26, rue de Vaux-de-Foletier
17041 LA ROCHELLE CEDEX 1

DSNA-DTI
1, avenue du Dr Maurice Grynfogel
31000 TOULOUSE

Rapport de stage Élève Ingénieur

2010

Sujet: Représenter le Trafic aérien de la zone de contrôle de Tahiti dans le logiciel Google Earth.

Maitre de Stage :

DELUCE Thierry
Tel : 05.62.14.54.77
Fax : 05.62.14.54.01
Mail: thierry.deluce@aviation-civile.gouv.fr

Stagiaire :

KERVIZIC Emmanuel
Tel : 06 71 56 76 10
Mail : manu@kervizic.fr

31/08/2010

Table des matières

Introduction

Etudiant en fin de 2e année d'un cusrcus ingénieur (Bac plus 4) j'ai réalisé un stage au sein de la Dgac à Toulouse. Le but de se stage a été de representer le trafic aérien de la zone de controle de Tahiti dans un logiciel banalisé qui est Google Earth. Pour arriver à ce but j'ai du comprendre le fonctionement de leur système de contrôle Tiare, en particulié Eurocat-X. Le logiciel permetant de récupérer les données dans le système et de les transferer dans Google Earth à été réalisé en Python.

1 Contexte

1.1 Sujet du stage

Représenter le trafic aérien de la Fir [1] de Tahiti dans le logiciel Google Earth. Le logiciel GoogleEarth permet de représenter un espace tridimensionnel et de placer, à l'intérieur du logiciel, des indicateurs tels que des marqueurs de position, des lignes et des polygones. Le logiciel GoogleEarth utilise des fichiers externes de type Xml [2] pour tracer ou représenter ces graphismes. Le format de ces fichiers est ouvert et publié par Google.

Il s'agit, à partir des traces fournies par le système de contrôle de Tahiti, d'afficher le trafic aérien circulant dans la Fir à des fins d'analyse, de vérification de trajectoire, de mesure de distance...

1.2 Présentation de l'environnement

1.2.1 Dsna/Dti

La Direction des Services de la Navigation Aérienne est chargée de rendre le service de navigation aérienne pour l'État français. A ce titre, la Dsna est responsable de rendre les services de circulation aérienne, d'information aéronautique et d'alerte sur le territoire national et ceux d'outre-mer (Dom, Tom , Pom). La Dsna s'appuie sur deux directions pour exécuter cette mission :
- La Direction des opérations ou Do,
- la Direction de la Technique et de l'Innovation ou Dti.

La Do est l'acteur opérationnel du contrôle aérien tandis que la Dti est chargé du volet technique. Celui-ci consiste à réaliser ou acquérir les systèmes qui participent à l'exercice du contrôle aérien. Il s'agit de systèmes informatiques permettant d'assister le contrôleur dans ses activités, de chaînes radios pour communiquer avec les aéronefs, de systèmes de traitement de l'information météorologique...

La Dti réalise également de nombreuses études pour traiter les besoins des utilisateurs et les évolutions réglementaires. La Dti réalise le déploiement et le support opérationnel des systèmes qu'elle acquiert ou réalise.

1. FIR : région d'information de vol, de l'anglais Flight Information Region) est une division de l'espace aérien permettant de faciliter le travail des organismes du contrôle aérien.
2. Xml : Extensible Markup Language («langage extensible de balisage»), est un langage informatique de balisage générique.

Enfin la DTI fait viser ses systèmes, procédures et formation par l'autorité de surveillance nationale (Direction de la Sécurité de l'Aviation Civile ou DSAC).

La DTI est structurée en domaines qui sont chacun en charge de plusieurs pôles de compétences :
- Recherche & développement, R et D
- Exigences opérationnelles des systèmes, EOS
- Gestion du trafic aérien, ATM
- Communication, navigation, surveillance, CNS
- Déploiement et Support Opérationnel, DSO

Chaque pôle couvre un ensemble de fonctions et d'expertises.

Pôle ATM/VIG :
- Le pôle « Vol et information générale » (VIG) est responsable de la maîtrise d'ouvrage systèmes de traitement des plans de vol et informations générales, à ce titre, le pôle assure le suivi industriel de leur réalisation ou de leur acquisition. Le pôle VIG est également chargé de leur maintien en condition opérationnelles lorsqu'ils sont déployés.
- Le pôle ATM/VIG est notamment responsable de la maîtrise d'ouvrage de systèmes déployés en outre-mer. L'aéroport de Tahiti (Polynésie française) a récemment été modernisé avec un système entièrement acquis auprès d'un industriel, couplé à un radar dans le cadre du projet TIARE, qui s'est terminé en 2009.

1.3 Le site de Tahiti

1.3.1 Objectif contrôle aérien

Le contrôle aérien est un ensemble de services (cf. 1.3.2 page suivante) rendus par les contrôleurs aériens aux aéronefs afin d'aider à l'exécution sûre, rapide et efficace des vols. Les services rendus sont au nombre de trois, appelés « services de la navigation aérienne », dans les buts de :
- prévenir les collisions entre les aéronefs et le sol ou les véhicules d'une part, et les collisions en vol entre aéronefs d'autre part (autrefois appelés « abordages »). Il consiste aussi à accélérer et ordonner la circulation aérienne,
- de fournir les avis et renseignements utiles à l'exécution sûre et efficace du vol : informations météorologiques, information sur l'état des moyens au sol de navigation, information sur le trafic (quand le service de contrôle n'est pas assuré dans cette zone),

– de fournir un service d'alerte pour prévenir les organismes appropriés lorsque les aéronefs ont besoin de l'aide des organismes de secours et de sauvetage, et de prêter à ces organismes le concours nécessaire.

1.3.2 Les services de la circulation aérienne

Comme nous l'avons vu plus haut, le contrôle aérien rend plusieurs services. Nous allons voir ces services plus en détail.

1.3.2.1 Le service de contrôle

Le service de contrôle est assuré dans les buts suivants :
– Prévenir les collisions entre aéronefs ou entre un aéronef et un obstacle
– Accélérer et ordonner la circulation aérienne

Le plus important reste donc la sécurité des vols. Le contrôleur s'assure que rien n'arrivera à l'aéronef pendant son vol par des causes extérieures (autre avion, obstacle), et qu'il arrivera à sa destination le plus vite possible. En outre le contrôleur est responsable de la sécurité des vols sous sa juridiction.

Les moyens qu'utilise le contrôleur pour prévenir les abordages sont la séparation (anciennement l'espacement) et l'information de trafic.
– La séparation consiste à ménager entre deux aéronefs une distance minimale, garantissant la sécurité de ces deux avions.
– L'information de trafic est une information précise sur la position d'un autre aéronef pouvant se rapprocher dangereusement. Le pilote peut ne pas voir qu'un avion se rapproche, l'information de trafic l'aide à voir, afin de permettre au pilote d'éviter l'aéronef conflictuel.

1.3.2.2 Le service d'information

Le service d'information de vol est assuré sur tout le territoire français. En espace aérien contrôlé, il est assuré par le service de contrôle. Dans les espaces aériens non contrôlés, il est assuré par un organisme UIV (dans les CRNA) et SIV (dans les approches) en vol, ou AFIS sur un aérodrome. Il consiste à délivrer aux aéronefs les renseignements et avis nécessaires à l'exécution sûre et efficace du vol. Ces renseignements peuvent être (liste non exhaustive) :
– Météorologiques : conditions météo sur un terrain, présences d'orages...
– Information sur le trafic (à ne pas confondre avec l'information de trafic) : information sur un trafic connu ou inconnu, en fonction des éléments disponibles, pouvant interférer avec un aéronef.
– État des aides à la navigation

– État des équipements sol d'un terrain
– Amendements de plan de vol
– Information sur la position, aide aux pilotes perdus
– Autres...

1.3.2.3 Le service d'alerte

Le service d'alerte est aussi vaste que naturel. Il consiste à répondre à tous les besoins des avions qui se disent en détresse, ou dont on peut penser qu'ils sont en détresse. Ce service recouvre des domaines très variés :
– Si un avion a déposé un plan de vol, et que le contrôle à l'arrivée a reçu confirmation qu'il a bien décollé, il doit surveiller que l'avion arrive bien à destination aux alentours de l'heure prévue, et lancer des recherches si ce n'est pas le cas.
– Si un avion ne répond plus à la radio et disparaît du radar, le contrôleur doit vérifier si l'aéronef a eu un problème et s'il s'est écrasé ou posé en urgence. Il déclenche alors les secours pour rechercher l'épave et secourir les occupants.
– Si un aéronef s'écrase sur la piste ou à proximité de l'aérodrome, il déclenche immédiatement les secours et coordonne leur action jusqu'à l'arrivée des renforts.
– Si un pilote signale avoir des problèmes avec son aéronef de nature à entraver le bon déroulement du vol, le contrôleur peut lui donner une priorité absolue à l'atterrissage en écartant tous les autres aéronefs.
– Si le contrôleur sait ou soupçonne qu'un aéronef est détourné, il prévient les autorités compétentes et leur apporte tout le secours nécessaire.

D'une manière générale, ce service est une autorisation légale à porter secours par tous les moyens à un pilote en difficulté. Tout être humain le ferait, mais le service d'alerte donne au contrôleur une justification légale pour retarder ou dérouter certains aéronefs afin de porter secours à un autre.

1.3.3 La zone de contrôle de Tahiti : FIR

1.3.3.1 Le transport aérien

L'île de Tahiti est desservie par l'Aéroport International Tahiti Faa'a, situé à 5km au Sud-Ouest de Papeete. Inauguré en 1961, et détenu à 57% par le Territoire de la Polynésie Française10, c'est le plus important aéroport de la Polynésie française, et le seul aéroport international du territoire. Il s'agit donc de l'unique point d'entrée pour l'immense majorité des visiteurs mais également pour les habitants des autres îles de la Polynésie française.

L'aéroport assure les liaisons avec une dizaine de destinations internationales : Los

Angeles, Paris, Auckland, Sydney, Tokyo, Rarotonga, Santiago, l'Île de Pâques, Noumea et Honolulu10. Conscient de l'importance des liaisons aériennes internationales dans le développement économique de l'île et du pays, le gouvernement a inauguré en 1998 sa propre compagnie aérienne : Air Tahiti Nui (ATN), qui dessert aujourd'hui 5 destinations à partir de Tahiti : Paris, Los Angeles, Tokyo, Auckland, Sydney.

Concernant le réseau domestique, l'aéroport dessert l'ensemble des archipels de la Polynésie. Air Tahiti est la seule compagnie à desservir régulièrement les îles polynésiennes, assurant la liaison avec une quarantaine d'îles et d'atolls. L'île de Moorea, située à 7 minutes de vol de Tahiti est desservie par Air Moorea, une filiale de la compagnie domestique d'Air Tahiti. L'aéroport de Tahiti est la plaque tournante du trafic aérien, puisque la majorité des destinations sont uniquement desservies par l'aéroport de Tahiti. La centralisation du réseau aérien accentue donc l'attraction et l'influence de Tahiti et de l'agglomération de Papeete sur le reste des îles polynésiennes.

1.3.3.2 Etendue de la FIR

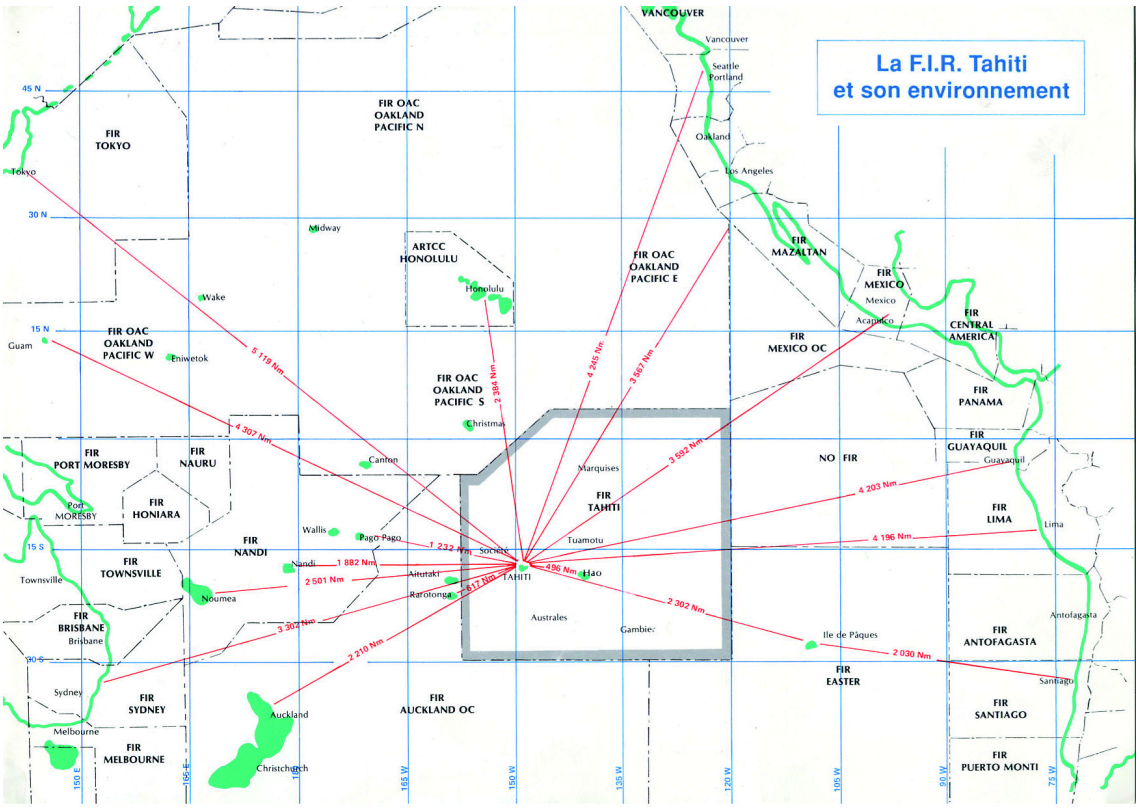

FIGURE **1.1** – FIR de Tahiti dans la région Pacifique-Sud.

La région d'information de vol de Tahiti ou « Flight Information Region » (FIR Tahiti) s'étend bien au-delà des eaux territoriales et déborde même sur l'hémisphère

nord pour atteindre le parallèle 03 ° 30' Nord, soit près de 3700 km de nord au sud et à peu près autant d'est en ouest, couvrant environ 12,5 millions de km^2.

Cette FIR constitue le volume au sein duquel la fourniture des services de la circulation aérienne sont assurés sous la responsabilité de l'administration française. Ces services comprennent :
- alerte et sauvetage
- information de vol
- contrôle de la circulation aérienne

La FIR Tahiti est fréquentée par différents types de trafic :
- les vols transpacifiques (entre la côte ouest des Etats-Unis et la Nouvelle-Zélande ou l'Australie)
- la desserte internationale de Tahiti (depuis et vers les Etats-Unis, la Nouvelle-Zélande, l'Australie, le Japon et le Chili)
- les vols intérieurs (desserte domestique des 47 aérodromes de Polynésie Française)

Plus de 40 contrôleurs aériens travaillent 24h/24 et 7j/7 dans la tour de contrôle de Tahiti-Faa'a.

Plus de 20 contrôleurs travaillent sur les aérodromes contrôlés des îles.

En 2006, le centre de contrôle a contrôlé 102 132 mouvements (+2,5%), dont 71477 mouvements IFR[3] et 30655 mouvements VFR[4].

1.3.3.3 La zone ACI :

Une fonction de contrôle spécifique, nommée ACI[5] ou zone ACI, a été développée dans le système EUROCAT-X pour répondre à des besoins de contrôle. Il s'agit d'une zone particulière limitrophe à la FIR (cf. 1.3.3.2 page précédente) de Tahiti, dons la limite se situe à 50 miles nautiques de la FIR. La zone ACI encercle la FIR. Il est à noter que cette zone n'est pas sous la responsabilité des contrôleurs aériens français, cependant, les vols pénétrant dans cette région sont visualisés par le système Eurocat-X

Ainsi en visualisant le trafic aérien dans la zone ACI, les contrôleurs peuvent maintenir les séparations entre les aéronefs. C'est-à-dire vérifier que les vols qui sont à l'extérieur et longent la FIR de Tahiti sont séparés des vols évoluant dans cette FIR.

3. IFR : (soit, en anglais, Instrument flight rules) règles de vols aux instruments
4. VFR : Visual flight rules, nom anglais de « Vol à vue »
5. ACI : Area Common Interest, soit une zone d'intérêts commun

1.3.4 Le système de contrôle : TIARE

TIARE est le nom donné au projet qui a débuté en 2007 pour s'achever fin 2010. Ce projet visait à moderniser les moyens informatiques de contrôle du centre de Tahiti, de remplacer les systèmes vieillissants de visualisation du trafic (VIVO) et de gestion de plans de vol et d'informations générales (SIGMA). La DTI a fait l'acquisition de deux systèmes différents pour couvrir l'ensemble des missions dévolue aux personnels du bureau de piste et du contrôle aérien.

Les situations de contrôle auxquelles doivent face les contrôleurs sont multiples, il y en a en effet à traiter les spécificités du contrôle océanique, du contrôle d'approche et inter-îles. Le système TIARE est construit à partir de plusieurs « produits sur étagère » :

- EUROCAT-X, système en charge du traitement radar et de la gestion plans de vols.
- ATALIS, système en charge de la préparation des vols, de la gestion des NOTAM, et de la présentation d'informations générales au contrôleur tour et approche.

Les systèmes EUROCAT-X et ATALIS sont connectés au commutateur CAGOU, raccordé aux liaisons externes (RSFTA). ATALIS reçoit également des informations météorologiques en provenance du système local d'acquisition de ces données appelé CAOBS. EUROCAT-X est raccordé au radar secondaire du mont Marau et au réseau ACARS.

La zone Aci (cf. 1.3.3.3 page précédente), a été développée spécifiquement dans le système EUROCAT-X pour répondre à des besoins de contrôle.

1.3.5 L'ADS-C

Avec l'ADS-C (Automatic Dependant Surveillance - Contract), l'avion utilise ses systèmes de navigation satellitaires ou inertiels pour automatiquement déterminer et transmettre au centre responsable sa position et d'autres informations.

Les informations transmises via l'ADS-C peuvent être :

- La position de l'avion,
- Sa route prévue,
- Sa vitesse (sol ou air),
- Des données météorologiques (direction et vitesse du vent, température...).
- Les informations de l'ADS-C sont transmises via des communications point à point, par VHF ou par satellite. Les systèmes sol et embarqués négocient les conditions suivant lesquelles ces transmissions s'effectuent (périodiques, sur événement, à la demande, ou sur urgence).

L'ADS-C est typiquement utilisé dans les zones désertiques ou océaniques où il n'y a pas de couverture radar.

Les avantages de l'ADS-C sont :
- L'utilisation pour la surveillance des zones sans couverture radar.
- La transmission de l'information "route prévue".
- La liaison de données air/sol (comme pour le Mode S et l'ADS-B).

L'inconvénient de l'ADS-C est qu'il dépend entièrement de l'avion et de la correction des données qu'il transmet.

2 Expression du besoin et gestion de projet

2.1 Expression du besoin

2.1.1 Les besoins initiaux

L'objectif initial était de pouvoir réaliser un logiciel banalisé et ergonomique permettant de représenter l'ensemble des données de contrôle (repères, balises, secteurs...) afin de pouvoir visualiser le trafic aérien circulant dans la Fir et la zone Aci. Les bénéfices attendus de cet outil sont :

- l'amélioration de l'analyse et de la compréhension visuelle du trafic aérien de Tahiti,
- la possibilité d'élaborer de statistiques à partir des fonctions de calcule du logiciel,
- une aide dans le travail de définition des points d'entrée dans la zone Aci que réalise le service de contrôle de Tahiti.

2.1.2 L'évolution des besoins

Au début du projet des besoins ont été définis. Nous verrons par la suite comment ceux-ci ont pu évoluer. Il faut noter que le client est assez dirigiste, il a déjà vu ce produit pour d'autres applications et a donc une vue global de ce qu'il souhaite en sortie. A savoir :

- Une application étant basée sur le logiciel Google Earth.
- Python comme langage de programmation.

L'objectif du choix de ces outils était aussi pour le client l'assurance de proposer à l'issue du stage une maquette complètement fonctionnelle. C'est-à-dire qu'il fallait déjouer la difficulté technique, comme la représentation du trafic sur une sphère, pour se concentrer sur les besoins suscités par cet outil. En outre la durée du stage et la part consacrée à la rédaction du rapport de stage ne permettaient pas d'innover en créant un logiciel de toute pièce.

C'est pourquoi nous avons orienté notre gestion de projet vers une méthode dite agile (cf. 2.2.2 page 14). Cette méthode nous permettra de redéfinir les besoins tout au long du projet en fonction de ce qui a déjà été réalisé. Et ainsi obtenir un produit correspondant au mieux à ce que le client aurait pu espérer.

Lors du lancement du projet les besoins étaient :

- Représenter le trafic aérien déposé par les plans de vol dans la zone de contrôle de TAHITI dans GOOGLE EARTH.
- Visualiser la configuration de la plate-forme TIARE (zones de contrôles, points remarquables ...)

Au fur et à mesure de la progression et des possibilités du logiciel, le client a affiné ses besoins et a rajouté les éléments suivants :

- Représenter le trafic aérien en fonction du temps
- Définir approximativement l'heure d'entrée de et sortie des avion dans la FIR (cf. 1.3.3.2 page 8) en fonction de leur plan de vol déposé.
- Visualiser le vol des avions en temps réel grâce aux données ADS-C[1].
- Visualiser le positionnement des avions estimé par le système TIARE entre deux reports ADS afin de visualiser l'interprétation des données reçue par le système.
- Différencier les types de vols en quatre catégories : Entrant, Sortant, Transit, Interne.

2.1.3 Les risques

Lorsque l'on a comme projet de réaliser une application qui a déjà été réalisée par le passé, nous avons une base sur la-quel se référencer (en terme de méthode, de temps, de coûts). Hors sur un projet tel que le nôtre ou même aucun prototype n'a encore été réalisé le risque que cela ne fonctionne pas est très élevé.

C'est pour cela qu'une méthode de gestion de projet, dite agile et décrite ci-dessous (cf. 2.2.2 page suivante), a été utilisée. Cette méthode nous a permis d'avancer petit à petit afin de susciter des besoins "réalisable".

Il faut aussi prendre en compte les défauts de la méthode agile qui suscite un besoin sans fin, une nécessité d'avoir un groupe restreint de personne et nécessité d'avoir un expert pour guider.

2.2 Gestion de projet

2.2.1 Choix de la méthode de gestion de projet

Comme nous l'avons vu précédemment, les besoins ne sont pas clairement définis dès le début. Il était donc difficile de pouvoir établir des spécifications explicites dans le but de mettre en place un cycle en V (cf. 2.2 page 15). Nous avons donc choisis une

1. Avec l'ADS-C (Automatic Dependant Surveillance - Contract), l'avion utilise ses systèmes de navigation satellitaires ou inertiels pour automatiquement déterminer et transmettre au centre responsable sa position et d'autres informations.

méthodologie de gestion de projet différente de celle appliquée en temps normal à la DTI.

Cette méthodologie devait nous permettre de débuter un projet sans en connaitre réellement l'aboutissement final tout en gardant de la rigueur et de l'organisation. Nous avons donc décidé d'utiliser une méthode agile[2]. Après quelques recherches et comparaisons nous nous sommes orientés sur l'extreme programming (cf. 2.2.2). Nous allons donc vous décrire cette méthodologie et la comparer avec le système utilisé habituellement.

2.2.2 L' Extreme Programming

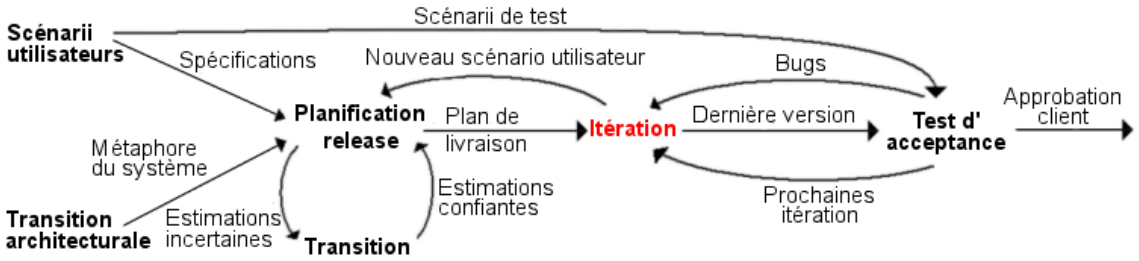

FIGURE **2.1** – Cycle de l'Exreme Programing.

L'Extreme Programming a été inventée par Kent Beck, Ward Cunningham et Ron Jeffries pendant leur travail sur un projet « C3 » de calcul des rémunérations chez Chrysler. Kent Beck, chef de projet en mars 1996 commença à affiner la méthodologie de développement utilisée sur le projet. La méthode est née officiellement en octobre 1999 avec le livre Extreme Programming Explained de Kent Beck. "Wikipedia".

Dans les méthodes traditionnelles, les besoins sont définis et souvent fixés au départ du projet, ce qui accroît les coûts ultérieurs de modifications. Extreme programming s'attache à rendre le projet plus flexible et ouvert au changement en introduisant des valeurs de base, des principes et des pratiques.

