PYROS specifications

VERSION: 27/01/19

1. FONCTIONS

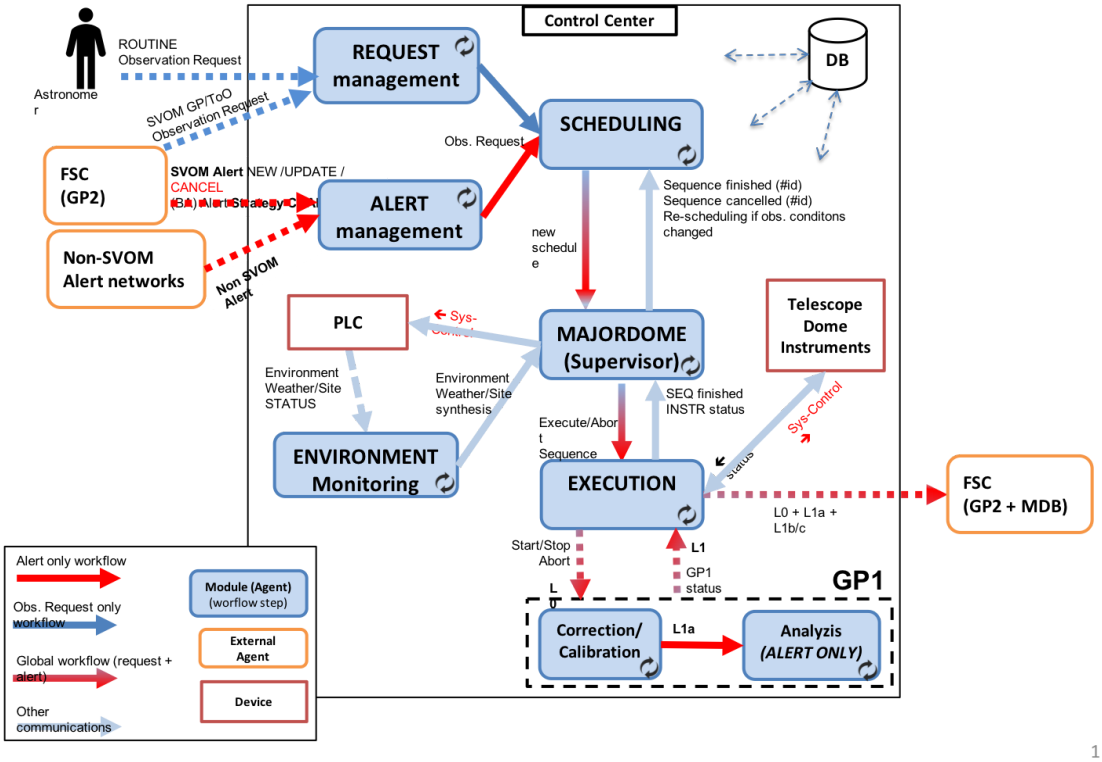
(updated 4/7/18

GFT-REQ-147: The **COLIBRI software** shall manage the following **functions** (see Figure 1):

- 1 - Alert management
- *2* - Observation Request management (*routine management)*
- 3 - Planning
- 4 - Observation execution (cd-ctrl) & Instruments monitoring
- 5 - Environment Monitoring (inside & outside observatory) for human & instruments safety
- **6 - Data reduction & analysis**
- 7 - Information system management (**Dashboard**)
- **8 - Scientific Programs management**
- **9 - Telescope & Instruments long term Monitoring & Calibration**
- **10 - Data archiving** (short and long term)

1.1. MAIN WORKFLOW

Main workflow: **Alerts** and Observation **Requests** management



1

All modules are under the “/src/” directory

	Agent ?	Place	Start
(ENV) Environment Monitor *(P. Maeght)*	YES	src/**monitoring**/tasks.py Monitoring.run()	$ pyros.py start_agent_envmonitor
(SCHED) Scheduler (Planner) *(A. Klotz)*	(NO)	src/**scheduler**/Scheduler.py (and simulator.py) and tasks.scheduling.run()	Appel de la méthode scheduler.tasks.scheduling.run()
(MAJ) Majordome	YES	src/**majordome**/tasks.py Majordome.run()	$ pyros.py start_agent_majordome.py
(ALERT) Alert Manager	YES	src/**alert_manager**/tasks.py AlertListener.run()	$ pyros.py start_agent_alert_manager.py
(REQ) Request Manager	NO		

(EYE) Observer (Executor)	(NO)		
(CAL) Calibrator *(A.K & K. Noysena)*	NO		
(NRTA) NRT Analyzer *(A.K & K. Noysena)*	NO		

1.2. FONCTION 1 - Alert Management (ALERT)

(in src/alert_manager/tasks.py)

⇒ **src/alert_manager/tasks.py/class AlertListener**(Task) :

⇒ **run**() : LOOP, each 1s check if new VOEvent to process and if so process them (parse, then create and save a request) :

⇒ analyze_event(event)
⇒ create_related_request()
⇒ req = create_request_from_strategy()
⇒ req.validate()
⇒ req.save()
⇒ **scheduler.tasks.scheduling.delay(first_schedule=True, alert=self.request.is_alert)**

Détail de la méthode run() :

```
def run(self):
    self.old_files = [f for f in os.listdir(VOEVENTS_PATH) if isfile(join(VOEVENTS_PATH, f))]
    Log.objects.create(agent="Alert manager", message="Start alert manager")

    while True:
        if (settings.DEBUG and DEBUG_FILE):
            log.info("Checking fresh events")
        fresh_events = self.get_fresh_events()
        for event in fresh_events:
            self.analyze_event(event)
            if (settings.DEBUG):
                log.info("Analyzed event : " + str(event))
        time.sleep(1)
```

1.3. FONCTION 2 - Observation Request Management (REQUEST)

(TODO:)

1.4. FONCTION 3 - Planning (PLANNER, SCHEDULER)

(updated 23/04/18)

Responsible : Alain Klotz

Fonctions de ce module :

- Get the list of sequences to be planned
- Plan sequence according to priorities, quotas, observing conditions, and sequence parameters
- Validate and save schedule
- Automatic Schedule update

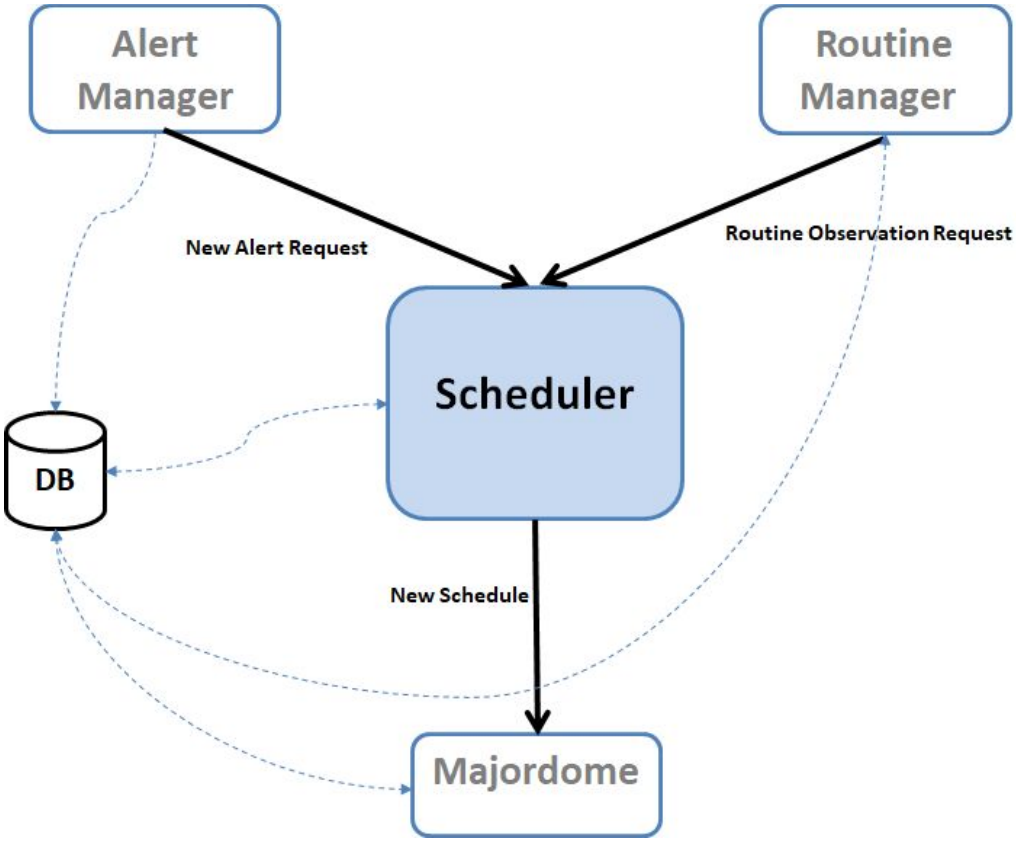
1.4.1. Specs

Heure de référence = heure UTC (GMT)
(AK: pas besoin d'afficher l'heure locale)

Telescope = monture (mount)
On aura un autre canal (camera) qui permettra de mesurer la qualité du ciel (il sera sur la monture)
Unit ("façade") = 1 mount (telescope) + N canaux
Composition : On compose une unit avec : une monture + des canaux
Une "unit" décrit un telescope et son environnement.
Node = N units = Colibri + GWAC-test + GFT chinois + …
= GFTs
Scheduler est au niveau d'une seul unit

Periode : 6 mois
1 SP = 1 quota et souvent 1 observer only
1 nuit = 24h = midi à midi
Ligne de visibilité = par pallier (toujours positif, 0 = visible) : en pointillé
Ligne (noire) de disponibilité du Tele
Observation : BESTELEV ou IMMEDIATE

The Scheduler module interfaces with other modules (inputs on top)



1.4.2. Deux phases principales de test

Phase 1 : tester le module Scheduler **de manière "statique", en "isolation",** c'est à dire seulement la fonction de planification toute seule :

- D'abord, **tester une planification "one shot"** (une seule planification et c'est fini) en vérifiant que les plannings obtenus selon différents lots de Sequences fournis en input (fixtures) sont bien ceux espérés (des séquences "simplistes" seront dans un premier temps créées en dur dans le code puis, dans un deuxième temps, on charger des séquences plus réalistes sour forme de fichiers XML ingérés dans la BD via RequestBuilder) :
    - Input = fixture1 ⇒ output = planning1,
    - Input = fixture2 ⇒ output = planning2,
    - …,
    - Input = fixtureN ⇒ output = planningN,

- Ensuite, **tester plusieurs “re-planifications” consécutives** : d’abord on planifie quelques séquences, puis on en ajoute 1 ou 2 autres, on re-planifie, et ainsi de suite… et vérifier que les plannings obtenus à chaque étape sont conformes à ce qui est attendu

Phase 2 : tester le module Scheduler **de manière “dynamique”, dans le contexte PyROS**, c’est à dire avec des Sequences soumises “au fil de l’eau” par l’utilisateur (User simulator) ou/et par l’agent AlertManager (replanif à chaque fois qu’une nouvelle séquence arrive), qui sont exécutées au fur et à mesure (et donc leur statut change dans le planning, et replanif), avec des “conditions d’observation” qui évoluent (et donc replanif), et enfin des alarmes “météo” et “site” qui viennent perturber le tout (envoyées par la collaboration PLC et Monitoring et lues par le Majordome)..., bref tout un (très gros) programme !
Processus de développement proposé :

- Version 1 : tester les plannings obtenus avec des Sequences soumises “au fil de l’eau” par l’utilisateur (User simulator)
- Version 2 : Version 1 + Sequences alertes soumises par l’agent AlertManager
- Version 3 : Version 2 + changement des conditions d’observation
- Version 4 : Version 3 + alarmes météo ou/et site
- Version 5 : Version 4 + exécution des séquences

1.4.3. Exécution des tests existants

Il existe déjà 14 tests unitaires dédiés au Scheduler actuel.
Ces tests sont dans src/scheduler/tests.py

Pour les éxécuter, activer l’environnement virtuel, puis :

```
(venv) $ cd src/
(venv) $ python manage.py test scheduler.tests

Creating test database for alias 'default'...

===== TEST_3_SEQ_MOVE_BOTH =====

.
===== TEST_3_SEQ_MOVE_LEFT =====

.
===== TEST_3_SEQ_MOVE_RIGHT =====
```

```
.
===== TEST_3_SEQ_PRIORITY =====

.
===== TEST_3_SEQ_PRIORITY_OVERLAP =====

.
...

Duration :  0.0984950065612793
.
----------------------------------------------------------------------
Ran 14 tests in 0.567s

OK
Destroying test database for alias 'default'...
```

1.4.4. Jouer avec le Scheduler (via le django shell)

Activer l'environnement virtuel, puis :

 # Lancer le django shell
(venv) $ cd src/
(venv) $ python manage.py shell

 # Créer une instance du Scheduler
>>> from scheduler.Scheduler import Scheduler
>>> scheduler = Scheduler()
>>> scheduler.max_overhead = 1

 # Créer un utilisateur usr1 (dans la BD)
>>> from common.models import *
>>> c = Country.objects.**create**()
>>> sp = ScientificProgram.objects.**create**()
>>> ul = UserLevel.objects.**create**()
>>> usr1 = PyrosUser.objects.**create**(username="toto", country=c, user_level=ul, quota=100)
>>> usr1
<PyrosUser: toto>

 # Créer une requete req1 (dans la BD)
>>> req1 = Request.objects.**create**(name="my request 1", pyros_user=usr1, scientific_program=sp)
>>> req1
<Request: my request 1>

 # Créer une requete req2 (dans la BD)
>>> req2 = Request.objects.**create**(name="my request 2", pyros_user=usr1, scientific_program=sp)
>>> req2
<Request: my request 2>

 # Créer des sequences pour ces requetes

 # Attention, s'il y a déjà des sequences dans la BD, il vaut mieux les supprimer avant :
sequences = Sequence.objects.all()
for s in sequences: s.delete()

 # Création de 3 nouvelles séquences (dans la BD)

>>> seq11 = Sequence.objects.**create**(request=req1, status=Sequence.TOBEPLANNED, name="seq1.1", jd1=0, jd2=2, priority=1, t_prefered=-1, duration=1)
>>> seq11
<Sequence: seq1.1>

>>> seq12 = Sequence.objects.**create**(request=req1, status=Sequence.TOBEPLANNED, name="seq1.2", jd1=4, jd2=6, priority=1, t_prefered=-1, duration=1)
>>> seq12
<Sequence: seq1.2>

