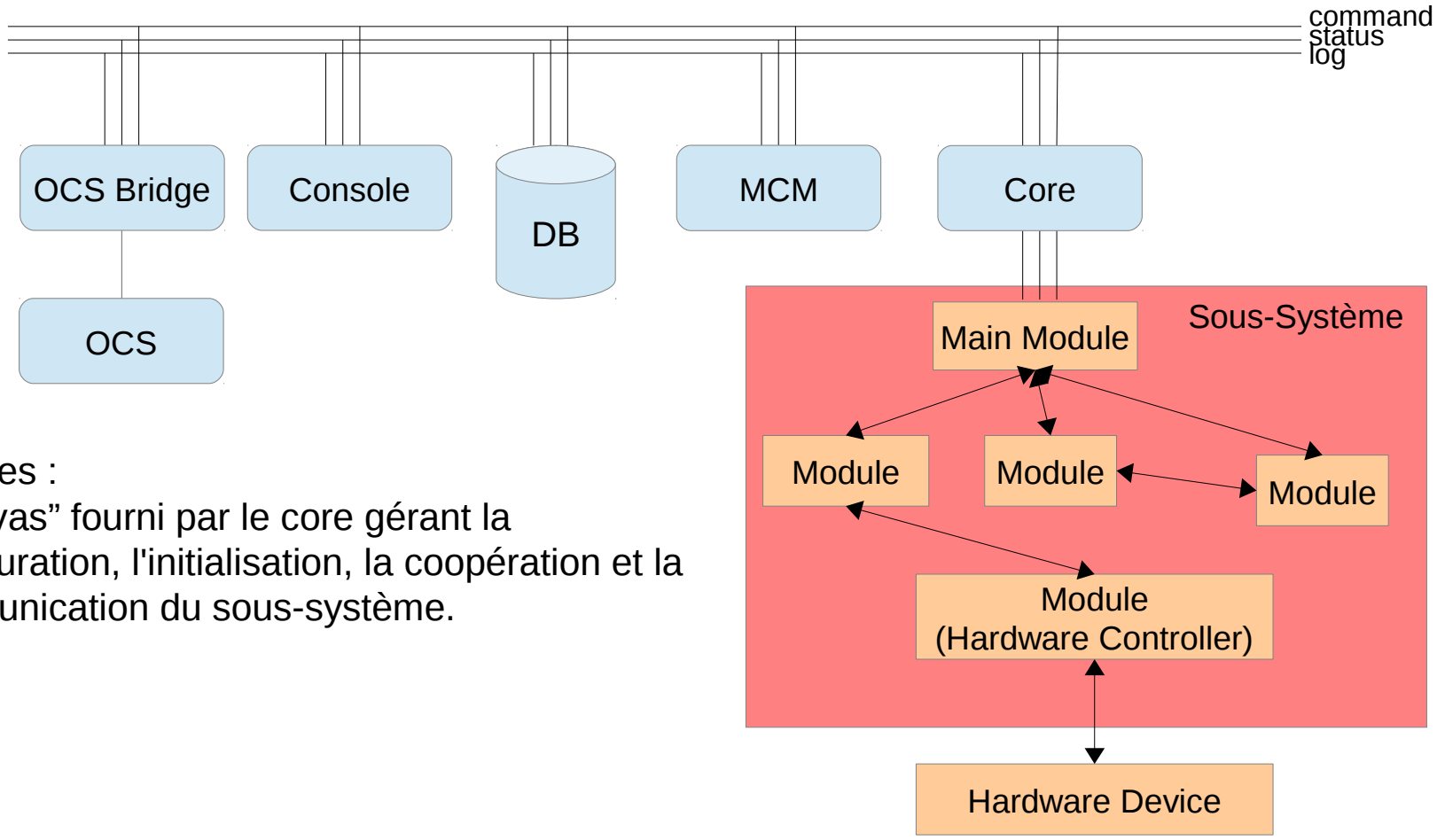


- Contrôler et coordonner les actions des différents sous-systèmes de la caméra. Il est l'interface entre le logiciel de contrôle du télescope (OCS) et les sous-systèmes de la caméra.

- Fournir des outils communs à tous les développeurs de sous systèmes :
  - Bus de communication
  - Configuration et développement de sous-systèmes
  - Bases de données de télémétrie et de configuration
  - Outils de tests (tests unitaires, tests d'intégration, logging)
  - Documentation