L'Extreme Programming repose sur des cycles rapides de développement (des itérations de quelques semaines voir dans notre cas quelques jours seulement) dont les étapes sont les suivantes :

– une phase d'exploration détermine les scénarios clients qui seront fournis pendant cette itération,

2. Les méthodes Agiles sont des groupes de pratiques pouvant s'appliquer à divers types de projets, mais se limitant plutôt actuellement aux projets de développement en informatique (conception de logiciel). Les méthodes Agiles se veulent plus pragmatiques que les méthodes traditionnelles. Elles impliquent au maximum le demandeur (client) et permettent une grande réactivité à ses demandes. Elles visent la satisfaction réelle du besoin du client et non les termes d'un contrat de développement.

– la transformation des scénarios en tâches à réaliser et en tests fonctionnels,
– lorsque tous les tests fonctionnels passent, le produit est livré.

Lorsqu'une tâche est terminée, les modifications sont immédiatement intégrées dans le produit complet. On évite ainsi la surcharge de travail liée à l'intégration de tous les éléments avant la livraison. Les tests facilitent grandement cette intégration : quand tous les tests passent, l'intégration est terminée.

Le cycle se répète tant que le client peut fournir des scénarios à livrer (cf. Fig. 2.1 page précédente). Généralement le cycle de la première livraison se caractérise par sa durée et le volume important de fonctionnalités embarquées. Après la première mise en production, les itérations peuvent devenir plus courtes (par exemple la séparation des plans de vol en catégories tel que : transit, interne ...)

Pour résumé, cette méthode nous amène à réaliser la boucle suivante :
– Analyse du besoin.
– Expression des spécifications
– Réalisation technique
– Test de la réalisation
– Revue logicielle (validations qui permettront de faire évoluer le produit)

2.2.3 Le cycle en V

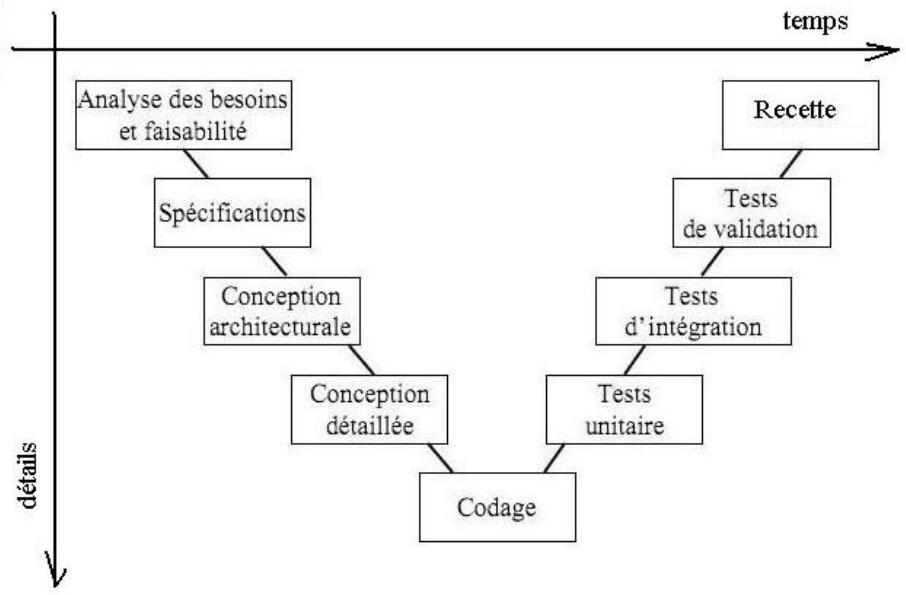

FIGURE **2.2** – Les phases à travers le temps et le niveau de détails.

Le modèle du cycle en V est un modèle conceptuel de gestion de projet étudié pour résoudre le problème de réactivité du modèle en cascade. Il permet, en cas

d'anomalie, de limiter un retour aux étapes précédentes. Les phases (cf. Fig. 2.2 page précédente) de la partie montante doivent renvoyer de l'information sur les phases en vis-à-vis lorsque des défauts sont détectés, afin d'améliorer le logiciel.

Le cycle en V est devenu un standard de l'Industrie logicielle depuis les années 1980 et depuis l'apparition de l'Ingénierie des Systèmes est devenu un standard conceptuel dans tous les domaines de l'Industrie. Le monde du logiciel ayant de fait pris un peu d'avance en termes de maturité, on trouvera dans la bibliographie courante souvent des références au monde du logiciel qui pourront s'appliquer au système.

Les étapes qui constituent cette méthode sont les suivantes :
- Analyse des besoins et faisabilité
- Spécification logicielle
- Conception architecturale
- Conception détaillée
- Codage
- Test unitaire
- Test d'intégration
- Test de validation (Recette Usine, Validation Usine - VAU)
- Recette (Vérification d'Aptitude au Bon Fonctionnement - VABF)

Dans notre cas, avec ces besoins indefinis, cette méthodologie engendrerait un risque. Ce risque serait que le logiciel final ne fonctionne pas ou ne reponde pas aux attentes du client.

2.2.4 Les avantages et inconvénients

Pour notre cas, les avantage de cette méthode sont les suivants :
- Enrichir le produit à chaque itération du cycle. Ci le logiciel est fonctionel, le client peut visualiser immédiatement les besoins qui étaient superficiels (dont il n'avait pas réellement besoin) et au contraire découvrir de nouveaux besoins.
- Rediriger rapidement la conduite du projet. Si le client souhaite rediriger son projet, ceci peut être fait dans les meilleurs délais (changement d'objectifs ou de priorités)

Cette méthode implique tout de même un certain nombre d'inconvénient tel que :
- Le client doit être disponible afin de faire avancer le projet. Chaque validation est vue avec le client et c'est celui-ci qui donne les nouveaux besoins. Ce qui implique que ci celui-ci n'est pas disponible, le projet peut vite être bloqué.
- Le projet peut vite dériver. Ce type de méthode requiert des personnes compétentes, aussi bien au niveau Maitre d'ouvrage, qu'au niveau maitre d'œuvre. Il

est facile de s'égarer c'est pourquoi une organisation et une rigueur doivent être entretenues tout au long du projet.
– Lors du début du projet, on manque cruellement de spécifications. On se lance alors dans le développement sans analyse complète.

3 Réalisation technique

3.1 Architecture du logiciel

3.1.1 L'interface Google Erath

Google Earth dispose d'une interface graphique qui sera mise à profit pour :
- représenter les points remarquables (points nommés, points de coordination, etc.),
- représenter les espaces de contrôles,
- représenter le trafic aérien.

Ces données sont, soit statiques, soit dynamiques.

Statiques : affichage de points fixes et affichage des espaces ou trajectoire plan de vol.

Dynamique : représentation du trafic aérien en fonction de coordonnées mises à jour et en fonction du temps.

3.1.2 Gestion de l'affichage

Google Earth peut être enrichi de données externes via un fichier descriptif de données (Kmz).

Ce fichier Kmz n'est autre qu'un fichier Zip compressant un fichier "doc.kml" ainsi que les fichiers vers les quel il pointe. Ce fichier "doc.kml" a pour but de regrouper les fichiers à utiliser (comme le ferait une liste de lecture pour les fichiers Mp3). Il indique donc à Google Earth de charger les fichiers suivants :

CharacteristicsPoints.kml : Le fichier contenant les points remarquables

Fir.kml : pour les zones de contrôle et la zone Aci

Routes.kml : regroupe toutes les routes définies

Fpl.kml : affiche les plans de vol déposés

Ads.kml : affiche le trafic aérien réel reçu par l'Ads-c

Les fichiers KML sont fabriqués à partir des fichiers de configuration et de traces définis dans le système Eurocat-X.

Le système EurocatX est configuré au moyen de fichiers de configuration statique. Ces fichiers seront «parsés» pour fabriquer les fichiers : "CharacteristicsPoints.kml", "Fir.kml" et "Ads.kml".

Ce système est également constitué de fichiers de traces qui enregistre les informations des vols. Ces fichiers seront «parsés» pour fabriquer les fichiers : "Ads.kml" et "Fpl.kml".

3.1.3 Les modules

Au final le logiciel se compose des modules suivants :

Manu : Fichier principal, peut être considéré comme l'exécuteur. (cf. A.1.1 page 41)

modules/Ads : Met en mémoire les information sur les report ADS. (cf. A.1.3 page 45)

modules/Aoi : Permet de définir tout les volume utilisé pour concevoir les zone de contrôles. (cf. A.1.4 page 51)

modules/CharacteristicPoints : Met en mémoire tous les points remarquables disponible sur le système. (cf. A.1.5 page 54)

modules/Convertion : Regroupe plusieurs fonctions utiliser pour convertir des donnée (ex : utilisé pour convertir les coordonnées). (cf. A.1.6 page 56)

modules/Fdp : définit et met en mémoire toutes les zone de contrôles. (cf. A.1.7 page 59)

modules/Fpl : définit et mets en mémoire les plans de vol. (cf. A.1.8 page 65)

modues/GetOfFiles : Coordonne la récupération des donnés, c'est lui qui va chercher la configuration et lance les modules tels que : Aoi, Fdp ou encore Fpl. (cf. A.1.9 page 72)

modules/KML : Ce module est utilisé pour mettre en forme les fichier KML à l'aide des données reçues en entrée (par exemple pour un point il reçoit sa description, ces coordonnées, son nom ...). (cf. A.1.10 page 74)

modules/MakeKML : C'est le module qui exploite toutes les données en mémoire et crée les fichiers KML. (cf. A.1.11 page 80)

modules/MakeKMZ : Récupère les fichiers KML pour les regrouper en un fichier KMZ plus maniable. (cf. A.1.12 page 91)

modules/Routes : définit et met en mémoire les routes. (cf. A.1.13 page 92)

modules/usualFonction : Regroupe plusieurs fonctions régulièrement utilisées. cela évite de les réécrire dans chaque module les utilisant. (cf. A.1.14 page 96)

Chaque module est décrit avec plus de précisions en annexe.

Le fichier principal (Manu.py) se lance à partir de la ligne de commande : "Python Manu.py" dans un système ou Python est installé et correctement configuré.

Les fichiers ".asf" contenant la configuration de l'eurocatx sont placés dans le répertoire "SurcesAsf" alors que les fichiers contenant les traces sont dans le répertoire "Sources".

Tout les modules sont appelés automatiquement, se qui signifie qu'après avoir renseigné le fichier de configuration et exécuté le fichier principal, aucune action n'est nécessaire pour concevoir le fichier KMZ. Il n'y aura donc plus qu'à exécuter ce fichier KMZ en l'ouvrant depuis GOOGLE EARTH.

L'exécution du programme génère aussi des fichiers de log. Ces fichiers seront utiles pour visualiser les message corrompu (mal interprété) ou encore des points non définis.

3.2 Le contexte technique opérationnel

3.2.1 EUROCATX

Il faut bien comprendre comment marche le système afin de bien visualiser d'où proviennent les informations. Comme décrit grossièrement dans le schéma (cf. Fig. 3.1), EUROCATX récupère les informations sur les plans de vol par l'intermédiaire de CAGOU [1]. Il récupère aussi le positionnement émis par l'avion à l'aide de la transmission Satellite, VHF [2] ou des données radars lors de son approche. Le système EUROCATX donne un accès à la bureautique protégé par un par-feu (FireWall) afin de rendre

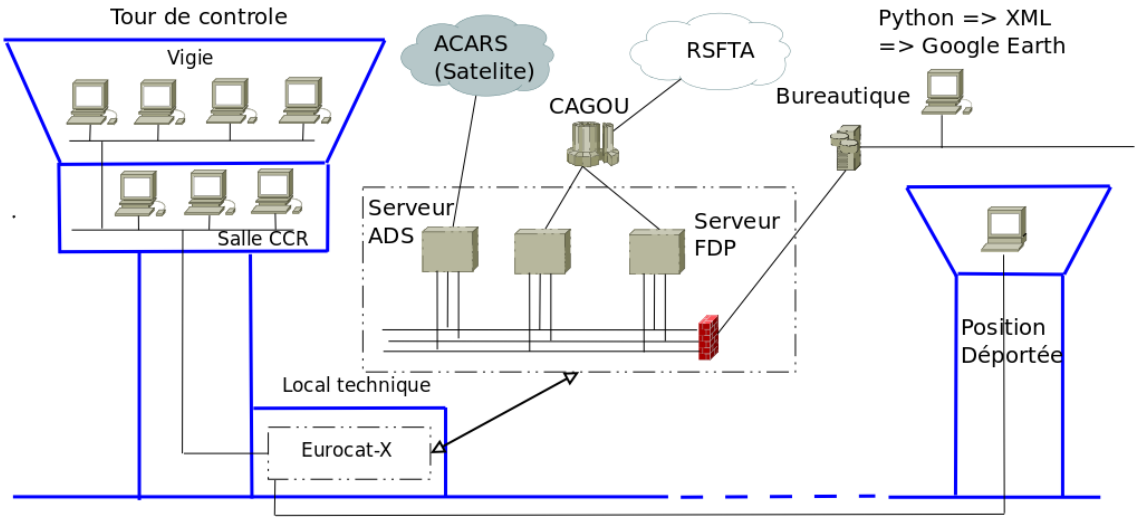

FIGURE **3.1** – Schématisation du système EUROCATX au niveau des tours de contrôle

1. CAGOU : nom donné au commutateur RSFTA
2. VHF : Very High Frequency, soit une bande radio de très haute fréquence

disponible sur ce réseau un certain nombre d'informations. Dans notre cas nous y récupérerons :

- toutes les données de configurations du système tels que les noms et coordonnées des balises référencées, la position des zones de contrôles et des zones ACI ou encore les routes utilisées pour décrire les plans de vols.
- Les fichiers de log du Commutateur CAGOU afin de pouvoir exploiter les plans de vol reçus par le réseau RSFTA.
- Tous les reports ADS reçus par satellite et traités par le système.

Le système envoie les informations récoltées et celle calculées au visues[3] situées dans la tour de contrôle au niveau de la Vigie ou de la salle CCR ainsi que de la position déportée à MORÉA.

Les données seront donc récupérées dans les fichiers ".asf" pour la configuration système, dans les fichier du FDP pour les plans de vol et dans les fichiers du serveur ADS pour les reports ainsi que pour la position calculée des aéronefs.

3.2.2 Le domaine de l'aviation

Il m'a été nécessaire de prendre connaissance de touts les termes, unité, convention, utilisés dans le domaine aéronautique.

3.2.2.1 Les coordonnées et unités :

Tout d'abord est vite venu le problème de conversion de coordonnées, J'ai donc du revoir les conversions de coordonnées sphériques ainsi que les conversions de distances. J'ai également du, comme expliqué ci dessous (cf. 3.5.2 page 28), me remémorer les solutions de calcul du point d'intersection de deux arcs de cercles en coordonnées sphériques.

3.2.2.2 Convention :

Plusieurs conventions on du être acquise comme celle utilisée par le système TIARE pour décrir les reports ADS ou encre celle utilisée par les compagnies pour le dépôt de plans de vol (cf. Bibliographie [4]).

3. Visue : Nom pour décrire les ordinateur utilisés pour visualiser les données de contrôles

3.3 Base de travail

3.3.1 Le langage Python

3.3.1.1 Bien coder :

Afin de pouvoir apprendre les bonne pratique de la programmation Python j'ai lu un livre intitulé "Programmation Python, conception et Optimisation"[6]. Celui-ci m'a permis de pouvoir d'une part revoir ce qui avait été appliqué lors de mes études et d'autre part avoir une vue global sur le langage et ainsi pouvoir prendre du recule lors du codage.

Celui ci m'a par exemple appris le nouveau style de programmation qui part du principe que chaque nouvel objet défini est basé sur un Objet existant, et que par la même occasion tout en python était Objet (même une simple variable booléenne). Ou encore la manière de vérifier si un objet était faux, égale à 0 ou encore une chaîne vide simplement en demandant si il existait (ex : `"if x != 0:"` devient `"if not x:"`)

3.3.1.2 Utiliser les expressions régulières :

L'apprentissage de l'utilisation des expressions régulières [4], m'a été grandement facilité grâce au site : `http://www.dsimb.inserm.fr/`[2] et a la documentation en ligne de Python (cf. Bibliographie [5]). Il s'est avéré après apprentissage que ces expressions régulières aurons grandement facilité la faisabilité du projet.

3.3.1.3 L'optimisation :

Je pourrais citer un passage du livre (cf. Bibliographie [6]) qui dit :

> Fourni dès le départ avec des modules de tests, Python est un langage agile. Le terme agile est originellement issu de la méthodologie de programmation agile (Beck et Al.), très proche de la programmation itérative. Cette méthodologie, qui réduit les risques liés à la conception de logiciels, introduit entre autres des principes de tests continus du code. Vincent Lozano.

En effet il m'a été rapidement nécessaire de réaliser des test, aussi bien pour vérifier que mon code était valide que pour vérifier que celui-ci s'exécutait normalement. Il s'est averé à plusieurs reprises que certaines parties de mon code étaient très gourmandes en processus. L'apprentissage de fonctions de test de code, tel que le module hotshot décrit plus tard (cf. 3.5.3 page 30), m'a été rapidement nécessaire.

4. Une expression régulière est en informatique une chaîne de caractères que l'on appelle parfois un motif et qui décrit un ensemble de chaînes de caractères possibles selon une syntaxe précise.

3.3.2 Google Earth

Google Earth est un logiciel, propriété de la société Google, permettant une visualisation de la terre en 3 dimensions avec un assemblage de photographies aériennes ou satellites. Ce logiciel donne la possibilité de configurer un environnement, ajouter des lignes, des points ou encore des polygones en 3D en passant par des fichiers de configuration au format Kml[5].

Ce format, qui repose sur le Xml[6], a l'avantage d'être simple à manipuler. Ça sémantique est définie sur le site de google (cf. Bibliographie [3])

3.4 Le programme réalisé et ses fonctions

3.4.1 Le fonctionnement

La configuration : Le programme réalisé ne possède pas encore d'interface (IHM) graphique. Il est donc nécessaire de configurer les options à l'aide d'un fichier de configuration (cf. annexe A.1.2 page 42). Nous pourrons régler par l'intermédiaire de celui-ci :

- Les fichiers Kml à recréer ou non, ce qui est utile afin de ne pas avoir à recréer des fichiers statiques (tel que la position des point caractéristique ou encore des zones de contrôles) a chaque utilisation tout en laissant a l'utilisateur la possibilité de les mettre a jour simplement.
- Les différants styles et couleurs.
- L'emplacement des fichiers de configuration.
- les descriptions et noms appliqués à chaques catégories.

L'exécution : Le fichier de configuration renseigné, le programme peut être lancé. Il est possible de le lancer par l'intermédiaire d'un Shell[7], par l'intermédiaire de l'interface Python ou encore en direct si les informations pour gérer et lancer les fichiers Python ont été renseignées dans le système d'exploitation.

Le résultat : L'exécution du programme réalise une suite d'actions :

5. Kml : Keyhole Markup Language, est un format de fichier et de grammaire Xml pour la modélisation et le stockage de caractéristiques géographiques comme les points, les lignes, les images, les polygones et les modèles pour l'affichage dans Google Earth, dans Google Maps et dans d'autres applications.
6. Xml : Extensible Markup Language («langage extensible de balisage»), est un langage informatique de balisage générique.
7. Shell : Interface en lignes de commandes

1. Lire le fichier de configuration afin de déterminer les actions à effectuer.
2. Lire les fichiers de configuration du système TIARE affin de récupérer toutes les variables nécessaires sous forme d'objets [8] (ex : points caractéristique ...)
3. Lire les fichiers de log afin de créer des objets tels que les plans de vol ou encore les reports ADS. Ces objets sont créés non seulement à partir de ses fichiers de log mais aussi à partir des objets créés précédemment (ex : les point des plan de vol désigné par un nom sont convertis en coordonnées à l'aide des points caractéristiques).
4. Créer les fichiers KML désignés dans le fichier de configuration à l'aide des objets instanciés.
5. Créer un fichier KMZ à l'aide de tous les fichiers KML afin d'avoir un fichier compact et facile a transporter.

3.4.2 L'exploitation dans GOOGLE EARTH

L'exécution du programme retourne en résultat un fichier KMZ. C'est ce fichier qui est utiliser pour exploiter les données dans GOOGLE EARTH. Pour cela il suffit d'ouvrir le fichier à l'aide de ce logiciel.

Nous allons vous présenter quelques exemples d'utilisations de ce logiciel.

3.4.2.1 Vue d'ensemble

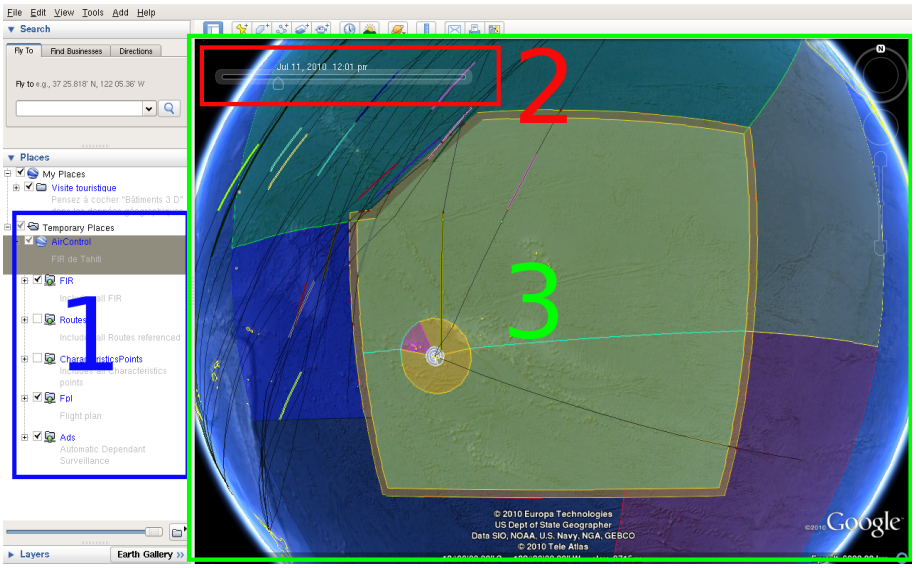

FIGURE **3.2** – Vue d'ensemble du trafic dans GOOGLE EARTH

8. Objet : structure de données valuées et cachées qui répond à un ensemble de messages. Cette structure de données définit son état tandis que l'ensemble des messages qu'il comprend décrit son comportement

Lors de l'ouverture du fichier la vue est centrée sur la zone de contrôle de Tahiti. Comme vous pouvez le voir (cf. Fig. 3.2 page précédente) nous pouvons distinguer trois zones dans le logiciel :

La zone de sélection : Cette zone, numérotée 1 sur la figure, sert à sélectionner les éléments à afficher ou non.

Chaque groupe d'éléments est représenté par un dossier. Ainsi il sera plus facile de sélectionner un groupe tel que les routes. Il sera aussi simple de dé-sélectionner un groupe (par exemple groupe Fpl qui contient tous les plans de vol) et de re-sélectionner un seul élément du groupe afin de le visualiser séparément (ex : le plan de vol d'un avion précis).

L'animation temporelle : Sur notre figure cet outil est représenté par le numéro 2. Nous pourrons grâce a lui visualiser l'évolution du trafic dans le temps. Les principales options utiles à notre cas seront :

- La sélection d'une date et une heure précise afin de visualiser où devrait se trouver un avion, ou encore avoir une vue de tous les vols en cours à cette heure.
- Un créneau compris entre deux dates et heures. Cette option nous permettra de visualiser un vol sur un partie de son parcours afin d'avoir une vue un peu plus globale. Nous l'avons utilisé lors de l'exemple suivant (cf. Fig. 3.3 page suivante) afin de mieux visualiser les points estimés par le système TIARE.

La vue : Cette zone, numérotée 3, nous permet visualisée notre sélection configurée à l'aide des deux zones citées précédemment.

3.4.2.2 Exploitation des données

Nous avons pris en exemple un zoom sur un plan de vol qui a été dévié de sa trajectoire initiale (son plan de vol). Nous pouvons donc apercevoir sur cette figure (cf. Fig. 3.3 page suivante) :

Les lignes noires Ces ligne représente les plans de vol des avions en cours a la date et l'heure sélectionnée. Ci ceux-ci ne sont pas recouverts par une ligne de couleur plus épaisse, cela signifie que l'avion n'est pas censé se trouver a cet endroit a l'instant défini.

La ligne verte : Cette ligne représente le plan de vol déposé. Chaque segment de cette ligne représente ou peut se situer l'avion a l'instant donné.

La ligne blanche : Cette ligne représente la trajectoire réellement effectuée par l'avion. Cette ligne est définie par les report ADS.

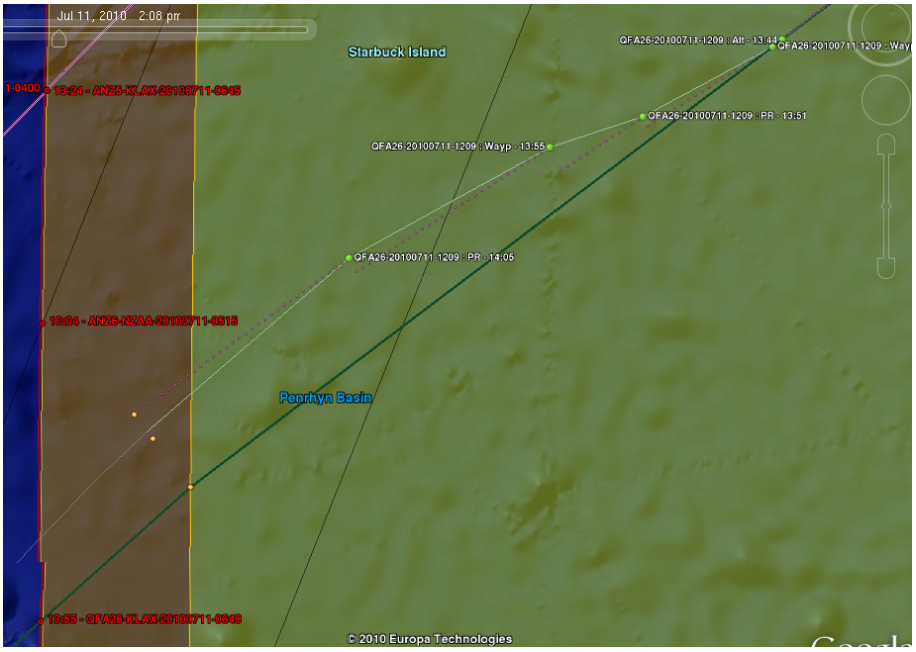

FIGURE **3.3** – Zoom su la déviation de la trajectoire d'un vol par rapport a son plan de vol déposé

Les points verts : Ces points représentent les reports ADS reçus par le système TIARE. Lorsque l'on clic sur l'un de ces points il est possible de voir sa description qui contient le message émis par l'avion.

