



Projet Ingénierie et Entrepreneuriat

PIE 034 - IA pour les Opérations Cockpits Civils

Auteurs :

M^{me} Lorène AUTHIER
M^{me} Elisa CAUDIN
M. Clément MAITRE
M. Mathis POTEL

Encadrants :

M. Hervé GIROD
M. François DEBLY

Version 1 du
9 mars 2020

Déclaration d'Authenticité

Sachant que le plagiat peut se définir par :

- ✈ la copie de phrases entières sans les mettre entre guillemets et/ou sans mentionner la source exacte ;
- ✈ la traduction d'un texte rédigé dans une autre langue que le français sans révéler la source originale ;
- ✈ l'emprunt d'informations précises à une source, y compris une image ou un graphique, à laquelle aucun renvoi n'est effectué ;

Nous, soussignés, Lorène Authier, Elisa Caudin, Clément Maitre, Mathis Potel, élèves du groupe PIE 034 - IA pour les opérations cockpits civils, attestons avoir conçu et rédigé nous-mêmes, dans notre style propre, le travail de maturité ci-joint. Nous attestons notamment ne pas avoir eu recours au plagiat et avoir systématiquement et clairement mentionné les emprunts faits à autrui.

Fait à Toulouse,

Le 09/03/2020

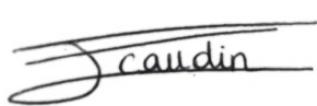
Lorène Authier



Clément Maitre



Elisa Caudin



Mathis Potel



Résumé

Le pilotage d'un avion civil est ponctué par des phases de vol nécessitant une concentration accrue de la part des pilotes. Les approches chargées autour des hubs internationaux ou les situations de déroutement demandent aux pilotes de savoir gérer leur environnement et leur appareil en acquérant des données peu utiles dans le cadre d'un pilotage usuel et donc peu accessibles rapidement. Le but du PIE proposé par Dassault Aviation et mené par notre équipe ces derniers mois est donc de fournir une **assistance** à ces pilotes sous forme d'une **interface numérique**.

Au cours de ce projet, il nous a été demandé d'identifier différents **scénarii** d'utilisation de l'interface et les données utiles aux pilotes. Nous avons ensuite organisé les données au sein d'une **ontologie** avant de réaliser une maquette de démonstration. La maquette permet de se placer dans la peau d'un pilote et de poser un ensemble de questions standardisées et doit s'exécuter en requêtant l'ontologie.

Notre projet s'est conclu par une analyse de l'élargissement du cadre d'utilisation de notre interface. Nous avons notamment étudié la possibilité d'intégrer les responsabilités de **Pilot Monitoring** du co-pilote à notre système ; une étude qui s'inscrit parfaitement dans le cadre des enjeux actuels autour des cockpits mono-pilote.

ISAE-SUPAERO
10, avenue Édouard Belin
BP 54032
31055 Toulouse CEDEX 4

Remerciements

Dans un premier temps, toute l'équipe du PIE-034 souhaite adresser ses plus sincères remerciements aux encadrants du projet qui nous ont guidés dans sa réalisation au cours des 6 derniers mois.

Merci à M. Hervé Girod pour sa proposition de sujet, sa disponibilité pour répondre à nos interrogations et son enthousiasme face aux travaux réalisés. Merci également à M. François Debly pour son suivi de notre gestion du projet et ses conseils.

Finalement, nous remercions tout particulièrement M. Emeric Lazard, ancien pilote de ligne et M. Pierre-Louis Sauvage, cadet Air France, pour leur aide précieuse dans l'identification des cas d'utilisation de notre système. Merci également à tous les pilotes privés qui ont bien voulu répondre à nos questions et qui ont alimenté notre réflexion tout au long du projet.

Table des matières

1	Introduction	1
1.1	Présentation Générale du Contexte	1
1.2	Objectifs du Projet	1
1.3	Vue d'Ensemble du Rapport	2
2	Organisation des Activités	3
2.1	Organisation de l'Equipe et des Travaux	3
2.2	Responsabilités au sein de l'Equipe	3
2.3	Processus de Développement	4
2.3.1	Logique de Développement	4
2.3.2	Définition des Jalons	5
2.3.3	Planning du Projet (Diagramme de Gantt)	5
2.4	Définition détaillée du projet	6
2.4.1	PBS et WBS	6
2.4.2	Spécification du Projet	7
2.4.3	Budget du Projet	9
2.5	Suivi et Contrôle	9
2.5.1	Suivi de l'Avancement	9
2.5.2	Maîtrise des Risques	11
3	Travail Réalisé	13
3.1	Scénarii	13
3.1.1	Navigation	14
3.1.2	Conditions Météorologiques	14
3.1.3	Performances Avions	14
3.2	Questions et Identification des Données	14
3.3	Architecture	16
3.3.1	Navigation	16
3.3.2	Performances	16
3.4	Implémentation de l'Ontologie	18
3.4.1	Les ontologies et le langage OWL	19
3.4.2	L'éditeur <i>Protégé</i>	19
3.4.3	Implémentation de l'architecture	19
3.5	Création d'un Jeu de Données Test	22
3.5.1	Définition du Plan de Vol	22
3.5.2	Vol sur Simulateur	24
3.5.3	Post-traitement des Données	26
3.6	Maquette Finale	26

3.6.1	Les requêtes	26
3.6.2	L'interface utilisateur	27
3.6.3	Réaliser une maquette simple et complète	29
3.7	Validation et Vérification	31
3.7.1	Méthode de vérification	31
3.7.2	Erreurs mises en lumière lors de la phase de vérification	32
3.7.3	Résultats et validation de la maquette	32
4	Travail annexe	33
4.1	Pilote de Surveillance ou Pilot Monitoring	33
5	Conclusion	35
A	Gantt Chart détaillé	36
B	Architecture Préliminaire de l'Ontologie	38
C	Plan de Vol Simulé	40
D	Traduction des Requêtes Textuelles en Requêtes SparQL	47
E	Formules Utilisées pour le Post-traitement des Données	50

Table des figures

2.1	Organigramme de l'Equipe	3
2.2	Matrice RACI	4
2.3	Schéma de Développement suivant le digramme en V	5
2.4	Gantt Chart du projet	6
2.5	PBS	7
2.6	WBS	7
2.7	Heures de travail réalisées et prévues pour le projet	10
2.8	Liste des Risques	11
2.9	Matrice des Risques	11
2.10	Listes de Actions Préventives et Curatives	12
3.1	Avion de référence utilisé pour le projet - Airbus A320 neo <i>source : https://pl.wikipedia.org/wiki/Plik:Airbus_A320neo_first_takeoff_at_Toulouse_Bagnac_Airport_05.jpg</i>	13
3.2	Représentation Graphique de l'Ontologie	18
3.3	Plan de vol au départ de LFBO à destination de LFPO	23
3.4	Plan de vol au départ de LFBO à destination de LFPO avec les STAR et SID	23
3.5	Modèle A320neo sur FlightGear durant le vol LFBO-LFPO	24
3.6	Écran d'accueil puis apparition du bouton pour lancer l'interface de requête	28
3.7	Après avoir lancé l'interface de requête	28
3.8	Cadre d'entrée de texte	28
3.9	Réponse à la requête	29
4.1	Fonctions de PM implémentées dans notre système	33
A.1	Gantt Chart détaillé - 1	36
A.2	Gantt Chart détaillé - 2	37
A.3	Gantt Chart détaillé - 3	37
B.1	Architecture préliminaire de l'ontologie	39
E.1	Courbe indiquant l'altitude optimale en fonction de la masse de l'avion(1)	51
E.2	Vitesses d'approche en fonction de la configuration d'hypersustentateur(1)	52
E.3	Distance d'atterrissage en fonction des conditions de vol(1)	52
E.4	Tableau ACN A321	53