>>> seq13 = Sequence.objects.create(request=req1, status=Sequence.TOBEPLANNED, name="seq1.3", jd1=7, jd2=9, priority=1, t_prefered=-1, duration=1)
*# On aurait aussi pu créer cette nouvelle séquence **par copie** d'une autre :*
>>> import copy
>>> seq13 = copy.copy(seq12)
>>> seq13
<Sequence: seq1.2>
>>> seq13.name = "seq1.3"
>>> seq13.jd1 = 7
>>> seq13.jd2 = 9
>>> seq13

<Sequence: seq1.3>

 # Voyons quelles sequences sont contenues dans la requete req1:
>>> req1.sequences.all()
<QuerySet [<Sequence: seq11>, <Sequence: seq1.2>, <Sequence: seq1.1>]>

 # On fait une planif
>>> scheduler.makeSchedule()
<Schedule: 2018-03-01 15:26:06.139674+00:00>

# On récupère le dernier planning (c'est à dire celui qu'on vient de créer) :
>>> schedule = Schedule.objects.order_by('-created').first()
>>> schedule
<Schedule: 2018-03-01 15:26:06.139674+00:00>

# Quelles sont les séquences associées à ce planning (combien) ? :
>>> shs_list = sched.shs.all()
>>> nbPlanned = len([shs for shs in shs_list])
>>> nbPlanned
3

1.4.5. Procédure concrète de soumission des requetes pour les tests :

1 - Créer des fichiers requetes XML dans simulators/resources/, tels que par exemple routine_request_01.xml :

```
<?xml version="1.0" ?>
<request submitted="1" relative="1" name="RequestSimulator" scientific_program="GRB" target_type="test">
        <sequence duration="10" jd1="15" jd2="60000" name="Sequence1 (200secs)" target_coords="10">
        <album detector="Visible camera" name="alb">
        <plan duration="10" filter="First infrared filter" name="simulation" nb_images="5"/>
        </album>
        </sequence>
</request>
```

2 - Appeler RequestBuilder pour créer un objet Request à partir de ce fichier XML, et le mettre dans la BD

3 - Soumettre ces requetes au Scheduler qui doit les planifier...

1.5. FONCTION 4 - Observation EXECUTION & Instruments Monitoring (EXEC, system control)

(updated 23/04/18)

Responsible : Quentin Durand

Includes : MAJORDOME and OBSERVER submodules (+ controleurs & simulateurs Tele et instrum)

Mode manuel des TAROT de AK (pour ref) :
http://cador.obs-hp.fr/ros/manual/cador_actions.html

TODO:
Ajouter :
- status meteo
- status jour/nuit
- status infra : toit ouvert…

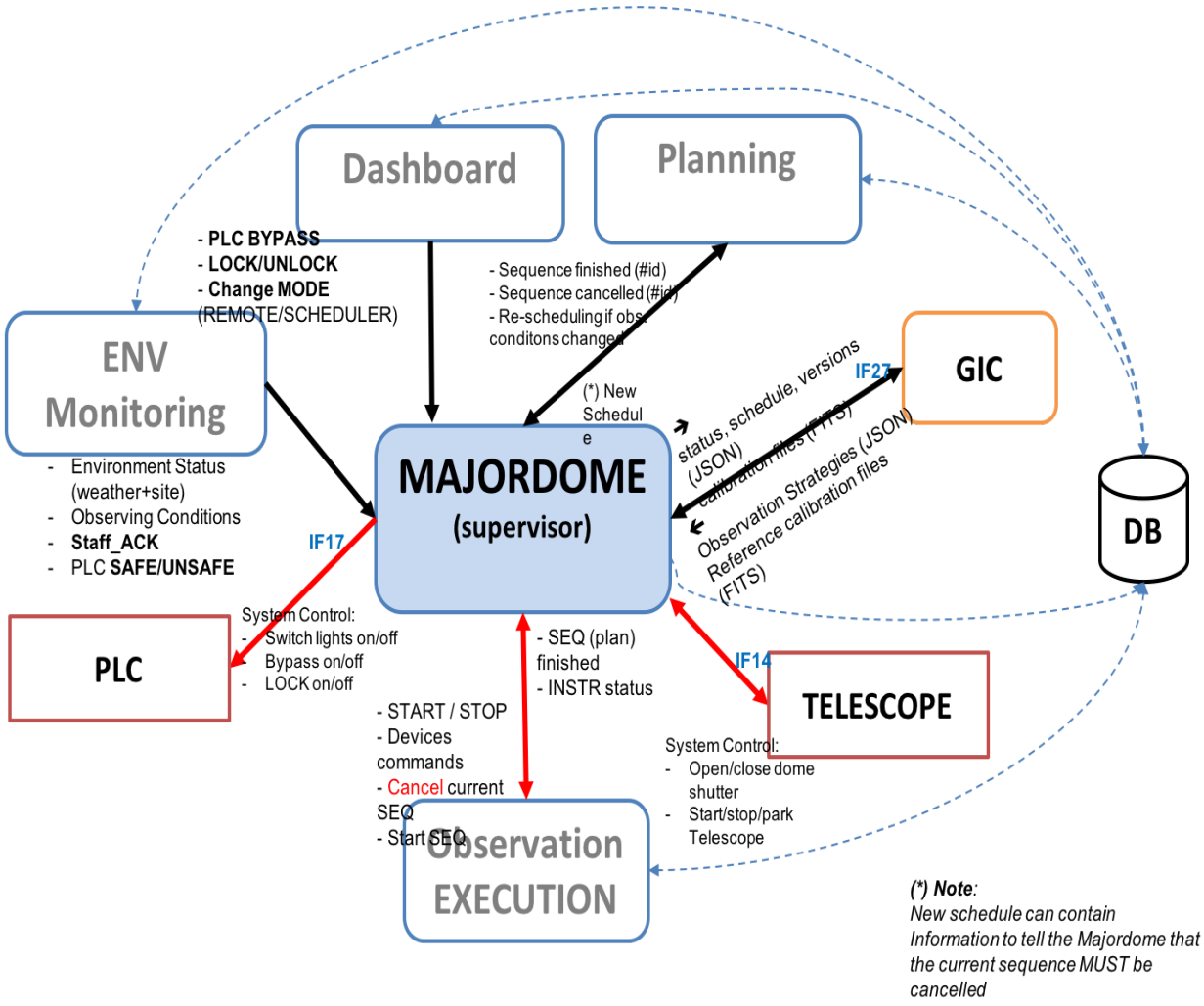
Fonctions de ce module :

- Check observation conditions
- System control of the telescope & instruments (manuel and auto modes)
- Monitor the telescope & instruments status
- Execute planned observation sequences
- Stop current sequence and run priority sequence instead
- Get raw images from instruments
- Add useful header to images

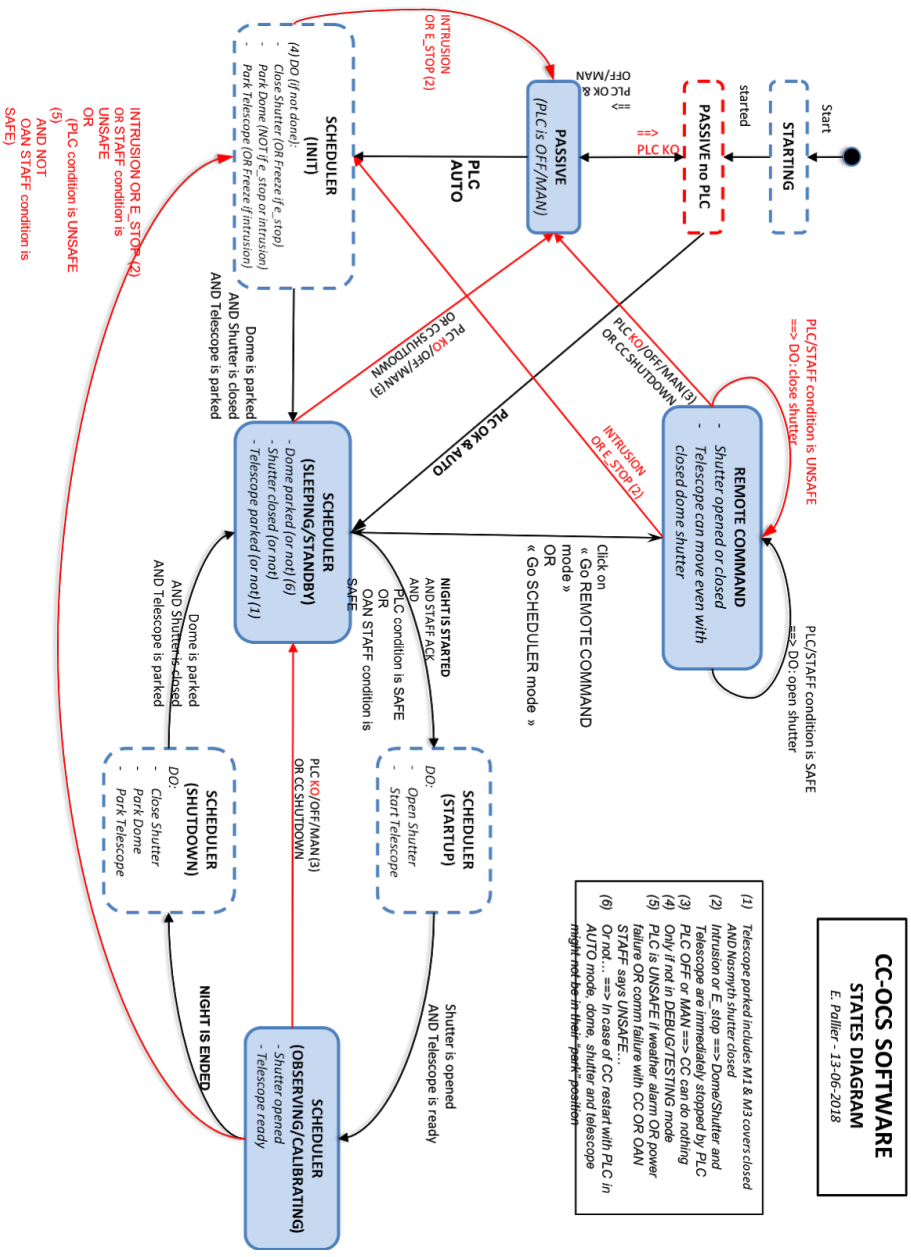
1.5.1. MODULE “Majordome (Conductor, Master)”

(updated 25/7/18, EP)

1.5.1.1. Context diagram



1.5.1.2. States diagram



1.5.1.3. DEVICES STATUS LIST to be sent to GIC

- **Telescope**: environ tous les 6 sec
  - Slewing 0/1
  - Homing 0/1
  - Park 0/1
  - Connected 0/1
  - Stop sensor 0/1 (est-il en fin de course ?)
  - Last connexion establishment (date de dernier début de connexion) : date UTC
  - Position du flat 0 1 (est-il sur la position de l'écran de flat ?)
  - 3 types de coordonnées de position du tele :
  - ra/dec
  - ha/dec
  - alt/az (suivre l'enchaînement des positions dans la nuit)
  - Date de last maintenance

- **DDRAGO & CAGIRE** :
  - Roue à filtre : homing + slewing
  - Position (sur quel filtre)
  - CCD temperature
  - Initialisation de la camera : cam init 0/1
  - Cam pending 0/1 : elle attend ou bien elle fait une pose (contraire du idle)
  - Idle 0/1
  - Date de last maintenance
  - Failure 0/1

- **Dome & shutter**
  - Homing
  - slewing
  - Position (degrés)
  - Shutter : opened/closed/intermediate/unknown

- **PLC** (weather status, observatory status) :
  - valeur de chaque capteur (sensor) : rain.sensor1 rain.sensor2 wind.sensor3
  - Rain
  - Good weather
  - Wind: Clear, Cloudy, very cloudy

1.5.1.4. RUN

1 - Agent Majordome seul

(venv) $ cd src/majordome/
(venv) $ ./start_agent_majordome.py

Le Majordome devrait afficher les messages suivants:
CURRENT OCS (MAJORDOME) STATE: STARTING
CURRENT OCS (MAJORDOME) STATE: PASSIVE_NO_PLC
Waiting for PLC connection...
Le majordome doit rester en attente de la connexion du PLC…

2 - Agent Majordome et les autres éléments nécessaires

Aller à la racine du projet
(venv) $ ./pyros.py start_agents_and_simulators_for_majordome

Ce script lance l'agent Majordome, l'agent Env-Monitoring et le simulateur de PLC

Le Majordome devrait afficher les messages suivants:
CURRENT OCS (MAJORDOME) STATE: STARTING
CURRENT OCS (MAJORDOME) STATE: PASSIVE_NO_PLC
Waiting for PLC connection...
Puis, il devrait passer à l'état PASSIVE (car le simulateur de PLC est lancé)...

NB: Vous pouvez aussi lancer le serveur web (./pyros.py server) pour voir ce qui se passe, surtout via la page web System

1.5.1.5. TEST

A - Test automatique:

(venv) $ cd src/majordome/
(venv) $ ./majordome_test.py

Ce test modifie certaines variables dans la BD pyros_test (notamment dans les tables config et plcdevicestatus) pour forcer le majordome à passer dans différents états

B - Test manuel (de visu):

1 - Dans un terminal, démarrer le serveur web pour voir le site pyros

Aller à la racine du projet et lancer le serveur web:
(venv) $ ./pyros.py server
Cliquer sur le menu “System” à gauche, pour aller sur la page de monitoring des agents.
Cette page vous permettra de suivre l’évolution des agents “majordome” et “environment-monitoring”
(http://localhost:8000/dashboard/system)

2 - Dans un autre terminal, démarrer tous les agents et simulateurs nécessaires pour que le majordome fonctionne

a - Avec le script ad hoc, c’est plus facile (il lance tout pour vous)
(venv) $ ./pyros.py start_agents_and_simulators_for_majordome
(ce script démarre les agents majordome et environment-monitoring, ainsi que le simulateur de plc)
Revenez sur la page web pour suivre l’évolution des agents “majordome” et “environment-monitoring”

Vous pouvez maintenant passer à l’étape 3 ci-dessous.

b - A la mano (vous démarrez vous-même chaque agent et simulateur, un par un)
Si vous n’aimez pas le script ad hoc ci-dessus et que vous préférez avoir un contrôle total de chaque élément, alors lancez vous même chaque agent et simulateur un par un:
Dans un nouveau terminal, démarrer l’agent majordome:
(venv) $ cd src/majordome/
(venv) $ ./start_agent_majordome.py
Il devrait afficher les messages suivants:
CURRENT OCS (MAJORDOME) STATE: STARTING
CURRENT OCS (MAJORDOME) STATE: PASSIVE_NO_PLC
Waiting for PLC connection...

