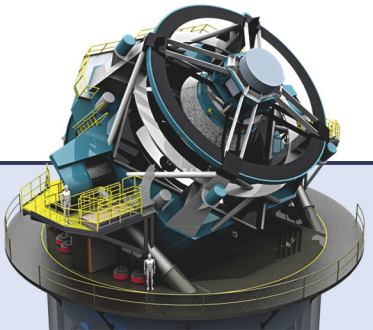
Développement d'un framework en Java pour le contrôle-commande de la caméra du LSST

Large Synoptic Survey Telescope

15 Octobre 2014 – JI 2014 Montpellier

Etienne Marin-Matholaz, IN2P3-APC, Paris, France



Plan

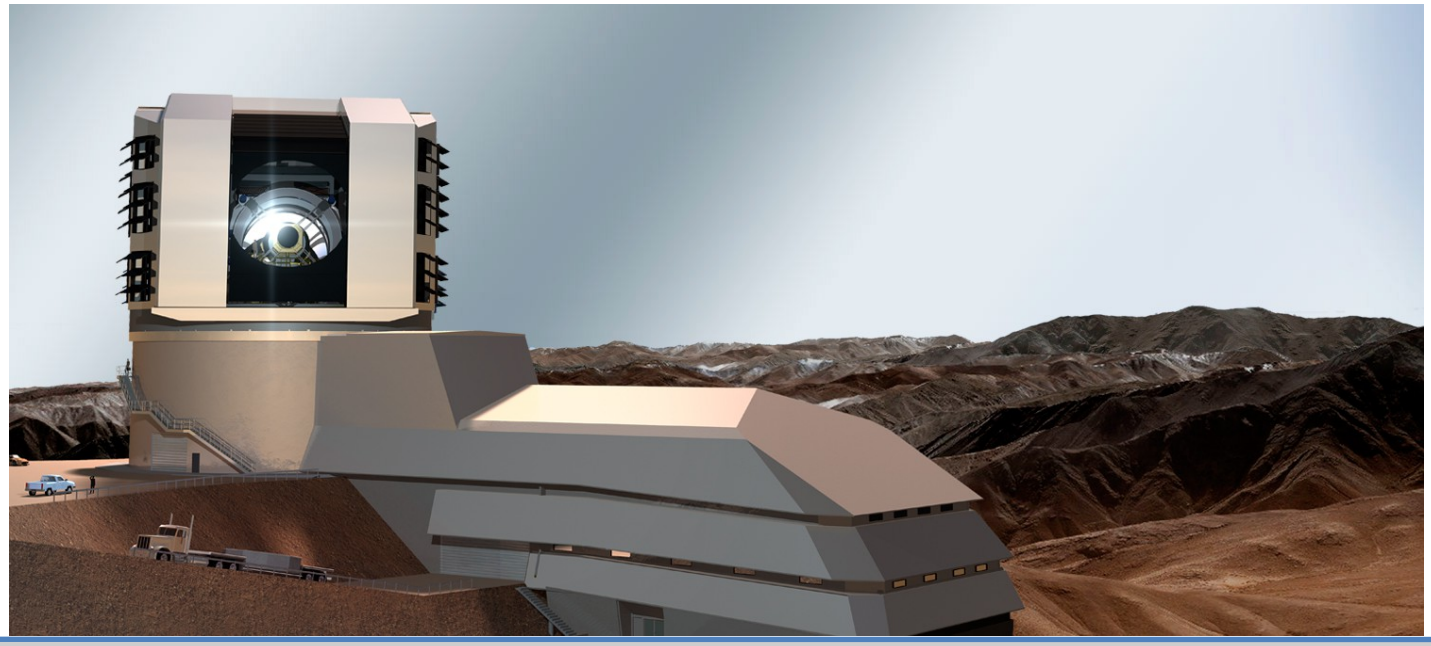
- Le LSST : Large Synoptic Survey Telescope
- La caméra
- Le Framework du CCS
  - Bus de communication
  - Sous-systèmes modulaires
  - Core : outils communs à tous les sous-systèmes
  - Bases de données
  - Consoles
  - Master Control Module
  - Logging
  - Tests
- Organisation de projet

Large Synoptic Survey Telescope

- Caméra : 3.2 GPixels
- Miroir primaire : 8.4 m
- 6 filtres
- 2 photos de l'hemisphère sud 2 fois par semaine pendant 10 ans, premières images en 2020
- 12 To / nuit de données, ~ 30 Po de données sur 10 ans

→ Obtention d'une carte précise du ciel dans le temps permettant :

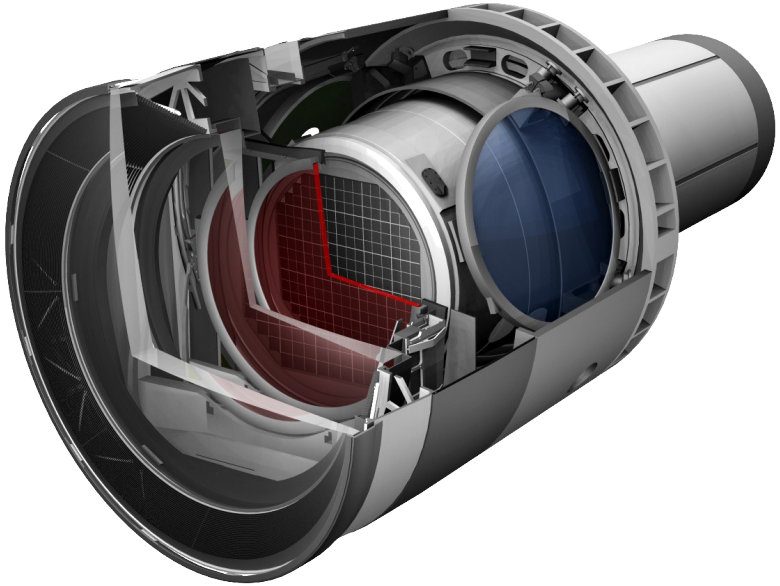
- La détection d'objets transitoires
- L'observation précise du système solaire et de la Voie Lactée
- La mise en évidence de la matière noire et de l'énergie noire