Liste des Sigles et Acronymes

ARFF	<i>Aircraft Rescue and Fire Fighting</i>
ACN	<i>Aircraft Classification Number</i>
FMC	<i>Flight Management Computer</i>
IA	<i>Intelligence Artificielle</i>
IAS	<i>Indicated Air Speed</i>
MLD	<i>Maximum Landing Distance</i>
MORA	<i>Minimum Off-Route Altitude</i>
MTOW	<i>Maximum Take Off Weight</i>
OACI	<i>Organisation de l'Aviation Civile Internationale</i>
PBS	<i>Product Breakdown Structure</i>
PCN	<i>Pavement Classification Number</i>
PIE	<i>Projet Innovation et Entreprise</i>
SID	<i>Standard Instrument Departure</i>
STAR	<i>Standard Terminal Arrival Route</i>
WBS	<i>Work Breakdown Structure</i>

Chapitre 1

Introduction

1.1 Présentation Générale du Contexte

Lors d'un vol, les pilotes peuvent être amenés à rencontrer des situations nécessitant des informations quantitatives et qualitatives sur l'état de l'appareil et son environnement indisponibles directement sur tableau de bord. Les procédures de pannes ou l'identification de pistes d'atterrissement à proximité suite à un déroutement ne sont que quelques exemples qui illustrent le besoin d'un pilote de détourner partiellement son attention du pilotage pour rechercher ou calculer des informations indispensables.

Au cours de ce projet proposé par Dassault Aviation, nous nous proposons de développer une IA intégrée aux cockpits civil dans le but d'alléger la charge de travail des pilotes notamment lors de situations compliquées. Notre IA repose sur la modélisation sous forme d'ontologie d'une partie de l'avion et de son environnement. Elle permettra aux pilotes de l'interroger de manière simple et intelligible pour obtenir des informations d'ordinaire inaccessibles rapidement pour l'aider dans sa prise de décision et son pilotage de l'appareil.

1.2 Objectifs du Projet

Ce projet a pour but de développer une maquette pouvant être interrogée par le pilote afin d'alléger sa charge de travail durant les phases critiques. Nous avons donc identifié plusieurs objectifs principaux qui sont les suivants :

- Identifier les principaux scénarios de sollicitation de l'IA par le pilote.
- Définir une architecture d'ontologie permettant de répondre en priorité aux scénarios identifiés et adaptable à de nouveaux cas d'utilisation.
- Implémenter en langage OWL l'ontologie permettant de modéliser l'avion civil, ses systèmes et son environnement.
- Créer un ensemble d'une vingtaine de requêtes standards pour interroger l'ontologie et les implémenter en SPARQL.
- Créer un jeu de données test cohérent et réaliste pour vérifier le fonctionnement de l'ontologie et des requêtes et valider notre modélisation. Le jeu de données doit simuler un court vol d'une heure.

Plus précisément, la maquette devra comporter les éléments tels que l'ontologie, une série de requêtes pour tester ladite ontologie et ainsi qu'un jeu de données représentatif d'un vol court-courrier. De plus, les différents objectifs présentés précédemment sont listés de manière chronologique pendant la durée totale du projet.

La première phase consistera à établir une architecture d'ontologie adaptée aux scénarios d'utilisation. Cette phase d'architecture est essentielle car c'est sur elle que reposera toute la structure de notre code.

La deuxième phase est donc une phase d'implémentation de l'ontologie en langage OWL et d'un ensemble de requêtes en langage SPARQL.

Finalement, nous vérifierons le fonctionnement de notre ontologie et des requêtes grâce à la création d'un jeu de données simulant un court vol.

1.3 Vue d'Ensemble du Rapport

Ce rapport présentera dans un premier temps, l'organisation des activités en détaillant les responsabilités de chaque membre de l'équipe et l'articulation des activités ainsi que les risques potentiels et leurs solutions. Nous exposerons également les problèmes rencontrés et leurs moyens de résolution. Dans un second temps, nous détaillerons la partie technique en commençant par la partie architecture, suivie de la partie code, comprenant l'ontologie, les requêtes et le jeu de données. Enfin, nous finirons par les parties vérification et validation.

Cette version est celle rendue le 9 mars 2020. Elle sera modifiée après la soutenance en prenant en compte les commentaires du jury et les évolutions du travail d'ici-là.

Chapitre 2

Organisation des Activités

2.1 Organisation de l'Equipe et des Travaux

Le projet comportant deux aspects principaux et complémentaires, nous avons décidé dès son commencement de nous diviser en deux équipes : l'une se concentre sur la définition de l'architecture, l'autre sur le code. Nous effectuerons dans un second temps la phase de test ensemble, puisque des problèmes liés aux deux aspects du projet sont susceptibles d'apparaître. L'organigramme ci-dessous présente l'affectation de chaque membre du groupe de PIE à son équipe.

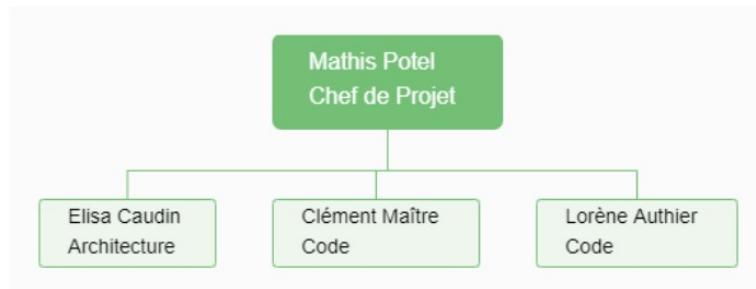


FIGURE 2.1 – Organigramme de l'Equipe

Nous organisons chaque semaine une réunion de suivi pour mettre en commun l'état d'avancement du travail de chacun. Cette réunion se suit d'une séance de travail à quatre dans une salle commune de manière à maintenir un esprit d'équipe et pallier le risque de manque de communication. Afin de maintenir une proximité avec le client et d'être réactifs face à ses exigences, nous nous entretenons avec lui toutes les 2 semaines par téléphone. A l'issu de chaque réunion, un compte-rendu est rédigé par le chef de projet et envoyé par mail au client afin de tenir compte de l'avancement des travaux.

2.2 Responsabilités au sein de l'Equipe

Afin de répartir les différentes tâches au sein de l'équipe, il est impératif d'établir une matrice des responsabilités. Cela nous informe sur la personne responsable ainsi que les personnes impliquées sur les différentes missions représentées par la matrice, qui est affichée figure 2.2. Le code couleur utilisé signifie en rouge "approbateur du travail réalisé",

en orange "réalisateur qui participe au travail", en violet "personne informée" et en vert "consultée".

Tâches	Sous-tâches							
	Lorène Authier	Elisa Caudin	Clement Maître	Mathis Potel	Hervé Girod	Rob Vingerhoeds	François Debly	
Gestion de projet	WBS, OBS, planning, risque, opportunités, plan de développement	R	R	R	A	I	I	C
Projet	Defintion Ontologie	C	R	C	A	C	I	I
	Implémentation Ontologie	A	C	R	C	C	I	I
	Identification des question types	C	R	C	A	C	I	I
	Traduction des questions pilote en requêtes	A	C	R	C	C	I	I
	Creation d'une base de données test	A	C	R	C	C		I
	Verification des réponses aux requêtes	R	R	R	A	C	I	I
Soutenance PIE	Revue technique	R	R	R	A	I	I	I
	Revue de projet	R	R	R	A	C	C	C
	Livrailles des fichiers techniques	R	R	R	A	I	I	I

FIGURE 2.2 – Matrice RACI

2.3 Processus de Développement

2.3.1 Logique de Développement

Le développement du projet suit un cycle en V. En premier lieu, la phase d'analyse, de spécification et conception conduit à la phase d'implémentation. L'architecture et les questions en langage naturel sont traduites sous forme de code pour aboutir à une ontologie OWL et un ensemble de requêtes SPARQL. Le fonctionnement de ces deux items sera ensuite validé par une phase de test sur un jeu de données simulant un court vol. Il est nécessaire dans notre projet d'effectuer autant de retours que possible entre le code et l'architecture. Cela nous permet de nous adapter en temps réel aux éventuels "problèmes de traduction" et de modifier l'architecture initiale. Une bonne communication régulière est donc essentielle au bon pilotage du projet.

