Étude et réalisation d'une application d'asservissement : positionnement d'un faisceau Synchrotron

Falilou Thiam

CONSERVATOIRE NATIONAL DES ARTS ET METIERS

PARIS

MEMOIRE

présenté en vue d'obtenir

le DIPLOME D'INGENIEUR CNAM

SPECIALITE : INFORMATIQUE

OPTION : ARCHITECTURE ET INGÉNIERIE DES SYSTÈMES ET DES LOGICIELS (AISL)

par

THIAM Falilou

Étude et réalisation d'une application d'asservissement

Positionnement d'un faisceau Synchrotron

Soutenu le 09 Octobre 2018

JURY

PRESIDENT : **Monsieur Yann POLLET**

MEMBRES : **Madame Amélie LAMBERT**
Monsieur Jean-Michel DOUIN
Monsieur Alain BUTEAU
Monsieur Florent LANGLOIS

Remerciements

Premièrement, je remercie M. Alain Buteau pour m'avoir confié ce projet et permis de réaliser mon mémoire de fin d'étude au sein du groupe ICA. Je remercie aussi M. Florent Langlois pour son accompagnement tout au long de ce projet et l'ensemble de l'équipe ICA pour leur disponibilité.

Je tiens aussi à remercier les membres de la ligne NANOSCOPIUM et plus particulièrement M. Kadda Medjoubi pour m'avoir permis de mettre en œuvre mon développement et m'avoir accompagné durant les différentes phases de tests.

Enfin, je remercie M. Alain Lestrade et l'ensemble de mes collègues du groupe Alignement de m'avoir soutenu tout au long de ma formation d'ingénieur CNAM.

Liste des abréviations

SOLEIL: Source Optimisée de Lumière d'Energie Intermédiaire de LURE

LURE : Laboratoire d'Utilisation du Rayonnement Électromagnétique

LINAC: LINnear ACcelerator (accélérateur linéaire)

MeV: Méga électron Volt

GeV: Giga électron Volt

ICA: Informatique Contrôle Acquisition

TANGO: TAco Next Generation Object

TACO: Telescope and Accelerator Controlled with Objects

OMG: Object Management Group

CORBA: Common Object Request Broker Architecture

IDL: Interface Definition Language

ORB: Object Request Broker

IHM: Interface Homme Machine

XBPM: X-ray Beam Position Monitor

PID : Proportionnel Intégral Dérivé

LIMA: Library for IMage Acquisition (librairie de traitement d'image)

DSDM: Dynamic Systems Development Method

API: Application programming interface

AMOA: Assistant à maîtrise d'ouvrage

MOA: Maitrise d'ouvrage

MOE : Maitre d'œuvre

ROI : Region Of Interest (Région d'intérêt)

Glossaire

Binding : Permet l'utilisation d'une bibliothèque logicielle dans un autre langage de programmation que celui d'origine.

Scintillateur : Un matériau qui émet de la lumière à la suite de l'absorption d'un rayonnement ionisant.

Centroide : Sur le plan mathématique, le centroide d'une forme géométrique plane correspond à la position moyenne de tous les points composant cette forme.

Méthodologies Agiles : Groupe de méthodes de pilotage de projets. Initiées à la suite du manifeste Agile (2001). Méthodes ayant pour but d'impliquer fortement les clients et d'obtenir une grande réactivité à ses demandes.

Workflow : Flux de travaux. Une représentation d'une suite de tâches et d'états pour un projet donné.

Classe : En programmation orientée objet, la classe rassemble l'ensemble des méthodes et attributs d'un objet. C'est en fait dans la classe que sont déclarés toutes les méthodes, attributs et états pour un type d'objet donné. Lorsque la classe est instanciée, un objet de cette dite classe est créé.

Thread : Aussi appelé processus léger, est une file d'exécution d'un programme (suite d'instructions programmées) pouvant être géré de manière indépendante. Au sein d'un même processus, plusieurs Threads partagent la même mémoire virtuelle.

Mutex : Un outil informatique permettant l'accès à une ressource partagée de manière exclusive.

Maven : Un outil d'automatisation de compilation. Permet de construire des logiciels en spécifiant la manière de les compiler et en précisant les dépendances nécessaires.

Software Development Kit (SDK) : Outils pour le développeur permettant le développement logiciel pour une plateforme donnée. Une trousse à outils composée de bibliothèques prêtes à l'emploi pour faciliter le développement.

Table des matières

Introduction

SOLEIL est un accélérateur de particules de type Synchrotron. Ce centre de recherche international offre un rayonnement d'électrons capable d'explorer la matière. Des scientifiques du monde entier et travaillant dans de nombreux domaines de recherche viennent à SOLEIL pour bénéficier de ce rayonnement synchrotron. De nombreuses publications sont ainsi produites chaque année. Dans ce contexte d'excellence scientifique, énormément de défis techniques sont à l'œuvre. Le projet de développement informatique qui m'a été attribué rentre dans le cadre de l'innovation du contrôle commande à SOLEIL. En outre, ce nouveau service pourrait être utilisé par l'ensemble de la communauté Synchrotron de par le monde, ce qui contribuerait à la notoriété de SOLEIL.

Dans une première partie, je commencerai par décrire le principe de fonctionnement du synchrotron SOLEIL, et les principaux objectifs d'un tel instrument. Je décrirai le fonctionnement de l'informatique distribuée mise en place dans le cadre du contrôle commande. Je présenterai les technologies utilisées pour satisfaire l'exploitation du synchrotron. J'introduirai la problématique qui m'a été posée, et les motivations liées à ce nouveau projet.

Dans une deuxième partie, je présenterai comment ce projet a été conduit. J'identifierai les participants, les phases du projet et les outils logiciels et méthodologiques nécessaires à son pilotage. Je présenterai ensuite chaque étape de réalisation du projet : de la phase d'analyse avec la description du principe applicatif mis en place, en passant par la phase de conception et les différents choix de design architecturaux, pour finir avec la phase de réalisation en exposant les difficultés rencontrées à chaque étape.

Pour finir, dans une troisième partie, j'exposerai le bilan de ce projet. Je ferai un point sur l'atteinte des objectifs ainsi que la satisfaction client. Nous verrons les différentes perspectives d'évolutions envisagées pour la suite de ce projet. Pour terminer, je ferai un bilan sur les apports personnels acquis.

I. Contexte

I.1. Le Synchrotron SOLEIL

I.1.1. La technologie Synchrotron

Un Synchrotron est un centre de recherche pluridisciplinaire permettant la production d'une source de lumière intense servant à l'exploration de la matière.

Un Synchrotron est un accélérateur de particules produisant un rayonnement de type Synchrotron. Pour obtenir un tel rayonnement, il faut accélérer des électrons dans un anneau de stockage jusqu'à ce qu'ils atteignent une vitesse proche de celle de la lumière. Dès lors, pour chaque courbe qu'il prendra afin respecter la trajectoire circulaire de l'anneau de stockage, il perdra de l'énergie et produira ainsi un rayonnement synchrotron.

Pour disposer de ce « précieux » rayonnement, des postes expérimentaux sont placés tout autour de l'anneau de stockage, nous les appelons des lignes de lumière (Figure 1).

Figure 1 : Les 7 étapes pour le rayonnement synchrotron (*tiré de la banque d'images du synchrotron-soleil)*

Description des étapes de la Figure 1 :

Les électrons sont produits grâce à un canon à électrons puis sont ensuite accélérés dans le LINAC, un accélérateur linéaire (1) à une énergie de 100 MeV (Méga électron volt). Ensuite, ce faisceau d'électrons est envoyé dans un deuxième accélérateur, cette fois circulaire, le Booster (2). A ce stade les électrons sont alors portés à une énergie de 2.75 GeV (Giga électron volt). Il est désormais possible de les injecter dans l'anneau de stockage (3), anneau mesurant 354 mètres de périmètre. L'ensemble LINAC, Booster et Anneau de stockage constitue ce que l'on appelle couramment la « Machine ». Une fois dans l'anneau de stockage, les électrons sont déviés de leur trajectoire par des dispositifs magnétiques (des dipôles, des onduleurs, des wigglers (4)), ce qui engendrera une perte d'énergie sous forme de rayonnement synchrotron. Pour chaque tour dans l'anneau, cette perte d'énergie va venir être compensée par des cavités radiofréquence (5). Le rayonnement synchrotron est ensuite dirigé vers les stations expérimentales (6), pour être utilisé par la ligne de lumière (7). Toutes ces opérations pour créer, monitorer et entretenir le faisceau synchrotron sont réalisées en salle de contrôle qui se trouve physiquement au centre de l'anneau et où des opérateurs y travaillent 24H/24 et 7 jours sur 7.

Le rayonnement synchrotron est en fait une lumière blanche qui est constituée de toutes les longueurs d'onde (de l'infra-rouge jusqu'aux rayons X). On note qu'auparavant, le rayonnement synchrotron était considéré comme un phénomène parasite, mais qu'il est désormais utilisé pour ses propriétés uniques. Il est principalement utilisé pour comprendre de quoi est composée la matière et quelle est son organisation atomique. L'exploitation de ce faisceau synchrotron est réalisée par les lignes de lumière. Une ligne de lumière correspond à un laboratoire permettant la mise sous faisceau de différents types d'échantillons. La constitution classique d'une ligne de lumière étant une cabane optique, une cabane d'expérience et une station de travail.

Elle peut être représentée de la manière suivante (Figure 2):

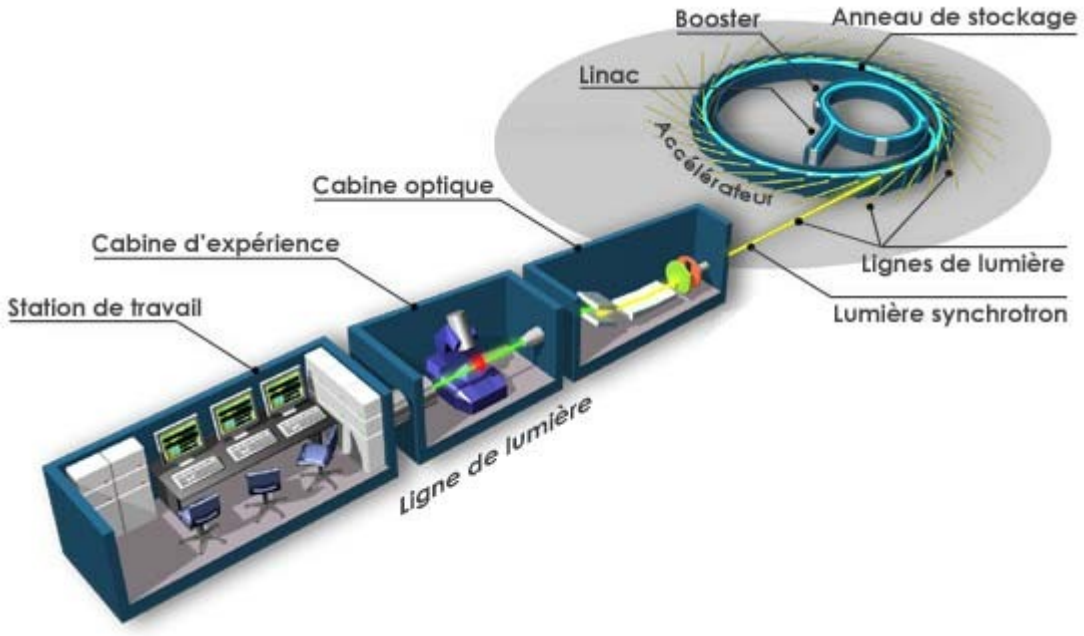


Figure 2 : Représentation schématique d'une ligne de lumière (*tiré de la banque d'images du synchrotron-soleil*)

Ces lignes de lumière sont construites dans le prolongement de chacune des sections droites de l'anneau. Chacune d'elle est indépendante, développant ses propres thématiques de recherche. Il existe en effet de nombreuses applications comme la chimie, la physique la médecine ou encore la biologie.

Il y existe plus de 50 centres de recherche de type synchrotron dans le monde (Figure 3). L'intérêt porté par la communauté scientifique pour ce type d'instrument est grandissante, il y a à ce jours plusieurs synchrotrons en cours de construction de par le monde (Brésil, Chine etc…).

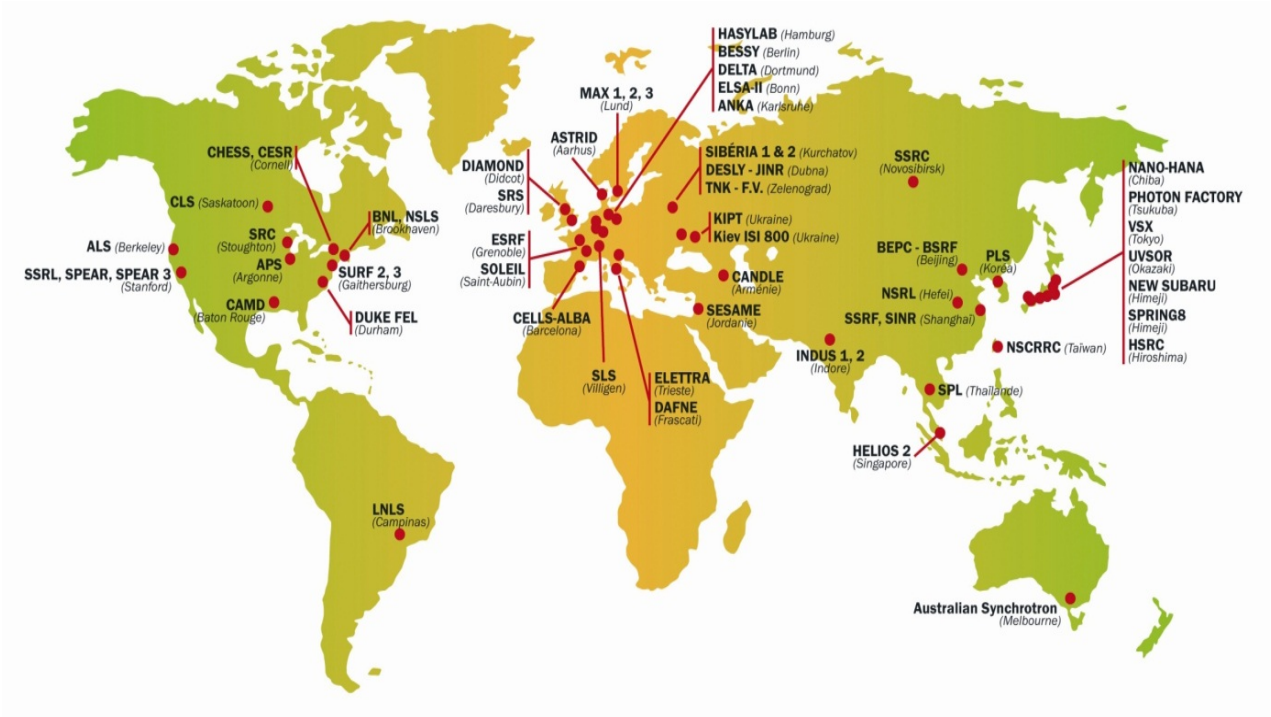


Figure 3 : Répartition des synchrotrons dans le monde (*tiré de la banque d'images du synchrotron-soleil*)

On note en effet que durant ces quinze dernières années, il a été construit quatre synchrotrons rien qu'en Europe : SOLEIL en France, DIAMOND en Angleterre, MAX-IV en Suède et Alba en l'Espagne.

Il s'est installé une véritable collaboration entre ces différents centres de recherche. Dans ce contexte d'échanges, le synchrotron SOLEIL a pu bénéficier de technologies développées dans d'autres centres. C'est notamment le cas du système de contrôle commande (appelé TANGO) utilisé dans la totalité du synchrotron (que ce soit sur les lignes de lumière ou par la Machine elle-même). Ce système a été à l'origine développé par l'ESRF (Synchrotron européen basé à Grenoble), et est aujourd'hui utilisé par la quasi-totalité des synchrotrons européens ainsi que de nombreux lasers de puissance.

I.1.2. Présentation du Synchrotron SOLEIL

SOLEIL est l'acronyme de « Source Optimisée de Lumière d'Energie Intermédiaire du LURE». LURE est en quelque sorte l'ancêtre de SOLEIL, pionnier dans le domaine du rayonnement synchrotron. Il a fermé ses portes en 2003, le projet SOLEIL ayant pris le relais.

Le synchrotron SOLEIL est un synchrotron de troisième génération qui est classé parmi les très grandes infrastructures de recherche françaises (Figure 4). Environ 500 Personnes y travaillent dont 357 permanents et des personnels temporaires (doctorants, post-doctorants, CDD, intervenants extérieurs, etc…). Par ailleurs SOLEIL accueille chaque année environs 4000 utilisateurs sur ses lignes de lumière lors des 5 périodes de faisceaux annuelles. Durant chacune de ces périodes de 8 à 12 semaines, les installations fonctionnent 24H sur 24H, 7j sur 7, le temps total de faisceau pour les lignes produit en 2017 étant de 5028 heures (soit une performance de 98,7%).

Figure 4 : Photo aérienne synchrotron soleil (*tiré de la banque d'images du synchrotron-soleil*)

Actuellement, SOLEIL est constitué de 29 lignes de lumière. Comme dit dans la présentation de la technologie synchrotron ci-dessus, ces lignes sont des laboratoires indépendants ayant chacune une thématique de recherche (Figure 5). Ces lignes vont en effet exploiter le faisceau synchrotron de différentes manières, grâce en particulier à un monochromateur situé dans leur cabane optique qui va sélectionner la longueur d'onde utilisée pour leur domaine d'application scientifique. L'ensemble du spectre disponible s'étend de l'infrarouge

(λ = 1,2 µm) au rayon X-durs (λ = 0,024 µm), cette large gamme spectrale permettant de couvrir un grand nombre de thématiques de recherche.

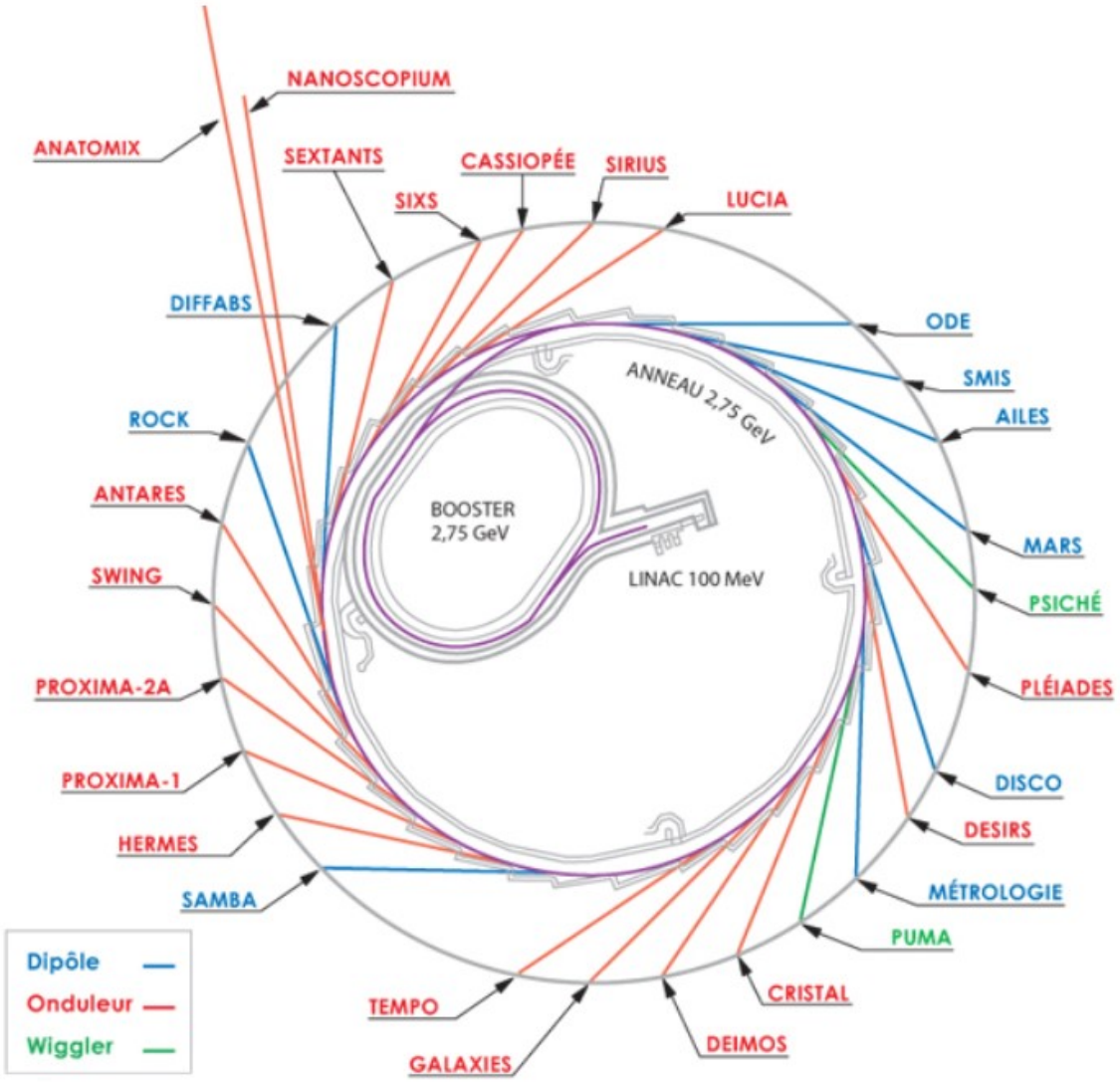


Figure 5 : Présentation des lignes de lumière à SOLEIL (*tiré de la banque d'images du synchrotron-soleil*)

On peut noter la spécificité des lignes « longues » NANOSCOPIUM et ANATOMIX, qui auront nécessité la réalisation d'une extension du bâtiment pour être construites. Nous présenterons NANOSCOPIUM plus en détail dans la partie I.3.1.1.

I.1.3. Organisation de SOLEIL

Dès les débuts de SOLEIL un ensemble de groupes supports techniques ont été créés (il en existe aujourd'hui 18), le but étant de mutualiser les moyens et compétences et de fournir des services techniques d'excellence aux scientifiques de ligne de lumière. Ils sont aujourd'hui organisés à travers deux divisions principales, la division Accélérateurs et Ingénierie et la division Expérience.

On retrouve notamment dans la division Accélérateurs et ingénierie :

- Le groupe **AM** (Alignement Métrologie) : Il réalise un alignement physique des éléments magnétiques (composants de l'accélérateur), des éléments optiques (composants des lignes de lumière) et fournit des études métrologiques de certains ensembles de haute précision.
- Le groupe **ICA** (Informatique de Contrôle Acquisition) : Le groupe ICA a pour mission de fournir aux lignes de lumière aux accélérateurs un ensemble cohérent de solutions logicielles répondant aux besoins de contrôle des équipements, de développement d'interfaces Homme-Machine, de développement d'applications de visualisation des données et d'automatisation des processus métiers.
- Le groupe **ECA** (Electronique de Contrôle et Acquisition) : Est en charge de l'ingénierie des systèmes électroniques et des logiciels embarqués nécessaires au contrôle-commande et à l'acquisition des accélérateurs et des lignes de lumière.

Dans la division Expérience :

- Le groupe Détecteurs : Le groupe détecteurs développe de nouveaux détecteurs en fonction des besoins des scientifiques des lignes de lumière, apporte conseils et assistance aux lignes de lumière dans le choix et l'achat de leur détecteurs, met en place et maintien un pool de détecteurs et d'électronique.
- Le groupe Optique : Il réalise les calculs et simulations optiques et développe et produit les optiques spécifiques, effectue la métrologie des surfaces optiques, conçoit et mettre en œuvre des ensembles optomécaniques.

Dans ce contexte j'occupe un poste transverse. Je suis en effet partagé à 50% pour le groupe AM et 50% pour le groupe ICA :

- **Activités au sein du groupe Alignement Métrologie**

Pour effectuer ces mesures physiques, le groupe utilise des instruments particuliers (hautement spécifiques), et a besoin d'applications informatiques particulières pour acquérir les données issues de ces instruments.

Mon rôle dans ce groupe est dans un premier temps de développer ces applications et dans un second temps d'effectuer des maintenances sur l'existant. La plupart de mes développements concernant les applications propres au groupe AM (applications Stand-Alone) ont été réalisés en Visual Basic.

- **Activités au sein du groupe ICA (Informatique de Contrôle Acquisition)**

Une des missions du groupe ICA est de fournir des systèmes de contrôle distribués basés sur le framework TANGO. TANGO peut être en première approche considéré comme un middleware assurant la communication entre des composants logiciels appelés « devices », chaque device ayant pour rôle de piloter un instrument ou d'encapsuler un processus métier.

Mon rôle au sein du groupe ICA est de développer ces devices TANGO, en priorité pour les instruments du groupe Alignement. Cependant en fonction des besoins à SOLEIL, je suis aussi amené à travailler dans un contexte plus large pour le compte des lignes de lumière ou d'autres groupes de supports. Dans ces différents cas, mon rôle est d'établir un cahier des charges, de proposer une architecture logicielle puis de réaliser la partie développement, les tests puis la mise en production. Ces développements dans le framework TANGO sont réalisés en C++.

I.2. Le système de contrôle commande à SOLEIL

Dans cette partie j'essayerai de décrire le fonctionnement du contrôle commande mis en place à SOLEIL à travers son architecture réseau spécifique. Comprendre par le terme « contrôle commande » le pilotage de tous les équipements de la machine (LINAC, Booster et Anneau de stockage) et des lignes de lumière ainsi que l'acquisition des données de l'ensemble des détecteurs.

I.2.1. L'architecture réseau

Comme nous l'avons vu dans la présentation du synchrotron, une ligne de lumière est un laboratoire indépendant et elle a donc besoin d'un système de contrôle à part entière qui lui est propre. De la même manière, la Machine est contrôlée comme une entité distincte. Au total, il faut donc être capable de gérer 30 systèmes de contrôle différents. De plus il faut pouvoir échanger des informations entre ces différents systèmes par exemple pour piloter de part et d'autre (Machine et Ligne) les éléments magnétiques à l'interface entre l'anneau et les lignes de lumière.

L'architecture réseau TCP/IP de SOLEIL est constituée des trois réseaux principaux :

- ReS : Réseau Site, réseau dédié à la fonction support:
    - Il sert aux laboratoires annexes aux lignes de lumière (labo : ECA, Alignement Métrologie, Magnétisme, Optique etc…).
    - Il sert aussi de base de développement et de tests, c'est par exemple via un serveur dédié au groupe ICA sur le ReS que les développeurs travaillent.
- RE : Réseau Expérience. Le réseau dans lequel se trouvent les différents sous réseaux correspondants à l'exploitation des lignes de lumière. Pour chaque ligne dans le réseau expérience sont déclarés deux sous réseaux :
    - Le REL : Le réseau expérience ligne. C'est via ce réseau que seront notamment exploitées les données issues des détecteurs, données accessible via le système de stockage centralisé de SOLEIL.
    - Le RCL : Le réseau de contrôle ligne. Sur ce réseau sont connectés l'ensemble des équipements serveurs, automates, frontaux d'acquisition, instruments de mesure et équipements propres à la ligne de lumière.
- RCM : Le réseau contrôle Machine. Sur ce réseau sont connectés l'ensemble des serveurs, automates, frontaux d'acquisition et équipements propre à la Machine.

Nous pouvons représenter l'infrastructure réseau à SOLEIL comme suit (Figure 6):

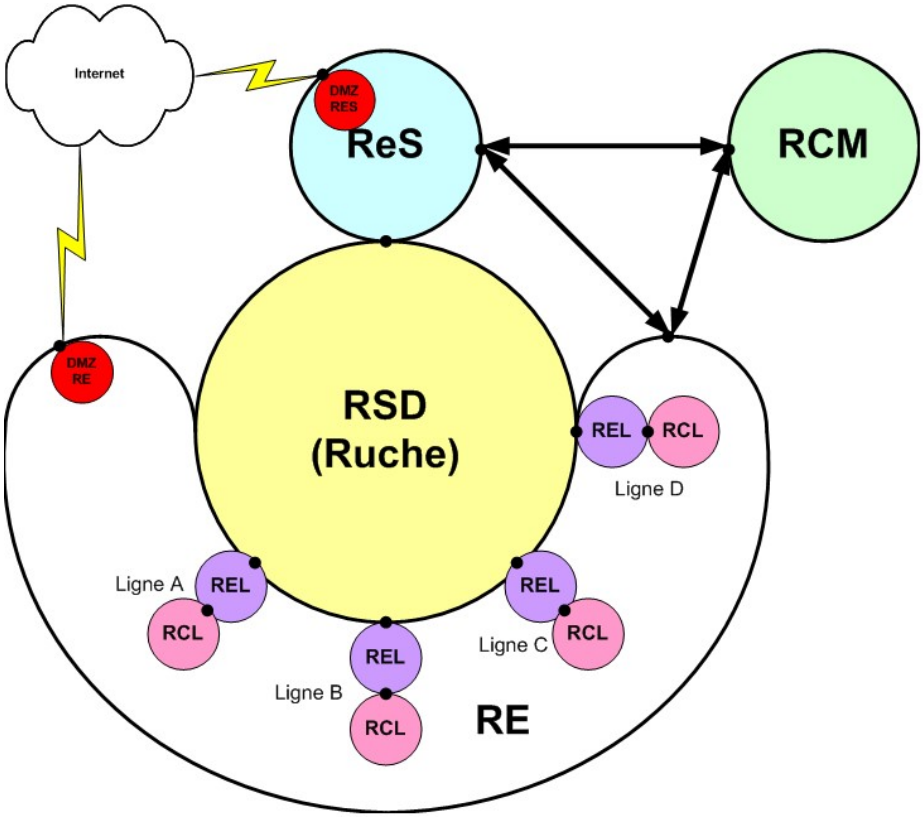


Figure 6 : Représentation de l'architecture réseau à SOLEIL (*tiré de la banque d'images du synchrotron-soleil*)

I.2.2. Architecture Logicielle

La principale problématique pour la mise en place d'un système de contrôle commande de cette envergure est de pouvoir faire communiquer un ensemble de sous-systèmes hétérogènes tant au niveau matériel que logiciel. En effet, si on prend l'exemple d'une ligne de lumière, elle est composée de systèmes industriels lents (comme des automates industriels), des systèmes d'acquisition rapide (des moniteurs de faisceau), des systèmes achetés clés en mains (le fournisseur fournit un système avec un contrôle commande spécifique). De plus, tous ces différents systèmes doivent pouvoir fonctionner sur des systèmes d'exploitation particuliers comme Linux ou Windows (ce qui nécessite de prendre en charge des pilotes spécifiques, en fonction du fournisseur).

L'objectif est en fait d'obtenir un tout cohérent à partir de ces matériels hétérogènes. Pour ce faire, le groupe ICA a décidé d'utiliser la technologie TANGO (TAco Next Generation Object), un système de contrôle commande orienté objet, basé sur la technologie CORBA (Common Object Request Broker Architecture), originellement développé par l'ESRF

(Synchrotron de Grenoble). Ce système TANGO est utilisé pour tous les systèmes de Contrôle/Commande de SOLEIL, que ce soit pour les lignes de lumière, pour la Machine ou pour les laboratoires annexes.

I.2.2.1. TANGO : Principe de fonctionnement

Le middleware CORBA :

CORBA est une norme d'implémentation de composants logiciels appelés ORB (Object Request Brocker). L'association de ces composants orientés objet va permettre la mise en œuvre d'un système complexe et partagé sur un réseau. L'objectif de cette approche est de masquer toutes les incompatibilités liées aux différences de langages de tous les composants pour un système donné. CORBA est maintenu par l'OMG (Object Management Group), un consortium Américain (composé de plusieurs entreprises comme : IBM, Sun ou Microsoft) qui s'est fixé pour rôle de standardiser le modèle objet dans sa globalité.

CORBA va pour fonctionner s'appuyer sur un langage spécifique : l'IDL (Interface Description Language). Tout ORB (objet sur le réseau) aura un IDL pour décrire son interface. Il faudra en fait, après avoir déclaré l'interface IDL d'un composant logiciel, se servir d'un compilateur pour générer un « Squelette » pour le serveur et un « Stub » pour le client, dans un langage donné (C++, Java, C etc…). C'est grâce au Stub (code généré automatiquement) que le client pourra faire des appels au serveur distant, le « Stub » permettra en effet de connaitre les fonctions distantes du serveur, et d'effectuer ces appels comme si l'objet distant était local. Ces appels arriveront par la suite sur le « Squelette » du serveur avant d'être propagé sur le serveur en question.

La communication entre ces objets est assurée via le protocole IIOP (Internet Inter ORB Protocole). Ce protocole va permettre la communication entre ORB sur un réseau en s'appuyant sur le protocole internet (IP).

Dans sa globalité, le système de communication entre un client et un serveur en utilisant la technologie CORBA peut être décrit comme sur la Figure 7:

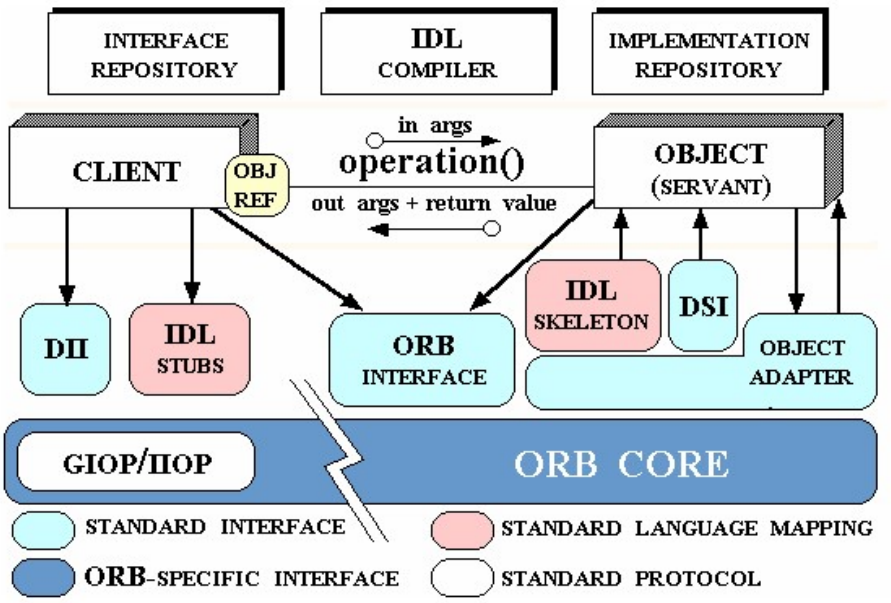


Figure 7 : CORBA ORB Architecture. Douglas C. Schmidt (2006).

Le retour d'expérience de système basé directement sur cette approche d'objets distribués montre qu'il est complexe à mettre en œuvre. Dans le but de simplifier le développement logiciel, que ce soit pour la partie spécifique IDL mais aussi pour standardiser les types de données échangés entre client et serveur, l'ESRF a développé l'outil TANGO (TAco Next Generation Object), en choisissant une implémentation de la norme CORBA (appelée OMNIORB) qui est gratuite (soumise à une licence GNU), robuste et compatible avec C++ et Python.