Les points roses : Ils représentent les points estimés par le système TIARE. Dans ce cas précis nous constatons que ces points ne correspondent pas avec la trajectoire réellement réalisée. Cette différence peut s'expliquer par le fait que le système utilise les point de reports suivant estimé par l'avion pour définir la position actuel de l'avion.

Les points oranges : Ces point représentent les reports suivant estimés (next report) par l'avion cité précédemment.

Les points rouges : Ces points représentent l'heure d'entrée, estimée par le programme, dans la zone ACI (cf. 1.3.3.3 page 9).

3.5 Problèmes techniques rencontrés et solutions apportées

Comme dans tout projet il y a une multitude de problèmes à résoudre. Nous verrons dans cette partie quelques exemples de ces problèmes rencontrés ainsi que la manière dont ils ont été résolus. Cette liste reste bien entendu exhaustive au regard de tous les petits problèmes auxquels nous avons du faire face.

3.5.1 Gestion des erreurs

Problématique : Le premier problème que nous avons rencontré a été celui de la gestion des erreurs. En effet, de la première mise en route du logiciel jusqu'à la fin du stage des erreurs ont du être gérées. Deux types d'erreurs sont revenues :

- Le premier type d'erreur était par exemple une réaction in attendue du logiciel, On pourrait prendre en exemple la conversion de coordonnées reçue en Système sexagésimal[9] en coordonnées décimales utilisées dans les fichiers KML 5 page 23, qui lors des premiers tests donnaient des donnés erronées.
- Le deuxième type était celui du aux erreurs contenues dans les fichiers de log utilisés pour récupérer les informations. Ces erreur faisaient effet boule de neige et venait se répercuter dans le fonctionnement du logiciel.

Résolution : La solution au premier problème a été de mettre en place des tests à chaque fonction implémentée ou après avoir réalisé chaque objectif fixé. On appelle cette méthode le test continu du code. Grâce à cela nous allons pouvoir déterminer plus rapidement lors d'une erreur futur d'où provient celle-ci. Une méthode simple de la mette en place est de définir un test a réaliser pour valider la fonction ou le code. On détermine donc quelle réaction doit avoir un fonction pour un environnement donné et l'on vérifie si le résultat correspond bien avec celui espéré. (Ex : on a la coordonnée 4530N10045E qui correspond a 45°30' Nord 100°45' Est. On envoie cette variable dans la fonction de conversion et l'on vérifie que le résultat retourné est bien en décimal : 45,5° en latitude et -100,75 en longitude). Si le résultat est correct la fonction ou le morceau de code est validé, sinon il doit être corrigé.

La solution du deuxième problème a été dans un premier temps d'afficher chaque erreur dans la console, mais cela est vite devenu trop compliqué du fait que la console ne retient par défaut qu'un nombre limité de lignes en mémoire et donc que les lignes trop anciennes sont simplement effacées. On a donc mis en place un système de log permettant, en plus d'avoir accès aux informations les plus anciennes, de pouvoir l'exploiter après avoir fermé la console, effectuer des recherches à l'intérieur et tous avantages que peut apporter un fichier texte. Pour les dernières versions de log, celles-ci sont crées avec des informations relatives aux types d'erreurs et l'emplacement de l'erreur dans le fichier source. Le tout enregistrées dans un fichier comprenant la date et l'heure actuel dans le nom afin de pouvoir les différencier de chaque exécution du logiciel.

9. (Système sexagésimal : Degrés (°) Minutes (') Secondes ("))

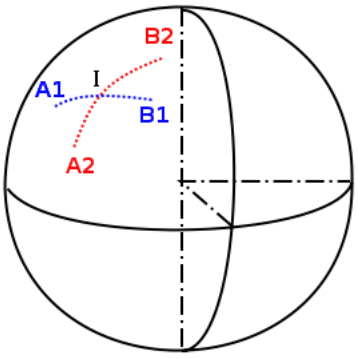

FIGURE **3.4** – Représentation grossière de l'intersection de deux arc de cercle respectivement formé par la trajectoire la plus courte entre deux points situé sur le Globe terrestre

3.5.2 Intersection entre plans de vol et zone ACI

Problématique : Afin de déterminer l'heure d'entrée approximative des avions dans la zone ACI (cf. 1.3.3.3 page 9) en fonction de leur plan de vol déposé Il est nécessaire de déterminer le point d'intersection entre leur plan de vol et la zone ACI. En théorie cela paraît simple, il suffit de prendre chaque portion du trajet du plan de vol composé de deux points et formant une droite, et de déterminer si cette droite coupe chaque droite composant la zone ACI. Dans la pratique il c'est averé que cela était un peu plus compliqué, en effet ces droites sont en réalité des arcs de cercles qui sont composés de deux extrémités définies par des points en coordonnées sphériques (cf. schéma fig. 3.4).

Résolution : Étant donné que j'ai effectué un BTS avant d'intégrer l'EIGSI[10], les notions de coordonnées sphériques ne me sont que peu familières. Après avoir en vainc cherché sur internet ainsi que dans mon entourage (maître de stage, collègues de travail) je me suis replié sur un forum de mathématique sur le quel j'ai déposé un sujet explicitant le problème (adresse, cf. bibliographie [1]). Un utilisateur nous a donné une solution qui, après connaissance, semble tellement simple qu'on se demande pourquoi personne n'y a pensé. Cette solution consiste a déterminer les plans définis par les deux points aux extrémités de chaque arc et par le centre de la terre (ainsi nous avons forcement la courbe qu'a suivi l'avion sur ce plan). Il faut ensuite déterminer la normal à chacun des plans pour en déduire la droite d'intersection de ces plans (passant par le centre de la sphère). Une foi cette droite acquise il faut définir son vecteur norme et le convertir en coordonnées sphériques. Ce qui nous donne un des point d'intersection de la droite avec la sphère, l'autre étant situé par définition à l'opposé.

10. EIGSI : École d'Ingénieurs en Génie des Systèmes Industriel située à La Rochelle

Une démonstration valant amplement un long discours, et a titre informatif, voici ce que cela donne en résolution mathématique. Pour cet exemple nous avons deux arcs représentent 2 trajectoires définies chacune par 2 points A et B (cf. Fig. 3.4 page précédente). Chaque pour sera défini par une latitude et une longitude.

Nous avons donc :
- lat_A la latitude de A
- $long_A$ la longitude de A
- (x_A, y_A, z_A) les coordonnées cartésiennes de A
- I_1 le point d'intersection n° 1
- I_2 le point d'intersection n° 2

Il faut tout d'abord convertir les coordonnées sphérique en vecteur de coordonnées cartésiennes pour A et B :

$$A = \begin{cases} x_A &= cos(lat) \times cos(long) \\ y_D &= cos(lat_A) \times sin(long_A) \\ z_A &= sin(lat_A) \end{cases}$$

Il faut ensuite déterminer le plan passant par O, A et B ayant alors pour équation :

$$ax + by + cz = 0$$

où

$$\begin{pmatrix} a \\ b \\ c \end{pmatrix} = \begin{pmatrix} x_A \\ y_A \\ z_A \end{pmatrix} \wedge \begin{pmatrix} x_B \\ y_B \\ z_B \end{pmatrix}$$

c'est à dire

$$\begin{cases} a = y_A z_B - z_A y_B \\ b = z_A x_B - x_A z_B \\ c = x_A y_B - y_A x_B \end{cases}$$

L'intersection des deux plans de coordonnées (a, b, c) et (a',b',c') contient le point O, mais aussi le point P de coordonnées (x_P, y_P, z_P) tel que :

$$\begin{pmatrix} x_P \\ y_P \\ z_P \end{pmatrix} = \begin{pmatrix} a \\ b \\ c \end{pmatrix} \wedge \begin{pmatrix} a' \\ b' \\ c' \end{pmatrix}$$

P n'étant pas forcément sur la sphère, il faut trouver un point de la droite (OP) sur cette sphère. Pour cela il suffit de diviser les 3 coordonnées de P par la norme de

$\overrightarrow{OP}$:

$$I_1 = \begin{cases} x_P/\sqrt{x_P^2 + y_P^2 + z_P^2} \\ x_P/\sqrt{y_P^2 + y_P^2 + z_P^2} \\ x_P/\sqrt{z_P^2 + y_P^2 + z_P^2} \end{cases}$$

nous avons donc I_1 et son opposé I_2, il nous reste donc plus qu'a vérifier si chacun de ces points appartient à un des 2 arcs.

Vous trouverez le code Python correspondant à ces calculs dans la fonction : "verifyIntersection (line, point) :" du module "ususalFonction.py" disponible en annexe A.1.14 page 96

3.5.3 Performance du logiciel

Problématique : Les premiers tests du logiciel ce sont déroulés sur un nombre limité de fichiers (représenté par un nombre limité d'heure de vol), ce affin de pouvoir les valider rapidement. Lors de l'apparition de fichiers plus volumineux (plus de 300Mo de donnés en entrée, environ 10% en sortie) c'est posé le problème de performance. Avant optimisation l'ordinateur moulinais des heures avant de pouvoir sortir un fichier. Il a donc fallût optimiser le code afin d'alléger le programme en ressources.

Résolution : En cherchant des conseils dans des forum d'informatique ainsi que dans le livre cité précédemment (cf. bibliographie [6], nous avons découvert que Python était un langage orienté par les tests et qu'il disposait donc de librairies spécialement conçues pour déterminer les points bloquants d'un programme et les fonctions appelées les plus gourmandes.

La fonction retenue pour repérer ce qui est appelé en anglais les Bottleneck[11] est la fonction "hotshot" qui à pour but d'analyser un programme dans sa totalité en indiquant notamment les ressources utilisées par chaque fonction appelée. Pour visualiser ce que donne le résultat d'une analyse veuillez vous reporter a la figure 3.5 page suivante.

Les bottlenecks repérés, une réécriture des parties bloquantes à du être effectuée. Cette analyse nous a permis de réduire les ressources et donc le temps d'exécution du logiciel de plus de 80%.

11. Bottlneck : (goulot d'étranglement) point d'un système limitant les performances globales, et pouvant avoir un effet sur les temps de traitement et de réponse.

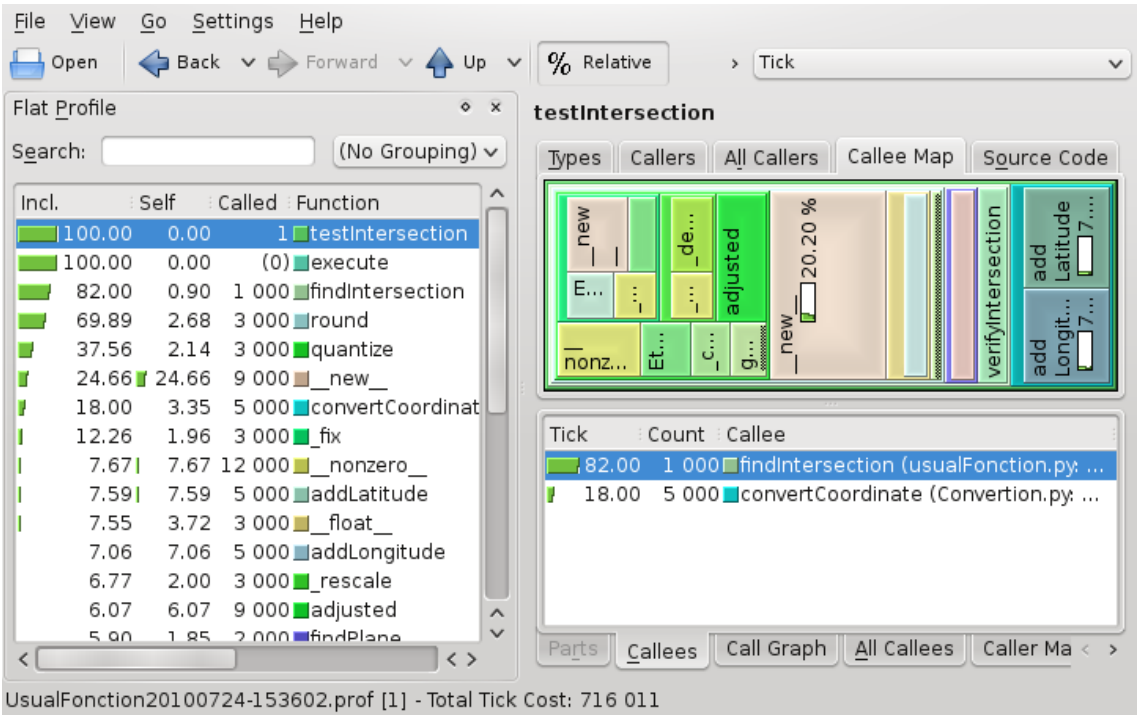

FIGURE **3.5** – Analyse avec Kcachegrind de l'exécution de la fonction "TestIntersection" du programme dans le but de l'améliorer.

4 Tests et validation de la réalisation

4.1 Les tests

Nous avons effectué au cours de ce projet deux types de tests : les tests unitaires et les tests globaux.

4.1.1 Les tests unitaires

En programmation informatique, le test unitaire est un procédé permettant de s'assurer du fonctionnement correct d'une partie déterminée d'un logiciel ou d'une portion d'un programme (appelée «unité» ou «module»).

On écrit un test pour confronter une réalisation à sa spécification. Le test définit un critère d'arrêt (état ou sorties à l'issue de l'exécution) et permet de statuer sur le succès ou sur l'échec d'une vérification. Grâce à la spécification, on est en mesure de faire correspondre un état d'entrée donné à un résultat ou à une sortie. Le test permet de vérifier que la relation d'entrée - sortie donnée par la spécification est bel et bien réalisée.

Rappel de définitions :

Test : il s'agit d'une vérification par exécution.

Vérification : ce terme est utilisé dans le sens de contrôle d'une partie du logiciel. (Une «unité» peut ici être vue comme «le plus petit élément de spécification à vérifier»)

Il s'agit pour nous de tester un module, indépendamment du reste du programme, ceci afin de s'assurer qu'il répond aux spécifications fonctionnelles et qu'il fonctionne correctement en toutes circonstances.

Mais ces tests ne sufisent pas car il ne donnent pas assez de recule pour visualiser si l'ensemble du programme est fonctionnel. Il nous donne seulement une confirmation théorique.

4.1.2 Les tests globaux

Comme nous l'avons cité précédement des tests unitaires ne sont pas sufisant dans notre cas. En effet ce projet étant un réél prototype, dans le sens ou rien n'a été effectué de semblable auparavant, les spécifications restent parfois mal déterminées. On pourra citer comme exemple la structure des messages Fpl qui sont cencés avoir toujours la meme forme, mais qui se retrouves souvant avec des erreurs du à la qualité de transmission.

Ces tests aurons donc pour but de valider le fait que toutes les parties développées indépendamment fonctionnent bien ensemble de façon cohérente.

Pour ce faire nous avons passé un grand nombre de fichiers sources à «parsé». Ce qui nous a permis de découvrir tout au long du projet un certain nombre d'erreurs. Nous pourons citer en exemple le fait de prendre en compte le nom de l'avion, aéroport de départ et heure de depart comme identifiant, celui-ci pouvant être le meme sur plusieurs jours pour les vols cylciques. Dans ce cas les tests globaux nous ont permis de découvrir l'errur et nous ont permis d'ajouter le jour de depart de l'avion dans l'identifiant afin que chaque identifiant reste bien unique.

4.2 La validation

Chaque fin de cycle de notre méthode de gestion de projet qui est l'Extreme Programing nous amene à une étape de validation. Celle ci conciste à verifier avec le client que le programme se comporte bien comme il le souhaitait.

Après chaque validation des spécification sont modifiées car, bien que répondant à leur definition, elle ne réponde pas réelement aux attentes du client. D'autres spécifications sont crées et certaines annulées.

4.3 Amélioration continue

A partir des tests réalisés et de la validation avec le client comme cité précedement nous redéfinissons les besoins ainsi que les spécifications. Cela nous amène donc à revenir au cycle des besoins (cf. 2.1 page 12) dans notre méthode de gestion de projet (cf. 2.2.2 page 14).

Nous reprenons alors un cycle ce qui nous permettra d'améliorer le programme en continu.

5 Synthèse

5.1 Spécifications obtenues

Comme vous l'avez compris, utiliser une méthode agile comme l'extreme programming (cf. 2.2.2 page 14) implique une perpétuelle réécriture des spécifications. Celles-ci sont améliorées, réécrites, ajoutées tout au long du projet. C'est pourquoi dans ce rapport sera cité la dernière version des spécifications.

Les spécifications du logiciel sont les suivantes :

Capture de fichiers de configuration : Les points caractéristiques, route, zone de contrôle et Aci, doivent être récupérés dans les fichiers de configuration du système Tiare afin d'avoir la représentation la plus juste de ce que le système a. Ils doivent être gardés en mémoire pendant toute l'exécution du logiciel afin de pouvoir être utilisés. Les données seront enregistrées dans des objets le temps de l'exécution du programme afin de faciliter leur exploitation. La configuration du logiciel doit laisser à l'utilisateur la possibilité de spécifier le chemin du fichier de configuration.

Capture des données Plan de vol : Les données Plan de vol doivent être récupérées dans les log du système Tiare. Par contre il doit être possible de les récupérer d'un autre fichier contenant des trames Fpl au format normalisé par la norme 4444 (cf. Bibliographie [4]). Chaque plan de vol serra enregistré dans un objet ayant un identifiant comprenant : L'identifiant de l'avion, son aéroport de départ ainsi que l'heure et le jour de départ. Cet identifiant a pour but de les différentier et de les référencer dans le temps. La configuration du logiciel doit laisser à l'utilisateur la possibilité de spécifier le chemin du fichier de log.

Capture des données Ads : Les données Ads doivent être récupérées dans les log du système Tiare. Il devra être aussi récupéré dans ces log les points de la position en fonction du temps calculé par le système entre deux reports Ads. Les reports Ads et points calculés seront instanciés par avion et par vol. L'identifiant de chaque vol sera donc composé de l'identifiant de l'avion ainsi que de la date et l'heure du message de login. La configuration du logiciel doit laisser à l'utilisateur la possibilité de spécifier le chemin du fichier de log.

Les points caractéristiques : Ces points devront être implémentés dans Google Earth avec la possibilité de les afficher ou non. La configuration du logiciel doit laisser à l'utilisateur la possibilité de spécifier la possibilité de rééditer ou non le fichier source Google Earth. Ces points seront représentés par un triangle de petite taille.

Les zones de contrôle et ACI : Les zones de contrôle et zones ACI devront être implémentés dans GOOGLE EARTH avec la possibilité de les afficher ou non. La configuration du logiciel doit laisser à l'utilisateur la possibilité de spécifier la possibilité de rééditer ou non le fichier source GOOGLE EARTH. Ces zones seront représentées par une surface colorée en 2 dimensions.

Les routes : Les routes devront être implémentées dans GOOGLE EARTH avec la possibilité de les afficher ou non. La configuration du logiciel doit laisser à l'utilisateur la possibilité de spécifier la possibilité de rééditer ou non le fichier source GOOGLE EARTH. Ces routes seront représentées par une ligne de couleur Jaune. Les points définissant cette route ne seront pas illustrés afin de ne pas faire de doublons avec les points caractéristiques. Les coordonnées des points de chaque route devront être définis à partir des points caractéristiques en mémoire.

Les plan de vol : Les plans de vol devront être implémentés dans GOOGLE EARTH avec la possibilité de les afficher ou non. La configuration du logiciel doit laisser à l'utilisateur la possibilité de spécifier la possibilité de rééditer ou non le fichier source GOOGLE EARTH. Les plans de vol doivent pouvoir être visualisés dans GOOGLE EARTH en fonction du temps. Pour se faire une heure théorique de passage sera calculée par le programme pour chaque point définissant le plan de vol. Toutes les informations concernant chaque plan de vol tel que ça route, les points constituant sa route et sa situation dans le temps devront être regroupés dans un dossier. Le message FPL de l'avion doit être visible dans la description de ce dossier. Les plans de vol seront visibles durant toute la durée du vol et représentés par une ligne noire. La visualisation dans le temps sera représentée par un segment de couleur choisie aléatoirement pour chaque vol défini par les deux points les plus proches de l'heure en paramètre dans le logiciel (un point avant et un point après). Ce segment et ses points ne seront visible qu'à partir de l'heure du premier point jusqu'à l'heure du deuxième.

l'intersection du plan de vol avec la zone ACI : L'intersection, si elle a lieu, entre le plan de vol et la zone ACI doit être calculée, définie et représentée dans GOOGLE EARTH par un point rouge accompagné du nom de l'avion et de l'heure d'intersection affichée en rouge également. Ces points devront être contenus dans le dossier du concerné.

Les reports ADS : Chaque report ADS sera composé de ces points de reports ainsi que des points calculés par le système. Chaque report sera regroupé dans un dossier par vol et aura comme description le message reçu. Chaque point calculé par le système sera attribué et regroupé avec le report précèdent. Les vols seront représentés par une ligne blanche retraçant tout les reports reçus, ainsi que chaque point affiché dans le temps. L'intérêt étant de visualisé l'écart

entre le chemin parcouru par l'avion et le plan de vol déposé ainsi que la différence entre la trajectoire de l'avion et celle calculée par le système TIARE.

5.2 La gestion de projet

L'utilisation de l'extreme programming pour gérer le projet aura été réellement bénéfique. On notera tout de même que cette méthode requière des clients extrêmement compétents et réactifs. En effet sans compétence de la part du client le projet peu rapidement tourner en rond.

Le fait de renouveler sans cesse les besoins et spécifications permet de réaliser un produit riche, adapté et performant. L'évolution quant à elle demande une maîtrise bien plus stricte qu'avec une méthode de gestion de projet plus classique sous peine de devenir rapidement désordonnée.

Nous avons utilisé la méthode agile pour préciser les besoins et obtenir une spécification applicable à la concrétisation d'un vrai logiciel. Ainsi nous pouvons aussi dire que nous sommes restés au niveau 1 d'un cycle en V

5.3 Le projet

Nous avons atteint un grand nombre d'objectif avec ce projet, il nous est capable d'analyser des plans de vol et de les mettre en corrélation avec les reports reçu de l'avion par l'intermédiaire de liaison satellite.

Le projet n'est pas fini. Un grand nombre d'améliorations restent à implémenter tels qu'une interface graphique ou encore une base de donnée (cf. 6 page 38).

5.4 Un stage formateur

5.4.1 Un apport technique

J'ai pu au cours de ce stage approfondir et mettre en pratique mes connaissances en programmation python. Mais j'ai surtout pu découvrir le monde de l'aéronautique.

En effet ce stage ma permis de voir les technologies utilisées dans les zones de contrôles. J'ai donc pu prendre connaissance des technologies de détection des avions de dernière génération (des fois pas encore mis en place) tels que les radars secondaires Mode S. Ou encore les systèmes de détections de collisions. J'ai également pu m'instruire sur leurs solutions de télécommunications (Satellite, VHF) qui ne

sont pas enseignées sous cet angle (pratique et non théorique) dans le cadre de mon cursus.

J'ai aussi pu découvrir le fonctionnement d'un serveur NTP, les principe de la para-virtualisation ou encore la mise en place de réseaux privé virtuel. Autant de domaines n'étant pas en relation direct avec le stage mais qui auront une grande utilité dans mon avenir professionnel.

Un autre point découvert dans la pratique aura été des méthodes de gestion de projet. La première étant le cycle en V du fait que ce soit celle qui est appliqué au sein de l'entreprise. La deuxième étant l'extreme programing utilisée pour le projet du stage.

L'ouverture d'esprit du personnel faisant partie ou travaillant en sous-traitance pour la DGAC y a fortement contribué.

5.4.2 Des rapport humains

Ce stage m'a également permis, notamment lors des pauses cafés ou repas du midi, d'échanger avec un grand nombre d'ingénieurs travaillant pour la DGAC ou en sous-traitance.

Ce sont grâce à ces échanges que j'ai pu acquérir une grande partie des connaissances en aéronautique et en informatique citées précédemment. En effet ces personnes n'ont pas héritées à prendre un peu de leur temps pour me faire des schémas sur le fonctionnement des différentes technologies de radars ou encore retrouver leur rapport de test (benchmark) réalisé sur différentes mise en place de virtualisations d'OS (Xen, Kvm, VMWere).

5.5 Conclusion

Ce stage aura été une expérience professionnel très enrichissante. Il m'aura permis de découvrir le monde de l'aéronautique. Il m'aura aussi permis de découvrir plusieurs méthodes de gestion de projet ainsi que des connaissances techniques variées.

Au delà des aspects pédagogiques techniques, il m'aura aussi permis de me familiariser avec le monde de l'entreprise du point de vue d'un ingénieur.