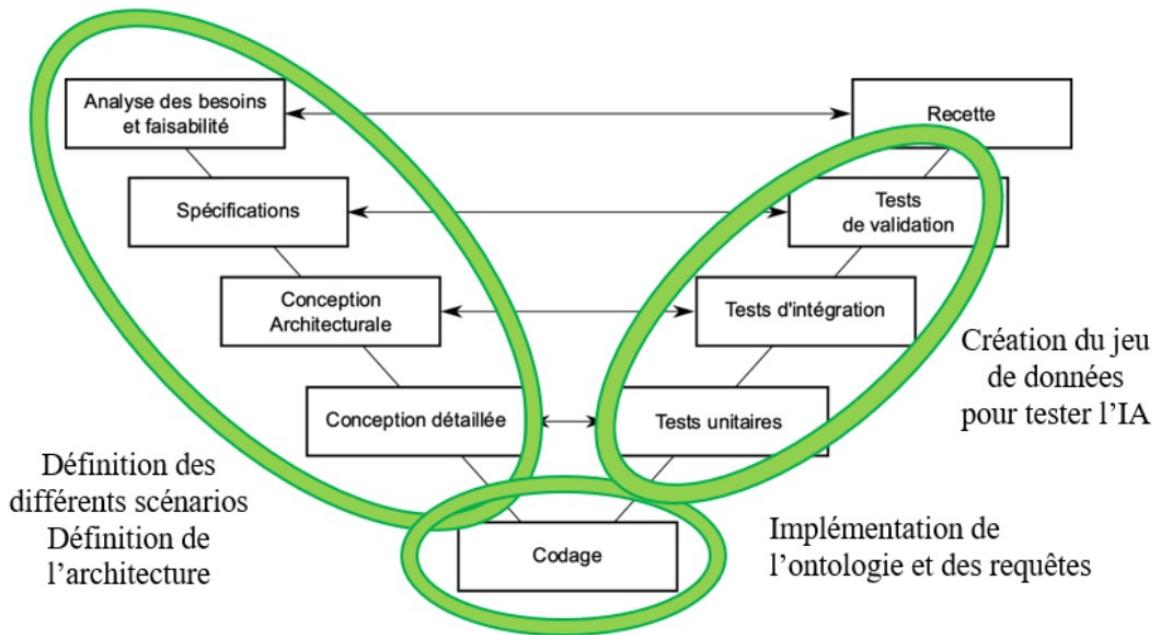


FIGURE 2.3 – Schéma de Développement suivant le diagramme en V

2.3.2 Définition des Jalons

L'école nous a fixé quatre jalons obligatoires au cours du processus de développement du projet. Ces jalons comprennent la soutenance finale ainsi que le rendu des livrables. Nous avons par ailleurs décidé d'ajouter à ces jalons deux jalons de revue technique avec notre client. Ces derniers permettent de nous assurer de la satisfaction des attentes client et d'adapter nos méthodes en fonction des retours effectués.

- 3 décembre 2019 : première revue de projet
- 15 janvier 2020 : première revue technique avec le client
- 27 janvier 2020 : deuxième revue de projet
- 25 février 2020 : deuxième revue technique avec le client
- 16 mars 2020 : soutenance PIE
- 25 mars 2020 : rendu des livrables

2.3.3 Planning du Projet (Diagramme de Gantt)

À partir des diagrammes WBS et PBS explicités figure 2.6 et figure 2.5 et des fiches de tâches, nous avons établi la chronologie de notre projet. Celle-ci est résumée dans le diagramme de Gantt suivant :

2.4. Définition détaillée du projet

6

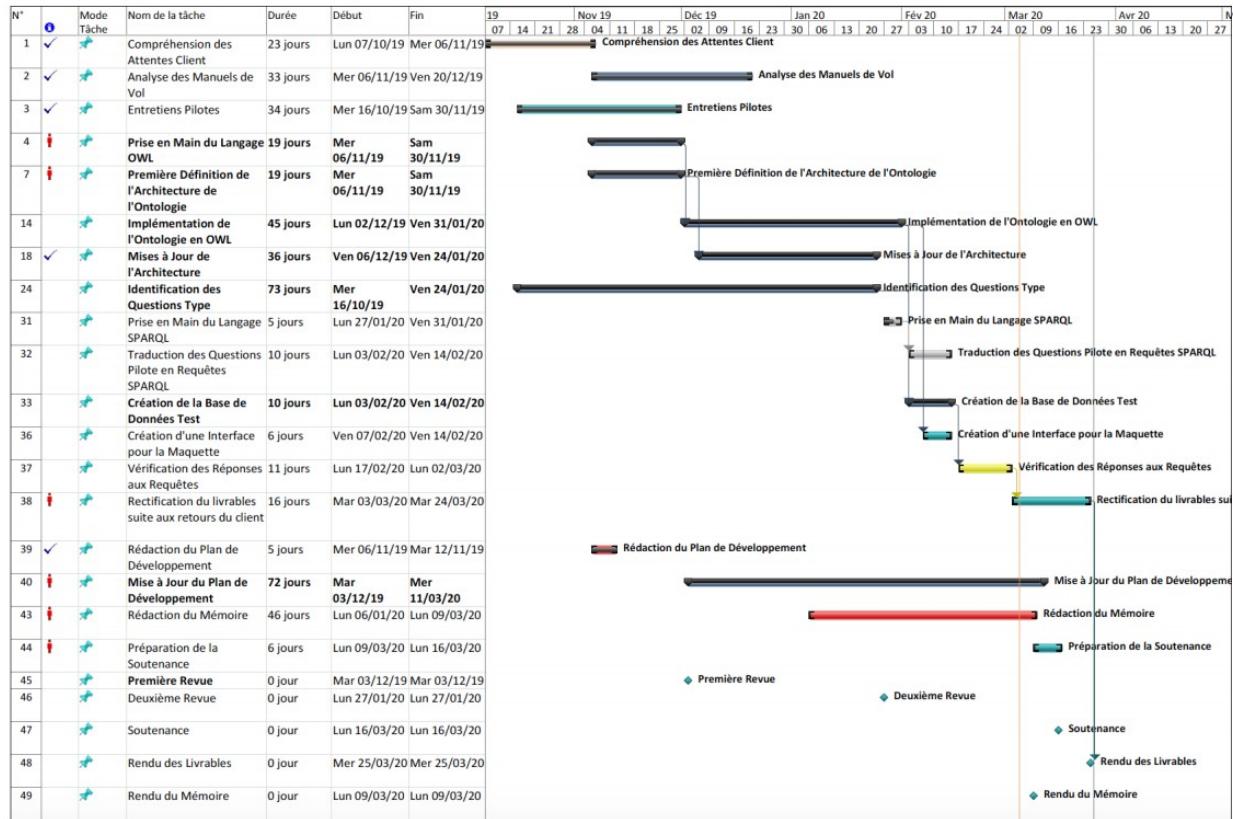


FIGURE 2.4 – Gantt Chart du projet

En annexe A se trouve le Gantt Chart détaillé. En effet, sur chaque graphe figure les sous tâches de la plupart des tâches principales.

2.4 Définition détaillée du projet

2.4.1 PBS et WBS

La définition du PBS a été structurante pour la planification de notre projet. En effet, celui-ci repose entièrement sur la réalisation de livrables successifs permettant d'aboutir à une maquette de démonstration finale. À partir du PBS, nous avons établi la liste des tâches menant à la réalisation de nos 5 produits. En établissant un ordre de priorité chronologique pour la réalisation de chaque produit, nous avons pu développer, dans les grandes lignes, notre première version du diagramme de Gantt. Nos diagrammes PBS et WBS sont présentés ci-dessous.

