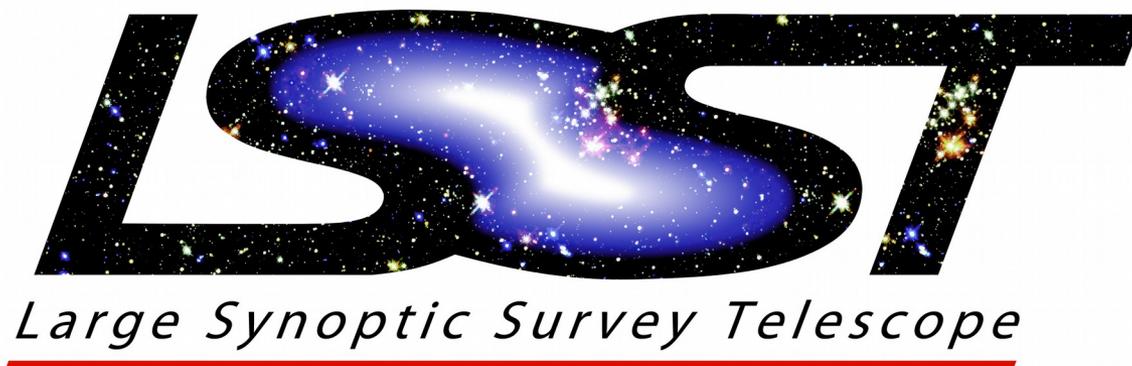
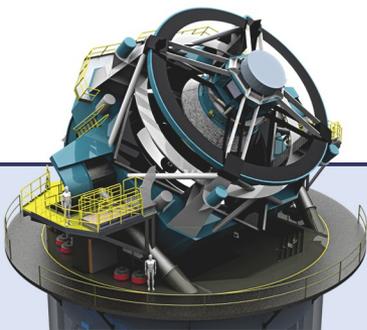




Développement d'un framework en Java pour le contrôle-commande de la caméra du LSST



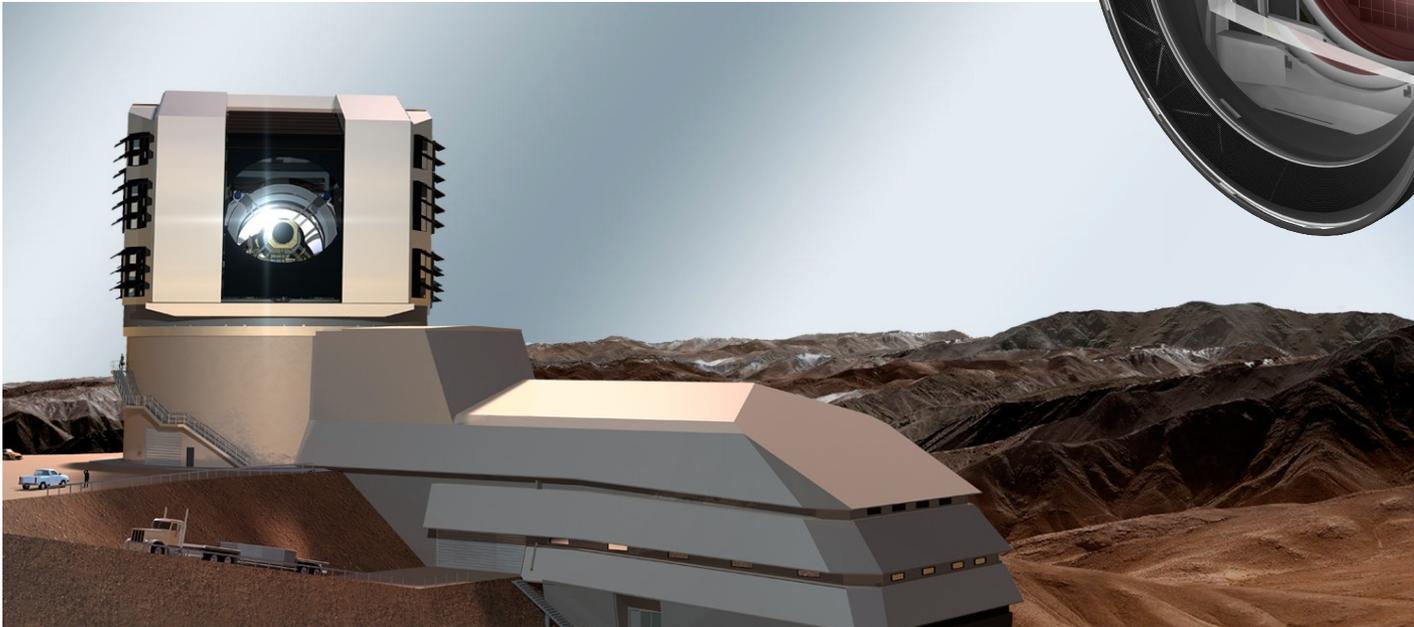
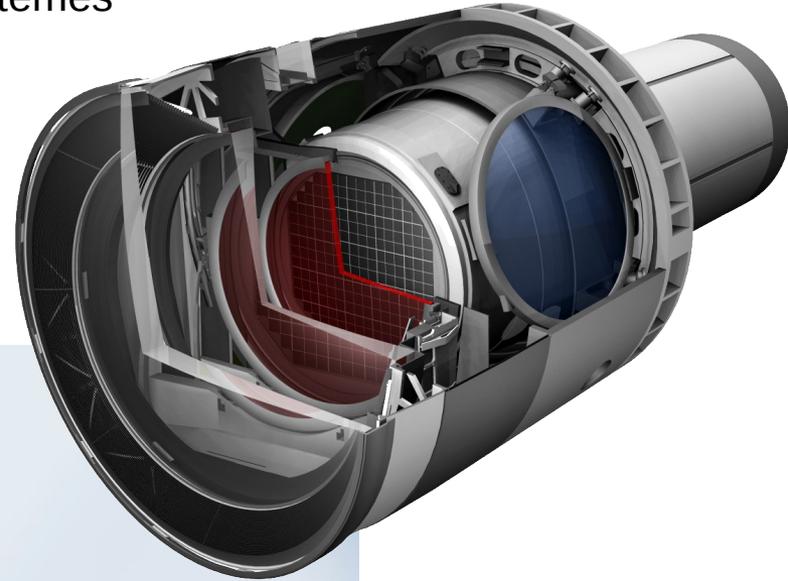
14 Octobre 2014 – JI 2014 Montpellier
Etienne Marin-Matholaz, IN2P3-APC, Paris, France