Le majordome attend la connexion du PLC

Démarrer le simulateur de PLC et l’agent env-monitoring qui remplit la BD à partir des données du PLC, ce qui permet au majordome de savoir que le PLC est vivant…

- Démarrage simulateur PLC (dans un nouveau terminal) :
  - (venv) $ cd simulators/plc/
  - (venv) $ ./plcSimulator.py
- Démarrage agent env monitoring (dans un nouveau terminal) :
  - (venv) $ cd src/monitoring/
  - (venv) $ ./start_agent_monitoring.py

Le majordome voit la connexion avec le PLC et passe donc de l’état PASSIVE_NO_PLC à “PASSIVE” puis “Standby” (en passant par “closing”)
Si on arrête (CTRL-C) le simulateur de PLC (et le env monitoring), le Majordome passe alors de nouveau à l’état PASSIVE puis PASSIVE_NO_PLC

3 - Suivez l'évolution du Majordome (ses différents "états")

Le majordome voit la connexion avec le PLC et passe donc de l'état PASSIVE_NO_PLC à "PASSIVE" puis "Standby" (en passant par "closing")

Maintenant, pour modifier l'état du Majordome, aller sur le site web, dans Préférences, puis Simulator

TODO: Activer certains boutons du Simulator pour influer sur l'état du Majordome
Notamment, il faudrait un bouton qui permette de dire que la communication avec le PLC est OK ou KO pour que le Majordome passe à l'état "PASSIVE" (resp. "PASSIVE no PLC")

1.5.1.6. General algorithm

in src/majordome/tasks.py

C'est lui qui lance les agents Monitoring et Alert Manager (cf majordome/tasks.py/Majordome (class)/handleTasks()) car il voit qu'ils n'existent pas encore (ensuite, il les check régulièrement pour les relancer si arrêtés) *

⇒ **src/majordome/tasks.py/class Majordome**(Task) :

⇒ **run**() :

createTask() ⇒ TaskId.objects.create(task_id=self.request.id,
task="majordome")
updateSoftware()
setContext() :
tel = TelescopeController()
vis_camera = VISCameraController()
nir_camera = NIRCameraController()
plc = PLCController()
dom = DomeController()
setTime() : # set timers and handlers (one handler per timer)
setTasks():
monitoring_task = TaskId.objects.**get**(task="monitoring")
alert_task = TaskId.objects.**get**(task="alert_manager")
loop() ⇒ LOOP (agent) :

- check devices status
- check if sequence is finished
- check environment (from DB) (and take action, re-schedule)
- check if **new schedule** available :
  - executeSchedule()
    - executeSequence() :
      - observation_manager.tasks.**execute_plan_nir**()

- observation_manager.tasks.**execute_plan_vis**()
- check if start of night ⇒ if so, launch a scheduling()
- check if end of night
- * check Monitoring and AlertManager (handleTasks()) :
  - monitoring.tasks.Monitoring.apply_async()
  - alert_manager.tasks.AlertListener.apply_async()

Détail de la méthode run() :

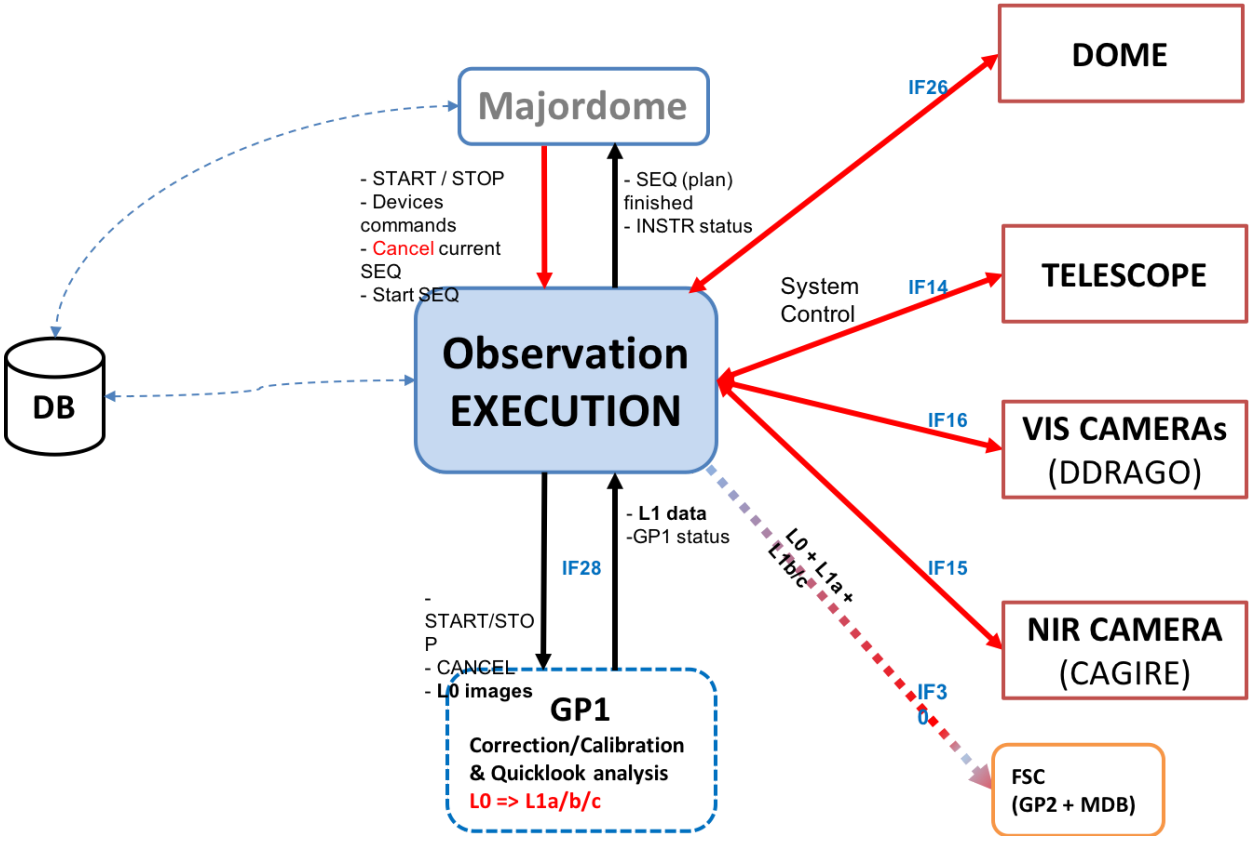
```
def run(self):
        self.createTask()
        self.updateSoftware()
        self.setContext()
        self.setTime()
        self.setTasks()
        self.loop()

def loop(self):
        while (self.current_status != "SHUTDOWN"):
                minimal_timer = min(self.timers, key=self.timers.get)
                if (self.timers[minimal_timer] > 0):
                        time.sleep(self.timers[minimal_timer])
                        self.timers = {key: value - self.timers[minimal_timer] for key, value in self.timers.items()}
                for timer_name, timer_value in self.timers.items():
                        if (timer_value <= 0):
                                if timer_name in self.functions:
                                        self.logDB("Executing timer " + str(timer_name))
                                        self.functions[timer_name]()
                                else:
                                        if (settings.DEBUG and DEBUG_FILE):
                                                log.info("Timer : " + str(timer_name) + "is not known by the Majordome")
                                                        self.logDB("Timer " + str(timer_name) + " unknown
                                if (settings.DEBUG and DEBUG_FILE):
                                        log.info("Timer : " + str(timer_name) + " executed")
```

1.5.2. MODULE “Observer (EXEC)”

(updated 4/7/18)

1.5.2.1. Context diagram



1.5.2.2. Telescope monitoring agent

The telescope monitoring agent currently implemented is only a prototype.
Its current functionalities are the following:

When launched, the agent watch the DB looking for requests TelescopeCommand created by the server when a command is submitted on the web (remote mode)

If requests are found, it executes them using the TelescopeController.send_command() Method of the TelescopeController instantiated in the agent.
For now **IT DOES NOT USE** the RemoteControl classes which translate the generic commands into specific ones

When the requests are executed, the agent fill the answer field of the request in the DB

The requests are created in the views submit_command_to_telescope* (views.py)
And for the expert_mode, the view sleep for some milliseconds before sending an answer to the web to let the time to the agent to execute the request and fill up the db with the answer, anwer which is sent by the view to the client as a response

1.6. Javascript files -> misc/static/js

1.7. Navbar -> misc/templates/base.html or base_unlogged.html

1.8. FONCTION 5 - ENVIRONMENT monitoring (ENV)

(updated 22/6/18 - EP)

Responsible : Patrick Maeght

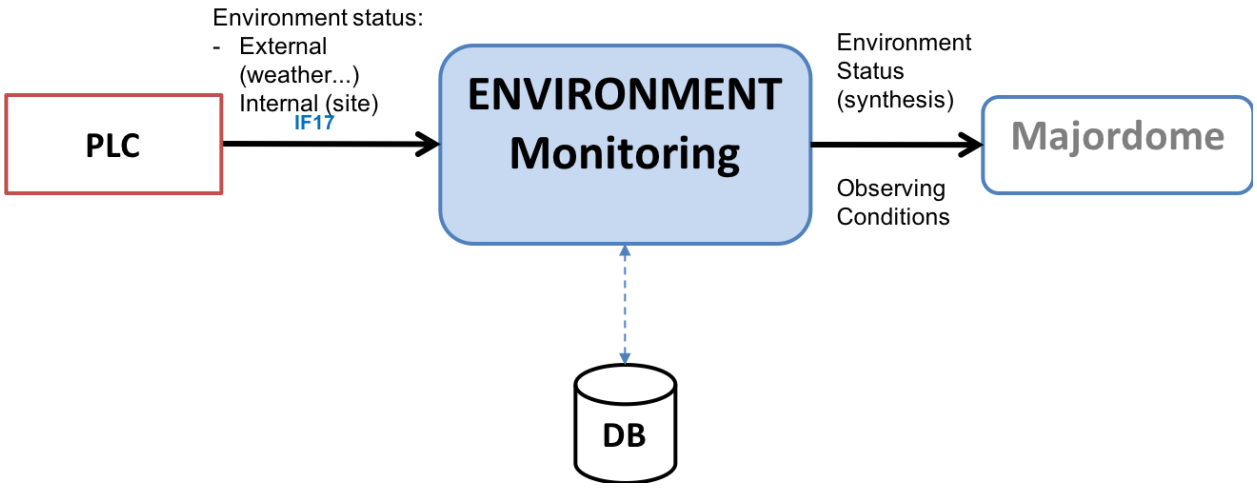
GFT-REQ-290: Environment monitoring shall take in charge the **following actions**:

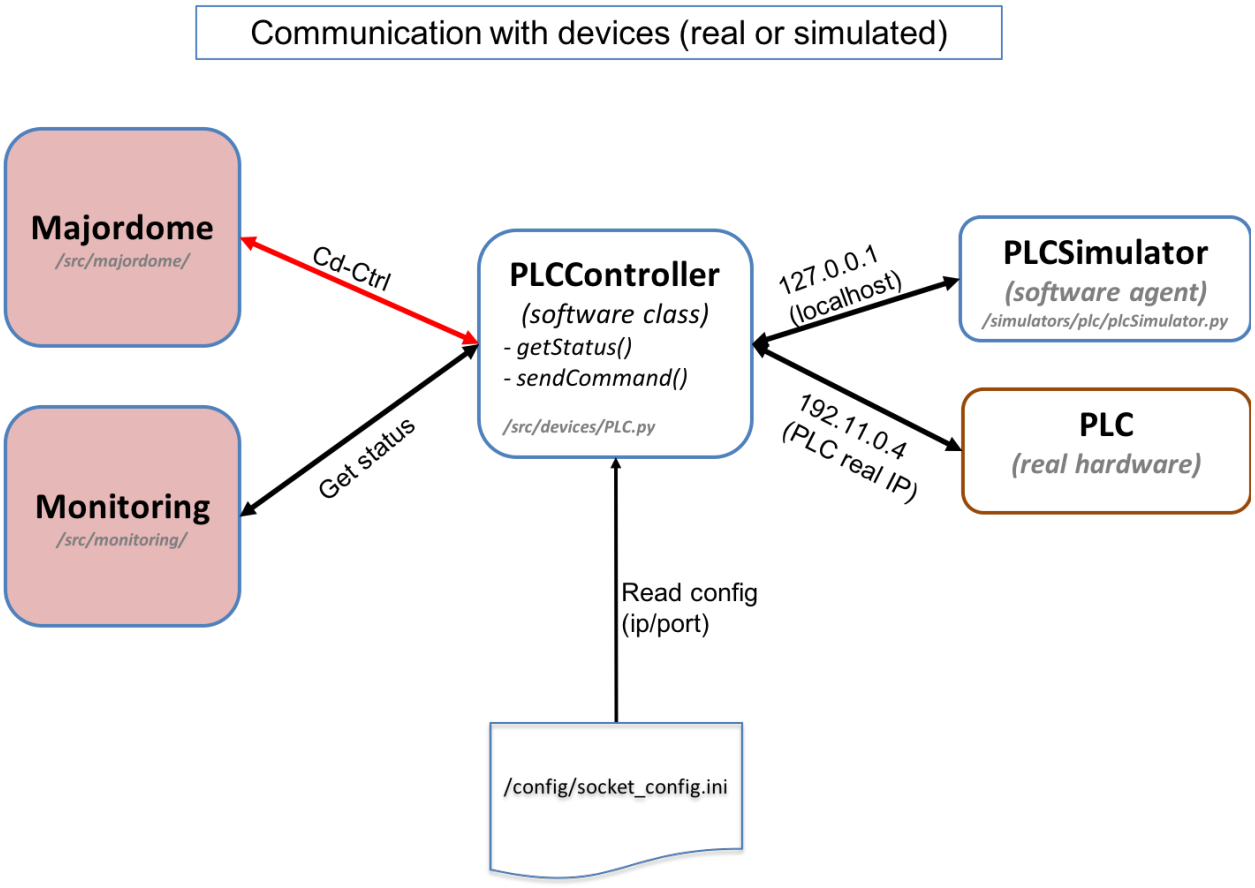
- Read outside environmental data (weather..., from the PLC) for instruments security
- Read inside environmental data (doors, lights..., from the PLC) for human safety
- Get PLC mode changes (off/manu/auto) and alarms (intrusion, e_stop)
- Correct raw data
- Compute and provide higher level (useful) parameters and synthesis from multiple detectors
- Save monitored data
- Keep a history of monitoring data
- Manage Observing conditions
- Show Weather & Observatory monitored data (in a convenient way)

Chaque capteur du PLC devra donner son time stamp
Redonder et prioriser les capteurs d'un même type de données (ex: pour l'humidité, on peut avoir 3 capteurs différents, dont un qui est le principal)

1.8.1. Contexte

The Environment Monitor module interfaces with other modules (inputs on the left)





1.8.2. PLC

1.8.2.1. Power management (onduleurs)

(FD):
Le PLC gère plusieurs niveaux de "warnings" pour les onduleurs :
- pas de pb
- passage sur batterie
- batterie faible, mise en protection.

