Le génie pour l'industrie

RAPPORT FINAL
Projet de fin d'études
Département de génie logiciel et des TI

Conception d'un système de compensation lumineuse pouvant varier en spectre et en intensité

PFE-H20-005

Auteurs
LÉANDRE ARSENEAULT - ARSL17099609
PHILIPPE DE LADURANTAYE - DELP31019503
BRIAN QUIRION - QUIB09109401

Professeur superviseur
ALAIN APRIL

MONTRÉAL, 11 AVRIL 2020

Table des matières

Introduction

L'éclairage d'une serre à lumière naturelle est un inconnu constant. On est à la merci de mère nature, ce qui peut occasionner des rendements variables à travers le temps. Pour pallier ce problème, les agriculteurs utilisent des luminaires qui peuvent garantir un certain éclairage, peu importe les conditions météorologiques. Malheureusement, cette source d'éclairage secondaire vient avec son lot de problèmes. En effet, l'intensité du soleil ainsi que sa composition spectrale varient avec le temps alors qu'un luminaire conventionnel ne fait qu'émettre une lumière fixe. Le marché actuel propose généralement un seul type de solution à ce problème: des capteurs de lumière qui vont arrêter ou mettre en marche les lampes en fonction de l'intensité du soleil. Cela assure l'agriculteur que ses lampes ne fonctionnent pas inutilement durant les jours plus ensoleillés. D'autres fabricants de lampes offrent des capteurs plus précis qui vont permettre d'atténuer les lampes graduellement pour assurer une intensité plus uniforme au courant de la journée. Cette approche est donc surtout axée sur l'intensité lumineuse; elle ignore complètement la composition spectrale de la lumière.

Sollum est une entreprise montréalaise qui œuvre dans la conception de lampes programmables à spectres dynamiques. L'équipe s'occupe de la conception mécanique de la lampe, la conception électrique des cartes embarquées, l'écriture du « firmware » et de toutes les couches logicielles nécessaires à la communication. En effet, l'entreprise offre une solution intelligente en connectant ses lampes à l'internet et en permettant aux clients d'en contrôler les paramètres. Sollum a déposé un projet comme candidat à une subvention de développement durable dont l'un des volets propose une solution plus complète à ce problème. Celle-ci est un système de compensation lumineuse pouvant varier en spectre en intensité. Pour ce faire, trois disciplines entrent en jeu: L'électrique pour la conception d'un capteur de lumière, la mécanique pour la conception d'un boîtier pour le capteur et l'informatique pour la transmission des données à travers les systèmes et les algorithmes de composition de la lumière. Dans le cadre de notre projet, la partie logicielle est celle sur laquelle nous concentrerons nos efforts.

Glossaire

Terme	Définition
Canal lumineux	Un canal lumineux est une série de LEDs de même couleur sur une lampe. La lampe SF3 que nous utiliserons pour le projet contient onze canaux lumineux différents.
Capteur	Les capteurs sont en période de conception dans le projet de fin d'études de l'équipe électrique et mécanique. Ils permettent de capter le spectre et l'intensité de la lumière fournie par l'environnement, notamment le soleil.
CCT	Le CCT, ou « *correlated color temperature* », est la couleur du rayonnement électromagnétique émis par un corps noir maintenu à une température constante et uniforme. Sa valeur est donnée Kelvin. Le CCT est utilisé comme couleur de référence pour la génération de spectres.
Client	Dans le contexte de ce rapport, le ou les clients sont les agriculteurs qui gèrent des serres qui seront équipées de compensateurs.
Compensation	Processus qui utilise une source de lumière quelconque, généralement le soleil, pour calculer un nouveau spectre qui assure de compléter adéquatement les lacunes de la lumière pour atteindre une cible déterminée par une recette.
Fichier d'acquisition	Fichiers générés durant le processus de fabrication des lampes. Ils contiennent les valeurs des spectres de chaque canal pour les valeurs d'intensité de 25%, 50%, 75% et 100%.
« Firmware »	Programme intégré directement à du matériel informatique.
Fonction d'évaluation	Fonction utilisée par la fonction d'optimisation pour calculer le score relatif à une combinaison quelconque de paramètres.

Fonction d'optimisation	Trouve le minimum global d'une fonction multi variable. Dans le cadre du projet, elle trouve la combinaison optimale des valeurs d'intensité de chaque canal de la lampe pour atteindre un spectre cible.
Longueur d'onde	La longueur d'onde de la lumière se veut comme étant la fréquence de la lumière produisant une couleur.
Lumière visible	Les couleurs visibles par l'œil humain se situent entre les longueurs d'onde de 400 et 700 nanomètres.[1]
Plateforme infonuagique	La plateforme infonuagique est le système par lequel l'agronome peut programmer les recettes des luminaires.
Promoteur	Sollum Technologies, l'entreprise qui a proposé le projet.
Recette	Une recette se définit comme étant une combinaison de plusieurs scénarios sur une journée, mais pouvant se répéter sur plusieurs jours, voire des semaines. On va généralement opter pour l'utilisation de recettes différentes selon les plants ou aliments à faire croître.
Rectification	Synonyme de compensation.
Scénario	Un scénario dans le contexte du système se définit comme étant l'interpolation entre des spectres et des intensités en fonction d'un temps relatif.
SF3	Une SF3 ou « Sollum Fixture » modèle 3 est le modèle de lampe que nous utiliserons pour le projet.
Slack	Logiciel de messagerie.
Sollum	Sollum est le promoteur du projet. L'entreprise fait la conception de lampes à spectres programmables qui reproduisent avec fidélité la lumière naturelle du soleil.
Spectre	Le spectre lumineux est la répartition de l'intensité d'une lumière en fonction de la longueur d'onde.

1 NATIONAL AERONAUTICS AND SPACE ADMINISTRATION (NASA). *Visible Light.* Consulté le 7 avril 2020. https://science.nasa.gov/ems/09_visiblelight

Spectromètre	Instrument permettant de décomposer une source de lumière en distribution spectrale, c'est-à-dire en quantité de photons par longueur d'onde. Dans notre cas, nous utilisons surtout ces données sur le domaine de la lumière visible.
SRU	« Sollum Routing Unit » est le routeur utilisé pour gérer la communication entre l'application infonuagique et les lampes.

Analyse du problème (Q4-i1)

Besoins et problème

Afin de valider que ce projet est réalisable, Sollum s'est vu demander une preuve de concept d'un compensateur spectral durant le processus d'application à la subvention de développement durable. Pour ce faire, l'équipe de développement logiciel de l'entreprise s'est basée sur leur algorithme de génération de spectre en Python pour proposer une solution de rectification démontrant la faisabilité du projet. Le code source de cette preuve de concept nous est fourni comme base, mais le promoteur nous assure que nous avons la liberté de l'utiliser dans la mesure qu'il nous convient pour réaliser le projet de fin d'études.

Bien évidemment, l'une des premières étapes de notre processus d'analyse et de conception a été de lire le code source de cet algorithme pour en comprendre le fonctionnement et identifier les problèmes et les solutions qui s'offrent à nous. Rapidement, nous avons réalisé que l'algorithme était beaucoup trop lent pour réaliser des compensations dynamiques en serres. En effet, sans compter les temps de communications à travers les systèmes, le calcul de rectification de spectre prend environ une minute et est très demandant au niveau du processeur. Aussi, le programme est totalement déconnecté des systèmes de Sollum qui permettent de contrôler les lampes à partir d'une application infonuagique.

De plus, nous avons vu que la solution choisie initialement par l'équipe de développement logiciel de l'entreprise est extrêmement près de l'algorithme de création des spectres. En effet comme les spectres sont générés à partir de courbes de CCT cibles, la preuve de concept ne fait qu'ajouter la lumière naturelle comme un canal lumineux fixe et utilise encore ces courbes de CCT comme cibles à l'algorithme d'optimisation. Cette méthode prouve en effet la possibilité d'effectuer des compensations en spectre à partir d'une lumière fixe, mais n'est pas adaptée à l'utilisation qui en sera faite par les clients réels. Ces derniers voudront en effet utiliser leurs recettes comme cibles spectrales au lieu de CCT cibles qui ne sont que des valeurs théoriques parfaites. Comme leurs recettes sont personnalisables, les limiter à ces valeurs ne ferait pas de sens.

Finalement, Sollum désire évidemment intégrer de vraies données lumineuses au système de compensation, ce que leur code ne fait pas encore. Ces données devront être acquises, sauvegardées et utilisées dans le processus de compensation. Idéalement, des graphiques seraient générés pour montrer au client la composition réelle de la lumière produite par ses luminaires et les différents impacts que la rectification a sur ses serres.

Objectifs du projet

Après avoir discuté avec le promoteur, analysé la preuve de concept et regardé l'architecture en place en ce moment, nous sommes en mesure de déterminer les objectifs du projet.

1. Installation du compensateur dans l'architecture de Sollum

   Le premier objectif et celui qui est le plus critique au projet est l'installation de l'algorithme de compensation dans l'architecture de Sollum. Étant donné que le système est distribué sur plusieurs couches (c.-à-d. application infonuagique, routeur et lampes), il est important d'assurer que chacune d'entre elles supporte le passage des données de compensation.

   Critère de réussite : le routeur peut récupérer les données des capteurs et les envoyer à l'algorithme de compensation. Ce dernier peut ensuite envoyer une commande au routeur qui applique le résultat du calcul de compensation à effectuer aux lampes.

2. Optimisation de la performance du système

   Un autre objectif majeur est d'augmenter la performance de l'algorithme. Présentement le temps d'environ une minute est trop élevé et devrait être considérablement réduit.

   Critère de réussite : l'algorithme prend moins de cinq secondes afin de calculer une compensation.

3. Implémentation de l’algorithme de compensation par recette

   Présentement, la preuve de concept ne répond pas aux réels besoins des clients. L’objectif serait d’implémenter une solution qui permet d’accepter les données d’une recette en entrée au lieu d’une valeur de CCT.

   Critère de réussite : l’algorithme prend des données de recettes en entrée.

4. Mise en place d’une base de données et conception de graphiques client

   Les objectifs principaux du projet sont surtout relatifs à la fonctionnalité de compensation en soi comme indiqué dans les trois objectifs précédents. Cependant, les clients désirent être au courant de ce qui se passe dans leurs serres. Par conséquent, l’un de nos objectifs sera de stocker les données du compensateur dans une base de données, puis de générer des graphiques significatifs disponibles dans la plateforme infonuagique destinée aux clients.

   Critère de réussite : la base de données est en place et des graphiques de compensation sont disponibles dans la plateforme client.

5. Amélioration des attributs d’extensibilité, de maintenabilité et de réutilisabilité du système

   Le système, entièrement codé en Python, manque visiblement de structure logicielle lui permettant d’être extensible, maintenable et réutilisable. L’objectif sera d’améliorer le plus possible ces attributs au fur et à mesure que nous implémentons des fonctionnalités à travers le système. Cet objectif n’est pas prioritaire, mais nous croyons que notre expertise en tant que futurs ingénieurs logiciels pourrait grandement bénéficier Sollum. Alors, nous tenterons de le garder en tête tout au long de la réalisation du projet.

   Critère de réussite : chaque nouvelle fonctionnalité ou modification d’une fonctionnalité initialement présente a amélioré les attributs d’extensibilité, de maintenabilité et de réutilisabilité du système.

Séparation en tâches

Durant les premières semaines, plusieurs tâches ont été soulevées après l'identification des problèmes à résoudre et des objectifs à atteindre. Nous avons donc listé la plupart des tâches à effectuer par couche logicielle pour pouvoir commencer à les entrer dans nos sprints. Ce document contient, entre autres, les tâches qui étaient initialement censées être réalisées et avaient été validées par l'entreprise. Le document se retrouve en annexe (ANNEXE I).

Méthodologie

Gestion des communications

Étant donné que deux des membres de notre équipe travaillaient chez Sollum durant la durée du présent projet, la gestion des communications avec le client s'est très bien déroulée. Léandre et Brian ont donc été en mesure de recueillir l'information nécessaire et les commentaires du client au fil de l'avancement du projet. Aussi, comme l'application des concepts liés à la méthode Scrum était déjà en place dans l'entreprise, cette méthode a été utilisée avec le client pour gérer les attentes et les besoins de celui-ci tout au long du projet. Nous nous sommes également présentés à plusieurs reprises dans les locaux de Sollum pour effectuer des rencontres en personne et pour présenter les différentes technologies au membre de l'équipe qui ne faisait pas partie de l'entreprise.

Méthodologie de travail (Q7-i3)

Afin d'assurer une communication efficace entre les différents partis impliqués dans le projet, nous avons mis en place quelques outils classiques de gestion de projet, en commençant par Trello. Effectivement, nous avons utilisé Trello pour gérer les jalons du projet, les différentes itérations (ou sprint) inspirées de la méthode Scrum et pour faire le suivi des tâches de manière efficace.

Nous avons également instauré des rencontres virtuelles hebdomadaires afin de partager nos avancements respectifs dans le projet. Lors de ces rencontres, nous procédions à l'attribution des tâches aux différents membres pour la semaine suivante, le but étant que les tâches de chacun soient terminées et prêtent pour révision avant la rencontre de la semaine suivante. Nous expliquions également nos tâches de la semaine afin de prendre en compte les commentaires de nos collègues et de faciliter la révision de la tâche pour ceux-ci.