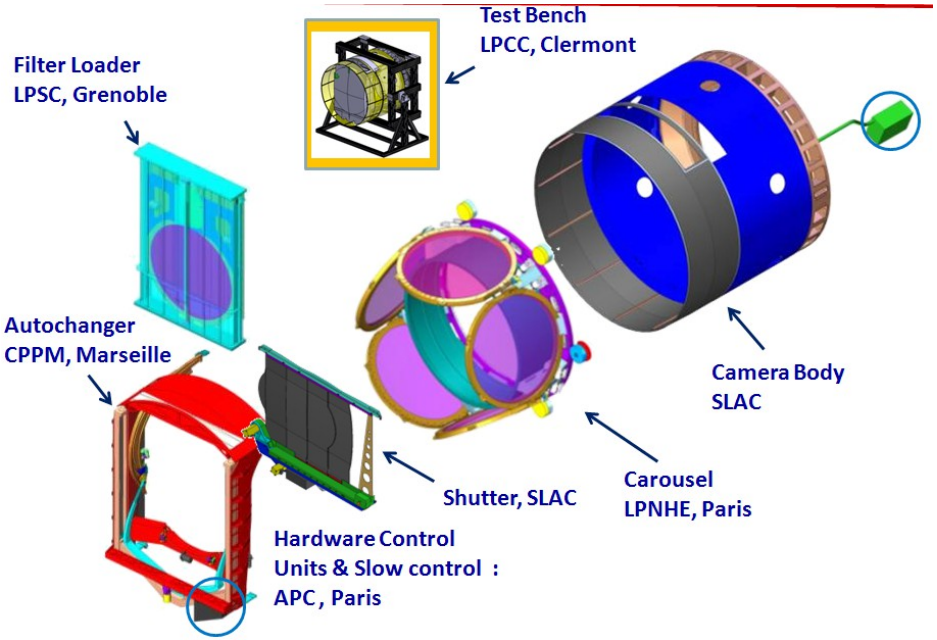


Caméra

- Architecture modulaire de 15 sous-systèmes :

- Carousel
- Changeur de filtre
- Loader
- Cryostat
- Gestion du vide
- Gestion de la puissance
- Obturateur
- Aquisition de données
- ...





- Spécifications

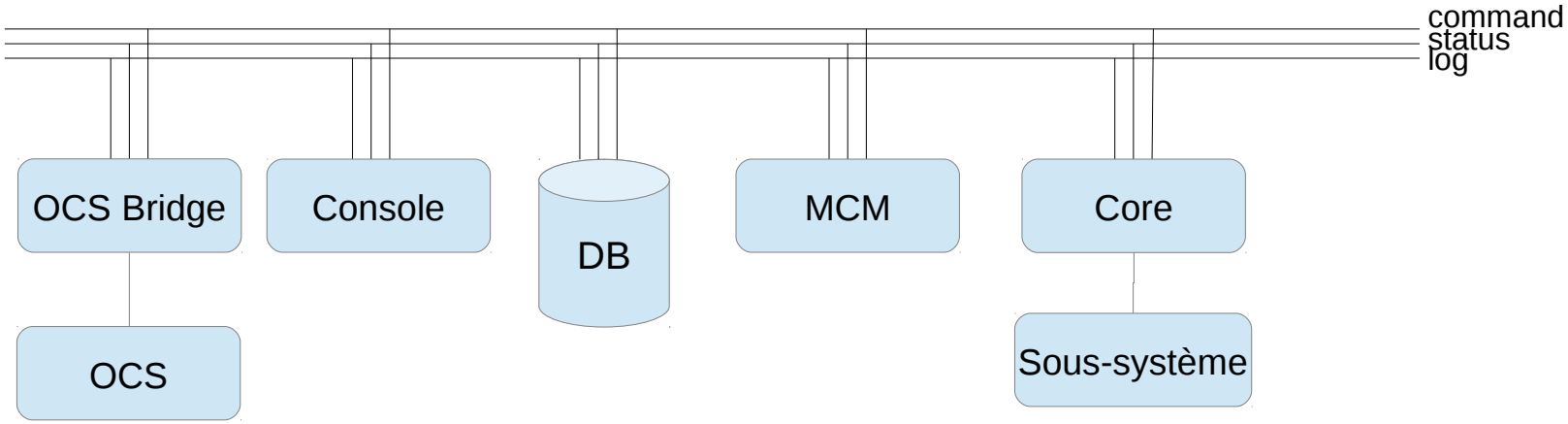
La durée de vie du logiciel est de 30 ans : le code doit être **durable** et facilement **maintenable**.

→ Langage choisi : Java :
- Lisibilité et modularité
- Interfaçage 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