Le contexte m'aura aussi sensibilisé sur la qualité. En effet dans l'aéronautique la gestion de la qualité est des plus importante du fait qu'un système ne peut pas tomber en panne sous peine d'avoir de grave conséquences.

6 Evolution projet

Ce projet est loin d'être arrivé a termes. Nous allons donc voir ici ce qui pourrait être fait afin de perfectionner ce logiciel. Les évolutions seront axées sur trois points :
- La mise en place d'une interface graphique.
- L'automatisation de l'acquisition.
- La pérennisation des données.

6.1 La mise en place d'une interface graphique

Comme il a été expliqué précédemment (cf. 3.4.1 page 23), la configuration du logiciel est effectuée manuellement par l'intermédiaire de fichiers textes et son exécution est effectuée en ligne de commande. C'est pourquoi une interface graphique faciliterait grandement son utilisation.

Cette interface devrait pouvoir faciliter la configuration et l'exécution du programme, elle pourrait être basée sur des technologie web afin de la rendre portable tout en séparant le traitement des donnés de l'utilisation du fichier final dans Google Earth. En effet le programme pourrait être lancé a distance sur une machine, cela permettrait de sécuriser l'accès au donnés tout en libérant les ressources du poste de l'utilisateur.

Pour faciliter la configuration un histogramme avec tous les vols figurant entre deux dates sélectionnées pourrait être réalisé, cela permettrait de mieux visualiser le trafic et de pouvoir cibler les vols à afficher.

Il pourrait aussi être intéressant d'inclure l'affichage final dans l'interface web, tout en laissant la possibilité à l'utilisateur de télécharger le fichier afin d'exploiter pleinement toutes les fonctionnalités du logiciel Google Earth tel que la mesure de distance entre deux points.

6.2 L'automatisation de l'acquisition

Actuellement chaque fichier à traiter est récupéré manuellement. On pourrait concevoir un système qui irait de lui même chercher les fichiers nécessaires dans le système Tiare et les mettre automatiquement à la disposition du programme.

6.3 La pérennisation des donnés

Dans une optique de pouvoir rejouer simplement des situations passées, on pourrait mettre en place un système de base de donnés légère tel que SQLite[1]. Contrairement aux serveurs de bases de donnés traditionnels, comme MySQL ou PostgreSQL, sa particularité est de ne pas reproduire le schéma habituel client-serveur mais d'être directement intégré aux programmes. L'intégralité de la base de donnés (déclarations, tables, index et donnés) est stockée dans un fichier indépendant de la plate-forme.

Ce procédé couplé à un traitement automatique permettrait de mettre et garder en mémoire tous les vols disponibles sur le système TIARE. Il permettrait donc de pouvoir rejouer des situations qui ont été enregistrées plusieurs mois avant.

1. SQLite est une bibliothèque écrite en C qui propose un moteur de base de donnés relationnelles accessible par le langage SQL.

Bibliographie

[1] KERVIZIC Emmanuel and internaute. Titre du thread. http://maths-forum.com/showthread.php?p=692707#post692707, june 2010.

[2] Patrick Fuchs and Pierre Poulain. Expressions régulières et parsing. http://www.dsimb.inserm.fr/~fuchs/python/python-node13.html, june 2010.

[3] Google. Documentation en ligne sur la sémentique des documents kml. http://code.google.com/apis/kml/documentation/kmlreference.html, june 2010.

[4] OACI. *Doc 4444. Règles de l'air et services de la circulation aérienne.* 2010.

[5] Python v2.7. Documentation en ligne de python. http://docs.python.org/, june 2010.

[6] Tareck Ziadé. *Programmation Python, Conception et optimisation.* Eyrolles, 2009.

A Annexes

A.1 Codes sources du projet

A.1.1 Manu

Ce fichier sert à executer tout le programme :

```
#coding: utf-8
__author__ = "KERVIZIC Emmanuel (kervizic@hotmail.com)"
__version__ = "0.0.1"
__license__ = ""
__copyright__ =""

import sys
sys.path.append(r'modules')
import CharacteristicPoints, Routes, KML, Fdp, GetOfFiles, MakeKML, Ads
import MakeKMZ, Aoi, os

def main():
    print ("""
###############################################################################
## Representation des traces et routes des avions dans Google Earth ##
#    Programme realise par: KERVIZIC Emmanuel                                 #
#       Pour :   La DTI de l'aviation civile                                  #
##      Le :     17/06/2010                                                  ##
###############################################################################
\nC'est un bon debut ;)\n""")

    allObjects = GetOfFiles.getOfFile()
    makeFile = allObjects['makeFile']
    if makeFile['characteristicsPoints'] == 'yes':
        kmlCP = MakeKML.addAllCharacteristicsPoints(allObjects)
    if makeFile['routes'] == 'yes':
        kmlRT = MakeKML.addRoutes(allObjects)
    if makeFile['fir'] == 'yes':
        kmlST = MakeKML.addSector(allObjects)
    if makeFile['fpl'] == 'yes':
        kmlFP = MakeKML.addFpl(allObjects)
    if makeFile['ads'] == 'yes':
        kmlFP = MakeKML.addAds(allObjects)
    if makeFile['main'] == 'yes':
        kmlMain = MakeKML.addMain(allObjects)

    kmz = MakeKMZ.makeFile(allObjects)
    print "Et ca fini bien !"

# Execute only if this file is main
if __name__ == '__main__':
    main()
```

/home/manu/DTI/Manu.py

A.1.2 Config

Nous avons ici le fichier de configuration. Celui ci sert notament à se passer temporairement d'une interface graphique.

```
################################################################################
##                                                                            ##
#    File of configuration                                                    #
#     Here is complete all the fields needed to run the program               #
#                                                                             #
#    WARNING !!!                                                              #
#     Do not edit this file without considering the consequences              #
##                                                                            ##
################################################################################
// option
# enable the deletion of the area during the approach : yes or no
deleteZone                          = no


// update or crate KML File
# if you want to update the KML file, set to: yes, the variable
# Otherwise, set it to no

# All characteristics points, very long to write
characteristicsPoints               = yes
#characteristicsPoints               = no
# All routes
routes                              = yes
#routes                              = no
# All fir
#fir                                 = yes
fir                                 = no
# All Flight Plan and ADS position
fpl                                 = yes
#fpl                                 = no
# All ADS
ads                                 = yes
#ads                                 = no
# The main KML File
main                                = yes
#main                                = no



// path of files
# warning, all directories used must be created

## WINDOWS
#mainPath                            = D:\Documents de Manu\DTI\
#characteristicsPointsFilePath       = .\Sources\CHARACTERISTIC_POINTS.ASF
#routesFilePath                      = .\Sources\ROUTES.ASF
#fdpFilePath                         = .\Sources\FDP_VOLUMES_DEFINITION.ASF
#aoiFilePath                         = .\Sources\AOI_VOLUMES.ASF
#fplFilePath                         = .\Sources\FDX
#absRoutesPath                       = .\AC\KML\Routes.kml
#absCharacteristicsPointsPath        = .\AC\KML\CharacteristicsPoints.kml
#absFirPath                          = .\AC\KML\Fir.kml
#absFplPath                          = .\AC\KML\Fpl.kml
#absMainPath                         = .\AC\doc.kml
#absKmlPath                          = .\AC
#absKmzPath                          = AirControl.kmz

# OTHER
mainPath                            = /media/manu/Documents de Manu/DTI/
characteristicsPointsFilePath       = SourcesAsf/CHARACTERISTIC_POINTS.ASF
routesFilePath                      = SourcesAsf/ROUTES.ASF
fdpFilePath                         = SourcesAsf/FDP_VOLUMES_DEFINITION.ASF
aoiFilePath                         = SourcesAsf/AOI_VOLUMES.ASF
fplFilePath                         = Sources/
adsFilePath                         = Sources/
```

```
absRoutesPath                   = AC/KML/Routes.kml
absCharacteristicsPointsPath    = AC/KML/CharacteristicsPoints.kml
absFirPath                      = AC/KML/Fir.kml
absFplPath                      = AC/KML/Fpl.kml
absAdsPath                      = AC/KML/Ads.kml
absMainPath                     = AC/doc.kml
absKmlPath                      = AC
absKmzPath                      = AirControl.kmz
# used in KMZ archive
kmlFilePath                     = files/
kmlRoutesPath                   = KML/Routes.kml
kmlCharacteristicsPointsPath    = KML/CharacteristicsPoints.kml
kmlFirPath                      = KML/Fir.kml
kmlFplPath                      = KML/Fpl.kml
kmlAdsPath                      = KML/Ads.kml


// Object for point KML Style
circle                          = placemark_circle.png - 0.2 - FFFFFFFF
redCircle                       = placemark_circle_highlight.png - 0.4 - FF0000FF
polygon                         = polygon.png - 0.2 - FF00FFFF
triangle                        = triangle.png - 0.2 - FF00FFFF
airports                        = airports.png - 0.2 - FFFF6633
avion                           = avion.png - 0.6 - FF888888
avion2                          = avion2.png - 0.4 - FFFFFFFF
avion3                          = avion2.png - 0.2 - FFFFFFFF
avion4                          = avion.png - 0.3 - FFFFFFFF
redRound                        = redround.png - 0.4 - FF0000FF
greenRound                      = greenround.png - 0.3 - FFFFFFFF
blueRound                       = blueround.png - 0.3 - FFFFFFFF
glossyRound                     = glossyround.png - 0.1 - FFFFFFFF
orangeRound                     = orangeround.png - 0.2 - FFFFFFFF
yellowRound                     = yellowround.png - 0.3 - FFFFFFFF
blackRound                      = blackround.png - 0.2 - FF888888

// General LookAt
longitude                       = -138.5
latitude                        = -13.0
altitude                        = 0
range                           = 6000000
tilt                            = 0
heading                         = 0

// list of Color
# FIR EXT
NZZO                            = 4D000000
SCIZ                            = 4D0000FF
KZAK                            = 4D00FF00
SCTZ                            = 4D00FFFF
NCRG                            = 4DFF0000
# FIR NTTT
VMOR                            = 4DFF00FF
VIWR                            = 4DFFFF00
VAPP                            = 4DFFFFFF
VCC1                            = 4D00CCFF
VCC2                            = 4DCCFF00
VCC3                            = 4DFF00CC
# AOI
VCC1_AOI                        = 4D00CCCC
VCC2_AOI                        = 4D0000CC

# Unused colors
#4D00CC00
#4DCC0000
#4DCC00CC
#4DCCCCCC
#4DCCFFFF
#4DCCCCFF
#4DFFFFCC
#4DFFCCCC
```

```
137 // Comment, name or description
138
139 CharacteristicsPointsFileName    = CharacteristicsPoints
140 CharacteristicsPointsFileDescription = Includes all Characteristics points
141 AirportFolderName                = Airport
142 AirPortFolderDescription        = The online CSCIs use AIRPORT_I, AIRPORT_II and
        AIRPORT_III to distinguish airports from the other kind of points.
143 DummyFolderName                 = Dummy
144 DummyFolderDescription          = The DUMMY type is used to define a sector
        crossing point associated to a sector for FDP. Dummy point coordinates are not
        significant, but have to be input for the point to be correctly validated
145 ReportFolderName                = Report
146 ReportFolderDescription         = not defined
147 VorFolderName                   = Vor
148 VorFolderDescription            = not defined
149 RoutesFileName                  = Routes
150 RoutesFileDescription           = Includes all Routes referenced
151 FplFileName                     = Fpl
152 FplFileDescription              = Flight plan
153 AdsFileName                     = Ads
154 AdsFileDescription              = Automatic Dependant Surveillance
155 CodedRoutesFolderName           = Simple coded routes
156 CodedRoutesFolderDescription    = Composed of a list of points defined in the
        CHARACTERISTIC_POINTS file.
157 SidFolderName                   = SID
158 SidFolderDescription            = Standard Instrument Departure coded routes
159 StarFolderName                  = STAR
160 StarFolderDescription           = Standard Arrival coded routes.
161 SectorFileName                  = FIR
162 SectorFileDescription           = Includes all FIR
163 FirNTTTFolderName               = FIR NTTT
164 FirNTTTFolderDescription        = All sector of NTTT FIR
165 FirExtFolderName                = FIR EXT
166 FirExtFolderDescription         = All sector of NTTT FIRAll sector of other FIR
167 MainFileName                    = AirControl
168 MainFileDescription             = FIR de Tahiti
169 AoiFolderName                   = AOI
170 AoiFolderDescription            = AOI
171 VCC1_AOI                        = ACI ACC1 TAHITI a 50NM de la FIR
172 VCC2_AOI                        = ACI ACC2 TAHITI a 51NM de la FIR
173 in                              = Entrant
174 out                             = Sortant
175 transit                         = Transit
176 internal                        = Interne
177
178
179 // Folder are open or not in Google Earth at the start
180 # 0 for False and 1 for True
181 Airport                         = 0
```

/home/manu/DTI/manu.cfg

A.1.3 modules/Ads

Ce module lit le fichier de trace Ads du système tiraré et crée pour chaque aéronef un Objet Python ayant pour identifiant le nom de l'avion suivi de la date et l'heure. A cet objet est ensuite associé tout report lui concernant. Il recois donc les message recu par l'Ads-c et converti les point en coordonée. Mais il récupère aussi tout les points intermédiaire calculé par le système.

```
1
2  #!/usr/bin/python
3  #coding: utf-8
4
5  __author__ = "KERVIZIC Emmanuel (kervizic@hotmail.com)"
6  __version__ = "0.0.1"
7  __license__ = ""
8  __copyright__ =""
9
10 print "ADS Charge =) "
11
12 import Convertion, os,re
13 from usualFonction import *
14 from time import gmtime, strftime
15 from datetime import datetime, timedelta
16
17 ads = {}      # dictionary of Ads: { (x) : instance}
18 adsId = {}
19
20 class Ads (object):
21
22
23     def __init__(self, name, description, time, latitude, longitude):
24         """
25         # Creation of a new route
26         #
27         # definition of variables :
28         #
29         # self.name = str         is the name of flight
30         # list of points of the flight with type, time and coordinate
31         # self.points = [{
32         #     'type' : type,
33         #    'time' : time,
34         #     'latitude' : lat,
35         #     'longitude' : long,
36         #     'altitude' : alt
37         #     }]
38           """
39         self.points          = {}
40         self.tracks          = {}
41         self.name              = name + '-' + time.strftime('%Y%m%d-%H%M')
42         adsId[name] = self.name
43         self.firstTime     = time
44         self.nbPoints = 0
45         self.nbTracks = 0
46         self.addPoints ('Logon', description, time, latitude, longitude)
47         ads[str(self.name)] = self
48
49
50     def addPoints (self, type, description, time, latitude, longitude,
51     altitude = 0, pointN = {}, pointN1 = {}) :
52         """
53         Add points
54         """
55         point = {
56             'type' : type,
57             'time' : time,
58             'latitude' : float(latitude),
59             'longitude' : float(longitude),
60             'altitude' : altitude,
61             'description' : description
```

```
62              }
63          if pointN:
64              lat = pointN['latitude']
65              long = pointN['longitude']
66              if lat and long :
67                  point['latitudeN'] = lat
68                  point['longitudeN'] = long
69          if pointN1:
70              lat = pointN1['latitude']
71              long = pointN1['longitude']
72              if lat and long :
73                  point['latitudeN1'] = lat
74                  point['longitudeN1'] = long
75          self.points[self.nbPoints] = point
76          self.nbPoints += 1
77
78      def addTracks (self, type, description, time, latitude, longitude,
79      altitude = 0) :
80          """
81          Add Track : Point calculated by the TIARE systeme
82          """
83          self.tracks[self.nbTracks] = {
84              'type' : type,
85              'time' : time,
86              'latitude' : latitude,
87              'latitude' : float(latitude),
88              'longitude' : float(longitude),
89              'description' : description
90              }
91          self.nbTracks += 1
92
93      def addPeriodic (self, description, time, latitude, longitude, altitude,
94      pointN = {}, PointN1 = {}) :
95          """
96          Add PERIODIC REPORT
97          """
98          self.addPoints('PR', description, time, latitude,
99              longitude, altitude, pointN, PointN1)
100
101     def addAltitudeEvent (self, description, time, latitude, longitude,
102     altitude) :
103         """
104         Add ALTITUDE RANGE DEVIATION EVENT
105         """
106         self.addPoints('Alt', description, time, latitude,
107             longitude, altitude)
108
109     def addWaypointEvent (self, description, time, latitude, longitude,
110     altitude, pointN = {}, PointN1 = {}) :
111         """
112         Add WAYPOINT CHANGE EVENT
113         """
114         self.addPoints('Wayp', description, time, latitude,
115             longitude, altitude, pointN, PointN1)
116
117 def initADS (adresse, points, routes, fpl):
118     """ Analyse le fichier ADS """
119     print 'Debut du traitement des ADS'
120     lstAdsFile = os.listdir(adresse)
121     dateRe = re.compile('(2[0-9]{3})([0-9]{2})([0-9]{2})')
122     for file in sorted(lstAdsFile) :
123         if 'ADS' in file :
124             fileAdresse = adresse + file
125             adsFile = open(fileAdresse,'r') # Open the file
126             adsLine = adsFile.xreadlines()
127             result = dateRe.search(file)
128             date = datetime (
129                 year = int(result.group(1)),
130                 month = int(result.group(2)),
131                 day = int(result.group(3))
132                 )
```

```
133             #print file
134             #print date
135             #print 'ADS file ok'
136             buildAds(adsLine, date)
137     #for key in ads :
138         #print ads[key].name
139         #for k in ads[key].points:
140             #print '-- Point: ' +  str(ads[key].points[k]['type'])
141             #for line in ads[key].points[k]['description']:
142                 #print '---- ' + str(line)
143 
144     print 'Fin du traitement des ADS'
145     return ads
146 
147 def buildAds (adsLine, date) :
148 
149     raz = True
150 
151     logonRe = re.compile("""([0-9]{2}):([0-9]{2}):([0-9]{2}) LOGON RECEIVED fr\
152 om (.+)""")
153     logonCoordinateRe = re.compile("""Aircraft indicates position ([NS])([0-9]\
154 +)([EW])([0-9]+)""")
155     periodicRe = re.compile("""PERIODIC REPORT REPORT RECEIVED for aircraft (.\
156 +) time stamped at : ([0-9]{2}):([0-9]{2}):([0-9]{2})""")
157     waypointRe = re.compile("""WAYPOINT CHANGE EVENT REPORT RECEIVED for aircr\
158 aft (.+) time stamped at : ([0-9]{2}):([0-9]{2}):([0-9]{2})""")
159     altitudeRe = re.compile("""ALTITUDE RANGE DEVIATION EVENT REPORT RECEIVED \
160 for aircraft (.+) time stamped at : ([0-9]{2}):([0-9]{2}):([0-9]{2})""")
161     coordinateRe = re.compile("""Basic Group  Lat :(.+) Long :(.+) Alt :(.+)\
162 """)
163     trackRe =  re.compile("""start extrapolation for (.+) at: ([0-9]{2}):([0-9\
164 ]{2}):([0-9]{2})""")
165     trackLatLongRe =  re.compile("""Position Lat : ?(.+) Long: ?(.+)""")
166     trackAltRe =  re.compile("""Altitude : ?(.+)""")
167     nextRe = re.compile("""NEXT Lat : *(-?[0-9]{1,2}.[0-9]+) *Long : *(-?[0-9]\
168 {1,3}.[0-9]+) *Alt : ([0-9]*) at ([0-9:]*)""")
169     next1Re = re.compile("""NEXT \+ 1 Lat : *(-?[0-9]{1,2}.[0-9]+) *Long : *(-\
170 ?[0-9]{1,3}.[0-9]+) *Alt : *([0-9]*)""")
171     reportRe = re.compile("""Report is correct""")
172     endTrackRe = re.compile("""TRACK extrapolation completed""")
173 
174     i = 0
175     l = 0
176     for line in adsLine :
177         l +=1
178         #print 'Line : ' + str(l)
179         i -= 1
180         # reset in case of errors
181         if i == 0 or raz :
182             if not raz :
183                 print 'error in ads file in line : ' + str(l)
184             logon = ''
185             periodic = ''
186             waypointEvent = ''
187             altitudeEvent = ''
188             track = ''
189             longitude = ''
190             latitude = ''
191             altitude = ''
192             pointN = {}
193             pointN1 = {}
194             description = []
195             time = False
196             raz = False
197             i = 0
198             searchNext = False
199 
200         # Logon
201         resultat = logonRe.search(line)
202         if resultat :
203             i = 5
```

```
204                 logon = resultat.group(4)
205                 time = datetime(
206                     date.year,
207                     date.month,
208                     date.day,
209                     int(resultat.group(1)),
210                     int(resultat.group(2)),
211                     int(resultat.group(3))
212                     )
213             resultat = logonCoordinateRe.search(line)
214             if resultat :
215                 lat = resultat.group(2)
216                 lon = resultat.group(4)
217                 latitude = (
218                     float(lat[0:2]) +
219                     float(lat[2:4] + '.' + lon[4:])/60
220                     )
221                 longitude = (
222                     float(lon[0:3]) +
223                     float(lon[3:5] + '.' + lon[5:])/60
224                     )
225                 if resultat.group(1) == 'S':
226                     latitude = latitude * -1
227                 if resultat.group(3) == 'W':
228                     longitude = -1 * longitude
229             # PERIODIC REPORT
230             result = periodicRe.search(line)
231             if result :
232                 searchNext = True
233                 i = 25
234                 pointN = {}
235                 pointN1 = {}
236                 periodic = result.group(1)
237                 time = datetime(
238                     date.year,
239                     date.month,
240                     date.day,
241                     int(result.group(2)),
242                     int(result.group(3)),
243                     int(result.group(4))
244                     )
245             # ALTITUDE RANGE DEVIATION EVENT
246             result = altitudeRe.search(line)
247             if result :
248                 i = 5
249                 pointN = {}
250                 pointN1 = {}
251                 altitudeEvent = result.group(1)
252                 time = datetime(
253                     date.year,
254                     date.month,
255                     date.day,
256                     int(result.group(2)),
257                     int(result.group(3)),
258                     int(result.group(4))
259                     )
260             # WAYPOINT CHANGE EVENT
261             result = waypointRe.search(line)
262             if result :
263                 searchNext = True
264                 i = 10
265                 pointN = {}
266                 pointN1 = {}
267                 waypointEvent = result.group(1)
268                 time = datetime(
269                     date.year,
270                     date.month,
271                     date.day,
272                     int(result.group(2)),
273                     int(result.group(3)),
274                     int(result.group(4))
```

```
275                 )
276             # AIRCRAFT INTENT TRACK: origin is EXTRAPOLATION
277             result = trackRe.search(line)
278             if result :
279                 i = 100
280                 track = result.group(1)
281                 time = datetime(
282                     date.year,
283                     date.month,
284                     date.day,
285                     int(result.group(2)),
286                     int(result.group(3)),
287                     int(result.group(4))
288                     )
289             # AIRCRAFT INTENT TRACK Coordinate
290             if track :
291                 result = trackLatLongRe.search(line)
292                 if result :
293                     latitude = result.group(1)
294                     longitude = result.group(2)
295                 result = trackAltRe.search(line)
296                 if result :
297                     altitude = result.group(1)
298             # Coordinate
299             result = coordinateRe.search(line)
300             if result :
301                 latitude = result.group(1)
302                 longitude = result.group(2)
303                 altitude = result.group(3)
304             # Coordinate for next point
305             if searchNext :
306                 # Next point
307                 result = nextRe.search(line)
308                 if result :
309                     pointN = {
310                         'latitude' : result.group(1),
311                         'longitude' : result.group(2),
312                         'altitude' : result.group(3),
313                         'at' : result.group(4)
314                         }
315                 # Next Point +1
316                 result = next1Re.search(line)
317                 if result :
318                     pointN1 = {
319                         'latitude' : result.group(1),
320                         'longitude' : result.group(2),
321                         'altitude' : result.group(3)
322                         }
323
324             # Add the file lines in description argument
325             if logon or periodic or waypointEvent or altitudeEvent or track:
326                 description.append(line)
327             else :
328                 description = []
329
330             # traitement des resultats
331
332             # creating a new object has the appearance of a logon.
333             if logon and longitude and latitude :
334                 #print 'Logon : ' + str(logon) + ' at ' + str(time) + ' Lat : ' +
335                     #str(latitude) + ' - Long : ' + str(longitude)
336                 adsObject = Ads(logon, description, time, latitude, longitude)
337                 # reset attributes
338                 raz = True
339
340             # Adding point at the onset of an event, periodic or not.
341             result = reportRe.search(line)
342             if result :
343                 if periodic :
344                     #print '\tPeriodic : ' + str(periodic) + ' at ' + str(time) +
345                         #' Lat : ' + str(latitude) + ' - Long : ' + str(longitude)
```

```
346                 ads[adsId[periodic]].addPeriodic(
347                     description,
348                     time,
349                     latitude,
350                     longitude,
351                     altitude,
352                     pointN,
353                     pointN1
354                     )
355             elif waypointEvent :
356                 #print '\tWaypoint : ' + str(waypointEvent) + ' at ' +
357                     #str(time) + ' Lat : ' + str(latitude) + ' - Long : ' +
358                     #str(longitude)
359                 ads[adsId[waypointEvent]].addWaypointEvent(
360                     description,
361                     time,
362                     latitude,
363                     longitude,
364                     altitude,
365                     pointN,
366                     pointN1
367                     )
368             elif altitudeEvent :
369                 #print '\tAltitude : ' + str(altitudeEvent) + ' at ' +
370                     #str(time) + ' Lat : ' + str(latitude) + ' - Long : ' +
371                     #str(longitude)
372                 ads[adsId[altitudeEvent]].addAltitudeEvent(
373                     description,
374                     time,
375                     latitude,
376                     longitude,
377                     altitude
378                     )
379             # reset attributes
380             raz = True
381
382         # Adding track.
383         result = endTrackRe.search(line)
384         if result and track:
385             ads[adsId[track]].addTracks(
386                 'TRACK',
387                 description,
388                 time, latitude,
389                 longitude,
390                 altitude
391                 )
392
393             # reset attributes
394             raz = True
```