Pour assurer les communications rapides durant la semaine, nous avons créé un canal « channel » Slack. Cet outil nous a grandement aidés dans nos communications tout au long du projet, puisqu'il est facilement accessible à tous sur toutes les plateformes. Puisque

nous utilisions toujours le même canal, Slack nous a également été très utile pour faire la synthèse de la démarche et du raisonnement ayant mené à certaines solutions, lorsque celles-ci avaient été abordées via les conversations Slack.

Finalement, la suite Google a été utilisée pour assurer le partage des documents et pour la rédaction de documents en équipe tels que les différents livrables et le rapport final. Comme nous disposons d'une licence avec notre courriel de l'école, cette solution nous a paru la plus simple et la plus logique.

Composition de l'équipe et rôles

Puisque Léandre avait déjà effectué un stage dans l'entreprise et qu'il connaissait l'architecture des systèmes, il a occupé le rôle de chef d'équipe et d'architecte de solution. Philippe et Brian avaient donc des rôles davantage axés sur le développement.

Prénom	Rôles	Responsabilités
1. Léandre	Chef d'équipe, concepteur et développeur middleware et infonuagique	● Chapeauter l'équipe concernant les décisions de l'architecture du système. ● Conception et développement des modules du compensateur et du SRU.
2. Philippe	Développeur infonuagique et concepteur de base de données	● Développement des modules situés dans le nuage. ● Optimisation des calculs de rectification, d'évaluation et de traitement de données.
3. Brian	Concepteur et développeur middleware et infonuagique	● Conception et développement des différents modules reliés à l'interface utilisateur et également les modules infonuagiques. ● Conception et développement des modules du compensateur et du SRU.

Tableau 1 : Rôles des membres de l'équipe

Livrables

Les trois principaux livrables pour ce projet étaient :1) le plan de projet initial; 2) la présentation orale; ainsi que 3) ce rapport final. En effet, la planification initiale du projet a été utilisée afin de présenter l'approche méthodologique ainsi que les objectifs du projet au client pour s'assurer que tous les partis avaient la même compréhension de ces aspects. Quant au rapport final, celui-ci décrit les différents processus décisionnels et raisonnements ayant mené aux solutions que nous avons mis en place dans le cadre de ce projet. La documentation d'architecture et les diagrammes de classe ont aussi été inclus dans le rapport final dans le but d'appuyer la présentation des solutions. Ces solutions ont quant à elles été décrites de vive voix lors de la présentation finale au client.

Nom de l'artéfact	Description
Plan de projet	Décrit les livrables et artéfacts à produire. Décris également les différents risques du projet, ainsi que l'approche méthodologique utilisée pour réaliser et bien gérer le projet.
Rapport final	Document technique comportant la description du projet, la description des travaux réalisés, les conclusions tirées des analyses du projet, ainsi que l'état final du projet.
Présentation orale	Présentation finale du projet devant les pairs.
Tableau Scrum (Trello)	Contient l'état d'avancement des travaux, leur description et leurs priorités pour que Sollum sache ce qui a été fait et ce qu'on n'a pas réussi à terminer.
Code source	Code du module de rectification de spectre.
Diagramme d'architecture	Diagramme représentant l'architecture globale des systèmes de Sollum interagissant avec le module de rectification de spectre.
Diagramme de classes	Diagramme des classes du module de rectification de spectre.

Tableau 2 : Description des livrables

Formulation des solutions (technique) / Conception (Q4-i2)

Connexion des systèmes

Solutions potentielles

L'architecture de Sollum est bien évidemment distribuée afin de répondre aux besoins des objets connectés. Cela nous donne une série d'options pour l'intégration d'une solution de compensation dans la structure actuelle.

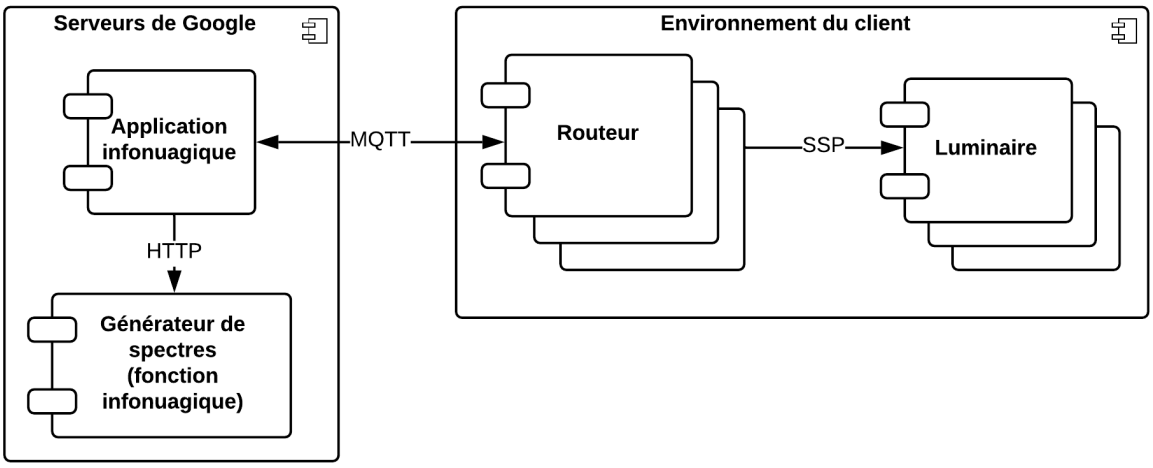

Figure 1 : Architecture initiale de Sollum

La figure 1 présente une vue générale des systèmes présents dans l'architecture de Sollum au début du projet. L'enjeu de la connexion des systèmes nécessitera minimalement l'ajout de spectromètres physiques et d'une manière de calculer les compensations.

Calculs de compensation dans la lampe

A.Llenas et J.Carreras estiment, dans un papier de recherche publié en mars 2019, que la meilleure manière de minimiser les délais dans les calculs de compensation en spectre est en effectuant ces calculs directement sur le microcontrôleur des lampes.[2] Nous pourrions effectivement opter pour une solution du même type afin de vraiment minimiser les délais et

2 LLENAS, CARRERAS. *Arbitrary spectral matching using multi-LED lighting systems*, Optical Engineering 58(3), 035105 (29 March 2019). Consulté le 12 février 2020. https://doi.org/10.1117/1.OE.58.3.035105

d’obtenir une compensation en direct imperceptible pour l’œil humain. Le plus gros problème avec cette solution est qu’elle nécessite un changement des composantes de la lampe, de leur « firmware » et en plus, une optimisation extrême de l’algorithme de calcul de compensation serait nécessaire. La portée de ces changements serait probablement beaucoup trop importante pour nos ressources actuelles.

Calculs de compensation dans le SRU

Une autre solution pourrait être de faire les calculs de compensation dans le routeur directement. Malheureusement, cela nécessiterait probablement un changement dans le choix des pièces qui le composent et possiblement une modification du boitier afin de mieux dissiper la chaleur produite par les calculs intensifs. Cette solution, comme celle des calculs directement dans les lampes, réduit les besoins de communication avec l’application infonuagique en déléguant certaines responsabilités aux machines sur le réseau local du client, mais a des limitations matérielles hors de notre contrôle.

Unités de calcul locales

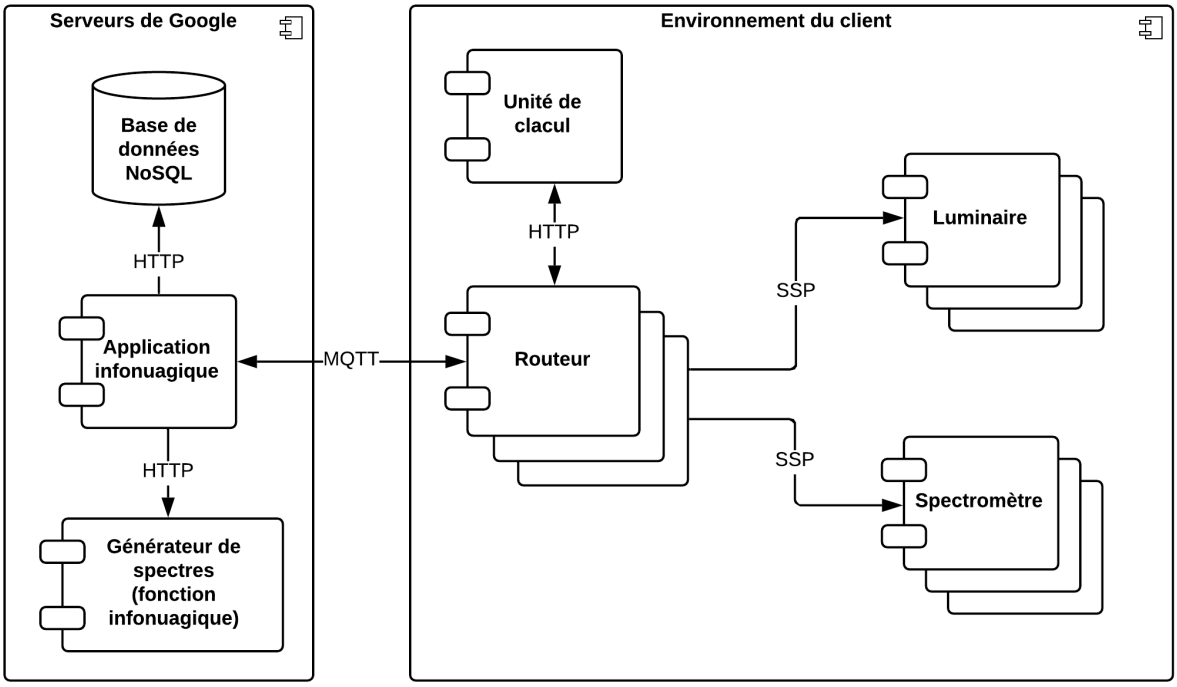

Figure 2 : Architecture avec unités de calcul locales

Une solution très intéressante, pour maximiser les attributs de disponibilité, de fiabilité et même de sécurité, serait l'utilisation d'unités de calcul locales pour calculer les compensations au courant de la journée. Cette solution nécessite l'ajout d'une ou de plusieurs machines dédiées à cette fonction chez chaque client. Cela réduit grandement la dépendance à l'internet puisque la compensation qui se fait à une haute fréquence se fait dans l'environnement local du client. Le trafic réseau vers l'extérieur est alors grandement réduit. Cette solution devrait être couplée à des changements majeurs à la structure et aux fonctions des routeurs. Le routeur qui n'a actuellement que le routage comme fonction devra assumer un rôle de conservation de l'état des lampes afin de maximiser le potentiel de cette solution. De cette manière, le routeur n'aura pas besoin de toujours faire des requêtes à l'application infonuagique afin d'obtenir l'information relative aux recettes du client. Aussi, il sera beaucoup plus facile d'ajouter de nouvelles machines que de modifier celles qui existent déjà.

Calculs de compensation dans le nuage

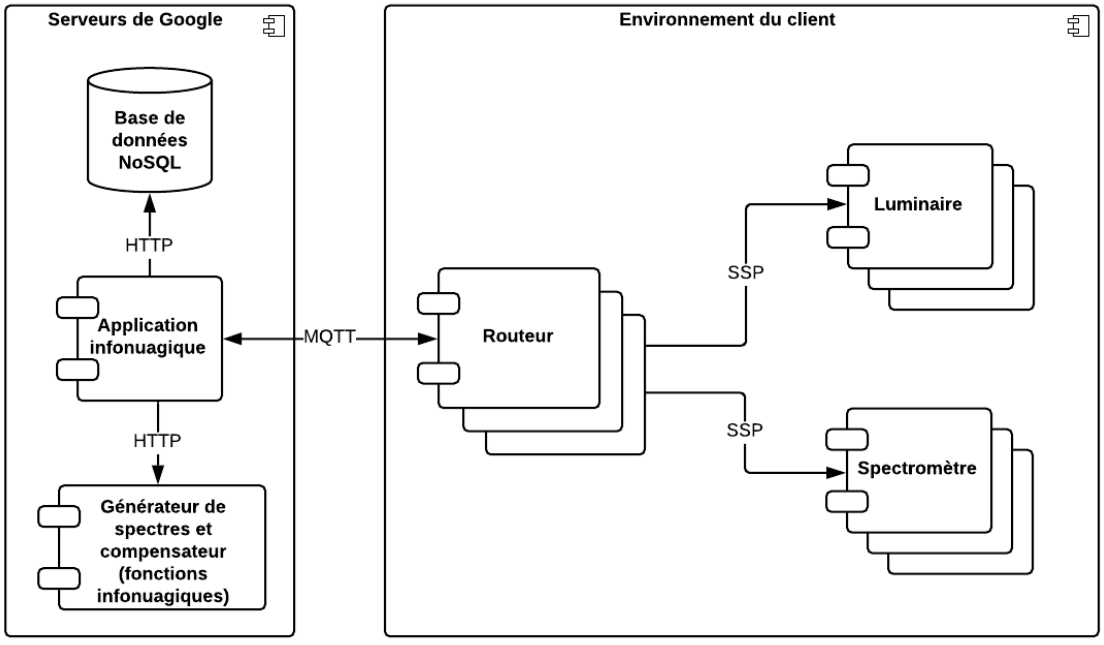

Figure 3 : Architecture avec compensateur dans une fonction infonuagique

L’exécution des calculs de compensation dans le nuage de Google est ce qui se rapproche le plus de la structure logicielle déjà en place chez Sollum. En effet, l’algorithme qui génère les spectres utilise une logique très proche de celle de la compensation et celui-ci se trouve dans une fonction cloud. Y ajouter cette nouvelle fonctionnalité serait donc assez naturel et facile. Par contre, cette solution occasionne un fort trafic entre l’environnement des clients et les serveurs de Google.