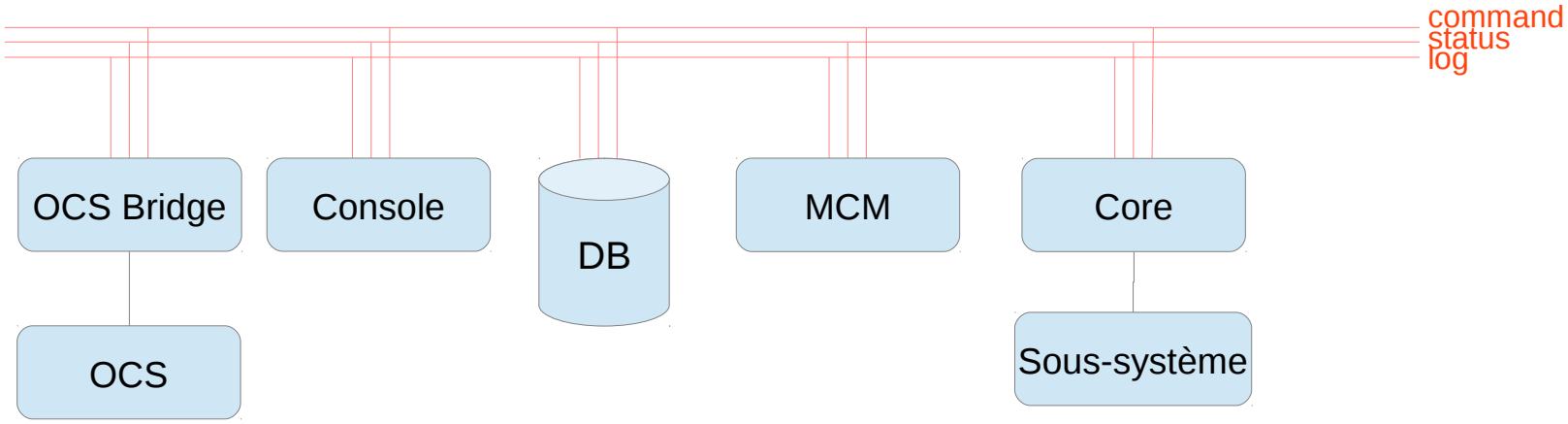


- Contrôler et coordonner les actions des différents sous-systèmes de la caméra. Il est l'interface entre le logiciel de contrôle du télescope (OCS) et les sous-systèmes de la caméra.

- Fournir des outils pour 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)
  - Gestion des messages d'erreur
  - Documentation

Camera Control System Framework

- Bus de communication

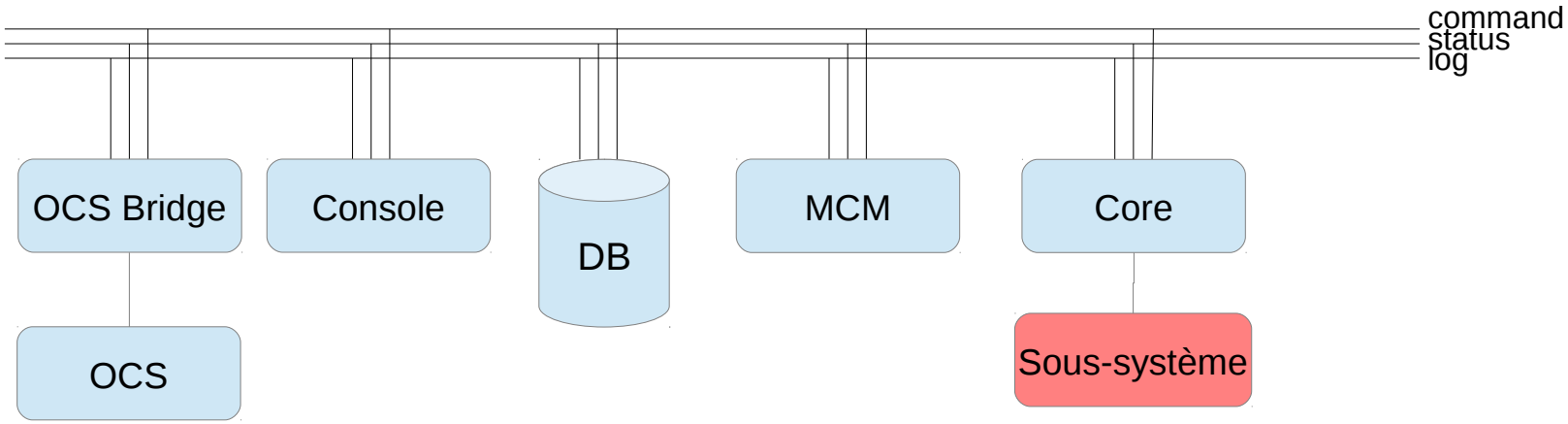


- Command : achemine les commandes vers les différents sous-systèmes
- Log : Recueille les données de chaque sous-système à des fins de détection d'erreur et de diagnostic
- Status : transmet à l'observatoire les informations de configuration et d'état des sous-systèmes