Le framework TANGO :

TANGO aura donc pour rôle de masquer la couche CORBA (Figure 8). Il évite aux développeurs de prendre en charge toute la complexité liée aux ORBs. Il aura aussi pour rôle de standardiser les types de données échangées entre objets.

Figure 8 : Imbrication de TANGO dans le système de contrôle (*tiré de la banque d'images du synchrotron-soleil)*

TANGO de l'acronyme : TAco Next Generation Object, est comme son nom l'indique, le successeur de TACO. TACO signifiant « Telescope and Accelerator Controlled with Objects».

La représentation logicielle de tout instrument/équipement dans le monde TANGO est appelée « device ». Tout instrument utilisable sur le bus TANGO (Figure 9) aura donc un développement particulier, un device.

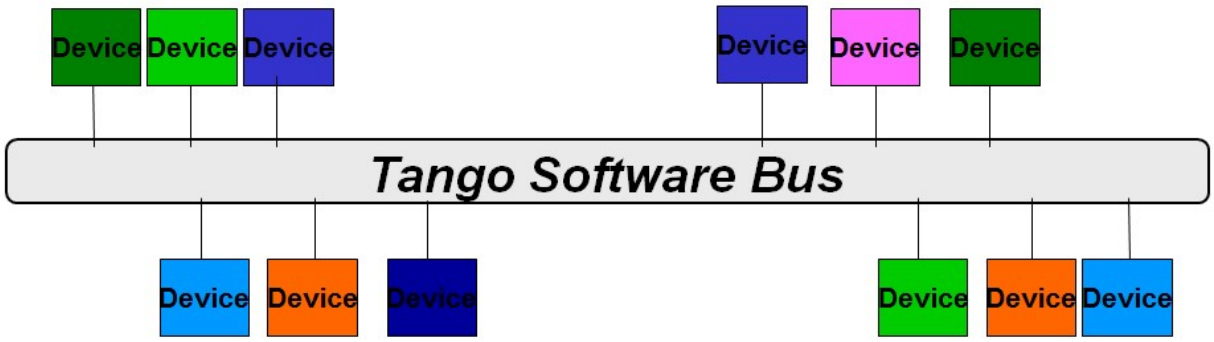


Figure 9 : Illustration du bus TANGO (*tiré de la banque d'images du synchrotron-soleil)*

A ce jour, il existe plus de 500 classes différentes de device dans la communauté. Aujourd'hui, TANGO est utilisé par la quasi-totalité des synchrotrons européens comme Desy (Hambourg, Allemagne), Elettra (Trieste, Italie), et comprend une communauté de développeurs de plus de 150 membres actifs. C'est aujourd'hui le rôle d'un comité formé de différents centres de recherche que de faire évoluer le cœur de TANGO.

I.2.2.2. Unité logicielle : Le Device server

Un device server est principalement caractérisé dans le monde TANGO par ses commandes et ses attributs voir Figure 10.

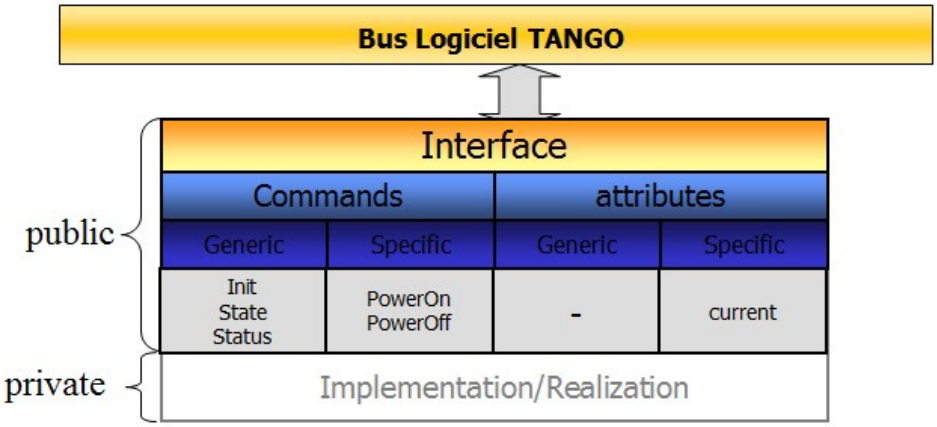


Figure 10 : Interface TANGO Device server (*tiré de la banque d'images du synchrotron-soleil)*

- **L'Attribut** : Grandeur physique, plusieurs types possibles : chiffre (exemple : une valeur de courant), chaine de caractères, image, graphique.
- **La Commande** : Correspond à une action à réaliser pour l'utilisateur (exemple : mettre sous tension une alimentation).

Il est aussi caractérisé par ses propriétés et ses états :

- **La Propriété** : Valeur fixée pour une instance[1] de device (par convention cette valeur n'est lue qu'au démarrage de ce dernier), elle est persistée en base de données. Surtout utilisée pour la phase de configuration (exemple : définir l'adresse IP de l'instrument sur lequel doit se connecter un device).
- **L'Etat** : Il existe un certain nombre d'états prédéfinis dans TANGO, voir Annexe 1. Ces états vont être déterminants dans la mise en œuvre du cycle de vie d'un device (exemple : OFF, ON, RUNNING). L'objectif pour le développeur est de décrire au mieux le système sous-jacent pour par exemple interdire certaines actions dans certains états.

Comme nous l'avons vu, un device correspond à une entité (unique) à contrôler. Mais il peut aussi être polymorphe, c'est-à-dire qu'il peut aussi bien représenter un matériel en particulier (comme une caméra) ou une collection de matériels (exemple : Un device

[1] En programmation objet, une instance correspond à une occurrence d'une classe donnée. Dans le monde Tango, une instance de device correspond à un exemplaire de device. (Exemple : Le device correspondant à un moteur pourra être instancié pour chaque moteur d'une ligne de lumière)

microscope est constitué d'un détecteur optique mais aussi d'un ensemble de moteurs afin de positionner un échantillon). Le device peut donc aussi être un agrégat de devices sous-jacents. La représentation d'un système complexe étant construite avec un système de briques logicielles indépendantes.

Notons qu'un device peut être à la fois un serveur et un client (on parle de device server par convention). En effet, dans le cadre d'un device « agrégat », ce device sera à la fois serveur pour les clients finaux, mais aussi client des devices sous-jacents. Il peut par exemple être nécessaire d'effectuer certains calculs dans des devices de haut niveau pour répondre à des problématiques particulières. Aussi, il existe un certain nombre d'applications de haut niveau utilisées à SOLEIL qui fonctionnent en tant qu'applications clientes de devices (Figure 11).

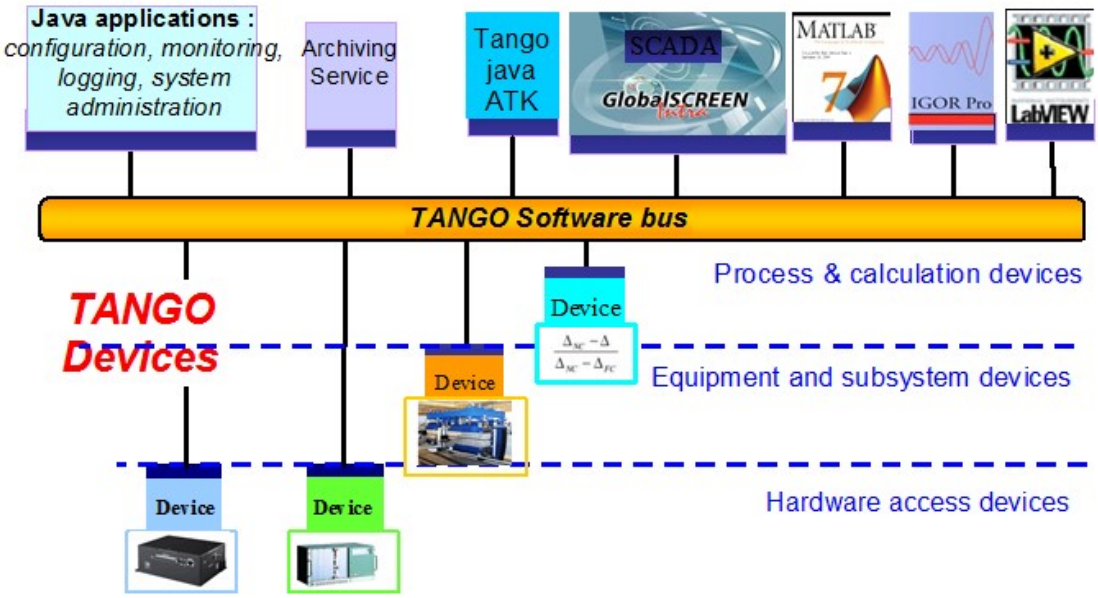


Figure 11 : Les différents types de clients TANGO (*tiré de la banque d'images du synchrotron-soleil)*

Pour permettre la mise en œuvre d'un device il faut au préalable l'avoir déclaré sur une base de données MySQL. C'est dans cette base de données qu'est identifiée chaque instance de device, elle est aussi utilisée pour persister les propriétés de ces derniers. Par convention sur une ligne de lumière, un device est déclaré en base via le modèle d'adresse suivant :

- Localisation / Type application / Id de l'instrument

Il existe plusieurs solutions pour déclarer ces devices, les démarrer, les configurer et les manipuler. On peut notamment citer les applications graphiques suivantes (applications JAVA basée sur la technologie SWING):

- **JIVE** : Permet de déclarer un device en base de données. Permet de définir ses propriétés et de lancer l'ATKPanel associé au device.
- **ATKPanel** (TANGO Application ToolKit) : Application générique permettant d'afficher les données issues d'un device de manière graphique. L'interface ATKPanel est dynamique, elle est construite en fonction des attributs et commandes présentes dans le device en question. A SOLEIL c'est la principale application utilisée pour piloter un device server.
- **ASTOR** : Permet de démarrer un device déclaré en base de données TANGO sur un serveur donné. Cette application permet aussi de visualiser quels devices sont démarrés ou arrêtés.

Pour nous rendre compte de l'envergure du système de contrôle commande à SOLEIL, nous pouvons donner ces quelques chiffres :

- Il existe en moyenne 700 devices par ligne de lumière. Rappelons qu'il existe 29 lignes de lumière à SOLEIL.
- Pour l'ensemble de la machine (LINAC, Booster et anneau de stockage) on dénombre aujourd'hui 9970 devices.

I.3. Introduction du projet

I.3.1. Un besoin formulé par NANOSCOPIUM

I.3.1.1. NANOSCOPIUM : une ligne particulière

Le besoin d'asservissement de la position faisceau à SOLEIL a été formulé par la ligne de lumière NANOSCOPIUM. C'est une ligne dite longue, c'est-à-dire qu'il y a 155 mètres entre la source et l'échantillon contre environs 35 mètres pour une ligne de lumière classique. Le but est de permettre de focaliser le faisceau à une taille de 30 nanomètres, alors que sur les autres lignes le diamètre de faisceau mesure quelques microns. Comme nous l'avons vu dans la présentation de SOLEIL, la fabrication de cette ligne a nécessité la construction d'une extension du bâtiment synchrotron. Cette ligne a donc des besoins particuliers, la recherche d'un positionnement au nanomètre près étant sa principale problématique.

I.3.1.2. Présentation du projet

Dans le contexte particulier de NANOSCOPIUM, il faut pouvoir asservir (maintenir à une valeur de consigne) la position spatiale du faisceau provenant de la Machine sur l'axe horizontal et vertical (X et Y). Cet asservissement doit être effectif sur une longue durée, il faut éviter les dérives longues, le faisceau doit en effet être stable pendant des expériences pouvant atteindre les huit heures.

Nous rappelons que le système d'acquisition (basé sur des devices TANGO) des lignes de lumière permet de contrôler tout type de moteur et d'acquérir des informations concernant le faisceau (comme une image, une tension etc...). Le besoin formulé par NANOSCOPIUM est en fait de pouvoir synchroniser la lecture d'information sur le faisceau, ici une image, avec son positionnement. Ce positionnement sera effectué grâce à deux axes de rotation capables d'orienter un miroir, miroir sur lequel le faisceau est reflété. Ces rotations sont contrôlées grâce à des moteurs de type « GalilAxis », ils sont interfacés sur TANGO comme n'importe quel autre actionneur à SOLEIL, via un device par axe de rotation. Ils sont contrôlables principalement via leur attribut « *position* », un attribut de type double, accessible en lecture et écriture. Chacun d'entre eux va permettre la rotation du miroir (ici, le miroir M2 de la ligne NANOSCOPIUM comme on peut le voir sur la Figure 12) suivant deux

axes, l'axe RS et RZ (RZ ayant une influence sur la position X du faisceau et RS sur la position Y).

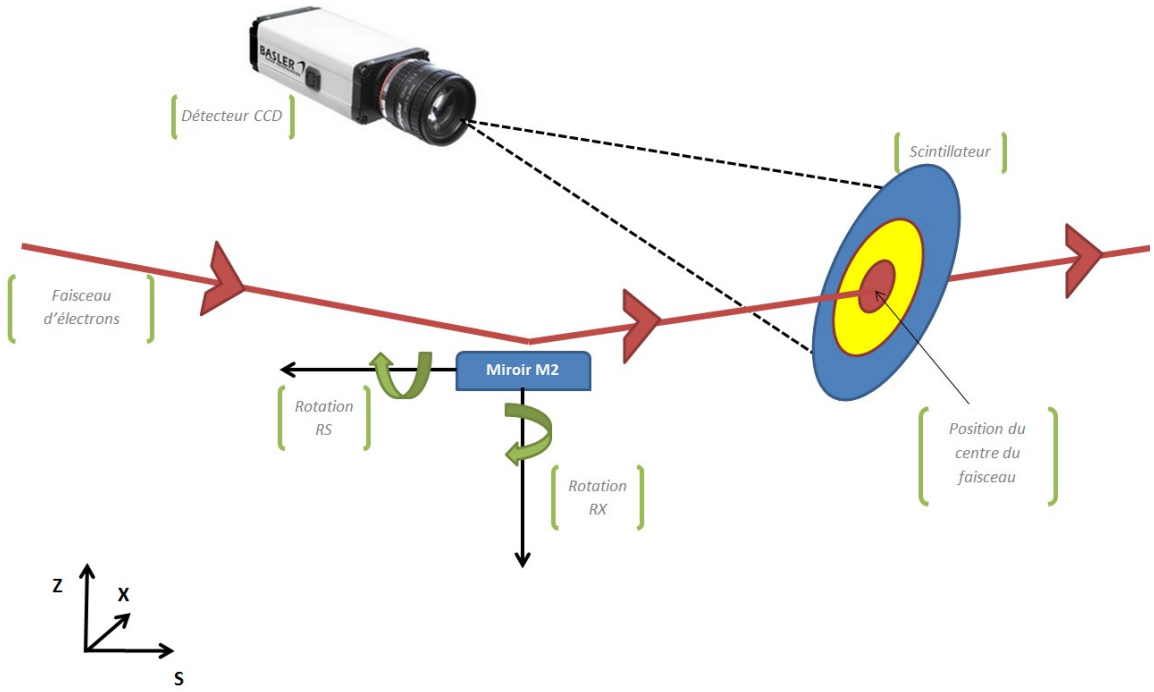


Figure 12 : Schéma d'intégration du systéme de monitoring de la position faisceau sur la ligne NANOSCOPIUM (vue du dessus)

Pour permettre de voir la position du faisceau sans pour autant le couper, la ligne NANOSCOPIUM a mis en place un scintillateur. En interagissant avec ce scintillateur, le faisceau d'électrons émettra une lumière dans le visible (d'une longueur d'onde comprise entre 390 et 710 nanomètres), il est alors possible de le voir et d'acquérir sa position grâce à une caméra, voir Figure 12. On précise que le scintillateur laisse passer la majorité de l'énergie faisceau, il n'est donc pas bloquant et peut être utilisé en parallèle d'une expérience placée en aval sur la ligne de lumière.

Jusqu'à présent, la ligne utilisait une solution « temporaire » (un script Matlab spécifique écrit par les scientifiques) pour asservir la position du faisceau, mais il existe plusieurs limites à ce développement :

- L'asservissement ne peut pas être contrôlé depuis le système de contrôle TANGO
    - Le système ne peut pas être accédé depuis le réseau contrôle de ligne (RCL)

    - Il ne peut pas être synchronisé avec d'autres éléments de la ligne
- Il ne présente pas d'interface homme-machine
    - Les utilisateurs externes ne peuvent pas s'en servir en autonomie sur la ligne
- Il n'est pas générique :
    - Le type d'actuateur mis en jeu ne peut pas être changé
    - Il ne peut fonctionner qu'avec un seul type de capteur : un détecteur 2D (une caméra)
- Il ne peut pas être configuré :
    - Les cibles d'asservissement (positions X et Y en pixels) sont hard-codées
    - Les devices concernés ne sont pas modifiables (choix du détecteur et des actionneurs)
    - L'asservissement n'est pas paramétrable

C'est pourquoi la ligne a émis le besoin d'une application générique qui lève ses différentes contraintes, c'est-à-dire une application qui puisse s'utiliser avec différents type d'acteurs (différents types de moteurs, de capteurs) à condition qu'ils soient interfacés au monde TANGO. Ensuite, cette application doit pouvoir asservir la position faisceau suivant un correcteur de type PID (Proportionnel, Intégrale, Dérivé). L'utilisateur devra être en mesure d'ajuster le correcteur en fonction d'un système donné. Pour finir, la ligne exprime le besoin d'utiliser une IHM (interface homme-machine) spécifique, permettant de visualiser le faisceau, la cible que l'utilisateur aura définit et les différentes contraintes ou informations nécessaires.

Kadda Medjoubi sera notre contact sur cette ligne, ce sera donc avec lui que nous échangerons pour fixer le cahier des charges de cette application.

I.3.2. Motivations du projet

Ce projet est innovant dans le sens ou aucune autre application fonctionnant via TANGO ne permet d'asservir la position du faisceau sur une ligne de lumière à SOLEIL. L'idée est de nous appuyer sur le retour d'expérience de la ligne NANOSCOPIUM et sa connaissance

métier dans le domaine de l'asservissement afin de pouvoir proposer cette nouvelle fonctionnalité à l'ensemble de la communauté scientifique du monde Synchrotron.

Nous devons aussi pouvoir échanger avec un maximum de personnes à SOLEIL potentiellement intéressées par ce type de développement afin de recueillir leur remarque pour satisfaire le plus de cas d'utilisation possible.

I.3.2.1. La note d'opportunité et d'organisation

Un des moyens mis à notre disposition pour lancer le projet est d'écrire une note pour la Réunion de direction (note RD) afin que la direction puisse avoir connaissance du projet et qu'elle donne son accord pour sa réalisation.

Il s'agit aussi de mettre la ligne NANOSCOPIUM en avant dans ce projet. C'est grâce au temps faisceau alloué par cette ligne que nous pourrons mener à bien nos tests et valider l'application. C'est aussi à la demande de cette ligne que le projet est né, elle aura donc un rôle moteur.

Dans ce document il nous faut définir notre projet dans son ensemble. Nous rappelons donc son contexte, les livrables envisagés du projet, ses différentes phases ainsi que son organisation. Voir en Annexe 2 la note RD.

C'est à l'issu de la validation de cette note RD (le 30/01/18) que nous pouvons officiellement démarrer le projet.

I.3.2.2. Les réunions spéciales lignes de lumière

Une possibilité pour communiquer à l'ensemble des lignes de lumière sur ce projet a été de le présenter en réunion de **Suivi Informatique Lignes**. C'est à travers cette réunion que le groupe ICA partage aux lignes les dernières avancées et quelle roadmap sera adoptée. C'est aussi pendant cette réunion que les lignes donnent leur retour sur les applications utilisées.

J'ai donc préparé une présentation d'environ quinze minutes pour communiquer sur les objectifs visés par ce projet dans sa version initiale pour répondre aux besoins de

NANOSCOPIUM. Cette présentation se devait d'être courte et claire pour intéresser le plus grand nombre. J'ai pu bénéficier d'une session de présentation en « suivi informatique lignes » pour présenter ce projet le 28 Février 2018. Cette présentation avait pour principal objectif de décrire le principe de fonctionnement visé et l'objectif souhaité par NANOSCOPIUM.

Pendant cette réunion, des lignes de lumière ont manifesté leur intérêt pour cette application. C'est notamment le cas des lignes SIRIUS, GALAXIES et DISCO. Ces lignes ont fait apparaître le besoin d'une utilisation légèrement différente de celle envisagée sur NANOSCOPIUM. En effet pour SIRIUS et GALAXIES, au lieu d'utiliser une caméra, les lignes envisageraient une détection via XBPM (voir partie I.4.1 sur les moyens de détection).

I.4. Etude préliminaire

Afin d'orienter notre projet, j'effectue une étude préliminaire permettant de définir les contours du projet et d'orienter le développement. Pour ce faire j'organise plusieurs réunions avec le client et les experts métiers concernés.

I.4.1. Plusieurs moyens de détection envisagés

Comme nous l'avons évoqué, il existe plusieurs types de capteurs pour détecter une position faisceau. J'ai organisé une réunion avec l'un des responsables des capteurs à SOLEIL : Kewin Desjardins pour avoir un premier retour concernant les solutions envisageables pour la détermination de centroïde faisceau[2]. Nous identifions deux principaux types de détecteurs, l'imageur (c'est-à-dire une caméra, déjà utilisé par la ligne NANOSCOPIUM) et le XBPM (X-ray Beam Position Monitor).

Imageur :

La première solution envisagée pour la détection du faisceau et le calcul de son centroïde est de nous baser sur un détecteur 2D. Il existe à SOLEIL environs une vingtaine de modèles de détecteurs 2D, basés sur différentes technologies : CCD (Charged Coupled Device), CMOS et

[2] Le centre d'intensité du faisceau. Correspond au barycentre pour une forme géométrique.

Pixels. Ces capteurs peuvent être utilisés pour effectuer du monitoring ou directement pour l'expérience (observation d'échantillons sous faisceau) sur une ligne de lumière. Dans notre cas, nous nous intéresserons aux capteurs permettant le monitoring du faisceau (étude qualitative en amont de l'expérience). On note que comme pour tout autre détecteurs 2D utilisé sur une ligne de lumière, chacun d'entre eux est interfacé au monde TANGO via un device générique appelé LIMA, la donnée principale remontée par ce device est une image. Voir partie II.3.2.3, pour plus de précisions sur le principe de fonctionnement de ce device. La principale complexité dans la mise en œuvre de ces capteurs dans notre application sera d'être en mesure de déterminer le centroïde faisceau, il nous faudra donc passer par une étape de traitement d'image.

XBPM (X-ray Beam Position Monitor) :

Comme décrit d'après Kewin Desjardins, Michel Bordessoule et Michal Pomorski (2018), ces capteurs sont capables de mesurer l'intensité du faisceau d'électrons et ce suivant deux axes (Y et X sur la Figure 13).

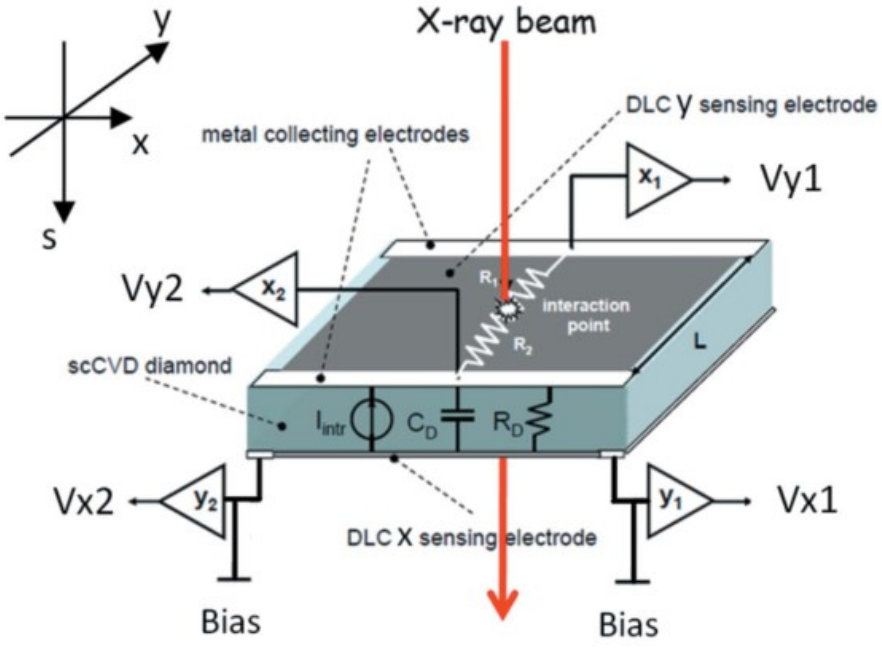


Figure 13 : Principe de fonctionnement d'un capteur XBPM. Kewin Desjardins, Michel Bordessoule et Michal Pomorski (2018).

Comme on peut le voir sur la Figure 13, le faisceau interagit avec la surface résistive du capteur, surface elle-même soumise à une certaine tension. Cette interaction va produire un courant qui sera relevé en quatre points : X1, X2, Y1, Y2.

Pour déterminer le centroïde (ou barycentre) du faisceau, on nous donne la formule suivante :

$$X = Kx * \frac{Ix1 - Ix2}{Ix1 + Ix2} \qquad Y = Ky * \frac{Iy1 - Iy2}{Iy1 + Iy2}$$

On note aussi dans la réunion que le device actuellement développé pour interfacer ce type de capteurs est d'ores et déjà capable de retourner ce centroïde faisceau. Il serait donc relativement rapide d'utiliser ce capteur à des fins d'asservissement dans notre application.

I.4.2. Problématique des coefficients PID

Rappelons que le principe d'une régulation est d'amener un système à une position voulue, et de le maintenir à cette position. Il s'agit donc de minimiser l'erreur entre la position de consigne et la valeur en sortie de notre système (schématiquement représenté Figure 14) :

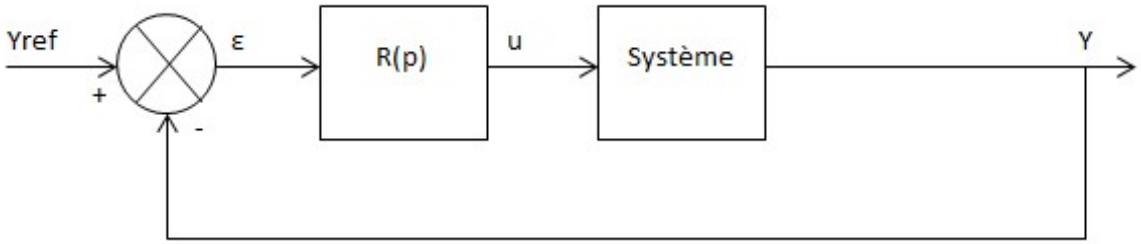


Figure 14 : Principe d'un asservissement

Nous distinguons en fait deux types de correcteurs pour effectuer cet asservissement, un correcteur linéaire dit simple, ou un correcteur PID (Proportionnel Intégral Dérivé). Après une première discussion avec la ligne NANOSCOPIUM, nous comprenons que dans certains cas, une correction linéaire simple n'est pas suffisante, il faut pouvoir mettre en place un correcteur PID.

- Correction Linéaire simple : Une correction mettant en jeu un ratio unique. Dans ce scenario on considère que pour un axe donné, il existe un ratio entre un nombre de pixels (pour une détection 2D) et un positionnement moteur.
- Correction PID (Proportionnel, Intégrale et Dérivé) : C'est un correcteur qui va permettre d'améliorer la performance de l'asservissement : temps d'approche, erreur statique, dépassement de consigne, voir la partie II.3.4 pour son principe de fonctionnement.

La principale problématique pour la mise en place d'un asservissement de type PID est la détermination des paramètres (des coefficients P, I et D). En parallèle à ce projet, il parait donc nécessaire de s'intéresser à une solution permettant la détermination de ces coefficients de manière automatique pour un système donné. On note que cette problématique ne fera pas partie de notre projet, comme exprimé dans la note RD (Annexe 2). Nous devrons néanmoins implémenter l'algorithme correspondant à ce correcteur dans notre application et permettre son réglage par l'utilisateur. Ainsi, il sera possible pour l'utilisateur de choisir ou non d'utiliser ce type de correcteur.

I.4.3. Contours du projet

Nous proposons donc de concevoir une application pouvant fonctionner avec l'ensemble des développements existants sur TANGO utilisés à SOLEIL et pouvant jouer un rôle dans l'asservissement faisceau. Pour ce faire, le Back-End (partie fonctionnelle contenant l'ensemble de la logique-métier) de cette application sera constitué de devices TANGO et de librairies fonctionnants en collaboration et pouvant s'interfacer avec la majorité des devices utiles dans l'asservissement du faisceau et ce, sur n'importe quelle ligne de lumière.

Orientation choisie pour une version initiale : Pour permettre le test et la validation du principe applicatif sur la ligne NANOSCOPIUM qui sera ici notre client, nous orientons notre développement en priorisant l'interfaçage des capteurs et actuateurs comme suit, ce développement initial fera l'objet d'un premier livrable :

- Notre développement initial sera basé sur des capteurs 2D. Le calcul du centroïde du faisceau sera donc fait à partir d'une image remontée d'un autre device server, un device LIMA (voir partie II.3.2.3).
- Les rotations étant mises en jeu sur cette ligne de lumière étant de type GalilAxis, nous commencerons par interfacer les devices correspondants.

Dans un second temps, nous interfacerons les capteurs de type XBPM. Progressivement et en fonction des besoins, nous intégrerons différents types d'actionneur.

Le Front-End (partie présentation de l'application) sera une IHM qui permettra le contrôle des devices sous-jacents de manière claire et n'affichant que les informations utiles lors d'un positionnement et d'un asservissement de la position faisceau.

Nous choisissons de nommer cette application **BeamPositionTracking**.

II. Réalisation de BeamPositionTracking

II.1. Mise en place du projet

II.1.1. Mise en place des responsabilités

Pour déterminer les différentes responsabilités dans ce projet, nous choisissons de scinder ce dernier en 5 grandes phases :

- **Etude de faisabilité** : comprend l'étude de l'existant, une orientation générale du projet.
- **Réalisation d'une version initiale de la partie fonctionnelle :** Réalisation du back-end permettant d'effectuer de premières simulations, permettra de s'assurer de la conformité du développement.
- **Réalisation d'une version initiale de l'IHM** : A l'issue de cette phase, une première version du front-end permettra d'échanger avec le client sur ses attentes initiales.
- **Itérations sur les parties fonctionnelles et graphiques** : Phase permettant de tester l'ensemble sur la ligne de lumière NANOSCOPIUM, ajuster au fur et à mesure le développement en fonction des remarques utilisateur et éventuels bugs.

- **Activités de transitions vers l'opération :** Phase comprenant le packaging de l'application, l'édition de notice utilisateurs, de formation, de communication autour du projet.

En accord avec mes responsables, nous fixons les responsabilités du projet suivant une matrice RAM (Responsability Assignment Matrix). Le but étant de déterminer le rôle de chaque acteur dans ce projet et ce pour chaque étape.

Pour chaque activité nous identifions les rôles de la manière suivante (Tableau 1):

- R : Responsible : Celui qui réalise l'action
- A : Accountable : Celui qui est responsable et qui doit rendre des comptes sur l'avancement de l'action
- C : Consulted : Personne qui peut être consultée pour l'activité
- I : Informed : Personne devant être informée

Tableau 1 : Matrice des responsabilités du projet BeamPositionTracking

RACI	Etude de faisabilité	Réalisation d'une version initiale de la partie fonctionnelle	Réalisation d'une version initiale de la partie IHM	Itérations sur les parties fonctionnelles et graphiques	Activités de transitions vers l'Opération
Réalisateurs	Falilou Thiam	Falilou Thiam	Falilou Thiam	Falilou Thiam	Falilou Thiam
Responsable	Falilou Thiam	Falilou Thiam	Falilou Thiam	Falilou Thiam	Falilou Thiam
Consultés	Alain Buteau, Florent Langlois, Kadda Medjoubi, APOLLON, THALES	Florent Langlois, Kadda Medjoubi	Florent Langlois, experts IHM ICA	NANOSCOPIUM, Florent Langlois	Alain Buteau, Florent Langlois
Informés	Florent Langlois	Alain Buteau	Alain Buteau, Kadda Medjoubi	Alain Buteau	Niveau 1, Niveau 2

II.1.2. Détermination de la méthodologie de gestion projet

Etant nommé responsable et réalisateur de l'ensemble des phases de ce projet, il est donc à ma charge de déterminer quelle méthodologie de gestion de projet sera utilisée pour le développement de cette application.

Mon choix s'est naturellement porté vers une approche Agile pour ce projet. En effet, il me faudra tenir compte au mieux des remarques de mon client afin d'améliorer sa satisfaction et donc l'impliquer dans chaque phase du projet.

Je me suis donc intéressé aux différentes méthodes Agiles pouvant s'appliquer au développement d'un projet informatique :

- Rapid Application Development (RAD, 1991)
- Dynamic systems development method (DSDM, 1995)
- Scrum (1996)
- Extreme programming (XP, 1999)
- Adaptive software development (ASD, 2000)
- Développement basé sur les fonctionnalités (FDD pour Feature Driven Development, 2003)
- Behavior-driven development (BDD, 2003)
- Crystal clear (2004)

Pour conduire à bien ce projet, j'ai retenu la méthodologie **DSDM** (Dynamic Systems Development Method). Je considère en effet que c'est la méthode la plus à même de favoriser la bonne conduite de ce projet et ce sur plusieurs plans (Figure 15):

- Implication des clients durant l'ensemble du cycle de développement
- Mise en avant de l'autonomie de l'équipe en charge du projet
- Livraisons régulières de l'application pour permettre un feedback permanent
- Favoriser l'adéquation entre le besoin métier et l'application
- Une approche itérative de développement basé sur le retour du client
- Laisser possible la modification du produit dans le cas de modification du besoin
- Détermination d'un patron directeur permettant la visualisation des grandes lignes du projet
- Faire des tests pour chaque étape de développement, valider chaque partie de l'application tout au long de son développement
- Faire coopérer les acteurs du projet, s'adapter aux diverses modifications de fonctionnalités

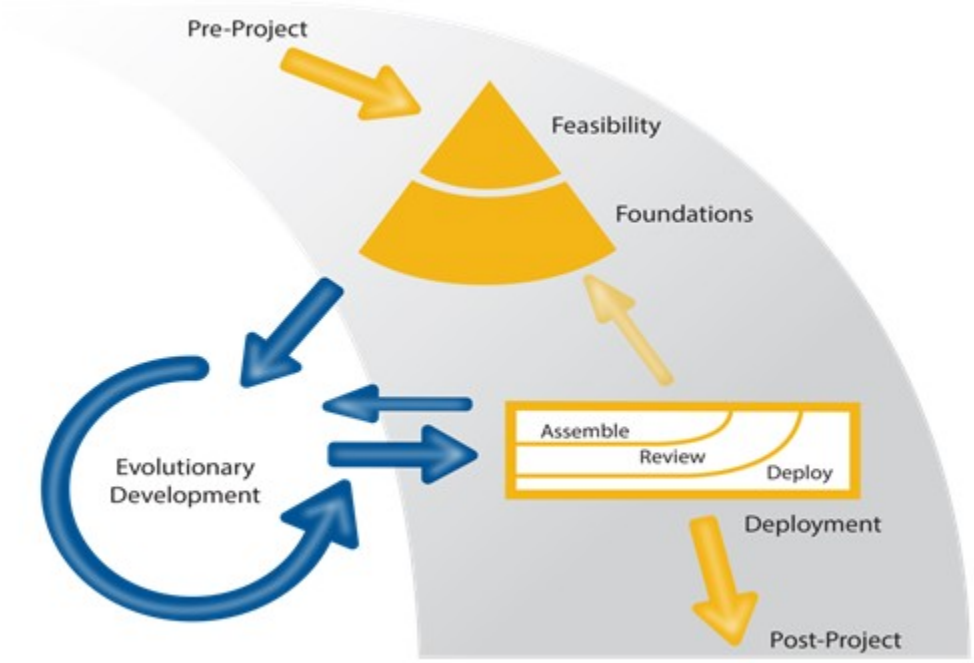


Figure 15 : Cycle de vie d'un projet DSDM (image provenant du site Agile Business)

Avec cette méthode de gestion, j'intègre l'étude de faisabilité, ainsi que l'analyse fonctionnelle dans ma démarche agile.