Solution sélectionnée

Étant donné les limites de temps et de ressources humaines, nous avons décidé d’opter pour la solution qui s’intègre le mieux avec les systèmes déjà présents, c’est-à-dire la solution de la compensation dans une fonction cloud. La portée de ce projet ne nous permet malheureusement pas de faire de modifications majeures à la conception matérielle des lampes ou des routeurs alors ces options ont été écartées assez rapidement. La solution d’unités de calcul local serait probablement l’approche la plus bénéfique à implémenter dans le futur étant donné la diminution des communications qui serait engendrée. Nous suggérons à Sollum de considérer sérieusement cette solution et tout ce qu’elle implique au niveau matériel. Pour notre part, la partie logicielle serait assez facile à implémenter pour satisfaire cette nouvelle structure.

Positionnement des capteurs

Solutions potentielles

Le positionnement des capteurs peut paraître anodin, mais il a un grand impact sur la façon d’implémenter la logique des algorithmes de compensation.

Positionner les capteurs au-dessus des lampes

Placer les capteurs au-dessus des lampes nous permet d’isoler très facilement la lumière produite par le soleil. De cette manière, nos algorithmes peuvent prendre ces données directement pour effectuer les calculs de compensation. En revanche, cette solution ne nous donne aucune information à propos de la lumière produite par les lampes.

Positionner les capteurs en dessous des lampes

Placer les capteurs en dessous des lampes, au niveau de la canopée des plantes par exemple, complexifie légèrement nos algorithmes étant donné que l'on doit être en mesure de séparer la lumière produite par nos lampes et celle du soleil avant d'effectuer quelque calcul que ce soit. Cependant, de nombreux avantages viennent avec cette solution. Avoir accès à la lumière projetée chez le client à tout moment peut nous permettre d'assurer une qualité supérieure étant donné que nous voyons directement le spectre réel que les plantes reçoivent.

Solution sélectionnée

Pour cet aspect, nous avions initialement sélectionné le positionnement des capteurs en dessous des lampes pour les avantages de diagnostic de la qualité, mais le département de mécanique qui s'occupe de concevoir les supports nous a finalement imposé le positionnement au-dessus des lampes puisque l'installation est beaucoup plus simple à faire de cette manière.

Optimisation du système

Solutions potentielles

En début de projet, les calculs de compensation et de rectification étaient minimalement fonctionnels, mais peu performants. Il y avait donc un besoin d'optimisation de la performance des calculs de compensation pour maximiser l'efficacité et la réactivité du système aux changements environnants afin que le spectre sortant tende vers le spectre cible autant que possible. Afin de réduire les temps de traitement, nous avons considéré plusieurs solutions qui sont les suivantes.

Changer de langage

Nous avons tout d'abord considéré de changer de langage pour tenter de gagner en performance. Le langage, initialement Python, pourrait donc être remplacé, pour certaines fonctions ou pour l'ensemble du projet, par un autre langage comme le C, plus adapté aux besoins du projet, dans l'espoir de gagner de l'efficacité dans le traitement des données.

Optimiser la fonction d'optimisation

La fonction d'optimisation sert à calculer la combinaison optimale de l'intensité de chacun des canaux, c'est-à-dire, les valeurs d'intensité qui permettent de former le spectre se rapprochant le plus de la cible. Cette fonction représente donc une très grande majorité de l'effort de calcul global nécessaire pour la compensation du spectre en temps réel. Une solution pourrait donc être de tenter d'optimiser cette fonction, en changeant de librairies ou les paramètres de la fonction actuellement en place, par exemple.

Optimiser la fonction d'évaluation de l'erreur

La fonction d'optimisation mentionnée ci-dessous utilise une fonction d'évaluation afin d'évaluer la précision de la solution trouvée, tout au long du calcul d'optimisation. Autrement dit, la fonction d'optimisation évolue afin que son résultat tende à minimiser le résultat retourné par la fonction d'évaluation de l'erreur qui retourne le pourcentage d'erreur entre le spectre généré par la combinaison actuelle et le spectre cible. Cette fonction est donc également un point central du calcul de la compensation du spectre, car elle sera appelée pour évaluer chacune des combinaisons parcourues par la fonction d'optimisation, soit plusieurs milliers d'itérations. Une solution intéressant dans le but d'améliorer la performance du calcul pourrait donc être d'optimiser la fonction d'évaluation de l'erreur, laquelle a un impact colossal sur le temps de traitement de données.

Prétraitement des fichiers d'acquisition

Les fichiers d'acquisition correspondent à la contribution (en intensité lumineuse) au spectre de chacun des canaux, à une intensité donnée pour une longueur d'onde donnée. En d'autres mots, il s'agit de données qui ont été recueillies lors d'un banc d'essai, pour chacune des lampes, qui détermine l'intensité lumineuse résultante des différents canaux de la lampe, en fonction de l'intensité appliquée à chacun des canaux et pour une longueur d'onde donnée. La fonction d'optimisation mentionnée précédemment base ses calculs sur les fichiers d'acquisition de la lampe pour obtenir le spectre selon une combinaison d'intensité donnée. Chacun des fichiers d'acquisition est propre à la lampe, puisque les données varient un peu d'une lampe à l'autre. De ce fait, le fait d'externaliser le traitement des fichiers pourrait être bénéfique puisque le traitement pourrait être effectué une seule fois (lors de l'installation de la lampe) et son résultat pourrait toujours être réutilisé par la

suite. Le fait de traiter ces fichiers d'acquisition pourrait donc avoir un impact positif sur la performance du système, en réduisant le temps de traitement du fichier qui est très volumineux.

Utiliser de nouvelles librairies dans le langage actuel

Pour optimiser la performance du système, nous pourrions également considérer d'autres librairies pour les fonctions principales du calcul de la combinaison optimale. Cette solution pourrait être considérée autant lors de l'optimisation des fonctions mentionnées précédemment que dans une optique de révision globale du code pour tenter de trouver des librairies plus efficaces que celles qui sont actuellement utilisées.

Faire du multithread

Pour optimiser la performance du système, nous pourrions également considérer de faire du multithread, soit exécuter plusieurs *threads* en parallèle pour une tâche ou une fonction centrale. Tout comme l'utilisation de nouvelles librairies, cette solution pourrait être considérée d'un point de vue global ou encore appliquée pour une fonction en particulier.

Solutions sélectionnées

Afin de réduire le temps nécessaire au calcul de compensation, nous avons décidé d'optimiser la fonction d'optimisation et la fonction d'évaluation de l'erreur, comme celles-ci représentent la très grande majorité de l'effort de calcul, tel qu'expliqué précédemment. Effectivement, un gain d'un millième de seconde sur la fonction d'évaluation de l'erreur peut avoir un impact de plus de 10 secondes sur le temps de calcul global, puisqu'elle est appelée près de 10 000 fois lors de l'exécution du calcul. Une autre portion importante du temps de traitement se situe lors du traitement des fichiers d'acquisition qui sont très volumineux. La solution de prétraitement des fichiers d'acquisition a donc également été retenue pour améliorer la performance du système. Cependant, comme les autres portions de code de l'entreprise, incluant la preuve de concept, étaient toutes réalisées avec le même langage (Python), nous avons écarté la solution qui impliquait de changer le langage, car nous avons jugé que cette tâche aurait demandé un effort trop important pour les résultats qu'elle pourrait avoir sur la performance du système. Nous avons aussi décidé de considérer plusieurs librairies et l'architecture à plusieurs *thread* lors de l'optimisation

des fonctions retenues, ce qui sera décrit dans la section dédiée à l'implémentation des solutions choisies.

Fournisseur de cloud

Puisque Sollum est une entreprise encore à ses débuts, il est pertinent de revoir les technologies déjà utilisées à l'interne en relation avec le projet de fin d'études avant d'entamer le développement du projet. Les différentes couches du système sont relativement encore en début de développement. Il est donc important de contre-valider les choix de technologies qui ont été faits aux débuts de l'entreprise pour voir si ces choix permettent un système facilement évolutif et maintenable à long terme. Nous avons donc intégré une partie dans notre projet qui était de réévaluer le fournisseur infonuagique utilisé.

Solutions potentielles

Actuellement, dans le marché, on peut noter que les trois principaux fournisseurs de services infonuagiques sont respectivement Amazon Web Services, Microsoft Azure et Google Cloud. AWS existe depuis 14 ans, tandis qu'Azure existe depuis 9 ans et GCP 8 ans. Cela fait en sorte qu'AWS a davantage de maturité que les deux autres et a réussi à mieux se positionner dans le marché en étant un des premiers leaders. Sur le plan de la maturité et les parts de marché, AWS domine également.

Il est important de s'intéresser à la performance de chacun des fournisseurs d'infonuagique puisque la latence de calcul aura un impact direct sur la responsivité du système de rectification du spectre qui est très intense en quantité de calculs. Un test a été réalisé par Ben Slater de chez instaclustr sur les trois grands fournisseurs d'infonuagique. Le test a été réalisé pendant une heure sur un différent nombre de « *clusters* ». On peut voir que généralement GCP se situe en tête du côté de la performance.

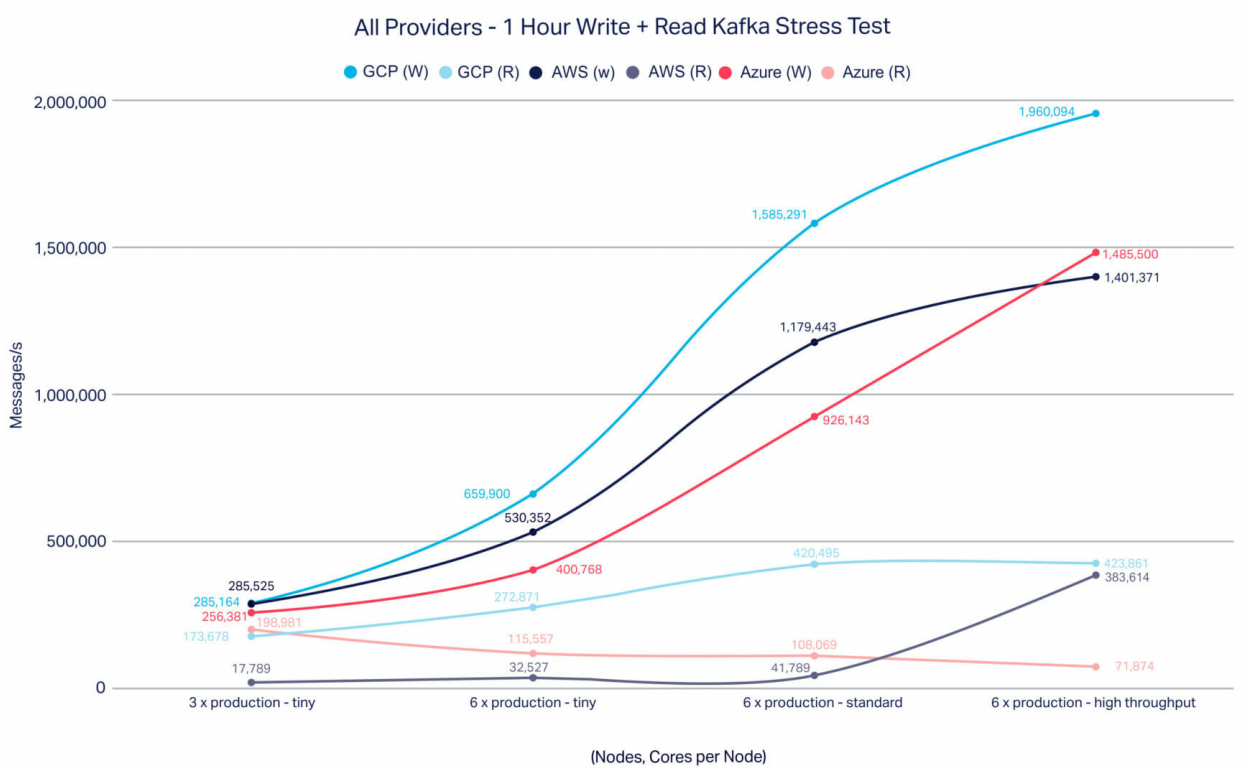

Figure 4 : Comparaison de la vitesse des différents services

Source : SLATER

Solution sélectionnée

Concernant les coûts, on peut constater une tendance générale que Google Cloud Platform est légèrement moins coûteux que AWS et Azure pour les besoins actuels de l'entreprise. Le changement de plateforme nécessiterait de modifier considérablement l'implémentation actuelle de la plateforme infonuagique. De plus, Google Cloud a une maturité grandissante et tout porte à croire qu'elle puisse bien supporter les besoins futurs et actuels de l'entreprise en termes de service infonuagique. L'effort nécessaire pour réécrire la couche du système qui est hébergée dans Google Cloud ne serait donc pas justifiable.