- Infrastructure “publish & subscribe”
- Bus Ethernet

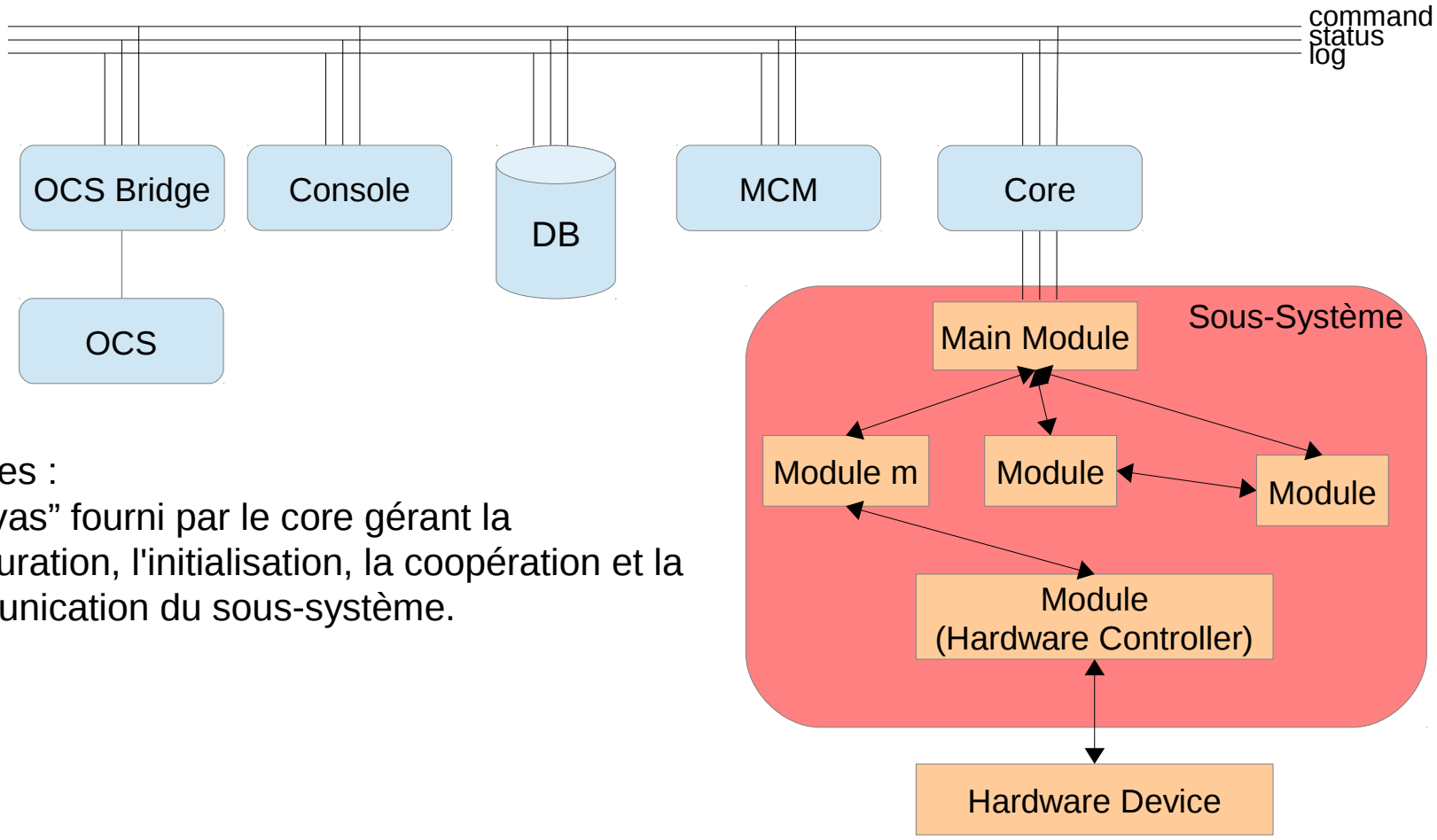
Camera Control System Framework

- Sous-systèmes modulaires



Camera Control System Framework

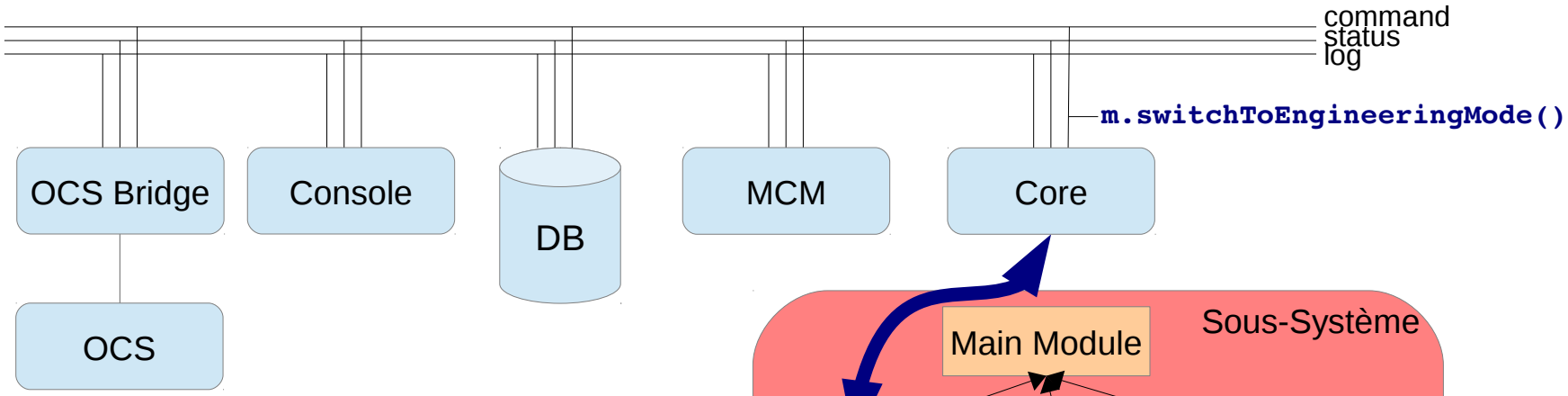
- Sous-systèmes modulaires

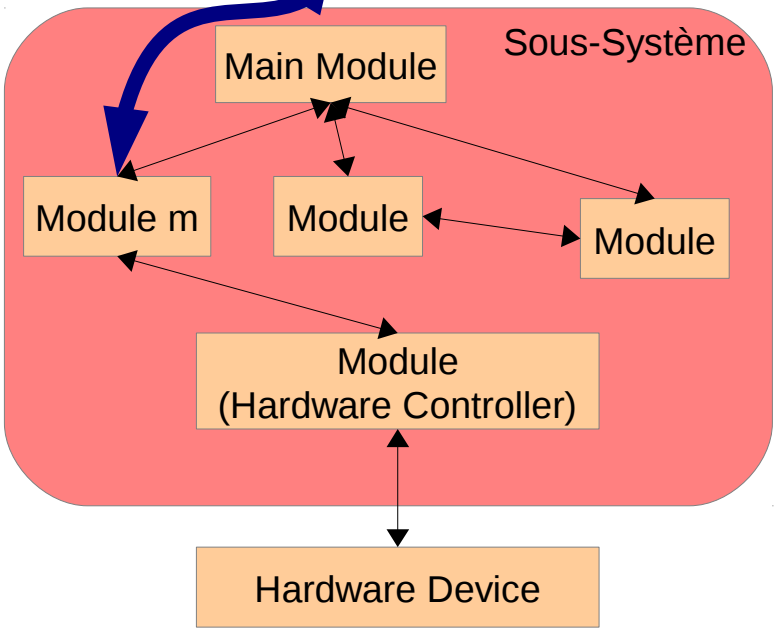


Modules :
“canevas” fourni par le core gérant la configuration, l'initialisation, la coopération et la communication du sous-système.