/home/manu/DTI/modules/Ads.py

A.1.4 modules/Aoi

Ce module permet de définir tout les volumes utilisés pour concevoir les zone de contrôles. Il récuper dans le fichiers Asf chaque volume qu'il stocke dans un obejet comprenant chaque coordonée du volume aisin que ca tranche d'altitude.

```
#!/usr/bin/python
#coding: utf-8

__author__ = "KERVIZIC Emmanuel (kervizic@hotmail.com"
__version__ = "0.0.1"
__license__ = ""
__copyright__ =""

print "Fir Charge =) "

import Convertion

from usualFonction import *

points     = {}
aoi    = {}

class Points (object):
    """All Points for the volume"""

    def __init__(self, donnees):
        """
        # Creat a new point
        #
        # definition of variables:
        #
        # self.name = str         is the name of route
        # self.coordinate = {}        is the coordinate of point
         """
        self.definition(donnees)
        # Add a route at the dictionary
        points[str(self.name)] = self

    def definition (self, donnees) :
        """
        # The data arrives in this form:
        # /Point/
        #
        #  NAME
        #    |    COORDINATE
        #    |       |
        # --V--|--V--
        #
        # The separation will therefore be using the separator "|"
        """

        donnees = donnees.strip(None)
        tabDonnees = donnees.split("|")
        for x in xrange(len(tabDonnees)) :
            tabDonnees[x] = tabDonnees[x].strip(None)

        # Assigning variables
        self.name = str(tabDonnees[0])
        self.coordinate = Convertion.convertCoordinate(str(tabDonnees[1]))

class Aoi (object):
    lastAoi = '' # name of the last sid that has been entered

    def __init__(self, donnees, points):
        """
        # Creation of a new Volume
        #
        # definition of variables :
        #
```

```
65          # self.name = str          is the name of route
66          # self.altitude = {}
67          # self.points = {}            is the list of points of the volume
68          """
69          self.defPoints = points
70          self.points = {}
71          self.altitude = {}
72          self.definition(donnees)
73          # Add a route at the dictionary
74          aoi[str(self.name)] = self
75          Aoi.lastAoi = str(self.name)
76
77      def definition (self, donnees):
78          """ Disperssion des édonnes
79          # The data arrives in this form:
80          # /SID/
81          # Convention is as follows:
82          # 1st & 2nd character for the SID point
83          # 3rd & 4th character for the SID number
84          # 5th & 6th character for runway number: 5L=05L, 5R=05R,
85          #                                        3L=23L, 3R=23R
86          #  NAME
87          #    |       COUCHE
88          #    |          |          POINTS
89          #    |          |            |
90          #    V          V            V
91          # ------|---------|----------------------
92          #
93          # La dispertion va donc de faire à l'aide du ésparateur "|"
94          """
95          donnees = donnees.strip(None)
96          tabDonnees = donnees.split("|")
97          for x in xrange(len(tabDonnees)) :
98              tabDonnees[x] = tabDonnees[x].strip(None)
99          # Assignation des variables
100         self.name = str(tabDonnees[0])
101         tabLevel = tabDonnees[1].split('-')
102         altMin = Convertion.convertLevel(str(tabLevel[0][1:].strip(None)))
103         altMax = Convertion.convertLevel(str(tabLevel[1][1:].strip(None)))
104         #print altMin
105         #print altMax
106
107         self.altitude['min'] = altMin
108         self.altitude['max'] = altMax
109         i = len(self.points)
110         tabPoints = tabDonnees[2].split(' ')
111         for x in tabPoints:
112             if x != '' :
113                 self.points[i] = self.defPoints[x]
114                 i += 1
115
116     def addPoints (self, donnees):
117         donnees = donnees.strip(None)
118         tabDonnees = donnees.split("|")
119         for x in xrange(len(tabDonnees)) :
120             tabDonnees[x] = tabDonnees[x].strip(None)
121         # Assignation des variables
122         tabPoints = tabDonnees[2].split(' ')
123         i = len(self.points)
124         for x in tabPoints:
125             if x != '' :
126                 self.points[i] = self.defPoints[x]
127                 i += 1
128
129
130 def initAOI (adresse):
131     """ Analysele fichier FDP_VOLUMES_DEFINITION.ASF """
132
133     fdp = open(adresse,'r') # Open the file
134     section = ""      # correct value: "points" or "aoi"
135                       # Used to be in the document.
```

```
136     lineClean = cleanLine(fdp)
137
138     #for line in fdp.xreadlines(): # acts on each line of file
139         #if line[0] != "-" and len(line) > 5 :
140         ##removes comment lines and blank lines
141             #line=line[0:-1]
142             #lineClean.append(line)
143
144     for line in lineClean :
145         if line[0] == "/" :
146             section = ""
147             if 'POINTS' in line :
148                 section = "points"
149             elif 'SECTOR_AOI' in line :
150                 section = "aoi"
151
152         if section == 'points' and line[0] != "/" :
153             pt = Points(line)
154         elif section == "aoi" and line[0] != "/" :
155             if line[0] != '|' :
156                 a = Aoi(line, points)
157             else :
158                 aoi[str(Aoi.lastAoi)].addPoints(line)
159
160
161     allObjectsAoi = {
162         'points'      : points,
163         'aoi'         : aoi}
164     return allObjectsAoi
```

/home/manu/DTI/modules/Aoi.py

A.1.5 modules/CharacteristicPoints

Ce module met en mémoire tous les points remarquables disponible sur le système. Ces points seront ensuite utilisé pour concevoire les routes et les plans de vols.

```
1  #!/usr/bin/python
2  #coding: utf-8
3
4  __author__ = "KERVIZIC Emmanuel (kervizic@hotmail.com"
5  __version__ = "0.0.1"
6  __license__ = ""
7  __copyright__ =""
8
9  import Convertion
10
11 print "CharcteristicPoints Charge =) "
12 # dictionaire de CharacteristicPoints : { (x) : instance}
13 characteristicPoints = {}
14
15 class CharacteristicPoint :
16     #number = 0 #énumro du points dans la liste
17
18     def __init__(self, donnees):
19         """
20         # éCration d'un point écaractristique
21         # éDfinition des variable :
22         #
23         # self.name = str
24         # self.coordinate = str
25         # self.latidtude = float
26         # self.longitude = float
27         # self.theType = str
28         # self.comment = str
29         # self.airportListFixes = str
30         # self.relFix = bool
31         # self.pilDisplay = bool
32         # self.dti = bool
33         """
34
35         self.addVariables(donnees)
36         # Ajout du point au dictionaire
37         characteristicPoints[str(self.name)] = self
38
39     def addVariables (self, donnees) :
40         """ Disperssion des édonnes """
41         # Les édonnes arrivent sous la forme:
42         # Name
43         # |         Lat/Long
44         # |            |        Type
45         # |            |          |       Rel fix
46         # |            |          |         |     Airport List fixes
47         # |            |          |         |       | PIL display
48         # |            |          |         |       |         | DTI
49         # |            |          |         |       |         |   | Comment
50         # |            |          |         |       |         |   |    |
51         # V            V          V         V       V         V   V    V
52         # -------|------------|----------|---|----------|---|---|--------
53         #
54         # La dispertion va donc de faire à l'aide du ésparateur "|"
55
56         donnees = donnees.strip(None)
57         tabDonnees = donnees.split("|")
58
59         # Assignation des variables
60         self.name = tabDonnees[0].strip(None)
61         self.coordinate = tabDonnees[1].strip(None)
62         self.theType = tabDonnees[2].strip(None)
63         self.airportListFixes = tabDonnees[4].strip(None)
64         self.comment = tabDonnees[7].strip(None)
65         if      tabDonnees[3].strip(None) == "Y" :
```

```
66              self.relFix = True
67          else :
68              self.relFix = False
69          if      tabDonnees[5].strip(None) == "Y" :
70              self.pilDisplay = True
71          else :
72              self.pilDisplay = False
73          if      tabDonnees[6].strip(None) == "Y" :
74              self.dti = True
75          else :
76              self.dti = False
77          # Convertion des écoordonnes en decimales
78          self.decimalCoordinate = Convertion.convertCoordinate(self.coordinate)
79          self.latitude = self.decimalCoordinate['latitude']
80          self.longitude = self.decimalCoordinate['longitude']
81
82
83
84
85  def initCP (adresse):
86      """ éCrer le dictionnaire de Points characteritiqueà
87            partir du fichier émention """
88      print 'Debut du traitement des points characteistique'
89      cp = open(adresse,'r') # Ouvre le fichier
90      for line in cp.xreadlines(): # agit sur chaque ligne du fichier
91          #supprime lignes écommentes et lignes vides
92          if line[0] != "-" and line[0] != "/" and len(line) > 10 :
93              line=line[0:-1]
94              cPoint = CharacteristicPoint(str(line))
95      cp.close()
96      # defines the airports owned by the Tahiti FIR
97      tahitiAirports = []
98      for key in characteristicPoints :
99          cp = characteristicPoints[key]
100         if cp.name[:2] == 'NT' and 'AIRPORT' in cp.theType:
101             tahitiAirports.append(cp.name)
102     print 'Fin du traitement des points characteistique'
103     return characteristicPoints, tahitiAirports
```

/home/manu/DTI/modules/CharacteristicPoints.py

A.1.6 modules/Convertion

Regroupe plusieurs fonctions utiliser pour convertir des donnée.

```
#coding: utf-8
__author__ = "KERVIZIC Emmanuel (kervizic@hotmail.com)"
__version__ = "0.0.1"
__license__ = ""
__copyright__ =""
import re

COORDINATE = re.compile("""\
([0-9]{2})([0-9]{2})?([0-9]{2})?([NS])([0-9]{3})([0-9]{2})?([0-9]{2})?([EW])\
""")

def convertCoordinate (coordinate):
    """ Convert coordinates to decimal format
    # This function saves the latitude and longitude of a point
    # In this system the North and East coordinates are positive
    # The South and West coordinates are negatives
    # hh:mm:ss d (hhmmd) => HH+MM/60+SS/3600*(D)
    """
    result = COORDINATE.search(coordinate)
    latH = result.group(1)
    latM = result.group(2)
    latS = result.group(3)
    latL = result.group(4)
    longH = result.group(5)
    longM = result.group(6)
    longS = result.group(7)
    longL = result.group(8)

    if latL == "N" :
        latD = float(1)
    elif latL == "S" :
        latD = float(-1)
    else :
        latD = None
    if longL == "E" :
        longD = float(1)
    elif longL == "W" :
        longD = float(-1)
    else :
        longD = None
    latitude = float(latH) * latD
    longitude = float(longH) * longD
    try :
        latitude += float(latM)/60 * latD
        longitude += float(longM)/60 * longD
    except :
        pass
    try :
        latitude += float(latS)/3600 * latD
        longitude += float(longS)/3600 * longD
    except :
        pass

    return {'latitude' : latitude, 'longitude' : longitude}


def convertLevel (level) :
    """
    This function converts a flight level in meters...
    """
    newLevel = float(level) * 30.479513
    return newLevel

import math


def distanceBetwennTwoPoint(lat1, long1, lat2, long2, altitude = 0):
```

```
68
69      rayon = 6356.7523142 #km
70      # Add the flight altitude
71      rayon += (float(altitude)/1000)
72      # Convert to milles
73      rayon = ( rayon / 1.852 )
74      # Convert latitude and longitude to
75      # spherical coordinates in radians.
76      degrees_to_radians = math.pi/180.0
77
78      # phi = latitude
79      phi1 = (lat1)*degrees_to_radians
80      phi2 = (lat2)*degrees_to_radians
81
82      # theta = longitude
83      theta1 = long1*degrees_to_radians
84      theta2 = long2*degrees_to_radians
85
86      # Compute spherical distance from spherical coordinates.
87
88      # For two locations in spherical coordinates
89      # (1, theta, phi) and (1, theta, phi)
90      # cosine( arc length ) =
91      #    sin phi sin phi' cos(theta-theta') + cos phi cos phi'
92      # distance = rho * arc length
93
94      cos = ((math.sin(phi1)*math.sin(phi2) +
95              math.cos(phi1)*math.cos(phi2)*math.cos(theta1 - theta2) ))
96      arc = math.acos( cos )
97
98      # Remember to multiply arc by the radius of the earth
99      # in milles
100     #print str(arc) + '*' + str(rayon)
101     distance = float(arc) * float(rayon)
102     return distance
103
104 def speedAndLevel(string) :
105     """
106     Cruising Speed or Mach Number
107     The True Airspeed for the first or the whole cruising portion of
108     the flight, in terms of:
109         - K followed by 4 NUMERICS giving the True Airspeed in kilometres
110         per hour, or
111         - N followed by 4 NUMERICS giving the True Airspeed in knots, or
112         - when so prescribed by the appropriate ATS authority, M followed by 3
113         NUMERICS giving the Mach Number to the nearest hundredth of unit Mach.
114     Requested Cruising Level
115         - F followed by 3 NUMERICS, or
116         - S followed by 4 NUMERICS, or
117         - A followed by 3 NUMERICS, or
118         - M followed by 4 NUMERICS, or
119     See data conventions in 1.6 of page 283 of 4444 :
120     - “”F followed by 3 decimal numerics: indicates a Flight Level Number,
121     i.e. Flight Level 330 is expressed as “”F330;
122     - “”S followed by 4 decimal numerics: indicates Standard Metric Level
123     in tens of metres, i.e. Standard Metric Level 11 300 metres
124     (Flight Level 370) is expressed as “”S1130;
125     - “”A followed by 3 decimal numerics: indicates altitude in hundreds
126     of feet, i.e. an altitude of 4 500 feet is expressed as “”A045;
127     - “”M followed by 4 decimal numerics: indicates altitude in tens of
128     metres, i.e. an altitude of 8 400 metres is expressed as “”M0840.
129     """
130     # Covert the speed in knots
131     speed = ''
132     altitude = ''
133     if string[0] == 'K' :
134         roughSpeed = int(string[1:5])
135         level = string [5:]
136         speed = (float(roughSpeed)/1.852)
137     elif string[0] == 'N' :
138         roughSpeed = int(string[1:5])
```

```
139         level = string [5:]
140         speed = roughSpeed
141     elif string[0] == 'M' :
142         roughSpeed = int(string[1:4])
143         level = string [4:]
144         speed = (float(roughSpeed)*6.6091283725993)
145     else :
146         print 'Convert Speed and Level error for : ' + str(string)
147     if level[0] == 'F' :
148         roughLevel = int(level[1:])
149         altitude = convertLevel(roughLevel)
150     elif level[0] == 'S' :
151         roughLevel = int(level[1:])
152         altitude = (float(roughLevel) * 10)
153     elif level[0] == 'A' :
154         roughLevel = int(level[1:])
155         altitude = (float(roughLevel) * 0.3048 * 100)
156     elif level == 'VFR' :
157         #  Visual flight rules
158         altitude = 0
159     else :
160         print 'Convert Speed and Level error for : ' + str(string)
161     ret = {
162         'speed' : speed,
163         'altitude' : altitude
164         }
165     return ret
166
167 def testCoordinate() :
168
169     coord = '803030N1201515W'
170
171     for i in xrange(100000):
172         coordi = convertCoordinate (coord)
173     print coordi
174
175 if __name__ == '__main__' :
176     import hotshot, os
177     from time import gmtime, strftime
178     print 'Execution du test : testCoordinate'
179     profiler = hotshot.Profile("/home/manu/DTI/stats/statistiques.prf")
180     profiler.runcall(testCoordinate)
181     profiler.close()
182     print 'Test OK, analise des donnees'
183     time = strftime("%Y%m%d-%H%M%S", gmtime())
184     name = ('convertCoordinate' + time +'.prof')
185     cmd = """\
186 cd /home/manu/DTI/stats/
187 hotshot2calltree -o %s statistiques.prf
188 """ % name
189     os.system(cmd)
190     print 'Analyse OK'
191     os.system('kcachegrind /home/manu/DTI/stats/%s' % name)
```

/home/manu/DTI/modules/Convertion.py

A.1.7 modules/Fdp

Définit et met en mémoire toutes les zone de contrôles. Pour definir ces zones les volumes mis en mémoire a l'aide du module Aoi (cf. A.1.4 page 51)

```
#!/usr/bin/python
#coding: utf-8

__author__ = "KERVIZIC Emmanuel (kervizic@hotmail.com)"
__version__ = "0.0.1"
__license__ = ""
__copyright__ =""

print "Fir Charge =) "

import Convertion
from usualFonction import *

points     = {}
layers     = {}
volume     = {}
sector     = {}
fir     = {}

class Points (object):
    """All Points for the volume"""

    def __init__(self, donnees):
        """
        # Creat a new point
        #
        # definition of variables :
        #
        # self.name = str         is the name of route
        # self.coordinate = {}        is the coordinate of point
         """
        self.definition(donnees)
        # Add a route at the dictionary
        points[str(self.name)] = self

    def definition (self, donnees) :
        """
        # The data arrives in this form:
        # /CODED_ROUTE/
        #
        #  NAME
        #    |    COORDINATE
        #    |       |
        # --V--|--V--
        #
        # The separation will therefore be using the separator "|"
        """

        donnees = donnees.strip(None)
        tabDonnees = donnees.split("|")
        for x in xrange(len(tabDonnees)) :
            tabDonnees[x] = tabDonnees[x].strip(None)

        # Assigning variables
        self.name = str(tabDonnees[0])
        self.coordinate = Convertion.convertCoordinate(str(tabDonnees[1]))

class Layer (object):
    """All Points for the volume"""
    lastLevel ='0'

    def __init__(self, donnees):
        """
        # Creat a new Layer
        #
```

```
66          # definition of variables :
67          #
68          # self.name = str         is the name of layer
69          # self.level = {}         is the level of layer
70            """
71          self.definition(donnees)
72          # Add a route at the dictionary
73          layers[str(self.name)] = self
74
75      def definition (self, donnees) :
76          """
77          # The data arrives in this form:
78          # /CODED_ROUTE/
79          #
80          #  Couche (NAME)
81          #    |    Plafond
82          #    |      |
83          # --V--|--V--
84          #
85          # The separation will therefore be using the separator "|"
86          """
87
88          donnees = donnees.strip(None)
89          tabDonnees = donnees.split("|")
90          for x in xrange(len(tabDonnees)) :
91              tabDonnees[x] = tabDonnees[x].strip(None)
92
93          # Assigning variables
94          self.name = str(tabDonnees[0])
95          newLevel = str(int(Convertion.convertLevel(float(tabDonnees[1][1:]))))
96          self.level = {
97              'min' : Layer.lastLevel,
98              'max' : newLevel}
99          Layer.lastLevel = newLevel
100
101
102 class Volume (object):
103     lastVolume = '' # name of the last sid that has been entered
104
105     def __init__(self, donnees, points, layers):
106         """
107         # Creation of a new Volume
108         #
109         # definition of variables :
110         #
111         # self.name = str         is the name of route
112         # self.altitude = {}
113         # self.points = {}          is the list of points of the volume
114         """
115         self.defPoints = points
116         self.layers = layers
117         self.points = {}
118         self.altitude = {}
119         self.definition(donnees)
120         # Add a route at the dictionary
121         volume[str(self.name)] = self
122         Volume.lastVolume = str(self.name)
123
124     def definition (self, donnees) :
125         """ Disperssion des édonnes
126         # The data arrives in this form:
127         # /SID/
128         # Convention is as follows:
129         # 1st & 2nd character for the SID point
130         # 3rd & 4th character for the SID number
131         # 5th & 6th character for runway number: 5L=05L, 5R=05R,
132         #                                        3L=23L, 3R=23R
133         #  NAME
134         #    |       COUCHE
135         #    |          |          POINTS
136         #    |          |            |
```

```
137         #   V         V           V
138         # ------|---------|---------------------
139         #
140         # La dispertion va donc de faire à l'aide du ésparateur "|"
141         """
142
143         donnees = donnees.strip(None)
144         tabDonnees = donnees.split("|")
145         for x in xrange(len(tabDonnees)) :
146             tabDonnees[x] = tabDonnees[x].strip(None)
147         # Assignation des variables
148         self.name = str(tabDonnees[0])
149         if '-' in tabDonnees[1] :
150             tabLevel = tabDonnees[1].split('-')
151             altMin = self.layers[str(tabLevel[0].strip(None))].level['min']
152             altMax = self.layers[str(tabLevel[0].strip(None))].level['max']
153             self.altitude['min'] = altMin
154             self.altitude['max'] = altMax
155         else :
156             self.altitude = layers[str(tabDonnees[1])].level
157         i = len(self.points)
158         tabPoints = tabDonnees[2].split(' ')
159         for x in tabPoints:
160             if x != '' :
161                 self.points[i] = self.defPoints[x]
162                 i += 1
163
164     def addPoints (self, donnees) :
165         donnees = donnees.strip(None)
166         tabDonnees = donnees.split("|")
167         for x in xrange(len(tabDonnees)) :
168             tabDonnees[x] = tabDonnees[x].strip(None)
169         # Assignation des variables
170         tabPoints = tabDonnees[2].split(' ')
171         i = len(self.points)
172         for x in tabPoints:
173             if x != '' :
174                 self.points[i] = self.defPoints[x]
175                 i += 1
176
177 class Sector (object):
178     """all volume of NTTT Fire """
179     lastSector = '' # name of the last sid that has been entered
180
181     def __init__(self, donnees, volume, points, layers):
182         """
183         # Creation of a new Sector
184         #
185         # definition of variables :
186         #
187         # self.name = str         is the name of sector
188         # self.volumes = {}      is the list of volumes of the sector
189         """
190         self.defVolume = volume
191         self.volumes = {}
192         self.definition(donnees)
193         # Add a route at the dictionary
194         sector[str(self.name)] = self
195         Sector.lastSector = str(self.name)
196
197     def definition (self, donnees) :
198         """ Disperssion des édonnes
199         # The data arrives in this form:
200         # /SID/
201         # Convention is as follows:
202         # 1st & 2nd character for the SID point
203         # 3rd & 4th character for the SID number
204         # 5th & 6th character for runway number: 5L=05L, 5R=05R,
205         #                                        3L=23L, 3R=23R
206         #  NAME
207         #    |       GCP
```

```
208         #    |          |        somme des volumes
209         #    |          |           |
210         #    V          V           V
211         # ------|----------|--------------------
212         #
213         # La dispertion va donc de faire à l'aide du ésparateur "|"
214         """
215
216         donnees = donnees.strip(None)
217         tabDonnees = donnees.split("|")
218         for x in xrange(len(tabDonnees)) :
219             tabDonnees[x] = tabDonnees[x].strip(None)
220         # Assignation des variables
221         self.name = str(tabDonnees[0])
222         tabVolumes = tabDonnees[2].split('+')
223         for x in xrange(len(tabVolumes)) :
224             tabVolumes[x] = tabVolumes[x].strip(None)
225         i = len(self.volumes)
226         for x in tabVolumes:
227             if x != '' :
228                 self.volumes[i] = self.defVolume[x]
229                 i += 1
230
231     def addVolume (self, donnees) :
232         donnees = donnees.strip(None)
233         tabDonnees = donnees.split("|")
234         for x in xrange(len(tabDonnees)) :
235             tabDonnees[x] = tabDonnees[x].strip(None)
236         # Assignation des variables
237         tabVolumes = tabDonnees[2].split('+')
238         i = len(self.volumes)
239         for x in xrange(len(tabVolumes)) :
240             tabVolumes[x] = tabVolumes[x].strip(None)
241         for x in tabVolumes:
242             if x != '' :
243                 self.volumes[i] = self.defVolume[x]
244                 i += 1
245
246 class Fir (object):
247     """all volume of NTTT Fire """
248     lastFir = '' # name of the last sid that has been entered
249
250     def __init__(self, donnees, volume, points, layers):
251         """
252         # Creation of a new Sector
253         #
254         # definition of variables :
255         #
256         # self.name = str         is the name of sector
257         # self.volumes = {}      is the list of volumes of the sector
258         """
259         self.defVolume = volume
260         self.volumes = {}
261         self.definition(donnees)
262         # Add a route at the dictionary
263         fir[str(self.name)] = self
264         Fir.lastFir = str(self.name)
265
266     def definition (self, donnees) :
267         """ Disperssion des édonnes
268         # The data arrives in this form:
269         # /FIR/
270         #
271         #  NAME
272         #    |           somme des volumes
273         #    |              |
274         #    V              V
275         # ------|-------------------------
276         #
277         # La dispertion va donc de faire à l'aide du ésparateur "|"
278         """
```

```
279
280         donnees = donnees.strip(None)
281         tabDonnees = donnees.split("|")
282         for x in xrange(len(tabDonnees)) :
283             tabDonnees[x] = tabDonnees[x].strip(None)
284         # Assignation des variables
285         self.name = str(tabDonnees[0])
286         tabVolumes = tabDonnees[1].split('+')
287         for x in xrange(len(tabVolumes)) :
288             tabVolumes[x] = tabVolumes[x].strip(None)
289         i = len(self.volumes)
290         for x in tabVolumes:
291             if x != '' :
292                 self.volumes[i] = self.defVolume[x]
293                 i += 1
294
295 def initFDP (adresse):
296     """ Analysele fichier FDP_VOLUMES_DEFINITION.ASF """
297     print 'Debut du traitement des FIR'
298     fdp = open(adresse,'r') # Open the file
299     section = ""     # correct value: "coded route" , "sid" ou "star"
300                      #Used to be in the document.
301     lineClean = cleanLine(fdp)
302
303     #for line in fdp.xreadlines(): # acts on each line of file
304         #if line[0] != "-" and len(line) > 5 :
305         ##removes comment lines and blank lines
306             #line=line[0:-1]
307             #lineClean.append(line)
308
309     for line in lineClean :
310
311         if line[0] == "/" :
312             section = ""
313             if 'POINTS' in line :
314                 section = "points"
315             elif 'VOLUME' in line :
316                 section = "volume"
317             elif 'LAYER' in line :
318                 section = "layer"
319             elif 'SECTOR' in line :
320                 section = "sector"
321             elif 'FIR' in line :
322                 section = "fir"
323
324         if section == 'points' and line[0] != "/" :
325             pt = Points(line)
326         elif section == 'layer' and line[0] != "/" :
327             lvl = Layer(line)
328         elif section == "volume" and line[0] != "/" :
329             if line[0] != '|' :
330                 vl = Volume(line, points, layers)
331             else :
332                 volume[str(Volume.lastVolume)].addPoints(line)
333         elif section == "sector" and line[0] != "/" :
334             if line[0] != '|' :
335                 sec = Sector(line, volume, points, layers)
336             else :
337                 sector[str(Sector.lastSector)].addVolume(line)
338         elif section == "fir" and line[0] != "/" :
339             if line[0] != '|' :
340                 sec = Fir(line, volume, points, layers)
341             else :
342                 fir[str(Sector.lastSector)].addVolume(line)
343
344     #for key in xrange(len(volume['G1'].points)) :
345         #print str(volume['G1'].points[key].name)+ '    :' + str(volume['G1'].points
            [key].coordinate)
346     #print 'test : ' + str(volume['G1'].points[0].coordinate)
347
348     #for key in xrange(len(sector['VCC1'].volumes)) :
```

```
349         #print str(sector['VCC1'].volumes[key].name)
350
351     allObjectsFdp = {
352         'points'      : points,
353         'volume'      : volume,
354         'layers'      : layers,
355         'sector'      : sector,
356         'fir'         : fir}
357     print 'Fin du traitement des FIR'
358     return allObjectsFdp
359
360
361 if __name__ == '__main__':
362     adresse = '../Sources/FDP_VOLUMES_DEFINITION.ASF'
363     test = initFDP(adresse)
```

/home/manu/DTI/modules/Fdp.py

A.1.8 modules/Fpl

définit et mets en mémoire les plans de vol. Il recupère tous les plan de vol dans les fichiers contenu dans le répertoire "source" et contenat "FPL" dans le nom.