Infrastructure Front-End

Solutions potentielles

Du côté Front-end, il a actuellement trois cadriciels qui dominent le marché : Angular, React et Vue.JS. Du côté de la communauté de développeurs, les trois cadriciels sont très populaires et il est assez facile de trouver des développeurs pour travailler sur les trois. En général, Vue.JS semble être plus populaire que les deux autres. Il a 157000 étoiles sur Github vs. 144000 pour React vs. 57000 pour Angular. Donc, on peut voir que Vue.JS est sur une bonne voie pour être supporté longtemps. Pour la facilité d'utilisation, de ce que nous avons remarqué du côté des critiques, Vue.JS est plus flexible et plus facile d'usage. Angular a une plus grande courbe d'apprentissage et React requiert souvent l'utilisation de librairies de tierces parties. Sinon, le plus gros désavantage de Vue.JS est qu'il est le cadriciel le moins mature dans le lot et n'est pas supporté par une grosse entreprise (Google pour Angular et Facebook pour React), mais est plutôt supporté par les communautés Github et Patreon.

Solution sélectionnée

En vue de sa popularité toujours grandissante, Vue.JS semble être le meilleur choix de cadriciel actuellement considérant sa popularité, sa flexibilité, sa facilité d'usage et le fait que Sollum l'utilise déjà.

Base de données

Solutions potentielles

Le système a des besoins assez simples en matière de base de données. Malgré cela, quelques options s'offrent tout de même à nous.

Base de données NoSQL

Nous pourrions opter pour l'utilisation d'une base de données NoSQL de type clé-valeurs. Cela est très facile à implémenter et à manipuler. Par contre, ce genre de base de données peut facilement devenir désordonné et mal géré lorsqu'elle prend de l'expansion. Aussi, dans certains cas, des données peuvent être dupliquées inutilement.

Base de données relationnelle SQL

Une autre solution serait d'utiliser une base de données SQL. Pour ce faire, nous devrions faire la conception de la base de données en tables au préalable et l'implémenter par la suite. Généralement le SQL demande un peu plus de structure dans son utilisation et peut être limitant si on veut effectuer souvent des changements dans notre modèle.

Stockage de données à froid (Cold Storage)

Une solution de stockage à froid pourrait être considérée. Le stockage à froid est généralement utilisé pour des données qui sont très rarement utilisées et qui n'ont donc pas besoin d'être accessibles rapidement et efficacement.

Solution sélectionnée / Scope final / Justification

Nous avons choisi d'utiliser une base de données NoSQL de type clé-valeurs pour quelques raisons. Tout d'abord, Sollum utilise déjà ce type de base de données dans son application infonuagique alors il serait très facile de faire de même pour l'information de compensation. Ensuite, comme Sollum est une start-up, tout tend à changer rapidement, alors nous préférons opter pour une solution qui pourrait être facilement modifiée. Aussi, comme les données sauvegardées sont surtout des valeurs de spectres générés dynamiquement, la duplication de données n'est vraiment pas un problème que l'on doit gérer; la grande majorité des valeurs sera unique.

Implémentation et intégration

Restructuration de la preuve de concept

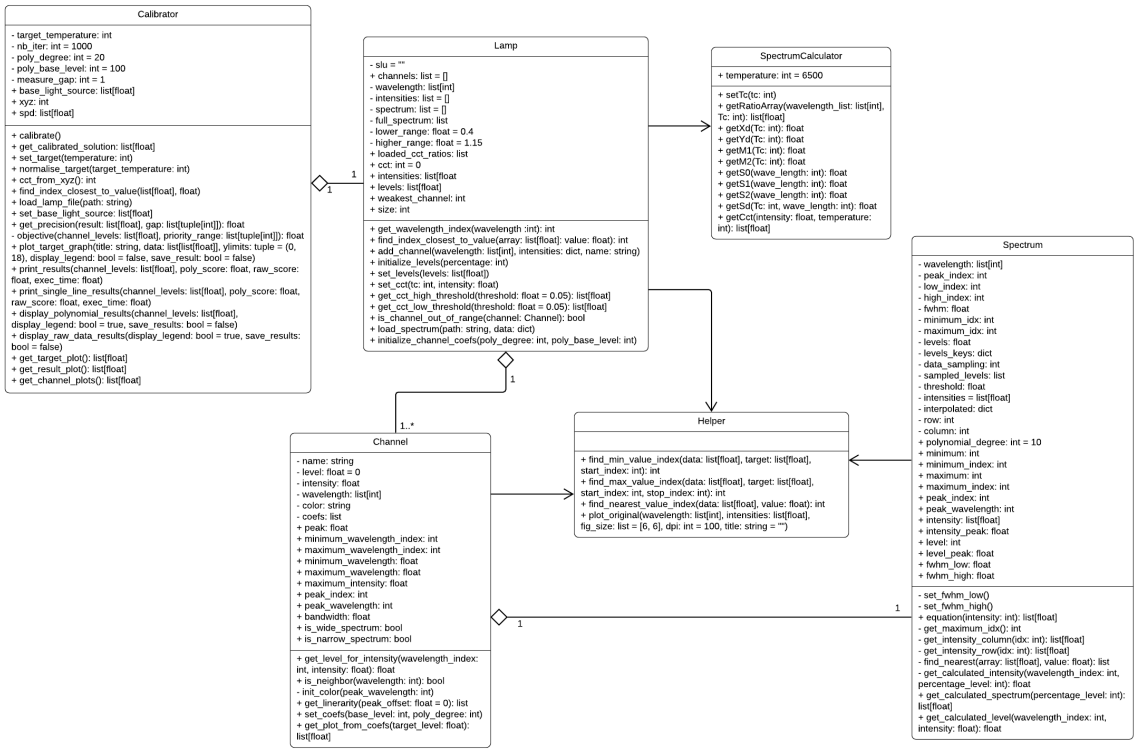

Figure 5 : Diagramme de classes de la preuve de concept originale

Initialement, la preuve de concept nous semblait très complexe pour le peu de fonctionnalités qu'elle supportait. Nous avons donc analysé le fonctionnement en profondeur pour comprendre la logique mathématique utilisée pour faire les calculs de spectres et d'optimisation. Nous avons vu que l'un des défis principaux de l'algorithme est de trouver la courbe d'un canal pour une valeur d'intensité donnée. En effet, le programme ne connaît que ces valeurs pour les intensités de 25%, 50%, 75% et 100%. Il utilise donc une régression linéaire pour calculer les valeurs d'intensités intermédiaires. Pour accélérer ce calcul de régression durant l'exécution de la fonction d'optimisation, l'algorithme génère initialement des polynômes qui représentent chaque canal et qui peuvent ensuite être rapidement modulés par un facteur afin de trouver directement la courbe à une intensité voulue.

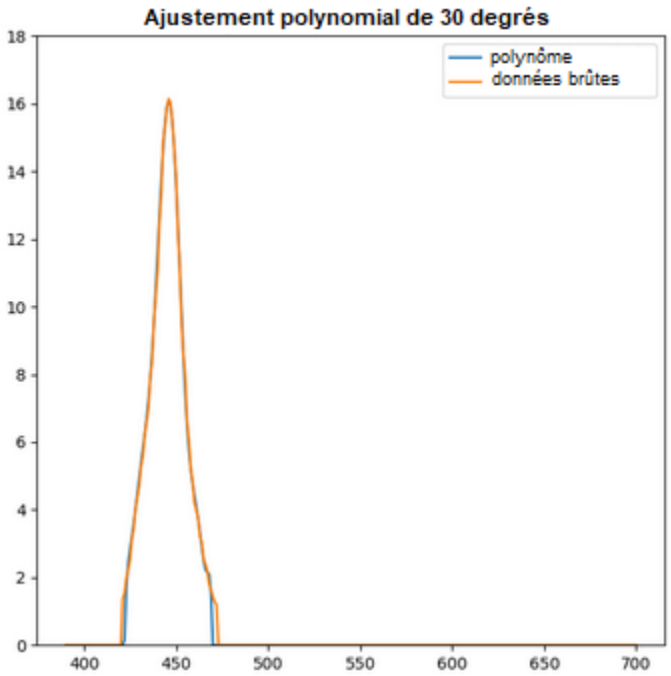

Figure 6 : Exemple de représentation par polynômes de la preuve de concept originale

Cette méthode nous semblait peu efficace alors nous avons demandé à l'équipe de Sollum pourquoi ils n'utilisent pas simplement les données spectrales à leur intensité maximale en les multipliant par des valeurs entre 0 et 1 pour obtenir toutes les autres courbes possibles. Ils nous ont indiqué qu'ils croyaient que cette méthode était fautive puisque la courbe dérivait en fonction de l'intensité électrique. Nous avons donc effectué des simulations pour valider ce phénomène et avons rapidement vu que la SF3 n'avait pas ce problème. Nous avons donc pu simplifier grandement la logique de l'algorithme en prenant seulement les données des fichiers d'acquisition à pleine puissance comme valeur de spectre pour rapidement obtenir chaque courbe. Nous avons donc évacué toute la logique de polynômes et de régression linéaire.

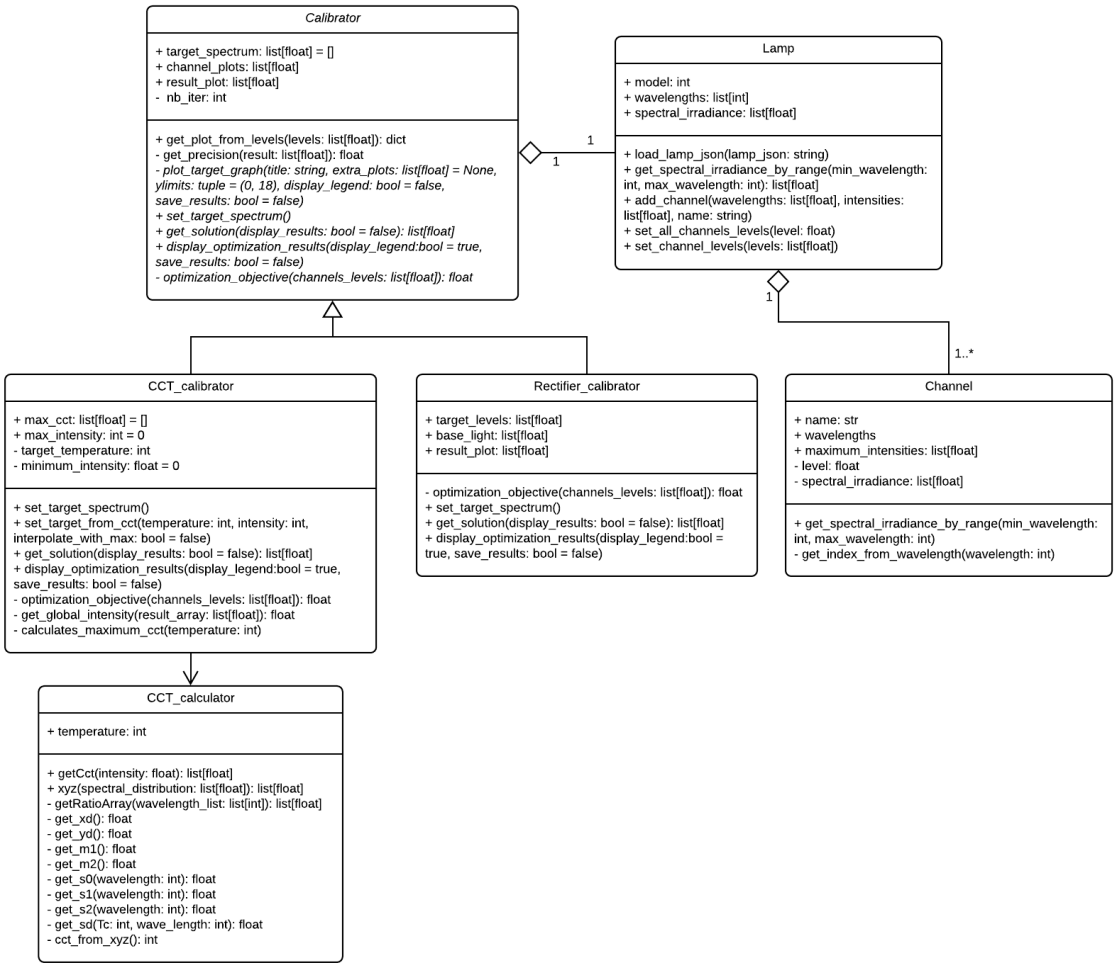

Figure 7 : Diagramme de classes de la preuve de concept après notre réusinage

Ensuite, en ajoutant l'algorithme de compensation au programme de génération des spectres, le code de la preuve de concept se retrouvait avec plusieurs parties de code dupliquées. Cela rendait le tout assez difficile à maintenir. Nous avons donc utilisé l'héritage avec le patron « template method » pour que le projet puisse supporter différents types de calibreurs. Dans notre situation, nous avons le calibreur par CCT qui permet de générer des spectres et le calibreur de rectification qui permet d'effectuer les opérations de compensation. Nous avons aussi évacué plusieurs fonctions utilitaires qui ne trouvaient plus d'utilité avec la nouvelle manière de procéder sans les polynômes et les régressions linéaires. Globalement, le résultat final est beaucoup plus facile à comprendre, à maintenir et à réutiliser.