Camera Control System Framework

- Sous-systèmes modulaires



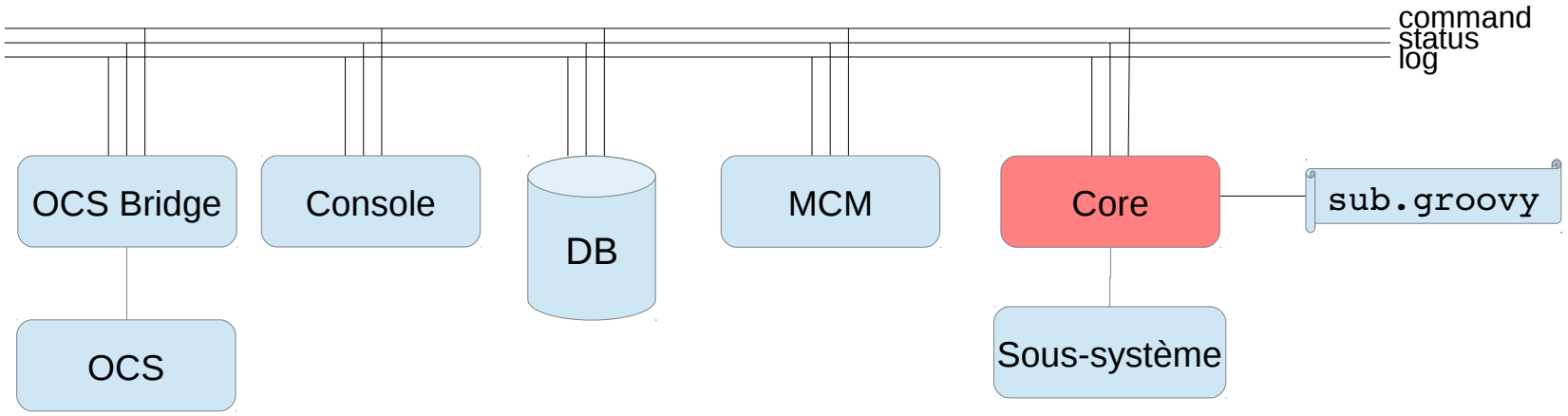


Modules :
“canevas” fourni par le core gérant la configuration, l'initialisation, la coopération et la communication du sous-système.

Engineering mode : tous les modules sont accessibles directement

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")
                            ))
                        {/*...*/}