Large Synoptic Survey Telescope

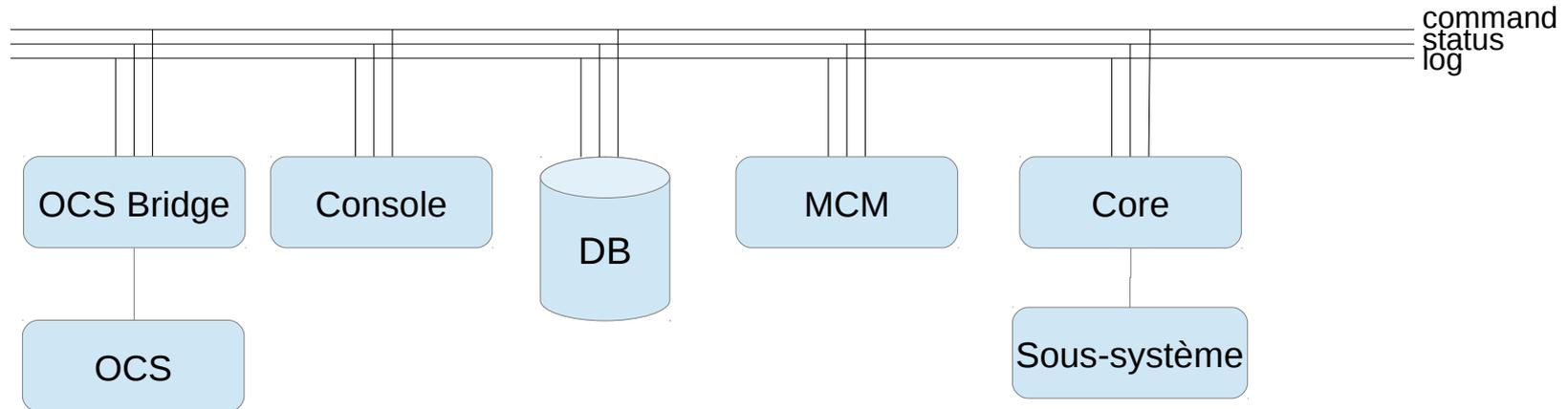
- 800 photos par nuit, 2 photos de l'hémisphère sud 2 fois par semaine pendant 10 ans, premières images en 2020
- 12 To / nuit de données, ~ 30 Po de données sur 10 ans
- Caméra : assemblage d'une quinzaine de sous-systèmes