Optimisation du système

Utilisation de Numpy

Le grand avantage de Python est sa facilité d'utilisation. Malheureusement, il n'est pas du tout optimisé en termes de performance. Une solution à ce problème est la librairie NumPy. Cette librairie est partiellement écrite en C afin d'optimiser la vitesse de la manipulation de données. Nous avons donc remplacé toutes les structures de données par des structures de Numpy, puis nous avons utilisé plusieurs fonctions de la librairie pour effectuer les manipulations mathématiques. Le résultat a été une accélération d'environ 80% pour l'algorithme de rectification.

Prétraitement des fichiers d'acquisition

Tel qu'expliqué précédemment, les fichiers d'acquisition contiennent essentiellement des données issues d'un banc d'essai effectué pour une lampe donnée et à des intensités données pour les onze canaux. Ce sont donc des valeurs d'intensité lumineuse pour chacun des canaux, en fonction de l'intensité appliquée à ce canal et de la longueur d'onde. Il y a un fichier d'acquisition par lampe puisque le spectre sortant peut varier en fonction de la lampe. Le fichier d'acquisition ne change jamais (ou rarement) puisqu'on suppose que la performance de la lampe demeure constante dans le temps et que la sortie sera donc toujours semblable.

Présentement, le fichier d'acquisition (en format JSON) est importé par l'algorithme de rectification et les données sont traitées et formatées à chacun des calculs de rectification. Dans un contexte d'utilisation réelle, le même fichier serait donc traité à chacun des calculs de rectification pour la même lampe, ce qui n'est pas optimal. Puisque le fichier pour une lampe donnée ne change pas, nous avons donc décidé d'externaliser la tâche de traitement des fichiers d'acquisition, afin que celle-ci soit effectuée une seule fois au début du processus. Le résultat de ce traitement pourrait donc être utilisé directement par l'algorithme de rectification à chacune des itérations. Pour ce faire, nous avons donc commencé par analyser les données brutes recueillies, ainsi que les besoins que nous avons en termes de précision pour que le calcul de rectification demeure précis et efficace.

Le fichier d'acquisition contient actuellement plus de 70 000 lignes, soit les valeurs d'intensité lumineuse pour les onze canaux, à quatre valeurs d'intensité différentes et pour plusieurs milliers de valeurs de longueurs d'onde entre 250 et 800nm.

En effectuant l'analyse des données, nous avons constaté que les valeurs d'intensité lumineuse pour un canal donné à une longueur d'onde donnée étaient proportionnelles à l'intensité appliquée à ce canal. Autrement dit, les données associées quatre valeurs d'intensité (25%, 50%, 75% et 100%) évoluent de manière proportionnelle, faisant en sorte qu'il est possible de garder seulement les données associées à 100% d'intensité, puisqu'il est possible de déduire les autres en fonction de celle-ci (c'est-à-dire en multipliant la valeur par 25%, par exemple). Nous avons donc été en mesure de réduire la taille des données d'acquisition de 75% de cette manière.

Il a également été déterminé que seules les longueurs d'onde entre 380 et 800nm avaient un impact notable sur la croissance des plantes, faisant en sorte que près de 20% des données étaient inutiles, soit les valeurs entre 250 et 380nm. Maintenant que l'ensemble des valeurs inutilisables avaient été retirées des données d'acquisition, nous avons procédé à l'analyse des besoins de précision, afin que le résultat de l'algorithme de rectification repose sur des données ayant un niveau de précision suffisant et le moins de données redondantes possible.

Pour ce faire, nous avons donc testé plusieurs fonctions d'interpolation à l'aide de différentes librairies, soit *Scipy*, *pandas* et *Numpy*, en utilisant une précision à l'unité près (donc des valeurs de longueurs d'onde de 380, 381, 382, etc.). Suite à plusieurs tests de variation de paramètres avec les différentes fonctions, nous avons constaté que la fonction *scipy.interpolate*.*UnivariateSpline* fournissait le résultat le plus précis. Les tests ont été réalisés en calculant la distance moyenne entre les points et la courbe résultante de l'interpolation, la distance moyenne la plus faible représentant donc l'interpolation la plus représentative des données.

La même technique de test a ensuite été utilisée pour faire varier les paramètres pertinents de la fonction, comme le degré de la fonction d'interpolation utilisée et la valeur de « *smooth* ». La fonction d'interpolation finale utilisée a donc été *UnivariateSpline* de la librairie *scipy*, avec un degré cubique et un *smooth*=0.1.

Comme les fichiers d'acquisition pour les lampes qui sont installées au même endroit (chez le même client) diffèrent très peu, nous avons également implémenté un script permettant de calculer les données d'acquisitions moyennes de plusieurs lampes, afin de combiner le tout dans un même fichier d'acquisition. De cette manière, le calcul du spectre peut s'effectuer une seule fois pour l'ensemble des lampes (c.-à-d. avec le fichier contenant la moyenne des fichiers d'acquisition des lampes du même client), au lieu d'être effectué une fois pour chacune des lampes. Cette mesure nous a également permis de gagner de la rapidité d'exécution, dépendamment du nombre de lampes en place chez le client.

Optimiser la fonction d'évaluation de l'erreur

Comme expliqué précédemment, la fonction d'évaluation de l'erreur sert à évaluer la précision du spectre résultant de la combinaison trouvée en calculant l'écart entre celui-ci et le spectre cible. Cette fonction est donc utilisée à chacune des itérations de la fonction d'optimisation, soit plusieurs milliers de fois.

Initialement, le calcul de l'erreur était effectué à l'aide d'une simple boucle *for* qui parcourait chacune de valeurs une à une et calculait le pourcentage d'erreur à l'aide de la formule classique :

$$Erreur = (Résultat - Cible) / Cible$$

L'erreur totale était donc calculée en effectuant la moyenne de chacune des valeurs d'erreur ainsi obtenue.

Afin de réduire le temps de calcul, nous avons tout d'abord considéré d'autres méthodes de calcul d'erreur, comme l'utilisation de la moyenne de l'erreur quadratique moyenne (*Mean Square Error)* pour chacune des valeurs. Les tests ont cependant révélé que cette méthode

ne réduisait pas significativement le temps de traitement, en plus de nuire à la qualité du résultat obtenu. Le calcul de l'erreur totale en utilisant l'erreur quadratique moyenne a donc été écarté.

Par la suite, nous avons testé une méthode de calcul d'erreur à l'aide de l'aire sous la courbe, soit l'utilisation d'intégrales pour calculer l'aire entre les courbes des deux spectres comme mesure de distance entre les points. On comprend donc que plus l'aire entre les deux courbes est petite, plus l'erreur est petite, faisant de cette valeur la valeur à minimiser. Toutefois, le résultat s'est avéré semblable au calcul de l'erreur quadratique moyenne, soit peu de gain au niveau de la performance et une réduction marquée de la qualité du spectre obtenu.

À la suite de nombreux tests peu concluants, nous avons décidé de poursuivre avec la solution actuellement en place, puis d'optimiser celle-ci. Cette solution comportait principalement deux portions à optimiser, soit le calcul de la distance entre chacun des points (qui est actuellement effectué avec une boucle *for* qui parcourt chacune des valeurs et fait la soustraction) ainsi que la formule utilisée pour le calcul de l'erreur présentée précédemment.

L'enjeu principal que nous avons rencontré avec ce calcul de l'erreur est la gestion des erreurs provoquées par la division par zéro. Nous devions donc trouver une solution qui faisait en sorte que les pourcentages d'erreur attribués lorsque la valeur cible est nulle sont réalistes et affectent le moins possible la qualité du résultat. La solution actuellement en place pour répondre à ce problème était d'attribuer une erreur de zéro lorsqu'on rencontrait une division par zéro, ce qui nuisait grandement à l'intégrité de notre calcul d'erreur. Suite à l'essai de nombreuses solutions pour pallier à ce problème, nous avons décidé de remplacer chacune des valeurs nulles dans le spectre cible par une valeur non nulle se rapprochant de zéro. Pour ce faire, nous avons utilisé la fonction « *where* » de la librairie *Numpy*, soit la fonction la plus rapide que nous ayons trouvée pour cette tâche. Nous avons par la suite procédé à l'essai de plusieurs valeurs différentes pour remplacer les valeurs nulles (ex. : 0.1, 0.01, 0.001). En effectuant des tests avec un échantillon de valeur, nous avons sélectionné la valeur 0.01.

Pour ce qui est de la méthode utilisée pour parcourir chacune des données, nous avons utilisé la librairie *numpy* afin de manipuler les tableaux de données (*array*) au lieu des données une à une. La fonction finale obtenue pour le calcul de l'erreur a donc permis de réduire le temps de traitement total d'environ 25%, en plus de grandement simplifier le code.

Nous avons par la suite tenté d'améliorer la performance davantage en utilisant seulement une valeur sur deux pour le calcul de la précision, mais cette mesure n'avait pas d'impact significatif sur la performance et affectait nécessairement la précision. Cette solution a donc été écartée.

Optimiser la fonction d'optimisation

La fonction d'optimisation est utilisée pour trouver le minimum local de la fonction d'évaluation de l'erreur, c'est-à-dire, le spectre qui se rapproche le plus du spectre cible. Cette fonction représente donc l'effort de calcul principal de l'algorithme de calcul de la rectification. La fonction utilisée pour accomplir cette tâche est « *differential_evolution* » de la librairie Scipy. Comme le travail de comparaison avec d'autres méthodes d'optimisation avait déjà été effectué lors de la preuve de concept, nous nous sommes concentrés sur l'optimisation des paramètres de la fonction déjà en place, soit « *differential_evolution* ».

Contrairement aux sections précédentes pour lesquelles la précision était très importante, notre processus décisionnel en lien avec les différents paramètres de cette fonction était en très grande partie axée sur la performance du système. La métrique utilisée pour évaluer les différentes solutions possibles était donc la rapidité de l'exécution de l'algorithme.

Pour réduire le temps de traitement, nous avons tout d'abord implémenté une méthode de calcul à plusieurs *threads* en parallèle (*multithreading*). Pour ce faire, la méthode de mise à jour du vecteur optimal devait être paramétrée à *differed*. Une fois le multithreading en place, nous avons principalement fait varier le nombre maximal d'itérations prévues pour l'évaluation de l'ensemble de la population (*maxiter*) ainsi que le nombre de *threads* en parallèle utilisés pour le calcul (*workers*), en comparant les résultats à l'aide du temps de

traitement. L'utilisation de 40 itérations maximales et de plusieurs *threads* en parallèle en utilisant tous les processeurs disponibles (*workers=-1)* nous a permis de réduire le temps de calcul de l'algorithme de plus de 50%.

Modifications de l'architecture du promoteur (SHI)

Modèle procédural

Au début du projet, le routeur qui envoie les commandes aux lampes ne faisait que recevoir les commandes de la plateforme infonuagique, les convertissait dans le format de données nécessaire et envoyait à son tour les commandes à des groupes entiers de lampes ou aux lampes individuellement. Cela fait en sorte que le routeur ne conserve en aucun cas l'information sur les lampes qui sont connectées à celui-ci. Il ne fait qu'envoyer les commandes à toutes les lampes et celles-ci déterminent si les commandes les concernent. Cela pose problème pour ajouter les capteurs au système pour effectuer la rectification. Lorsqu'un capteur enverrait ses données recueillies à la plateforme infonuagique ou à une unité locale de calcul, le routeur n'aurait aucun moyen de connaître quelles lampes et quels groupes font partie de quelle zone de capteur pour pouvoir envoyer la commande.

Pour pallier le problème, il a été convenu d'effectuer la réingénierie du routeur pour passer d'un modèle procédural à un modèle objet. Voici à quoi ressemblait le modèle procédural au début du projet :

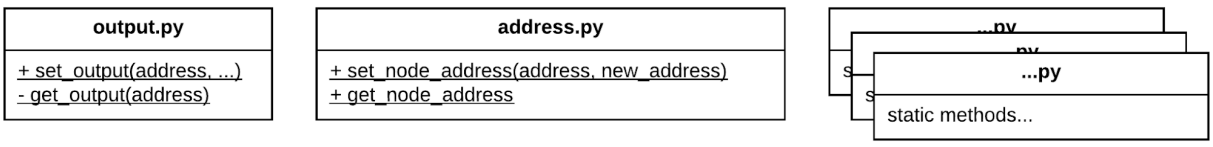

Figure 8 : Diagramme de scripts initial avant réingénierie

On peut voir que toutes les méthodes sont statiques et sont simplement réparties dans différents scripts sans aucune classe. Essentiellement, on ne fait que recevoir les

commandes de la plateforme infonuagique et on appelle les commandes appropriées du firmware.