2.4. Définition détaillée du projet

7

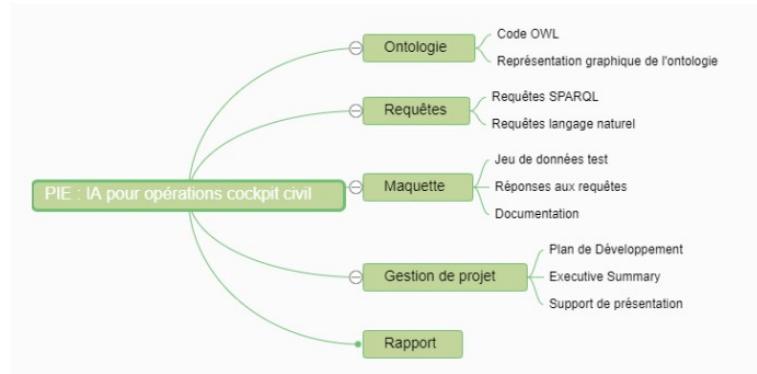


FIGURE 2.5 – PBS

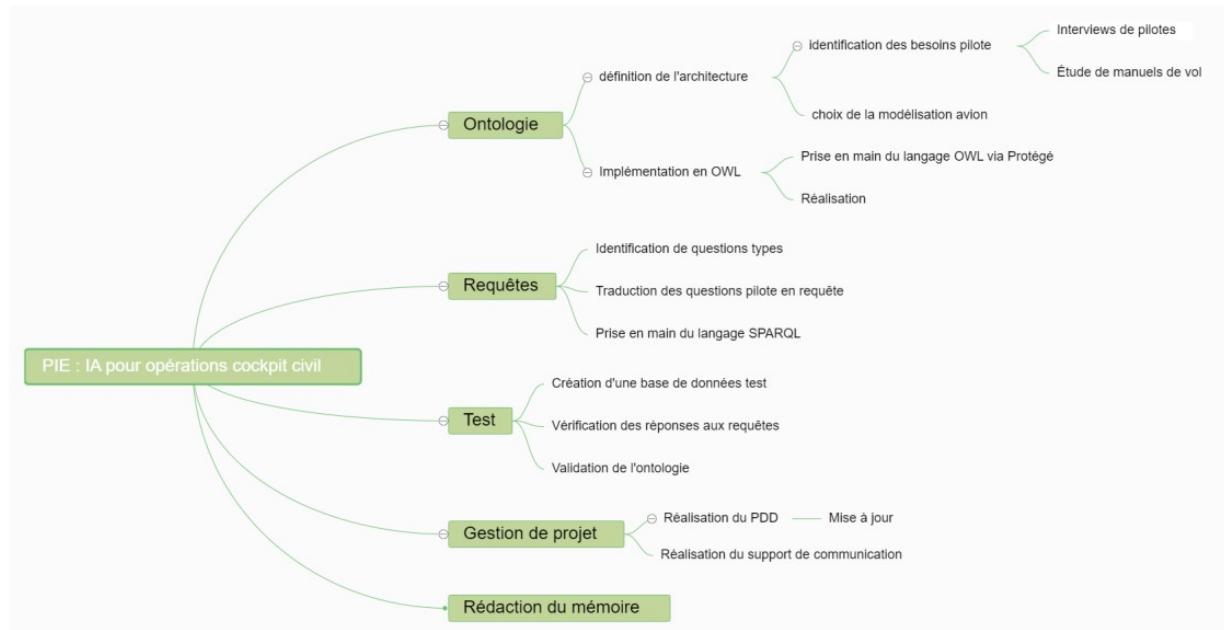


FIGURE 2.6 – WBS

2.4.2 Spécification du Projet

Après concertation avec le client, nous avons identifié ses attentes et avons convenu des résultats attendus suivants :

- Identifier les scénarios d'utilisation plausibles de l'IA par un pilote d'aviation civile.
- Identifier et organiser sous forme d'ontologie les données nécessaires à chaque scénario d'utilisation.
- Etre capable d'interroger l'ontologie et de fournir les réponses adéquates. Les questions et les réponses devront être intelligibles pour un utilisateur anglophone.

Tout au long de notre projet, nous avons gardé à l'esprit notre exigence principale : l'IA doit pouvoir être interrogée par l'utilisateur et lui répondre de manière intelligible. Les questions devront être posées selon un format standard. Les réponses seront données sous forme de phrases affichées sur l'écran du cockpit. Notre client réalisant des ventes à l'international, notre IA devra pouvoir comprendre et répondre aux questions posées en langue anglaise afin d'être comprise par tous ses utilisateurs. Dans l'optique de diminuer la charge de travail du pilote, notre IA doit être simple d'utilisation. L'utilisation de phrases standardisées pour les questions et les réponses sera donc privilégiée.

Les contraintes du projet sont les suivantes :

- Les langages utilisés pour le projet doivent être OWL pour la modélisation de l'ontologie et SPARQL pour les requêtes.
- La maquette doit pouvoir s'exécuter sous MacOS et Windows.
- Le but de l'ontologie étant de simplifier les conditions de pilotage, les réponses aux requêtes devront s'afficher immédiatement après la formulation de la question.
- Les réponses aux questions identifiées dans chaque scénario ne doivent pas conduire à une redondance de l'information face aux données déjà disponibles sur le tableau de bord.

Les hypothèses du projet sont :

- Nous nous plaçons ici dans le cadre de développement d'une IA pour un avion civil commercial. Bien que notre client commercialise des avions civils de type jet d'affaire, nous avons privilégié la modélisation d'un avion de ligne dont les données de vol et caractéristiques sont plus facilement accessibles.
- Nous supposons que notre ontologie a accès à toutes les informations nécessaires grâce aux capteurs ou aux systèmes embarqués de l'avion, notamment le *Flight Management System (FMS)*.

Les spécifications du projet sont les suivantes :

- Ontologie
 - Le code de l'ontologie sera réalisé en OWL 2.0.0 via l'éditeur Protégé.
- Maquette
 - La maquette sera fonctionnelle sur n'importe quel ordinateur fonctionnant sous Windows ou MacOS, disposant d'une installation Python 3.7 et ne bloquant pas les ports de connexion permettant l'installation de packages Python.
 - Le code source de la maquette sera réalisé en Python 3.7
 - La maquette pourra être téléchargée en intégralité via un dossier GitHub.
 - Un manuel d'utilisation de la maquette devra être accessible en page d'accueil du dossier GitHub d'hébergement de la maquette.
 - La maquette permettra de simuler de manière réaliste un vol d'avion civil d'une durée d'une heure. Toutes les données utilisées seront préalablement introduites dans l'algorithme et ne seront pas produites en temps réel.

- La maquette présentera une interface qui permettra à l'utilisateur de taper une question parmi une liste exhaustive d'une vingtaine de requêtes prédéfinies. Les questions devront être posées sous une forme standardisée.
- La maquette permettra à l'utilisateur de poser plusieurs questions au cours du vol simulé.

L'école attend également des résultats concernant la mise en oeuvre de ce projet, qui sont les suivants :

- Travailler et s'intégrer individuellement au sein d'une équipe pluridisciplinaire.
- Maitriser les outils de gestions de projet de manière à planifier et effectuer un suivi du projet pour le piloter avec rigueur.

2.4.3 Budget du Projet

Le budget global du projet pour chaque membre du groupe représente 80 heures, soit 320 heures au total. Ce volume horaire est réparti sur chaque tâche définie par le diagramme de Gantt. Le budget prévisionnel attribué à chaque tâche est décrit dans le tableau 2.1.