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

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

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")}))),

                            ))
```

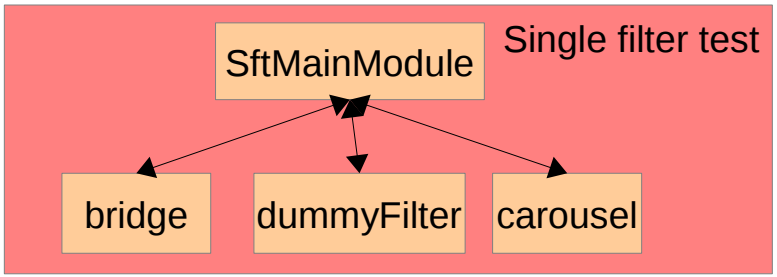
Single-filter-test.groovy

```
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;
    }
```

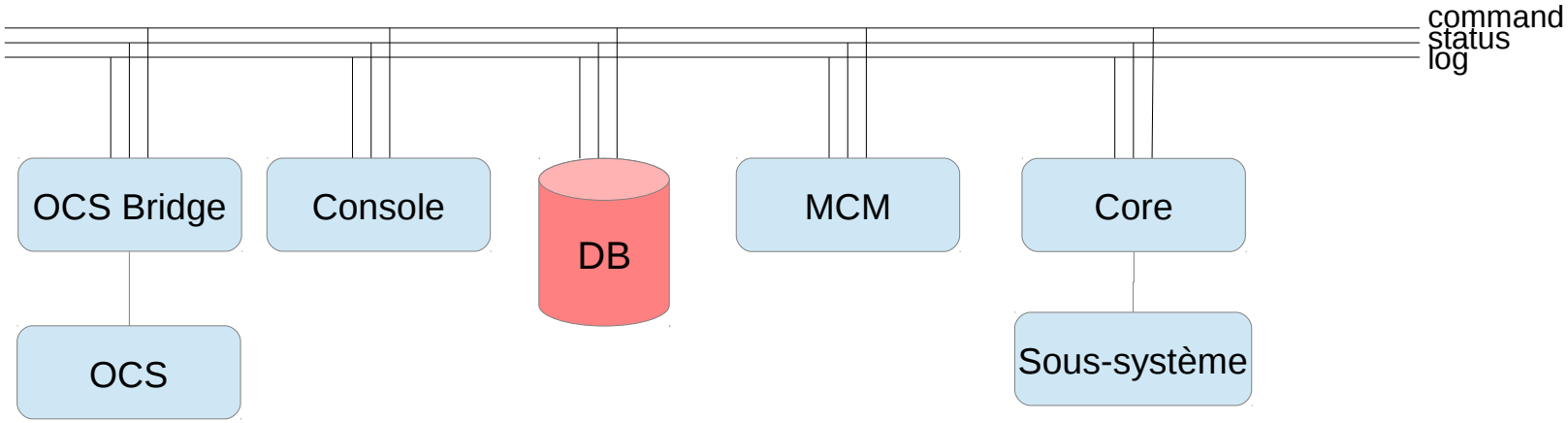
SftMainModule.java



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

Camera Control System Framework

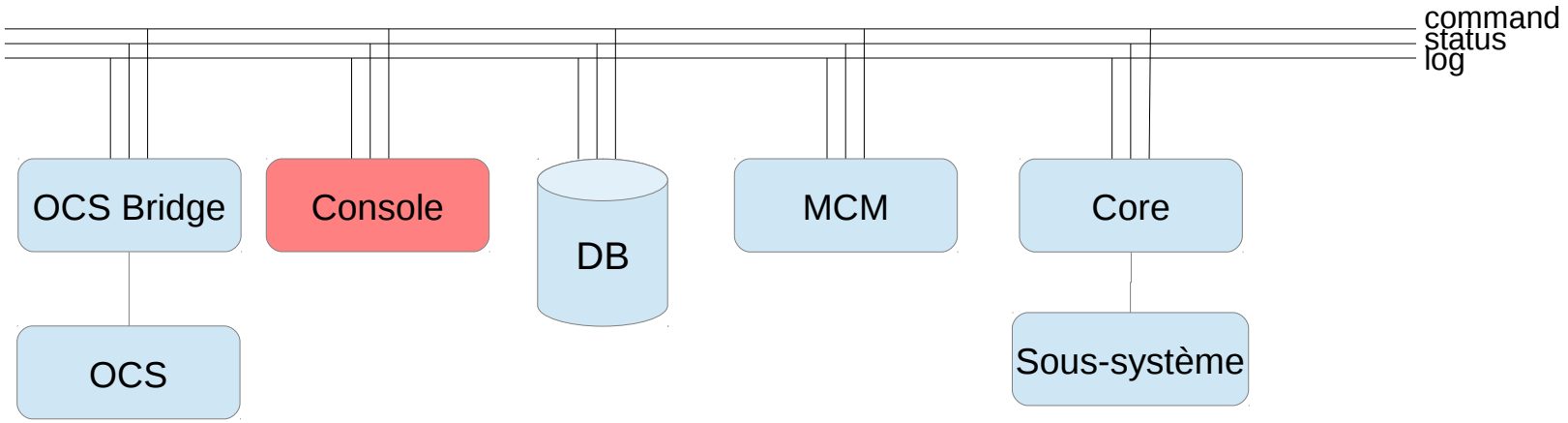
- Bases de données



- Base de donnée de configuration : enregistre la configuration d'un sous-système (configurations en cas de conditions spéciales)
- Base de donnée de télémétrie : enregistre l'état de tous les sous-systèmes à intervalles réguliers, par l'intermédiaire du bus de status

Camera Control System Framework

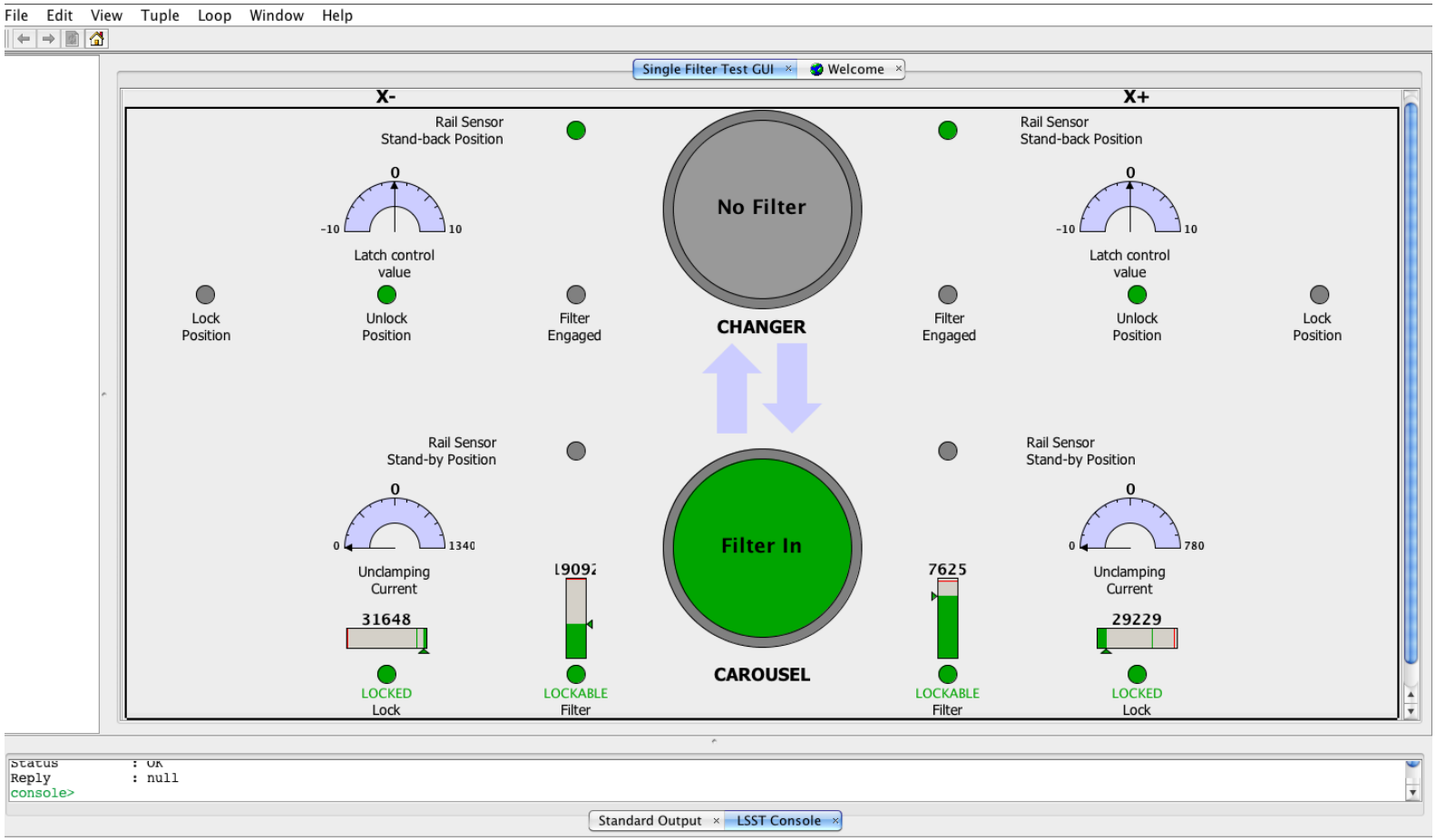
- Consoles



- Console JAS :
  - Affichage de statistiques, graphiques sur des valeurs de capteurs enregistrées dans la base de donnée de télémétrie
  - Gestion des interfaces graphiques des sous-systèmes
- LogTracer
- Console d'invocation de commandes
- ...

Camera Control System Framework

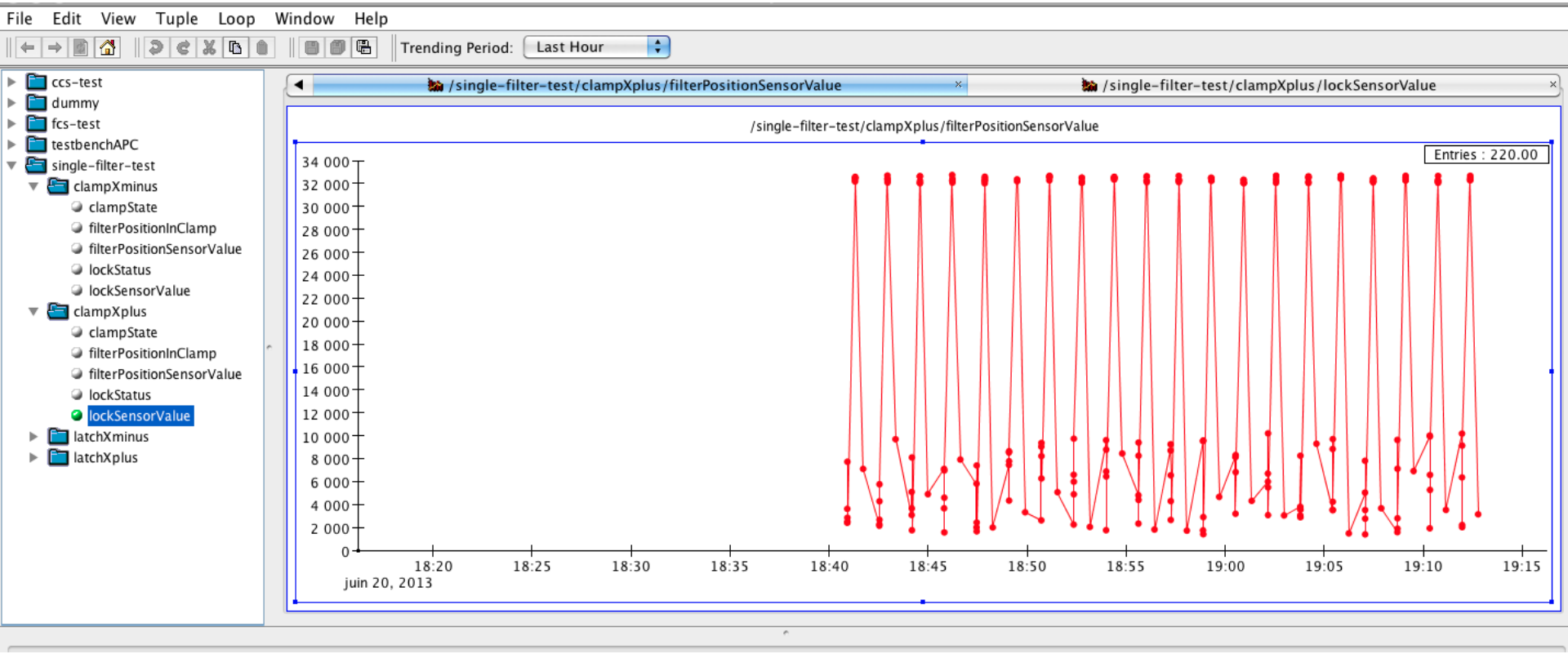
- Exemple de console : single-filter-test



Camera Control System Framework

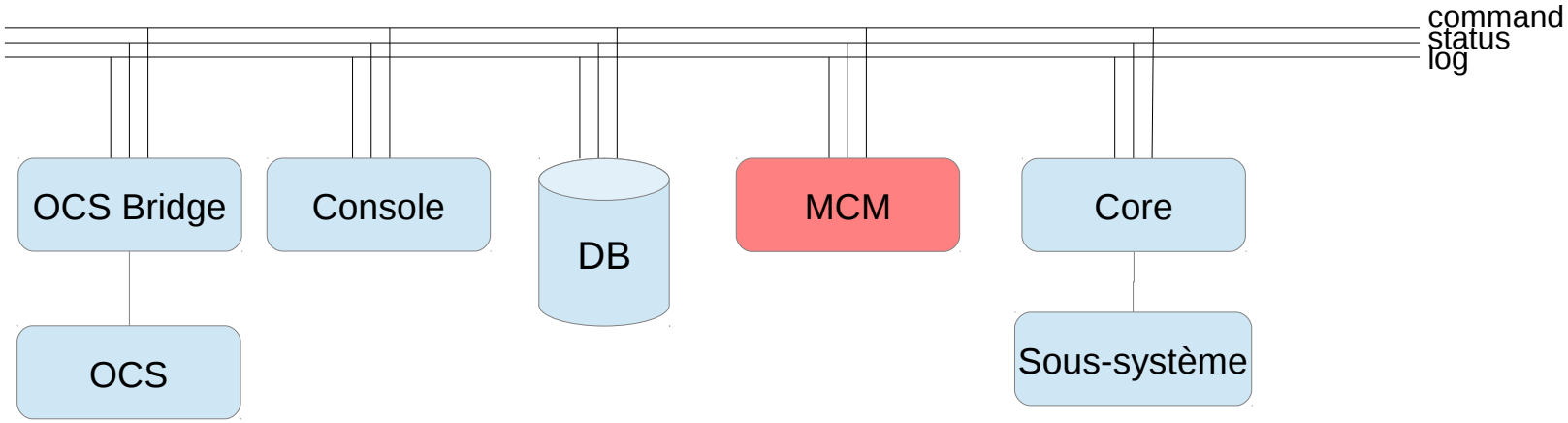
- Exemple de console : JAS Console