Ensuite, en fonction de comment il communiquera avec l'onduleur, il sera possible d'avoir d'autres diagnostiques

A priori, on ne gère pas le cas "tension plus faible qu'un seuil" (à définir, par ex: 107V au lieu de 110V nominal), cas qui n'est pas critique car l'onduleur est toujours en charge...

6.8.3. Fonctionnement du module

Principe général de fonctionnement :
⇒ A chaque itération, l'Agent Monitoring envoie une commande de status au PLC
⇒ Le PLC lui retourne son état actuel (status)
=> Le Monitoring en fait une synthèse qu'il met dans la BD

Fonctionnement détaillé :
(in src/monitoring/tasks.py)
C'est lui qui lance les agents Majordome et Alert Manager (cf monitoring/tasks.py/Monitoring (class)/handleTasks()) car il voit qu'ils n'existent pas encore (ensuite, il les check régulièrement pour les relancer si arrêtés) *

⇒ **src/monitoring/tasks.py/class Monitoring**(Task) :
⇒ **run**() :
createTask() ⇒ TaskId.objects.**create**(task_id=*self*.request.id,
task=*"**monitoring**"*)
setContext() ⇒ plc = PLCController()
setTime()
setTasks() :
majordome_task = TaskId.objects.**get**(task=*"majordome"*)
alert_task = TaskId.objects.**get**(task=*"alert_manager"*)
loop() ⇒ LOOP (agent) :
Get PLC status :
handleTimerStatus() : status_plc = self.plc.getStatus() + SAVE in DB
* Check if the majordome and alert_manager are running (otherwise, relaunch) :
handleTasks() :
- majordome.tasks.Majordome.apply_async()
- alert_manager.tasks.AlertListener.apply_async()

Détail de la méthode run() :

```
def run(self):
    self.createTask()
    self.setContext()
    self.setTime()
    self.setTasks()
```

```
        self.loop()
```

Détail de la BOUCLE :

```
def loop(self):
        while (self.state != "SHUTDOWN"):
                minimal_timer = min(self.timers, key=self.timers.get)
                time.sleep(self.timers[minimal_timer])
                self.timers = {key: value - self.timers[minimal_timer] for key, value in self.timers.items()}
                for timer_name, timer_value in self.timers.items():
                        if (timer_value <= 0):
                                if (timer_name in self.functions):
                                        self.functions[timer_name]()
                                else:
                                        if (settings.DEBUG and DEBUG_FILE):
                                                log.info("Timer : " + str(timer_name) + "is not known by the monitoring")
                                        self.logDB("Timer " + str(timer_name) + " unknown")
                        if (settings.DEBUG and DEBUG_FILE):
                                log.info("Timer : " + str(timer_name) + " executed by monitoring")
```

1.8.4. EXECUTION

(updated 22/6/18 - EP)

Dans la phase de dev du module Monitoring (agent), **inutile de s'encombrer de Celery** pour tester ce module en isolation, donc **pas besoin non plus de démarrer RabbitMQ**.
On peut donc désactiver Celery ; dans 2 fichiers (pyros.py et src/pyros/settings.py), il faut s'assurer d'avoir ceci:

```
        USE_CELERY = False
```

⇒ Pour la version avec Celery, voir la section "EXECUTION AVEC CELERY" ci-dessous. Elle donne beaucoup plus de détail sur le déroulement de l'exécution.

L'exécution se fait à partir de 2 ou 3 terminaux :

(1) - (Agent) Terminal 1 - Le serveur web (OPTIONNEL)

Optionnellement, on peut lancer le serveur web dans un 1er terminal, afin de mieux voir ce qui se passe au niveau de la météo et de l'observatoire :

```
(venv) $ ./pyros.py start_web

Ou encore :
(venv) $ cd src/
(venv) $ ./manage.py runserver

Starting development server at http://127.0.0.1:8000/
(keep it running...)
```

Puis se connecter sur http://localhost:8000
For your information, general syntax is :
$ ./manage.py runserver IP:PORT
Example: $ ./manage.py runserver localhost:8001
(obsolète: To check that this service is actually running, type "$ netstat -an |grep 8000" and you should get "tcp 0 0 127.0.0.1:8000 0.0.0.0: LISTEN")*

On peut maintenant cliquer sur l'icone WEATHER pour voir les infos météo données par le PLC.
On peut aussi cliquer sur l'icone OBSERVATORY pour voir les infos observatoire données par le PLC.
Bien sûr, pour l'instant ces infos ne changent pas puisqu'on n'a pas lancé l'agent Env monitoring, ni le PLC.

(2) - (Agent) Terminal 2 - L'Agent "Environment Monitoring" (ENV)

Voici comment faire pour démarrer cet agent (depuis l'environnement virtuel), dans un 2ème terminal :

```
(venv) $ ./pyros.py start_agent_monitoring
(Windows: python pyros.py start_agent_monitoring)

Ou encore :
(venv) $ cd src/monitoring/
(venv) $ ./start_agent_monitoring
(Windows: python start_agent_monitoring)
```

Ou bien encore, depuis le django shell :

```
(venv) $ cd src/
(venv) $ python manage.py shell
>>> from monitoring.tasks import Monitoring
>>> Monitoring().run()
>>> ...
```

```
Ou encore :
>>> m = Monitoring()
>>> m.run()
>>> ...
```

Voilà, l'agent Env monitoring est lancé, il est en attente d'informations du PLC, mais il n'y a toujours pas de PLC, donc pas d'infos…

(3) - (Agent) Terminal 3 - Le simulateur de PLC

Dans un 3ème terminal (T3), activer l'environnement virtuel, puis :

```
(venv) $ cd simulators/plc/
(venv) $ ./plcSimulator.py scenario_plc.json
(Windows: python plcSimulator.py scenario_plc.json)
```

(pour info, scenario_plc.json est lu dans simulators/config/)
On peut aussi lancer le simulateur sans lui passer de scenario :

```
(venv) $ ./plcSimulator.py
```

Sans scénario, le simulateur du PLC donnera toujours la même réponse à celui qui l'interroge (l'agent Environment Monitoring). Avec un scénario, la réponse pourra varier.

Ca y est enfin, tout est prêt.
Si on a lancé le serveur web (étape 1 optionnelle), on peut maintenant cliquer sur l'icone WEATHER du dashboard pour voir évoluer les infos météo données par le PLC.
On peut aussi cliquer sur l'icone OBSERVATORY pour voir évoluer les infos observatoire données par le PLC.

6.8.4.1. Execution AVEC CELERY (version complète)

6.8.4.2. - (Agent) Terminal 0 - Un worker Celery dédié au Monitoring

Il attend des tâches à exécuter pour le Monitoring

A lancer dans un premier terminal (qu'on appellera **T0)**
⇒ E**n fait, ce worker recevra seulement une tâche à exécuter : le run() de src/monitoring/tasks.py**
⇒ Voici comment faire :
Activer l'environnement virtuel, puis :

```
(venv) $ cd src/monitoring/
(venv) $ ./start_celery_worker.py
```

NB1: cela est équivalent à exécuter cette commande :

```
$ celery worker -A pyros -Q monitoring_q -n pyros@monitoring -c 1
```

NB2: pour les curieux, voici le chemin parcouru par cet appel à celery :

site-packages/celery/worker/__init__.py
site-packages/celery/bootsteps.py
site-packages/celery/worker/consumer.py
site-packages/celery/worker/loops.py
site-packages/celery/apps/worker.py
site-packages/celery/utils/dispatch/signal.py
src/pyros/__init__.py

⇒ Cela doit afficher le message suivant qui dit que le worker est en attente de tâche :
[2018-02-19 11:07:04,023: WARNING/MainProcess] ***pyros@monitoring ready***
⇒ Sinon, si le message affiché est une erreur (en rouge), cela signifie que RabbitMQ n'est pas démarré :
*[2018-02-19 11:26:11,956: ERROR/MainProcess] consumer: Cannot connect to amqp://guest:**@127.0.0.1:5672//: [Errno 61] Connection refused.*
Trying again in 2.00 seconds…

Pour l'instant, **rien ne se passe, le worker est seulement en attente** de tâche à exécuter. Cette instruction a seulement créé une **file d'attente** nommée **monitoring_q** et **un worker** qui lit cette "queue" en attendant quelque chose à faire, mais pour le moment il se tourne les pouces…
NB:

- *Pour stopper ce worker, taper CTRL-C*

- *Si après avoir stoppé puis relancé ce worker, vous recevez toujours des messages de l'agent Monitoring (qui n'a donc pas été "tué" proprement), voici comment arrêter définitivement cette tâche :*
  `(venv) $ ./stop_celery_worker.py`

6.8.4.3. - (Agent) Terminal 1 - L'Agent "Monitoring"

(dans **src/monitoring**/, le fichier **tasks.py**, et plus précisément, sa méthode **run**())

OK. On a un worker dédié au monitoring qui ne fait rien pour l'instant, à part attendre qu'on lui donne du boulot. Et bien, on va lui en donner du boulot ! On va lui donner 1 tâche à exécuter, qui sera notre agent Monitoring (ou ENV).

A lancer dans un deuxième terminal (qu'on appellera **T1).**
Voici comment faire pour démarrer cet agent dans le worker celery :
Dans un 2ème terminal donc (autre que celui qui exécute le worker ci-dessus), activer l'environnement virtuel, puis lancer le Django shell :

```
(venv) $ cd src/
(venv) $ python manage.py shell
>>> import monitoring.tasks
>>> task_id = monitoring.tasks.Monitoring.dispatch()
>>> task_id
<AsyncResult: f225350c-e6c4-49b7-af26-d6b99fe9c596>
```

La tâche est lancée et on peut voir sur le 1er terminal (T0) les messages affichés par la tâche monitoring en cours (en fait, l'agent Monitoring, en boucle infinie) :

```
AGENT Monitoring: startup...
FAILED TO CONNECT TO DEVICE PLC
AGENT Monitoring: config PLC is (ip=127.0.0.1, port=5003)
AGENT Monitoring: my timers (check env status every 2s, check other agents
every 5s)
AGENT Monitoring: Other Agents id read from DB (majordome=None, alert=None)

AGENT Monitoring (ENV): iteration 0, (my state is RUNNING) :
FAILED TO CONNECT TO DEVICE PLC
Invalid PLC status returned (while reading) : NOT_SET
Timer : timer_status executed by monitoring

AGENT Monitoring (ENV): iteration 1, (my state is RUNNING) :
```

```
FAILED TO CONNECT TO DEVICE PLC
Invalid PLC status returned (while reading) : NOT_SET
Timer : timer_status executed by monitoring

AGENT Monitoring (ENV): iteration 2, (my state is RUNNING) :
TaskId matching query does not exist.
TaskId matching query does not exist.
Timer : timer_tasks executed by monitoring

AGENT Monitoring (ENV): iteration 3, (my state is RUNNING) :
...
```

L'id de la tâche Monitoring (task_id) est sauvegardé dans la table "task_id" (par la fonction createTask() de monitoring.tasks.run()).
Si vous avez un **phpmyadmin** installé*, vous pourrez voir une ligne de la table (base de données pyros) comme celle-ci :

id	task	created	task_id
11	monitoring	2018-02-19 14:52:15.640867	f225350c-e6c4-49b7-af26-d6b99fe9c596

**NB: Si vous n'avez pas de phpmyadmin, vous pouvez aussi utiliser la page administration de pyros : voir pour cela la section "Accéder à la page d'administration de PyROS"*

On peut constater dans les messages de celery (ci-dessus), la ligne suivante :
FAILED TO CONNECT TO DEVICE **PLC**
⇒ C'est normal, car il n'y a pas de PLC pour l'instant !!!
⇒ Notre agent Monitoring passe son temps à interroger (à chaque itération) un device (le PLC) qui n'existe pas !!!
⇒ Ca n'est pas bloquant, le Monitoring continue de faire sa boucle infinie
⇒ Il va donc falloir lancer un simulateur de PLC à un moment ou un autre… (voir étape 3 ci-dessous).