2.5 Suivi et Contrôle

2.5.1 Suivi de l'Avancement

Afin de suivre régulièrement l'avancement du projet, le temps de travail de chaque membre de l'équipe a été consigné dans un tableau. Nous avions ainsi un indicateur quantitatif nous permettant de respecter le budget initialement prévu. De plus, nous indiquions qualitativement l'état d'avancement de chaque tâche en cours. La combinaison de ces deux indicateurs nous a permis d'adapter en temps réel le budget alloué à chaque tâche. En effet, nous avons très vite constaté que les tâches du début du projet avaient été surestimées en terme de volume horaire tandis que la maquette nous aura pris plus de temps qu'initialement prévu. La figure 2.7 et le tableau 2.1 permettent de résumer le comparatif entre notre budget prévisionnel et les heures consommées. Plus particulièrement, la figure compare par mois les heures prévues et réalisées alors que le tableau résume par tâche.

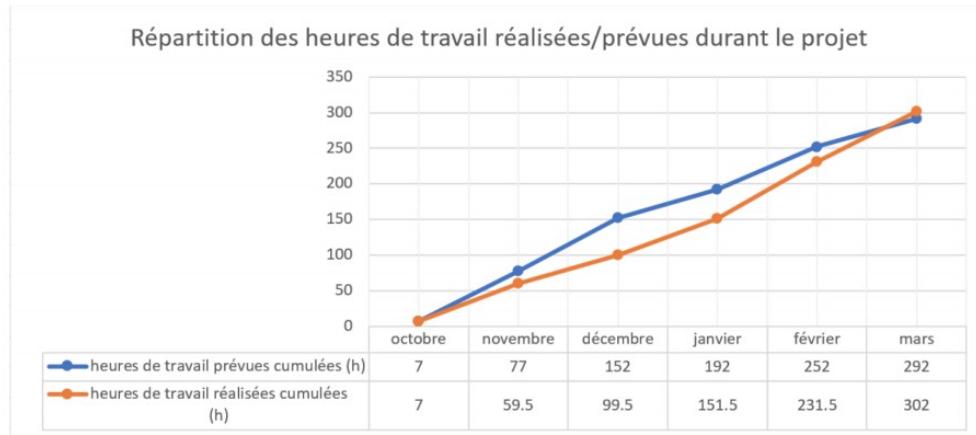


FIGURE 2.7 – Heures de travail réalisées et prévues pour le projet

Tâche	Budget Prévisionnel	Horaires Effectifs
Réalisation du Projet		
Compréhension des Attentes Client	16h	16h
Analyse de Manuels de Vol	10h	5h
Entretiens Pilotes	10h	5h
Prise en main du Langage OWL	10h	12h
Première Définition de l'Architecture	30h	10h
Implémentation de l'Ontologie en OWL	50h	53h
Identification des Questions Type	10h	15h
Prise en main du Langage SPARQL	20h	21h
Traduction des Questions en Requêtes SPARQL	20h	40h
Création de la Base de Données Test	20h	22h
Réalisation de l'Interface Graphique	0h	8h
Vérification des Réponses aux Requêtes	20h	4h
Rectifications du Livrable	0h	11h
Pilot Monitoring	0h	10h
Sous-Total	216h	232h
Gestion du Projet		
Rédaction du Plan de Développement	16h	20h
Mise à Jour du Plan de Développement	10h	10h
Rédaction du Mémoire	40h	50h
Préparation de la Soutenance	10h	-
Sous-Total	76h	70h
Total	292h	302h

TABLE 2.1 – Budget Prévisionnel et Effectif par Tâche

Au total, nous avons consommé 302 heures sans comptabiliser les heures de préparation de la soutenance ce qui surpasse notre budget prévisionnel. Néanmoins, ce total est bien plus cohérent avec le budget de 320h prévu par l'école ce qui est un bon indicateur du respect de la quantité de travail attendue sur l'année. La sur-estimation des premières tâches nous a permis de compenser le surplus de travail engendré par la maquette lors des deux derniers mois. En particulier, nous avons pu consacrer des ressources à la création d'une interface graphique pour l'utilisateur ce qui n'était pas initialement prévu. Nous avons également pu approfondir notre sujet en nous livrant à une étude d'un second cas

d'utilisation de notre maquette : le *pilot monitoring* que nous aurons l'occasion d'expliquer plus amplement au chapitre 4.

2.5.2 Maîtrise des Risques

Afin d'anticiper tout événement pouvant compromettre la réalisation du projet avant la date impartie, nous avons réalisé dès les premiers jours une analyse des risques encourus. Nous avons répertorié 13 risques répartis selon 4 catégories regroupés dans le tableau 2.8. Nous avons procédé à l'étude de ces risques selon deux axes : la probabilité de l'évènement et sa nuisance à la réalisation du projet.

Num.	Catégorie	Description			Évaluation pré-risque	
		Causes	Évenement à risque	Conséquence	Probabilité	Impact
1	client	Licenciement, conflit, décision de la hiérarchie	Perte de contact avec le client	Arrêt du projet	Faible	Grave
2	client	Redéfinition tardive des objectifs	Refonte totale du projet	Les livrables ne sont plus en adéquation avec les objectifs	Faible	Grave
3	humain	Conflit personnel au sein du groupe	Perte de cohésion	Perte de temps à gérer le conflit	Faible	Grave
4	humain	Démotivation d'un membre du groupe	Un membre ne travaille plus	Perte de capacité de travail et de temps	Faible	Moyen
5	organisation	Réponse tardive des pilotes	Identification tardive du besoin	Prise de retard sur la définition de l'architecture	Moyenne	Moyen
6	organisation	Mauvaise communication entre les deux équipes	Divergence des architectures entre les deux équipes	Prise du retard et possible conflit entre les deux groupes	Fort	Moyen
7	organisation	Mauvaise estimation du temps prévu pour une tâche	Prise de retard sur une tâche	Possible retard sur le rendu. Retard assuré sur tâche situé sur un des chemins critiques	Moyenne	Moyen
8	technique	Incompatibilité des langages SPARQL et OWL	Impossibilité d'implémenter les requêtes	Impossibilité de valider la modélisation par l'ontologie	Faible	Grave
9	technique	Manque de compétences en OWL	Impossibilité d'implémenter le code	Impossibilité d'implémenter l'ontologie	Moyenne	Grave
10	technique	Problème d'implémentation	La maquette ne fonctionne pas	Impossibilité de valider le fonctionnement de l'ontologie	Faible	Grave
11	technique	Manque d'informations, hypothèses fausses	Mauvais choix d'architecture	Modélisation inutilisable	Moyenne	Grave
12	technique	Mauvaise gestion du planning	Etre incapable de livrer à temps une maquette fonctionnelle	Echec du projet	Faible	Grave
13	technique/humain	Oubli de sauvegarder Bug informatique Perte ou vol de matériel	Perte d'une partie du code	Prise de retard sur la réalisation du livrable	Moyenne	Grave

FIGURE 2.8 – Liste des Risques

Par la suite, nous les avons représentés graphiquement sur une matrice des risques standardisée que l'on peut trouver figure 2.9. Celle-ci nous a permis de vérifier l'absence de risques critiques au projet (représentés par la zone rouge de la matrice). Les risques identifiés en orange sont alors à surveiller particulièrement.

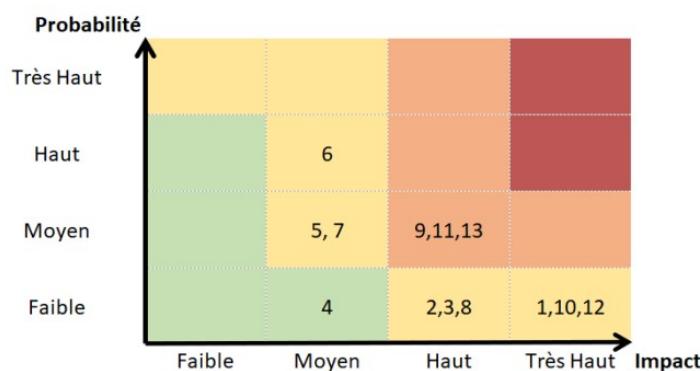


FIGURE 2.9 – Matrice des Risques

Pour conclure notre analyse, nous avons élaboré un ensemble de solutions préventives et curatives pour pallier ces risques regroupées au sein du tableau 2.10.