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



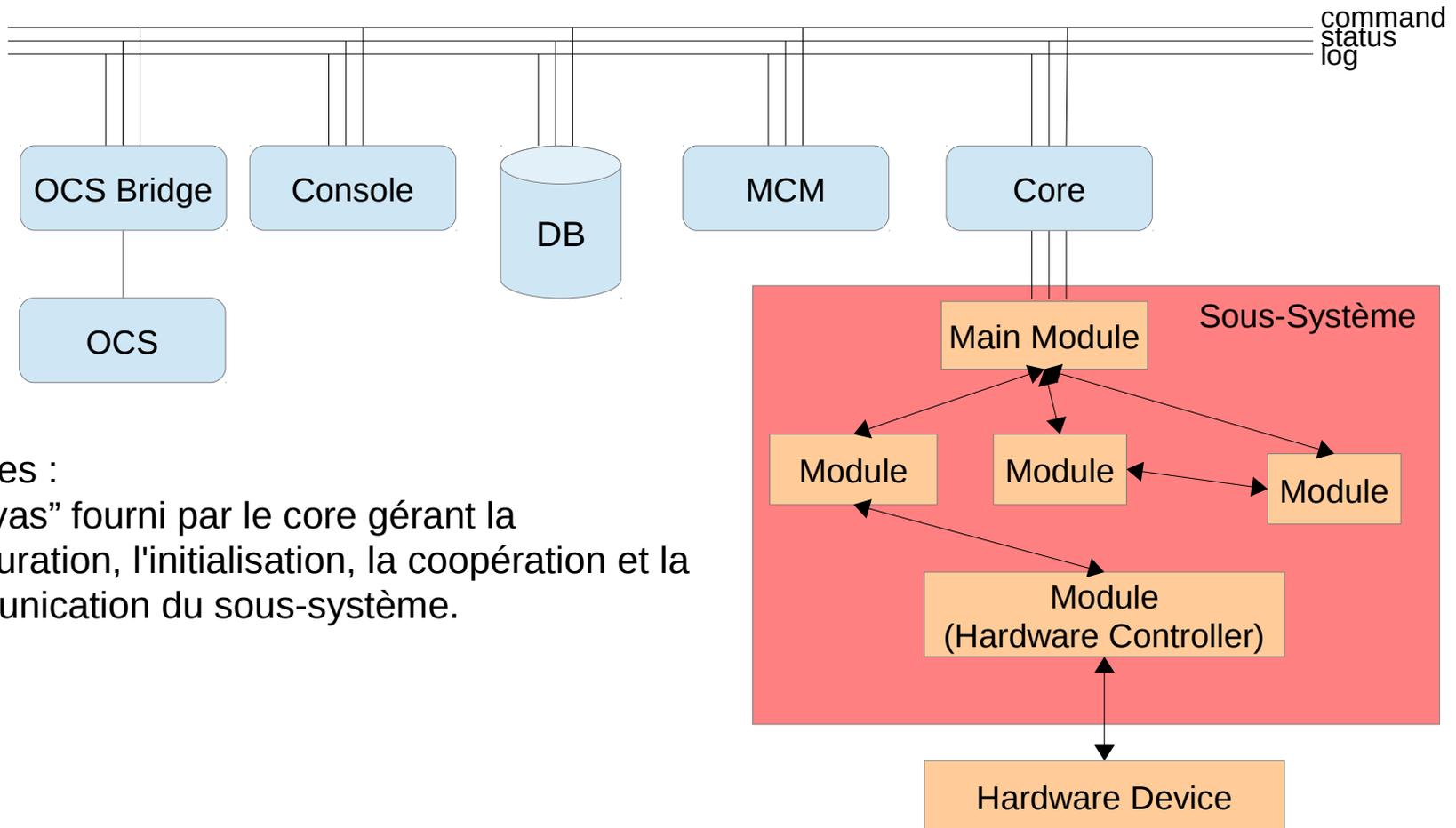
- Spécifications du CCS
 - Langage choisi : Java :
 - Lisibilité et modularité
 - Interfacage avec les librairies open-source
 - Documentation pour l'utilisateur du télescope et le développeur de sous-systèmes (Javadoc)
 - Utilisation de librairies open-source répandues
 - Communication : JMS et JGroups
 - Description de sous-systèmes : Groovy
 - Bases de données : Hibernate