L'agent Monitoring contient un pointeur vers le **contrôleur du PLC** (qui s'appelle plcController)
La config du PLC (adresse IP et port) a été lue dans le fichier /**config/socket_config.ini**

Voici le contenu de ce fichier :

[Telescope]
ip=127.0.0.1
port=5000

[CameraVIS]
ip=127.0.0.1
port=5001

[CameraNIR]
ip=127.0.0.1
port=5002

[**PLC**]
ip=127.0.0.1
port=5003

[Dome]
ip=127.0.0.1
port=5004

On y voit que le PLC (en l'occurrence, le simulateur de PLC, voir étape suivante) écoute sur l'adresse localhost (127.0.0.1) et sur le port 5003

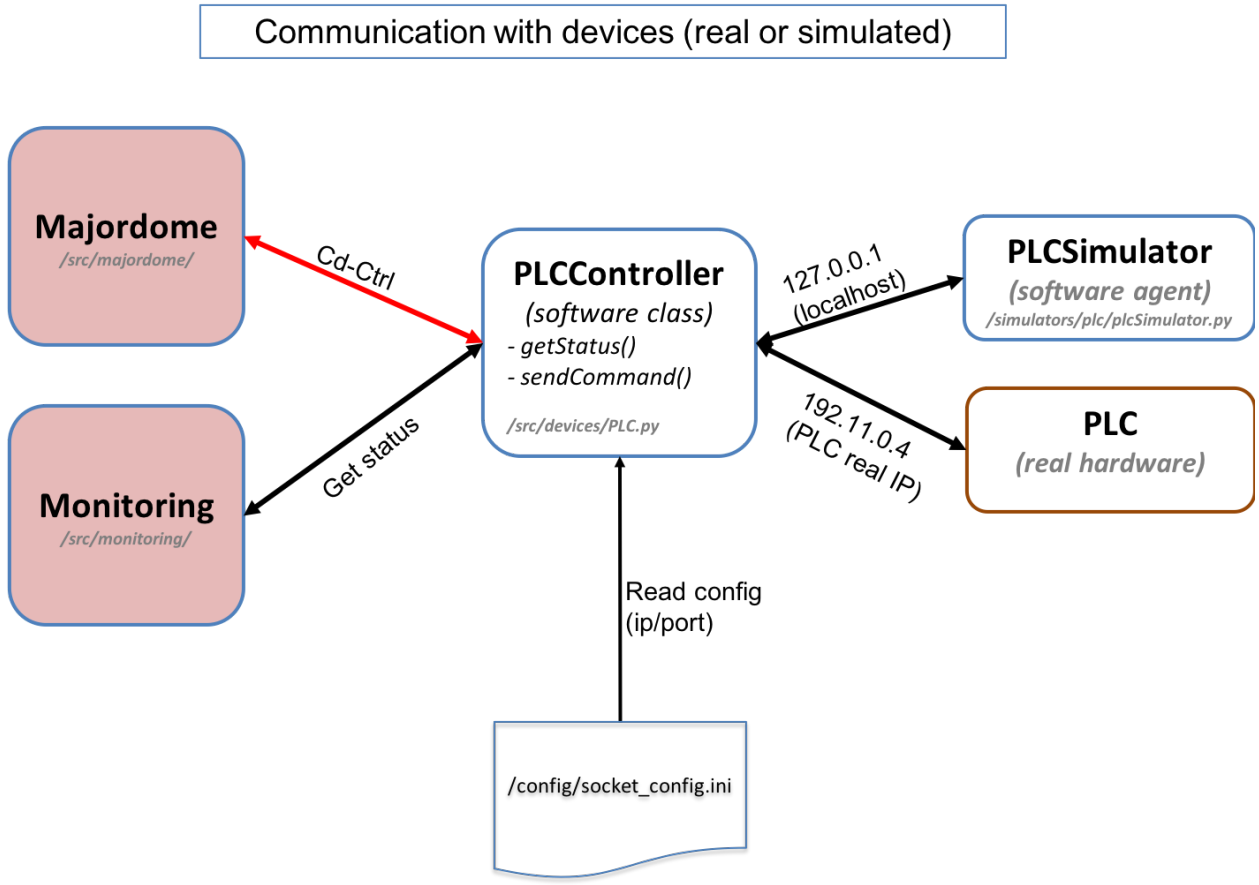
Le **contrôleur du PLC** est un **intermédiaire entre l'agent monitoring et le PLC**, qui permet de **dialoguer avec le PLC** (envoi de commande, et réception de la réponse).
Cette classe **PLCController** est définie dans **src/devices/PLC.py** (TODO: le dossier devices devrait plutot s'appeler device_controllers, on fera le changement un jour…)
⇒ elle hérite de la classe **DeviceController** (définie dans src/devices/Device.py ; TODO: ce fichier devrait plutôt s'appeler DeviceController.py, on fera ça aussi un jour...)
⇒ elle est utilisée directement par l'agent Monitoring pour envoyer des commandes au PLC (**ce n'est pas un agent, c'est juste une classe**) et récupérer le résultat (listes de status)

NB: On peut aussi regarder le contenu du **fichier LOG** (de Monitoring) qui contient quelques infos utiles sur ce qui se passe... : **/logs/Monitoring.log**

6.8.4.4. - (Agent) Terminal 2 - Le simulateur de PLC

Bon, récapitulons : on a un agent Monitoring qui tourne en boucle (lancé dans T1), et un worker celery dédié qui exécute cet agent (lancé dans T0)…
Mais il nous manque un PLC avec qui taper la causette ! En attendant le vrai PLC, on va utiliser un “simulateur” (très simplifié) dont la fonction sera seulement de répondre à nos questions.

Ce simulateur (PLCSimulator) **remplace donc le futur vrai PLC hardware, et sera donc capable de répondre à une requête envoyée par Monitoring (via un dictionnaire json**).



Il est défini dans simulators/plc/**plcSimulator**.**py**

Attention, le dossier “simulators/” est à la racine du projet et donc en dehors de Django (qui est dans src/). C’est du python pur. Rien à voir avec django ou celery.

Le fichier **plcSimulator.py** définit la classe **PLCSimulator.**
Elle hérite des 2 classes simulators/utils/device.py/**DeviceSim** et simulators/utils/StatusManager.py/**StatusManager** :

- la **super-classe DeviceSim** sert à définir le comportement général d’un simulateur (comportement surchargé/adapté par les méthodes de PLCSimulator).

- la **super-classe StatusManager** sert à définir le comportement général d'un **générateur d'événements** à partir de la lecture d'un **fichier scénario** (json) qui lui dit quels sont les événements à générer et quand. Par exemple, le simulateur de PLC (PLCSimulator) pourra lire un fichier de scénario scenario_plc.json qui lui dit de "faire croire" qu'il pleut à partir de sa 3ème itération de boucle, et qu'il ne pleut plus à partir de sa 7ème itération, ou bien encore de "faire croire" qu'il est en panne (TODO:) pendant N itérations…

Il devra écouter sur l'adresse localhost (127.0.0.1) et sur le port 5003. Cette configuration du PLC (adresse IP et port) a été lue dans le fichier /**config/socket_config.ini** (voir étape 2) qui contient entre autres ces lignes :
[PLC]
ip=127.0.0.1
port=5003

Pour lancer TOUS les simulateurs, voir la fonction sims_launch() de pyros.py ($ python **pyros.py sims_lauch).** Mais ce n'est pas ce qu'on veut pour tester le Monitoring tout seul...

En fait lancer UNIQUEMENT le simulateur de PLC (et pas les autres). Pour cela il devrait suffire de faire (dans un processus ou un thread, lancé avec "subprocess.Popen()") quelque chose du genre **:**

sim = PLCSimulator(scenario_plc.json)
sim.run()

Rappel : ce run() n'est pas exécuté dans Celery, ça n'a rien à voir, c'est juste du python, rien d'autre

(le fichier scenario_plc.json doit contenir les événements que le simulateur doit générer)

NB : ce simulateur PLC ne sera pas utilisé quand on aura le vrai PLC, mais il continuera toujours d'être utilisé dans les tests.

Voici comment faire :
Dans un 3ème terminal (T2), activer l'environnement virtuel, puis :

```
(venv) $ python plcSimulator.py scenario_plc.json
```

NB: plcSimulator.py peut être appelé avec ou sans argument. Sans argument (pas de fichier scenario), il le génèrera aucun événement, c'est à dire qu'il répondra toujours de la même façon à une même question posée par le Monitoring (alors qu'avec un scenario, la réponse pourra varier, ainsi que son comportement).

Si on regarde maintenant les messages reçus par le worker (terminal T0), on voit qu'il **n'affiche plus** le message :

FAILED TO CONNECT TO DEVICE PLC

Mais uniquement la ligne :

Timer : timer_status executed by monitoring

Cela montre que la connexion au PLC (c'est à dire au simulateur) se passe bien.

Voici en fait le résultat qu'on est censé obtenir avec le scénario scenario_plc.json :
Ce scénario contient les événements suivants (pour l'instant "time" signifie plutôt "numéro d'itération de la boucle du PLC" :

```
[
  10,
  {
      "time" : 3,
      "plcSimulator" : {"device_name":"WXT520", "value_name":"RainRate",
"value":12.0}
  },
  {
      "time" : 3,
      "plcSimulator" : {"device_name":"VantagePro",
"value_name":"RainRate", "value":12.0}
  },
  {
      "time" : 7,
      "plcSimulator" : {"device_name":"WXT520", "value_name":"RainRate",
"value":0.0}
  },
  {
      "time" : 7,
      "plcSimulator" : {"device_name":"VantagePro",
"value_name":"RainRate", "value":0.0}
  }
]
```

Comme on peut le deviner, à l'itération 3, les capteurs de pluie WXT520 et VantagePro devront indiquer un niveau de pluie de 12.0, puis un niveau 0.0 à l'itération 7.
Voyons si notre agent Monitoring (ENV) constate bien ces mêmes faits (voir les données en rouge ci-dessous). Sur T0, on obtient normalement les messages ci-dessous (au moment du lancement du simulateur sur T2, en admettant qu'il a été lancé un peu avant l'itération 19 du Monitoring) :

AGENT Monitoring (ENV): iteration 19, (my state is RUNNING) :
[2018-02-20 17:54:16,012: WARNING/Worker-1] Status received from PLC (read and parsed ok):
[2018-02-20 17:54:16,012: WARNING/Worker-1] [{"name": "STATUS", "from": "Beckhoff", "version_firmware": "20170809", "site": "OSM-Mexico", "date": "2017-03-03T13:45:00", "device": [{"name": "**WXT520**", "type": "meteo", "serial_number": "14656423", "valid": "yes", "values": [{"name": "OutsideTemp", "value": 12.12, "unit": "Celcius", "comment": ""}, {"name": "OutsideHumidity", "value": 64.1, "unit": "percent", "comment": ""}, {"name": "Pressure", "value": 769.2, "unit": "mbar", "comment": "At site elevation"}, {"name": "**RainRate**", "value": **0.0**, "unit": "mm/h",

"comment": ""}, {"name": "WindSpeed", "value": 3.1, "unit": "m/s", "comment": ""}, {"name": "WindDir", "value": 202.3, "unit": "deg", "comment": "(N=0, E=90)"}, {"name": "DewPoint", "value": 8.3, "unit": "deg", "comment": ""}]}, {"name": "DRD11", "type": "meteo", "serial_number": "RET6789", "valid": "yes", "values": [{"name": "analog", "value": 2.345, "unit": "V", "comment": "3V=Dry <2.5V=Rain"}, {"name": "digital", "value": 1, "unit": "bool", "comment": "1=Dry 0=Rain"}]}, {"name": "**VantagePro**", "type": "meteo", "serial_number": "ERTRY2344324", "valid": "no", "values": [{"name": "OutsideTemp", "value": 12.45, "unit": "Celcius", "comment": ""}, {"name": "InsideTemp", "value": 20.28, "unit": "Celcius", "comment": ""}, {"name": "OutsideHumidity", "value": 65.3, "unit": "percent", "comment": ""}, {"name": "InsideHumidity", "value": 45.6, "unit": "percent", "comment": ""}, {"name": "Pressure", "value": 768.7, "unit": "mbar", "comment": "At site elevation"}, {"name": "**RainRate**", "value": **0.0**, "unit": "mm/h", "comment": ""}, {"name": "WindSpeed", "value": 2.8, "unit": "m/s", "comment": ""}, {"name": "WindDir", "value": 207.0, "unit": "deg", "comment": "(N=0, E=90)"}, {"name": "WindDirCardinal", "value": "SW", "unit": "string", "comment": ""}, {"name": "DewPoint", "value": 8.5, "unit": "deg", "comment": ""}]}, {"name": "MLX90614-1", "type": "meteo", "serial_number": "Unknown", "valid": "yes", "values": [{"name": "SensorTemperature", "value": 23.56, "unit": "Celcius", "comment": ""}, {"name": "SkyTemperature", "value": -15.67, "unit": "Celcius", "comment": "Clear<-10 VeryCloudy>4"}]}, {"name": "LAMP_FLAT_CAGIRE", "type": "calib", "serial_number": "REF_3434", "valid": "yes", "values": [{"name": "status", "value": "on", "unit": "string", "comment": "on or off"}, {"name": "current", "value": 0.234, "unit": "Ampere", "comment": ""}]}, {"name": "LAMP FLAT CAGIRE", "type": "calib", "serial_number": "REF_3434", "valid": "yes", "values": [{"name": "status", "value": "on", "unit": "string", "comment": "on or off"}, {"name": "current", "value": 0.234, "unit": "Ampere", "comment": ""}]}]}]}]
Timer : **timer_status** executed by monitoring

AGENT Monitoring (ENV): iteration 20, (my state is RUNNING) :
TaskId matching query does not exist.
TaskId matching query does not exist.
Timer : **timer_tasks** executed by monitoring

AGENT Monitoring (ENV): iteration 21, (my state is RUNNING) :
[2018-02-20 17:54:18,049: WARNING/Worker-1] Status received from PLC (read and parsed ok):
[2018-02-20 17:54:18,049: WARNING/Worker-1] [{"name": "STATUS", "from": "Beckhoff", "version_firmware": "20170809", "site": "OSM-Mexico", "date": "2017-03-03T13:45:00", "device": [{"name": "**WXT520**", "type": "meteo", "serial_number": "14656423", "valid": "yes", "values": [{"name": "OutsideTemp", "value": 12.12, "unit": "Celcius", "comment": ""}, {"name": "OutsideHumidity", "value": 64.1, "unit": "percent", "comment": ""}, {"name": "Pressure", "value": 769.2, "unit": "mbar", "comment": "At site elevation"}, {"name": "**RainRate**", "value": **12.0**, "unit": "mm/h", "comment": ""}, {"name": "WindSpeed", "value": 3.1, "unit": "m/s", "comment": ""}, {"name": "WindDir", "value": 202.3, "unit": "deg", "comment": "(N=0, E=90)"}, {"name": "DewPoint", "value": 8.3, "unit": "deg", "comment": ""}]}, {"name": "DRD11", "type": "meteo", "serial_number": "RET6789", "valid": "yes", "values": [{"name": "analog", "value": 2.345, "unit": "V", "comment": "3V=Dry <2.5V=Rain"}, {"name": "digital", "value": 1, "unit": "bool", "comment": "1=Dry 0=Rain"}]}, {"name": "**VantagePro**", "type": "meteo", "serial_number": "ERTRY2344324", "valid": "no", "values": [{"name": "OutsideTemp", "value": 12.45, "unit": "Celcius", "comment": ""}, {"name": "InsideTemp", "value": 20.28, "unit": "Celcius", "comment": ""}, {"name": "OutsideHumidity", "value": 65.3, "unit": "percent", "comment": ""}, {"name": "InsideHumidity", "value": 45.6, "unit": "percent", "comment": ""}, {"name": "Pressure", "value": 768.7, "unit": "mbar", "comment": "At site elevation"}, {"name": "**RainRate**", "value": **12.0**, "unit": "mm/h", "comment": ""}, {"name": "WindSpeed", "value": 2.8, "unit": "m/s", "comment": ""}, {"name": "WindDir", "value": 207.0, "unit": "deg", "comment": "(N=0, E=90)"}, {"name": "WindDirCardinal", "value": "SW", "unit": "string", "comment": ""}, {"name": "DewPoint", "value": 8.5, "unit": "deg", "comment": ""}]}, {"name": "MLX90614-1", "type": "meteo", "serial_number": "Unknown", "valid": "yes", "values": [{"name": "SensorTemperature", "value": 23.56, "unit": "Celcius", "comment": ""}, {"name": "SkyTemperature", "value": -15.67, "unit": "Celcius", "comment": "Clear<-10 VeryCloudy>4"}]}, {"name": "LAMP_FLAT_CAGIRE", "type": "calib", "serial_number": "REF_3434", "valid": "yes", "values": [{"name": "status", "value": "on", "unit": "string", "comment": "on or off"}, {"name": "current", "value": 0.234, "unit": "Ampere", "comment": ""}]}, {"name": "LAMP FLAT CAGIRE", "type": "calib", "serial_number": "REF_3434", "valid": "yes", "values": [{"name":