En me basant sur la méthodologie DSDM, je détermine le cycle de vie de ce projet de la manière suivante, en trois principales étapes :

- **Phase I : Pre-Project**
  - Identification/clarification du projet
  - Accord de financement
  - Go/NoGo
- **Phase II : Project life cycle**
  - Etude de faisabilité
  - Etude métier
  - Modèle fonctionnel
  - Conception et réalisation
  - Mise en œuvre
- **Phase III : Post-Project**

- Documentation et déploiement en production
- S'assurer que le système est opérationnel
- Effectuer sa maintenance

Je précise que la Phase I (pré projet) a déjà été réalisée à ce stade, nous avons en effet identifié le projet ainsi que son contour. L'accord de financement nous a été donné par la direction via notre note d'opportunité et mes supérieurs m'ont donné leur feu vert pour démarrer ce projet.

Le fait de s'appuyer sur cette méthodologie dans ce projet aura pour principal intérêt d'intégrer mon client dans toutes les étapes de développement et d'adapter le produit en fonction des tests conduits sur NANOSCOPIUM ainsi que du retour des utilisateurs.

II.1.3. Choix des outils de gestion de projet

Le choix des différents outils de gestion projet utilisés pour ce développement a été fait en accord avec les outils utilisés couramment à SOLEIL. Aussi, je m'assure d'utiliser des outils en accord avec ma méthodologie de gestion de projet Agile :

Git : Logiciel de gestion de version de code, à travers le service GitHub. GitHub est en fait un service permettant d'héberger le code source d'une application. Une fois le projet BeamPositionTracking créé sur la plateforme Git, chaque modification sur ce dernier fera l'objet d'un « *Commit »* qui portera un descriptif du changement. Le but est de pouvoir conserver une trace de toute modification sur le projet, tout au long de son développement pour comprendre comment il a évolué et pourquoi certaines modifications ont étés réalisées.

Jira : Système de gestion et suivi projet développé par la société Atlassian. Cet outil est aujourd'hui utilisé par de nombreuses équipes à SOLEIL. C'est un outil de gestion Agile qui permet des échanges en temps réel entre toutes les parties prenantes d'un projet donné. Le principe de fonctionnement de JIRA est l'utilisation de tickets. Par exemple pour chaque demande de développement, un ticket est créé. Dans chaque ticket il est possible d'interagir avec l'équipe de développement, le client, le responsable projet à travers des commentaires,

d'échanger des fichiers. Pour un ticket donné, il peut y avoir un certain nombre d'association d'autres tickets.

A SOLEIL, il existe plusieurs types de tickets (et de workflows de traitement associés) définis en amont et principalement par l'équipe ICA, par exemple les suivants :

- Type : TANGO-Device
    - Ticket servant à décrire un nouveau développement ou un changement dans un device courant.
    - Ce ticket peut avoir lieu dans le cadre d'une correction, d'une amélioration, ou d'une nouvelle fonctionnalité.
- Type : Problème
    - Suite à un incident, si un problème logiciel est constaté.
    - La personne qui remarque ce problème créé ce ticket en le décrivant au mieux.
    - Si il nécessite un changement alors un autre ticket de type TANGO-Device sera créé et lié à ce ticket problème.
- Type : Suivi-Equipe
    - Ticket permettant de résumer une réunion.
    - Permet notamment de continuer à échanger sur un sujet donné en fonction des disponibilités de chacun.

Ces différents cycles de vie ont étés déterminés pour standardiser la gestion de projet, le suivi d'incidents de problème etc… Je peux m'appuyer sur cette plateforme pour échanger plus simplement avec mon client et interagir avec toute personne pouvant apporter son expertise sur le projet.

Gant-Project : Afin de pouvoir organiser mes tâches et produire un planning pour communiquer au client, je choisi d'utiliser le logiciel libre GanttProject. Une fois le projet découpé en plusieurs phases, je pourrai me servir de la représentation de Gantt pour établir un planning prévisionnel.

Visual Paradigme : Je peux m'appuyer sur le formalisme UML pour représenter le système au fur et à mesure de son développement. J'ai décidé d'utiliser l'outil Visual Paradigme, il permet de créer de manière simple n'importe quel type de diagramme UML et de facilement

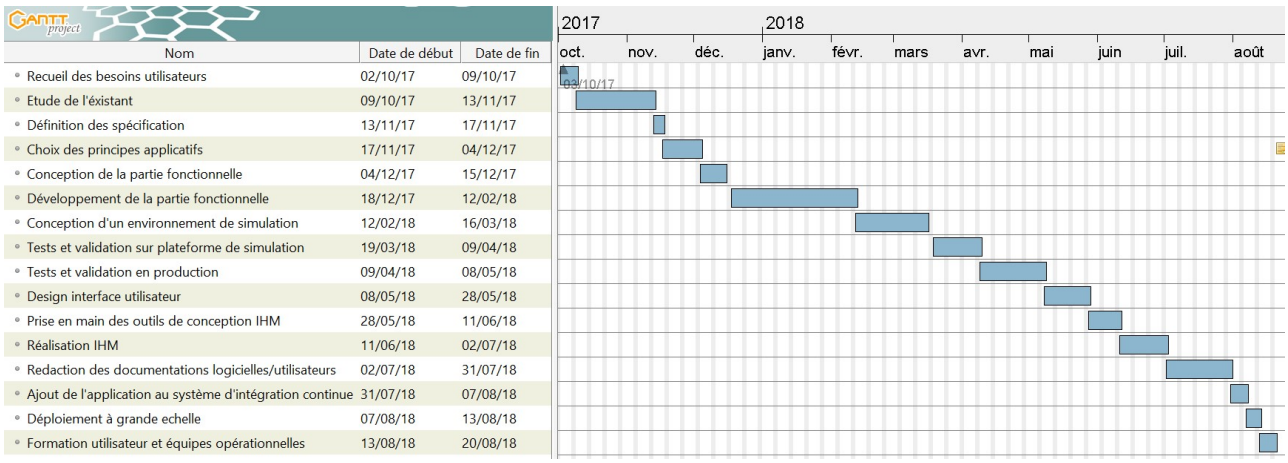
les adapter au fur et à mesure de l'évolution des demandes. Je peux ainsi commencer par établir avec mon client les différents cas d'utilisation dans le cadre l'édition du cahier des charges.

II.1.4. Définition d'un planning

Toujours d'après l'approche processus décrite dans la méthodologie DSDM, je découpe le projet en différentes étapes :

- Etude de l'existant
- Choix des technologies à employer
- Conception de l'architecture logicielle
- Développement de la partie fonctionnelle
- Création d'une plateforme de tests locale
- Validation de la partie fonctionnelle sur plateforme de tests
- Tests et validation sur ligne NANOSCOPIUM
- Prise en main des outils de conception graphique utilisés à SOLEIL
- Design et Réalisation de l'IHM
- Rédaction de la documentation utilisateur
- Packaging logiciel, déploiement global à SOLEIL

Etant en autonomie sur ce projet, ce diagramme (Figure 16) n'est composé que d'une seule ressource :



Nom	Date de début	Date de fin
Recueil des besoins utilisateurs	02/10/17	09/10/17
Etude de l'éxistant	09/10/17	13/11/17
Définition des spécification	13/11/17	17/11/17
Choix des principes applicatifs	17/11/17	04/12/17
Conception de la partie fonctionnelle	04/12/17	15/12/17
Développement de la partie fonctionnelle	18/12/17	12/02/18
Conception d'un environnement de simulation	12/02/18	16/03/18
Tests et validation sur plateforme de simulation	19/03/18	09/04/18
Tests et validation en production	09/04/18	08/05/18
Design interface utilisateur	08/05/18	28/05/18
Prise en main des outils de conception IHM	28/05/18	11/06/18
Réalisation IHM	11/06/18	02/07/18
Redaction des documentations logicielles/utilisateurs	02/07/18	31/07/18
Ajout de l'application au système d'intégration continue	31/07/18	07/08/18
Déploiement à grande echelle	07/08/18	13/08/18
Formation utilisateur et équipes opérationnelles	13/08/18	20/08/18

Figure 16 : Planning GANT pour le projet BeamPositionTracking

II.2. Lancement du projet

II.2.1. Une étude de l'existant

J'entame ici la phase II de la méthodologie DSDM, c'est-à-dire le cycle de vie du projet. Une première étude de l'existant m'est nécessaire afin d'orienter ce projet, c'est la première phase de réalisation. Pour ce faire, je m'appuie sur nos contacts dans le monde TANGO ayant travaillé sur des projets s'approchants de notre besoin. Avec l'aide d'Alain Buteau, nous avons trouvé un développement pour un besoin similaire réalisé au Laboratoire d'Utilisations des Lasers Intenses (LULI) et chez THALES Services, avec qui nous échangeons régulièrement sur des problématiques communes.

Nous avons donc invité les membres des équipes du LULI et de THALES à SOLEIL pour qu'ils puissent nous présenter leur développement et ainsi bénéficier de leur retour d'expérience.

II.2.1.1. Le projet APOLLON – équipe LULI

Le projet APOLLON correspond à la mise en œuvre d'un laser de puissance sur le plateau de Saclay, plus précisément au Centre interdisciplinaire de lumière extrême (CILEX). Ce laser a été initié en 2007 et est en cours de construction, il devra atteindre une puissance de 10 PétaWatts afin d'interagir avec la matière à très haute densité.

L'enjeu pour ce projet est donc de pouvoir positionner le faisceau émis par le laser sur une position fixe, une cible. Ce projet a été développé en Python, et n'a pour le moment pas pu être mis en œuvre en production.

Le principe de fonctionnement est de déterminer le centroïde du faisceau à partir d'une image et de calculer un écart entre consigne (cible entrée par le laseriste[3]) et position courante du faisceau. Ensuite, en fonction d'un ratio entre pixels et pas moteur (entré par l'utilisateur), il s'agit de calculer la nouvelle position pour chaque axe (X et Y). Aucun correcteur n'est mis en œuvre, l'asservissement se faisant grâce à un simple calcul de différence.

[3] Un spécialiste laser, il a en charge son encadrement et son optimisation.

Pour déterminer le centroïde du faisceau (la position suivant l'axe x et y), l'algorithme effectue un seuil sur l'image afin de retirer le bruit, j'appellerai ce seuil « threshold » dans la suite de cette étude. D'après leur approche, ce threshold est fixe (13%), il est défini par les experts métiers c'est-à-dire les laseristes.

On note aussi que ce développement n'est pas fait pour s'adapter à d'autres installations que celle d'APOLLON. Il ne peut être utilisé qu'avec un détecteur spécifique ainsi que les moteurs de translations utilisées pour cette expérience.

II.2.1.2. Le projet ELI-NP – équipe THALES

Le projet ELI-NP correspond aussi à un laser de puissance, il est installé en Roumanie et est développé par TOSA (Thales optronique SAS). C'est TOSA qui a formulé le besoin d'une application de positionnement faisceau à Thalès Services. Ce laser de puissance devant atteindre une puissance de 2x10 PétaWatts et être opérationnel en 2018. Ce projet s'est appuyé sur les développements back-end effectués pour le projet APOLLON. Le but étant simplement de porter ces développements en C++ pour obtenir de meilleures performances, et de créer des applications graphiques pour laseristes. Le principe algorithmique dans la détermination d'écart entre position faisceau et consigne utilisateur est donc le même que celui exprimé pour APOLLON.

Nous pouvons néanmoins noter que pour les besoins du projet ELI-NP, il faut pouvoir positionner le faisceau en deux points : NearField et FarField. Ce double positionnement permettra aux laseristes de pouvoir déterminer un axe faisceau. Deux asservissements fonctionnants en parallèle sont donc mis en place (Figure 17).

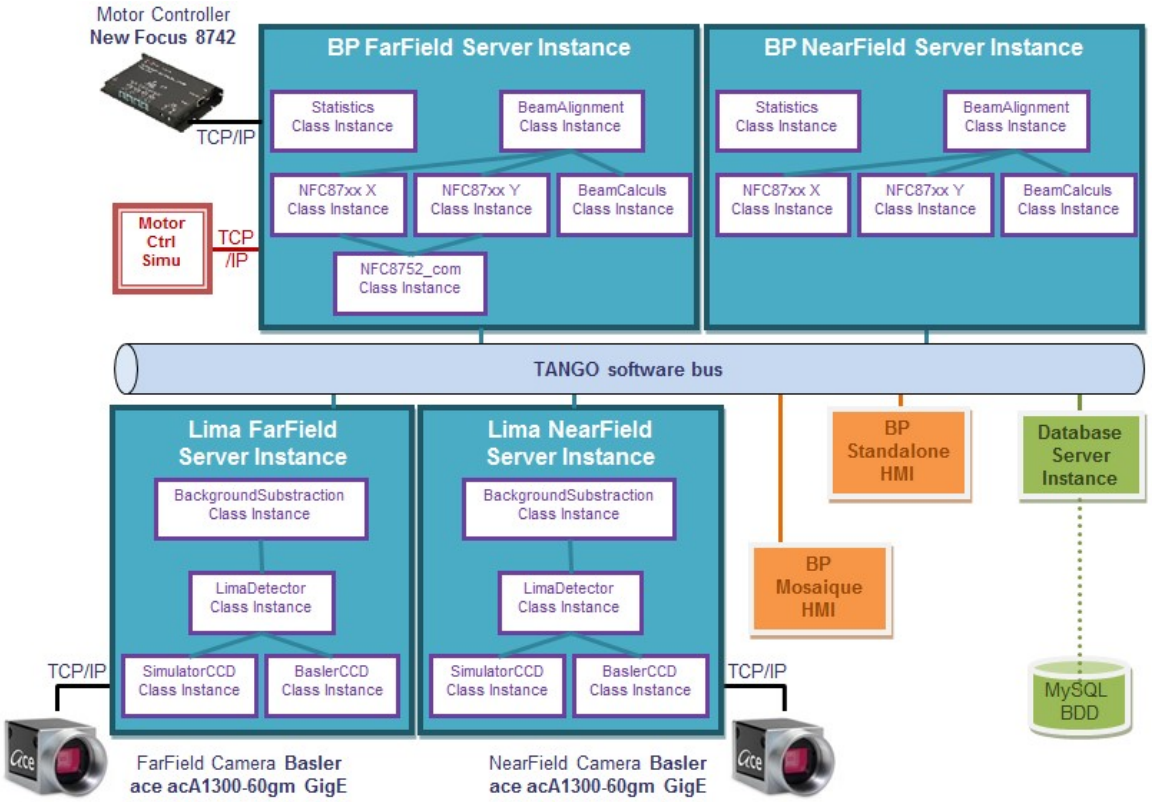


Figure 17 : Système asservissement de position Laser par ELI-NP (image fournie par THALES)

L'approche de l'asservissement faisceau pour ces deux projets semble donc pouvoir s'adapter à nos besoins. On note tout de même quelques limites :

- Leur conception ne peut prendre en charge qu'un seul type de capteur :
  - Cette application ne peut être utilisée qu'avec des device LIMA. Le calcul du centroïde est exclusivement réalisé sur des images.
  - Il n'est pas possible d'étendre son fonctionnement à d'autres types de détecteurs.
- Leur application n'est utilisable que pour un seul type d'image, une image 8 bits :
  - Nous aurons besoin de pouvoir travailler avec le plus grand nombre de détecteurs 2D possible, or ils ne fournissent pas tous une image 8 bits certains fournissant des images en 16, 32 ou 64 bits.
- L'application ne peut communiquer qu'avec des moteurs pas à pas de type « NFC87 »
  - De la même manière que pour les capteurs, nous ne pouvons pas nous servir d'autre type d'actuateurs, on ne peut pas étendre son fonctionnement.

- Le seuil à effectuer sur l'image ne peut pas être déterminé par l'utilisateur
    - Pour notre besoin, en fonction de l'énergie du faisceau, il faudra absolument pouvoir régler dynamiquement ce seuil.
- La détermination de l'erreur n'est effectuée que par un calcul de différence
    - Il est probable que dans certains cas, un asservissement plus sophistiqué soit nécessaire (correcteur PID pour NANOSCOPIUM).

II.2.1.3. Une solution temporaire sur NANOSCOPIUM

Pour satisfaire ses besoins concernant l'asservissement de la position faisceau sur une cible fixe, afin d'éviter les dérives de longue durée, la ligne a missionné un étudiant en alternance. Son rôle était de déterminer les coefficients d'asservissement PID à mettre en œuvre ainsi que d'écrire un script Matlab pour permettre de lire et écrire sur des devices TANGO les consignes d'asservissement nécessaires.

Si on compare cette solution avec celles vu précédemment, son avantage réside dans l'implémentation d'un régulateur PID. Sur le reste, le principe reste le même, une image 8 bits est analysée et un centroïde est calculé, le résultat de la différence entre consigne et centroïde est alors envoyé au correcteur.

On note que même si cette solution est temporaire, elle est en production sur la ligne et les résultats qu'elle fournit sont corrects pour notre client. Je pourrai donc m'assurer du bon fonctionnement de mon développement en m'appuyant sur les résultats produits par ce script, il fera office de référence.

II.2.2. Edition d'un cahier des charges pour la partie fonctionnelle

Je peux désormais cadrer plus précisément le projet et établir un premier cahier des charges avec mon client, Kadda Medjoubi. J'organise donc une réunion avec lui afin de pouvoir le déterminer à partir des besoins formulés. Je décide dans un premier temps de centrer ce cahier des charges sur la partie fonctionnelle uniquement. Je rappelle que cette partie fonctionnelle reposera sur l'écriture de devices TANGO. Compte tenu de l'environnement

informatique d'une ligne de lumière, cette application doit être développée de manière à ce qu'elle puisse être déployée sur des serveurs applicatifs Linux en priorité. Comme vu précédemment, la phase de création de l'IHM n'interviendra qu'après avoir validé le principe applicatif sur la ligne NANOSCOPIUM. Nous définissons ainsi les caractéristiques fonctionnelles suivantes pour l'application BeamPositionTracking :

Les paramètres « dynamiques » de l'application

Il faut comprendre par paramètres « dynamiques », des données de configuration qui peuvent être modifiées au cours du temps, en fonction des besoins de l'utilisateur : ce type de données est appelé un attribut dans le système TANGO.

- Seuil de l'image : **percentageDetection**
    - Il est primordial de pouvoir gérer le seuil à effectuer sur l'image avant détermination du centroïde faisceau.
    - En fonction des lignes sur lesquelles cette application sera utilisée, l'énergie ne sera pas la même, le niveau de bruit (rapport Signal/Bruit) n'est donc pas constant et doit être optimisé grâce au seuil de l'image.
- L'image sur laquelle est estimée le centroïde : **thresholdedImage**
    - Il faut afficher l'image une fois seuillée à l'utilisateur pour confirmer que le centroïde est bien effectué sur le faisceau et non sur un artéfact.
    - Cette image permettra aussi un retour visuel à l'utilisateur sur l'évolution de la position du faisceau en asservissement.
- La position courante du centroïde : **XAxisCurrentBeamPosition, YAxisCurrentBeamPosition**
    - Via deux attributs, position X et Y, l'utilisateur pourra relire le centroïde actuel du faisceau.
    - L'unité sera ici le pixel, c'est en effet la position du faisceau sur l'image retournée par le device responsable de l'acquisition d'image.
- La définition des cibles : **XAxisTarget, YAxisTarget**
    - L'utilisateur pourra à travers deux attributs scalaires définir le point de visée sur l'axe horizontal (axe X) et vertical (axe Y).
- La définition d'une zone d'asservissement : **warningZone [Radius, XCenter, YCenter]**

- L'utilisateur pourra définir une zone dans laquelle le faisceau sera asservi.
- Cette zone est composée d'un centre (coordonnée X et Y sur l'image en pixels), et d'un rayon (nombre de pixels).
- En dehors de cette zone l'asservissement du faisceau ne pourra pas être activé, il sera arrêté s'il était en cours d'exécution.
- Il s'agit de définir une zone de sécurité pour l'utilisateur, si le faisceau sort de cette zone, une action manuelle est nécessaire.

- Le seuil à partir duquel il faut déclencher un mouvement sur les translations sous-jacentes : **XAxisRegulationThreshold, YAxisRegulationThreshold**
  - Pour chaque axe, il s'agit de définir une valeur seuil (en pixel) à partir de laquelle un positionnement sera possible. En dessous de cet écart, on ne doit pas déplacer l'axe en question.
  - En fonction de l'état[4] de la ligne : Si elle est dite « chaude» des mouvements plus petits pourront être effectués sur les translations que lorsqu'elle est considérée comme « froide ».
  - Ces seuils auront principalement pour rôle d'éviter de trop solliciter les axes motorisés sur des petits déplacements lorsque la ligne est dite froide.

Les paramètres « fixes » de l'application

Il faut comprendre par paramètre « fixe », des données de configuration qui sont positionnées par l'utilisateur au lancement de l'application et dont la valeur reste constante tout au long de l'exécution de l'application : ce type de données est appelé une Propriété dans le système TANGO.

- La période de rafraichissement : **DevicePeriod**
  - Il se peut que pour certains systèmes, une période de rafraichissement particulière soit nécessaire (temps de relaxation du système physique).
  - Il sera possible pour l'utilisateur de définir la fréquence de fonctionnement de l'application.

[4] Une ligne de lumière est dite « chaude » lorsqu'elle est à l'équilibre en température. A l'ouverture d'une ligne, lorsque cette dernière est soumise au faisceau d'électrons, certains éléments (comme des miroirs sur lesquels est reflété le faisceau) vont chauffer pour atteindre une certaine température de fonctionnement, en dessous de cette température, la ligne est considérée comme « froide ».

- Le nombre d'images à échantillonner : **NbImgToAlign**
    - Le nombre d'images nécessaire pour le calcul d'un centroïde, et sur lequel on effectue une moyenne.
    - Il faut en effet pouvoir éviter l'influence de certains bruits sur la détermination d'un centroïde image.
- Le type de capteur à utiliser : **SensorType**
    - Dans un premier temps, la détection sera effectuée via une caméra (un capteur 2D).
    - Il sera par la suite possible de choisir parmi d'autres (XBPM).
- Le nom du capteur à utiliser : **SensorDeviceName**
    - Quel device est utilisé pour calculer le centroïde faisceau (adresse dans la base de données TANGO).
    - La classe de ce device doit correspondre à une classe interfacée dans notre système.
- Quel type d'actuateur pour les axes X et Y : **XAxisType, YAxisType**
    - Dans un premier temps je développerai les translations (ou rotations) de type GalilAxis.
- Le nom des actuateurs pour les axes X et Y : **XAxisDeviceName, YAxisDeviceName**
    - De même que pour le capteur, il faut pointer sur les devices concernant les moteurs correspondants aux axes concernés.
    - Un moteur permettant de bouger le faisceau sur l'axe X (horizontal) et l'autre sur l'axe Y (vertical).
- La définition des PID si utilisés : **XPIDDefinition, YPIDDefinition**
    - Un correcteur PID sera possiblement utilisé par axe.
    - Il faut pouvoir définir les coefficients de ces correcteurs pour X et Y.

<u>Les actions nécessaires</u>

Les commandes possibles sur les devices TANGO :

- Le démarrage de l'asservissement du faisceau : **StartBeamTracking**

- Au déclenchement de cette commande, la position du faisceau doit être asservie de manière continue sur la cible définie par l'utilisateur jusqu'à arrêt utilisateur, ou erreur critique pendant l'asservissement.

- Arrêt de l'asservissement : **StopBeamTracking**
  - Au déclenchement on arrête l'asservissement en cours.

Je suis donc désormais capable de fournir le use case global de l'application BeamPositionTracking (Figure 18). Ce use case me permet de figer les demandes utilisateur avant de passer au développement de notre application.

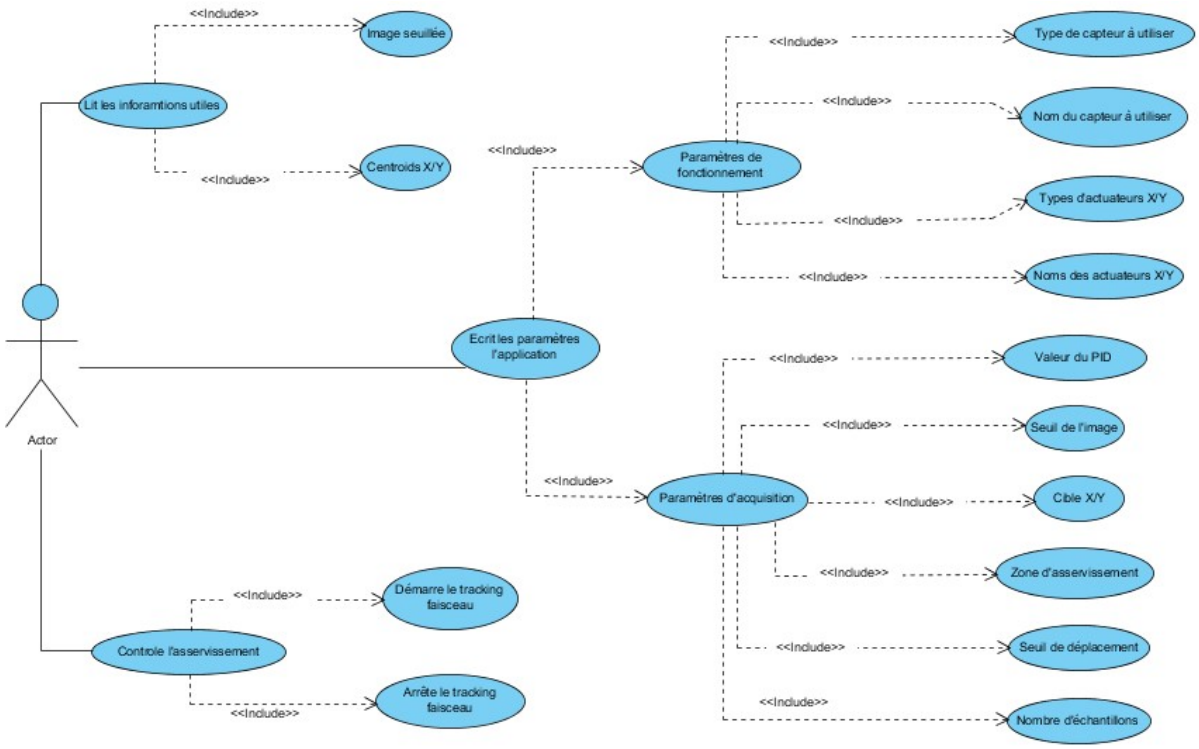


Figure 18 : Use case général de l'application BeamPositionTracking

II.3. Principe applicatif

Dans cette partie j'essayerai de déterminer quels principes applicatifs mettre en jeu pour pouvoir répondre au cahier des charges défini précédemment. Je décrirai quelles technologies je mettrai en œuvre et les principes algorithmiques utilisés dans les étapes clés de cette application. Cette analyse me permettra de déboucher sur la phase de conception.

Je précise que l'ensemble de la phase de développement a été effectuée à partir d'un serveur linux mutualisé (virtuel) sur lequel l'environnement TANGO est déployé, pour reproduire les mêmes conditions que sur une ligne de lumière. Ce serveur étant accessible depuis le ReS (voir partie I.2.1).

II.3.1. Choix de la librairie de traitement image

Pour mettre en place ce traitement d'image afin de déterminer le centroïde faisceau, je m'appuierai sur une librairie de calcul tierce partie, il serait en effet trop lourd pour ce projet de redévelopper le nécessaire à la main et le résultat serait difficilement maintenable et évolutif. Lors de nos échanges avec l'équipe de développement de THALES, il a été convenu de récupérer les sources de leur développement pour comprendre leur logique applicative. J'ai donc étudié leur solution et j'ai pu voir qu'ils se basaient sur la librairie OpenCV.

En complément à cette étude, je précise que la librairie OpenCV est déjà utilisée à SOLEIL pour un device appelé ImageBeamAnalyzer. Ce device provient d'un besoin commun entre la machine et les lignes de lumière pour traiter une image et déterminer les caractéristiques du faisceau. Le device actuel peut notamment fournir des traitements tel que :

- Image flip : Rotations image
- AutoROIs : Région d'intérêt automatique
- UserRoi : Région d'intérêt spécifique
- Histogramme : le profile en X et Y d'une image

De plus, le framework TANGO étant OpenSource, il parait logique d'utiliser une librairie tierce ayant les mêmes droits de diffusion ce qui est le cas d'OpenCV. L'application ainsi produite pourra être partagée avec l'ensemble de la communauté ce qui entre aussi en compte dans notre cahier des charges.

Pour finir, OpenCV est une librairie de traitement d'image qui propose des algorithmes de détection de contour, j'ai pu observer lors de mon étude de la solution proposée par Thalès qu'ils avaient utilisé un des algorithmes proposés par OpenCV. Aussi, il existe grand nombre d'utilisateur qui a pu mettre en œuvre ce genre de détections, communauté sur laquelle je pourrai m'appuyer pour mettre en place le calcul de centroïde faisceau. Cette librairie est

beaucoup utilisée dans le monde scientifique, c'est notamment le cas du CEA (Commissariat à l'Énergie Atomique) avec qui nous entretenons de nombreuses collaborations sur différents projets informatiques, ce qui va conforter mon choix aux vues des retours expérience possibles.

Je choisis d'utiliser OpenCV en version 2.3.1 pour commencer mon développement. En effet, pour des raisons de compatibilités avec l'environnement déployé à SOLEIL et notamment la version du compilateur C++ utilisé, je ne peux pas pour le moment utiliser de version plus récente d'OpenCV (la dernière version étant la 3.4.2). Une première lecture de cette librairie me permet de voir qu'il n'y a pas de changements conséquents entre la version 2.3.1 et les dernières versions d'OpenCV dans le cadre d'un calcul de centroïde, la mise à jour de cette librairie dans la suite du projet ne sera donc pas impactant.

II.3.2. Prise en main de TANGO

II.3.2.1. Exemple de device server

Comme décrit dans le contour du projet ainsi que le cahier des charges, je dois fournir une implémentation de la partie fonctionnelle basée sur des devices TANGO afin de proposer une solution s'intégrant le plus simplement possible au système de contrôle déjà existant à SOLEIL et utilisé par l'ensemble des lignes de lumière. Pour des raisons de performance de fiabilité et d'environnement logiciel, ces devices seront développés en C++.

Pour commencer mon développement, je décide de créer un device de tests qui me servira ensuite de base d'environnement de développement, afin de tester au fur et à mesure les principes mis en jeu. Pour construire un device server, les développeurs travaillant sur TANGO peuvent s'appuyer sur un premier outil de conception : POGO (Class Generator). POGO est une application permettant de générer le squelette d'un device C++, Java ou encore Python. C'est grâce à cet outil qu'un développeur va pouvoir spécifier le nom de la classe du device, ses attributs, ses commandes, propriétés et états.

Il est à noter qu'un ensemble de bonne pratique de développement ont étés mises en place à SOLEIL afin d'harmoniser les développements, cela concerne aussi les règles de nommage de l'interface des devices.

« Extrait des bonnes pratiques de développement à SOLEIL : »

Nom d'une propriété : Le nom d'une propriété commence toujours par une majuscule. Tous les mots composant le nom d'une propriété doivent commencer par une majuscule. Les majuscules serviront de séparateur, tout autre séparateur est déconseillé. Si la propriété est de type array, alors il convient de la mettre au pluriel.

```
std::string      UndulatorProxyName;
```

Nom d'un attribut : Le nom d'un attribut commence toujours par une minuscule. Les autres mots composant le nom de cet attribut doivent commencer par une majuscule. Les majuscules serviront de séparateur, tout autre séparateur est déconseillé.

```
TANGO::DevDouble   computedGap;
```

Nom d'une commande : Le nom d'une commande commence toujours par une majuscule. Tous les mots composant le nom d'une commande doivent commencer par une majuscule. Les majuscules serviront de séparateur, tout autre séparateur est déconseillé.

```
void ChooseMode( TANGO::DevShort mode );
```

Pour mon exemple, je choisis de créer un device de test que je nomme **TestBPTDevice,** sa composition est la suivante :

- Un attribut de type Scalaire (nombre simple), « *attributDouble* »
  - READ_WRITE : L'utilisateur pourra lire et écrire sur l'attribut
  - TANGO::DEV_DOUBLE : Type TANGO correspondant au type primitif « Double »
- Un attribut de type image, « *imageTest* »
  - READ : L'utilisateur ne peut que lire l'attribut
  - TANGO::DEV_UCHAR : Chaque point de l'image (du tableau 2D constituant l'image) est de type « unsigned char »
- Une commande, « *ProcessImage* » : Ne prend pas d'argument, ne retourne pas de valeur
- Deux états :

- TANGO::ON : Décrit un système en fonctionnement
- TANGO::OFF : Décrit un système à l'arrêt

En suivant ce cahier des charges, je me sers de POGO pour générer automatiquement le squelette de notre device (Figure 19) :

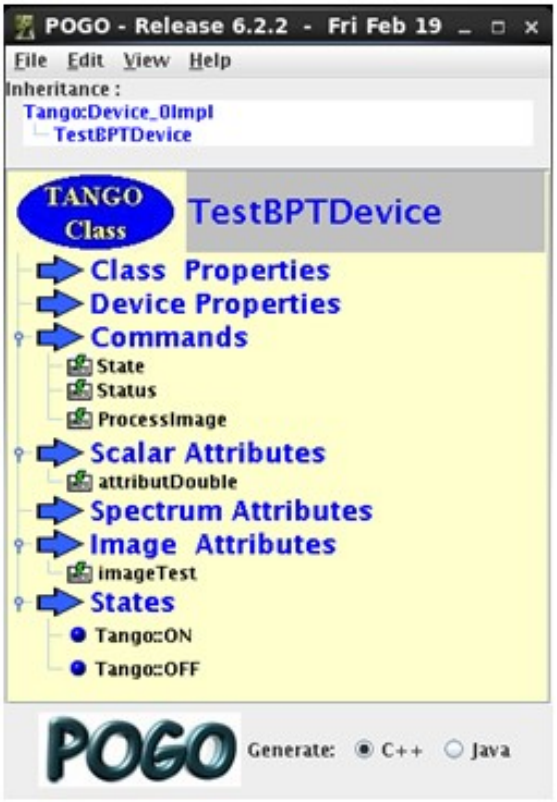


Figure 19 : Vu du POGO pour TestBPTDevice

Une fois mon architecture mise au point, il me suffit de générer le code du squelette. L'utilitaire va créer automatiquement les classes nécessaires à un device TANGO dans le répertoire souhaité (Figure 20) :

```
C++ source code generation:

/home/techniques/alignement/thiam/CNAM/Memoire/PROJET/DEV/SRCs/Devices/TestDevice/TangoClassID.txt     Created
/home/techniques/alignement/thiam/CNAM/Memoire/PROJET/DEV/SRCs/Devices/TestDevice/Makefile for C++     generated
/home/techniques/alignement/thiam/CNAM/Memoire/PROJET/DEV/SRCs/Devices/TestDevice/main.cpp     Created
/home/techniques/alignement/thiam/CNAM/Memoire/PROJET/DEV/SRCs/Devices/TestDevice/ClassFactory.cpp     Created
/home/techniques/alignement/thiam/CNAM/Memoire/PROJET/DEV/SRCs/Devices/TestDevice/TestBPTDeviceClass.h     Generated
/home/techniques/alignement/thiam/CNAM/Memoire/PROJET/DEV/SRCs/Devices/TestDevice/TestBPTDeviceClass.cpp     Generated
/home/techniques/alignement/thiam/CNAM/Memoire/PROJET/DEV/SRCs/Devices/TestDevice/TestBPTDevice.h     Generated
/home/techniques/alignement/thiam/CNAM/Memoire/PROJET/DEV/SRCs/Devices/TestDevice/TestBPTDevice.cpp     Generated
/home/techniques/alignement/thiam/CNAM/Memoire/PROJET/DEV/SRCs/Devices/TestDevice/TestBPTDeviceStateMachine.cpp     Generated
```

Figure 20 : Résultat de génération POGO

La classe **TestBPTDeviceClass** est la classe responsable de la création de l'interface du device. C'est notamment dans cette classe que seront déclarées les sous classes d'attributs et de commandes nécessaires au device.