En pratique, l'ensemble de ces recommandations nous a été très utile. En effet, une grande partie de notre projet repose sur l'implémentation de la maquette qui allie 3 langages de code : OWL, SPARQL, et Python. Au cours de celui-ci, l'équipe code a donc dû s'approprier ces deux premiers langages dans un temps restreint. Ils ont ainsi dû faire face à un des risques identifiés : l'impossibilité d'implémenter les premières requêtes à cause d'un manque de compétences en SPARQL. Dès les premières difficultés, l'équipe a immédiatement mis en place les actions de la table des risques et a contacté plusieurs chercheurs du département informatique de l'ISAE-SUPAERO pour obtenir de l'aide. Ils ont également suivi plusieurs tutoriels en ligne pour les aider dans leur démarche. L'ensemble de ces actions leur a ainsi permis de surpasser cet obstacle très rapidement et de ne pas impacter le déroulement du projet.

Num.	Impact	Description du risque	Actions préventives	Actions curatives	Responsable de l'action
1	Grave	Perte de contact avec le client	Convenir du prochain créneau de réunion en face à face	Essayer de contacter une personne du même service où le client travaille afin d'en comprendre le raisons et ainsi de trouver une solution. En parler à Rob Vingerhoeds	Chef de projet
2	Grave	Refonte totale du projet	Trace écrite par e-mail récapitulatif après chaque réunion	Discussion avec le client pour limiter les changements. Informer François Debly et Rob Vingerhoeds de l'incident. Et probablement revoir les objectifs à la baisse.	Chef de projet
3	Grave	Perte de cohésion	Réunions de médiation Évenements de teambuilding	Essayer de discuter en dehors du PIE afin de comprendre et trouver une solution au problème	Chef de projet
4	Moyen	Un membre ne travaille plus	Réunions hebdomadaires sur l'avancée du travail de chacun	Comprendre les raisons et trouver une solution si possible. Aller voir Rob Vingerhoeds pour lui en parler et allouer des heures de travail en plus aux autres personnes du groupe	Chef de projet
5	Moyen	Identification tardive du besoin	Avoir assez de réponses de pilotes privés rapidement pour avoir une première idée du besoin	Se faire aider par le client	Équipe architecture
6	Moyen	Divergence des architectures entre les deux équipes	Travail en groupe dans la même salle et résumer du travail effectué à chaque séance	Réunion afin de connaître les différents points de divergence et discussion afin de trouver des compromis	Chef de projet
7	Moyen	Prise de retard sur une tâche	Granulariser les différentes tâches du Gantt afin de faire un suivi détaillé du volume horaire de chaque tâche	Réallocation des heures des tâches si possible. Sinon, nécessité de faire des heures supplémentaires	Chef de projet
8	Grave	Impossibilité d'implémenter les requêtes	Utilisation de Protégé pour s'assurer de la compatibilité tout au long du projet	Changer le langage de programmation et demander de l'aide	Équipe code
9	Grave	Impossibilité d'implémenter le code	Suivre des formations et tutoriels	Demander de l'aide à une personne compétente pour ce type de langage	Équipe code
10	Grave	La maquette ne fonctionne pas	Tests intermédiaires	Utiliser le versioning afin de retrouver une maquette qui fonctionne	Équipe code
11	Grave	Mauvais choix d'architecture	S'assurer que les informations disponibles sont valides	Allouer des heures équipe code au profit de la partie architecture afin de redéfinir rapidement une modélisation	Équipe architecture
12	Grave	Etre incapable de livrer à temps une maquette fonctionnelle	Suivre les indicateurs du projet afin d'anticiper un éventuel retard	Utiliser le versioning afin de rendre une maquette moins complète mais fonctionnelle	Chef de projet
13	Grave	Perte d'une partie du code	Sauvegardes régulières Utilisation d'un logiciel de versioning type GitHub	Allouer des heures équipe architecture au profit de la partie code afin de réécrire le code perdu	Équipe code

FIGURE 2.10 – Listes de Actions Préventives et Curatives

Chapitre 3

Travail Réalisé

3.1 Scénarii

Afin de concevoir la maquette, nous avons restreint notre étude et effectué des choix sur l'environnement à simuler. Tout d'abord, nous avons émis l'hypothèse que notre avion est un Airbus de la famille A320 pour des raisons de simplicité en terme de documents techniques accessibles sur Internet.



FIGURE 3.1 – Avion de référence utilisé pour le projet - Airbus A320 neo
source : https://pl.wikipedia.org/wiki/Plik:Airbus_A320neo_first_takeoff_at_Toulouse_Bagnac_Airport_05.jpg

Dans un second temps, nous avons établi des limites d'utilisation de notre maquette. Pour répondre aux attentes client, nous avons réfléchi à une vingtaine de questions qu'un équipage pourrait se poser durant un vol. Pour mener à bien cette étape dans le projet et pour cerner au mieux les attentes des équipages, le groupe a contacté plusieurs pilotes dont un pilote de ligne expérimenté, un cadet Air France et des pilotes privés. Au cours d'entretiens individuels, chacun nous a fait part des cas d'utilisation qu'il envisagerait pour notre système ainsi que quelques questions qu'il pourrait être amené à poser. Cette phase d'entretiens a été essentielle à la définition de notre projet et nous a permis d'identifier plusieurs cas d'utilisation de notre système. Rappelons que l'objectif du système est d'aider l'équipage à surmonter des situations stressantes en lui allégeant sa charge de travail. Ainsi, le système ne vise pas à en remplacer d'autres comme le FMS mais à simplifier certaines recherches qui peuvent nécessiter plusieurs secondes de concentration sur l'ordinateur de bord plutôt que sur le pilotage lui-même.

En considérant cet objectif principal et les entretiens des pilotes nous avons donc retenu 3 scenarii d'utilisation principaux que nous détaillons par la suite.

3.1.1 Navigation

La navigation est très souvent revenue dans les entretiens puisque c'est une des tâches qui mobilisent le plus l'équipage. Les informations qui intéressent principalement les pilotes sont celles qui sont difficilement atteignables en un lapse de temps assez court. Par exemple, obtenir le carburant restant, l'heure d'arrivée ou l'altitude prévue au prochain point de cheminement sont des requêtes envisageables pour la réalisation de notre système. Dans certaines phases d'urgence comme le déroutement, il paraît très utile de requérir le système afin qu'il donne rapidement un aéroport de dégagement. Le choix de l'aéroport est conditionné selon les paramètres de vol, des conditions météo et les caractéristiques de la piste et de l'appareil.

3.1.2 Conditions Météorologiques

Un autre intérêt pour les pilotes est de pouvoir connaître la météo en temps réel à certains endroits du vol. Ceci leur permet d'appréhender les risques potentiels. Un des risques majeurs liés aux conditions météorologiques est la formation de givre sur les hypersustentateurs ou les surfaces de contrôle. Un pilote pourrait donc demander à notre système s'il est conseillé d'activer l'anti-icing en fonction des conditions météo actuelles.

3.1.3 Performances Avions

Il est souvent difficile d'atteindre rapidement les paramètres de vol et les performances de l'avion en temps réel dans l'ordinateur de bord. Ces informations, très peu utilisées lors d'un vol, peuvent devenir essentielles lorsque l'équipage est soumis à un imprévu. Par exemple, sous certaines configurations de vol (volets, spoilers, becs), les différentes vitesses d'approche, d'atterrissement sont des informations ayant une utilité pour les pilotes. Obtenir l'altitude optimale en fonction de l'indice de coût via notre système est aussi un autre exemple exploitable par les pilotes.