"status", "value": "on", "unit": "string", "comment": "on or off"}, {"name": "current", "value": 0.234, "unit": "Ampere", "comment": ""}]}]}]
Timer : **timer_status** executed by monitoring

AGENT Monitoring (ENV): iteration 22, (my state is RUNNING) :
[2018-02-20 17:54:18,049: WARNING/Worker-1] Status received from PLC (read and parsed ok):
[2018-02-20 17:54:18,049: WARNING/Worker-1] [{"name": "STATUS", "from": "Beckhoff", "version_firmware": "20170809", "site": "OSM-Mexico", "date": "2017-03-03T13:45:00", "device": [{"name": "**WXT520**", "type": "meteo", "serial_number": "14656423", "valid": "yes", "values": [{"name": "OutsideTemp", "value": 12.12, "unit": "Celcius", "comment": ""}, {"name": "OutsideHumidity", "value": 64.1, "unit": "percent", "comment": ""}, {"name": "Pressure", "value": 769.2, "unit": "mbar", "comment": "At site elevation"}, {"name": "**RainRate**", "value": **12.0**, "unit": "mm/h", "comment": ""}, {"name": "WindSpeed", "value": 3.1, "unit": "m/s", "comment": ""}, {"name": "WindDir", "value": 202.3, "unit": "deg", "comment": "(N=0, E=90)"}, {"name": "DewPoint", "value": 8.3, "unit": "deg", "comment": ""}]}, {"name": "DRD11", "type": "meteo", "serial_number": "RET6789", "valid": "yes", "values": [{"name": "analog", "value": 2.345, "unit": "V", "comment": "3V=Dry <2.5V=Rain"}, {"name": "digital", "value": 1, "unit": "bool", "comment": "1=Dry 0=Rain"}]}, {"name": "**VantagePro**", "type": "meteo", "serial_number": "ERTRY2344324", "valid": "no", "values": [{"name": "OutsideTemp", "value": 12.45, "unit": "Celcius", "comment": ""}, {"name": "InsideTemp", "value": 20.28, "unit": "Celcius", "comment": ""}, {"name": "OutsideHumidity", "value": 65.3, "unit": "percent", "comment": ""}, {"name": "InsideHumidity", "value": 45.6, "unit": "percent", "comment": ""}, {"name": "Pressure", "value": 768.7, "unit": "mbar", "comment": "At site elevation"}, {"name": "**RainRate**", "value": **12.0**, "unit": "mm/h", "comment": ""}, {"name": "WindSpeed", "value": 2.8, "unit": "m/s", "comment": ""}, {"name": "WindDir", "value": 207.0, "unit": "deg", "comment": "(N=0, E=90)"}, {"name": "WindDirCardinal", "value": "SW", "unit": "string", "comment": ""}, {"name": "DewPoint", "value": 8.5, "unit": "deg", "comment": ""}]}, {"name": "MLX90614-1", "type": "meteo", "serial_number": "Unknown", "valid": "yes", "values": [{"name": "SensorTemperature", "value": 23.56, "unit": "Celcius", "comment": ""}, {"name": "SkyTemperature", "value": -15.67, "unit": "Celcius", "comment": "Clear<-10 VeryCloudy>4"}]}, {"name": "LAMP_FLAT_CAGIRE", "type": "calib", "serial_number": "REF_3434", "valid": "yes", "values": [{"name": "status", "value": "on", "unit": "string", "comment": "on or off"}, {"name": "current", "value": 0.234, "unit": "Ampere", "comment": ""}]}, {"name": "LAMP FLAT CAGIRE", "type": "calib", "serial_number": "REF_3434", "valid": "yes", "values": [{"name": "status", "value": "on", "unit": "string", "comment": "on or off"}, {"name": "current", "value": 0.234, "unit": "Ampere", "comment": ""}]}]}]
Timer : **timer_status** executed by monitoring

AGENT Monitoring (ENV): iteration 23, (my state is RUNNING) :
[2018-02-20 17:54:22,103: WARNING/Worker-1] Status received from PLC (read and parsed ok):
[2018-02-20 17:54:22,103: WARNING/Worker-1] [{"name": "STATUS", "from": "Beckhoff", "version_firmware": "20170809", "site": "OSM-Mexico", "date": "2017-03-03T13:45:00", "device": [{"name": "**WXT520**", "type": "meteo", "serial_number": "14656423", "valid": "yes", "values": [{"name": "OutsideTemp", "value": 12.12, "unit": "Celcius", "comment": ""}, {"name": "OutsideHumidity", "value": 64.1, "unit": "percent", "comment": ""}, {"name": "Pressure", "value": 769.2, "unit": "mbar", "comment": "At site elevation"}, {"name": "**RainRate**", "value": **0.0**, "unit": "mm/h", "comment": ""}, {"name": "WindSpeed", "value": 3.1, "unit": "m/s", "comment": ""}, {"name": "WindDir", "value": 202.3, "unit": "deg", "comment": "(N=0, E=90)"}, {"name": "DewPoint", "value": 8.3, "unit": "deg", "comment": ""}]}, {"name": "DRD11", "type": "meteo", "serial_number": "RET6789", "valid": "yes", "values": [{"name": "analog", "value": 2.345, "unit": "V", "comment": "3V=Dry <2.5V=Rain"}, {"name": "digital", "value": 1, "unit": "bool", "comment": "1=Dry 0=Rain"}]}, {"name": "**VantagePro**", "type": "meteo", "serial_number": "ERTRY2344324", "valid": "no", "values": [{"name": "OutsideTemp", "value": 12.45, "unit": "Celcius", "comment": ""}, {"name": "InsideTemp", "value": 20.28, "unit": "Celcius", "comment": ""}, {"name": "OutsideHumidity", "value": 65.3, "unit": "percent", "comment": ""}, {"name": "InsideHumidity", "value": 45.6, "unit": "percent", "comment": ""}, {"name": "Pressure", "value": 768.7, "unit": "mbar", "comment": "At site elevation"}, {"name": "**RainRate**", "value": **0.0**, "unit": "mm/h", "comment": ""}, {"name": "WindSpeed", "value": 2.8, "unit": "m/s", "comment": ""}, {"name": "WindDir", "value": 207.0, "unit": "deg", "comment": "(N=0, E=90)"}, {"name": "WindDirCardinal", "value": "SW", "unit": "string", "comment": ""}, {"name": "DewPoint",

"value": 8.5, "unit": "deg", "comment": ""}]}, {"name": "MLX90614-1", "type": "meteo", "serial_number": "Unknown", "valid": "yes", "values": [{"name": "SensorTemperature", "value": 23.56, "unit": "Celcius", "comment": ""}, {"name": "SkyTemperature", "value": -15.67, "unit": "Celcius", "comment": "Clear<-10 VeryCloudy>4"}]}, {"name": "LAMP_FLAT_CAGIRE", "type": "calib", "serial_number": "REF_3434", "valid": "yes", "values": [{"name": "status", "value": "on", "unit": "string", "comment": "on or off"}, {"name": "current", "value": 0.234, "unit": "Ampere", "comment": ""}]}, {"name": "LAMP FLAT CAGIRE", "type": "calib", "serial_number": "REF_3434", "valid": "yes", "values": [{"name": "status", "value": "on", "unit": "string", "comment": "on or off"}, {"name": "current", "value": 0.234, "unit": "Ampere", "comment": ""}]}]}]}]
Timer : **timer_status** executed by monitoring
TaskId matching query does not exist.
TaskId matching query does not exist.
Timer : **timer_tasks** executed by monitoring

AGENT Monitoring (ENV): iteration 24, (my state is RUNNING) :
[2018-02-20 17:54:22,103: WARNING/Worker-1] Status received from PLC (read and parsed ok):
[2018-02-20 17:54:22,103: WARNING/Worker-1] [{"name": "STATUS", "from": "Beckhoff", "version_firmware": "20170809", "site": "OSM-Mexico", "date": "2017-03-03T13:45:00", "device": [{"name": "**WXT520**", "type": "meteo", "serial_number": "14656423", "valid": "yes", "values": [{"name": "OutsideTemp", "value": 12.12, "unit": "Celcius", "comment": ""}, {"name": "OutsideHumidity", "value": 64.1, "unit": "percent", "comment": ""}, {"name": "Pressure", "value": 769.2, "unit": "mbar", "comment": "At site elevation"}, {"name": "**RainRate**", "value": **0.0**, "unit": "mm/h", "comment": ""}, {"name": "WindSpeed", "value": 3.1, "unit": "m/s", "comment": ""}, {"name": "WindDir", "value": 202.3, "unit": "deg", "comment": "(N=0, E=90)"}, {"name": "DewPoint", "value": 8.3, "unit": "deg", "comment": ""}]}, {"name": "DRD11", "type": "meteo", "serial_number": "RET6789", "valid": "yes", "values": [{"name": "analog", "value": 2.345, "unit": "V", "comment": "3V=Dry <2.5V=Rain"}, {"name": "digital", "value": 1, "unit": "bool", "comment": "1=Dry 0=Rain"}]}, {"name": "**VantagePro**", "type": "meteo", "serial_number": "ERTRY2344324", "valid": "no", "values": [{"name": "OutsideTemp", "value": 12.45, "unit": "Celcius", "comment": ""}, {"name": "InsideTemp", "value": 20.28, "unit": "Celcius", "comment": ""}, {"name": "OutsideHumidity", "value": 65.3, "unit": "percent", "comment": ""}, {"name": "InsideHumidity", "value": 45.6, "unit": "percent", "comment": ""}, {"name": "Pressure", "value": 768.7, "unit": "mbar", "comment": "At site elevation"}, {"name": "**RainRate**", "value": **0.0**, "unit": "mm/h", "comment": ""}, {"name": "WindSpeed", "value": 2.8, "unit": "m/s", "comment": ""}, {"name": "WindDir", "value": 207.0, "unit": "deg", "comment": "(N=0, E=90)"}, {"name": "WindDirCardinal", "value": "SW", "unit": "string", "comment": ""}, {"name": "DewPoint", "value": 8.5, "unit": "deg", "comment": ""}]}, {"name": "MLX90614-1", "type": "meteo", "serial_number": "Unknown", "valid": "yes", "values": [{"name": "SensorTemperature", "value": 23.56, "unit": "Celcius", "comment": ""}, {"name": "SkyTemperature", "value": -15.67, "unit": "Celcius", "comment": "Clear<-10 VeryCloudy>4"}]}, {"name": "LAMP_FLAT_CAGIRE", "type": "calib", "serial_number": "REF_3434", "valid": "yes", "values": [{"name": "status", "value": "on", "unit": "string", "comment": "on or off"}, {"name": "current", "value": 0.234, "unit": "Ampere", "comment": ""}]}, {"name": "LAMP FLAT CAGIRE", "type": "calib", "serial_number": "REF_3434", "valid": "yes", "values": [{"name": "status", "value": "on", "unit": "string", "comment": "on or off"}, {"name": "current", "value": 0.234, "unit": "Ampere", "comment": ""}]}]}]}]
Timer : **timer_status** executed by monitoring

Monitoring (ENV): iteration 25, (my state is RUNNING) :
(ENV) Could not send message (on socket) to PLC : {"command": [{"name": "STATUS"}]} -> [Errno 32] Broken pipe
Invalid PLC status returned (while reading) : NOT_SET1
Timer : **timer_status** executed by monitoring

AGENT Monitoring (ENV): iteration 26, (my state is RUNNING) :
TaskId matching query does not exist.
TaskId matching query does not exist.
Timer : **timer_tasks** executed by monitoring

…

Bien sûr, le temps des itérations du Monitoring n'est pas le même que celui des itérations du PLC, donc elles ne se correspondent pas… Mais on constate bien le passage du niveau de pluie de 0 à 12, puis de nouveau à 0, sur les 2 capteurs WXT520 et VantagePro. IT WORKS !