La classe **TestBPTDeviceStateMachine** sera générée en fonction des états crées dans POGO et les contraintes de chacun : On peut directement définir via l'utilitaire que certaines commandes ou certains attributs ne pourront pas être accessibles dans certains états (par exemple dans l'état OFF d'un système, il n'est pas possible de l'éteindre), ils seront alors automatiquement interdits à l'utilisateur, une exception sera automatiquement levée si un appel non cohérent est effectué.

Reste à la charge du développeur d'implémenter la logique de son device. L'implémentation se fera tout d'abord dans la classe **TestBPTDevice**.

Utilisation de l'API TANGO : Pour pouvoir implémenter ce device, en plus de la logique-métier, il faut mettre à jour les valeurs des attributs, lire les valeurs entrées par l'utilisateur, mettre à jour les états du device etc… L'ensemble de cette API est décrite dans un document de référence (environ 420 pages): « TANGO Control System Manual ». Nous pouvons néanmoins donner quelques exemples d'utilisations basiques.

Lecture/Ecriture d'un attribut :

Comme nous l'avons vu, POGO a généré le squelette de notre device. Ainsi la classe TestBPTDevice contient les méthodes nécessaires pour implémenter la lecture et l'écriture de chacun des attributs définis dans notre device. Toutes ces méthodes sont vides, il est à la charge du développeur de les implémenter (nous pouvons lancer notre device en l'ayant au préalable déclaré sur la base de données TANGO, il n'affichera rien et aura un état inconnu « UNKNOWN » si rien n'est fait).

Si on prend l'exemple de notre attribut : « attributDouble », deux méthodes seront créées automatiquement dans notre classe **TestBPTDevice**, une méthode pour la lecture de l'attribut, une autre pour son écriture. Nous implémentons ces méthodes pour donner un exemple de lecture et d'écriture comme suit :

« Extrait du fichier TestBPTDevice.cpp »

```
//+----------------------------------------------------------------------------
----
//
// method :            TestBPTDevice::read_attributDouble
//
// description :       Extract real attribute values for attributDouble
acquisition result.
//
//-----------------------------------------------------------------------------
----
void TestBPTDevice::read_attributDouble(TANGO::Attribute &attr) {
        //Dans la partie read de l'attribut, on renseigne la valeur à
afficher
        double valeurFixe = 100;
        attr.set_value(&valeurFixe);
}


//+----------------------------------------------------------------------------
----
//
// method :            TestBPTDevice::write_attributDouble
//
// description :       Write attributDouble attribute values to hardware.
//
//-----------------------------------------------------------------------------
----
void TestBPTDevice::write_attributDouble(TANGO::WAttribute &attr){
        //Dans la partie write, on récupère la valeur entrée par
l'utilisateur
        double valeurAttribut = 0;
        attr.get_write_value(valeurAttribut);
        std::cout<<"Entered Value : "<<valeurAttribut<<std::endl;
}
```

A partir de tout client connecté sur ce device (ATK Pannel, client Python ou autre…), si une lecture ou écriture est effectuée sur l'attribut « attributDouble », ce seront ces méthodes qui seront appelées, de manière synchrone.

Mise à jour d'un état :

Pour mettre à jour un état dans un device, il est recommandé d'utiliser la méthode appelée « always_executed_hook() ». Cette méthode est présente dans tout device, elle fait partie des méthodes virtuelles de la classe TANGO::Device_4Impl, dont tout device hérite. Sa spécificité est qu'elle est appelée de manière automatique avant l'exécution de chaque commande sur le device. L'état d'un device devant être rafraichi le plus souvent possible (c'est lui qui décrit l'état du système) et étant retourné au client via la commande « get_state » du device, il parait logique de le mettre à jour à cet endroit :

« Extrait du fichier TestBPTDevice.cpp »

```
//+----------------------------------------------------------------------------
----
//
// method :            TestBPTDevice::always_executed_hook()
//
// description :       method always executed before any command is
executed
//
//-----------------------------------------------------------------------------
----
void TestBPTDevice::always_executed_hook(){
        //On choisit de fixer la valeur de l'état à OFF
        set_state (TANGO::OFF);
}
```

Compilation – Lancement d'un device :

Une fois mon implémentation terminée, je peux compiler ce device. J'utilise Maven pour ce faire. Cette compilation sera paramétrée grâce à un fichier descriptif appelé « pom.xml ». C'est entre autres dans ce fichier que nous allons donner toutes les informations de dépendances liées au device, on donnera aussi l'objet de la compilation (une librairie ou un device). Voir en Annexe 4 le fichier pom.xml correspondant au device TestBPTDevice.

Il faut organiser chaque projet de manière à avoir tous les fichiers sources dans un répertoire appelé « src », au même niveau que le fichier pom.xml. A partir de ce répertoire dans un terminal, il faut envoyer la commande Maven qui fera appel au compilateur de Linux : « mvn clean install » pour compiler le projet. Il est à noter que la procédure est la même sous Windows (au lieu du GCC, l'appel pointera sur le compilateur Visual C++), d'où l'intérêt d'utiliser Maven qui est indépendant de la plateforme utilisée.

Une fois compilé, il est possible de démarrer ce device (déclaré sur la base de données TANGO en tant que tmp/testtango/bpttest), on pourra alors le contrôler en utilisant l'ATK-Panel, voir Figure 21.

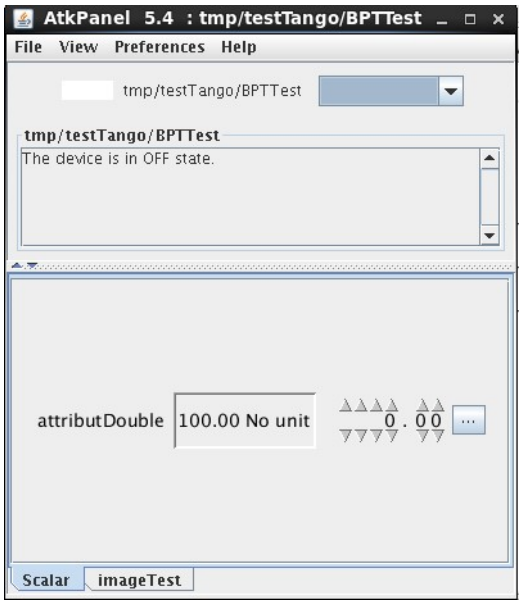


Figure 21 : Vue de l'ATK-Panel, device testBPTDevice

II.3.2.2. Devices collaboratifs : La classe DeviceProxy

Pour mettre en œuvre ce projet, il faut être capable de faire collaborer plusieurs devices (lire une image sur un device, piloter un moteur à travers un autre device). Pour rappel, le système TANGO est capable de faire communiquer des devices entre eux, toute cette communication est réalisée à travers le middleware CORBA (vu en introduction).

L'API TANGO fournit la classe DeviceProxy, classe qui aura pour rôle de représenter un device distant. Nous utiliserons cette classe pour faciliter l'échange d'information entre deux devices. En effet, l'API TANGO prend soin de masquer toutes les complexités liées à l'adressage du device distant (récupérer son adresse en base de données en fonction de son nom), l'encapsulation des données à faire transiter etc… Voir Figure 22.

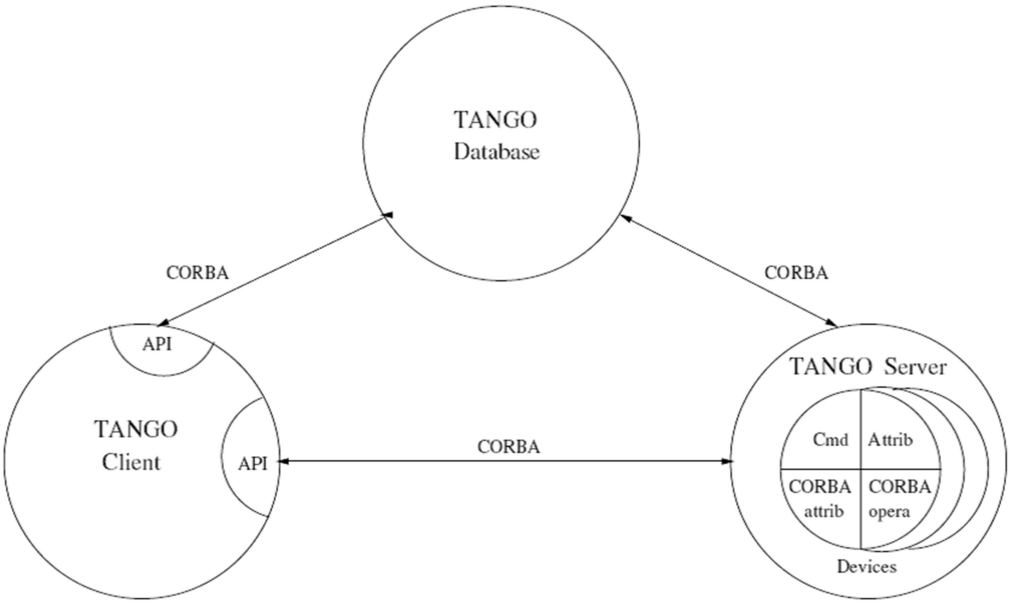


Figure 22 : Principe de communication TANGO (image provenant du site TANGO controls)

On note que cet échange de données va pouvoir intervenir entre deux processus différents et potentiellement sur deux serveurs (au sens matériel) différents, à travers un réseau donné.

Les types de données échangées sont fixés, ainsi la liste des types disponibles nous est donnée, voir Annexe 3. Je dois donc me baser sur ces types de références pour développer ces devices et échanger des données entre eux.

II.3.2.3. Manipulation d'un détecteur 2D : Le device LIMA detector

L'acronyme LIMA signifie : Library for Image Acquisition. C'est en fait un framework permettant l'unification du contrôle des détecteurs 2D à SOLEIL. Ce développement est réalisé en collaboration avec l'ESRF.

Il permet d'une part de séparer le code spécifique à chaque détecteur (la partie spécifique due au Hardware du détecteur Figure 23) du code commun et nécessaire à tous détecteur pour leur contrôle comme l'enregistrement d'une image, le déclenchement d'une acquisition etc…

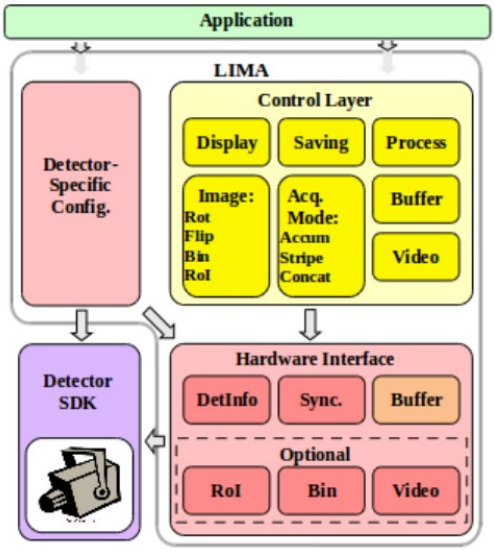


Figure 23 : Architecture device LIMA (*tiré de la banque d'images du synchrotron-soleil*)

Il propose donc une interface commune, c'est-à-dire que tout détecteur interfacé dans ce framework sera représenté à travers un device server TANGO possédant les mêmes attributs, commandes, propriétés, machine à état : le device LIMADetector. Les traitements d'images intégrés à LIMA sont les mêmes pour tous nos détecteurs, il sera par exemple possible de choisir une ROI (Region of Interest) ou d'effectuer une rotation sur l'image. Le code nécessaire sera factorisé. C'est aussi dans ce device que le déclenchement de l'acquisition sera effectué.

Pour communiquer avec le hardware mis en jeu, le device LIMADetector fait appel à un plugin, le type de plugin à utiliser est défini dans les propriétés de ce device. C'est dans ce plugin que sera placé le code spécifique au SDK (Software Development Kit) du détecteur correspondant. Chaque interfaçage de détecteur 2D dans ce framework fait donc l'objet d'un développement de plugin, il est alors à la charge du développeur de respecter l'interface connue d'un plugin LIMA.

Si nécessaire, un device spécifique supplémentaire est développé, en fonction des particularités du détecteur. Le device server LIMADetector peut en fait être composé de deux classes de device TANGO. On précise qu'un device server peut en effet héberger plusieurs classes de device. Dans ce cas précis, le device server LIMADetector peut héberger une classe de device LIMADetector (traitement générique inhérent à tout détecteurs) et une

classe de device spécifique. La classe de ce device spécifique dépendra du détecteur utilisé. Dans ce device spécifique sont développées les fonctions particulières au détecteur utilisé, l'ensemble du code qui ne peut pas être factorisé. Certains détecteurs sont par exemple capables d'effectuer des traitements en interne particuliers sur l'image. Il faut pouvoir accéder à ces traitements dans le monde TANGO, d'où le développement d'un device particulier. Dans le cas d'un détecteur n'ayant pas de fonction particulière, il ne sera pas nécessaire de créer ce device spécifique.

Si on prend l'exemple du détecteur Basler (détecteur CCD couramment utilisé à SOLEIL) on a principalement un ds_LIMADetector accompagné d'un ds_BaslerCCD, Figure 24 :

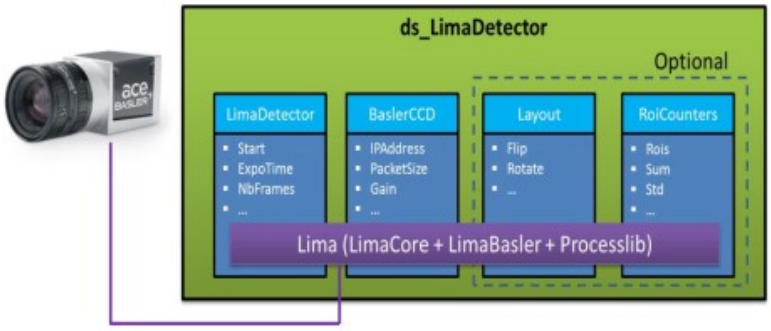


Figure 24 : Imbrication logicielle LIMA pour camera BASLER (*tiré de la banque d'images du synchrotron-soleil*)

Pour conduire mes tests de faisabilité, j'ai pu me servir du device LIMADetector avec comme device spécifique le LIMASimulator. Ce développement a été réalisé principalement pour tester le la mise en œuvre de nouveaux calculs sur une image faisceau sans avoir à utiliser de matériel physique. Cette image sera en fait fabriquée de manière logicielle et non récupérée à partir d'un détecteur (voir partie II.3.3.1, avec l'exemple du seuil).

II.3.3. Principe de détection de centroïde

Pour tester cet algorithme de détection, afin de valider mes hypothèses, je me baserai donc sur l'étude d'une image de faisceau dite « parfaite » fournie par le device LIMA avec son plugin Simulator. De cette manière je peux m'assurer du bon fonctionnement de ma solution avant de contacter le client pour effectuer des tests sur un vrai faisceau.

Objectif de l'algorithme de traitement d'image :

- Sur une image produite par un device sous-jacent, déterminer si oui ou non le faisceau est présent
- Si le faisceau est présent, déterminer son centroïde : sa position horizontale et verticale en pixels
- L'image peut être d'une forme et taille quelconque
- L'image peut avoir un format de 8bits, 16bits ou 32bits

Des recherches à partir de l'existant (à partir du développement de THALES) et dans la librairie OpenCV ont conduites à l'étude de l'algorithme **« Creating Bounding boxes and circles for contours »** (proposé par OpenCV). A priori cet algorithme ne s'apparente pas à une détection de faisceau (détection d'une intensité lumineuse), mais plutôt à de la détection de forme et dessin de contour. Il est par exemple possible de détecter des formes telles que celles représentées en Figure 25 :

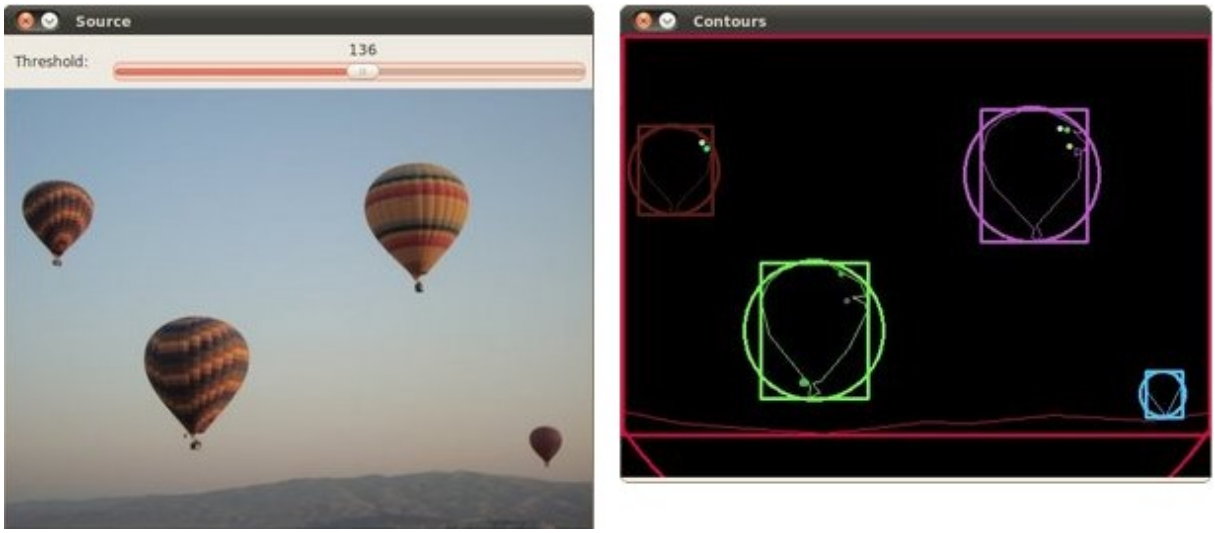


Figure 25 : Exemple de contour sur une image (image provenant du site OpenCV)

Comme on peut le voir sur l'image ci-dessus, l'algorithme **« Creating Bounding boxes and circles for contours »** mis en place va permettre de déterminer à la fois des rectangles et des cercles à partir de présence de formes quelconques.

L'idée serait donc d'utiliser cette première partie de traitement pour détecter le faisceau. Pour ce faire, il faut effectuer une première étape de « seuillage d'intensité ».

II.3.3.1. Un seuil sur l'image

Il est possible dans certains cas que l'image du faisceau ne soit pas assez claire pour déterminer le centroïde faisceau. C'est-à-dire que l'intensité produite n'est pas suffisante pour qu'il puisse être détecté, le rapport signal/bruit peut être trop faible pour en déduire le faisceau. La première étape est donc de nous assurer que le faisceau soit suffisamment intense pour être traité, et nous pouvons nous appuyer sur un seuil pour ce faire.

Pour effectuer ce seuil, Il faut déterminer une valeur de référence puis comparer chaque valeur de pixel de l'image à cette dernière. Si la valeur d'un pixel y est supérieure il est conservé sinon il est ramené à 0 (par défaut). Dans notre cas, l'idée revient à « binariser » l'image à partir de cette valeur de seuil. De cette manière nous pouvons nous assurer qu'en dessous d'une certaine intensité, il ne sera pas possible de calculer le centroïde, afin de ne pas prendre en compte un bruit résiduel.

Threshold Binaire sur une image :

La première étape consiste à déterminer la valeur pour laquelle l'intensité (valeur du pixel) sera suffisante pour le traitement de l'image. Cette valeur de threshold sera à déterminer par l'utilisateur (cahier des charges en partie II.2.2), en effet seul l'expert métier sera en mesure de régler cette information. Néanmoins pour plus de lecture, il parait plus simple de ramener cette valeur à un pourcentage, tel que :

- Threshold_Final = (Intensité max – intensité min) x Threshold_utilisateur(%)

La valeur de threshold sera déterminée en fonction du minimum et maximum (intensité de chaque pixel) de l'image. De cette manière, pour un Threshold_utilisateur à 0% toute l'image est gardée, à 100% on ne garde plus rien. Nous avons donc toute la latitude nécessaire, et ce pour n'importe quelle image, puisqu'on se base sur un minimum et maximum relatif à l'image traitée.

Ensuite, grâce à la librairie OpenCV et sa fonction threshold, nous pouvons procéder comme suit pour réaliser le threshold sur notre image :

$$\texttt{dst}(x,y) = \begin{cases} \texttt{maxVal} & \text{if } \texttt{src}(x,y) > \texttt{thresh} \\ 0 & \text{otherwise} \end{cases}$$

- Si l'intensité du pixel src(x,y) est plus grande que celle fixée à « thresh », le nouveau pixel vaudra maxVal, sinon il vaudra 0.
- On note que nous choisirons de fixer maxVal à 1.

Voir figure Figure 26, au-dessus d'une certaine valeur le pixel prendra la valeur 1, en dessous il sera ramené à 0.

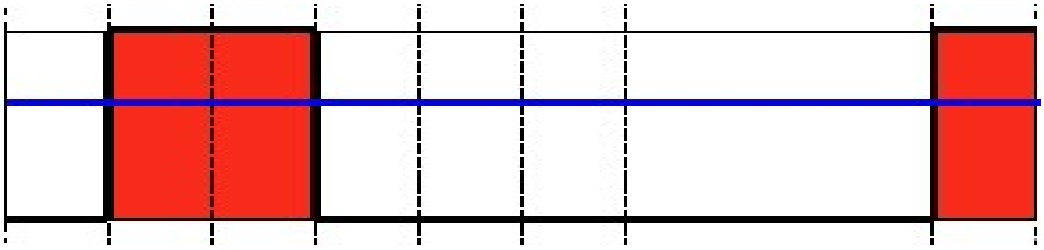


Figure 26 : Diagramme threshold (image provenant du site OpenCv)

De cette manière nous somme capable de construire une nouvelle image constituée uniquement de 0 ou de 1, une image qui sera donc binaire. Si nous choisissons de travailler sur notre image d'un faisceau simulé (provenant du device LIMASimulator) pour illustrer cette conversion, nous obtiendrons en choisissant un threshold à 50% (Figure 27):

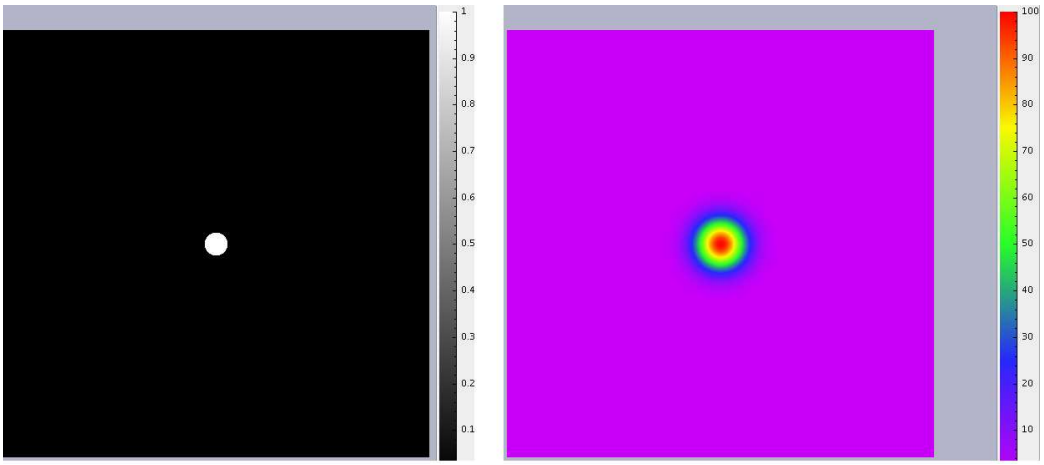


Figure 27 : Exemple de binarisation d'une image de test

II.3.3.2. Calculs de contours

Un contour correspond à une liste de points qui représentent une forme dans une image. La méthode de détermination de ces contours est basée sur la notion d'arbre de contours, une liste de contours englobants (récursivement) présents dans une image. On précise que cette fonction ne sera capable de traiter que des images binaires, ce qui est notre cas puisque nous effectuerons ce calcul qu'après «binarisation» de notre image source.

D'après Gary Bradski et Adrian Kaehler (2008) nous pouvons nous appuyer sur l'image Figure 28 pour comprendre comment sont déterminés ces arbres de contours. Le profil test du haut correspond à l'image en entrée de la fonction FindContours, en sortie on retrouve les différents contours déterminés par OpenCV :

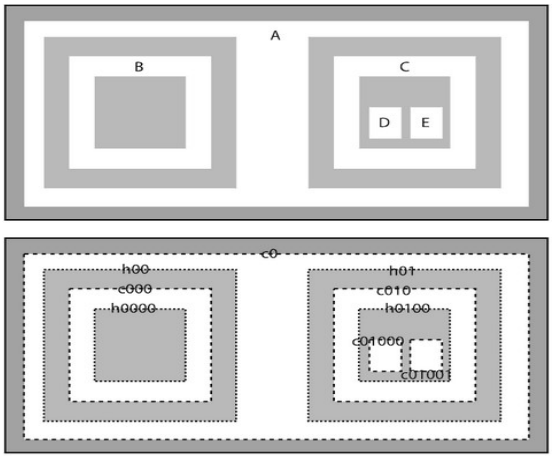


Figure 28 : Définition des contours de la librairie OpenCV. Gary Bradski et Adrian Kaehler (2008).

Comme on peut le voir, les contours sont de différents types, la fonction FindContours fait la distinction entre deux types de contours, de type C et H :

- C : Contour
  - Les tracés pointillés épais
  - Correspondent aux limites extérieures aux zones blanches
- H : Hole
  - Les tracés pointillés fin
  - Les limites intérieures des zones blanches

Il existe alors plusieurs possibilités pour récupérer les contours, l'utilisateur d'OpenCV peut choisir de récupérer l'intégralité de l'arbre des contours (Figure 29) ou préciser lesquels l'intéresse :

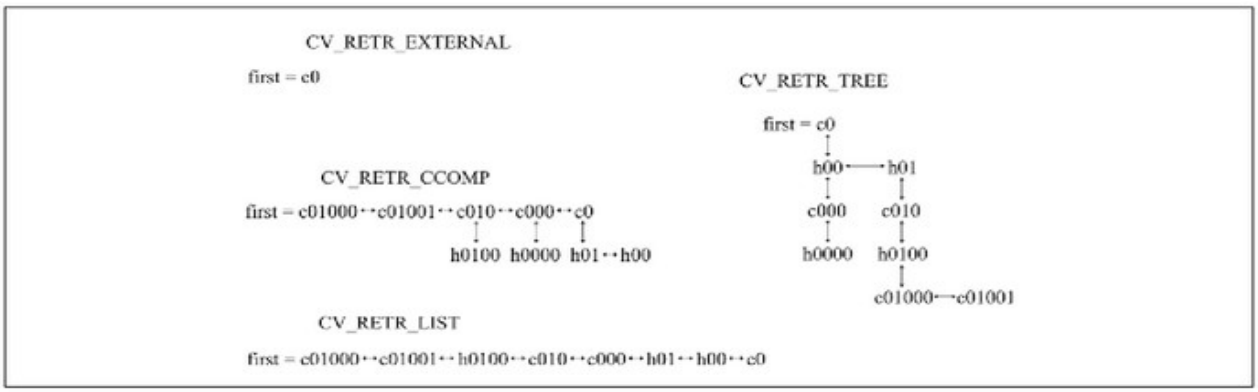


Figure 29 : Arbre des contours OpenCV. Gary Bradski et Adrian Kaehler (2008).

Dans notre cas, il nous faudra simplement récupérer le contour extérieur à notre zone blanche principale (le faisceau), nous utiliserons donc la directive **CV_RETR_EXTERNAL**. On note que dans l'exemple, c'est bien avec cette directive que l'on récupère le contour externe C0, qui correspondrai au contour de notre faisceau.

II.3.3.3. Détermination de rectangles englobants

Nous allons désormais créer des rectangles à partir des contours déterminés précédemment. En effet, sur notre image parfaite nous n'obtiendrons qu'un seul contour, mais dans la réalité, il est possible que le faisceau émette des plus petites « tâches » (dues à une diffraction parasite en amont ou un problème de détecteur par exemple).

Nous utiliserons la fonction d'OpenCV « BoundingRect » pour créer une liste de rectangles englobants. Cette fonction va prendre en entrée un vecteur de points c'est-à-dire le contour. Le fait de passer par des rectangles va en fait nous permettre de réaliser de manière simple deux actions de contrôle sur les contours déterminés.

Contrôle de la taille minimale du faisceau :

- On crée un rectangle de référence d'une taille connue (pour nos tests on considère qu'en dessous de 5 pixels, il n'y a pas de faisceau).

- Pour tous les rectangles créés avec nos contours, on s'assure que leur hauteur et largeur est supérieure à celle de notre rectangle minimum de référence.
    - On élimine ainsi les artefacts liés à des défauts sur le détecteur

Détermination du rectangle le plus grand :

- Nous savons qu'il n'est pas possible d'obtenir deux sources faisceaux sur une image.
- L'utilisateur a, en amont de ce traitement, effectué les réglages de threshold nécessaires pour que le faisceau apparaisse nettement sur l'image : même s'il y a du bruit, il doit représenter la forme la plus grande après « binarisation » sur l'image afin de garantir l'asservissement.
- Il parait logique que le rectangle englobant concernant le faisceau corresponde au plus grand contour.
- Nous devons donc comparer les rectangles les uns aux autres afin de trouver lequel est le plus grand, simplement en nous basant sur les hauteurs et largeurs de chacun d'entre eux.

A ce point, nous avons déterminé quel contour est celui qui correspond à celui de notre faisceau.

II.3.3.4. Détermination du centroïde faisceau

Une fois avoir trouvé le bon contour, il faut déterminer son centre de masse afin d'obtenir de manière précise le centroïde faisceau. En m'appuyant sur mes recherches quant à la détermination d'un centre de contour avec la librairie OpenCV, je décide de me servir de la méthode de déterminations des moments, CvMoment :

- On donne en entrée une liste de points (notre contour)
- On récupère en sortie les moments du contour

<u>Définition de la notion de moment (d'après la librairie OpenCV) :</u>

Les moments sont des mesures quantitatives utilisées à la base en mécanique ou en statistique pour décrire une distribution spatiale à partir d'un ensemble de points. Dans le

traitement d'une image il s'agit à la base de déterminer son centre de masse en fonction son intensité, on nous donne l'équation de détermination d'un moment suivante :

$$M_{ij} = \sum_x \sum_y I(x,y)$$

- Avec i ordre du moment sur l'axe X, et j ordre du moment sur l'axe Y
- I(x,y) correspond à l'intensité du pixel à la position x et y

Les points d'une image forment sa masse. Le moment d'ordre 0, correspond au total de la masse (*somme de tous les points*). Le premier moment (moment d'ordre 1) pour chaque axe divisé par le total de masse correspond au **centre de masse**.

Utilisation de la fonction CvMoments : Cette fonction nous retourne trois types de moments, le Moment spatial (m), le Moment central normalisé (nu) et le Moment central (mu). D'après mes recherches, je choisis de me baser sur le moment central (mu) pour déterminer le centre du contour.

Pour comprendre à quoi il correspond au sens mathématique, la documentation disponible dans OpenCV nous donne l'équation suivante, avec x et y les coordonnées de chaque point composant le contour :

$$\mathtt{mu}_{ji} = \sum_{x,y} \left(\mathtt{array}(x,y) \cdot (x - \bar{x})^j \cdot (y - \bar{y})^i \right)$$

Avec une estimation des centres de masse tels que :

$$\bar{x} = \frac{m_{10}}{m_{00}},\ \bar{y} = \frac{m_{01}}{m_{00}}$$

M10 correspond au moment d'ordre 1 sur X, M01 correspond à celui de l'axe Y. Comme dit plus haut, pour avoir le centre de masse il faut les diviser par la masse totale (M00), c'est une moyenne pondérée.

En sortie de ce processus, nous obtenons finalement le centroïde de notre faisceau sur nos deux axes (horizontal et vertical), en pixel.

II.3.4. Description du correcteur

Comme convenu dans le cahier des charges, il faut mettre en place un correcteur de type PID : Proportionnel Intégral Dérivé. On rappelle qu'un correcteur est en fait un algorithme de calcul qui détermine une commande à partir d'une différence entre consigne mesure.

La particularité du système BeamPositionTracking est que la consigne (cible sur l'image) entrée par l'utilisateur est en fait constituée de deux sous consignes, une consigne sur X et l'autre sur Y. Il existe en effet un moteur responsable de chaque axe. Il faut donc mettre en place deux PID, chaque axe ayant ses propres coefficients de régulation.