Lorsque aucune date n'est renseigné dans le message Fpl une date est crée arbitrairement en fonction de la date et l'heure d'envoi du message et l'heure de décolage de l'avion.

les trames Fpl (expliqué a la ligne 60 de la source) sont sous la forme :
(FPL-THT712-IX-A343/H-SXJIRYGWZ/SD-NTAA1630-N0479F400 DCT MOANA DCT TEANO DCT KARNO DCT 1755S14905W DCT PASTI DCT CORAL DCT OVINI DCT ONIDO DCT 18S149W/N0477F410 DCT DEBUT DCT FULL DCT FIN DCT BENKO/N0321F050 DCT TETIA DCT MANEV DCT BB/N0321F200 DCT IDUTA DCT-NTAA0345 NCRG-REG/FOSUN SEL/BMER DAT/SV NAV/RNP10 DLE/ONIDO 0061 DLE/BB 0027 RMK/CHARTER FLIGHT FOR ECLIPSE TRACK SOUTH OF TAHITI AND MEHETIA AT FL 410 EXPECT SIGHT SEEING REQUEST FOR DEPARTURE OVER MOOREA AND TAHITI AFTER THE ECLIPSE EXPECT SAME REQUEST UP TO TETIAROA AND BORA BORA BEFORE LANDING IN PPT EXPECT A 5000FT REQUEST OVER BORA BORA-E/0940 P/TBN R/VE S/M J/L D/8 440 C YELLOW A/BLUE/WHITE)

Toutes erreur est enregistrée dans un fichier de log.

```
1  #!/usr/bin/python
2  #coding: utf-8
3
4  __author__ = "KERVIZIC Emmanuel (kervizic@hotmail.com)"
5  __version__ = "0.0.1"
6  __license__ = ""
7  __copyright__ =""
8
9  print "Fpl Charge =) "
10
11 import Convertion, os, re
12 from usualFonction import *
13 from time import gmtime, strftime
14 from datetime import datetime, timedelta
15
16 fpl = {}      # dictionary of fpl : { (x) : instance}
17 ETX = chr(3)
18 FPL = re.compile("""\
19 \(FPL-*(.+?) *- *(.+?) *- *(.+?) *- *(.+?) *- *([A-Z]{4}[0-9]{4}).*- *([KNM][\
20 0-9]{3,4}([FSA]|VFR)[0-9]{0,4}) (.+?) *- *([A-Z]{4}[0-9]{4}).*-(.*?) *(\)|$)\
21 """)
22
23 class Fpl (object):
24
25     log = ''
26     def __init__(self, donnees, date, points, routes, airport):
27         """
28         # Creation of a new route
29         #
30         # definition of variables :
31         #
32         # self.name = str         is the name of route
33         # self.sense = str         is the sens of route
34         # self.odds = str         ODDS/EVENS or NONE
35         # self.points = []          is the list of points of the route
36           """
37         self.defPoints       = points
38         self.defRoutes       = routes
```

```
39          self.defAirport       = airport
40          self.description      = donnees
41          self.messageTime      = date
42          self.points           = {}
43          self.altitude         = {}
44          self.speed            = {}
45          self.name             = ''
46          self.trajectTime      = {}
47          self.deparatureTime   = datetime(2000,1,1)
48          self.estimatedTime    = timedelta(0)
49          self.flightStatus     = ''
50
51          self.definition(donnees)
52          # Add a fpl at the dictionary
53          if str(self.name) in fpl :
54              del fpl[str(self.name)]
55
56          fpl[str(self.name)] = self
57
58      def definition (self, donnees) :
59          """
60          # The data arrives in this form:
61          # (FPL-.......)
62          #
63          #  FPL
64          #   |    Aircraft Identification
65          #   |      |    Flight Rules and Type of flight
66          #   |      |      |  Type of Aircraft / Wake Turbulence Category
67          #   |      |      |      |    Radio Communication, Navigation and
68          #   |      |      |      |    Approach Aid Equipment / Surveillance
69          #   |      |      |      |    Equipment
70          #   |      |      |      |      |  Departure Aerodrome / Time
71          #   |      |      |      |      |      | Point, Routes ...
72          #   |      |      |      |      |      |      |  Destination Aerodrome
73          #   |      |      |      |      |      |      |  / Total Estimated
74          #   |      |      |      |      |      |      |  Elapsed Time
75          #   |      |      |      |      |      |      |      |   EET ...
76          #   |      |      |      |      |      |      |      |     |
77          #   V      V      V      V      V      V      V      V     V
78          # ____-_____-_____-_____-_____-_____-_____-_____-_____
79          #
80          # The separation will therefore be using the separator "-" and " "
81          """
82
83          donnees = donnees.strip(None)
84          tabDonnees = donnees.split("-")
85          for x in xrange(len(tabDonnees)) :
86              tabDonnees[x] = tabDonnees[x].strip(None)
87
88          # Assigning variables
89          if 'FPL' in tabDonnees[0] :
90              self.aircraftID = str(tabDonnees[1]) #Aircraft Identification
91              self.flightRulesAndType = str(tabDonnees[2])
92              self.aircraftType = str(tabDonnees[3])
93              self.equipement = str(tabDonnees[4])
94              self.deparatureAerodrome = str(tabDonnees[5][0:4])
95              self.dTime = str(tabDonnees[5][4:8])
96              self.pointsAndRoutes = tabDonnees[6].split(" ")
97              self.destinationAerodrome = str(tabDonnees[7][0:4])
98              self.eTime = str(tabDonnees[7][4:8])
99              self.comments = tabDonnees[8]
100             self.estimatedTime = timedelta(
101                 hours = int(self.eTime[0:2]),
102                 minutes = int(self.eTime[2:])
103                 )
104             self.setDeparatureTime()
105             self.addPoints()
106             self.setDistance()
107             self.addTime()
108             self.setFlightPlanStatus()
109             self.name = (str(self.aircraftID) +'-'+
```

```
110                 str(self.departureAerodrome) + '-' +
111                 self.departureTime.strftime("%Y%m%d-%H%M")
112                 )
113
114     def setFlightPlanStatus(self):
115         """
116         define with the departure and arival aerodrome if the flight is :
117         in, out, transit or internal.
118         """
119         if self.departureAerodrome in self.defAirport :
120             if self.destinationAerodrome in self.defAirport :
121                 self.flightStatus = 'internal'
122             else :
123                 self.flightStatus = 'out'
124         elif self.destinationAerodrome in self.defAirport :
125             self.flightStatus = 'in'
126         else :
127             self.flightStatus = 'transit'
128
129
130     def setDeparatureTime(self):
131         dTime = self.dTime
132         time= self.messageTime
133         if r'DOF/' in self.comments :
134             debut = self.comments.find(r'DOF/')
135             dof = '20' + self.comments[debut+4:debut+10]
136             if dof.isdigit() :
137                 self.departureTime = datetime(
138                     int(dof[0:4]),
139                     int(dof[4:6]),
140                     int(dof[6:]),
141                     int(dTime[0:2]),
142                     int(dTime[2:])
143                     )
144                 #print self.departureTime
145             else :
146                 Fpl.log += ('DOF error = Flight : ' +
147                     self.name + ' - new departure time :' +
148                     str(self.departureTime) + '\n')
149
150         if self.departureTime.year == 2000 :
151             # determines the day of departure from the day and time
152             # the message was sent. the day of departure is the same as
153             # that of sending the following message it was sent within
154             # 2 hours after the departure of the aircraft.
155             dTimeMn = int(dTime[0:2])*60 + int(dTime[2:]) # Deparature
156             fTimeMn = int(time.hour)*60 + int(time.minute) # Message
157             # + 120 min for the 2 hours
158             deltaTime= dTimeMn + 120 - fTimeMn
159             if deltaTime >= 1440 :
160                 self.departureTime = datetime(
161                     time.year,
162                     time.month,
163                     time.day - 1,
164                     int(dTime[0:2]),
165                     int(dTime[2:])
166                     )
167             elif deltaTime < 0 :
168                 self.departureTime = datetime(
169                     time.year,
170                     time.month,
171                     time.day + 1,
172                     int(dTime[0:2]),
173                     int(dTime[2:])
174                     )
175             else :
176                 self.departureTime = datetime(
177                     time.year,
178                     time.month,
179                     time.day,
180                     int(dTime[0:2]),
```

```
181                     int(dTime[2:])
182                     )
183             Fpl.log += ('No DOF = Flight : ' +
184                 self.name + ' - new deparature time :' +
185                 str(self.deparatureTime) + '\n')
186 
187     def addPoints (self) :
188         pointsAndRoutes = self.pointsAndRoutes
189         self.listOfPoints=[]
190         pointRoute = []
191         convert = Convertion.speedAndLevel(pointsAndRoutes[0])
192         curentAltitude = convert['altitude']
193         curentSpeed = convert['speed']
194         self.listOfPoints.append(self.deparatureAerodrome)
195         self.altitude[self.deparatureAerodrome] = curentAltitude
196         self.speed[self.deparatureAerodrome] = curentSpeed
197         #print 'debut: ' + str(pointsAndRoutes[0])
198         for x in xrange(1,len(pointsAndRoutes)):
199             if "/" in pointsAndRoutes[x]:
200                 tab = pointsAndRoutes[x].split("/")
201                 if tab[1] :
202                     convert = Convertion.speedAndLevel(tab[1])
203                     curentAltitude = convert['altitude']
204                     curentSpeed = convert['speed']
205                 if tab[0] :
206                     pointRoute.append(tab[0])
207                     self.altitude[tab[0]] = curentAltitude
208                     self.speed[tab[0]] = curentSpeed
209             else :
210                 if pointsAndRoutes[x] :
211                     pointRoute.append(pointsAndRoutes[x])
212                     self.altitude[pointsAndRoutes[x]] = curentAltitude
213                     self.speed[pointsAndRoutes[x]] = curentSpeed
214         for x in xrange(len(pointRoute)):
215             if pointRoute[x] in self.defPoints :
216                 point = self.defPoints[pointRoute[x]]
217                 self.listOfPoints.append(point.name)
218             elif pointRoute[x] in self.defRoutes :
219                 route = self.defRoutes[pointRoute[x]]
220                 curentRouteAltitude = self.altitude[pointRoute[x]]
221                 curentRouteSpeed = self.speed[pointRoute[x]]
222                 yDebut = -1
223                 yFin = -1
224                 listPoint = []
225                 for y in xrange(len(route.points)) :
226                     if route.points[y] == pointRoute[x-1] :
227                         yDebut = y
228                     elif route.points[y] == pointRoute[x+1] :
229                         yFin = y
230                 if yDebut == -1 or yFin == -1 :
231                     pass
232                 elif yDebut < yFin :
233                     for i in range(yDebut+1, yFin) :
234                         point = route.points[i]
235                         listPoint.append(point)
236                         self.altitude[point] = curentRouteAltitude
237                         self.speed[point] = curentRouteSpeed
238                     for i in xrange(len(listPoint)) :
239                         point = listPoint[i]
240                         self.listOfPoints.append(point)
241                 else :
242                     for i in range(yFin+1, yDebut) :
243                         point = route.points[i]
244                         listPoint.append(point)
245                         self.altitude[point] = curentRouteAltitude
246                         self.speed[point] = curentRouteSpeed
247                     leng = len(self.listOfPoints)
248                     for i in xrange(len(listPoint)) :
249                         point = listPoint[i]
250                         self.listOfPoints.insert(leng, point)
251             else :
```

```
252                 self.listOfPoints.append(pointRoute[x])
253         self.listOfPoints.append(self.destinationAerodrome)
254         self.altitude[self.destinationAerodrome] = curentAltitude
255         self.speed[self.destinationAerodrome] = curentSpeed
256 
257         i = 0
258         for x in xrange(len(self.listOfPoints)):
259             point = self.listOfPoints[x]
260             if point in self.defPoints :
261                 defPoint = self.defPoints[self.listOfPoints[x]]
262                 coordinate = defPoint.decimalCoordinate
263                 name = defPoint.name
264                 self.points[i]= {
265                     'coordinate' : coordinate,
266                     'name' : name,
267                     'altitude' : self.altitude[name],
268                     'speed' : self.speed[name]
269                     }
270                 i += 1
271             elif point == 'DCT' or point == '':
272                 pass
273             else :
274                  try :
275                     coordinate = Convertion.convertCoordinate(point)
276                     name = point
277                     self.points[i]= {
278                         'coordinate' : coordinate,
279                         'name' : name,
280                         'altitude' : self.altitude[name],
281                         'speed' : self.speed[name]
282                         }
283                     i += 1
284                  except :
285                     Fpl.log += ('POINT error = Flight : ' +
286                     self.name + '     Point: ' +
287                     str(point) + '\n')
288         #print self.name
289         #if selfprint self.points
290 
291     def setDistance (self) :
292         """
293         Add distance in milles
294         """
295         # initiate the first point
296         self.points[0]['totalDistance'] = 0
297         self.points[0]['lastPointDistance'] = 0
298         for i in xrange(1,len(self.points)):
299             point = self.points[i]
300             lastPoint = self.points[i-1]
301             ltd = lastPoint['totalDistance']
302             distance = Convertion.distanceBetwennTwoPoint(
303                 point['coordinate']['latitude'],
304                 point['coordinate']['longitude'],
305                 lastPoint['coordinate']['latitude'],
306                 lastPoint['coordinate']['longitude'],
307                 lastPoint['altitude']
308                 )
309             lpd = distance
310             td = ltd + lpd
311             point['totalDistance'] = td
312             point['lastPointDistance'] = lpd
313         #print self.points
314 
315     def addTime (self):
316         """
317         Add time for all point
318         """
319         # Depending on the estimate of time elapsed
320         # Determine the total distance
321         totalDistance = self.points[len(self.points)-1]['totalDistance']
322         totalTime = self.estimatedTime
```

```
323         self.points[0]['estimatedTotalTime'] = timedelta(minutes=0)
324         self.points[0]['estimatedPointTime'] = timedelta(minutes=0)
325         for i in xrange(1,len(self.points)):
326             point = self.points[i]
327             lastPoint = self.points[i-1]
328             distance = point['lastPointDistance']
329             lastTime = lastPoint['estimatedTotalTime']
330             time = totalTime *int(distance * 1000000000 / totalDistance)
331             time = time / 1000000000
332             point['estimatedPointTime'] = time
333             point['estimatedTotalTime'] = lastTime + time
334 
335             #print 'Estimated = Total :' + str(totalTime) + ' time :'+
336                         #str(time) + ' lastTime :' + str(lastTime + time)
337         # Depending on the speed of the aircraft
338         # speed is given in miles per hour and the distance in miles
339         self.points[0]['calculateTotalTime'] = timedelta(minutes=0)
340         self.points[0]['calculatePointTime'] = timedelta(minutes=0)
341         for i in xrange(1,len(self.points)):
342             point = self.points[i]
343             speed = point['speed']
344             lastPoint = self.points[i-1]
345             distance = point['lastPointDistance']
346             lastTime = lastPoint['calculateTotalTime']
347             h = distance / speed
348             ms = int((h * 3600) * 10**6)
349             time = timedelta(microseconds=ms)
350             point['calculatePointTime'] = time
351             point['calculateTotalTime'] = lastTime + time
352 
353             #print 'Calculate = Total :' + str(totalTime) + ' time :'+
354                             #str(time) + ' lastTime :' + str(lastTime + time)
355 
356 def initFPL (adresse, points, routes, airport):
357     """ Analysele fichier FDX """
358     print 'Debut du traitement des FPL'
359     log = ''
360     lstFplFile = os.listdir(adresse)
361     for file in sorted(lstFplFile) :
362         if 'FDX' in file :
363             log += '==========\n'
364             log += 'File : ' + str(file) + '\n\n'
365             lineClean = []
366             line2 = ''
367             lstFpl = []
368             isFPL = False
369             theLine = ''
370             test = ''
371             date = ''
372             fplLine = []
373             getLine = False
374             fileAdresse = adresse + file
375             fplFile = open(fileAdresse,'r') # Open the file
376             for line in fplFile.xreadlines(): # acts on each line of file
377                 line=line[0:-1]
378                 line = line.strip(None)
379                 if line :
380                 #removes comment lines and blank lines
381                     fplLine.append(line)
382             #print 'file ok'
383             fplFile.close()
384 
385             for line in fplLine: # acts on each line of file
386                 if (r'(FPL-' in line[0:8] and r')' in line
387                                     and r'Text' not in line):
388                     line2 = str(line)
389                     getLine = True
390                 elif r'(FPL-' in line[0:8] and r'Text' not in line:
391                     isFPL = True
392                     theLine = ''
393                     line2 = ''
```

```
394                     theLine += str(line) + ' '
395                 elif isFPL == True and ETX in line:
396                     isFPL = False
397                     line2 = str(theLine)
398                     getLine = True
399                 elif isFPL == True and r')' not in line:
400                     theLine += str(line) + ' '
401                 elif isFPL == True:
402                     isFPL = False
403                     getLine = True
404                     theLine += str(line) + ' '
405                     line2 = str(theLine)
406                 # Find the date of the flight without DOF
407                 elif 'Updating RX event (for center AFTN) at  ' in line :
408                     debut = line.find('at  ')
409                     tabDate = line[debut+4:].split(' ')
410                     #print tabDate
411                     date = datetime(
412                         int(tabDate[0]),
413                         int(tabDate[1]),
414                         int(tabDate[2]),
415                         int(tabDate[3]),
416                         int(tabDate[4]),
417                         int(tabDate[5])
418                         )
419                     #print date
420 
421                 if getLine and line2 and line2 not in lineClean and date :
422                     getLine = False
423                     lineClean.append(line2)
424                     theFpl = {
425                         'line' : line2,
426                         'date' : date
427                         }
428                     lstFpl.append(theFpl)
429 
430             for theFpl in lstFpl :
431                 #print theFpl
432                 line = theFpl['line']
433                 date = theFpl['date']
434                 result = FPL.search(line)
435                 if result :
436                     line = result.group(0)
437                     fplObject = Fpl(line, date, points, routes, airport)
438                 else :
439                     log += '  Bad line for :' + line + ' at ' + str(date) +'\n'
440 
441                 #fplObject = Fpl(line, date, points, routes)
442     allLog = log + '\n\n\n' + Fpl.log
443     writeLog('BadFpl',allLog)
444     print 'Fin du traitement des FPL'
445     return fpl
```

/home/manu/DTI/modules/Fpl.py

A.1.9 modues/GetOfFiles

Coordonne la récupération des donnée, c'est lui qui va chercher la configuration et lance les modules tel que Aoi, Fdp ou encore Fpl.

```
#coding: utf-8
__author__ = "KERVIZIC Emmanuel (kervizic@hotmail.com)"
__version__ = "0.0.1"
__license__ = ""
__copyright__ =""

import CharacteristicPoints, Routes, KML, Fdp, Aoi, Fpl, os, Ads
from usualFonction import *

cfg = open(r'manu.cfg','r') # Open the file
section     = '' # correct value: "path" , "Style"
style       = {} # define the style dictionary
path        = {} # define the path dictionary
lookAt      = {} # define the LookAt dictionary
color       = {} # define the Color dictionary
comment     = {}
isOpen      = {}
makeFile    = {}
option      = {}
allObjects  = {}
lineClean   = []

def getOfConfigFile ():
    lineClean = cleanLine(cfg) # acts on each line of file

    for line in lineClean :
        if line[0:2] == "//" :
            section = ""
            if 'option' in line :
                section = 'option'
            elif 'path' in line :
                section = 'path'
            elif 'Style' in line :
                section = 'style'
            elif 'LookAt' in line :
                section = 'LookAt'
            elif 'Color' in line :
                section = 'Color'
            elif 'Comment' in line :
                section = 'Comment'
            elif 'open' in line :
                section = 'isOpen'
            elif 'update' in line :
                section = 'make'
        elif section == 'option' and line[0] != '/' :
            tmpLine = line.split('=')
            option[str(tmpLine[0].strip(None))] = str(tmpLine[1].strip(None))
        elif section == 'path' and line[0] != '/' :
            tmpLine = line.split('=')
            path[str(tmpLine[0].strip(None))] = str(tmpLine[1].strip(None))
        elif section == 'style' and line[0] != '/' :
            tmpLine = line.split('=')
            tmpSplit = tmpLine[1].split('-')
            name =str(tmpLine[0].strip(None))
            icon = tmpSplit[0].strip(None)
            scale = tmpSplit[1].strip(None)
            textColor = tmpSplit[2].strip(None)
            style[name] = {
                'icon' : icon,
                'scale' : scale,
                'color' : textColor
                }
        elif section == 'LookAt' and line[0] != '/' :
            tmpLine = line.split('=')
            lookAt[str(tmpLine[0].strip(None))] = str(tmpLine[1].strip(None))
```

```
66          elif section == 'Color' and line[0] != '/' :
67              tmpLine = line.split('=')
68              color[str(tmpLine[0].strip(None))] = str(tmpLine[1].strip(None))
69          elif section == 'Comment' and line[0] != '/' :
70              tmpLine = line.split('=')
71              comment[str(tmpLine[0].strip(None))] = str(tmpLine[1].strip(None))
72          elif section == 'isOpen' and line[0] != '/' :
73              tmpLine = line.split('=')
74              isOpen[str(tmpLine[0].strip(None))] = str(tmpLine[1].strip(None))
75          elif section == 'make' and line[0] != '/' :
76              tmpLine = line.split('=')
77              makeFile[str(tmpLine[0].strip(None))] = str(tmpLine[1].strip(None))
78
79      allObjects ={
80          'option' : option,
81          'style' : style,
82          'path': path,
83          'lookAt' : lookAt,
84          'color' : color,
85          'comment' : comment,
86          'makeFile' : makeFile,
87          'isOpen' : isOpen
88          }
89      return allObjects
90
91
92  def getOfFile () :
93      allObjects = getOfConfigFile ()
94
95      adresse = str(path['routesFilePath'])
96      routes = Routes.initRT(adresse)
97
98      adresse = path['characteristicsPointsFilePath']
99      charactPoints, tahitiAirports = CharacteristicPoints.initCP(adresse)
100
101     adresse = path['fdpFilePath']
102     fdp = Fdp.initFDP(adresse)
103
104     adresse = path['aoiFilePath']
105     aoi = Aoi.initAOI(adresse)
106
107     adresse = path['fplFilePath']
108     fpl = Fpl.initFPL(adresse, charactPoints, routes['codedRoutes'],
109         tahitiAirports)
110
111     adresse = path['adsFilePath']
112     ads = Ads.initADS(adresse, charactPoints, routes['codedRoutes'], fpl)
113
114     allObjects['routes'] = routes
115     allObjects['charactPoints'] = charactPoints
116     allObjects['tahitiAirport'] = tahitiAirports
117     allObjects['fdp'] = fdp
118     allObjects['aoi'] = aoi
119     allObjects['fpl'] = fpl
120     allObjects['ads'] = ads
121     return allObjects
```