Et maintenant, regardons ce qui s'est passé côté base de données. L'agent Monitoring met à jour 2 tables pour l'environnement :

- **weatherwatch**, pour la météo (environnement **externe** de l'observatoire)
- **sitewatch**, pour le site (environnement **interne** de l'observatoire)

Pour voir le contenu de ces tables, utilisez PhpMyAdmin (ou bien la page administration de pyros : voir pour cela la section "Accéder à la page d'administration de PyROS")

Voici le contenu de ces tables, après exécution du simulateur :

Table **weatherwatch** :

id	global_status	updated	humidity	wind	wind_dir	temperature	pressure	rain	cloud
249	OK	2018-02-21 15:32:17.118269	65.3	2.8	207.0	NULL	768.7	0	NULL
250	**RAINING**	2018-02-21 15:32:19.156377	65.3	2.8	207.0	NULL	768.7	**12**	NULL
251	**RAINING**	2018-02-21 15:32:21.183933	65.3	2.8	207.0	NULL	768.7	**12**	NULL
252	OK	2018-02-21 15:32:23.218375	65.3	2.8	207.0	NULL	768.7	0	NULL
253	OK	2018-02-21 15:32:25.245074	65.3	2.8	207.0	NULL	768.7	0	NULL

Table **sitewatch** :

id	global_status	**updated**	lights	dome	doors	temperature	shutter	pressure	humidity
249	OK	2018-02-21 15:32:17.119959	NULL	NULL		NULL	NULL	NULL	45.6
250	OK	2018-02-21 15:32:19.157529	NULL	NULL		NULL	NULL	NULL	45.6
251	OK	2018-02-21 15:32:21.185069	NULL	NULL		NULL	NULL	NULL	45.6

252	OK	2018-02-21 15:32:23.219474	*NULL*	*NULL*		*NULL*	*NULL*	*NULL*	45.6
253	OK	2018-02-21 15:32:25.246233	*NULL*	*NULL*		*NULL*	*NULL*	*NULL*	45.6

Autre méthode imaginable (à tester, ne marche pas pour l'instant…)

(venv) $ python

>>> from simulators.plc.plcSimulator import PLCSimulator

>>> sim = PLCSimulator('config/conf.json')

>>> sim.run()

(mais pour l'instant, ça plante…)

1.8.5. DEVELOPMENT

Informations techniques nécessaires pour le dev :

Toute la BD est décrite dans le fichier src/common/models.py

Quand on clique sur les icones Weather et Observatory, ça déclenche les actions weather et site qui sont dans src/dashboard/views.py

Ces actions utilisent respectivement les vues reload_weather.html et reload_site.html qui sont dans src/dashboard/templates/dashboard/

Ces vues déclenchent respectivement les actions views.py.weather_current et views.py.site_current qui utilisent respectivement les vues current_weather.html et current_site.html (toujours dans src/dashboard/templates/dashboard)

1.8.6. TODO LIST : Que reste-t-il à faire ?

(1) - Quelques bugfixes sont encore nécessaires :

- ~~Le log de Monitoring fonctionne bien (cf /logs/Monitoring.log) mais pas les autres utilisés (Devices.log pour DeviceController et DeviceSim.log pour PLCSimulator)~~

(2) - Dans quel sens doit-on faire progresser le module Monitoring ? :

- Déplacer les actions (views.py) et vues (templates) du dossier dashboard vers le dossier envmonitor
- Changer l'affichage des pages weather et observatory pour qu'elles affichent seulement la dernière ligne de la table (au lieu d'afficher toutes les lignes). Ensuite, ajouter un joli graphique de l'évolution depuis le début de la nuit, avec un point toutes les 10 minutes.
- **Démarrage dans n'importe quel ordre :**
    - Tester que ça marche quand on lance l'agent Monitoring avant le PLC
    - Tester que ça marche quand on lance le PLC avant l'agent Monitoring
- **Tolérance de panne (résilience**) : la communication monitoring-PLC doit continuer à fonctionner même dans ces 3 cas :
    - **(1) ~~Le PLC ne répond plus pendant N secondes, puis répond à nouveau ⇒ c'est à dire, le simulateur est stoppé puis relancé~~**
    - **(2) ~~L'agent monitoring lui-même (task.run) est stoppé et relancé~~**
    - **(TODO: cf mail envoyé à Quentin vendredi soir)**
    - **(3)** Le worker celery dédié au monitoring est stoppé et relancé ⇒ que se passe-t-il dans ce dernier cas ?
- Faire une **page web** (une VUE django, dédiée au monitoring) des 2 tables synthèse du Monitoring, et garder cette page affichée (doit se rafraichir automatiquement) ; actuellement, la seule vue qu'on a de ces tables c'est via l'interface admin de django (localhost:8000/admin) mais c'est pas terrible, il faut une vue unique pour les 2 tables en même temps (et plus jolie). Cette page devra être accessible depuis le "dashboard" (http://localhost:8000) dans le menu de gauche, avec une entrée "**Environment**" ; on peut aussi imaginer faire évoluer cette page plus tard pour :
    - Qu'elle serve aussi à envoyer des commandes manuelles au PLC ("getStatus" ou autre)
    - qu'elle serve aussi à lancer des événements pour le PLC à la main (genre "il pleut maintenant"...)
    - Qu'elle permette de stopper et relancer manuellement :
        - Le simulateur de PLC
        - L'agent Monitoring
        - Le worker celery

- Ecrire une série de **tests unitaires et fonctionnels automatiques** : **ATTENTION, ces tests ne devront pas venir polluer la BD de production**, mais ils devront écrire dans une autre BD à part (de test), qui n'est pas forcément mysql mais peut être du sqlite par exemple…
- Il faudrait créer un agent "**superagent**" dont le seul rôle est de surveiller TOUS les agents (Monitoring (ENV), Majordome, AlertManager) et de les relancer si besoin

A réfléchir et préciser (avec Alain K) :
Monitoring doit faire une **synthèse intelligente** (à destination du Majordome qui prendra la décision adéquate). Pour chaque élément (pluie, nuages, vent, humidité, ...) faire une synthèse avec 3 valeurs {valeur élement ; interprétation ; décision à prendre}. Par exemple :
Pluie = 5 ; "il pleut fort" ; mise en sécurité = YES
Pluie = 0 ; "il ne pleut pas" ; mise en sécurité = NO
Cloud = 1 ; "un peu nuageux" ; mise en sécurité = NO

1.9. FONCTION 6 - DATA REDUCTION & ANALYSIS

1.9.1. Basic packages requirement.

Scientific package requirements for this module:

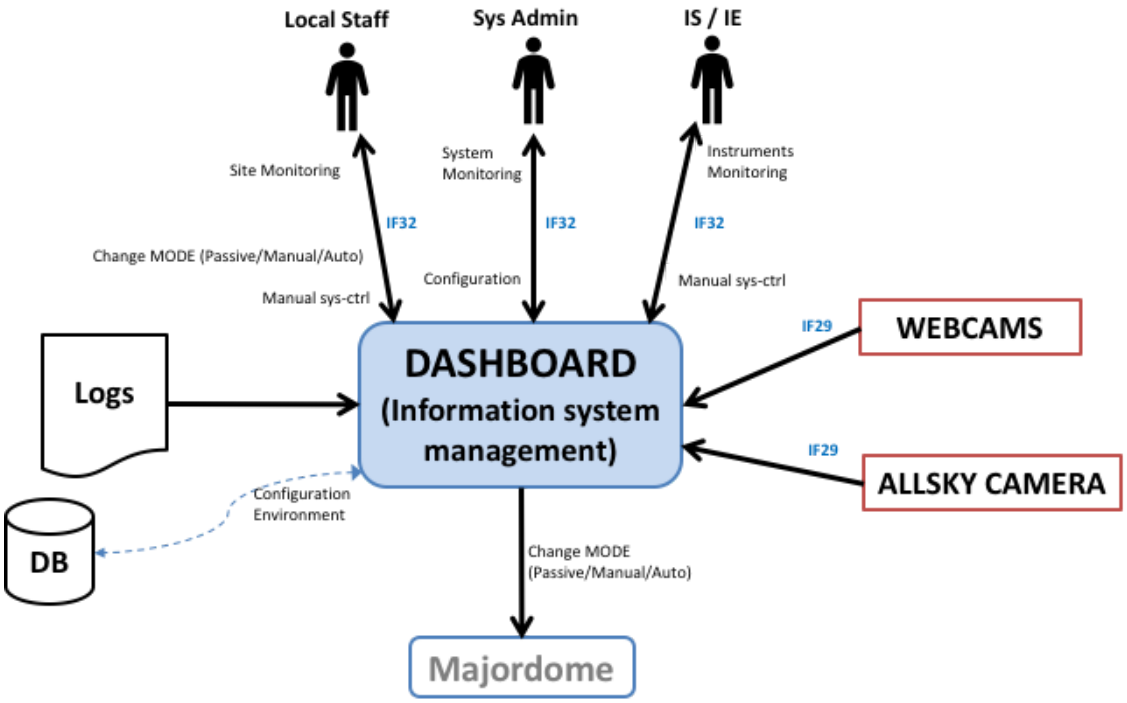
1. Astropy 3.0 : "Installation"
   with other packages: numpy, h5py, BeautifulSoup, PyYAML, scipy, xmlilint, matplotlib, pytz, scikit-image, pandas, objgraph, setuptools, bleach, bintrees.
2. Scikit-Learn : "Installation".
3. Healpy : "Installation".
4. SExtractor "Installation" (if possible), need configuration files.
5. Astrometry.net "Installation" (if possible), need database.

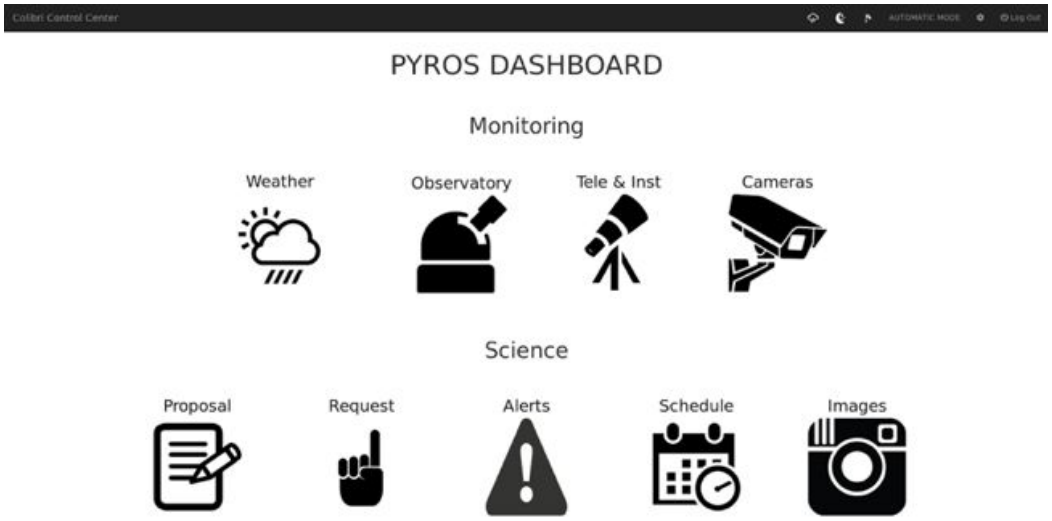
6.3.2 Image Calibration
6.3.3 Image editor
6.3.4 Image analysis
6.3.5 Source Extraction
6.3.5 Distortion correction

1.10. FONCTION 7 - DASHBOARD (Information system management)

(updated 18/05/18)

Responsible : Théo Puhl





Fonctions du module :

GFT-REQ-299: the “Information System management” must take in charge the following actions:

- Manage users, profiles, users priority and quota
- Supply a short-term follow-up and a middle term of the observations
- Prepare and schedule correction & calibration sequences (flatfield, dark, bias, ...), and update calibration parameters
- Manage logs of the application
- Show the status of the different subsystems
- Allow to change the current MODE (manual/auto)
- Manual system control of the telescope (& instruments)
- Software & Hardware configuration
- Infrastructure management
- Backup and Restoration
- Webcams display

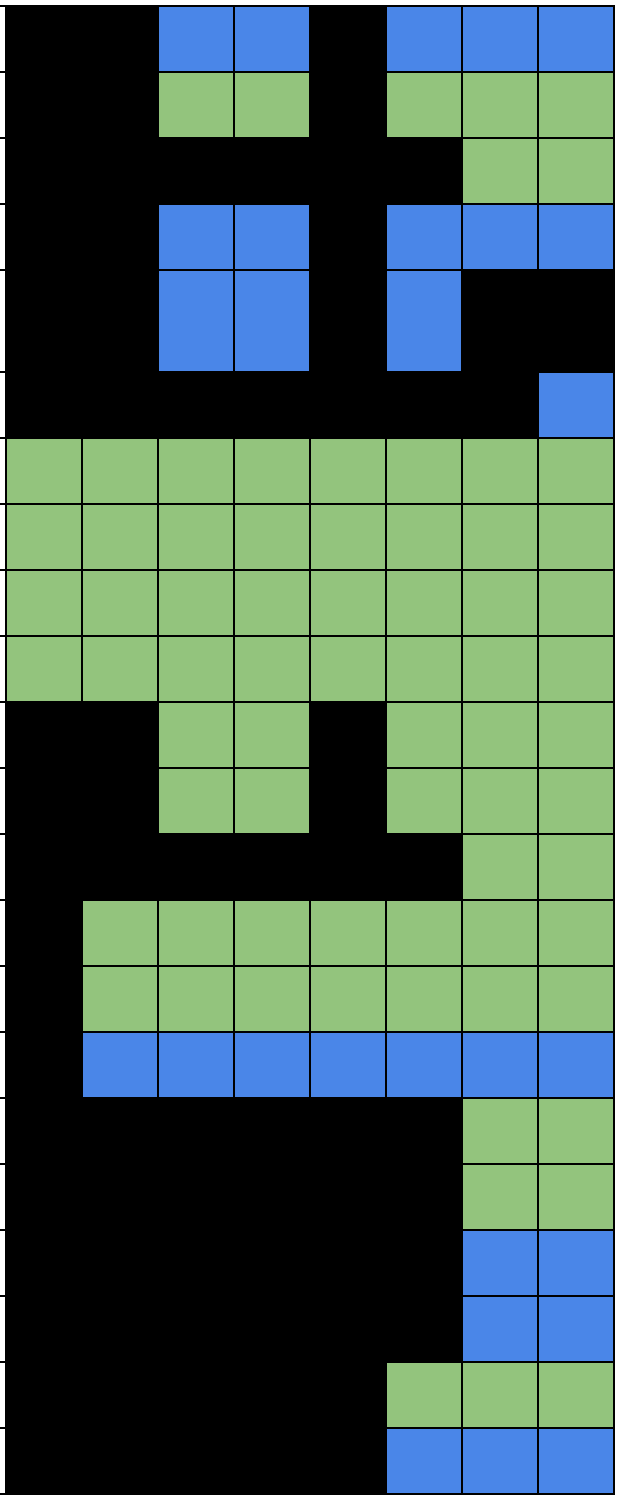
1.10.1. Users management

6.10.1.1. Profiles

N°	Profile	Rights
0	**None (Visitor)**	+ Weather + Observatory + Webcams + Alert + Schedule
1	**TAC**	+ Proposal
2	**Observer (astronomer)**	+ Proposal (pour la période suivante) + Request (associée à un SP) + Images + Profil perso (edit only “other email”, “password”) + Tel & Inst (voir status only)
3	**IE/IS**	+ Observatory control page (Change operating mode (REMOTE/SCHEDULER)) + Tele & Inst (ctrl cde)
4	**Operator (LOCAL)**	+ Give starting night ACK, PLC bypass-authorize/forbid in Observatory control page - Request - Proposal - Images
5	**Superoperator (AK)**	+ Config
6	**PI**	+ User rights management + Proposal (+ prio + quota) *(Prévoir protections quand même !!!)*
7	**SysAdmin**	Super user rights (sans filet)