Pour résumer son intégration dans notre système, nous pouvons donner le schéma suivant (Figure 30) :

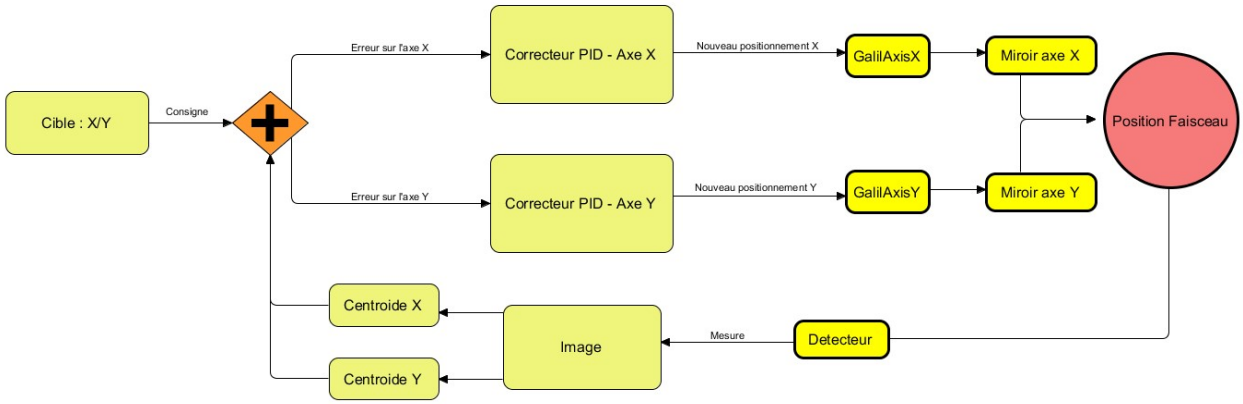


Figure 30 : Description du système d'asservissement

En entrée de notre système d'asservissement nous aurons donc, en pixels :

- **La consigne utilisateur**
  - Cible sur l'axe X
  - Cible sur l'axe Y
- **La lecture du centroïde actuel**
  - Centroïde X
  - Centroïde Y

En sortie des correcteurs PID parallèles, en unité correspondant à l'attribut position du device de motorisation correspondant :

- **Consigne de déplacement sur chaque moteur**
  - GalilAxis X
  - GalilAxis Y

Ces déplacements engendreront un déplacement faisceau qui modifiera en toute logique la valeur des centroïdes X et Y. En me basant sur l'étude d'asservissement fournie en amont par la ligne NANOSCOPIUM, l'expression mathématique d'un correcteur PID est la suivante :

- Proportionnel P : $up_k = K . e_k$
- Intégral I : $ui_k = ui_{k-1} + K . \frac{Te}{Ti} . e_k$
- Dérivé D : $ud_k = K . \frac{Td}{Te} . (e_k - e_{k-1}) . ud_{k-1}$

On note qu'en fonction de son système, l'utilisateur pourra alors choisir d'associer ces coefficients pour former plusieurs types de correcteurs (il lui suffira de placer certains coefficients à 0). On précise aussi que ce sera dans la phase de qualification du correcteur PID qu'il faudra déterminer le ratio entre Pixels et unités moteurs :

- P : u = up
- PI : u = up + ui
- PID : u = up + ui + ud

Cette implémentation ne présente pas de complexité particulière, et fera l'objet d'une classe spécifique (voir partie II.4 sur la conception).

II.3.5. Synchronisation capteur/actuateur

Pour permettre l'asservissement de la position faisceau, il faut donc venir lire à période régulière une image sur le device LIMA sous-jacent.

Toutes les N lectures d'image un calcul de centroïde faisceau est effectué (sur la moyenne des N images acquises par le processus de calcul). Dans le cas où le faisceau est présent, s'il se trouve dans la warning zone définie par l'utilisateur, un calcul d'écart est effectué entre la consigne utilisateur et la valeur de centroïde relue. Si cet écart est non nul, il est injecté dans

le correcteur PID. En sortie, le correcteur PID nous retourne une nouvelle erreur, et cette erreur sera envoyée sur la consigne du moteur correspondant.

Au total, pour effectuer cet asservissement il faudrait donc être capable de communiquer à trois devices à partir de notre device, un device LIMA et deux devices GalilAxis.

Si nous essayons de formaliser cette approche, nous obtiendrons un diagramme de séquence tel que Figure 31 :

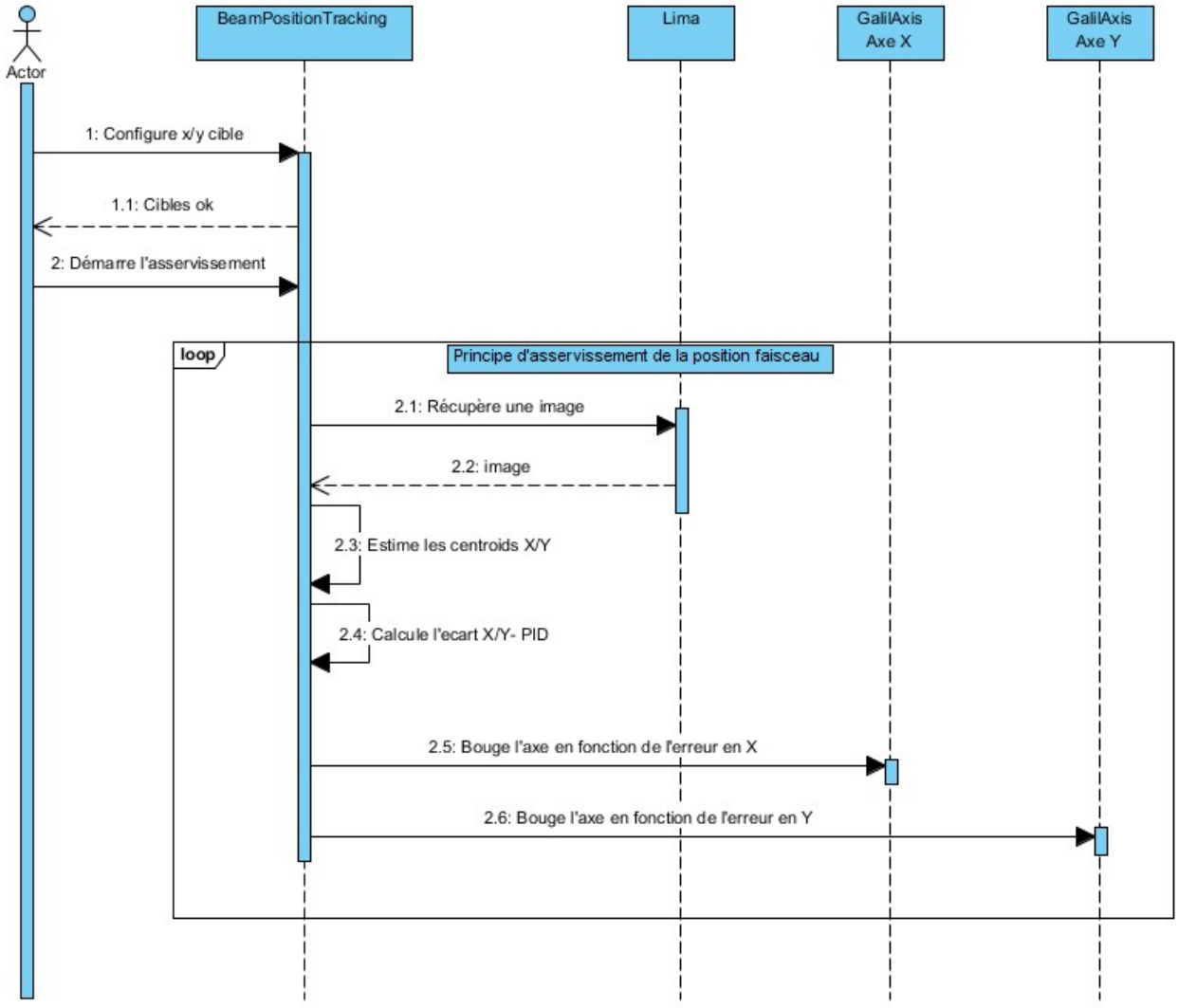


Figure 31 : Synchronisation Capteur/Actuateur

II.4. Conception

II.4.1. Considérations de conception

II.4.1.1. Modularité/Généricité

Comme je l'ai décrit dans le recueil de besoins utilisateur, je dois concevoir une application « hautement générique », capable de fonctionner avec différents types de capteurs (voir partie I.4.1 sur les premiers capteurs envisagés) et d'actionneurs (les différentes motorisations interfacées sur TANGO). Le but étant de pouvoir proposer ce service à un plus grand nombre.

Pour ce faire, je dois concevoir cette application de manière à ce qu'elle puisse intégrer ces nouveaux acteurs (moteurs et capteurs) au fur et à mesure des besoins exprimés à SOLEIL. On considère en effet ne pas savoir à l'avance quels seront les acteurs engagés dans de futures utilisations.

Je peux m'inspirer des patrons de conception du GoF (Gang of Four, d'après le site Wikibooks, Patrons de conception/Patrons du « Gang of Four »). Pour répondre à cette problématique de généricité, j'ai choisi de m'appuyer sur le patron « Stratégie » (Figure 32). En fonction des acteurs mis en œuvre dans l'asservissement, une implémentation différente de l'interface correspondante sera employée.

Le Patron Stratégie:

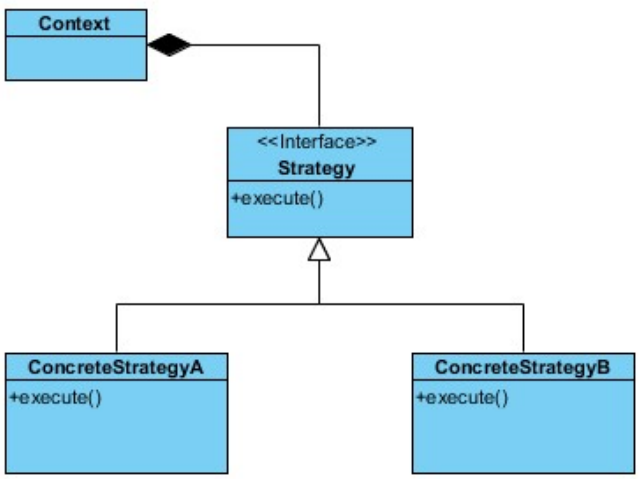


Figure 32 : Patron Stratégie (d'après le site Wikibooks, Patrons de conception/Patrons du « Gang of Four »)

Je peux donc utiliser cette approche pour représenter nos capteurs et nos actuateurs dans ce système. Il s'agit alors de définir une interface, un contrat à réaliser, pour chacun d'entre eux. De cette manière je m'assure que si de nouveaux capteurs sont nécessaires, il suffira d'écrire une nouvelle implémentation de cette interface (comme par exemple un nouveau type de moteur), au lieu de modifier le code source de l'application, nous parlerons ainsi de « plugins ».

Je m'assure aussi que dans le cas d'un système fonctionnant avec deux translations différentes (deux translations représentées par deux classes de device différentes), il suffira de choisir l'implémentation adéquate en fonction de l'axe. La stratégie correspondante sera en fait chargée au lancement de l'application, en fonction des configurations renseignées. Nous verrons par la suite que le code propre aux acteurs spécifiques pourra être encapsulé dans une librairie, cette approche sera donc modulaire.

II.4.1.2. Rôles identifiés

Pour que ce système soit plus simple de mise en œuvre, je décide de le découper en deux principaux devices collaboratifs (voir partie II.3.2.2), deux parties fonctionnelles indépendantes. Un device sera responsable de mettre en œuvre les translations et le deuxième sera responsable des calculs de positionnement et de coordonner l'asservissement.

De cette manière, je peux plus aisément décrire l'état du système d'axes et ce indépendamment du calcul d'asservissement à proprement dit. Le but ici est de fournir à l'utilisateur un retour direct sur l'état des axes mis en jeu. Il sera aussi plus simple d'agir directement sur ces derniers, même si l'asservissement est lancé en cas d'incidents. Ensuite, les déplacements mis en jeu dans ce genre de système sont extrêmement critiques, les miroirs embarqués font en effet l'objet de réglages très poussés, il faut absolument éviter de les dérégler. Si un envoi de positionnement n'est pas correct et qu'aucune sécurité de déplacement n'a été définie sur le device correspondant au déplacement par la ligne de lumière pour une raison particulière, c'est l'ensemble de la ligne de lumière qui peut être compromise. Je pourrai donc me servir de cet étage supplémentaire pour forcer l'utilisateur à définir les valeurs min et max de déplacement par exemple.

Pour finir, pendant la phase de déploiement et de tests sur ligne, cette approche permettra de mettre en œuvre notre application étape par étape.

Je décide de nommer le device responsable des axes moteurs : **ActuactorSystem**, le device responsable des calculs et de la coordination : **BeamPositionTracking**.

II.4.1.3. Réactivité

Pour fournir une application réactive, c'est-à-dire une application répondant aux meilleurs délais à l'utilisateur, il parait nécessaire de séparer la couche métier de la couche service.

Je décide donc de mettre en place pour chacun des devices (BeamPositionTracking et ActuactorSystem) un thread indépendant, c'est en fait dans ces threads (ou tâches) de calculs que seront embarquées les logiques métiers de nos devices. L'étage device server n'aura alors pour rôle que de fournir les résultats à l'utilisateur et lui permettre d'agir (à travers l'écriture d'attributs ou l'envoi de commandes) sur le modèle.

Si nous faisons l'analogie avec les patrons de conception du GoF, nous nous rapprochons ainsi du modèle MVC (Modèle Vue Contrôleur). Le device sera en fait constitué de la vue et fera aussi office de contrôleur sur notre modèle, qui sera ici notre tâche de calcul.

Par exemple pour le device BeamPositionTracking, c'est dans cette tâche que seront effectués l'ensemble des calculs concernant le centroïde faisceau et l'estimation d'un nouveau positionnement. Dès que nécessaire, le device pourra alors demander à cette tâche les dernières valeurs calculées.

Les threads de calcul et les devices servers étant séparés, l'utilisateur pourra à n'importe quel moment effectuer une requête non bloquante sur le device server. Cette architecture sera aussi intéressante s'il existe plusieurs clients utilisant cette application, la montée en charge n'impactera pas la couche métier.

II.4.1.4. Maintenabilité/Evolutivité

Pour concevoir cette application je peux m'appuyer sur des exemples de développements de devices réalisés à SOLEIL, je prends aussi rendez-vous avec Florent Langlois, mon contact dans le groupe ICA pour connaître les bonnes pratiques de conception.

Une des principales considérations lors de la phase de conception est de fournir une application maintenable et évolutive à moindre coûts. Pour aider à produire une telle application je pourrai m'appuyer sur une librairie C++ couramment utilisée par les développeurs à SOLEIL : YAT4TANGO. Cette librairie a été développée à l'origine en interne par Nicolas Leclercq, un développeur C++ à SOLEIL, elle est aujourd'hui utilisée dans la plupart des développements de devices servers TANGO. Le but de cette librairie est d'une part de limiter les dépendances d'un device vers d'autres librairies, mais aussi d'éviter de réécrire des fonctions déjà développées, elle facilitera la phase de développement. Cette librairie propose en effet un ensemble de fonctions adaptées aux devices TANGO. Nous nous en servirons principalement pour mettre en place nos tâches de calculs et nos plugins. Cela évitera de développer des composants trop spécifiques et donc plus difficile à maintenir par des développeurs tiers. Si on prend l'exemple des plugins, il sera en effet plus cohérent de les coder d'après une architecture connue, cela permettra de futurs développements à moindre coût.

D'après les exemples sur lesquels j'ai pu m'appuyer pour avoir une idée précise de la conception, je retiens donc principalement deux classes de la librairie YAT4TANGO, les classes DeviceTask et DevicePlugin.

La classe DeviceTask : Comme décrit dans les considérations précédentes, il nous faut fournir une application réactive où la logique-métier se trouve dans un thread différent de celui du device server, on peut aussi parler de « *Subsystem Manager* ». DeviceTask nous fournit un modèle de thread simple de mise en œuvre, basé sur le traitement d'une file de messages. Le développeur n'a pas à se soucier des détails d'implémentation de la classe thread de la librairie standard C++. En effet, il suffira de déclarer des messages (exemple : start_acquisition, stop_acquisition) dans cette classe et d'implémenter la fonction abstraite process_message, héritée de la classe DeviceTask. Cette méthode sera appelée périodiquement dès le lancement de la tâche via l'instruction « go », elle permettra de

traiter les messages poussés en file dans l'ordre d'attente (FIFO : First in First Out) via l'instruction post, de manière synchrone ou asynchrone.

La classe DevicePlugin : Comme nous l'avons vu, l'application sera modulaire et composée de différents types de plugins, des plugins pour représenter les moteurs, et des plugins pour les capteurs. Leur rôle sera d'encapsuler le code spécifique lié aux moteurs et capteurs à utiliser dans BeamPositionTracking. Il s'agira de définir une interface pour décrire chacun d'entre eux.

Le fait d'hériter de la classe DevicePlugin nous permettra de déclarer des propriétés et des attributs TANGO spécifiques dans ces plugins. Ces propriétés et attributs seront alors automatiquement accessibles en lecture et écriture si besoin à partir du device dans lequel le plugin sera chargé. Si l'on prend l'exemple du plugin correspondant au device LIMA, il faudra en effet déclarer un attribut spécifique : Image_threshold (valeur de seuil sur l'image). Cet attribut ne sera pas forcément utile pour d'autres capteurs (comme les XBPM qui n'ont pas de seuil à effectuer sur une image).

Pour utiliser ces plugins, il faudra être en mesure de charger la librairie correspondante au moment de l'initialisation du device (c'est le rôle de la classe PluginManager), chaque plugin une fois compilé fera en effet l'objet d'une librairie indépendante au device. Ensuite, pour qu'il soit connu du monde TANGO, il faudra nous servir de la classe DevicePluginHelper afin de l'instancier. Je peux m'appuyer sur des exemples de référence en termes de développement à SOLEIL pour mettre en œuvre ces plugins.

Ces différents choix de technologies vont influencer ma conception, il me faudra en tenir compte dans les étapes suivantes. Je peux d'ores et déjà donner le schéma de fonctionnement global de l'application, voir Figure 33.

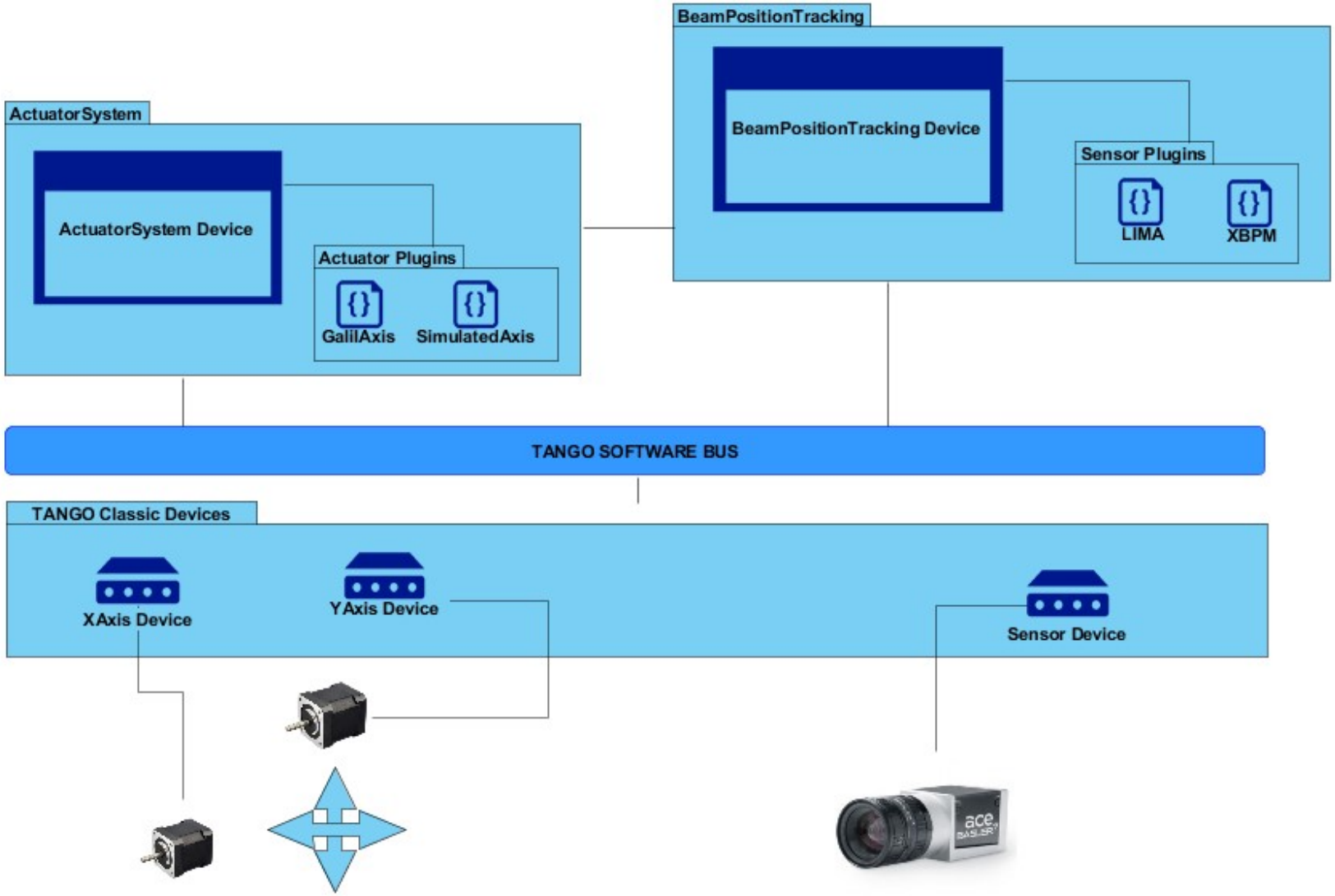


Figure 33 : Fonctionnement schématique de BeamPositionTracking

II.4.2. Design d'ensemble

Comme décrit en partie II.1.3 concernant le choix des outils de gestion de projet, j'utiliserai le formalisme UML pour décrire le fonctionnement de ce système avant de commencer à le développer. A l'issue de cette étape de design, j'organise une présentation avec les consultants de la phase de réalisation de la partie fonctionnelle (voir partie II.1.1, Tableau 1 : Matrice des responsabilités du projet BeamPositionTracking). L'objectif est d'obtenir leur retour afin d'améliorer mon design en fonction de leurs remarques éventuelles.

Ensuite je pourrai m'appuyer sur ce formalise UML pour pouvoir communiquer au client. Ce formalisme est en effet accessible pour un non expert en développement logiciel, il me permettra de recueillir ses impressions et de valider avec lui le lancement du développement à proprement dit. Pour finir, ce design permettra d'encadrer mon développement pour être plus efficace et pour fournir un code de bonne qualité.

Nous nous limiterons toutefois à trois types de diagrammes de la standardisation UML pour décrire ce système, le diagramme de classe, de séquences et d'états.

II.4.2.1. Diagrammes de classes

En m'appuyant sur le principe applicatif et sur les considérations de conception exprimées précédemment, je suis désormais capable de proposer un diagramme de classe pour notre système. On rappelle que ce dernier sera composé de deux devices collaboratifs : ActuatorSystem et BeamPositionTracking.

ActuatorSystem : Comme nous l'avons décrit, ce device va permettre de contrôler les deux axes ayant un rôle dans l'asservissement, le device BeamPositionTracking s'appuiera sur ce dernier pour déplacer le faisceau. L'utilisateur pourra aussi directement utiliser ce device dans une phase de paramétrage (optimisation de la position de départ par exemple), Il sera donc possible de vérifier les états des axes et leurs positions mais aussi de les déplacer ainsi que de les arrêter au besoin de manière indépendante.

Ce device permettra notamment à l'utilisateur d'assurer qu'une limite de positionnement ne pourra pas être dépassée pour chaque axe. Par mesure de sécurité, même si le device

responsable de la translation doit être configuré de manière à éviter tout risque (les butées physiques ne sont pas toujours suffisante, il faut pouvoir limiter les valeurs min et max de positionnement lors de la configuration de l'axe de manière « logicielle »), j'ai choisi d'ajouter un étage de vérification afin d'assurer au client qu'aucun risque de casse matérielle n'est possible et ce même si une erreur d'ordre algorithmique se produisait dans BeamPositionTracking.

Toujours par mesure de sécurité, je décide d'ajouter un booléen par axe qui devra représenter l'état de l'axe sous-jacent : « l'axe est-il prêt à être utilisé ? » (à l'initialisation du device ce booléen sera égal à faux). L'utilisateur devra manuellement vérifier que le ratio ainsi que la position de l'axe sont cohérents, il pourra alors valider ce booléen. Si cette vérification n'est pas faite ce dernier empêchera de déclencher l'asservissement.

On précise qu'il existe deux façons de considérer un déplacement moteur à SOLEIL, de manière relative ou absolue :

- Un **mouvement relatif** prendra en entrée une valeur en pas moteur ou une certaine grandeur physique si l'axe est configuré (c'est-à-dire qu'un ratio de conversion a été déterminé pour l'axe donné dans le device sous-jacent), l'axe effectuera alors un mouvement correspondant à sa position courante à laquelle on ajoutera la valeur entrée par l'utilisateur. On note que ce positionnement relatif passera généralement par un envoi de commande avec un argument (la valeur de déplacement).
- Pour un **mouvement absolu**, l'utilisateur entrera une valeur absolue sur l'axe, la valeur devra être comprise entre les min et max définis dans les paramètres du device responsable de l'axe. Le positionnement de l'axe se fera directement sur cette valeur. Ce positionnement est effectué via l'écriture de l'attribut position.

Si on reprend l'exemple des translations de type GalilAxis, l'écriture de l'attribut position fait donc l'objet d'un positionnement absolu (on entre une valeur à laquelle on veut que l'axe se positionne). Il peut néanmoins arriver que certains développements soient réalisés avec des positionnements relatifs. Ce sera alors le rôle du plugin correspondant que de masquer cette différence. J'ai choisi d'implémenter les deux types de positionnement dans le device ActuatorSystem.

Voir le diagramme UML suivant (Figure 34 : Diagramme de classes simplifié du device ActuatorSystem) résumant l'architecture du device ActuatorSystem.

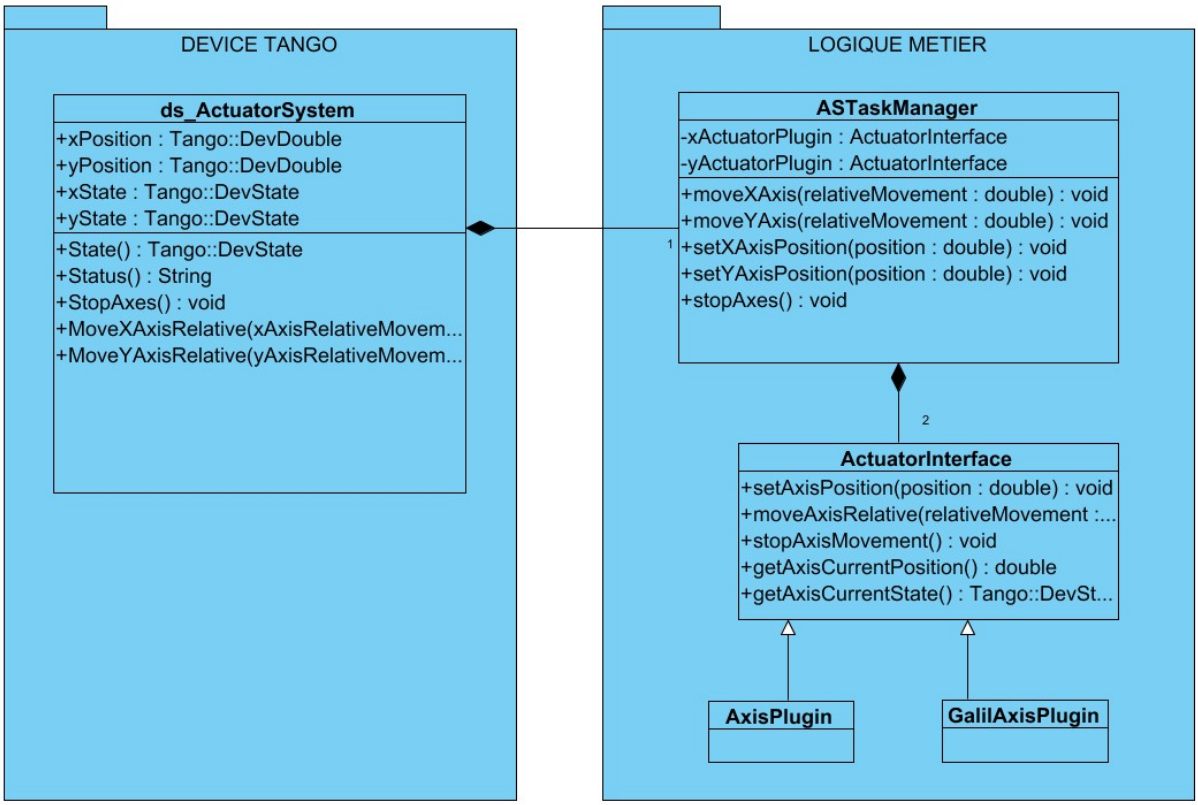


Figure 34 : Diagramme de classes simplifié du device ActuatorSystem

Voir en Annexe 5 : Diagramme de classes pour ActuatorSystem, le diagramme de classes complet du projet ActuatorSystem.

BeamPositionTracking : Le device BeamPositionTracking portera le cœur applicatif du système et déléguera la gestion des axes à ActuatorSystem. Son interface correspondra au cahier des charges décrivant la partie fonctionnelle défini en partie II.2.2. Côté fonctionnement interne, l'architecture choisie sera similaire à celle d'ActuatorSystem, utilisation d'un thread de calcul et mise en œuvre de plugins pour les capteurs (voir Figure 35 : Diagramme de classes simplifié du device BeamPositionTracking) :

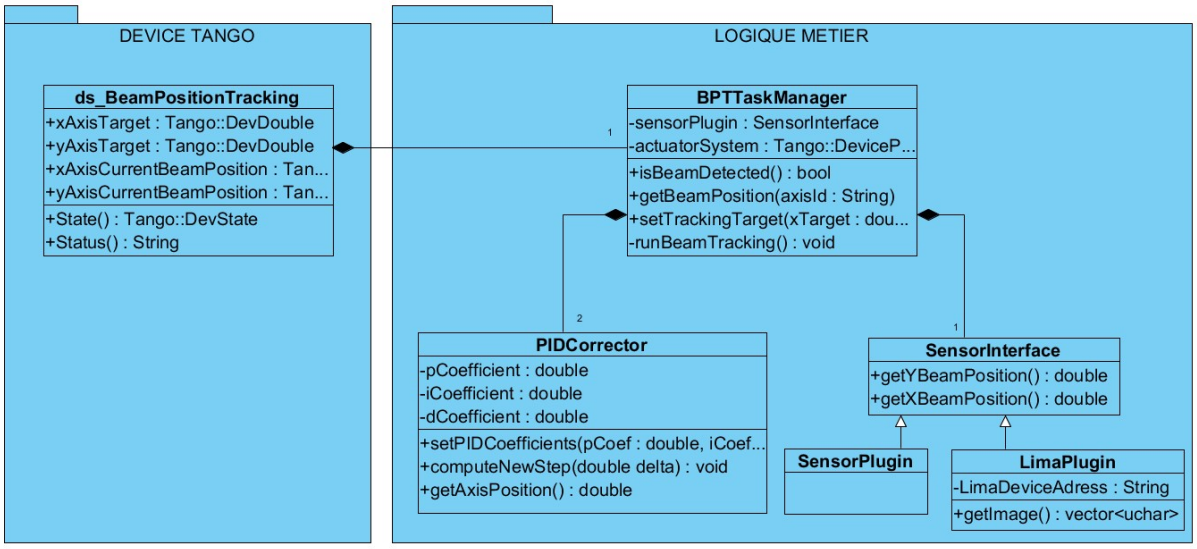


Figure 35 : Diagramme de classes simplifié du device BeamPositionTracking

Le diagramme de classes complet du device BeamPositionTracking est disponible en Annexe 6.

II.4.2.2. Diagrammes d'activités

Après avoir déterminé l'ensemble des acteurs responsables de l'asservissement dans cette application, je peux décrire leurs interactions. Pour ce faire, je peux m'appuyer sur des diagrammes de séquence. Le fonctionnement global de ce système peut être résumé à travers deux principaux diagrammes : le déplacement des axes motorisés et la gestion de l'asservissement.

Positionnement des axes motorisés : (Figure 36: Diagramme de séquences d'un déplacement simultané de moteurs) Dans le cadre d'un asservissement, on note que le client du device représenté sur le schéma par « Actor » sera le device BeamPositionTracking. Pour décrire le fonctionnement du device ActuatorSystem nous pouvons utiliser le cas d'un positionnement simultané sur l'axe X et Y. L'utilisateur pourra s'il le souhaite déplacer un axe indépendamment de l'autre. Comme on peut le voir sur la Figure 35, la tâche du device « ASTaskManager » sera responsable de la coordination des axes. De manière transparente lors d'un déplacement synchrone des axes X et Y, elle enverra sur le devices représentés par leurs plugins le déplacement souhaité. Ces appels se feront successivement (X puis Y), mais on ne va pas attendre que la première translation ait terminé son déplacement avant

d'envoyer la seconde. A partir du moment où au moins une translation affiche un état MOVING, l'état de ActuatorSystem sera aussi MOVING. Voir la machine à état de ce device (II.4.2.3) pour plus de précisions.