Ancien modèle objet qui avait déjà été implémenté

Un modèle objet avait déjà été réalisé par le passé par un programmeur de chez Sollum, mais un des problèmes était qu'il y avait une classe Lampe qui devenait un peu une « god class », dans le sens qu'elle contenait beaucoup trop de logique et de données. Il y avait également une classe « Address » dont presque toutes les classes héritaient pour pouvoir envoyer les commandes aux adresses des lampes. Malheureusement, cela ne tenait pas en compte qu'on a plusieurs types d'adresses. L'intervalle d'adresses pour les lampes n'est pas le même que pour les adresses de groupes ou l'adresse de diffusion « broadcast address ». Également, cela ne fait pas vraiment de sens qu'un module de spectre hérite d'une classe d'adresse. Un spectre n'est pas une adresse. Un autre problème que cette implémentation avait était que toutes les classes avaient plusieurs méthodes pour convertir les objets en json et reconvertir en objet. Cela n'était pas en ligne directe avec le principe de la séparation des préoccupations « separation of concerns » puisque ces classes convertissaient les données, enregistraient les données et envoyaient des commandes. Nous avons donc pris en compte les différents problèmes soulevés que l'ancienne implémentation objet avait pour ne pas reproduire les mêmes erreurs.

En recueillant ces différentes informations, nous avons également découvert qu'il y avait un autre besoin qui justifiait de revenir à un modèle objet et dont devrait répondre l'implémentation. On devrait avoir un moyen d'offrir de la persistance d'une quelconque façon, soit par de la sérialisation, par une base de données dans le routeur ayant une table de routage ou par une gestion de fichiers JSON. La persistance permet d'avoir une meilleure gestion d'erreur. Par exemple, le routeur n'enverra plus de commandes à des lampes qui ne sont plus connectées à celui-ci. Également, la persistance permettrait d'avoir un meilleur monitorage. On pourrait renvoyer les données des lampes à la plateforme infonuagique où des techniciens pourraient vérifier l'état de la configuration des lampes à distance et sans envoyer continuellement des requêtes aux lampes physiques qui ont une capacité limite de requêtes, attribuable au fait qu'on pourrait retrouver des centaines, voire des milliers de lampes connectées à un seul routeur. La persistance dans le routeur permet

aussi de réduire la dépendance à l'internet; en cas de panne, le routeur posséderait l'information nécessaire afin de redonner aux lampes les configurations programmées. Au final, nous avons exclu de la portée du projet d'implémenter la persistance, mais nous avons conçu la réingénierie de façon à permettre ces différentes possibilités. Nous devions finalement et également avoir une optique d'extensibilité, de réutilisabilité et de testabilité.

Modèle objet implémenté

Voici le modèle que nous avons conçu et commencé à implémenter :

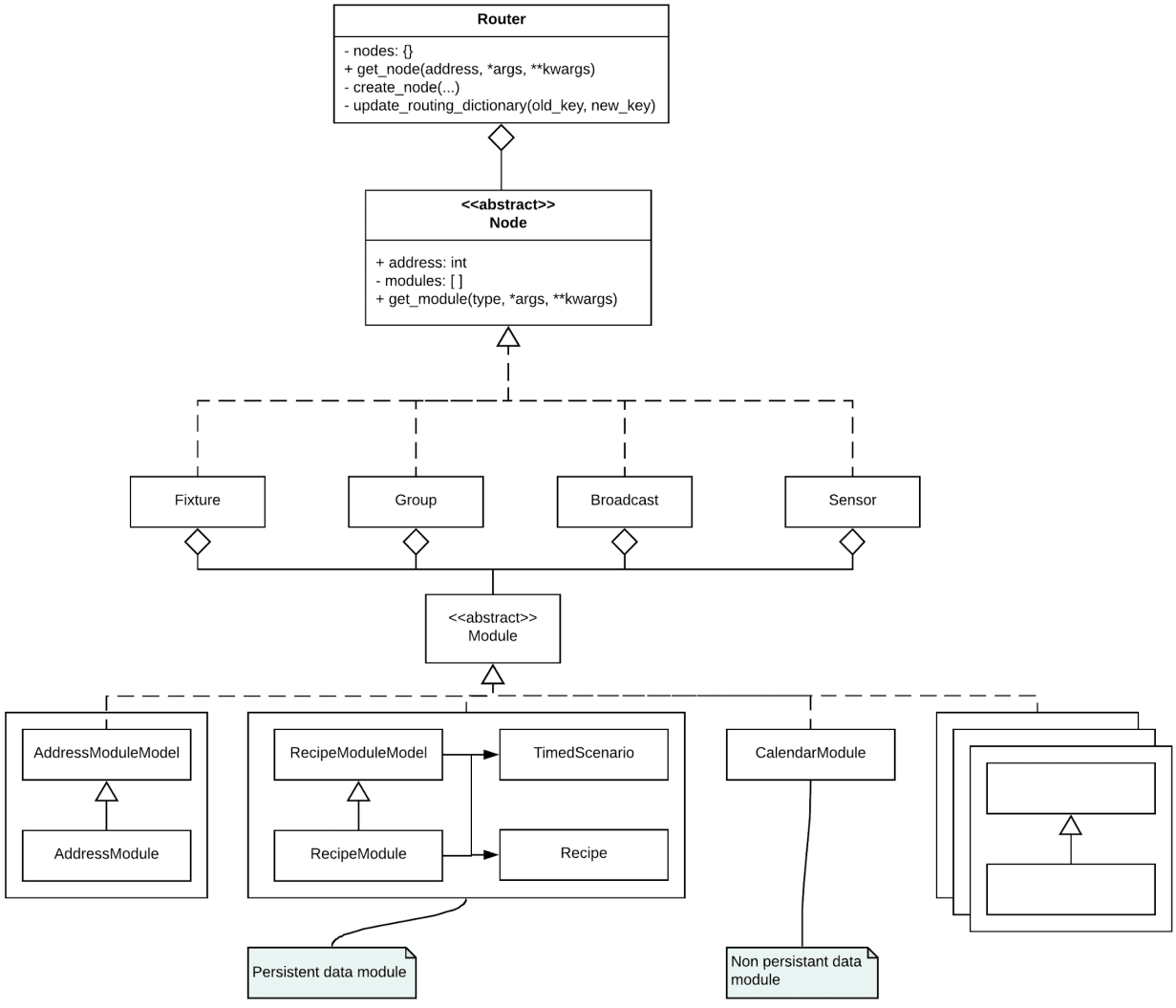

Figure 9 : Diagramme de classes après réingénierie

Essentiellement, on a un routeur qui contient des nœuds qui contiennent plusieurs modules. Le routeur est essentiellement la table de routage. Il contient les nœuds qui sont

connectés au routeur. Chaque nœud a une adresse et l'adresse permet d'identifier si le nœud est une lampe, un groupe, un broadcast ou un capteur seulement en vérifiant dans quel intervalle se situe l'adresse. Il est bien de mentionner, pour la compréhension, que les adresses sont des nombres entiers entre 0x000000 et 0xFFFFFF. Lorsqu'on veut communiquer avec un nœud, on n'a qu'à appeler la fonction « *get_node* » qui va soit retourner l'instance d'un nœud existant ou retourner une première instance du nœud s'il n'existait pas déjà. Cela nous permet de nous assurer qu'on a seulement un maximum d'une instance par nœud.

Ensuite, chaque nœud peut contenir plusieurs modules, mais seulement une instance maximum de chaque. Chaque module représente un module dans le « firmware », par exemple, le module d'adresse, le module de recette ou le module de sortie (output). Lors de notre analyse et prise d'information, nous avons constaté qu'il y a environ les deux tiers des modules dont nous avons seulement besoin d'envoyer les commandes directement aux lampes sans conserver les données envoyées, par exemple, lorsqu'on synchronise l'horloge des lampes. Il n'est pas utile de conserver l'information sur l'heure de la lampe et il n'est pas utile non plus de conserver les données concernant les configurations de base des lampes. Pour cette raison, nous avons permis dans notre conception de faire la distinction entre les modules de données persistantes et les modules de données non persistantes. Tout simplement, les modules de données non persistantes ne conservent pas les données envoyées dans les commandes et n'ont pas de classe de modèle. Les modules de données persistantes vont simplement conserver les données dans des objets de modèle qui pourront éventuellement être sérialisés ou avoir des méthodes pour enregistrer leurs données dans une base de données. Les modules pourront communiquer entre eux, au besoin, en demandant simplement à leur nœud l'instance du module auquel ils veulent communiquer. Par exemple, cela est particulièrement utile pour les recettes qui contiennent des scénarios. Le module de recettes doit donc communiquer avec le module de scénarios.

Qualités de l'implémentation

Sur le plan de l'extensibilité, ce modèle est particulièrement extensible puisqu'il permet d'ajouter très facilement d'autres types de nœuds. Il est possible qu'un jour, d'autres types

de capteurs soient ajoutés. Également, avec ce modèle, il est extrêmement simple d'ajouter de nouveaux modules. Il ne suffit que d'hériter de la classe Module.

Sur le plan de la testabilité, il est facile d'écrire des tests unitaires pour chaque module individuellement. Il suffit d'émuler « mock » les commandes du « firmware » et le tour est joué. Le principe de « separation of concerns » a été pris en compte et, donc, chaque class ne s'occupe que d'un besoin précis. Il est facile de comprendre l'implémentation de toutes les classes puisqu'elles contiennent, au plus, cinq méthodes en général. Les tests sont donc faciles à rédiger. Il est bien de noter qu'en termes de couverture de code par les tests, nous avons atteint le 100% pour tout ce qui a été implémenté dans cette réingénierie.

Finalement, sur le plan de la réutilisabilité, on peut penser notamment aux classes Nœud et Module qui sont des classes abstraites desquelles on peut hériter pour obtenir à un seul endroit, notamment, l'adresse du nœud ou, aussi, les instances des modules. Les modules peuvent communiquer entre eux, au besoin, donc ceux-ci n'auront pas à ré implémenter certaines fonctions. On s'assure donc que chaque besoin est implémenté seulement à un endroit.

Intégration dans l'architecture

Au début du projet, la preuve de concept permettait seulement de générer des spectres à partir d'une courbe de CCT, mais ne permettait pas de calculer la rectification à partir d'un spectre. La preuve de concept, architecturalement, était située dans des fonctions cloud. Cela signifie qu'on démarre une instance de l'environnement chaque fois qu'on envoie une requête pour générer un spectre à partir de l'application infonuagique. Cela a le bénéfice d'être indépendant des instances plus lourdes de l'application infonuagique et de simplement pouvoir augmenter le nombre d'instances de l'algorithme de génération de spectres selon les besoins vu que celui-ci est très énergivore.

Au cours du projet, nous avons donc réutilisé ce générateur de spectre pour implémenter le calcul de la rectification. Lorsqu'on effectue une rectification, on parcourt en réalité toutes les couches du système de l'entreprise. Pour implémenter cela, nous avons dû apprendre rapidement comment intégrer les différentes commandes dans toutes les couches.

Voici à quoi ressemble les différents « code repositories » impliqués dans la rectification et quels systèmes sont impliqués.

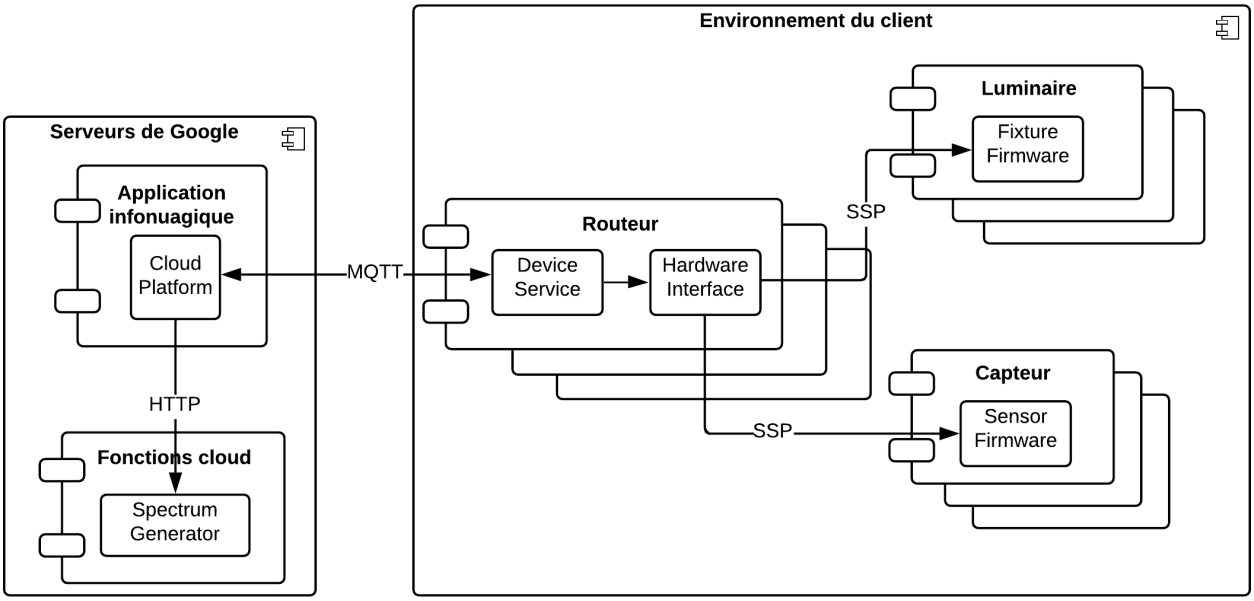

Figure 10 : Architecture démontrant l'interaction entre les différents « code repositories » à travers les composantes

On peut voir premièrement cinq composantes majeures : application infonuagique, fonctions infonuagiques, routeur, luminaires et capteurs. On peut avoir plusieurs instances de l'application infonuagique et plusieurs instances des mêmes fonctions cloud. Le « scaling » est effectué automatiquement. On retrouve ensuite le « Device Service » dans le routeur qui gère la communication avec le nuage et le Hardware Interface qui gère la communication avec les capteurs et les lumières. Dans les capteurs et les lumières, on retrouve également un « firmware ».