Detailed rights :

PROFILE / RIGHT	0	1	2	3	4	5	6	7
Weather								
- Log								
Observatory (status)								
- Log								
Observatory (Control)								
- Give night ACK								
- Bypass PLC (SAFE/UNSAFE)								
- Change Operating Mode (PASSIVE/REMOTE/SCHEDULER)								
Tel & Inst								
- Status								
- Control Command								
Webcams								
- Log								
Proposals								
- See all proposals								
- Create								
- Update								
- View Vote								
- Vote								
- Set Quota & Prio								
- Delete (**personal** before SP)								
- Delete (**all** before SP)								
Request								

- Create	
- Personal SP	
- All request	
- Update	
- Delete (personal one, only if not submitted)	
- Delete ALL (warning, attention)	
Alerts	
- List of alerts	
Schedule	
- Planning	
Images	
- Personal images	
- All images	
User profile	
- Personal info	
- Edit (only “other email”, “password”)	
Users Management	
- All users info	
- Edit (all users, all fields)	
- Deactivate	
Config	
- Edit Config	

Black block - Hidden data for the user
Green block - Informative data for the user
Blue block - Interaction with the data for the user

6.10.1.2. Scenario (workflow)

Version 1

1) un nouvel utilisateur s'enregistre via un registration form ("sign in"), dans lequel il décrit ses intentions
2) il reçoit un mail l'informant que son compte a bien été créé mais qu'il est en attente de validation par le PI Colibri
3) the Colibri PI reçoit un mail contenant un lien qui permet d'activer ce nouveau compte (ou pas)
NB: pour l'instant, le mail contient seulement un lien vers la page web qui permet d'activer le compte du nouvel utilisateur (en cliquant sur un bouton "Activate") et il faudra que le PI soit connecté pour y accéder… on fera mieux plus tard...
4) le nouvel utilisateur reçoit alors aussi un mail l'informant que son compte a été activé
5) il peut alors se connecter au site et faire les actions suivantes :
- soit déposer un new proposal pour la prochaine période de 6 mois
- soit déposer une requete d'observation pour un SP (Scientific Program) de la période de 6 mois en cours, SP auquel il est déjà associé (car il l'avait soumis à la période précédente)

A tout moment, le PI peut décider de désactiver un utilisateur (il pourra toujours le réactiver si besoin). L'utilisateur concerné recevra alors un email pour l'en informer.

Un utilisateur pourra soumettre PLUSIEURS proposals et donc il pourra plus tard soumettre des requetes d'observation sur plusieurs SP. Au moment de soumettre une requete d'observation, il doit sélectionner le SP concerné dans une liste déroulante ; si cette liste est vide, il ne pourra tout simplement pas soumettre de requete.

Dans cette version, il y a un **compte UNIQUE pour un SP** (prog scientifiq). Les différentes personnes contribuant à ce SP utiliseront donc le MEME compte et s'arrangeront entre elles pour gérer leur quota...

Version 2

- la personne qui dépose un proposal (futur SP) en devient automatiquement le PI (on dira le **SP-PI**)
- les autres utilisateurs peuvent demander à être associé à un ou plusieurs SP(s) par demande directe au SP-PI (email).
- le SP-PI doit donc avoir accès à une liste de TOUS les utilisateurs pour sélectionner ceux qui sont demandeurs pour son SP ou dé-selectionner ceux qui ne le sont plus...
- les utilisateurs sélectionnés (ou dé-selectionnés) reçoivent alors un email pour les en informer
- Un utilisateur pourra soumettre (**ou être associé à)** PLUSIEURS proposals.

1.10.2. Observatory Control page

(updated 4/7/18)

Le ACK est donné une fois pour toutes en début de nuit (il n'est pas enlevé ensuite, sauf automatiquement après la fin de la nuit)
Attention, ce ACK n'est pas donné via la page web, mais via un bouton physique dans la control room ; on reçoit donc l'info via le PLC

Voici les BOUTONS d'action qui seront disponibles sur la page web "OBSERVATORY CONTROL PAGE" :

(tous ces boutons seront bien sûr soumis à une confirmation, of couse !)

Bouton 1 - "PLC_BYPASS (set to SAFE)" ou "STOP PLC_BYPASS"
=> affiché ssi UNSAFE
- "PLC_BYPASS" fait passer à SAFE.
- "STOP PLC_BYPASS" permettra de mettre fin au bypass (le CC tiendra donc à nouveau compte du PLC status, qu'il soit SAFE ou UNSAFE)
Remarque : attention, ça ne marche que si UNSAFE (pas si SAFE), comme demandé par François ; dans le cas SAFE, on utilise plutôt le bouton 2

Bouton 2 - "LOCK OBSERVATORY" ou "UNLOCK OBSERVATORY"
==> affiché ssi SAFE
- "LOCK OBSERVATORY" fait passer à STANDBY (après close dome...) et y reste bloqué (LOCKED = true)
- "UNLOCK OBSERVATORY" permettra de repasser la variable LOCKED à False ce qui permettra de sortir du mode STANDBY dans lequel on était bloqué
Remarques:
- attention, ces boutons ne changent pas l'état SAFE)
- pour moi, ces boutons doivent pouvoir aussi être activables depuis le mode REMOTE-COMMAND

Bouton 3 - "GO REMOTE-COMMAND MODE" ou "GO SCHEDULER MODE"
Ces boutons ne sont activables QUE si la variable LOCKED est False (voir bouton 2)

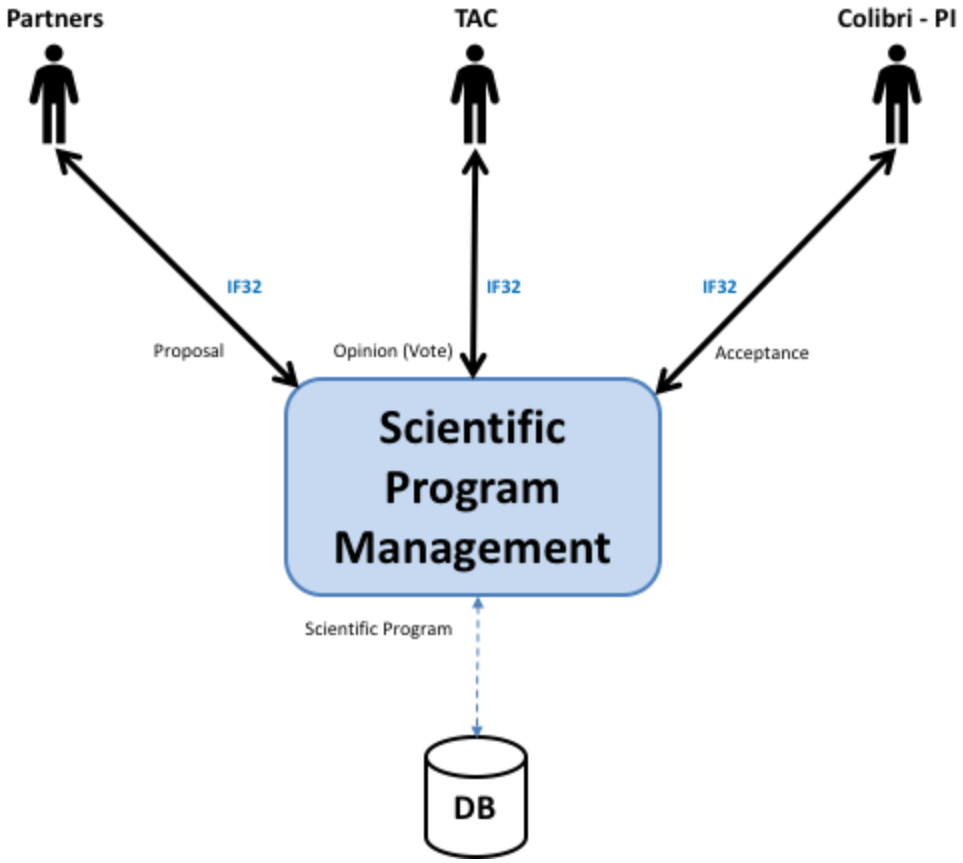
Dans le détail:

Bouton	Visible ssi	Action
"**BYPASS_PLC** (set to SAFE)" **OU** **"STOP BYPASS_PLC"**	**UNSAFE & OPERATOR** **SAFE & OPERATOR**	**Bypass le mode UNSAFE du PLC en SAFE**. Désormais CC ne tient plus compte du status du PLC... **Annule le bypass**. CC tient à nouveau compte du PLC status. Ces 2 actions sont transmises au PLC pour qu'il le prenne en compte (bypass=1 ou 0). CC doit vérifier ensuite qu'il reçoit bien cette information du PLC, ce qui prouve que le PLC l'a bien prise en compte (sinon, au bout d'un timeout, "PLC KO" error)
(staff LOCK/UNLOCK) **"LOCK OBSERVATORY (CLOSE)"** **OU** **"UNLOCK OBSERVATORY (OPEN)"** UNLOCK ne fait que supprimer le "LOCK")	**OPERATOR & SAFE** ⇒ ssi mode SCHED-READY (ou SCHED-STANDBY ?) Ou mode REMOTE ⇒ ssi mode SCHED-STANDBY et STOPPED is TRUE (le mode SCHEDULER doit pouvoir se réactiver tout seul automatiquement après chaque passage à SAFE par le PLC, si LOCKED est à false, ce qui est le cas par défaut)	⇒ **passe variable LOCKED à TRUE, et revient au mode STANDBY** (pour y **rester bloqué**) après être passé par SCHEDULER-CLOSING (ou pas si c'était REMOTE) ⇒ **supprime le LOCK** ; repasse la variable LOCKED à FALSE (ce qui devrait faire passer automatiquement au mode SCHEDULER-READY) si toutes les conditions sont favorables)

	>= IE/IS	
“GO REMOTE MODE” **OR**	⇒ ssi **UNLOCKED** ET mode **SCHED-STANDBY** ou **SCHED-READY** (peu importe qu’on soit en SAFE ou UNSAFE)	⇒ **passe en mode REMOTE-COMMAND**
“GO SCHEDULER MODE”	⇒ ssi **UNLOCKED & SAFE** ET mode **REMOTE**	⇒ **passe en mode SCHEDULER-STANDBY** (qui passera automatiquement à **READY**)

Question en suspens : quid du mode **REMOTE** ? doit-on autoriser l'action "STOP OBSERVING" depuis ce mode pour forcer la fermeture du dome par exemple ?

1.11. FONCTION 8 - (Proposals) Scientific Programs Management



1.12. FONCTION 9 - Telescope & Instruments long term Monitoring and Calibration

Cette fonction est assurée par le GIC (@CPPM)

1.13. FONCTION 10 - Data Archiving

Données (L1) envoyées au LAM et stockées sans doute à l'IN2P3 (TBC)
Voir Christian Surace (Colibri Database)

MODULE	AGENT	WEB ACTIONS	NON-WEB ACTIONS (AGENT)
USER Users management	**NO**	View LOG (users history) Register Login/logout Edit profile	
REQUEST (REQ) Observation Requests management	**NO**	View LOG Observation Request **submission** (via form or XML file) ⇒ call Planner.schedule() View/Edit Request (+ Sequences, Albums & Plans) (CRUD) Get Images (test) Generate “Fake request”	
ALERT GRB Alerts management (SVOM and other)	**YES**	View LOG View alert request (can be done by the REQUEST module above) History : see all past alerts and their outcome (test) Generate “Fake alert”	**LOOP** : 1 - Wait for new alert (VOEvent) from alert network (SVOM/FSC or other) (VTP/XMPP protocol) 2 - Process VOEvent and create an Observation Request (save to DB) 3 - Call Planner.schedule()
PLANNER (SCHEDULER?) Sequences planning	**NO** *Just call its method* ***schedule()***	View LOG View current plan (updated real time) View past plans (admin) Manual call to re-schedule()	Schedule (make planning from sequences in DB) ⇒ call Majordome.new_schedule()
ENVIRONMENT (ENV) Environment	**YES**	View LOG View current synthesis (environment status for Weather	**LOOP**: 1 - Ask PLC for all statuses

Monitoring		and Site) View past synthesis *(admin) Manual Restart* *(test) Fake alarm (it rains)*	2 - Wait for PLC answer 3 - Make synthesis (save to DB)
MAJORDOME (MAJOR) General conductor of the system (scheduler)	**YES**	View LOG View Majordome current status (pause/idle/executing sequence/…) View Majordome current mode (AUTO/MAN) (admin) Change mode (AUTO/MAN) (admin) MANUAL actions (cd-ctrl): - Telescope - Dome - Lights (via PLC) - Power (via PLC) - … View Instruments current status (telescope, detectors)	LOOP: 1 - (every 5 s) Read Environment synthesis (from DB) and take relevant action if necessary 1 - (every 5 s) Read Time (night start/end) and take relevant action if necessary 1 - (every 10 s) Read Instruments status (from DB, **synthesis** done by the OBSERVER module) and take relevant action if necessary 1 - (on new schedule available from Planner) Read new schedule (from DB) and take relevant action 1 - (on Alert CANCEL from ALERT module) Cancel current alert
OBSERVER (EXEC, EYE) Sequence execution	**NO**	View LOG View current status (idle, executing sequence/plan/acquisition...)	**Not necessarily an Agent** : this process can be called by MAJORDOME each time there is a new sequence to be executed (or cancelled) **If Agent** : LOOP: 1 - (every 5 s) Get status from instruments and make synthesis (saved to DB) 1 - (on NEW sequence to be executed, from MAJORDOME) : Execute sequence 1 - (on CANCEL sequence, from MAJORDOME) : Cancel sequence

			1 - (on sequence execution finished) : tell MAJORDOME
CONDITIONS (Observing Conditions) Monitoring of the Sky observing conditions	**?**		Asked regularly by MAJORDOME ?
GP1 **1 - Images Correction & Calibration** **2 - Images NRT Analysis (only if alert...)**			**If Agent** : (on new image in folder) : process image **BUT Not necessarily an Agent** : this process can be called by MAJORDOME or OBSERVER each time there is a new image available