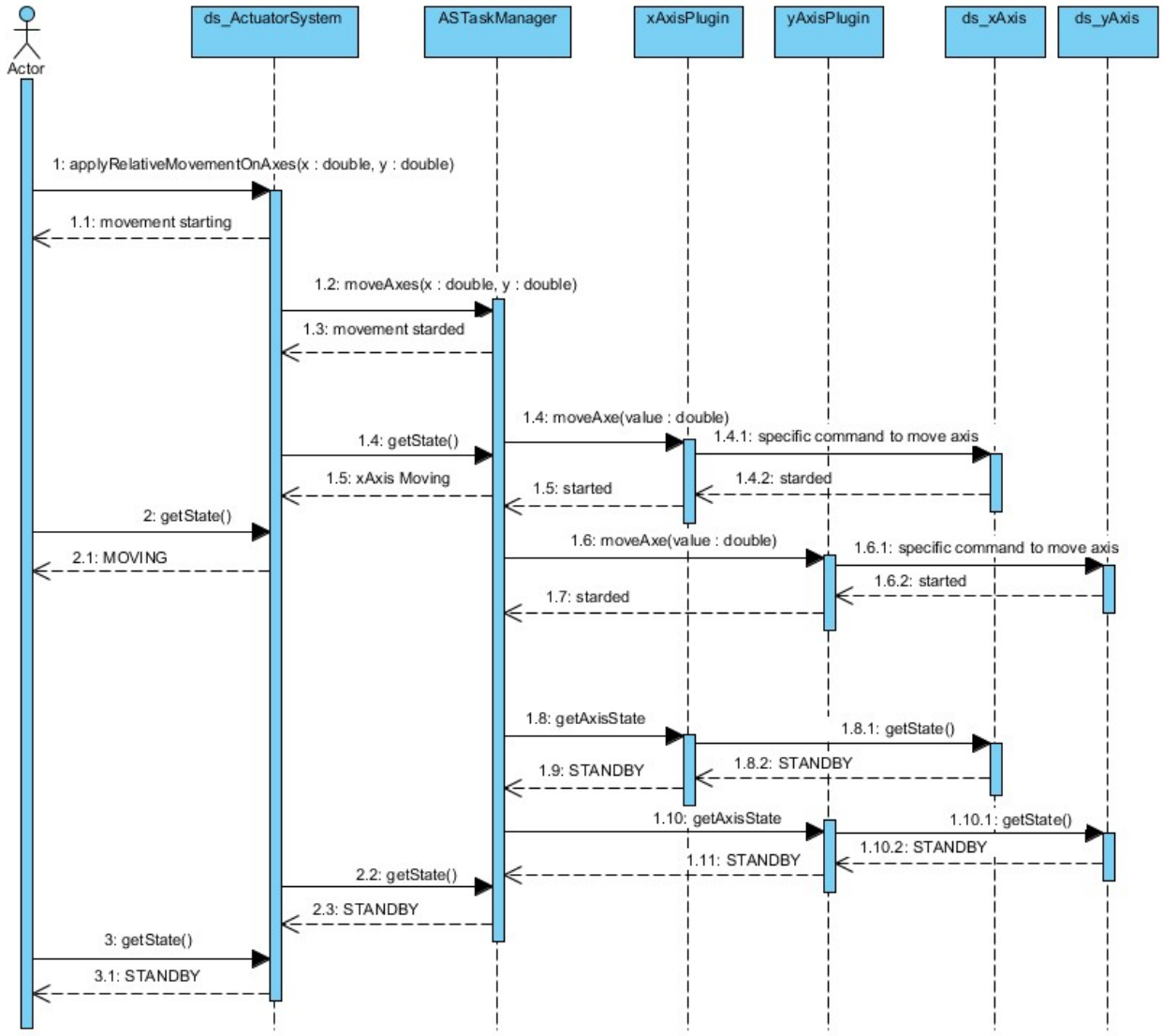


Figure 36: Diagramme de séquences d'un déplacement simultané de moteurs

Gestion de l'asservissement : (Figure 37 : Diagramme de séquences d'un asservissement de la position faisceau) Pour exprimer ces interactions, nous décrivons un scenario d'utilisation classique. Un utilisateur entre une cible sur laquelle il souhaite asservir le faisceau, il déclenche ensuite l'asservissement. On précise qu'il n'y a pas d'ordre particulier, l'asservissement peut être déclenché avant d'avoir précisé une cible, la valeur de la cible par

défaut sera alors la position courante du faisceau, rien n'empêche de modifier celle-ci une fois l'asservissement démarré.

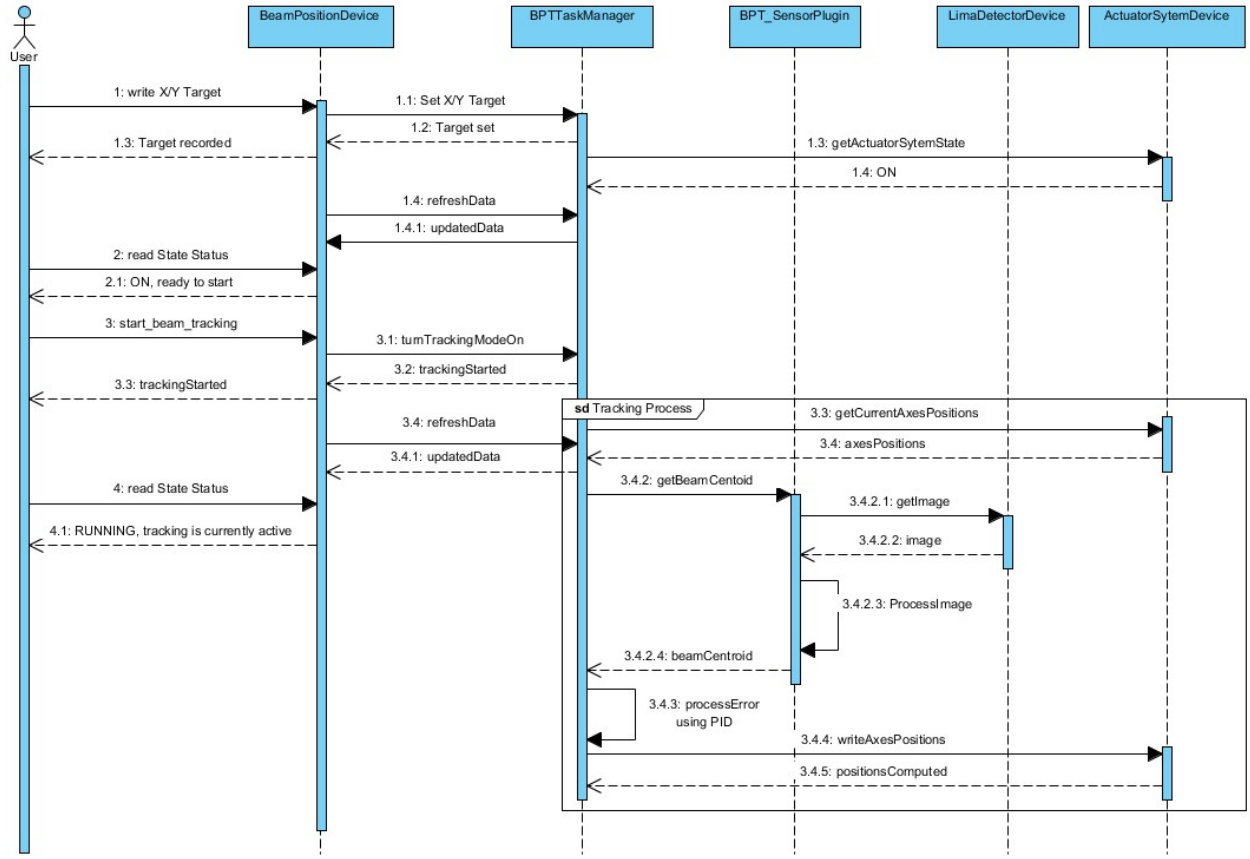


Figure 37 : Diagramme de séquences d'un asservissement de la position faisceau

Comme on peut le voir sur la Figure 37, certaines interactions entre le device et la tâche de calcule (BPTTaskManager) seront asynchrones pour ne pas être bloquantes. C'est notamment le cas du rafraichissement de la position faisceau ou de l'état du système. On peut en effet voir que de manière périodique, le device BeamPositionTracking va venir rafraichir ses données membres avec l'instruction refreshData. Par souci d'encombrement pour la représentation, on suppose que refreshData constitue un appel consécutif aux fonctions permettant de mettre à jour l'ensemble des données nécessaires au device (position du faisceau, position des axes, état du système etc…). Comme décrit dans les considérations de conception, je m'assure ainsi que le système ne sera pas ralentit par les demandes client. Cette approche est similaire sur le device ActuatorSystem, avec sa classe ASTaskManager.

Cette étape de conception me permet de me rendre compte qu'il faudra nécessairement s'assurer que les données retournées par la classe BPTTaskManager dans BeamPositionTracking et ASTaskManager dans ActuatorSystem ne seront pas accédées au même moment en lecture et en écriture, il faudra gérer donc gérer l'aspect transactionnel. Une des phases de développement sera alors de déterminer l'ensemble des sections critiques ainsi générées par ces collaborations.

Pour finir et par convention, les commandes effectuées sur le device tel que le déclenchement de l'asservissement ou son arrêt seront effectuées en synchrone. Le but est d'éviter l'envoi simultané de commandes qui pourraient poser des problèmes de consistance. On précise aussi que si l'asservissement n'est pas démarré, BPTTaskManager effectuera quand même le calcul de centroïde de manière continue, en s'appuyant sur le plugin du capteur associé (BPT_SensorPlugin), afin de retourner à l'utilisateur la position courante du faisceau à tout moment.

II.4.2.3. Diagrammes d'états

La dernière phase de conception consiste à décrire les états de ces devices. Comme nous l'avons décrit dans la présentation des device TANGO, les états des devices sont déterminants pour décrire l'état des systèmes sous-jacents. Ces diagrammes d'états me permettront aussi de communiquer au client, je pourrai m'appuyer sur ces derniers pour m'assurer que le fonctionnement du système sera compris par l'utilisateur.

La machine à état pour ActuatorSystem

Les états de ce device devront permettre de décrire à la fois l'axe X et Y mais aussi de permettre au client (le device BeamPositionTracking) de savoir si l'asservissement peut être déclenché ou non et pourquoi.

Je détermine les états suivants, en m'appuyant sur les bonnes pratiques (définition des états Annexe 1) pour décrire mon système de la manière la plus compréhensible possible.

INIT : Le device est en train de se construire, il initialise ses objets internes. C'est notamment dans cette phase qu'il construira sa tâche de calcul. L'étape INIT ne sera accomplie que

lorsque la tâche de calcul aura construit l'ensemble de ses objets, et que les proxies vers les devices sous-jacents auront étés initialisés. Cette phase permettra de nous assurer que les données de paramétrage (comme par exemple l'adresse des devices) sont correctes.

STANDBY : Le device est correctement initialisé, il n'effectue pas d'action particulière. Au moins un de ses axes n'est pas calibré ou ne retourne pas l'état ON. On rappelle que de manière périodique la tâche de calcul du device va venir lire les états et positions des axes sous-jacents en passant par les plugins appropriés.

ON : Les axes sont prêts à être utilisés, l'état remonté par les plugins correspondant est ON. Ils sont calibrés (l'utilisateur a validé les deux booléens correspondants), le device est prêt à jouer son rôle dans l'asservissement de la position.

MOVING : Au moins un des axes est en cours de déplacement.

FAULT : Dans le cas d'une erreur de communication, il faudra dans la plupart des cas réinitialiser le système avec la commande « Init ». Si l'utilisateur n'appelle pas cette commande, le système reste dans l'état FAULT. On précise que tout device est composé d'une commande « Init » par défaut, à son appel et de manière successive sont appelés le destructeur et le constructeur du device en question. Cela revient à arrêter et relancer ce dernier, ses données sont toutes réinitialisées.

ALARM : Si au moins un des axes est en dehors des limites définies par l'utilisateur, le device basculera dans un état qui ne permettra plus d'asservir la position faisceau. Il sera alors nécessaire de ramener l'axe à une valeur cohérente ou redéfinir les min et max autorisés.

Une fois cet ensemble d'états déterminé, je mets en place les transitions possibles. On note qu'il existe deux types de transitions : Evènementielle (exemple : un problème survient sur l'axe), ou provenant d'une action utilisateur (exemple : déplacer un axe). Une fois l'ensemble des transitions déterminées, il m'est possible de mettre en place le diagramme d'état du système (Figure 38).

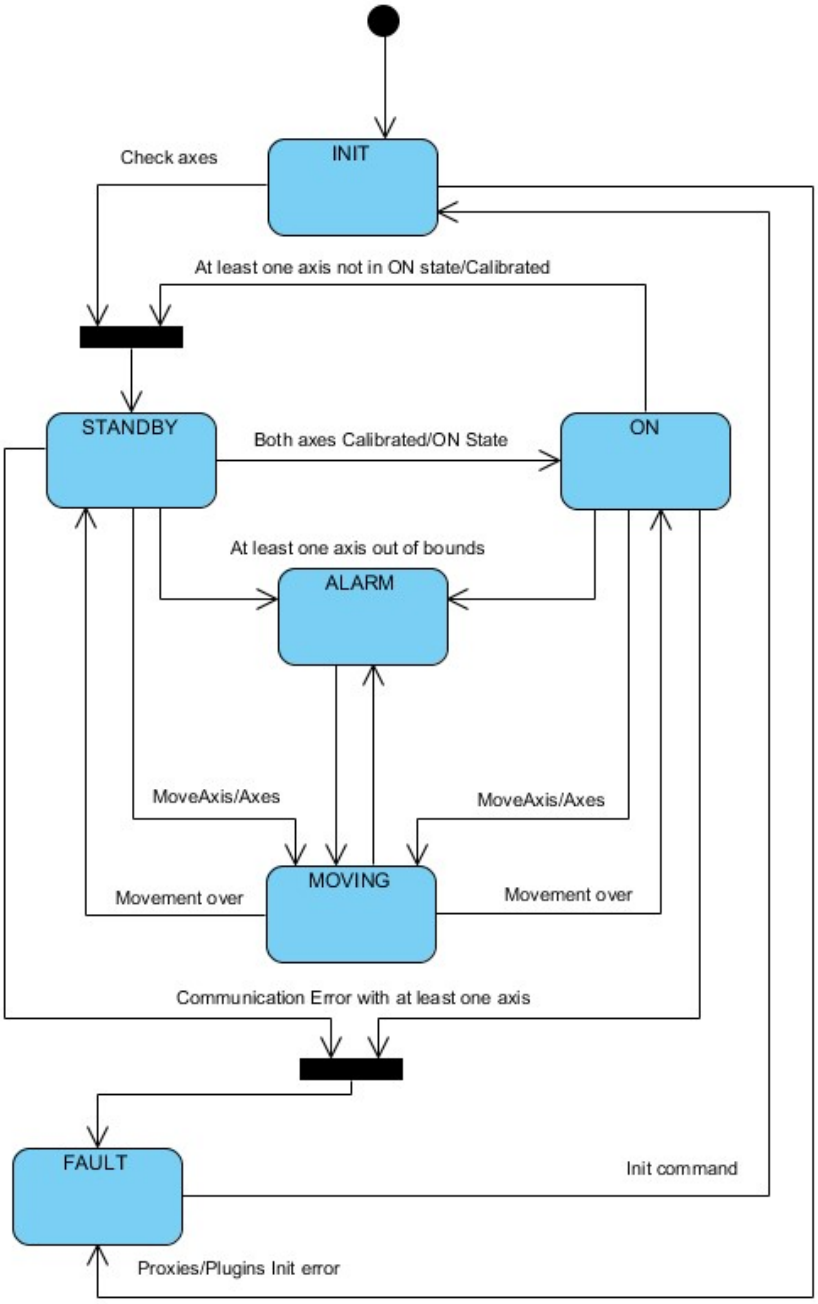


Figure 38 : Diagramme d'états du device ActuatorSystem

La machine à état pour BeamPositionTracking

De la même manière que pour ActuatorSystem, je commence par déterminer l'ensemble des états nécessaire à la description du système, certains états comme **INIT** et **FAULT** sont similaires à ActuatorSystem car ils ont le même objectif.

STANDBY : Le système n'est pas prêt à effectuer l'asservissement pour plusieurs raisons possibles. ActuatorSystem n'est pas ON, il faudra alors se reporter au « statut » de ce device

pour comprendre plus en détail pourquoi. Il est aussi possible d'être en STANDBY si le faisceau n'est pas trouvé ou ne se trouve pas dans la « warning zone », l'asservissement ne pourra alors pas être déclenché.

ON : Le système est prêt pour lancer l'asservissement, cela signifie qu'ActuatorSystem est nécessairement dans l'état ON (les axes sont prêts). En outre il faudra aussi que le faisceau soit détecté, c'est-à-dire que le calcul de la position du centroïde ait abouti, et que cette position faisceau se trouve dans la « warning zone » définie par l'utilisateur.

RUNNING : L'asservissement est démarré et en cours d'exécution. L'asservissement prendra fin après plusieurs évènements : L'arrêt utilisateur, la perte faisceau ou sa sortie de zone, ou un état du device ActuatorSystem différent de ON ou MOVING.

J'en déduis le diagramme d'état du système BeamPositionTracking Figure 39 :

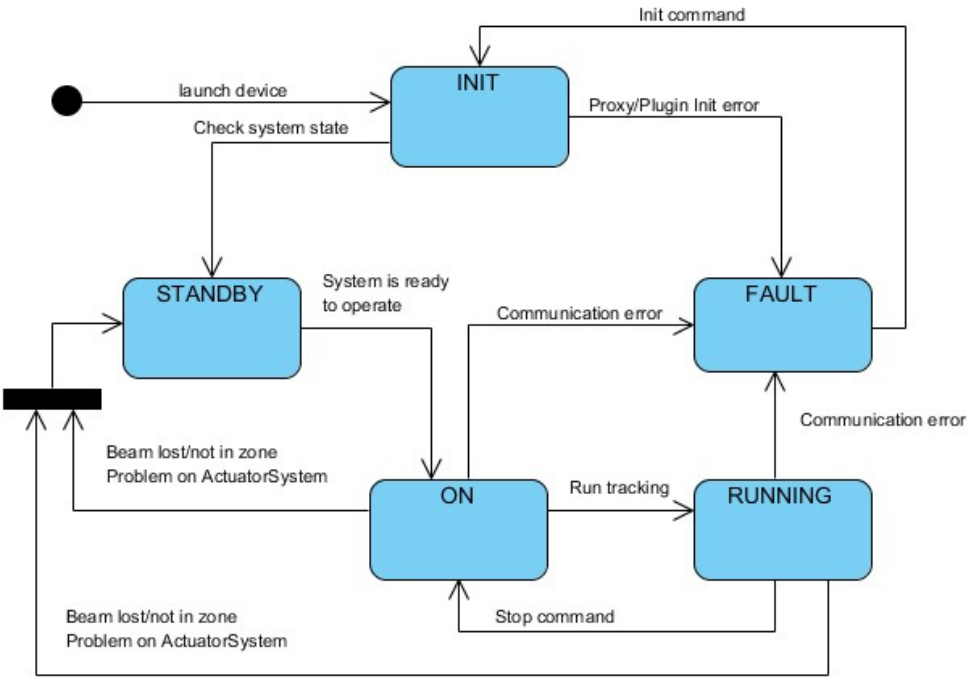


Figure 39 : Diagramme d'états du device BeamPositionTracking

On précise qu'en complément de ces états, les devices sont composés d'un statut. Le statut permet de décrire (généralement en une phrase) ce pourquoi le device se trouve dans son état courant, il a vocation à accompagner l'état pour aider l'utilisateur à comprendre la situation. Si par exemple un device se trouve en état FAULT, c'est dans le statut que le développeur devra informer au mieux la source d'erreur.

A l'issue de cette phase de conception, conformément à la méthodologie DSDM, j'essaie d'impliquer au mieux mon client. Le but sera d'échanger sur les choix d'architecture que j'ai pris. J'organise une réunion de présentation de concept avec Kadda Medjoubi et Florent Langlois pour présenter mon travail et justifier mes choix. Cette étape est essentielle et évitera de commencer la phase de réalisation sur des possibles incompréhensions des demandes métier.

II.5. Réalisation

II.5.1. Construction du projet

Une fois avoir clairement définit mon architecture, j'utilise POGO pour générer les classes des devices correspondantes. A ces classes de devices classiques viendront s'ajouter les classes annexes (voir les diagrammes de classe). La particularité des devices développés contrairement à la plupart des développements à SOLEIL est l'utilisation des plugins YAT4TANGO pour la réalisation de fonctions spécifiques. A la racine de chaque device, on trouvera un pom « général » permettant de lancer la compilation successive des devices et des plugins correspondants.

Pour chaque plugin et chaque interface de plugin, un sous répertoire sera nécessaire, il faudra en effet mettre en place des sous projets Maven (voir Figure 40 : Représentation de l'ensemble des fichiers générés pour la partie fonctionnelle). Contrairement aux devices, les plugins ne sont pas des exécutables mais des librairies, puisque nous travaillons sous linux, ces librairies seront de type Shared Object (SO).

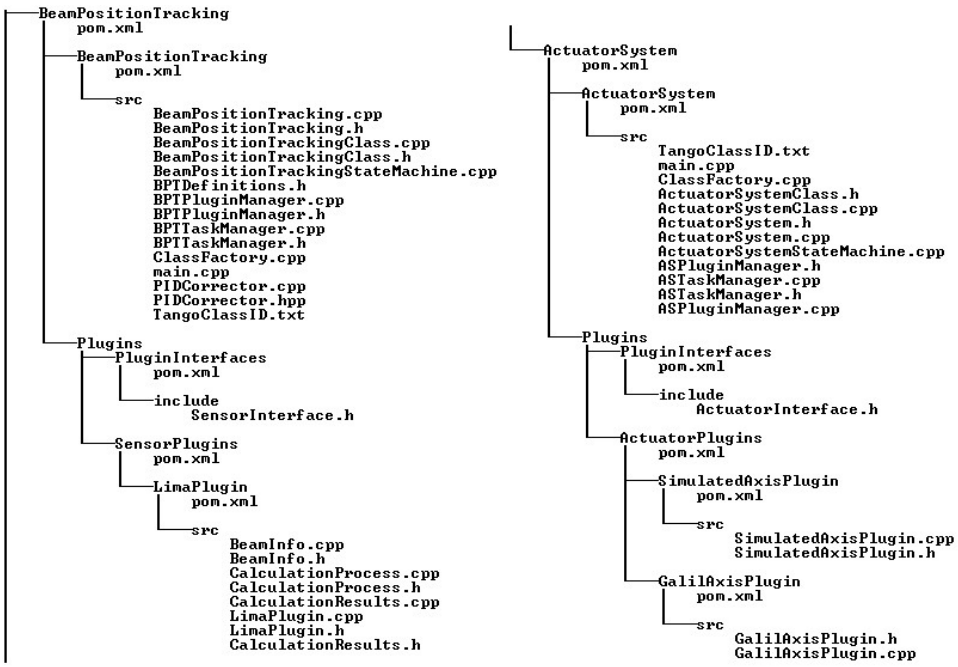


Figure 40 : Représentation de l'ensemble des fichiers générés pour la partie fonctionnelle

Après implémentation, le projet BeamPositionTracking représente au total environ 13 000 lignes de code, à travers plus de 50 classes C++.

II.5.2. Problèmes rencontrés

Traitement d'image

Pour mettre en place le traitement de l'image je me suis basé sur une image de tests provenant du device LIMA, avec son plugin Simulator, comme décrit dans le principe applicatif. J'ai commencé mon développement en me servant d'une image en 8 bits (le même type d'images fourni par le device LIMA mis en œuvre dans l'asservissement sur NANOSCOPIUM), c'est-à-dire une image constituée d'un ensemble de valeurs de type unsigned_char. Or il est possible qu'un détecteur nous retourne des images de type différent, comme par exemple une image 16 bits (format unsigned_short), j'ai donc dû modifier mon algorithme de traitement d'image pour qu'il puisse s'adapter de manière transparente aux différents types d'images provenant des devices LIMA sous-jacents. J'ai commencé par ajouter le traitement d'images 16 bits, en effet pour ce type d'application, il sera rare d'utiliser un détecteur plus résolu (les ajouts de traitement pour des images en 32 bits et 64 bits se feront dans un second temps). Pour ce faire, il m'a fallu m'appuyer sur la

valeur de « pixel depth » remontée dans un attribut du device LIMA. En fonction de la valeur de ce dernier, j'applique une conversion particulière avant de commencer à traiter l'image. Si le format de l'image ne correspond pas à un des cas traités, l'initialisation du device n'aboutira pas, l'utilisateur sera notifié via le statut du device de l'erreur. Pour simplifier le traitement, toute image en entrée du processus de calcul de centroïde sera convertie en 8 bits, nous n'avons en effet pas besoin d'une image hautement définie puisqu'elle sera ensuite binarisée. La valeur maximale d'une image 16 bits est de 65535, celle d'une image 8 bits est de 255. En regardant la valeur maximale de l'image d'entrée grâce à la fonction d'OpenCV « minMaxLoc », je peux déterminer s'il est nécessaire de ramener les valeurs de cette dernière à un maximum de 255 avant constituer la nouvelle image 8 bits. Avec cette approche de normalisation je maximise l'information autour de la valeur la plus grande de l'image. C'est en effet cette zone qui va nous intéresser, plus l'intensité est grande plus nous sommes proche du centre du faisceau.

Aspect transactionnel

Une autre complexité d'implémentation a été de déterminer les sections critiques liées à la collaboration des threads au sein de chaque device. En effet à partir du moment où au moins deux threads se servent d'une même donnée et où au moins un thread la modifie, il est nécessaire de la protéger. Il faut garantir que les lectures et écriture ne peuvent pas se produire au même moment. Si on ne sécurise pas suffisamment notre programme, son exécution peut conduire à un crash aléatoire (de type « segmentation fault », problème d'accès aux données), et ce genre d'erreur peut s'avérer très difficile à localiser. Comme nous l'avons décrit en phase de conception, les devices BeamPositionTracking et ActuatorSystem vont s'appuyer sur un autre thread pour calculer et mettre à jour leurs données membres. Du point de vue du device, périodiquement, un appel sera effectué sur l'objet correspondant à sa tâche pour lire les données. Il faut comprendre qu'en parallèle, ces données seront écrites et ce de manière totalement asynchrone par la tâche correspondante. Pour assurer qu'un seul thread ne lit ou écrit sur cette donnée, nous pouvons nous servir d'un sémaphore initialisé à un, c'est à dire qu'il n'autorisera qu'un seul thread à la fois. Nous pouvons en fait parler de mutex et dans un souci de maintenabilité, je choisis d'utiliser une implémentation du mutex de la librairie C++ standard, le mutex YAT. YAT étant une classe de facilitation de développement au même sens que YAT4TANGO,

excepté qu'elle couvre plus de domaines que le développement de devices TANGO. J'identifie en outre un autre type de section critique, les ressources externes. Comprendre par ressource externe les devices sous-jacents ou les plugins utilisés. Par mesure de sécurité et pour éviter tout crash potentiel, je décide de protéger les proxies utilisés dans les threads de calcul grâce au même mécanisme de mutex.

Qualité du code produit

Pour m'assurer de la qualité du code produit, je peux m'appuyer sur l'outil « CPP check ». Ce logiciel Open source sert à détecter des erreurs potentielles qu'un compilateur n'aurait pas pu remonter. Il permet notamment de voir en amont à quels endroits une fuite mémoire sera possible. Il permet aussi de corriger un grand nombre d'erreur de style et de nettoyer le code de toute variable inutile et optimiser leur portée.

II.5.3. Création d'une plateforme de tests

II.5.3.1. Enjeux des tests

En accord avec la méthodologie de gestion de projet agile mise en place, il me faut pouvoir développer et tester de manière itérative.

La problématique principale est que pour mettre en œuvre BeamPositionTracking, il faut pouvoir bénéficier d'un faisceau sur une ligne de lumière (ici ce sera notre client : NANOSCOPIUM) et déplacer des moteurs qui agissent directement sur des miroirs. La mise en œuvre de tests directs constituerait donc un risque pour la ligne, mon client m'explique que même si des sécurités hardware existent, il est toujours possible de provoquer des casses ou des dérèglements des positions miroirs suite à de mauvais positionnements. Au-delà du prix élevé du matériel mis en jeu, le temps nécessaire pour l'ajuster est particulièrement long et inclus de faire intervenir différentes équipes sur place, nous ne pouvons donc pas nous le permettre.

Ensuite, lors de la phase de tests, il ne sera pas possible de conduire d'expérience en parallèle sur la ligne en question. En effet, même si le principal objectif de BeamPositionTracking est d'asservir la position faisceau sur une cible donnée et donc de

maintenir ce dernier en place afin d'optimiser sa position sur l'échantillon, pour valider notre application il nous faudra le déplacer afin de nous assurer de son bon positionnement dynamique, sans quoi nous ne pourrons pas assurer son repositionnement suite à changement de cible ou à un déplacement d'un des éléments physiques constituant la ligne. Or le temps de faisceau et les expériences sont déterminés longtemps en avance sur une ligne (sur une année) en fonctions d'appels à projets. Il sera donc complexe de pouvoir tester notre application sur place. Mon client me propose tout de même d'organiser des sessions d'essais en fonction des disponibilités, mais à certaines conditions. Les sessions ne pourront pas dépasser 1H30, elles devront être réalisées entre Février et Mars (pour être en accord avec leur planning), et je devrai m'assurer en amont qu'aucune erreur de positionnement n'est possible, afin d'évoluer en toute sécurité.

Pour répondre à ces exigences, il me faut donc constituer une plateforme de tests. Cette plateforme devra s'approcher au mieux des conditions expérimentales de NANOSCOPIUM.

II.5.3.2. Simulation du domaine

J'ai accès à un serveur linux servant au groupe alignement métrologie de serveur de contrôle. L'ensemble des composants logiciels nécessaire au fonctionnement de TANGO y est déployé. J'ai l'autorisation de me servir de ce serveur comme d'une base d'essai pour valider cette application.

Dans la phase de développement j'ai eu recours au device LIMA avec son plugin Simulator. Comme décrit précédemment, ce plugin va fabriquer une image d'un faisceau « parfait ». J'ai donc pu dans un premier temps valider le calcul du centroïde du faisceau. Mais pour simuler le domaine, c'est-à-dire nous rapprocher au mieux d'une utilisation de BeamPositionTracking sur une ligne, il me faut être capable de déplacer la position du faisceau simulé sans quoi le calcul d'asservissement ne pourra pas être testé. J'ai donc décidé d'apporter une modification au code source du plugin Simulator, responsable la fabrication de cette image.

Pour ce faire il m'a fallu récupérer le code de LIMA, à partir de GitHub. L'idée est d'ajouter au plugin spécifique deux attributs d'offset, un attribut offsetXPosition et un attribut

offsetYPosition, ayant tous deux pour unité le pixel. Par défaut le faisceau simulé a pour centroïde le centre de l'image. L'image produite étant de dimension [1024,1024], le centroïde par défaut a pour valeur [512,512] pixels. L'objectif étant de pouvoir déplacer le centre du faisceau simulé grâce à ces nouveaux attributs : Si par exemple offsetXPosition passe à 10, la valeur du nouveau centroïde sera alors de [522, 512].

En parallèle à cette modification, il m'a fallu créer un device représentant une translation moteur « fictive ». Le principe de fonctionnement de BeamPositionTracking étant d'agir sur ActuatorSystem, qui va lui-même agir sur des devices représentants des moteurs, pour tester au mieux l'ensemble et nous rapprocher du domaine, j'ai choisis de développer un device particulier : AxisSimulation. Ce device prendra le rôle du device GalilAxis utilisé sur NANOSCOPIUM. L'idée est de construire un device prenant en propriété l'adresse d'un device LIMA Simulator avec le nom de l'attribut offset (offsetXPosition ou offsetYPosition), et d'y ajouter un attribut position. A l'écriture de l'attribut position, au lieu de déplacer un axe moteur, ce device écrira (moyennant un ratio) sur l'attribut offset du device LIMA Simulator correspondant, ce qui aura pour effet de déplacer le centroïde faisceau. Etant donné que l'écriture d'un attribut est instantanée, pour simuler au mieux le domaine, j'ajoute une temporisation dans le device AxisSimulation. Lorsqu'ActuatorSystem enverra un positionnement sur AxisSimulation, ce dernier passera à l'état MOVING (pendant un certain temps) puis écrira l'offset position correspondant. Pour intégrer AxisSimulation à ActuatorSystem, comme pour n'importe quel autre développement, je dois écrire le plugin correspondant. Il me suffit donc d'implémenter l'interface ActuatorInterface, et d'ajouter au projet ActuatorSystem ce nouveau plugin.

II.5.3.3. Spécificités d'implémentation

Pour modifier le plugin Simulator j'ai tout d'abord dû comprendre son fonctionnement, j'ai donc pris rendez-vous avec les responsables de développement LIMA à SOLEIL, Florent Langlois et Arafat Noureddine qui m'ont expliqué où intervenir pour mener à bien cette modification. Comme tout plugin fonctionnant sous LIMA, il doit implémenter une certaine interface et répondre à un comportement défini. Lorsqu'une acquisition est démarrée à partir du device principal LIMA Detector, ce dernier viendra appeler la méthode

« execStartAcq » sur le plugin correspondant. Dans cette méthode est implémenté l'ensemble du code spécifique aux capteurs (appels spécifiques sur les API propriétaires). C'est aussi dans cette méthode que nous retournons les données acquises (via un pointeur) sur les détecteurs. Dans le plugin qui nous intéresse, il s'agit de fabriquer une image, c'est le rôle d'une classe spécifique : FrameBuilder. J'ai donc choisi de modifier cette classe pour ajouter la possibilité de modifier le centre de cette image, cela revient principalement à deux modifications :

Extraits des modifications la classe « FrameBuilder » :

Ajout d'une méthode pour fixer les offsets:

```
/*************************************************************//**
 * @brief Sets the x and y offset for a GAUSS mode
 *
 * @param[in] x_offset  a double
 * @param[in] y_offset  a double
 ****************************************************************/
void FrameBuilder::setXYOffsets(double x_offset, double y_offset)
{
  m_x_offset = x_offset;
  m_y_offset = y_offset;
}
```

On ajoute en fait deux variables membres correspondants aux offsets, et réutilisées par la suite lors de la construction de l'image par la fonction fillData via son appel à dataXY, effectué pour déterminer l'intensité de chaque pixel. Par défaut et pour éviter le plus possible de modifier le comportement de ce plugin, on initialise ces nouvelles valeurs à 0.

Modification du calcul d'intensité pour une coordonnée donnée :

```
/*************************************************************//**
 * @brief Calculates the summary intensity at certain point
 *
 * @param[in] x  int X-coord
 * @param[in] y  int Y-coord
 * @return    intensity  double
 ****************************************************************/
double FrameBuilder::dataXY( const PeakList &peaks, int x, int y )
{
  double val=0.0;
  PeakList::const_iterator p;

  for( p = peaks.begin( ); p != peaks.end( ); ++p ) {
    val += gauss2D(x, y, (p->x0 + m_x_offset), (p->y0 + m_y_offset), p-
>fwhm, p->max);
  }
  val *= (1 + m_grow_factor * m_frame_nr);
  return val;
}
```

La fonction Gauss2D retourne une intensité en fonction d'une coordonnée. Comme son nom l'indique, elle retourne une répartition gaussienne de l'intensité, c'est-à-dire que le pic d'intensité se trouvera au centre X et Y. Il suffit donc d'ajouter un offset entre la coordonnée en entrée de la fonction dataXY et celle de gauss2D pour simuler le déplacement du faisceau.

Après avoir implémenté cette modification et validé son fonctionnement, je l'ai présentée aux responsables LIMA. Ils ont convenu que cette nouvelle fonctionnalité pouvait avoir un intérêt pour d'autres problématiques et m'ont proposé d'intégrer de manière officielle cette dernière au projet LIMA.