On peut voir dans la figure suivante le flux d'exécution pour obtenir les données du capteur. Le flux commence dans une boucle dans le « Device Service » qui est ré exécuté toutes les 30 secondes. En vert, on retrouve les appels et en bleu, on retrouve les réponses des systèmes.

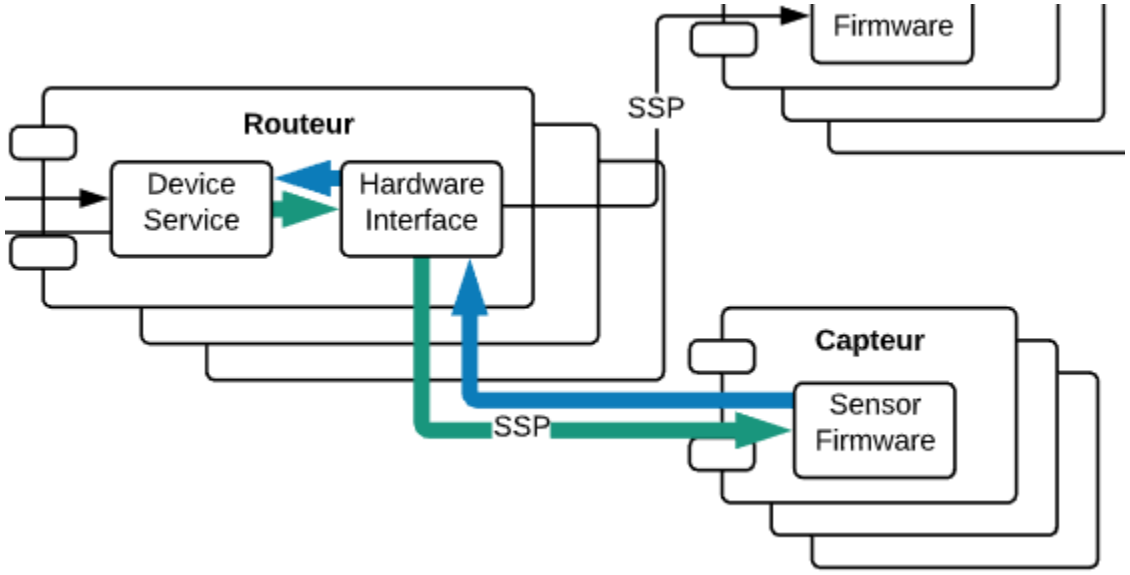

Figure 11 : Flux d'exécution inter système pour obtenir les données du capteur

Actuellement, l'intégration entre le Hardware Interface et le « firmware » du capteur n'a pas pu être implémentée en raison de divers problèmes qui seront discutés plus tard. Un simulateur de capteur a toutefois été implémenté pour pallier ce problème. Nous en discuterons plus en profondeur dans la prochaine section : Simulateur de capteur.

On retrouve dans la figure suivante le flux d'exécution pour calculer la rectification une fois qu'on a obtenu les données. Le flux commence dans le « Device Service ».

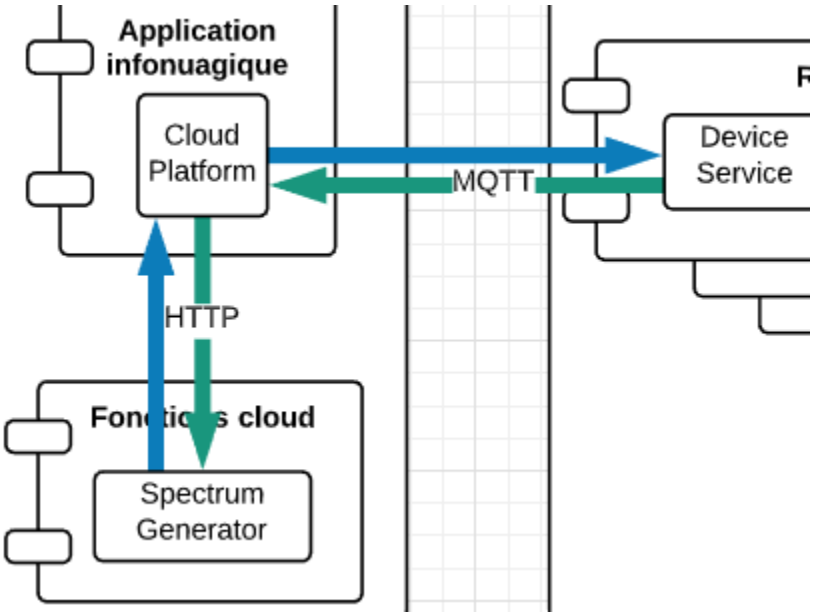

Figure 12 : Flux d'exécution inter système pour calculer la rectification à partir de données du capteur

On retrouve dans la figure suivante le chemin final pour appliquer la rectification dans les lampes. Le routeur envoie la rectification à une ou des lampes, ou même un ou plusieurs groupes de lampes.

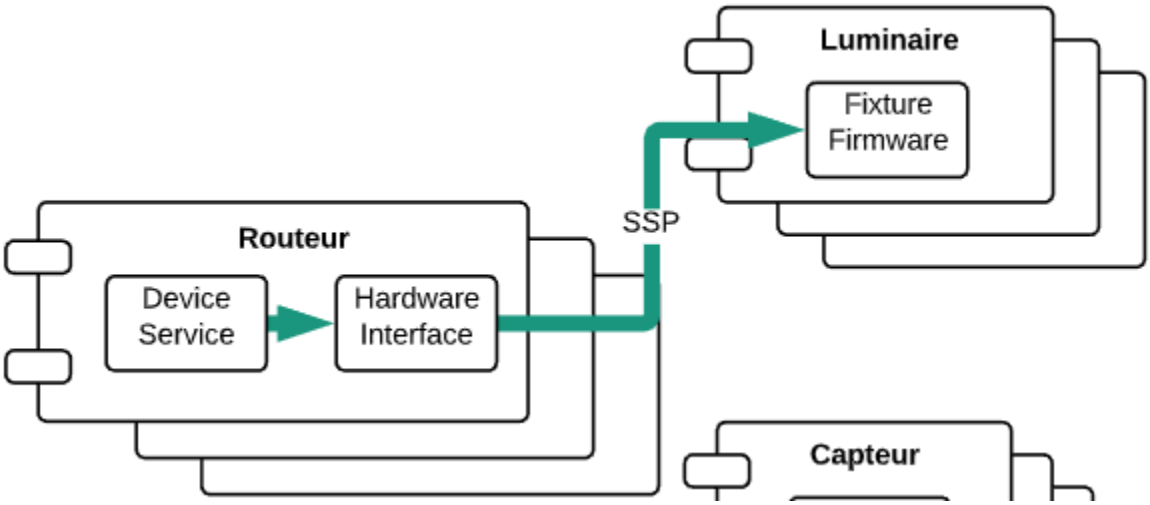

Figure 13 : Flux d'exécution pour appliquer la rectification dans les lampes

Bref, l'intégration n'a pas été simple, car il a fallu acquérir des informations sur tous ces systèmes qui comportent tous des moyens de communiquer différents. L'intégration la plus complexe a été entre le « Device Service » et la Cloud Platform qui ne comportait pas au départ un moyen d'initier une requête à partir du routeur. Auparavant, le routeur ne faisait que répondre aux requêtes de la Cloud Platform. La complexité vient du fait que la plateforme infonuagique est en multi instances et est également le serveur MQTT. Un client MQTT comme le « Device Service » qui souhaite communiquer en MQTT au serveur qui est multi instance est théoriquement impossible. Il doit en réalité communiquer avec toutes les instances du nuage. La solution a été d'ajouter un « endpoint HTTP » dans le nuage qui retourne l'« ID » de son instance au « Device Service ». Ce dernier ajoute l' « ID » de l'instance du nuage dans sa requête. Ensuite, toutes les instances du nuage reçoivent la requête du « Device Service », mais seulement l'instance dont son « ID » correspond à celui dans la requête répond. Sans cela, on aurait malheureusement plusieurs instances du nuage qui répondraient à la requête et ce serait une erreur majeure d'implémentation.

Simulateur de capteur

Étant donné l'impossibilité d'accès aux vrais capteurs durant le développement puisque ces derniers étaient en conception par l'équipe électrique au même moment, nous avons dû trouver une alternative pour nous fournir des données dynamiques en entrée. La preuve de concept initiale utilisait simplement une lecture fixe du soleil prise auparavant, mais dans notre cas, nous voulions valider que notre algorithme pouvait bien s'adapter à toute source de lumière, peu importe sa composition spectrale. Nous avons donc implémenté un simulateur de capteur de lumière positionné directement dans le routeur pour tester la transition des données le plus possible en même temps. Ce simulateur est tout simple; un nombre initial aléatoire est généré pour la première entrée du domaine. Par la suite, une boucle additionne ou soustrait un nombre aléatoire à celui précédent jusqu'à ce que tout le domaine désiré soit complété. De cette manière, nous obtenons une vraie courbe continue qui représente assez bien une lumière réelle au lieu d'une série de valeurs dénuée de sens. Finalement, pour assurer que le spectre ne contient pas de valeurs négatives, nous modulons la courbe vers le haut jusqu'à ce que chaque valeur soit dans le positif, puis on applique une transformation finale à toute la courbe pour lui donner une intensité raisonnable. Nous avons aussi ajouté un paramètre optionnel qui nous permet de lisser la courbe pour tester la tolérance au bruit de notre algorithme de compensation.

Atteinte des objectifs

Installation du compensateur dans l'architecture de Sollum

Cet objectif a été atteint à 95%. L'intégration a été faite dans toutes les couches de l'application et les données peuvent transiter avec succès. Les données peuvent même être appliquées aux lampes physiques. Le seul élément manquant est la connexion avec les capteurs développés par l'équipe d'électrique. Heureusement, nous avons un simulateur du soleil qui peut remplacer temporairement ceux-ci et qui nous assure que notre solution fonctionne.

Optimisation de la performance du système

Cet objectif a été réussi bien au-delà de nos attentes. Alors que nous voulions que la compensation s'effectue en moins de cinq secondes, nous avons réussi à l'exécuter avec constance en moins d'une seconde.

Implémentation de l'algorithme de compensation par recette

Cet objectif a bien été atteint. L'algorithme peut désormais prendre les données de recettes en entrée sans problème pour effectuer la compensation. Par contre, nous n'avons pas lié les vraies recettes des clients à l'algorithme encore étant donné l'absence de vrais capteurs. Cette fonctionnalité de l'algorithme est donc disponible, mais pas encore utilisée pour dans un contexte réel.

Mise en place d'une base de données et conception de graphiques client

Cet objectif n'a pas du tout été atteint. Nous avons dû à quelques reprises changer légèrement la portée du projet à cause des découvertes que nous faisions dans le code source. Au final, à force d'ajouter des éléments, nous avons dû couper à certains endroits. Comme la base de données et les graphiques étaient prévus pour les dernières semaines du projet et que ces fonctionnalités avaient beaucoup moins de valeur pour le promoteur à court terme, nous avons décidé de les évacuer du projet pour nous concentrer sur des problèmes plus critiques comme la structure de certaines couches de leur architecture.

Amélioration des attributs d'extensibilité, de maintenabilité et de réutilisabilité du système

L'objectif d'amélioration des attributs de qualité a été lui aussi réalisé avec succès. L'implémentation de chaque fonctionnalité s'est faite de manière à bien se fondre dans l'architecture en place tout en améliorant la structure existante. Nous avons amélioré la structure du routeur, augmenté grandement l'extensibilité et la réutilisabilité de l'algorithme de génération de spectres et de compensation et simplifié les fichiers d'acquisition des lampes.

Problèmes rencontrés

Synchronisation entre les équipes (Q7-i3)

Le projet a débuté quelques semaines en retard dues au fait qu'il était difficile de se synchroniser avec les membres du département de mécanique et électrique. Le nombre d'étudiants inclus dans le projet a été modifié maintes fois. La communication entre les départements laissait également à désirer. Les éléments d'évaluation semblent différer d'un département à l'autre, ce qui complexifie encore plus les projets multidisciplinaires. Il nous a donc fallu quelques semaines pour démêler toute l'information des différents départements, rencontrer toute l'équipe (7 membres, 1 superviseur, 3 professeurs) et trouver des solutions avec les professeurs concernant les évaluations qui diffèrent. Pour pallier le problème de synchronisation, nous avons décidé de scinder le projet en deux, c'est-à-dire la partie logicielle et la partie mécanique et électrique. Ces deux parties sont assez indépendantes. Nous avons tout de même eu une rencontre par mois pour faire une mise à jour sur les avancements et synchroniser les deux équipes.