3.2 Questions et Identification des Données

Grâce aux différents scénarios retenus, nous avons imaginé plusieurs types de questions ou requêtes qui pourraient être posées par un pilote durant un vol. Par exemple, la requête suivante, "*Give me the nearest airport I can land on*" est utile lorsqu'un équipage est en urgence absolue et doit se dérouter vers un aéroport adapté à son avion. Les données d'entrée associées à chaque requête, nous ont été très utiles pour la réalisation de l'architecture de l'avion, cette partie étant traitée dans la section suivante. De plus, nous supposons que les paramètres permettant de répondre aux requêtes sont fournis par les systèmes embarqués de l'avion. Le FMS notamment est riche en informations pour répondre aux différentes questions. Mais comme nous l'avons déjà exposé, il peut être très coûteux en temps et en concentration d'aller chercher certaines données. L'ensemble des questions retenues est présenté dans la table 3.1

3.2. Questions et Identification des Données

15

Question Index	Scenario	Question	Input Data	Source of the Data
1	NAVIGATION	Give me the approach checklist	Checklist text file	Local Storage
2	PERFORMANCES	Under this configuration of flight (spoilers, flaps, slats), what is the approaching speed to consider ?	Position of flaps, slats, instant weight	FMS
4	PERFORMANCES	Under this configuration of flight (spoilers, flaps, slats), what is the landing speed to consider ?	Position of flaps, slats, instant weight	FMS
4	PERFORMANCE	Under this configuration of flight (spoilers, flaps, slats), what is the landing distance ?	Position of spoilers, flaps, slats	FMS
5	WEATHER	Give me the wind orientation at the airport	Airport wind orientation	FMS
6	WEATHER	Give me the wind speed at the airport	Airport wind speed	FMS
7	WEATHER	Give me the weather at the airport	Airport weather	FMS
8	NAVIGATION	Give me the nearest airport	Aircraft coordinates	Navigation Database
			Airport coordinates	Navigation Database
9	NAVIGATION	Give me the nearest airport I can land on	Aircraft coordinates	Navigation Database
			Airport coordinates	Navigation Database
			Airport Firefighter Level (ARFF index)	Navigation Database
			Airport Runway Length	Navigation Database
			Airport Runway Width	Navigation Database
			Runway PCN	Navigation Database
			A/C maximum achievable distance	FMS
			A/C width	Aircraft characteristics (local storage)
			Minimum Landing Distance	FMS
			A/C ACN table	Aircraft characteristics (local storage)
			A/C ARFF index	Aircraft characteristics (local storage)
10	NAVIGATION	What is my current achievable range ?	Current achievable range	FMS
11	NAVIGATION	What is the ETA at the waypoint 'NAME_OF_WAYPOINT' ?	Expected time at waypoint	FMS
12	NAVIGATION	What is the expected altitude at the waypoint 'NAME_OF_WAYPOINT' ?	Expected altitude at waypoint FMS	FMS
13	NAVIGATION	What is the expected time of arrival ?	Expected time of arrival given by the FMS	FMS
14	WEATHER	Do I need to activate the anti-icing system ?	Icing probes	Icing system
15	WEATHER	Give me the weather at waypoint NAME_OF_WAYPOINT	Weather conditions	FMS (METAR + Radar)
16	WEATHER	Give me the weather at airport ICAO_CODE	Weather conditions	FMS (METAR + Radar)
17	PERFORMANCES	Give me the current fuel volume	Current fuel volume	FMS
18	PERFORMANCES	Give me the expected remaining fuel at waypoint NAME_OF_WAYPOINT	SFC, coordinates of A/C and waypoint, current fuel volume	FMS (METAR + Radar)
19	PERFORMANCES	Give me the expected remaining fuel at airport ICAO_CODE	SFC, coordinates of A/C and airport, current fuel volume	FMS
20	PERFORMANCES	What is my current optimal altitude ?	Zp,opt	FMS

TABLE 3.1 – Définition des différentes requêtes avec les données nécessaires et leurs sources

3.3 Architecture

Après avoir restreint le domaine d'utilisation de notre maquette grâce aux scenarii, nous avons élaboré l'architecture de l'ontologie. Notre ontologie doit donc contenir toutes les données précédemment identifiées grâce aux questions. Notre travail principal a consisté à regrouper ces données de la manière la plus pertinente possible afin de :

- Faciliter la retranscription de l'architecture en code OWL.
- Faciliter l'accès aux données de l'ontologie avec les requêtes SPARQL.

Nous avons d'abord séparé les données selon les trois thèmes de questions vus précédemment : Navigation, Performance et Météo.

3.3.1 Navigation

Au sein des données de navigation, qui regroupent les positions GPS ou les caractéristiques de pistes d'atterrissement, il nous a été tout de suite très facile d'organiser les données selon trois catégories internes :

- Les données Avion
- Les données Aéroport
- Les données Point de Cheminement

La catégorie **navigation** nous a permis de regrouper l'ensemble des données liées aux aéroports et aux points de cheminement. Nous avons également remarqué que l'ensemble des questions de météo portaient sur les conditions météorologiques à un aéroport ou à un point de cheminement. Nous avons donc décidé d'intégrer la catégorie Météo à la catégorie Navigation en créant une sous-catégorie *conditions météo* au sein des regroupements *airport* et *waypoint*.

En plus de la catégorie navigation, nous avons créé une catégorie **Documentation**. Cette catégorie a pour objectif de contenir tous les documents texte de référence comme les checklists.

3.3.2 Performances

Les données de performances en vol quant à elles ont été plus difficiles à organiser. En effet, les performances en vol pouvaient aussi bien être liées à des paramètres dynamiques (masse instantanée, rayon d'action spécifique, altitude optimale) qu'à des données de référence (table de vitesses d'approche en fonction des positions des hypersustentateurs). De plus, nous avons réalisé que l'implémentation des requêtes et de certains calculs associés ne pourrait se faire sans connaître la phase de vol dans laquelle évolue l'avion à un instant t . Nous avons donc imaginé 3 catégories liées aux calculs de performances en vol de l'avion :

- **Aircraft Systems** qui regroupe toutes les données liées aux systèmes de l'avion comme le dégivrage ou aux surfaces de contrôle
- **Flight Parameters** qui regroupe les données liées au pilotage de l'appareil comme l'altitude ou les vitesses. Cette catégorie est divisée en deux sous-catégories :

- *Paramètres de référence* qui correspondent aux objectifs d'altitude, vitesse de montée ou mach de croisière du plan de vol
- *Paramètres dynamiques* qui correspondent au paramètres de vol comme l'altitude, aux vitesses horizontales et verticales, etc ... et qui varient dans le temps avec la progression du vol

→ **Flight Phase** qui regroupe les dénominations des différentes phases d'un vol.

Finalement, nous avons créé une sixième et dernière catégorie, **Aircraft Characteristics** qui regroupe l'ensemble des données caractéristiques de l'appareil comme sa MTOW, sa longueur ou sa capacité de carburant et qui peuvent être utilisées pour répondre à toutes les questions.

Le travail de l'équipe architecture s'est conclu par la transmission de la représentation graphique de l'architecture présentée en annexe B à l'équipe code pour son implémentation en OWL.