/home/manu/DTI/modules/GetOfFiles.py

A.1.10 modules/Kml

Afin de pouvoir réaliser les document KML, un module a été implémenté. Celui-ci a pour objectif de mettre en forme le document final. Il ne réalise aucun calcul. Lors de l'initiation une variable est instencié. Celle-ci accumulera toute la mise en forme du document jusqu'a l'appel de la foction de fin qui permettera de clore cette variable et de l'enregistrer dans un fichier texte.

```
__author__ = "KERVIZIC Emmanuel (kervizic@hotmail.com"
__version__ = "0.0.2"
__license__ = ""
__copyright__ =""

#__author__ = "Jon Goodall <jon.goodall@gmail.com> - http://www.duke.edu/~jgl34"
#__version__ = "0.0.1"

class KML_File (object):
    """ For creating KML files used for Google Earth """

    def __init__(self, filePath, name, description, kmlFilePath, lookAt, isOpen):
        """
        Inport array whith general longitude, latitude and all
        data necessary for a general LookAt.
        lookAt['altitude'] for all lookat
        lookAt['longitude'] for the folder longitude
        lookAt['llatitude'] for the folder latitude
        lookAt['tilt'] for all lookat
        lookAt['range'] for all lookAt
        lookAt['heading'] for all lookAt
        """
        self.filePath = filePath
        self.lookAt = lookAt
        self.kmlFilePath = kmlFilePath
        self.kml = ''
        self.name = name
        self.isOpen = str(isOpen)
        self.ind = ''
        #adds the kml header to a file
        self.kml += """\
<?xml version="1.0" encoding="UTF-8"?>
<kml xmlns="http://www.opengis.net/kml/2.2">
<Document>
<name> %s </name>
<open> %s </open>
<description> %s </description>
<LookAt>
    <longitude> %s </longitude>
    <latitude> %s </latitude>
    <altitude> %s </altitude>
    <range> %s </range>
    <tilt> %s </tilt>
    <heading> %s </heading>
    <altitudeMode>absolute</altitudeMode>
    <gx:altitudeMode>absolute</gx:altitudeMode>
</LookAt>
""" % (
name, # String
self.isOpen, # String
description, # String
lookAt['longitude'], # String
lookAt['latitude'], # String
lookAt['altitude'], # String
lookAt['range'], # String
lookAt['tilt'], # String
lookAt['heading'] # String
)

    def addPlacemark(self, name ,description , position, visibility,
    style = '', begin = '', end = ''):
```

```
62          # adds the point to a kml file
63          self.kml += """\
64 <Placemark>
65 <name> %s </name>
66 <visibility> %i </visibility>
67 <description><![CDATA[ %s ]]></description>
68 <styleUrl>#%s</styleUrl>
69 """ % (
70 name,
71 visibility,
72 description,
73 style,  # String
74 )
75          if begin and end :
76              self.kml += """\
77 <TimeSpan>
78 <begin> %s </begin>
79 <end> %s </end>
80 </TimeSpan>
81 """ % (
82 begin, # String
83 end # String
84 )
85          self.kml += """\
86 <Point>
87 <altitudeMode>relativeToGround</altitudeMode>
88 <coordinates>%f,%f</coordinates>
89 </Point>
90 </Placemark>
91 """ % (
92 position['longitude'], # Float
93 position['latitude'] # Float
94 )
95
96      def addLine (self, name ,description, color, points, visibility,
97      width, begin = '', end = '') :
98          # adds the line to a kml file
99          self.kml += """\
100 <Placemark>
101 <name> %s </name>
102 <visibility> %i </visibility>
103 <description><![CDATA[ %s ]]></description>
104 <Style>
105 <LineStyle>
106 <color> %s </color>
107 <width> %s </width>
108 </LineStyle>
109 </Style>
110 """ % (
111 name,
112 visibility,
113 description,
114 color,
115 width
116 )
117          if begin and end :
118              self.kml += """\
119 <TimeSpan>
120 <begin> %s </begin>
121 <end> %s </end>
122 </TimeSpan>
123 """ % (
124 begin, # String
125 end # String
126 )
127          self.kml += """\
128 <LineString>
129 <tessellate>1</tessellate>
130 <coordinates>
131 """
132          for i in sorted(points) :
```

```
133                 self.kml += ('%f,%f\n') % (
134                     points[i]['longitude'], # Float
135                     points[i]['latitude'] # Float
136                     )
137             self.kml += """\
138 </coordinates>
139 </LineString>
140 </Placemark>
141 """
142 
143     def writeLookAt (self, lookAtTmp) :
144         self.kml += """\
145 <LookAt>
146 <longitude> %s </longitude>
147 <latitude> %s </latitude>
148 <altitude> %s </altitude>
149 <range> %s </range>
150 <tilt> %s </tilt>
151 <heading> %s </heading>
152 </LookAt>
153 """ % (
154 lookAtTmp['longitude'], # String
155 lookAtTmp['latitude'], # String
156 lookAtTmp['altitude'], # String
157 lookAtTmp['range'], # String
158 lookAtTmp['tilt'], # String
159 lookAtTmp['heading'] # String
160 )
161 
162     def addRegion (self) :
163         self.kml += """\
164 <Region>
165 <LatLonAltBox>
166 <north>90</north>
167 <south>-90</south>
168 <east>180</east>
169 <west>-180</west>
170 <minAltitude>0</minAltitude>
171 <maxAltitude>-1</maxAltitude>
172 </LatLonAltBox>
173 <Lod>
174 <minLodPixels>-1</minLodPixels>
175 <maxLodPixels>200000</maxLodPixels>
176 <minFadeExtent>0</minFadeExtent>
177 <maxFadeExtent>200000</maxFadeExtent>
178 </Lod>
179 </Region>
180 """
181 
182     def addTimeSpan (self, begin, end):
183         self.kml += """\
184 <TimeSpan>
185 <begin> %s </begin>
186 <end> %s </end>
187 </TimeSpan>
188 """ % ( begin, end ) # String
189 
190     def open_folder(self, name, description, isOpen):
191         self.kml += """\
192 <Folder>
193 <name> %s </name>
194 <description><![CDATA[ %s ]]></description>
195 <open> %s </open>
196 <LookAt>
197 <longitude> %s </longitude>
198 <latitude> %s </latitude>
199 <altitude> %s </altitude>
200 <range> %s </range>
201 <tilt> %s </tilt>
202 <heading> %s </heading>
203 </LookAt>
```

```
""" % (
name, # String
description, # String
str(isOpen), # String
self.lookAt['longitude'], # String
self.lookAt['latitude'], # String
self.lookAt['altitude'], # String
self.lookAt['range'], # String
self.lookAt['tilt'], # String
self.lookAt['heading'] # String
)

    def close_folder(self):
        self.kml += '</Folder>\n'

    def openPlacemark (self, name, description, visibility):
        self.kml += """\
<Placemark>
<name> %s </name>
<visibility> %i </visibility>
<description><![CDATA[ %s ]]></description>
""" % (
name,
visibility,
description
)

    def addIconStyle (self, name, imgName, scale = '1', color = 'FFFFFFFF') :
        textScale = float(scale) + 0.3
        self.kml += """\
<Style id="%s">
<IconStyle>
<Icon>
<href>%simages/%s</href>
</Icon>
<scale> %s </scale>
</IconStyle>
<LabelStyle>
<color> %s </color>
<scale> %f </scale>
</LabelStyle>
</Style>
""" % (
name,
self.kmlFilePath,
imgName,
scale,
color,
textScale
)

    def addLineStyle (self, color, width) :
        self.kml += """\
<LineStyle>
<color> %s </color>
<width> %s </width>
</LineStyle>
""" % (
color, # String
width # String
)

    def addPolyStyle (self, color) :
        self.kml += """\
<PolyStyle>
<color> %s </color>
</PolyStyle>
""" % (color)  # String

    def addPolygon (self, polyPoints, altitudeMode):
        self.kml +="""\
```

```
<Polygon>
<altitudeMode> %s </altitudeMode>
<tessellate> %i </tessellate>
<outerBoundaryIs>
<LinearRing>
<coordinates>
""" % ( altitudeMode, 1 )
        for i in xrange(len(polyPoints)) :
            self.kml += '%s \n' % (polyPoints[i])  # String
        self.kml += """\
</coordinates>
</LinearRing>
</outerBoundaryIs>
</Polygon>
"""

    def addSurface (self, name, description, points, visibility, color) :
        # adds the line to a kml file
        self.kml += """\
<Placemark>
<name> %s </name>
<visibility> %i </visibility>
<description><![CDATA[ %s ]]></description>
<Style>
<LineStyle>
<color> %s </color>
<width> %s </width>
</LineStyle>
<PolyStyle>
<color> %s </color>
</PolyStyle>
</Style>
""" % (
name,
visibility,
description,
'FF' + str(color[2:]),
'0.5',
color
)
        self.kml +="""\
<Polygon>
<altitudeMode> %s </altitudeMode>
<tessellate> %i </tessellate>
<outerBoundaryIs>
<LinearRing>
<coordinates>
""" % ( 'clampToGround' , 1 )
        for i in xrange( len(points) - 1 ) :
            self.kml += ('%f,%f\n') % (
                points[i].coordinate['longitude'],
                points[i].coordinate['latitude']
                )
        self.kml += """\
</coordinates>
</LinearRing>
</outerBoundaryIs>
</Polygon>
</Placemark>
"""

    def addNetworkLink (self, name, description, isOpen, path,
                        refreshMode='onChange',refreshInterval=0 ):
        self.kml += """\
<NetworkLink>
<name> %s </name>
<description><![CDATA[ %s ]]></description>
<open> %s </open>
<LookAt>
<longitude> %s </longitude>
<latitude> %s </latitude>
```

```
346 <altitude> %s </altitude>
347 <range> %s </range>
348 <tilt> %s </tilt>
349 <heading> %s </heading>
350 </LookAt>
351 <Url>
352 <href> %s </href>
353 <refreshMode> %s </refreshMode>
354 <refreshInterval> %s </refreshInterval>
355 </Url>
356 </NetworkLink>
357 """ % (
358 name, # String
359 description, # String
360 str(isOpen), # String
361 self.lookAt['longitude'], # String
362 self.lookAt['latitude'], # String
363 self.lookAt['altitude'], # String
364 self.lookAt['range'], # String
365 self.lookAt['tilt'], # String
366 self.lookAt['heading'], # String
367 path, # String
368 refreshMode, # String
369 refreshInterval # String
370 )
371
372     def close(self):
373         self.kml += '</Document>\n</kml>'
374         print 'creation du fichier : ' + str(self.name)
375         file = open(self.filePath,"w")
376         with file :
377             file.write(self.kml)
378         print 'fichier: ' + str(self.name) + ' est cree'
```

/home/manu/DTI/modules/KML.py

A.1.11 modules/MakeKML

C'est le module qui exploite toutes les données en mémoire et crée les fichiers KML.

```
#coding: utf-8
__author__ = "KERVIZIC Emmanuel (kervizic@hotmail.com)"
__version__ = "0.0.1"
__license__ = ""
__copyright__ =""

import CharacteristicPoints, Routes, KML, Fdp, GetOfFiles, usualFonction
import Convertion
from datetime import datetime, timedelta
from random import random

def addStyle (kml, allObjects) :
    for key in allObjects['style'] :
        theStyle = allObjects['style'][key]
        kml.addIconStyle(
            key,
            theStyle['icon'],
            theStyle['scale'],
            theStyle['color']
            )
    #print 'Style OK !'
    return kml

def addCharctPoint (kml, point, visibility, style, lookAt) :
    position = lookAt.copy()
    position['longitude'] = point.longitude
    position['latitude'] = point.latitude
    description = (
        '<p>'+
        'Type of the point : ' + point.theType + '<br>'+
        'Rel fix :' + str(point.relFix) + '<br>'+
        'Airport List fixes : ' + str(point.airportListFixes) + '<br>'+
        'PIL display : ' + str(point.pilDisplay) + '<br>'+
        'DTI : ' + str(point.dti) + '<br>'+
        'Comment : ' + str(point.comment) + '<br>'+
        '</p>'
        )
    kml.addPlacemark(
        point.name ,
        description ,
        position,
        visibility ,
        style)
    return kml

def addVolume (kml, volume, color) :
    writeKml = kml.addMultiGeometry(
        volume.name,
        '',
        volume.points,
        volume.altitude,
        True,
        color)
    return kml

def addSurface(kml, volume, color) :
    writeKml = kml.addSurface(
        volume.name,
        '',
        volume.points,
        True,
        color)
    return kml

def addAllCharacteristicsPoints(allObjects):
    print "Start Characteristics points"
    kmlFileName = allObjects['comment']['CharacteristicsPointsFileName']
```

```
68      kmlFileDescription = allObjects['comment']['
            CharacteristicsPointsFileDescription']
69      lookAt = allObjects['lookAt']
70      path = allObjects['path']['absCharacteristicsPointsPath']
71      kml = KML.KML_File(
72          path,
73          kmlFileName,
74          kmlFileDescription,
75          allObjects['path']['kmlFilePath'],
76          lookAt ,
77          1)
78
79      kml = addStyle(kml, allObjects)
80      name = allObjects['comment']['AirPortFolderDescription']
81
82      description = allObjects['comment']['AirportFolderName']
83      isOpen = 0
84      kml.open_folder(name, description, isOpen)
85      cp = allObjects['charactPoints']
86      for key in sorted(cp) :
87          if 'AIRPORT' in cp[key].theType :
88              style = 'airports'
89              visibility = True
90              kml = addCharctPoint(kml, cp[key], visibility, style, lookAt)
91      #print "Aeroports ok"
92      kml.close_folder()
93
94      name = allObjects['comment']['DummyFolderName']
95      description = allObjects['comment']['DummyFolderDescription']
96      isOpen = 0
97      kml.open_folder(name, description ,isOpen)
98      cp = allObjects['charactPoints']
99      for key in sorted(cp) :
100         if 'DUMMY' in cp[key].theType :
101             style = 'polygon'
102             visibility = True
103             kml = addCharctPoint(kml, cp[key], visibility, style, lookAt)
104     #print "Dummy ok"
105     kml.close_folder()
106
107     name = allObjects['comment']['ReportFolderName']
108     description = allObjects['comment']['ReportFolderDescription']
109     isOpen = 0
110     kml.open_folder(name, description ,isOpen)
111     cp = allObjects['charactPoints']
112     for key in sorted(cp) :
113         if 'REPORT' in cp[key].theType :
114             style = 'triangle'
115             visibility = True
116             kml = addCharctPoint(kml, cp[key], visibility, style, lookAt)
117     #print "Report ok"
118     kml.close_folder()
119
120     name = allObjects['comment']['VorFolderName']
121     description = allObjects['comment']['VorFolderDescription']
122     isOpen = 0
123     kml.open_folder(name, description , isOpen)
124     cp = allObjects['charactPoints']
125     for key in sorted(cp) :
126         if 'VOR' in cp[key].theType :
127             style = 'triangle'
128             visibility = True
129             kml = addCharctPoint(kml, cp[key], visibility, style, lookAt)
130     #print "Vor ok"
131     kml.close_folder()
132
133     kml.close()
134     print "Characteristics points file OK\n"
135     return kml
136
137 def addRoutes(allObjects) :
```

```
138     print "Start ROUTES"
139     kmlFileName = allObjects['comment']['RoutesFileName']
140     kmlFileDescription = allObjects['comment']['RoutesFileDescription']
141     lookAt = allObjects['lookAt']
142     path = allObjects['path']['absRoutesPath']
143     kml = KML.KML_File(
144         path,
145         kmlFileName,
146         kmlFileDescription,
147         allObjects['path']['kmlFilePath'],
148         lookAt ,
149         1)
150 
151     kml = addStyle(kml, allObjects)
152 
153     name = allObjects['comment']['CodedRoutesFolderName']
154     description = allObjects['comment']['CodedRoutesFolderDescription']
155     isOpen = 0
156     kml.open_folder(name, description, isOpen)
157     cp = allObjects['routes']['codedRoutes']
158     for key in sorted(cp) :
159         color = '7F00FFFF'
160         visibility = True
161         width = 1
162         name = cp[key].name
163         description = 'sense : ' + str(cp[key].sense)
164         i=0
165         points = {}
166         for p in cp[key].points :
167             points[i] = allObjects['charactPoints'][p].decimalCoordinate
168             i += 1
169         writeKml = kml.addLine(name ,description, color,
170             points, visibility, width)
171     #print "Coded line OK"
172     kml.close_folder()
173 
174     name = allObjects['comment']['SidFolderName']
175     description = allObjects['comment']['SidFolderDescription']
176     isOpen = 0
177     kml.open_folder(name, description, isOpen)
178     cp = allObjects['routes']['sid']
179     for key in sorted(cp) :
180         color = '9F00FF00'
181         visibility = True
182         width = 1
183         name = cp[key].name
184         description = ('<b>airport: </b>' + str(cp[key].airport) + '<br>'+
185             '<b>Assigned RWY</b> : ' + str(cp[key].assigned))
186         i=0
187         points = {}
188         for p in cp[key].points :
189             points[i] = allObjects['charactPoints'][p].decimalCoordinate
190             i += 1
191         writeKml = kml.addLine (name ,description, color, points,
192             visibility, width)
193     #print "SID OK"
194     kml.close_folder()
195 
196     name = allObjects['comment']['StarFolderName']
197     description = allObjects['comment']['StarFolderDescription']
198     isOpen = 0
199     kml.open_folder(name, description, isOpen)
200     cp = allObjects['routes']['star']
201     for key in sorted(cp) :
202         color = '9F0000FF'
203         visibility = True
204         width = 1
205         name = cp[key].name
206         description = ('<b>airport: </b>' + str(cp[key].airport) + '<br>'+
207             '<b>Assigned RWY</b> : ' + str(cp[key].assigned))
208         i=0
```

```
209             points = {}
210             for p in cp[key].points :
211                 points[i] = allObjects['charactPoints'][p].decimalCoordinate
212                 i += 1
213             writeKml = kml.addLine (name ,description, color, points,
214                 visibility , width)
215         #print "STAR OK"
216         kml.close_folder()
217 
218         kml.close()
219         print "ROUTES file OK\n"
220         return kml
221 
222 def addSector (allObjects) :
223         print "Start FIR"
224         kmlFileName = allObjects['comment']['SectorFileName']
225         kmlFileDescription = allObjects['comment']['SectorFileDescription']
226         lookAt = allObjects['lookAt']
227         path = allObjects['path']['absFirPath']
228         kml = KML.KML_File(
229             path,
230             kmlFileName,
231             kmlFileDescription,
232             allObjects['path']['kmlFilePath'],
233             lookAt ,
234             1)
235 
236         kml = addStyle(kml, allObjects)
237 
238         # enable the deletion of the area during the approach
239         if allObjects['option']['deleteZone'] == 'yes' :
240             kml.addRegion()
241 
242         color = allObjects['color']
243 
244         name = allObjects['comment']['FirNTTTFolderName']
245         description = allObjects['comment']['FirNTTTFolderDescription']
246         isOpen = 0
247         kml.open_folder(name, description, isOpen)
248         sector = allObjects['fdp']['sector']
249         for key in sorted(sector) :
250             kml.open_folder(sector[key].name, '', False)
251             for key2 in sorted(sector[key].volumes):
252                 volume = sector[key].volumes[key2]
253                 add = addSurface(kml, volume, color[sector[key].name])
254             kml.close_folder()
255         kml.close_folder()
256 
257         name = allObjects['comment']['AoiFolderName']
258         description = allObjects['comment']['AoiFolderDescription']
259         isOpen = 0
260         kml.open_folder(name, description, isOpen)
261         aoi = allObjects['aoi']['aoi']
262         for key in sorted(aoi) :
263             clr = color[aoi[key].name]
264             description = allObjects['comment'][aoi[key].name]
265             kml.open_folder(aoi[key].name, description, 0) # is not open
266             add = addSurface(kml, aoi[key], clr)
267             kml.close_folder()
268         kml.close_folder()
269 
270         name = allObjects['comment']['FirExtFolderName']
271         description = allObjects['comment']['FirExtFolderDescription']
272         isOpen = 0
273         kml.open_folder(name, description, isOpen)
274         fir = allObjects['fdp']['fir']
275         for key in sorted(fir) :
276             clr = color[fir[key].name]
277             kml.open_folder(fir[key].name, '', 0) # is not open
278             for key2 in sorted(fir[key].volumes):
279                 volume = fir[key].volumes[key2]
```

```
280             add = addSurface(kml, volume, clr)
281         kml.close_folder()
282     kml.close_folder()
283 
284     kml.close()
285     print "FIR file OK\n"
286     return kml
287 
288 def addFpl(allObjects) :
289     print "Start FPL"
290     kmlFileName = allObjects['comment']['FplFileName']
291     kmlFileDescription = allObjects['comment']['FplFileDescription']
292     lookAt = allObjects['lookAt']
293     path = allObjects['path']['absFplPath']
294     kml = KML.KML_File(
295         path,
296         kmlFileName,
297         kmlFileDescription,
298         allObjects['path']['kmlFilePath'],
299         lookAt ,
300         1)
301 
302     kml = addStyle(kml, allObjects)
303     status = ['in', 'out', 'transit', 'internal']
304     fpl = allObjects['fpl']
305 
306     for stat in status :
307         name = allObjects['comment'][stat]
308         kml.open_folder(name, '', '0')
309 
310         for key in sorted(fpl) :
311             if fpl[key].flightStatus == stat :
312                 theFpl = fpl[key]
313                 name = theFpl.name
314                 description = fpl[key].description
315                 isOpen = 0
316                 kml.open_folder(name, description, isOpen)
317                 deparatureTime = theFpl.deparatureTime
318                 estimatedTime = theFpl.estimatedTime
319                 arrivalTime = deparatureTime + estimatedTime
320                 color = '8F000000'
321                 visibility = True
322                 width = 1
323                 i=0
324                 points = {}
325                 description = (fpl[key].name + fpl[key].description)
326                 for p in theFpl.points :
327                     points[i] = fpl[key].points[p]['coordinate']
328                     i += 1
329                 begin = deparatureTime.strftime("%Y-%m-%dT%H:%MZ")
330                 end = arrivalTime.strftime("%Y-%m-%dT%H:%MZ")
331                 writeKml = kml.addLine (name ,description, color, points,
332                     visibility, width, begin, end)
333                 x = random()
334                 x = x *16**6
335                 x = hex(int(x))
336                 tmpColor = 'FF' + str(x[2:].zfill(6))
337                 # add the flight in thge time
338                 for i in xrange(len(theFpl.points)-1) :
339                     tmpPoints = {}
340                     tmpPoints[0] = theFpl.points[i]['coordinate']
341                     tmpPoints[1] = theFpl.points[i+1]['coordinate']
342                     if tmpPoints[0] != tmpPoints[1] :
343                         tmpName = (
344                             str(theFpl.points[i]['name']) + '-' +
345                             str(theFpl.points[i+1]['name'])
346                             )
347                         tmpBegin = (
348                             deparatureTime +
349                             theFpl.points[i]['estimatedTotalTime']
350                             )
```

```
351                         tmpEnd = (
352                              deparatureTime +
353                              theFpl.points[i+1]['estimatedTotalTime']
354                              )
355                         begin = tmpBegin.strftime("%Y-%m-%dT%H:%MZ")
356                         end = tmpEnd.strftime("%Y-%m-%dT%H:%MZ")
357                         tmpWidth = 3
358                         tmpDescription = (
359                               str(tmpBegin.strftime("%H:%M")) + '-' +
360                               str(tmpEnd.strftime("%H:%M"))
361                               )
362                         position = lookAt.copy()
363                         position['longitude'] = tmpPoints[0]['longitude']
364                         position['latitude'] = tmpPoints[0]['latitude']
365                         description =''
366                         style = 'blackRound'
367                         ptName = str(tmpBegin.strftime("%H:%M-")) + str(name)
368                         kml.addPlacemark(
369                              ptName,
370                              description ,
371                              position ,
372                              visibility ,
373                              style ,
374                              begin ,
375                              end
376                              )
377                         kml.addLine (
378                              tmpName,
379                              tmpDescription ,
380                              tmpColor ,
381                              tmpPoints ,
382                              visibility ,
383                              tmpWidth,
384                              begin ,
385                              end
386                              )
387                         # Find intrsection with AOI
388                         kml = findAoiIntersection(i,
389                              kml,
390                              tmpPoints,
391                              allObjects,
392                              theFpl,
393                              lookAt,
394                              tmpBegin
395                              )
396                 kml.close_folder()
397         kml.close_folder()
398     #print "FPL line OK"
399
400     kml.close()
401     print "FPL file OK\n"
402     return kml
403
404 def findAoiIntersection(i, kml, tmpPoints, allObjects, theFpl, lookAt,
405 tmpBegin) :
406     # Find intrsection with AOI
407     deparatureTime = theFpl.deparatureTime
408     estimatedTime = theFpl.estimatedTime
409     arrivalTime = deparatureTime + estimatedTime
410     line1 = {
411         'lat1' : tmpPoints[0]['latitude'],
412         'long1' :  tmpPoints[0]['longitude'],
413         'lat2' : tmpPoints[1]['latitude'],
414         'long2' :  tmpPoints[1]['longitude'],
415         }
416     aoi = allObjects['aoi']['aoi']
417     aoiPoints = aoi['VCC1_AOI'].points
418     for j in xrange(len(aoiPoints)-1) :
419         line2 = {
420             'lat1' : aoiPoints[j].coordinate['latitude'],
421             'long1' :  aoiPoints[j].coordinate['longitude'],
```

```
422             'lat2' : aoiPoints[j+1].coordinate['latitude'],
423             'long2' : aoiPoints[j+1].coordinate['longitude'],
424             }
425         intersection = usualFonction.findIntersection(line1, line2)
426         if intersection:
427             distance = Convertion.distanceBetwennTwoPoint(
428                 intersection['latitude'],
429                 intersection['longitude'],
430                 tmpPoints[0]['latitude'],
431                 tmpPoints[0]['longitude'],
432                 theFpl.points[i]['altitude']
433                 )
434             pToPTime = theFpl.points[i+1]['estimatedPointTime']
435             pToPDistance =  theFpl.points[i+1]['lastPointDistance']
436             iTime = pToPTime *int(distance * 1000000000 / pToPDistance)
437             iTime =iTime  / 1000000000
438             intersectionTime = tmpBegin + iTime
439             begin = deparatureTime.strftime("%Y-%m-%dT%H:%MZ")
440             end = arrivalTime.strftime("%Y-%m-%dT%H:%MZ")
441             position = lookAt.copy()
442             position['longitude'] = intersection['longitude']
443             position['latitude'] = intersection['latitude']
444             description =''
445             style = 'redCircle'
446             visibility = 1
447             ptName = ( str(intersectionTime.strftime("%H:%M - ")) +
448                 str(theFpl.name))
449             kml.addPlacemark(
450                 ptName,
451                 description ,
452                 position,
453                 visibility,
454                 style,
455                 begin,
456                 end
457                 )
458     return kml
459
460 def addAds(allObjects) :
461     print 'Start ADS'
462     kmlFileName = allObjects['comment']['AdsFileName']
463     kmlFileDescription = allObjects['comment']['AdsFileDescription']
464     lookAt = allObjects['lookAt']
465     path = allObjects['path']['absAdsPath']
466     kml = KML.KML_File(
467         path,
468         kmlFileName,
469         kmlFileDescription,
470         allObjects['path']['kmlFilePath'],
471         lookAt ,
472         1)
473
474     kml = addStyle(kml, allObjects)
475
476     ads = allObjects['ads']
477
478     for key in sorted(ads) :
479         theAds = ads[key]
480         points = theAds.points
481         tracks = theAds.tracks
482         name = theAds.name
483         description = str(theAds.firstTime)
484         isOpen = 0
485         kml.open_folder(name, description, isOpen)
486
487         # Add ADS and track
488
489         deparatureTime = theAds.firstTime
490         arrivalTime = points[len(points)-1]['time']
491         color = '8F999999'
492         visibility = True
```

```
493             width = 1
494             description = str(theAds.firstTime)
495             begin = deparatureTime.strftime("%Y-%m-%dT%H:%MZ")
496             end = arrivalTime.strftime("%Y-%m-%dT%H:%MZ")
497             writeKml = kml.addLine (name ,description, color, points,
498                 visibility, width, begin, end)
499             x = random()
500             x = x *16**6
501             x = hex(int(x))
502             tmpColor = 'FF' + str(x[2:].zfill(6))
503             # add the flight in thge time
504             if len(points) > 1 :
505                 for i in xrange(len(points)) :
506                     tmpPoints = {}
507                     point = False
508                     nextPoint = False
509                     tmpPoints[0] = point = points[i]
510                     try :
511                         tmpPoints[1] = nextPoint = points[i+1]
512                         tmpName = (
513                             str(point['time'].strftime("%H:%M")) + '-' +
514                             str(nextPoint['time'].strftime("%H:%M"))
515                             )
516                         tmpBegin = (
517                             point['time']
518                             )
519                         tmpEnd = (
520                             nextPoint['time']
521                             )
522                     except:
523                         pass
524
525                     begin = tmpBegin.strftime("%Y-%m-%dT%H:%MZ")
526                     end = tmpEnd.strftime("%Y-%m-%dT%H:%MZ")
527                     tmpWidth = 1
528                     tmpDescription = (
529                         str(name),
530                         str(tmpBegin.strftime("%H:%M")) + '-' +
531                         str(tmpEnd.strftime("%H:%M"))
532                         )
533                     position = lookAt.copy()
534                     position['longitude'] = point['longitude']
535                     position['latitude'] = point['latitude']
536                     description = '<p>'
537                     for line in point['description']:
538                         description += line[:-1] + '<br>'
539                     description += '</p>'
540                     style = 'greenRound'
541                     ptName = (str(theAds.name) + ' : ' + str(point['type']) +
542                         ' - ' + str(point['time'].strftime("%H:%M")))
543                     kml.open_folder(
544                         ptName,
545                         '',
546                         isOpen
547                         )
548                     kml.addPlacemark(
549                         ptName,
550                         description ,
551                         position ,
552                         visibility ,
553                         style ,
554                         begin ,
555                         end
556                         )
557                     try :
558                         position['longitude'] = float(point['longitudeN'])
559                         position['latitude'] = float(point['latitudeN'])
560                         ptName = ptName + 'NEXT'
561                         description = ''
562                         style = 'orangeRound'
563                         kml.addPlacemark(
```

```
564                         ptName,
565                         description ,
566                         position ,
567                         visibility ,
568                         style ,
569                         begin ,
570                         end
571                         )
572                 except :
573                     pass
574                 try :
575                     position['longitude'] = float(point['longitudeN1'])
576                     position['latitude'] = float(point['latitudeN1'])
577                     ptName = ptName + ' +1'
578                     description = ''
579                     style = 'orangeRound'
580                     kml.addPlacemark(
581                         ptName,
582                         description ,
583                         position ,
584                         visibility ,
585                         style ,
586                         begin ,
587                         end
588                         )
589                 except :
590                     pass
591                 if nextPoint :
592                     kml.addLine (
593                         tmpName,
594                         tmpDescription ,
595                         tmpColor ,
596                         tmpPoints ,
597                         visibility ,
598                         tmpWidth ,
599                         begin ,
600                         end
601                         )
602                     kml = addTrack(
603                         tracks ,
604                         lookAt ,
605                         kml,
606                         theAds ,
607                         tmpBegin ,
608                         tmpEnd)
609                 else :
610                     kml = addTrack(
611                         tracks ,
612                         lookAt ,
613                         kml,
614                         theAds ,
615                         tmpEnd)
616
617
618                 kml.close_folder()
619
620         # End Add ADS and Track
621         kml.close_folder()
622     kml.close()
623     print "ADS file OK\n"
624     return kml
625
626 def addTrack (tracks, lookAt, kml, theAds, tmpBegin,
627 tmpEnd = False) :
628     # Add TRACK
629     folder = False
630     visibility = True
631     if not tmpEnd:
632         tmpEnd = tracks[len(tracks)-1]['time']
633     for i in xrange(len(tracks)) :
634         track = tracks[i]
```

```
635         #if i > 0 :
636             #lastTrack = tracks[i-1]
637             #tmpBegin = lastTrack['time']
638         #else :
639             #tmpBegin = track['time']
640         #
641         #if i < len(tracks)-1:
642             #nextTrack = tracks[i+1]
643             #tmpEnd = nextTrack['time']
644         #else :
645             #tmpEnd = track['time']
646         if tmpBegin < track['time'] <= tmpEnd:
647             if not folder :
648                 kml.open_folder(
649                     'TRACK',
650                     'Point calculated by the TIARE systeme',
651                     '0'
652                     )
653                 folder = True
654             begin = tmpBegin.strftime("%Y-%m-%dT%H:%MZ")
655             end = tmpEnd.strftime("%Y-%m-%dT%H:%MZ")
656             position = lookAt.copy()
657             position['longitude'] = track['longitude']
658             position['latitude'] = track['latitude']
659             description = '<p>'
660             for line in track['description']:
661                 description += line[:-1] + '<br>'
662             description += '</p>'
663             style = 'glossyRound'
664             visibility = 1
665             ptName = (str(theAds.name) + ' : ' + str(track['type']) + ' - ' +
666                 str(track['time'].strftime("%H:%M")))
667             kml.addPlacemark(
668                 ptName,
669                 description ,
670                 position ,
671                 visibility ,
672                 style ,
673                 begin ,
674                 end
675                 )
676     if folder :
677         kml.close_folder()
678         # End Add TRACK
679     return kml
680
681 def addMain (allObjects) :
682     print "Start MAin"
683     kmlFileName = allObjects['comment']['MainFileName']
684     kmlFileDescription = allObjects['comment']['MainFileDescription']
685     lookAt = allObjects['lookAt']
686     path= allObjects['path']['absMainPath']
687     kml = KML.KML_File(
688         path,
689         kmlFileName,
690         kmlFileDescription,
691         allObjects['path']['kmlFilePath'],
692         lookAt ,
693         1)
694
695     isOpen = 0
696     lookAt = allObjects['lookAt']
697
698     kml.addNetworkLink(
699         allObjects['comment']['SectorFileName'],
700         allObjects['comment']['SectorFileDescription'],
701         isOpen,
702         path = allObjects['path']['kmlFirPath'])
703     kml.addNetworkLink(
704         allObjects['comment']['RoutesFileName'],
705         allObjects['comment']['RoutesFileDescription'],
```

```
706             isOpen,
707             path = allObjects['path']['kmlRoutesPath'])
708         kml.addNetworkLink(
709             allObjects['comment']['CharacteristicsPointsFileName'],
710             allObjects['comment']['CharacteristicsPointsFileDescription'],
711             isOpen,
712             path = allObjects['path']['kmlCharacteristicsPointsPath'])
713         kml.addNetworkLink(
714             allObjects['comment']['FplFileName'],
715             allObjects['comment']['FplFileDescription'],
716             isOpen,
717             path = allObjects['path']['kmlFplPath'])
718         kml.addNetworkLink(
719             allObjects['comment']['AdsFileName'],
720             allObjects['comment']['AdsFileDescription'],
721             isOpen,
722             path = allObjects['path']['kmlAdsPath'])
723         kml.close()
724         print "Main file OK\n"
```