II.5.3.4. Mise en œuvre des tests en labo

Une fois ces développements réalisés, il ne reste plus qu'à déclarer ces devices sur la base TANGO du labo Alignement de manière cohérente, voir Figure 41 :

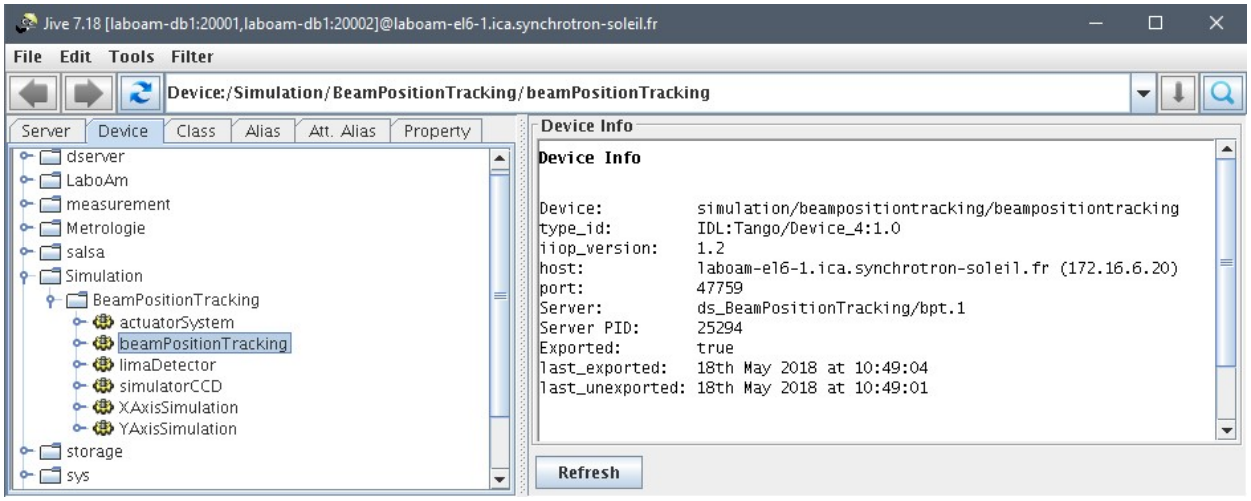


Figure 41 : Vue de la base de TANGO tests au Labo Alignement à travers l'application JIVE

Comme on peut le voir sur la Figure 41, on rassemble les devices concernant l'application BeamPositionTracking sous un répertoire dans la base de données TANGO : Simulation/BeamPositionTracking. On retrouve les devices ActuatorSystem, BeamPositionTracking, LIMADetector et son device spécifique SimulatorCCD,

XAxisSimulation et YAxisSimulation. Tous ces devices étant des devices Linux, nous pouvons les faire tourner sur le même serveur : laboam-el6-1.

Pour être le plus complet possible dans la conduction de mes tests, j'écris un cahier de tests fonctionnel, voir Annexe 7. Ce cahier spécifie chaque test fonctionnel devant être validé avant de pouvoir demander à mon client un test in situ. Ce cahier s'articule autour de chaque cas d'utilisation défini au préalable. Pour chaque scenario on décrit le test réalisé, la situation initiale, le comportement attendu ainsi que le comportement observé. Si on prend l'exemple du test de démarrage du device BeamPositionTracking en cas d'erreur, on écrit le test suivant :

Tableau 2 : Extrait du document de test de l'application BeamPositionTracking

Description du test	Démarrage du device BeamPositionTracking (cas d'erreur)
Etat initial	Au moins un des cas ci-après est avéré : • Le device ActuatorSystem n'est pas joignable • Le device LIMA n'est pas joignable • Le PATH du plugin n'est pas correct • Le type du plugin n'est cohérent
Comportement attendu	Le device démarre : ➢ il passe en FAULT et son « status » reflète la nature exacte de l'erreur
Comportement réel	Le device démarre correctement et affiche en fonction des erreurs générées la source dans son statut. [Labo Alignement – 18/01/18]

Je pourrai aussi me resservir de ce cahier (qui regroupe 30 tests fonctionnels dans sa version initiale) pour valider sur la ligne que les résultats correspondent aux attentes. Ce document attestera de la qualité de l'application et certifiera au client final son bon fonctionnement, il me faut donc balayer l'ensemble des cas d'utilisation possible.

II.6. Tests et validation sur NANOSCOPIUM

II.6.1. Itérations sur les tests

Après avoir validé l'ensemble des tests fonctionnels en laboratoire de test, je contacte mon client pour rendre compte de mon état d'avancement et le notifier que je suis prêt à réaliser des tests sur la ligne. Nous convenons ensemble de plusieurs sessions de tests pour

l'application. J'organise en amont les tests à réaliser en fonction des sessions afin d'optimiser ces dernières. Conformément à la méthodologie DSDM, je fais participer mon client, cette coopération me permettra d'adapter mon développement de manière dynamique, en fonction de ses remarques.

Préparation des tests, ajout d'un mode Simulation :

Pour conduire au mieux ces tests j'ai choisi de modifier mon code afin d'assurer à mon client qu'aucun mouvement ne pourra être déclenché par le système BeamPostionTracking avant d'avoir pu nous assurer qu'il n'y avait aucun danger. J'ai choisi de mettre en œuvre une sécurité à deux niveaux. Dans les deux device (BeamPositionTracking et ActuatorSystem) j'ajoute une propriété « DeviceMode », elle prendra deux valeurs possibles :

- NORMAL : Chaque device effectue ses tâches comme décrit dans le cahier des charges, rien n'est modifié, le fonctionnement standard est conservé.
- SIMULATED :
    - Dans BeamPositionTracking on s'assure qu'aucune consigne de déplacement ne peut être envoyée sur ActuatorSystem et ce à n'importe quel endroit du code. On ajoute aussi un mode Pas à Pas pour l'asservissement, avec une commande « AcknowledgeStep » : notre boucle d'asservissement effectuera le calcul de positionnement itérativement et attendra à chaque pas que l'utilisateur envoie cette commande pour continuer. Ce mode Pas à Pas nous permettra de vérifier les valeurs normalement envoyées sur ActuatorSystem au fur à mesure via des sorties d'écran.
    - Dans ActuatorSystem, on s'assure qu'aucun déplacement n'est réellement envoyé sur les axes sous-jacents. Dans ce mode simulé, les déplacements moteurs sont remplacés par des sorties d'écran permettant de s'assurer que les positionnements sont corrects.

Nous pouvons résumer les tests effectués sur la ligne NANOSCOPIUM à travers 3 principales sessions.

Session de tests N°1 (13/02/18) – 30 minutes :

Cette première session fût brève, je n'ai pu obtenir qu'une session de 30 min. Comme dit plus haut, la réelle complexité est de pouvoir dégager du temps entre des expériences utilisateur. Pour cette première session, mon client ne me permet pas d'agir sur la position des moteurs. La ligne était en effet dans l'attente d'une manipulation utilisateur importante et sensible le jour suivant, le moindre dérèglement l'aurait mis en péril. Ce premier test me sert donc principalement à vérifier la bonne installation de l'application, le bon fonctionnement de l'imbrication logicielle. J'utilise donc le mode SIMULATED de chacun des devices.

Pour gagner du temps, je déclare en amont (avant le test sur la ligne) les devices nécessaires en me connectant à distance sur NANOSCOPIUM. J'utilise le logiciel NX (NoMachine) pour me connecter sur le serveur « HADAR » et à partir de ce dernier, il m'est possible d'ouvrir une session SSH sur la ligne (chaque ligne est accessible à travers le serveur HADAR). Toujours pour optimiser mon temps de test, je compile l'ensemble du code, récupéré à partir de la ligne sur le serveur GitHub dans le répertoire : /home/ica/thiam/BPT/. Le jour du test il ne me reste plus qu'à lancer les exécutables correspondants.

Pendant cette courte session, je vérifie que les devices se lancent correctement (première partie du cahier de tests), je vérifie que les communications entre ces derniers sont correctes. Je peux aussi m'appuyer sur le développement réalisé par NANOSCOPIUM (application temporaire d'asservissement) pour vérifier que le centroïde calculé par BeamPositionTracking est le même mais je note une légère différence, il faudra attendre la deuxième session pour avoir le temps d'investiguer sur ce problème.

Session de tests N°2 (13/03/18) – 1 heure :

Après quelques recherches, je repère que la différence de centroïde entre l'application locale (le script Matlab temporaire) et BeamPositionTracking provient de la valeur entrée pour le threshold. Pendant cette deuxième session, on optimise cette dernière pour obtenir le même centroïde qu'avec l'application développée par NANOSCOPIUM (c'est-à-dire 20% du maximum - minimum de l'intensité).

Pour cette deuxième session mon client m'autorise à mettre en œuvre les translations. Nous choisirons tout de même avant cela de tester les devices via leur mode SIMULATED. On se rend alors compte, après avoir testé et validé que BeamPositionTracking fonctionnait correctement et donc de l'avoir passé en mode NORMAL, que les positions supposées être envoyées sur les axes à partir d'ActuatorSystem (toujours en mode SIMULATED) étaient toutes égales à 0. J'observe que cela est dû à un problème de type (Short au lieu de Double) dans le plugin GalilAxis (plugin qui n'a pas été possible de tester au laboAM puisque nous nous sommes servis d'un axe de type simulatedAxis). Je corrige en ligne et recompile ce dernier.

Le premier test complet mis en œuvre consistait à lancer l'asservissement de la position faisceau alors que ce dernier était déjà en position. Ce test n'a pas posé de problème, et a été validé.

Ensuite, nous déplaçons le faisceau manuellement puis nous déclenchons l'asservissement, pour observer comment l'application le ramène en position. Nous nous sommes rendu compte que la phase d'approche était correcte mais qu'ensuite, le faisceau oscillait autour du point de cible (ne convergeait pas exactement au point ciblé) de manière indéfinie.

Je vérifie que les paramètres entrés pour le PID correspondent à ceux que le client utilise déjà avec son application locale. N'ayant pas trouvé la source de l'erreur, il me faudra investiguer sur cette problématique pour la prochaine session. On note tout de même qu'à ce point, il ne reste que cette problématique à régler pour valider l'ensemble de la partie fonctionnelle de BeamPositionTracking.

<u>Session de tests N°3 (17/04/18) – 1 heure</u> :

Avant de conduire cette session de tests, j'investigue sur la problématique d'oscillation relevée lors de la session précédente. Je finis par me rendre compte d'une légère différence d'implémentation et d'utilisation de l'algorithme PID décrit par le client par rapport à celle que j'ai réalisée. La différence se trouve dans le calcul de la composante Intégral :

$$\text{I}: \ \text{ui}_{\text{k}} \ = \ \text{ui}_{\text{k}-1} + \ \text{K} \, . \frac{\text{Te}}{\text{Ti}} \, . \, \text{e}_{\text{k}}$$

A T=0 (au déclenchement de l'asservissement ou au changement de cible), Uik doit être initialisé à la valeur de la position moteur courante (au lieu de 0). La sortie de l'algorithme PID (ou PI) ainsi décrit doit être directement une nouvelle consigne de position moteur et non une différence (un nombre de pas) à envoyer. Tout au long de l'asservissement c'est sur la valeur de la position que l'asservissement sera effectué. Cette légère nuance m'a amené à modifier mon code, la subtilité est de faire en sorte que s'il n'y a pas de composante intégrale (l'utilisateur place I=0), le correcteur ne s'utilisera pas de la même manière, il devra alors retourner une différence (nombre de pas) au lieu d'un positionnement.

Cette troisième session de tests m'as permis de valider cette modification et donc de valider l'ensemble de la partie fonctionnelle de BeamPositionTracking.

Bilan des tests :

Ces tests m'auront permis de corriger les problèmes que je ne pouvais pas simuler en labo. Suite à ces derniers, nous pouvons valider l'ensemble de la partie fonctionnelle de BeamPositionTracking. On peut donc créer une première version de cette application, c'est à dire un premier livrable pour NANOSCOPIUM, en attendant de développer le reste (les plugins supplémentaires). Le client est satisfait et pourra continuer d'éprouver ce développement afin d'offrir un premier feedback opérationnel. Je propose donc de laisser les devices tels quels sur la ligne, en attendant de réaliser une IHM dédiée. Nous avons reporté l'ensemble des tests dans le cahier de tests, il sera par la suite sauvegardé dans la documentation de l'application.

II.6.2. Ajustements aux demandes clientes

Ayant impliqué mon client tout au long de la phase de tests, il a pu assister à la mise en oeuvre de BeamPositionTracking. En plus des ajustements fonctionnels que j'ai dû mettre en place pour répondre au cahier des charges initial, mon client m'a demandé plusieurs fonctionnalités supplémentaires. Fonctionnant dans une méthodologie agile, j'ai développé et livré ces nouvelles fonctionnalités au fur et à mesure de leur avancement.

Le mode «Fixed» :

Par mesure de sécurité, le client a souhaité ajouter un mode de fonctionnement au device BeamPositionTracking, le mode « fixed ». Dans ce mode, les seules actions disponibles à l'utilisateur sont de démarrer et d'éteindre l'asservissement. Il s'agira d'ajouter des nouvelles propriétés (valeurs fixées au démarrage) au device : Un booléen «FixedMode», les valeurs à utiliser pour la cible sur laquelle asservir le faisceau (sur X et Y) la valeur du threshold à effectuer sur l'image, la définiton de la warning zone (centre X et Y et rayon). Si FixMode est vrai, à l'initialisation du device, les attributs correspondants aux nouvelles propriétés seront initialisés avec les valeurs de ces dernières, il ne sera ainsi plus possible à l'utilisateur de les modifier via les attributs correspondants.

La commande d'auto calibration :

Pour faciliter une première utilisation de cette application sur une ligne de lumière n'ayant pas déterminé ses coefficients PID, l'idée est venue de mettre en oeuvre une commande de calibration linéaire. En amont de l'utilisation d'un correcteur PID, il serait envisageable dans certains cas d'utiliser un simple ratio entre pixel et déplacement moteur (un ratio linéaire). C'est à dire une valeur à laquelle on mutlitplierait l'écart entre la cible et la position courante du faisceau (en pixel) pour obtenir un déplacement moteur. L'algorithmie retenue pour déterminer ces ratios (un par axe) est la suivante : On commence par enregistrer la position courante du centroïde, ensuite de manière séquentielle (axe après axe), on déplace le moteur d'une valeur spécifiée en propriété « CalibrationStepNbXAxis» (pour l'axe X) et on relève à nouveau la valeur du centroïde. On peut ainsi déterminer le ratio correspondant à l'axe donné :

- Ratio = CalibrationStepNbXAxis / (newCentroid – oldCentroid)

Pour finir on écrit sur le device ActuatorSystem l'attribut correspondant au ratio linéaire de l'axe en question (xLinearRatio pour l'axe X), et pour chaque déplacement on multipliera la valeur à effectuer par ce dernier.

Les Alias :

Mon client me précise que les axes utilisés dans l'asservissement ne sont pas [X, Y] mais [X, Z], en effet par convention à SOLEIL l'axe verticale est appelé Z. Pour ne pas contraindre mon développement et pouvoir fournir une application hautement générique, je décide de mettre en place des alias. Je créé alors une nouvelle propriété dans le device BeamPositionTracking, « AxesAliases». Cette propriété sera composée de deux strings (chaines de caractères) séparées d'un retour clavier : la première chaine de caractère définira le nom de l'axe horizontal et la deuxième l'axe vertical. Ces valeurs seront ensuite utilisées dans l'initialisation du device pour changer la valeur des noms des attributs nécessaires (Target, Centroid etc…) par concaténation. L'utilisateur sera alors libre de définir les noms de son choix.

II.7. Réalisation d'une interface homme machine

II.7.1. Design général de l'application

Conformément à l'organisation choisie pour mener à bien ce projet, une fois avoir validé l'ensemble de la partie fonctionnelle, je commence le développement d'une interface homme machine (IHM) pour l'application BeamPositionTracking. Comme pour la partie fonctionnelle, mon client est Kadda Medjoubi de la ligne NANOSCOPIUM. On rappelle qu'aucune logique-métier ne sera embarquée dans cette interface, il nous suffira simplement de concevoir une application cliente des devices TANGO conçus précédemment. Il s'agira donc de développer la partie Vue du patern MVC (Modèle Vue Contrôleur). L'objectif de cette IHM sera simplement de faciliter l'usage de l'application BeamPositionTracking et de masquer à l'utilisateur final les différents devices TANGO mis en jeu. Ce « masque » permettra aussi de cacher les informations spécifiques « expertes » auxquelles un utilisateur non expert de la ligne ne devra pas avoir accès. C'est notamment le cas de l'écriture des coefficients PID. J'oriente mon design pour fournir l'application la plus simple possible.

Pour répondre au mieux au besoin client, je commence par designer les vues de l'application de haut niveau via un sketch (dessin de l'IHM), pour ce faire j'utilise le logiciel de

dessin « *Pencil* ». Ce prototype fera office de point de départ pour présenter l'orientation du design de l'application au client. Je choisis une application constituée de deux parties, deux différents panels (voir Figure 42 et Figure 43).

La première partie sera utilisée pour la phase de connexion et configuration de BeamPositionTracking. C'est notamment dans cette partie que l'utilisateur expert choisira quel device BeamPositionTracking il utilisera. En effet, on ne perd pas de vue qu'il est probable qu'une ligne de lumière se serve de plusieurs instances de ce device, pour par exemple asservir le faisceau à plusieurs endroits, il sera alors possible de lancer en parallèle plusieurs instances de cette application. C'est aussi dans ce panneau de configuration que l'utilisateur pourra vérifier que les propriétés (non accessible en écriture contrairement au device) définies sur les devices BeamPositionTracking et ActuatorSystem sont cohérentes. J'en déduis le panel suivant (Figure 42):

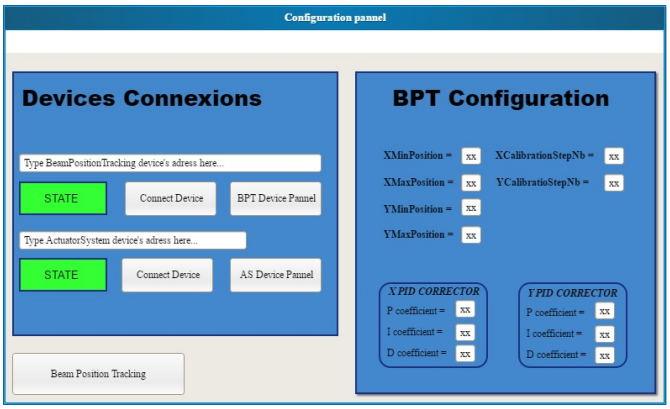


Figure 42 : Prévisionnel du panneau de configuration de BPT

Une fois avoir choisi les devices à utiliser, et vérifié que les propriétés de ces derniers sont correctes, l'utilisateur pourra se rendre sur le deuxième panneau qui correspond aux fonctionnalités de BeamPositionTracking. Nous appellerons ce deuxième niveau, le niveau de contrôle (Figure 43), il permettra à l'utilisateur de visualiser le faisceau (image binarisée) et de définir la cible sur laquelle il souhaite le positionner et l'asservir. Il sera aussi possible de voir les positions des axes ainsi que leurs états, ce qui pourra s'avérer utile en cas de problème.

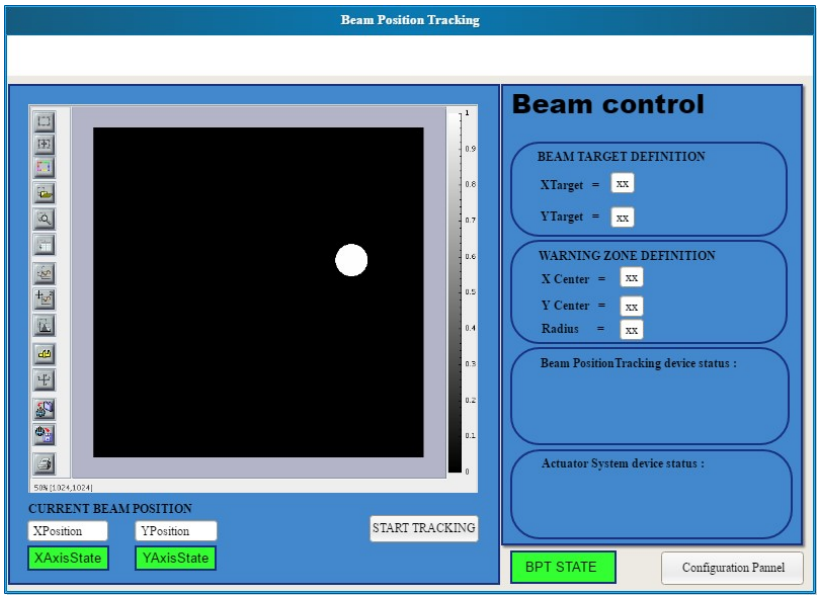


Figure 43 : Prévisionnel du panneau de configuration de BPT

J'ai présenté ce design à mon client qui l'a validé. Je me servirai donc de ces prototypes pour modeler l'IHM à réaliser, ils me serviront de support.

II.7.2. Choix de la technologie

Il existe un certain nombre de technologies disponibles pour créer une interface spécifique pour un device TANGO donné. En effet, des « Bindings » ont étés écrits pour permettre l'utilisation de logiciels commerciaux tel que GlobalScreen (logiciel superviseur de contrôle et d'acquisition de données développé par la société Ordinal), Matlab ou Labview. Ces logiciels sont souvent utilisés par les scientifiques eux-mêmes et permettent la mise en forme de données issues de leurs expériences d'une manière particulière, ils peuvent aussi servir à optimiser le contrôle d'éléments de leur ligne afin par exemple d'automatiser certains processus. Ces applications ne sont néanmoins pas factorisées, elles sont développées spécifiquement par chaque ligne et sont donc difficilement maintenable. En outre, ce type d'approche nécessite la mise en œuvre d'un environnement spécifique, c'est-à-dire l'installation d'un runtime particulier.

Pour favoriser l'approche MVC afin de constituer un client léger, étant donné que la logique applicative de BeamPositionTracking est exclusivement contenue dans les devices serveur,

ainsi que pour faciliter le déploiement de l'application graphique, il me parait plus cohérent d'écrire une application Java. Pour ce faire, il existe un framework permettant la visualisation de données et l'exécution de commandes TANGO : Le framework COMETE. De nombreuses applications ont d'ores et déjà été développées grâce à ce framework à SOLEIL. Je peux donc bénéficier de l'expérience de développeurs Java, j'organise une réunion avec Gregory Viguier et Raphael Girardot. Ils m'expliquent les principes de fonctionnement de cette API et comment l'utiliser à travers Eclipse (un logiciel de développement libre). Comme la plupart des IHM Java, il me faudra me servir du framework SWING (bibliothèque graphique pour le langage de programmation Java). L'API COMETE, est en fait basée sur l'utilisation de composants graphiques Java hérités de SWING, c'est-à-dire que tous les composants dont nous nous servirons pour afficher ou écrire des données sur un device TANGO implémentent la classe JComponent. Pour simplifier mon développement, j'ai écrit une classe spécifique me permettant de connecter tous les composants COMETE au monde TANGO : TANGOConnexion.

Pour chacun des types de composants SWING utilisés dans l'application, je créé une surcharge d'une fonction appelée connectAttribute tel que :

Extrait de la classe « TANGOConnexion »:

```
/*************************************************************
*  connectAttribute
*
*  For a WheelSwitch component
* ************************************************************/
public void connectAttribute(String tangoDataType, String deviceAdress, String
attributeName, WheelSwitch component, boolean metaData){
	ScalarCometeBox box = getBoxWithType(tangoDataType);
	if (box == null)
		return;
	TangoKey tangoKey = new TangoKey();
	TangoKeyTool.registerAttribute(tangoKey, deviceAdress, attributeName);
	if (metaData)
		box.connectWidget(component, tangoKey);
	else
		box.connectWidgetNoMetaData(component, tangoKey);
}
```

Il me suffit de passer par le TangoKeyTool pour enregistrer l'attribut à connecter en fonction d'une TangoKey. Ensuite, il faut utiliser une ScalarCometeBox dont l'implémentation dépendra du type d'attribut à connecter (nombre, chaine de caractère, booléen…) pour connecter notre objet COMETE à l'attribut distant.

II.7.3. Réalisation

Après avoir pris en main l'API COMETE, je développe les classes correspondantes au design préalable. L'application Java ainsi produite sera relativement simple puisqu'il suffira de créer les composants graphiques nécessaires à la réalisation des prototypes, elle ne comportera pas de logique métier. Je fais le choix de concevoir trois principales classes Java :

- BeamPositionTracking : Point d'entrée de l'application (constituée du «*main*» pour lancer l'application)
- ConfigPanel : Panel de configuration de l'application (Figure 44)
- ControlPanel : Panel de contrôle de l'application (Figure 45)

On note tout de même que pour pouvoir afficher une information cohérente en fonction des conditions d'utilisation, il faut mettre en place un thread de rafraichissement pour chacun de ces panels (nos deux classes ConfigPanel et ControlPanel implémenteront en fait java.lang.Runnable). De cette manière, après leur lancement (à l'appel de la méthode run) il sera par exemple possible d'afficher le bouton « Start Beam Tracking » si l'état du device sous-jacent le permet, le masquer sinon. En couvrant le plus de cas possible j'évite ainsi à l'utilisateur de générer des exceptions inutiles, le but étant de permettre une meilleure utilisation de cette application, la plus intuitive possible.

En parallèle des demandes formulées par le client, j'ajoute la possibilité de déplacer le faisceau en cliquant sur l'image à partir du panel de contrôle, pour plus d'interactivité. Il s'agit en fait de récupérer la coordonnée X et Y sur l'image au moment du clic et de l'écrire sur les attributs targets correspondants du device BeamPositionTracking.

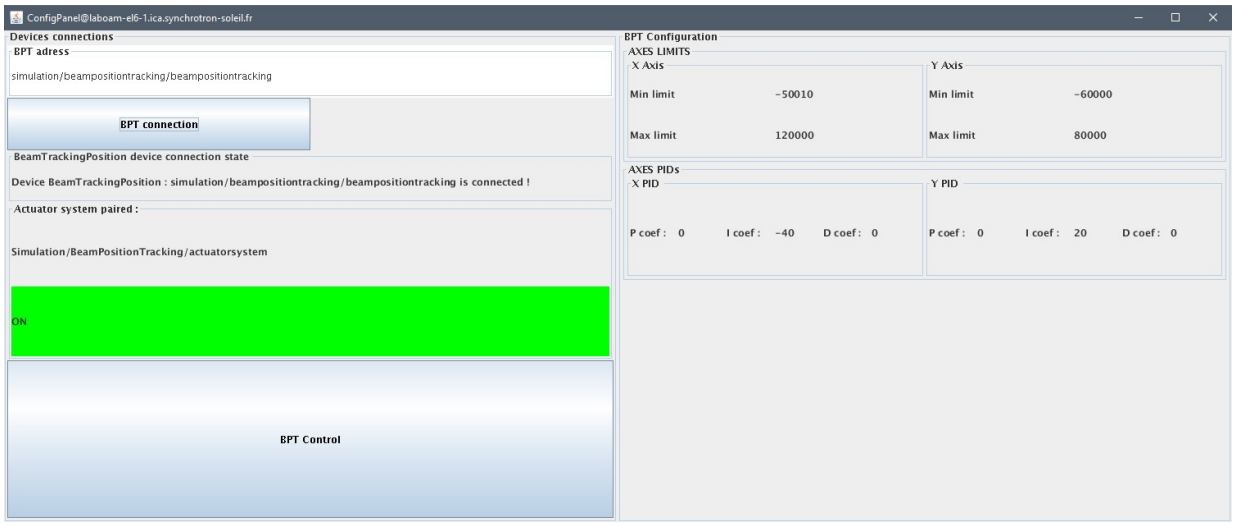


Figure 44 : BPT - IHM, le panel de configuration

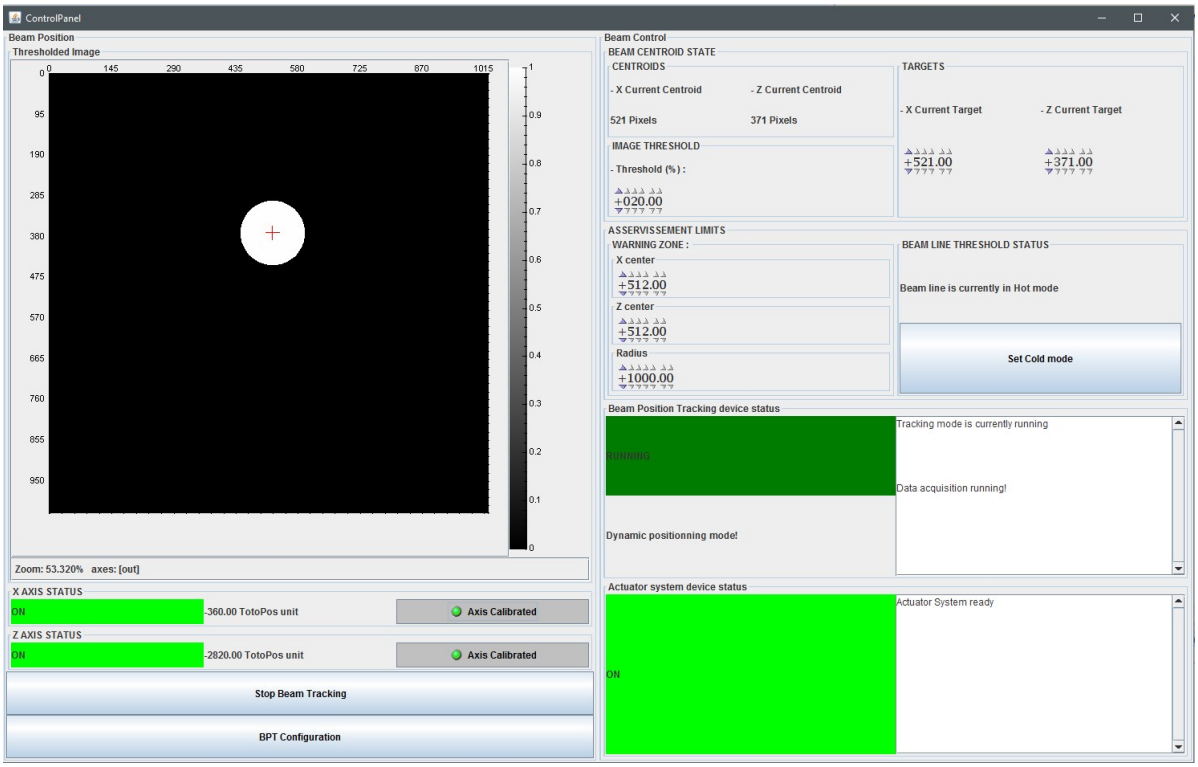


Figure 45 : BPT - IHM, le panel de contrôle

Une fois avoir développé cette interface, je prends rendez-vous avec mon client pour lui présenter le résultat sur sa ligne de lumière (ou les devices sont toujours actifs depuis la dernière session de tests). Cette IHM lui convient et fera donc l'objet d'un deuxième livrable. Comme pour les devices TANGO, je la laisse en production sur NANOSCOPIUM pour recueillir

les impressions de mon client au fur et à mesure de son utilisation et prendre connaissance des bugs éventuels.

II.8. Transition vers l'opération

Pour reprendre ma méthodologie de gestion de projet, je passe désormais à la phase III. Il s'agira en effet de m'assurer du bon fonctionnement du projet et ce à l'échelle de SOLEIL. Pour effectuer cette transition vers l'opération, il faut vérifier que le nécessaire logiciel et bibliographique (les documentations logicielles) soit correctement déployé.

II.8.1. Intégration continue

Pour une intégration complète de cette application à SOLEIL, il faut qu'elle fasse partie de notre système d'intégration continue. Le principe de l'intégration continue est de vérifier que chaque modification du code source ne produit pas de régression et donc de permettre la maitrise du changement. Nous serons en effet amenés à apporter des modifications à nos devices pour corriger d'éventuels problèmes ou les faire évoluer en fonction de nouvelles exigences. A SOLEIL, l'intégration continue repose uniquement sur l'automation de la compilation des projets, des études sont en cours pour aller plus loin : vers la mise en place de tests unitaires automatisés. Nous utilisons JENKINS pour ce faire, un logiciel Open Source pouvant se connecter à différents systèmes de gestion de version tels que Git, Subversion ou CVS. On note qu'à SOLEIL, nous n'avons pas encore la possibilité de nous servir de JENKINS pour un projet hébergé sur GitHub. Pour que mon application soit intégrée il me faut alors faire une demande de création de projet SVN (logiciel de gestion de version : Subversion), n'ayant pas l'autorisation de le faire moi-même, je formule cette demande à travers un ticket JIRA. De la même manière, pour l'IHM il faudra créer un autre projet sous SVN.

Ainsi nous créons une version initiale pour notre projet. Nos devices et application haut niveau seront « tagués » en version 1.0.1. Nous devrons pour la suite du projet créer des tags en fonctions des changements réalisés suivant le format release_X_Y_Z :

- Z : un changement léger
- Y : un changement important

- X : un changement lourd et impactant (exemple : dans le cadre de la modification de l'interface d'un device, tous les clients seront impactés)

Ces différents tags nous permettrons de retrouver facilement une version stable du projet en cas de problème. On note aussi que chaque tag devra être tracé et documenté à l'aide d'un ticket JIRA.

Pour chaque changement (nouvelle version spécifiée dans le fichier POM) comité dans nos projets, JENKINS va automatiquement tenter une nouvelle compilation (dans notre cas on aura configuré JENKINS pour qu'il compile les devices et l'IHM sur un serveur Linux). Si une erreur apparait, le développeur responsable du projet recevra un mail de la part de JENKINS avec les LOGs correspondants. On note que ce processus est le même lorsqu'une librairie tierce est modifiée ce qui permet de maintenir à jour le projet automatiquement.

II.8.2. Rédaction de documentation

Pour pérenniser le développement logiciel produit, j'accompagne mon projet de documentations techniques. Je joins aux projets SVN correspondants les diagrammes de classe UML. Le but est ici de fournir le plus d'information possible aux futurs intervenants. En effet, plus la documentation technique sera complète moins le coût de maintenance évolutive du logiciel sera élevé. Je me sers aussi de POGO pour générer automatiquement une documentation à partir du code lui-même. POGO se base en fait sur l'outil Doxygen, un logiciel open source permettant de générer une documentation à partir de code C++, Java PHP etc… POGO va en fait créer un certain nombre de fichiers HTML. On retrouve à partir d'un fichier index les différents composants d'un device à travers par exemple les fichiers Attributes.html, DevCommands.html ou encore Properties.html. L'outil POGO va générer tous ces fichiers à partir des commentaires entrés par le développeur lors du développement de chaque commande, attribut, état ou propriété. Le but est de fournir une documentation technique pour les devices TANGO produits (voir l'exemple Tableau 3 avec les attributs scalaires de BeamPositionTracking).

Tableau 3 : Attributs scalaires du device BeamPositionTracking

Scalar Attributes			
Attribute name	**Data Type**	**R/W Type**	**Expert**
computedTime: Time of the calculation in ms	DEV_DOUBLE	READ	No
xAxisTarget: Define xAxis target in pixels	DEV_SHORT	READ_WRITE	No
yAxisTarget: Define yAxis target in pixels	DEV_SHORT	READ_WRITE	No
xAxisCurrentBeamPosition: Read current beam position on x axis	DEV_SHORT	READ	No
yAxisCurrentBeamPosition: Read current beam position on y axis	DEV_SHORT	READ	No
warningZoneXCenter: Define the center of warning zone on X Axis	DEV_DOUBLE	READ_WRITE	No
warningZoneYCenter: Define the center of warning zone on Y Axis	DEV_DOUBLE	READ_WRITE	No
warningZoneRadius: Define the size of the warning zone	DEV_DOUBLE	READ_WRITE	No
xAxisRegulationThreshold: Represents the threshold of regulation activation on X axis. Under this value, no correction will be send !	DEV_USHORT	READ_WRITE	No
yAxisRegulationThreshold: Represents the threshold of regulation activation on Y axis. Under this value, no correction will be send !	DEV_USHORT	READ_WRITE	No
fixedMode: If true, then Target and warning zone will be define in properties. User will not be able to change those values in runtime using attributes.(To be set using property FixMode)	DEV_BOOLEAN	READ	No

En parallèle de la documentation technique, pour permettre une meilleure mise en œuvre de cette application, j'ajoute à ce projet une documentation utilisateur. Elle comportera un descriptif complet de l'application, du principe de fonctionnement de l'asservissement mis en place. Elle décrira les étapes nécessaires pour la mise en œuvre de BeamPositionTracking sur une ligne de lumière. Chaque commande, attribut et propriété y sera décrit, on retrouvera donc les descriptions générées dans la partie précédente, accompagnées de commentaires pour l'utilisateur.