  Courbe des valeurs mesurées par le capteur de position d'un filtre dans le verrou qui fixe le filtre sur le carrousel (Filter Clamp Position Sensor Values)



Camera Control System Framework

- Master Control Module



“Chef d'orchestre” du CCS :

- Envoi de commandes haut niveau.
- Décompose la commande pour les différents sous-systèmes.
- Gère la cohérence de l'envoi et de la réception des réponses.

Exemple : “Positionner le filtre r devant la caméra”

- Logging

- Utilisation de notre propre système de logging, compatible avec les existants (java.util.logging, log4j)

- Possibilité de régler le niveau de log en fonction :
  - De la classe java
  - D'un “thème” (logging transversal)

- Possibilité de changer dynamiquement le niveau de log

Camera Control System Framework

- Tests

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

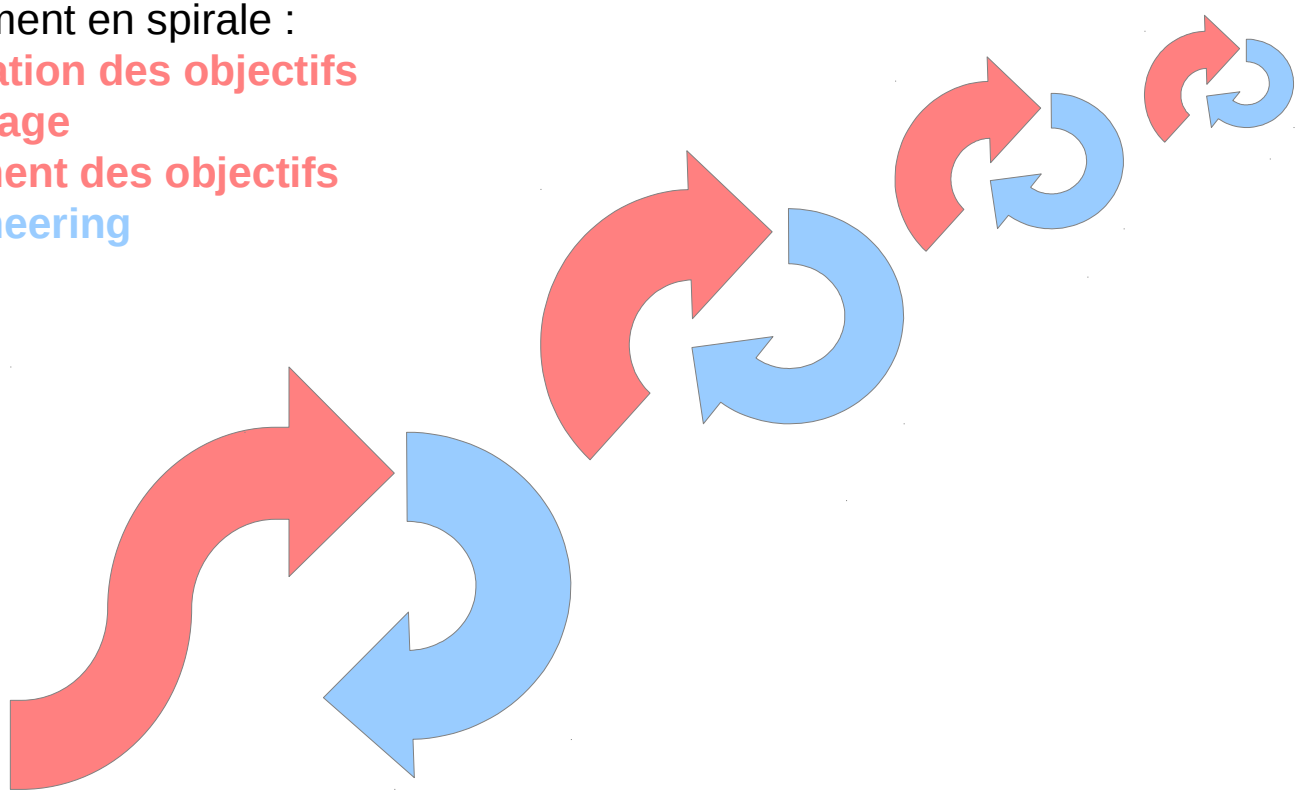
```
_withObjects toTest _method 'getAckForCommand' _group {
    _test RUN_ACTION : [ 'main', 'sleepWell', 10000] _pre { Thread.sleep(timeSleep); } _xpect {
        _failIfNot("state should be ACTIVE", subsystem.getInnerState().equals(activeState))
    }
    // we should fail when execution another ACTION
    _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
```

Management de projet

- Processus de développement

Développement en spirale :

- **Identification des objectifs**
- **Prototypage**
- **Raffinement des objectifs**