/home/manu/DTI/modules/MakeKML.py

A.1.12 modules/MakeKMZ

Récupère les fichiers KML pour les regrouper en un fichier KMZ plus maniable. (cf. A.1.12)

```
#!/usr/bin/python
#coding: utf-8

__author__ = "KERVIZIC Emmanuel (kervizic@hotmail.com"
__version__ = "0.0.1"
__license__ = ""
__copyright__ =""

from contextlib import closing
from zipfile import ZipFile, ZIP_DEFLATED
import os, sys

def makeFile(allObjects) :
    basedir = allObjects['path']['absKmlPath']
    archivename = allObjects['path']['absKmzPath']
    assert os.path.isdir(basedir)
    with closing(ZipFile(archivename, "w", ZIP_DEFLATED)) as z:
        for root, dirs, files in os.walk(basedir):
            #NOTE: ignore empty directories
            for fn in files:
                absfn = os.path.join(root, fn)
                zfn = absfn[len(basedir)+len(os.sep):] #XXX: relative path
                z.write(absfn, zfn)
```

/home/manu/DTI/modules/MakeKMZ.py

A.1.13 modules/Routes

Définit et mets en mémoire les routes. Il utilise les points caractéristique précedement enregistrer pour associer les points de chaque route à des coordonées.

```
#!/usr/bin/python
#coding: utf-8

__author__ = "KERVIZIC Emmanuel (kervizic@hotmail.com"
__version__ = "0.0.1"
__license__ = ""
__copyright__ =""

print "Routes Charge =) "
from usualFonction import *
from time import gmtime, strftime
from datetime import datetime, timedelta

codedRoutes = {}      # dictionary of coded routes : { (x) : instance}
sid = {}              # dictionary of sid : { (x) : instance}
star = {}             # dictionary of star : { (x) : instance}

class CodedRoute (object):
    # name of the last route that has been entered
    lastRoute = ""

    def __init__(self, donnees):
        """
        # Creation of a new route
        #
        # definition of variables :
        #
        # self.name = str         is the name of route
        # self.sense = str         is the sens of route
        # self.odds = str         ODDS/EVENS or NONE
        # self.points = []          is the list of points of the route
         """
        self.definition(donnees)
        # Add a route at the dictionary
        codedRoutes[str(self.name)] = self
        CodedRoute.lastRoute = str(self.name)
        #time = strftime("%d%b%Y-%H:%M", gmtime())
        #file = open('LogRoutes' + time + '.log',"a")
        #file.write(self.log)
        #file.close()

    def definition (self, donnees) :
        """
        # The data arrives in this form:
        # /CODED_ROUTE/
        #
        #  NAME
        #    |   SENSE
        #    |     |   ODDS/EVENS or NONE
        #    |     |     |
        #    |     |     |
        #    |     |     |     LIST OF POINTS
        #    |     |     |           |
        # --V--|--V--|---V--|--------V--------|
        #
        # The separation will therefore be using the separator "|"
        """

        donnees = donnees.strip(None)
        tabDonnees = donnees.split("|")
        for x in xrange(len(tabDonnees)) :
            tabDonnees[x] = tabDonnees[x].strip(None)

        # Assigning variables
        if len(tabDonnees) < 5 and len(tabDonnees) > 2 :
```

```
66              self.name = str(tabDonnees[0])
67              self.sense = str(tabDonnees[1])
68              self.odds = str(tabDonnees[2])
69              self.points = tabDonnees[3].split(" ")
70          elif len(tabDonnees) > 5 :
71              self.points.extend(tabDonnees[5].split(" "))
72          self.log = self.name + ' :\n'
73          for point in self.points :
74              self.log += '\t' + point + ' :\n'
75
76
77  class Sid (object):
78      lastSid = "" # name of the last sid that has been entered
79
80      def __init__(self, donnees):
81          """
82          # Creation of a new sid
83          #
84          # definition of variables :
85          #
86          # self.name = str         is the name of route
87          # self.airport = str
88          # self.acft = str
89          # self.assigned = str
90          # self.points = []          is the list of points of the route
91          # self.eligibleRoute = str
92          """
93
94          self.definition(donnees)
95          # Add a route at the dictionary
96          sid[str(self.name)] = self
97          Sid.lastSid = str(self.name)
98
99      def definition (self, donnees) :
100         """ Disperssion des édonnes """
101         # The data arrives in this form:
102         # /SID/
103         # Convention is as follows:
104         # 1st & 2nd character for the SID point
105         # 3rd & 4th character for the SID number
106         # 5th & 6th character for runway number: 5L=05L, 5R=05R,
107         #                                        3L=23L, 3R=23R
108         #  NAME
109         #    |       AIRPORT
110         #    |         |      ACFT PERFORMANCE CATEGORY
111         #    |         |            |    ASSIGNED RWY
112         #    |         |            |       |      LIST OF POINTS
113         #    |         |            |       |        |
114         #    V         V            V       V        V
115         # ------|---------|--------|-------|----------
116         #
117         # La dispertion va donc de faire à l'aide du ésparateur "|"
118
119         donnees = donnees.strip(None)
120         tabDonnees = donnees.split("|")
121         for x in xrange(len(tabDonnees)) :
122             tabDonnees[x] = tabDonnees[x].strip(None)
123
124         # Assignation des variables
125         self.name = str(tabDonnees[0])
126         self.airport = str(tabDonnees[1])
127         self.acft = str(tabDonnees[2])
128         self.assigned = str(tabDonnees[3])
129         self.points = tabDonnees[4].split(" ")
130
131     def setEligibleRoute (self, route) :
132         tabRoute = route.split("|")
133         route = tabRoute[1].strip(None)
134         self.eligibleRoute = str(route)
135
136
```

```
137 class Star (object):
138     # name of the last star that has been entered
139     lastStar = ""
140 
141     def __init__(self, donnees):
142         """
143         # Creation of a new sid
144         #
145         # definition of variables :
146         #
147         # self.name = str         is the name of route
148         # self.airport = str
149         # self.acft = str
150         # self.atg = str
151         # self.assigned = str
152         # self.points = []          is the list of points of the route
153         # self.eligibleRoute = str
154         """
155         self.definition(donnees)
156         # Add a route at the dictionary
157         star[str(self.name)] = self
158         Star.lastStar = str(self.name)
159 
160     def definition (self, donnees) :
161         """ Disperssion des édonnes """
162         # The data arrives in this form:
163         # /STAR/
164         # Convention is as follows:
165         # 1st & 2nd character for the SID point
166         # 3rd & 4th character for the SID number
167         # 5th & 6th character for runway number:
168         #          5L=05L, 5R=05R, 3L=23L, 3R=23R
169         #  NAME
170         #    |       AIRPORT
171         #    |         |     ACFT PERFORMANCE CATEGORY
172         #    |         |           |      ATG ILS
173         #    |         |           |        |    ASSIGNED RWY
174         #    |         |           |        |      |     LIST OF POINTS
175         #    |         |           |        |      |       |
176         #    V         V           V        V      V       V
177         # ------|---------|-------|------|------|-------------
178         #
179         # La dispertion va donc de faire à l'aide du ésparateur "|"
180 
181         donnees = donnees.strip(None)
182         tabDonnees = donnees.split("|")
183         for x in xrange(len(tabDonnees)) :
184             tabDonnees[x] = tabDonnees[x].strip(None)
185         #print tabDonnees[0] + " " + tabDonnees[2]
186 
187         # Assignation des variables
188 
189         self.name = str(tabDonnees[0])
190         self.airport = str(tabDonnees[1])
191         self.acft = str(tabDonnees[2])
192         self.atg = str(tabDonnees[3])
193         self.assigned = str(tabDonnees[4])
194         self.points = tabDonnees[5].split(" ")
195 
196     def setEligibleRoute (self, route) :
197         tabRoute = route.split("|")
198         route = tabRoute[1].strip(None)
199         self.eligibleRoute = str(route)
200 
201 
202 
203 def initRT (adresse):
204     """ Analysele fichier ROUTE.ASF """
205 
206     cp = open(adresse,'r') # Open the file
207     section = ""     # correct value: "coded route" , "sid" ou "star"
```

```
208                     #Used to be in the document.
209     lineClean = cleanLine(cp)
210 
211     #for line in cp.xreadlines(): # acts on each line of file
212         #if line[0] != "-" and len(line) > 5 :
213         ##removes comment lines and blank lines
214             #line=line[0:-1]
215             #lineClean.append(line)
216 
217     for line in lineClean :
218         if line[0] == "/" :
219             section = ""
220             if line [1:12] == "CODED_ROUTE" :
221                 section = "coded route"
222             elif line[1:4] == "SID" :
223                 section = "sid"
224             elif line[1:5] == "STAR" :
225                 section = "star"
226 
227         if section == "coded route" and line[0] != "/" :
228             if line[0] != "|" :
229                 crt = CodedRoute(line)
230             else :
231                 codedRoutes[str(CodedRoute.lastRoute)].definition(line)
232         elif section == "sid" and line[0] != "/" :
233             if line[0:14] != "ELIGIBLE_ROUTE" :
234                 sd = Sid(line)
235             else :
236                 sid[str(Sid.lastSid)].setEligibleRoute(line)
237         elif section == "star" and line[0] != "/" :
238             if line[0:14] != "ELIGIBLE_ROUTE" :
239                 sd = Star(line)
240             else :
241                 star[str(Star.lastStar)].setEligibleRoute(line)
242 
243     cp.close()
244     routes = {
245         'codedRoutes' : codedRoutes,
246         'sid' : sid,
247         'star' : star }
248     return routes
```

/home/manu/DTI/modules/Routes.py

A.1.14 modules/usualFonction

Regroupe plusieurs fonction régulièrement utilisées.

```
from math import *
from decimal import *
from Convertion import *
from time import gmtime, strftime
from datetime import datetime, timedelta

AROUND = 10e15

def cleanLine (txtFile):
    lineClean = []
    for line in txtFile.xreadlines(): # acts on each line of file
        line=line[0:-1]
        line = line.strip(None)
        if line and line[0] != "#" and line[0] != '-':
        #removes comment lines and blank lines
            lineClean.append(line)
            #print '.' + str(line) + '.'
    return lineClean

def cleanLineWithComment (txtFile):
    lineClean = []
    for line in txtFile.xreadlines(): # acts on each line of file
        line=line[0:-1]
        line = line.strip(None)
        if line :
        #removes comment lines and blank lines
            lineClean.append(line)
            #print '.' + str(line) + '.'
    return lineClean

def round (num) :
    num = int(num*AROUND)
    return float(num/AROUND)

def sphericalToCartesian (lat,long):
    # converts spherical coordinates to cartesian coordinates in a point
    lat = radians(float(lat)) # converts degrees to radians
    long = radians(float(long))
    x = cos(lat) * cos(long)
    y = cos(lat) * sin(long)
    z = sin(lat)
    coordinate = {
        'x' : x,
        'y' : y,
        'z' : z
        }
    return coordinate

def findPlane (lat1,long1,lat2,long2):
    #calculate the Cartesian coordinates (x, y, z) points 1 and 2
    #using their spherical coordinates
    c1 = sphericalToCartesian(lat1,long1)
    c2 = sphericalToCartesian(lat2,long2)
    # the point 0 is the center of the earth
    # the plane through 0, c1 and c2 then the equation ax + by + cz = 0
    # a = y1 * z2 - z1 * y2
    # b = z1 * x2 - x1 * z2
    # c = x1 * y2 - y1 * x2
    a = c1['y'] * c2['z'] - c1['z'] * c2['y']
    b = c1['z'] * c2['x'] - c1['x'] * c2['z']
    c = c1['x'] * c2['y'] - c1['y'] * c2['x']
    plane = {
        'a' : a,
        'b' : b,
        'c' : c
        }
    return plane
```

```
68
69  def verifyIntersection (line, point):
70      positive = 360
71      lat = float(point['latitude']) + positive
72      long = float(point['longitude']) + positive
73      lat1 = float(line['lat1']) + positive
74      lat2 = float(line['lat2']) + positive
75      long1 = float(line['long1']) + positive
76      long2 = float(line['long2']) + positive
77      tolerance = 0.01
78      if lat1 > lat2 :
79          maxLat = lat1 + tolerance
80          minLat = lat2 - tolerance
81      else :
82          maxLat = lat2 +tolerance
83          minLat = lat1 - tolerance
84      if long1 > long2 :
85          maxLong = long1 + tolerance
86          minLong = long2 - tolerance
87      else :
88          maxLong = long2 + tolerance
89          minLong = long1 - tolerance
90
91      if minLat < lat < maxLat :
92          if minLong < long < maxLong :
93              return point
94
95  def findIntersection(line1 ,line2) :
96      """
97      line consists of two points defined by latitude and longitude :
98      line = {
99          'lat1' : lat1,
100         'long1' : long1,
101         'lat2' : lat2,
102         'long2' : long2
103         }
104     in decimal
105     """
106     # find the plane of the line in cartesian coordianates
107     p1 = findPlane(line1['lat1'],line1['long1'],
108         line1['lat2'],line1['long2'],)
109     p2 = findPlane(line2['lat1'],line2['long1'],
110         line2['lat2'],line2['long2'],)
111     # The intersection of two planes contains of course the
112     # point of origin, but also the point P : (x,y,z)
113     # x = b1 * c2 - c1 * b2
114     # y = c1 * a2 - a1 * c2
115     # z = a1 * b2 - b1 * a2
116     x = p1['b'] * p2['c'] - p1['c'] * p2['b']
117     y = p1['c'] * p2['a'] - p1['a'] * p2['c']
118     z = p1['a'] * p2['b'] - p1['b'] * p2['a']
119
120     norme = sqrt(x**2+y**2+z**2)
121     lat1 = degrees(asin(round(z/norme)))
122     long1 = degrees(atan2(round(y),round(x)))
123     lat2 = - (lat1)
124     if long1 <= 0 :
125         long2 = long1 + 180
126     else :
127         long2 = long1 - 180
128     intersection1 = {'latitude' : lat1, 'longitude' : long1}
129     intersection2 = {'latitude' : lat2, 'longitude' : long2}
130     point = {'point1' : intersection1, 'point2' : intersection2}
131     intersection = ''
132     for key in point:
133         p = point[key]
134         i1 = verifyIntersection(line1,p)
135         i2 = verifyIntersection(line2,p)
136         if i1 == i2 and not intersection :
137             intersection = i1
138     return intersection
```

```
139
140 def writeLog(fileName, string):
141     date = strftime("%Y%m%d-%H%M%S", gmtime())
142     #time = strftime("%a, %d %b %Y %H:%M:%S (DST Time): ", gmtime())
143     file = open('log/' + fileName + date + '.log', 'w')
144     file.write(string)
145     file.close()
146
147
148 def testIntersection() :
149     for i in xrange(10000) :
150         p11=convertCoordinate ('325111N0925021W')
151         p12=convertCoordinate ('385942N0811837W')
152         p21=convertCoordinate ('394023N0915033W')
153         p22=convertCoordinate ('312426N0843429W')
154         p=convertCoordinate ('353843N0880419W')
155
156         line1 = {
157             'lat1' : p11['latitude'],
158             'long1' : p11['longitude'],
159             'lat2' : p12['latitude'],
160             'long2' : p12['longitude']
161             }
162         line2 = {
163             'lat1' : p21['latitude'],
164             'long1' : p21['longitude'],
165             'lat2' : p22['latitude'],
166             'long2' : p22['longitude']
167             }
168         #print 'point trouve :'
169         intersection = findIntersection(line1 ,line2 )
170         #print 'point lu:'
171         #print p
172
173 if __name__ == '__main__' :
174     import hotshot, os
175     from time import gmtime, strftime
176
177     print 'Execution du test'
178     profiler = hotshot.Profile("/home/manu/DTI/stats/statistiques.prf")
179     profiler.runcall(testIntersection)
180     profiler.close()
181     print 'Test OK, analise des donnees'
182     time = strftime("%Y%m%d-%H%M%S", gmtime())
183     name = ('UsualFonction' + time +'.prof')
184     cmd = """\
185 cd /home/manu/DTI/stats/
186 hotshot2calltree -o %s statistiques.prf
187 """ % name
188     os.system(cmd)
189     print 'Analyse OK'
190     os.system('kcachegrind /home/manu/DTI/stats/%s' % name)
```

/home/manu/DTI/modules/usualFonction.py