- Objectifs



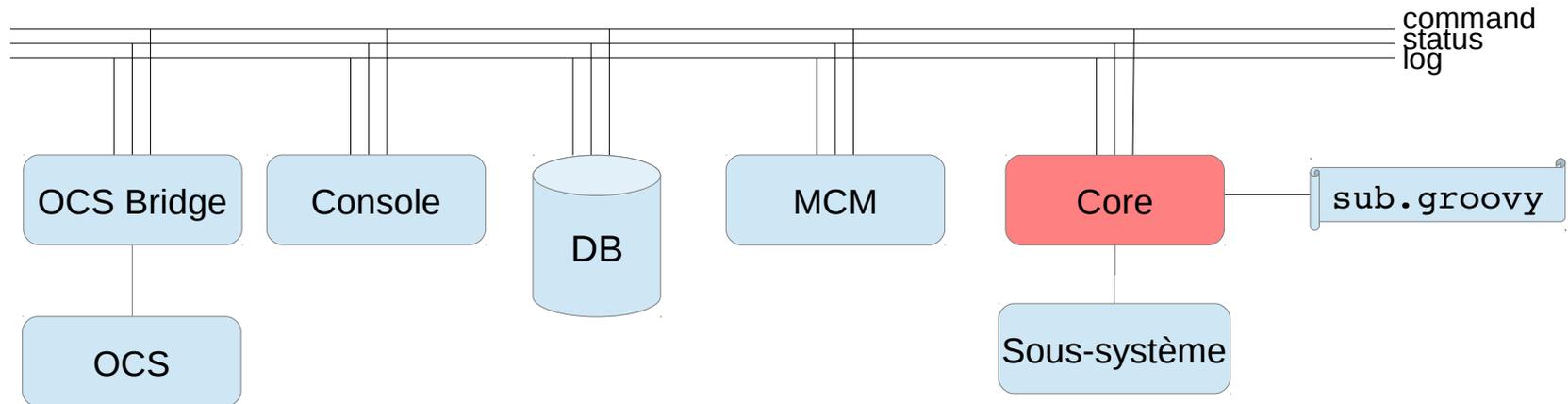
- Contrôler et coordonner les actions des différents sous-systèmes de la caméra. Il est l'interface entre le logiciel de contrôle du télescope (OCS) et les sous-systèmes de la caméra.
- Fournir des outils communs à tous les développeurs de sous systèmes :
 - Bus de communication
 - Configuration et développement de sous-systèmes
 - Bases de données de télémétrie et de configuration
 - Outils de tests (tests unitaires, tests d'intégration, logging)
 - Documentation

- Sous-systèmes modulaires



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

- 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

- Description Groovy : single filter test

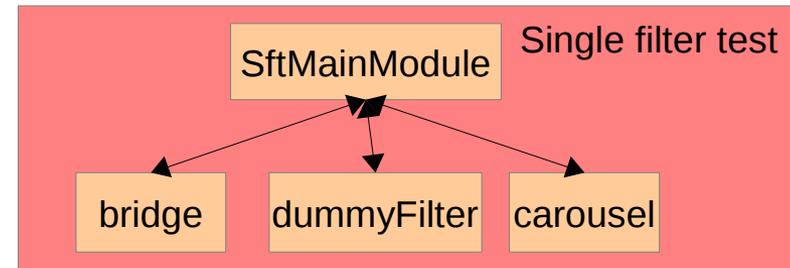


```
CCSBuilder builder = ["single filter test"]
```

```
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"),  
                              aDb("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")})),  
          ))  
  }  
)
```

```
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



Single-filter-test.groovy

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

- 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
```

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 débogage pour le développeur de sous-systèmes et pour l'utilisateur de la caméra
- Utilisation de bibliothèques open source fiables, découplées du reste du logiciel par l'utilisation d'interfaces

Logiciel durable et maintenable pour la durée de vie de LSST