- **Re-engineering**
- **Tests ...**



Management de projet

- Outils informatiques

- Gestionnaire de version : **Subversion**
- Gestion des builds et centralisation des tests : **Jenkins**
- Wiki collaboratif : **Confluence**
- Organisation du code en projets modulaires **Maven** : permet de gérer les versions de dépendances de librairies externes et internes.
  → Les développeurs de sous-systèmes peuvent travailler en parallèle des développeurs du Core
- Organisation des versions du Core en Snapshot/Release.
  → Les développeurs de sous-systèmes disposent de versions stables pour aller sur les bancs de tests

Management du projet

- Equipe

- A **SLAC** :
  - Tony Johnson : chef de projet technique pour tout le CCS consoles, “trending data”
  - Max Turri : GUI pour le FCS, “trending data”
  - Stuart Marshall : system engineering
  - Owen Saxton : développeur des logiciels de contrôles pour le “shutter”, le refrigeration system, les bancs de test de l’électronique
  - Heather Kelly : banc de test des CCD
- A l’**APC** :
  - Eric Aubourg : physicien référent, auteur des concepts de base du CCS
  - Bernard Amade : développeur du framework
  - Etienne Marin-Matholaz : développeur du framework
  - Françoise Virieux : développeur du FCS (Changeur de filtre)
- Au **LPNHE** :
  - Eduardo Sepulveda, Diego Terront : banc de test des CCD

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