Réévaluation de la portée du projet

Vers la moitié du projet, nous nous sommes rendu compte que la rectification ne pourrait être réalisée avec la notion de groupes ou de zones sans effectuer une réingénierie majeure du routeur qui communique avec les luminaires. Actuellement, le routeur ne fait que recevoir des commandes de la plateforme infonuagique et les achemine directement à toutes les lampes. Le routeur ne conserve aucune information sur les groupes de lampes ou sur les lampes qui sont connectées à celui-ci. Puisque les capteurs seront également connectés au routeur, le routeur a besoin de connaître quelles lampes et quels groupes sont physiquement connectés à celui-ci pour envoyer des commandes de rectification à la plateforme infonuagique. Il peut effectivement y avoir plusieurs routeurs dans une serre et on ne veut pas envoyer les commandes de rectification de chaque capteur à toutes les lampes. On a donc dû mettre de côté le fait de faire la rectification par zone pour prioriser la réingénierie du routeur en modèle objet plutôt qu'en modèle procédural. Bien entendu, la réingénierie du routeur est une tâche trop énorme pour la portée du projet. Nous en avons

tout de même commencé une partie puisqu'à la fin du projet, nous continuerons de travailler sur le projet en tant qu'employés et que c'est ce qui était le plus important pour l'entreprise.

Prototype du capteur non complété

En raison de la pandémie mondiale, les membres de l'équipe en mécanique et électrique n'ont pas pu terminer leur prototype de capteur à temps et le « firmware » n'a donc pas pu être terminé non plus. Cela a fait en sorte que nous ne connaissons pas encore le format de données final qui sera reçu des capteurs. Nous avons pour le moment utilisé le format de données d'un spectromètre que l'entreprise utilise déjà qui risque d'être similaire au vrai capteur pour implémenter l'algorithme et l'intégration de la rectification dans toutes les couches du système. Malheureusement, vu que nous ne connaissons pas encore le format de données final des capteurs, il aurait été contreproductif d'entamer la conception du schéma de la base de données pour recueillir les statistiques de rectification. De plus, nous aurions aussi eu besoin de la logique des zones de capteurs pour bien concevoir les schémas.

Aussi, le fait que nous n'avions pas de base de données implémentée ni conçue fait en sorte qu'il ne serait pas possible de concevoir et développer une interface utilisateur pour afficher les statistiques de rectification et d'économies d'énergie puisque nous ne pouvons pas afficher de données si nous ne connaissons pas le format des données stockées.

Nous avons donc concentré nos efforts sur d'autres tâches comme l'optimisation, la réingénierie du routeur, la rectification en intensité, etc.

Enjeux environnementaux (Q9-I2)

Optimisation des algorithmes de calcul

L'algorithme de rectification est extrêmement gourmand puisqu'il est non déterministe et son calcul est réalisé suivant le principe des mathématiques d'approximation comme la minimisation et l'évolution différentielle. Comme démontré précédemment, la durée d'exécution de l'algorithme a été réduite de 98% durant notre projet. Cela est bien en termes d'utilisabilité, mais il faut comprendre que cette optimisation aura un effet énorme sur la quantité de ressources nécessaires à grande échelle. Éventuellement, cet algorithme sera probablement exécuté à chaque 30 minutes pour chaque capteur, pour chaque groupe de luminaires, pour chaque serre de tous les clients de l'entreprise. Si l'algorithme prenait 1 minute à s'exécuter chaque fois avec du calcul très intense, cela aurait nécessairement augmenté les coûts en calcul et eu un impact à long terme sur l'environnement par son utilisation intensive d'électricité vu que celui-ci sera utilisé à grande échelle et très fréquemment.

Réduction de la consommation électrique

La nature même du projet s'inscrit dans une perspective de développement durable puisque son objectif principal est essentiellement de réduire la consommation électrique de l'utilisation de lumière artificielle dans les serres. En réduisant en intensité le spectre fourni artificiellement pour atteindre la même cible en utilisant le spectre fourni naturellement, on consomme évidemment moins d'électricité. Cela engendre donc une atténuation des impacts négatifs des serres dans les pays froids, comme le Canada, sur l'environnement.

Améliorations et travaux à venir

La portée actuelle du projet n'était en réalité qu'une partie à réaliser de tout ce qu'il y a à faire pour rendre le produit commercialisable. Le projet va continuer à évoluer au sein de la compagnie dans les prochains mois pour éventuellement être mis sur le marché. Le développement à venir du projet comporte plusieurs volets :

Calcul de rectification à partir du spectre actuel

Actuellement, nous avons réussi à prouver que l'intégration entre les différents systèmes fonctionne en utilisant simplement un spectre fixe prédéterminé. Un élément absolument nécessaire qui doit être implémenté est le calcul d'interpolation du spectre actuel pour un certain groupe de luminaires. Ce calcul va permettre de rectifier en temps réel le spectre associé au groupe et non pas seulement un spectre fixe qui n'a aucun lien avec le spectre programmé.

Intégration avec les capteurs physiques

Comme discuté plus tôt, un des problèmes qui est survenu durant le projet est le fait que le logiciel interne du capteur n'a pas vraiment pu être implémenté complètement, faute d'avoir un prototype de capteur final en main du côté de l'équipe électrique et mécanique. Il faudra nécessairement que cette partie soit complétée pour que nous puissions utiliser le vrai format de données en entrée dans l'algorithme de rectification.

Zones de capteurs

Notre preuve de concept actuelle fait fi du fait qu'on peut vouloir avoir plusieurs capteurs dans une serre pour avoir une rectification encore plus représentative du spectre fourni naturellement. Il y a plusieurs facteurs externes qui peuvent diminuer la quantité de lumière fournie dans une zone de la serre, mais pas dans une autre. On peut penser, par exemple, à un petit nuage qui ne passe que sur une partie de la serre, de la neige qui couvre une partie de la toile de la serre ou même des néons allumés dans une certaine partie, mais pas dans une autre. Il peut donc être bon d'avoir plusieurs capteurs pour pouvoir rectifier indépendamment plusieurs zones.

Dans la portée du projet actuel, on demandait seulement de gérer un capteur par serre. Pour gérer plusieurs capteurs, le principe de zones de capteurs doit cependant être implémenté avant d'être commercialisable. Lors de l'installation des capteurs dans la serre, on associerait à un capteur les luminaires situés à proximité de celui-ci. Pour configurer les zones de capteurs, cela nécessitera également de mettre en place un panneau de configuration destiné aux techniciens d'installation qui pourront configurer les zones de rectification et également activer ou désactiver la rectification au besoin.

Graphiques de rectification destinés aux agronomes

Pour rendre le produit commercialisable, il est important de prendre en compte qu'on doit d'une certaine façon justifier au client l'utilisation de capteurs de rectification et prouver son efficacité. Il pourra être pertinent, dans un premier temps, d'afficher tout simplement dans un graphique le spectre cible, le spectre fourni naturellement ainsi que le spectre fourni artificiellement pour montrer visuellement à quoi ressemble la rectification en temps réel. Dans un deuxième temps, on pourrait permettre d'observer un certain historique de rectification pour pouvoir analyser selon les différentes journées et différents mois quelle est la rectification requise. L'élément qui pourrait être considéré comme étant le plus important serait de calculer l'économie d'énergie par jour, par mois ou même par année engendrée par la rectification des spectres, car c'est essentiellement ce que veut savoir l'agriculteur. Évidemment, toutes ces données seraient affichées dans la plateforme infonuagique, mais, avant, on doit mettre en place la structure nécessaire pour collecter toutes ces données de façon optimisée et maintenable.

Boucle de rétroaction

L'utilisation de LEDs pour l'éclairage vient avec deux défis particuliers au niveau de la qualité spectrale. En effet, la dégradation des LEDs avec le temps ainsi que leur température ont un impact difficile à prédire sur leur spectre produit. La seule réelle manière de s'assurer de la composition spectrale finale projetée aux plantes est donc d'établir une boucle de rétroaction. Cela était hors de la portée du projet de fin d'études, mais serait idéal à ajouter dans le futur. La boucle de rétroaction fonctionnerait simplement à l'aide de capteurs positionnés au niveau de la canopée des plantes qui nous indiqueraient la vraie lumière projetée par les luminaires. Cela nous permettrait de corriger les différences entre

les spectres théoriques et les spectres réels pour assurer une qualité maximale. Bien entendu, nos algorithmes devraient être modifiés pour permettre de retirer le soleil de l'équation. Au final, cette solution pourrait aussi être intégrée dans un processus d'acquisition de données qui nous permettrait de prédire l'impact de la chaleur et de la dégradation sur la lumière et qui nous guiderait vers des améliorations substantielles aux algorithmes de création de spectres et de rectification.

Conclusion

Nous pouvons conclure que le projet a été un succès. Bien que nous n'ayons pas réussi à livrer tout ce que nous avions planifié après notre analyse initiale, notre réajustement de cap nous a permis de fournir à Sollum des changements architecturaux et algorithmiques de qualité et pour lesquels ils sont grandement satisfaits. En tant que futurs ingénieurs logiciels, nous avons dû user de plusieurs de nos compétences acquises durant notre formation afin de mener à bien le projet. Nous avons dû recueillir des exigences, analyser des architectures et des algorithmes, planifier une gestion de projet, proposer des solutions techniques au promoteur, implémenter ces solutions et bien documenter le tout.

Aussi, tout cela n'aurait pas réussi sans une collaboration étroite entre les trois membres de l'équipe ainsi qu'une communication efficace avec les équipes de Sollum. Le produit que nous avons fait évoluer est sur une bonne lancée pour devenir commercialisable dans un futur assez proche et nous sommes extrêmement contents d'avoir pu participer à un projet aussi motivant. Nous sommes très confiants par rapport au potentiel du compensateur à aider les agriculteurs à mieux gérer la lumière de leur serre et à produire des plants de qualité supérieure à moindres coûts et espérons avoir pu aider le mouvement en croissance de la souveraineté alimentaire qui fait de plus en plus partie du discours de cette année historique de 2020.

Références

BOUTHOT, Danny. *PFE-H20_005_Conception d'un système de compensation lumineuse pouvant varier en spectre et en intensité.* Document d'offre de projet de fin d'études, Sollum Technologies inc., 2019.

DIGNAN, Larry. *Top cloud providers 2019*, ZDNet, 15 août 2019, consulté le 27 janvier 2020. https://www.zdnet.com/article/top-cloud-providers-2019-aws-microsoft-azure-google-cloud-ibm-makes-hybrid-move-salesforce-dominates-saas/

LLENAS, CARRERAS. *Arbitrary spectral matching using multi-LED lighting systems*, Optical Engineering 58(3), 035105 (29 March 2019). Consulté le 12 février 2020. https://doi.org/10.1117/1.OE.58.3.035105

NATIONAL AERONAUTICS AND SPACE ADMINISTRATION (NASA). *Visible Light.* Consulté le 7 avril 2020. https://science.nasa.gov/ems/09_visiblelight

SLATER, Ben. *Apache Kafka Benchmarks for AWS, GCP and Azure*, instaclustr, 27 novembre 2018, consulté le 27 janvier 2020. https://www.instaclustr.com/apache-kafka-benchmarks-for-aws-gcp-and-azure/

SOLLUM TECHNOLOGIES inc. *Sollum Technologies – Proposition détaillée pour l'obtention d'un financement de Technologies du développement durable Canada.* Solution d'éclairage horticole intelligent optimisée pour la croissance et les économies d'énergie. 6 novembre 2019.

Annexes

ANNEXE I – Analyse initiale des besoins

Documentation

- Produire un diagramme d'architecture.
- Produire un diagramme de classe du spectrum rectifier middleware.

Middleware (Spectrum Rectifier)

- Développer une "cron job" pour acquérir les données des capteurs à un intervalle de temps donné. Commentaire: Utiliser des spectres aléatoires en attendant les vrais capteurs.
- Module de communication qui envoie les données de lumière naturelle à la plateforme infonuagique.

Middleware (Sollum Routing Unit)

- Développement d'un "endpoint" qui reçoit les valeurs d'ajustement du nuage pour le spectre des lampes d'un ou plusieurs groupes et les achemine au firmware des lampes appropriées.

Cloud functions

- Analyse comparative des librairies de calcul scientifique (différents langages) surtout axé sur l'optimisation.
- Module de calcul de compensation requise pour une recette à un moment donné en fonction des données de spectre naturel actuelles.
- Optimisation du calcul de calibration pour réduire les coûts de traitement des données. Commentaire: Prétraitement sur les données, réduction de la précision, réduction des itérations, diminution de la fréquence des mesures, multi-threading, algorithmes alternatifs.
- Écriture d'une fonction qui envoie les configurations aux SRUs appropriés en fonction des groupes de lampes.

Front-end/Cloud API

- Revue et analyse des technologies déjà utilisées chez Sollum. Commentaire: Google infonuagique, Vue.js, Vuetify, chart.js, moment.js, python 3.7, flask
- Développement d'un graphique qui affiche les dernières données solaires reçues par le Nuage (représentation graphique de la lumière naturelle).
- Développement d'un graphique qui affiche la compensation effectuée par les lampes par rapport à la cible associée à un groupe de lampe.
- Développement d'un graphique qui affiche l'historique des compensations.
- Développement d'un module d'exportation des données de production en TXT, excel, CSV et JSON.
- Développement d'une page pour configurer les paramètres du spectrum rectifier: Seuil de rectification du spectre, temps entre chaque lecture des capteurs.

Base de données

- Recherche d'un "host" pour la base de données.
- Design d'une base de données pour les historiques de spectres du soleil et de compensation.
- Implémentation de la base de données pour les historiques de spectres du soleil et de compensation.