Camera Control System Framework

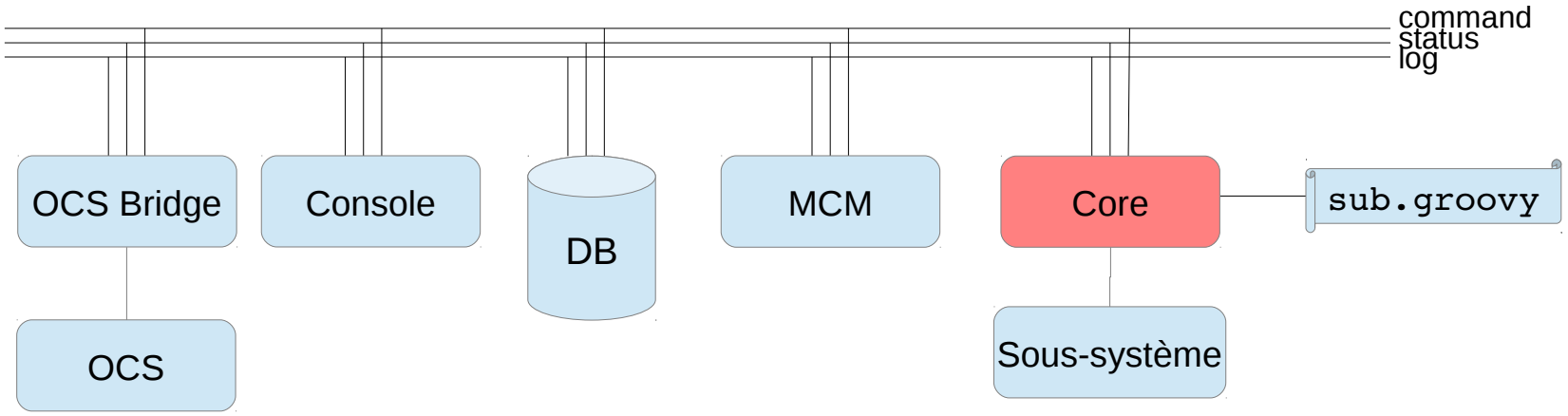
- Sous-systèmes modulaires



Modules :
“canevas” fourni par le core gérant la configuration, l'initialisation, la coopération et la communication du sous-système.

Camera Control System Framework

- Core



- Construit le sous-système modulaire décrit dans le fichier de configuration
  - `Subsystem`
  - `Module`
  - `MainModule`
  - `Configurable`
  - ...
- Gère le démarrage et l'arrêt du sous-système
- Gère l'invocation de commandes
- Permet la communication du sous-système sur les bus de communication
- Gère le déploiement de chaque sous-système

Camera Control System Framework

- Description Groovy : single filter test

```
CCSBuilder builder = ["single filter test"]

builder.
        "Main Module" ( SftMainModule,
                argMap(a("name", "main"),
                        anInt("tickMillis", 3000),
                        ref("bridge"),
                        ref("dummyFilter")
                        ))
                {
                bridge (SimuBridgeToCanOpenHardware,
                        argMap(aString("name", "bridge"),
                            anInt("tickMillis", 1000),
                            ref("clampActuatorXminus"),
                            ref("clampActuatorXplus")
                            ))
                        {//begin description of bridge's children
                                clampActuatorXminus (anImpl(Actuator, SimuClampActuatorModule),
                                    argMap( a("name", "clampActuatorXminus"),
                                    anInt("tickMillis",3000),
                                    anInt("sentCurrentMaxValue", 1780)
                                    ))

                                clampActuatorXplus (anImpl(Actuator, SimuClampActuatorModule),
                                    argMap( a("name", "clampActuatorXplus"),
                                    anInt("tickMillis",3000),
                                    anInt("sentCurrentMaxValue", 560)
                                    ))
                        }

                dummyFilter (Filter, argMap(aString("name", "dummyFilter"),
                                    aDbl("weight", "44"))) //ok

                carousel (anImpl(Carousel, SimuSftCarouselModule),
                        argMap( a("name", "carousel"),
                                anInt("tickMillis",5000),
                                //use getChildren for following lines
                                aRef("carouselMotor", "simuCarouselMotor"),
                                aRef("brake", "carouselLatch"),
                                anInt("nbSockets", socketNumber) ,
                                a("sockets", loopList(socketNumber, {ref ("socket$it")}))),

                        ))
```

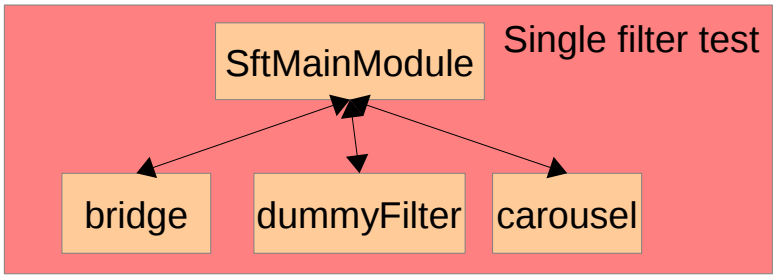
Single-filter-test.groovy

```
public class SftMainModule extends MainModule {

    private final Filter dummyFilter;
    private SftCarouselModule carousel;
    private SftAutoChangerModule autochanger;
    private boolean filterLocated;


    public SftMainModule(String aName, int aTickMillis, BridgeToHardware bridge, Filter dummyFilter) {
        super(aName, aTickMillis, bridge);
        this.dummyFilter = dummyFilter;
    }
```

SftMainModule.java



→ Le code de démarrage instancie le sous-système à partir de sa description Groovy

Camera Control System Framework

- Tests

Développé à l'aide d'un DSL basé sur Groovy

```
_withObjects toTest _method 'getAckForCommand' _group {
    _test RUN_ACTION : [ 'main', 'sleepWell', 10000] _pre { Thread.sleep(timeSleep); } _xpect {
        _failIfNot("state should be ACTIVE", subsystem.getInnerState().equals(activeState))
    }
    // we should fail when execution another ACTION
    _test FAIL_ACTION : [ 'main', 'sleepWell', 10000]  _xpect {
        _failIfNot ("action should be refused when one is already running", _result instanceof  NegativeAck)
    }
    // we should not fail sending a QUERY
    _test RUN_QUERY: [ 'main', 'querySleep', 10000]  _xpect {
        _failIf("query should run", _result instanceof  NegativeAck)
    }
    // we should not fail sending a SIGNAL
```

Conclusion

Caractéristiques du framework “maison” du CCS :

- Architecture modulaire
- Documentation présente
- Code Java lisible
- Mise à disposition de tous les outils de développement, de test et de déboguage pour le développeur de sous-systèmes et pour l'utilisateur de la caméra
- Utilisation de librairies open source fiables, découplées du reste du logiciel par l'utilisation d'interfaces

Logiciel durable et maintenable pour la durée de vie de LSST