En complément à cette notice d'utilisation, comme nous l'avons vu, j'ajoute le cahier de tests dûment renseigné grâce aux tests réalisé sur la ligne NANOSCOPIUM.

II.8.3. Déploiement

Nous entrons désormais dans la dernière phase de ce projet, le déploiement. Pour fournir ce nouveau service à l'ensemble des lignes de lumière à SOLEIL, il faut en effet intégrer ce développement au package de devices déployés à chaque arrêt Machine. Environ tous les deux mois, la Machine est arrêtée pour maintenance pendant une durée en moyenne d'une dizaine de jours. Les lignes de lumière ne peuvent donc plus fonctionner pendant ce temps.

Le groupe ICA profite de cet arrêt pour effectuer le déploiement de tous les changements logiciels ainsi que les nouveaux développements. En effet, il n'est pas possible de livrer de manière continue l'ensemble des changements en production, seuls des patchs sont occasionnellement livrés dans le cas d'un problème devant être corrigé au plus vite. Le choix qui a été fait est donc d'organiser la livraison du package logiciel de contrôle commande (ensemble des développements réalisés par ICA ainsi que librairies tierces nécessaires au fonctionnement de ces derniers) en fonction des arrêts Machine. Chaque arrêt fera donc l'objet d'une nouvelle version du package «DeviceServers».

J'ai fait la demande d'intégration de l'application BeamPositionTracking dans DeviceServers à travers un ticket JIRA (ticket lié au projet ICA_DeliveryManagement). Elle a été déployée le 09/08/18 dans le package DeviceServers 18.5. Ce déploiement marque la clôture du projet BeamPositionTracking.

III. Bilan

III.1. Satisfaction client

Le cahier des charges initialement défini avec la ligne NANOSCOPIUM a été respecté, nous avons donc atteint le résultat escompté. De plus, les demandes supplémentaires de la ligne ont été satisfaites. Aujourd'hui cette application est déployée et couramment utilisée par notre client qui en est satisfait. Les scientifiques de ligne sont désormais capables de monitorer la position du faisceau et de s'assurer de son asservissement en utilisant une application totalement intégrée au système de contrôle commande de SOLEIL. De plus nous avons livré l'application dans les temps.

III.2. Perspectives du projet

Suite à la présentation réalisée en réunion suivi informatique ligne et au déploiement en production de l'application, plusieurs lignes m'ont contacté pour savoir s'il était possible d'utiliser ce développement afin de répondre à leur besoin spécifique. Une perspective à court terme pour ce projet sera d'implémenter le plugin XBPM. En effet, j'ai été contacté par les lignes SIRIUS et GALAXIES, toutes deux intéressées pour une utilisation de

BeamPositionTracking avec ce type de capteur. Cette nouvelle situation d'utilisation permettra de valider que l'application produite est modulaire et peut s'adapter à un grand nombre de situations.

Comme nous l'avons décrit dans les contours du projet, nous ne prenons pas en charge la détermination automatique des coefficients PID pour un système d'asservissement de position donné. Or je me rends compte après plusieurs discussions avec les experts métiers concernés (personnes du groupe détecteurs et contacts sur ligne de lumière SIRIUS notamment) que cette détermination pose problème. Après avoir échangé avec Kewin Desjardins, mon contact au groupe détecteurs, nous pensons qu'il serait envisageable d'étendre l'utilisation de l'application BeamPositionTracking pour lui ajouter un mode « PIDOptimisation ». Dans ce mode, l'utilisateur expert (qui aura pour rôle de déterminer les coefficients PID) pourrait faire évoluer les valeurs de chaque coefficient (P, I et D) de manière itérative (grâce à la construction d'attributs dynamiques dans le device BeamPositionTracking) et observer les courbes résultantes des positionnements ainsi produits. Il faudrait donc fournir une extension à l'IHM actuelle, c'est à dire y ajouter des graphiques permettant d'enregistrer, au déclenchement d'un positionnement, les valeurs des centroïdes projetés sur les axes X et Y périodiquement.

En discutant avec d'autres lignes, une autre idée pour une suite à ce projet serait de donner la possibilité à l'utilisateur d'asservir l'échantillon au lieu du faisceau. Pour certaines expériences (c'est notamment le cas de la ligne DISCO) l'échantillon est embarqué sur un porte échantillon, et les scientifiques ont besoin d'asservir sa position en permanence. Dans le cas de la ligne DISCO, ce porte échantillon est filmé en continu par une caméra (de type Basler) et une cible de référence est fixée sur sa surface. Ce porte échantillon étant lui-même embarqué sur deux translations (une horizontale et une verticale), on peut alors facilement imaginer une utilisation de BeamPositionTracking en modifiant l'algorithme de détection de centroïde (en écrivant un plugin spécifique) pour asservir le centre du porte échantillon sur une position donnée.

III.3. Apports personnel

Pour conduire ce projet j'ai tenu le rôle d'assistance à maitrise d'ouvrage (AMOA). J'ai en effet aidé la maitrise d'ouvrage (MOA) à définir le projet, clarifier ses besoins, décrire la solution fonctionnelle. De plus j'ai piloté l'ensemble des phases du projet pour respecter le planning prévisionnel. Ces différentes étapes m'ont amené à réaliser une étude de l'existant, notamment grâce aux présentations des équipes du LULI et de Thalès. J'ai dû effectuer une première analyse des possibilités d'orientation techniques pour ce projet. J'ai ensuite décrit une solution fonctionnelle pour arriver à déterminer une architecture cohérente afin de répondre à la problématique de la MOA. En plus du pilotage du projet et de l'organisation de réunions avec mon client pour rendre compte de l'état d'avancement, j'ai dû contrôler la qualité du développement et fournir l'ensemble des documentations nécessaires.

Etant donné le contexte particulier de ce projet (je l'ai réalisé en autonomie étant la seule ressource allouée), j'ai aussi tenu le rôle de maitre d'œuvre (MOE) c'est-à-dire le responsable du développement. J'ai donc été amené à choisir l'ensemble des technologies à mettre en œuvre pour répondre à la problématique posée et j'ai eu la responsabilité des choix de conception logicielle à réaliser. J'ai organisé et conduit diverses réunions avec mon client et les experts métiers. J'ai piloté l'ensemble des phases de conception, de développement et de tests pour cette application. Je me suis donc véritablement positionné en tant qu'ingénieur logiciel pour ce projet.

Conclusion

Le projet BeamPositionTracking a été conduit à son terme et le planning prévisionnel a pu être respecté. Cette nouvelle application d'asservissement de la position faisceau est aujourd'hui utilisée en continue (6 jours sur 7 et 24H/24) par notre client initial, NANOSCOPIUM. Cette utilisation soutenue de BeamPositionTracking m'a permis de valider la bonne tenue de cette application vis-à-vis des contraintes opérationnelles. Après quelques ajustements, BeamPositionTracking répond désormais pleinement aux attentes utilisateur, ce premier livrable fait donc office de version de référence. Les scientifiques de la ligne sont désormais capables de maitriser l'asservissement de la position faisceau à distance (comme n'importe quel autre device de leur système de contrôle), et de l'ajuster au besoin. L'application haut niveau (l'IHM JAVA fournie dans un second temps) leur permet de suivre l'évolution du positionnement de manière simplifiée et intuitive, les utilisateurs ponctuels de la ligne NANOSCOPIUM peuvent facilement la prendre en main. Les retours d'utilisation de BeamPositionTracking sont très positifs, des gains de temps d'exploitation faisceau non négligeables ont été soulignés (jusqu'à 4h d'exploitation faisceau supplémentaire chaque semaine).

Ce système est innovant et intéresse d'autres lignes de lumière à SOLEIL, n'utilisant pas nécessairement les mêmes outils de détection et de déplacements que ceux déployés sur NANOSCOPIUM. Les choix de conception ont été guidés par la volonté de produire une application hautement générique et modulaire. Je serai donc amené à continuer le développement de BeamPositionTracking pour la rendre utilisable dans d'autres situations, pour différents clients.

Bibliographie

Ouvrages:

Bradski G et Kaehler A. *Learning OpenCV: Computer Vision with the OpenCV Library*. O'Reilly Media Inc, 2008.

Desjardins K, Bordessoule M et Pomorski M. X-ray position-sensitive duo-lateral diamond detectors. *J. Synchrotron Rad* 25, 2018.

Meyer B. Conception et programmation orientées objet. Eyrolles, 2007.

Sources internet:

Douglas C. Schmidt. Overview of CORBA. Washington University in St.Louis, 2006. [En ligne] <http://www.cs.wustl.edu/~schmidt/corba-overview.html> (consulté le 20/10/17).

TANGO framework. Tango Controls. [En ligne] <http://www.TANGO-controls.org> (consulté le 20/10/17)

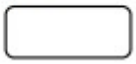
The DSDM Agile Project Framework. Agile Business Consortium. [En ligne] <https://www.agilebusiness.org> (consulté le 25/10/17).

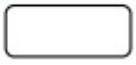
Creating Bounding boxes and circles for contours. OpenCV. [En ligne] <https://opencv.org/2.4/doc/tutorials/imgproc/shapedescriptors/bounding_rects_circles/bounding_rects_circles.html> (consulté le 29 /12/17).

Patrons de conception/Patrons du « Gang of Four ». Wikibooks. [En ligne] <https://fr.wikibooks.org/wiki/Patrons_de_conception/Patrons_du_%C2%AB_Gang_of_Four_%C2%BB> (consulté le 20 /01/18).

Annexes

Annexe 1 : Les différents états d'un device TANGO

tate	Associated color	meaning
UNKNOWN	grey 	The device cannot retrieve its state. It is the case when there is a communication problem to the hardware (network cut, broken cable etc…) It could also represent an incoherent situation
INIT	beige 	This state is reserved to the starting phase of the device server. It means that the software is not fully operational and that the user must wait
FAULT	red 	The device has a major failure that prevents it to work. For instance, A powersupply has stopped due to over temperature A motor cannot move because it has fault conditions. Usually we cannot get out from this state without an intervention on the hardware or a reset command.
DISABLE	magenta 	The device cannot be switched ON for an external reason. E.g. the powersupply has it's door open, the safety conditions are not satisfactory to allow the device to operate
OFF	white 	The device is in normal condition but is not active. E.g the powersupply main circuit breaker is open; the RF transmitter has no power etc…
STANDBY	yellow 	The device is not fully active but is ready to operate. This state does not exist in many devices but may be useful when the device has an intermediate state between OFF and ON. E.g the main circuit breaker is closed but there is no output current. Usually Standby is used when it can be immediately switched ON. While OFF is used when a certain time is necessary before switching ON.
MOVING	light blue 	The device is in a transitory state. It is the case of a device moving from one state to another.(E.g a motor moving from one position to another, a big instrument is executing a sequence of operation, a macro command is being executed.)

ON	green 	This state could have been called OK or OPERATIONAL. It means that the device is in its operational state. (E.g. the powersupply is giving its nominal current, the motor is ON and ready to move, the instrument is operating). This state is modified by the Attribute alarm checking of the DeviceImpl:dev_state method. i.e if the state is ON and one attribute has it's quality factor to ALARM, then the state is modifiend to ALARM
ALARM	orange 	The device is operating but one of this attribute is out of range. It can be linked to alarm conditions set by attribute properties or a specific case. (E.g. temperature alarm on a stepper motor, end switch pressed on a steppermotor, up water level in a tank, etc…) In alarm, usually the device does it's job but the operator has to perform an action to avoid a bigger problem
RUNNING	dark green 	This state does not exist in many devices but may be useful when the device has a specific state above the ON state. (E.g. the detector system is acquiring data, An automatic job is being executed). Note that this state is different from the MOVING state.
OPEN	green 	Synonym of ON state. Can be used when ON is not adequate for the device. E.g case of a valve, a door, a relay, a switch.
CLOSE	white 	Synonym of OFF state. Can be used when OFF is not adequate for the device. E.g case of a valve, a door, a relay, a switch.
EXTRACT	green 	Synonym of ON state. Can be used when ON is not adequate for the device. Case of insertable/extractable equipment, absorbers, etc… This state is here for compatibility reason we recommend to use ON or OPEN when possible.
INSERT	white 	Synonym of OFF state. Can be used when OFF is not adequate for the device. Case of insertable/extractable equipment, absorbers, etc… This state is here for compatibility reason we recommend to use OFF or CLOSE when possible.

Réunion de Direction n° RD735

Réf. : RD735 AN / D3349

Rédigé par : A. Buteau, A. Somogyi, K. Medjoubi et A. Nadji

Réf DAI: : I/AI/AN/18.0005

Date : 30/01/2017

OBJET : NOTE D'OPPORTUNITE ET D'ORGANISATION

PROJET : « PROJET D'ASSERVISSEMENT DU FAISCEAU SUR UN POINT DE VISEE »

Rappel du contexte

Les systèmes de contrôle et d'acquisition des lignes de lumière de SOLEIL permettent de déplacer tout type d'actionneur (*qu'il soit physique, tel un axe réel ou logique, tel que l'énergie du monochromateur*), d'acquérir des images du faisceau et d'en déduire des informations sur le faisceau (position du centroïde, taille du faisceau, etc…).

Il n'existe cependant aucune application logicielle, fournie par le groupe ICA comme un service logiciel standardisé, qui permette de réaliser un asservissement de la position du faisceau sur un point de visée.

La ligne NANOSCOPIUM, ayant de fortes contraintes de suivi de la position du faisceau sur leur échantillon, a ainsi du développer une solution pour leurs besoins propres par le biais d'une apprentie, et souhaitait « industrialiser » cette fonctionnalité et revenir dans le cadre d'un produit logiciel standard géré par le groupe ICA comme tout autre application TANGO.

Par ailleurs, grâce à ses contacts avec l'équipe contrôle/commande du laser APOLLON et de la société THALES, le groupe ICA a pu prendre connaissance du projet BeamPointing développé au sein de la communauté Laser (pour APOLLON et E.L.I Roumanie) pour des besoins analogues d'asservissement du faisceau sur un point de visée.

Motivations du projet

La conjonction du besoin exprimé par NANOSCOPIUM et d'une première solution proposée par THALES dans le cadre du projet APOLLON, a prouvé l'intérêt fonctionnel de proposer une telle application logicielle, qui vise à être potentiellement utilisable par toutes les lignes de lumière.

Par ailleurs la possibilité de s'appuyer sur F. THIAM dans le cadre de son mémoire d'ingénieur CNAM a permis de démarrer une phase d'analyse fonctionnelle et technique, qui nous conduit à proposer les livrables et l'organisation du projet ci-dessous.

Livrables du projet

Le service logiciel fourni par le groupe ICA (*appelé « BeamPositionTracking* ») sera constitué d'un ensemble de devices (et librairies associées) ainsi que d'une IHM simplifiant leur utilisation et permettant à l'utilisateur d'interagir de façon ergonomique avec les fonctions d'asservissement (*quels actionneurs et quels capteurs sont utilisés, quel algorithme d'asservissement avec quels paramètres*). Compte tenu de l'expérience initiale de NANOSCOPIUM, il parait suffisant, dans un premier temps, de disposer dans le service logiciel *BeamPositionTracking* d'un mécanisme d'asservissement simple (linéaire ou PID), en laissant la problématique de calcul des paramètres PID d'être traitée de façon indépendante par un code de calcul externe.

Phases du projet

Pour la partie ICA :

Phase de faisabilité : 4ème trimestre 2017/janvier 2018

- Etude de l'existant APOLLON/THALES et NANOSCOPIUM ;
- Architecture logicielle de l'ensemble ;
- Mise en place d'un environnement de tests au labo ;
- Rédaction du document de faisabilité.

Phase de réalisation d'une version initiale de la partie fonctionnelle : janvier/mars 2018

- Conception puis réalisation du logiciel ;
- Validation sur plateforme de tests ;
- Tests sur la ligne NANOSCOPIUM.

Phase de réalisation d'une version initiale de la partie IHM : mars/mai 2018

- Design interface utilisateur ;
- Conception puis réalisation de l'IHM ;
- Validation sur plateforme de tests ;
- Tests sur la ligne NANOSCOPIUM.

Itérations sur les parties fonctionnelles et graphiques : juin 2018

- Présentation du projet en réunion suivi Lignes ;
- Déploiement sur un ensemble de lignes candidates et adaptation/ajustements aux nouvelles demandes.

Activités de transitions vers l'Opération

- Packaging logiciel pour déploiement sur l'ensemble de SOLEIL ;
- Rédaction de documentation utilisateur et techniques ;
- Formation des utilisateurs ;
- Formation des équipes opérationnelles (Niveau 1 et Niveau 2).

Pour la partie NANOSCOPIUM (K. Medjoubi)

- Fourniture à ICA du code de l'outil développé en Matlab par l'apprentie de NANOSCOPIUM (Gwendoline Jacquy, période 2013 – 2016) encadré par K. Medjoubi et mise à disposition de l'ensemble de ses rapports détaillant les méthodes développées et testées ;
- Participation aux réunions de synthèse ;
- Validation sur la ligne du développement réalisé par ICA et comparaison avec l'outil Matlab.

Organisation du projet

Le groupe Projet est constitué des personnes suivantes :

RACI	Etude de faisabilité	Réalisation d'une version initiale de la partie fonctionnelle	Réalisation d'une version initiale de la partie IHM	Itérations sur les parties fonctionnelles et graphiques	Activités de transitions vers l'Opération
Responsable	TF	TF	TF	TF	TF
Réalisateurs	TF	TF	TF	TF	TF
Consultés	AB, FL, KM, APOLLON, THALES	FL, KM	FL, experts IHM ICA	Lignes betas test, FL	AB, FL
Informés	FL	AB	AB, KM	AB	Niveau 1, Niveau 2

Recommandation

Les directeurs sont invités à :

- approuver la réalisation du projet « Asservissement logiciel du faisceau sur un point de visée » ;
- approuver la note d'organisation présentée dans ce document.

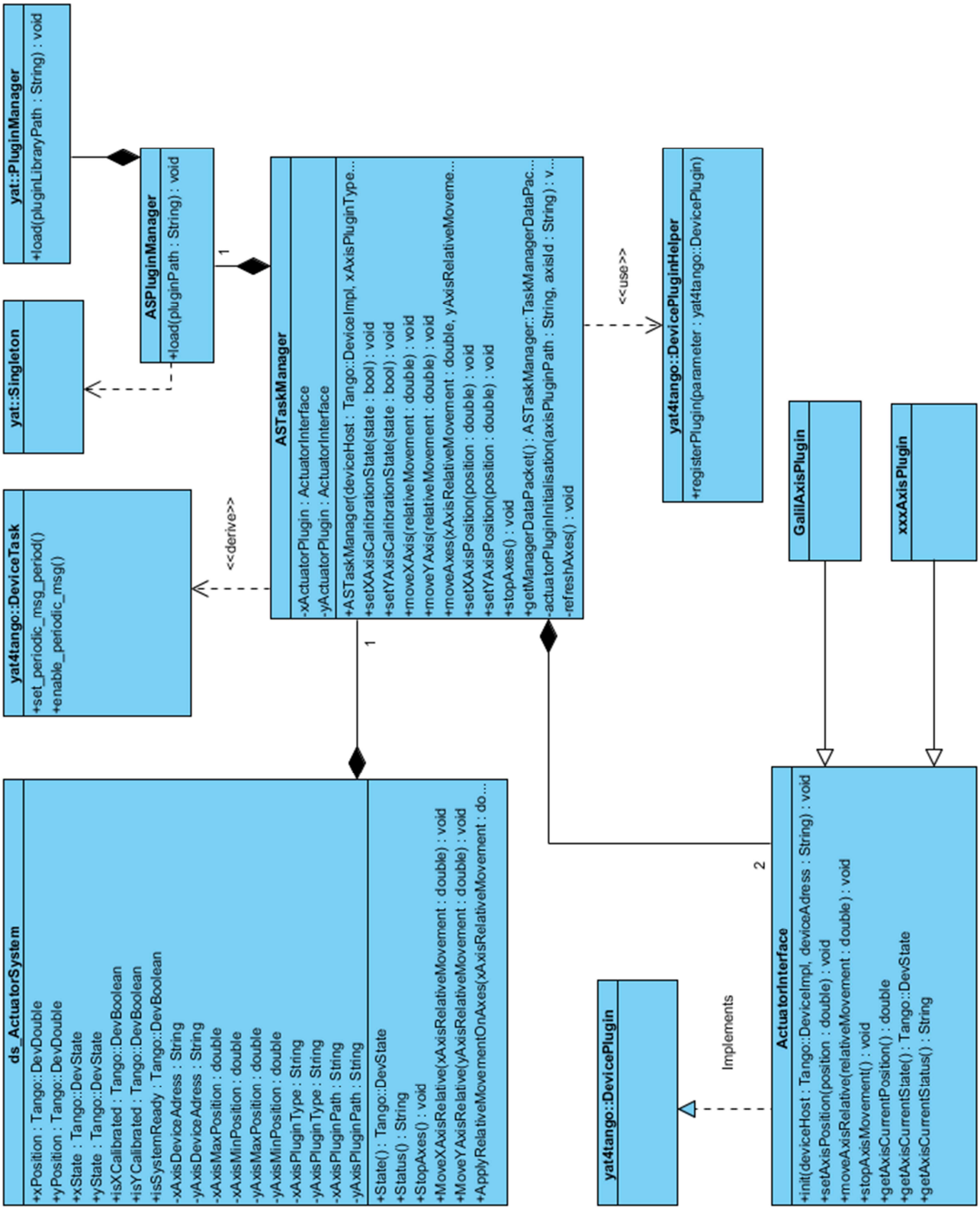
Annexe 3 : Les différents types de données de TANGO

Types TANGO	Types équivalents en C++
TANGO::DevBoolean	`boolean`
TANGO::DevShort	`short`
TANGO::DevEnum	enumeration (only for attribute / See chapter on advanced features)
TANGO::DevLong	`int` (always 32 bits data)
TANGO::DevLong64	`long long` on 32 bits chip or `long` on 64 bits chip (always 64 bits data)
TANGO::DevFloat	`float`
TANGO::DevDouble	`double`
TANGO::DevString	`char \*`
TANGO::DevEncoded	`structure` with 2 fields: a string and an array of unsigned char
TANGO::DevUChar	`unsigned char`
TANGO::DevUShort	`unsigned short`
TANGO::DevULong	`unsigned int` (always 32 bits data)
TANGO::DevULong64	`unsigned long long` on 32 bits chip or `unsigned long` on 64 bits chip (always 64 bits data)
TANGO::DevState	TANGO specific data type

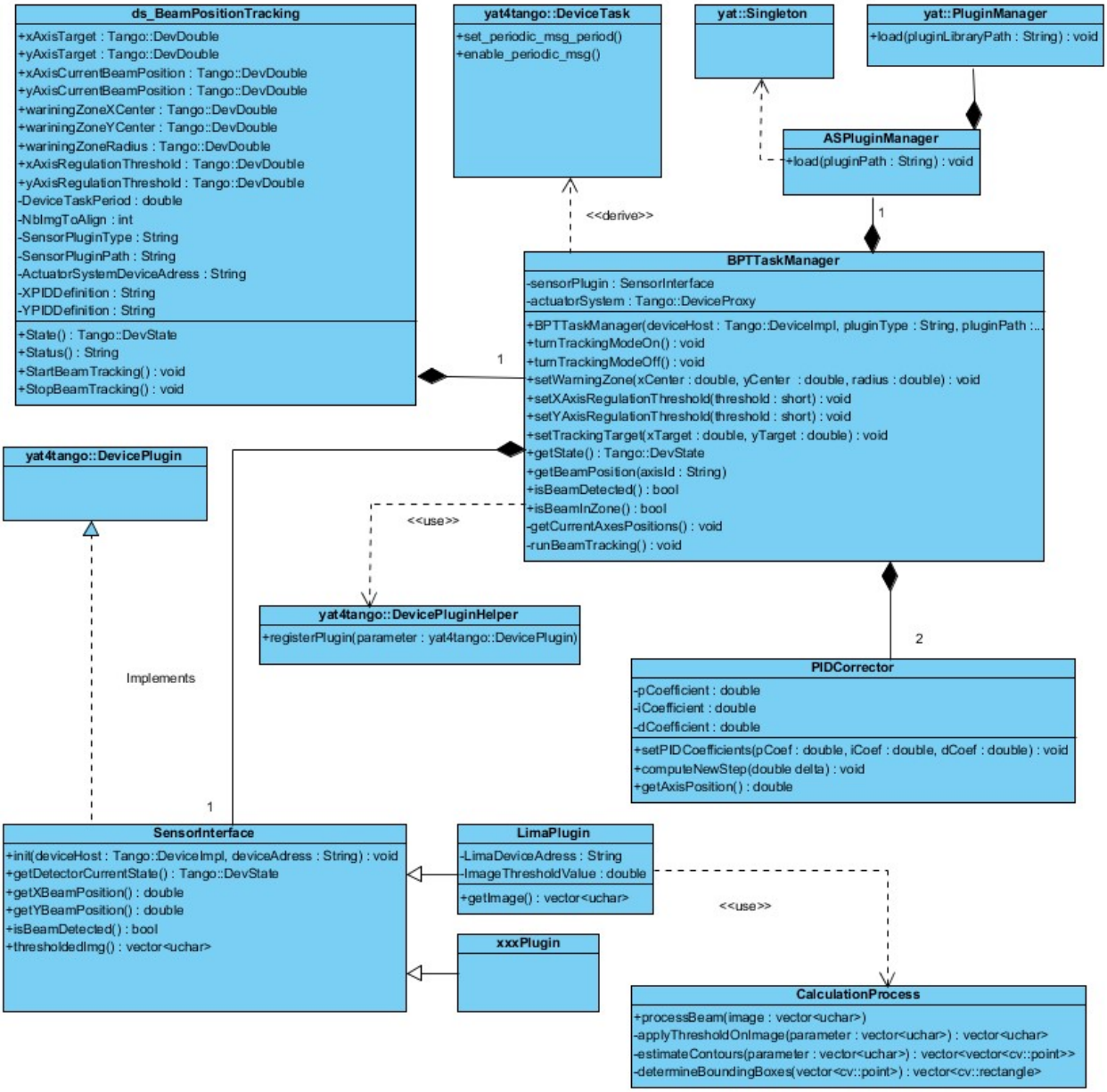
Annexe 4 : Fichier Pom du device TestBPTDevice

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/maven-v4_0_0.xsd">
        <modelVersion>4.0.0</modelVersion>
        <parent>
                <groupId>fr.soleil</groupId>
                <artifactId>super-pom-C-CPP-device</artifactId>
                <version>1_0_0</version>
        </parent>
        <groupId>fr.soleil.device</groupId>
        <artifactId>TestBPTDevice-${aol}-${mode}</artifactId>
        <version>1.0.1-SNAPSHOT</version>
        <packaging>nar</packaging>
        <name>TestBPTDevice</name>
        <description>TestBPTDevice device</description>

        <build>
                <plugins>
                        <plugin>
                                <groupId>org.freehep</groupId>
                                <artifactId>freehep-nar-plugin</artifactId>
                                <configuration>
                                        <os>Linux</os>
                                </configuration>
                        </plugin>
                </plugins>
        </build>

        <developers>
                <developer>
                        <id>thiam</id>
                        <name>thiam</name>
                        <url>http://controle/</url>
                        <organization>Synchrotron SOLEIL</organization>
                        <organizationUrl>http://www.synchrotron-
                        soleil.fr</organizationUrl>
                        <roles>
                                <role>manager</role>
                        </roles>
                        <timezone>1</timezone>
                </developer>
        </developers>
</project>
```

Annexe 5 : Diagramme de classes pour ActuatorSystem



Annexe 6 : Diagramme de classes pour BeamPositionTracking



Introduction

Pour tester l'ensemble du système BeamPositionTracking, il est impératif de commencer par les tests unitaires du device ActuatorSystem. Si les envois de positionnement sont cohérents, on pourra passer à la suite sans s'en préoccuper (transformation entre déplacement relatif et absolu sur galilAxis…).

Une fois les tests sur AS validés, il faudra changer son mode pour le passer en SIMULATED afin d'assurer dans un premier temps qu'aucun déplacement ne sera envoyé malencontreusement.

Commencer les tests de BPT en mettant son mode de fonctionnement à SIMULATED. Une fois la majorité des tests effectués, s'assurer que la valeur du premier pas envoyé (le deuxième n'étant pas cohérente puisque le correcteur PID a dans notre cas une composante Intégrale, le temps est donc important…) pendant le mode tracking est correct. S'il parait cohérent, passer à la fois BPT et AS en mode NORMAL et poursuivre les tests.

Device ActuatorSystem - AS

Initialisation :

Description du test	Démarrage du device (cas d'erreur)

Etat initial	Au moins un des cas ci-après est avéré :
	• Au moins un device n'est pas joignable :
	  ○ Celui de la translation X
	  ○ Celui de la translation Y
	• Le PATH d'au moins un plugin n'est pas correct
	• Le type du plugin n'est cohérent

Comportement attendu	Le device démarre : ➢ il passe en FAULT et son statut reflète la nature exacte de l'erreur
Comportement réel	Le device démarre correctement et affiche en fonction des erreurs générées la source dans son statut. [Labo Alignement – 18/01/18]

Basculement d'états : Système ON, système STANDBY, système ALARM

Description du test	Le système est prêt, il bascule dans un autre état
Etat initial	Le système est prêt : State ON
Comportement attendu	Le booléen associé à cet état : SystemReady passe à faux, le state bascule
Comportement réel	Dès le changement d'état (de ON à n'importe quel autre), le booléen SystemReady passe à faux [Labo Alignement –

	18/01/18]

Envoie de positionnement relatif

Description du test	Envoie d'une commande de positionnement relative simultanée (sur X et Y) : apply_relative_movement_on_axes
Etat initial	Le système est prêt : State ON, ou le système n'est pas prêt : State STANDBY
Comportement attendu	De manière simultanée, les translations passent à MOVING (doit correspondre à l'état des devices sous-jacents). L'état d'AS doit passer à MOVING le temps des translations. L'état final (lorsque le déplacement est terminé) d'AS dépend de l'état des translations sous-jacentes, voir digramme d'état.
Comportement réel	Le comportement attendu est atteint [Labo Alignement – 19/01/18]

Device BeamPositionTracking - BPT

Initialisation :

Description du test	Démarrage du device (cas d'erreur)
Etat initial	Au moins un des cas ci-après est avéré : • Le device ActuatorSystem n'est pas joignable • Le device LIMA n'est pas joignable • Le PATH du plugin n'est pas correct • Le type du plugin n'est cohérent
Comportement attendu	Le device démarre : ➢ il passe en FAULT et son « status » reflète la nature exacte de l'erreur
Comportement réel	Le device démarre correctement et affiche en fonction des erreurs générées la source dans son statut. [Labo Alignement – 18/01/18]

Basculement d'état : Détection faisceau

Description du test	Le faisceau est perdu ou sort de sa zone
Etat initial	Faisceau trouvé en zone, state : ON
Comportement attendu	Le device bascule en état STANDBY Le device notifie l'utilisateur via son statut qu'il n'est pas possible de lancer le mode tracking dans l'état STANDBY. Il

	précise la raison exacte.
Comportement réel	Dès que le faisceau n'est plus dans sa zone, le state du device bascule à STANDBY [Labo Alignement – 18/01/18]

Comportement du mode tracking

Description du test	Déclanchement du mode tracking
Etat initial	Le device est en state ON : • Le faisceau est dans la warning zone • Le device AS est state ON
Comportement attendu	Le device passe à l'état RUNNING Chaque période « DeviceTaskPeriod » une estimation du centroïde est effectuée (sur «NbImgToAlign» images): • Une différence est calculée entre position actuelle du faisceau et consigne utilisateur • Cette différence est injectée dans le correcteur PID qui calcule une erreur • Cette erreur est estimée pour les axes X/Y et est envoyée sur le device AS La position faisceau converge vers le point visé en un certain nombre d'itération, en fonction des valeurs du PID
Comportement réel	Validé en mode linéaire (coefficient proportionnel uniquement) [Labo Alignement – 22/01/18]

Description du test	Le tracking est en route, le faisceau sort de sa WarningZone
Etat initial	Pas d'erreur, state : RUNNING
Comportement attendu	Le device coupe le mode tracking et passe en STANDBY. Son statut affiche que le faisceau n'est plus en zone, il n'est plus possible de démarrer l'asservissement en position.
Comportement réel	Dès que le faisceau n'est plus dans sa zone, le state du device bascule à STANDBY [Labo Alignement – 22/01/18]

Liste des figures

Liste des tableaux

Étude et réalisation d'une application d'asservissement. Positionnement d'un faisceau Synchrotron. Mémoire d'Ingénieur C.N.A.M., Paris 2018.

Résumé

Le centre de recherche SOLEIL est un accélérateur de particules de type Synchrotron de troisième génération. Il est opérationnel depuis 2006 et fait aujourd'hui partie des grands instruments Français. Constitué d'un ensemble de laboratoires spécifiques aussi appelés lignes de lumière, SOLEIL bénéficie d'un système de contrôle commande informatique particulier appelé TANGO, initialement développé au Synchrotron de Grenoble par l'ESRF. Ce système de contrôle d'objets distribués est utilisé pour uniformiser le pilotage de nombreux instruments scientifiques.

Le projet BeamPositionTracking a pour but de fournir une application permettant la tenue en position spatiale du faisceau d'électrons produit par l'accélérateur, une fois injecté sur une ligne de lumière. L'objectif est d'être en mesure d'utiliser le système TANGO pour piloter et synchroniser l'ensemble des éléments nécessaires au positionnement dynamique du faisceau ainsi qu'à la prise d'information quant à sa position courante afin de mettre en œuvre un asservissement de longue durée.

Mots clés: **TANGO, CORBA, Synchrotron SOLEIL, Programmation orientée objet**

Summary

The SOLEIL research center is a third generation Synchrotron particle accelerator. Operational since 2006 it is now one of the major French instruments. Consisting of a set of specific laboratories also called beamlines, SOLEIL benefits from a specific software control system called TANGO, initially developed by Grenoble's Synchrotron, the ESRF. This distributed object control system is used to standardize the control of many scientific instruments.

The purpose of the BeamPositionTracking project is to provide an application that allows the beam generated by the accelerator to be held in spatial position, once injected onto a beamline. The aim is to be able to use the TANGO system to control and synchronize all the elements responsible for the dynamic positioning of the beam as well as to take information about its current position in order to implement a long duration enslavement.

Key words: **TANGO, CORBA, Synchrotron SOLEIL, Object-oriented programming**