3.4 Implémentation de l'Ontologie

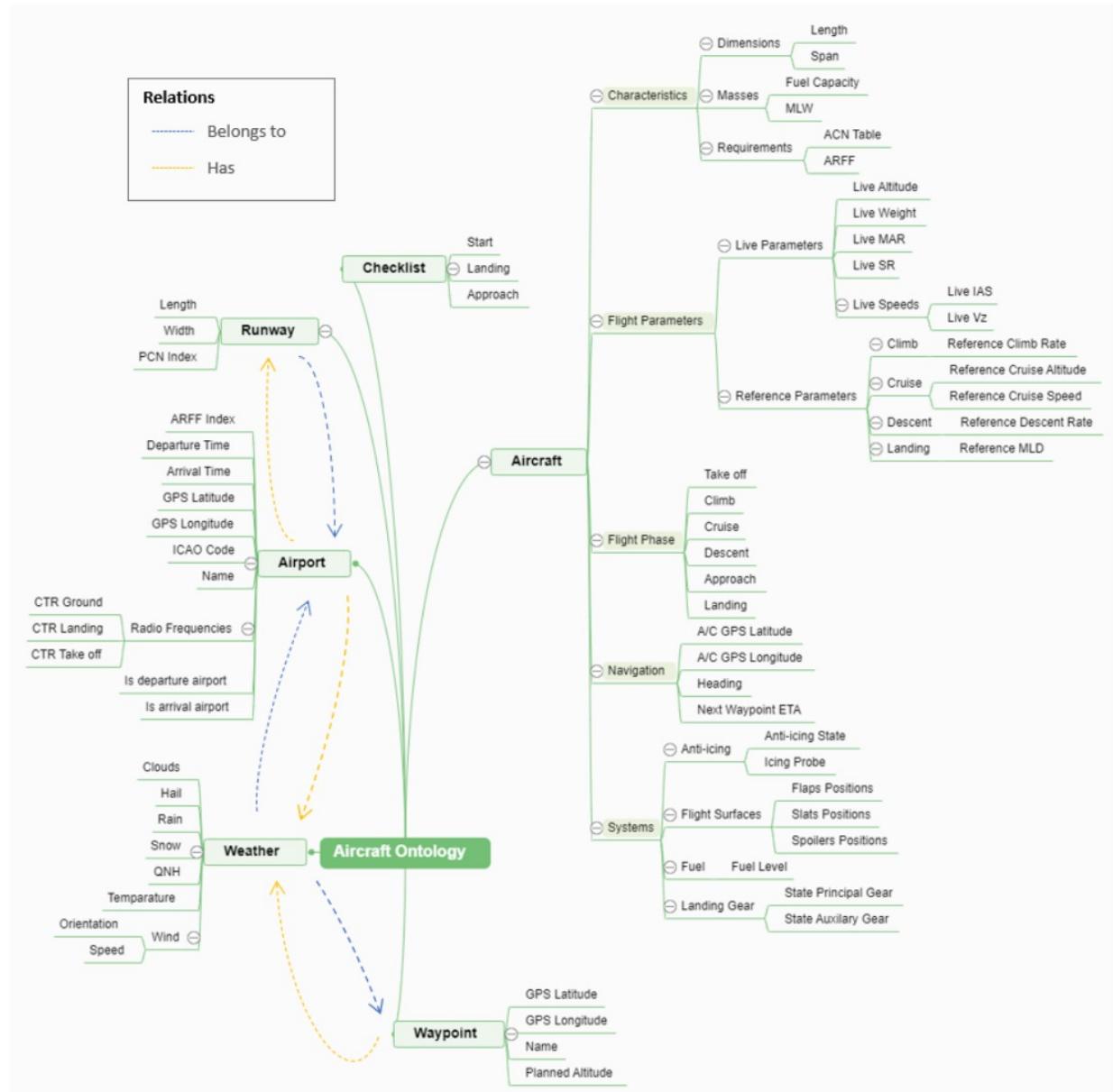


FIGURE 3.2 – Représentation Graphique de l'Ontologie

Nous implémentons l'architecture représentant le système avion sous forme d'une ontologie. Nous proposons ici de revenir sur les langages et technologies utilisés avant de décrire comment nous sommes passés de l'architecture à l'implémentation sous forme d'ontologie.

3.4.1 Les ontologies et le langage OWL

Une **ontologie** (2) est un ensemble structuré de concepts permettant de modéliser un champ d'informations. Elle permet de définir des concepts et catégories et de représenter les relations entre ceux-ci (dépendances, appartences, propriétés partagées par un groupe d'objets).

Les composants d'une ontologie sont les suivants :

Classes : Ensemble, collections, concepts ou types d'objets

Attributs : Propriétés, caractéristiques ou paramètres que les objets peuvent posséder

Relations : Liens que les objets peuvent avoir entre eux

Individus : Représentants d'une classe

Le langage OWL (*Web Ontology Language*) est une langue de représentation de connaissances qui permet de définir les ontologies.

3.4.2 L'éditeur *Protégé*

Protégé (6) est un éditeur gratuit et open-source pour les ontologies. Il permet de créer des ontologies de manière rapide et naturelle grâce à son interface graphique. L'extension *Protégé-OWL* (8) permet d'enregistrer l'ontologie en format OWL, pour être ensuite utilisée et requêtée.

3.4.3 Implémentation de l'architecture

Nous avons utilisé *Protégé* pour créer la structure et générer le code OWL de l'ontologie. Notre choix s'est porté sur cet éditeur pour plusieurs raisons : il est tout d'abord gratuit, bien documenté et sa prise en main est facilitée par la présence de tutoriels et d'une documentation étayée.

À partir de l'architecture du système avion, nous avons en premier lieu déterminé les classes, relations et attributs à implémenter.

Classes

Pour déterminer les classes à implémenter dans l'ontologie à partir de l'architecture, nous avons d'abord identifié les différents objets distincts de l'architecture : l'**appareil**, les **points de cheminement** et les **aéroports**. Ces objets sont distincts puisqu'aucun n'est une sous-partie d'un autre. Nous les implémenterons donc comme des **classes** dans l'ontologie.

Sont ensuite présents dans l'architecture d'autres objets qui sont cette fois des sous-parties

des classes identifiées plus haut : les **checklists**, **pistes d'atterrissages** et **bulletins météo**.

Les **checklists** sont des constituants de l'appareil : un avion en possède en général une dizaine. Dans notre cas, l'avion considéré en possède trois : une pour la phase d'approche, une pour l'atterrissement et une pour le démarrage de l'appareil. Il serait possible de les implémenter sous forme de trois attributs de la classe de l'appareil à l'aide de propriétés *HasApproachChecklist*, *HasLandingChecklist* et *HasStartingChecklist*. Nous avons fait le choix d'implémenter une **classe Checklist** qui contient ces trois attributs pour ne pas surcharger la classe représentant l'appareil et faciliter une éventuelle adaptation de l'ontologie à d'autres modèles d'avion, qui possèderaient potentiellement plus de checklists.

Les **pistes d'atterrissages** sont aussi implémentées sous formes de **classes**, la motivation de ce choix étant que le nombre de pistes peut varier en fonction de l'aéroport considéré. Il sera aisément de créer des individus, représentants de la classe Piste d'atterrissement, et de spécifier ensuite leur appartenance à un aéroport donné.

Enfin, les **bulletins météo** auraient aussi pu être implémentés sous forme d'attributs des classes Aéroport et Points de cheminement : il possèderaient dans ce cas des propriétés traduisant la présence ou non de pluie, de neige, la force et la direction du vent, etc. Cependant, il nous est apparu qu'une telle implémentation était redondante (les attributs relatifs à la météo de la classe aéroport se retrouvaient dans la classe points de cheminement), aussi nous avons trouvé préférable d'implémenter les bulletins météo sous forme de **classes**.

En conclusion, les classes retenues dans l'ontologies sont les suivantes :

- *Aircraft* : appareils
- *Airport* : aéroports
- *Checklist* : checklists
- *Runway* : pistes d'atterrissement et de décollage
- *Waypoint* : points de cheminement
- *Weather* : bulletins météo

Relations

Les relations permettent de définir les liens entre les classes précédemment citées. Nous avons implémenté les relations suivantes :

- *BelongsToAirport* : décrit l'appartenance d'une piste ou d'un bulletin météo à un aéroport
- *BelongsToWaypoint* : décrit l'appartenance d'un bulletin météo à un point de cheminement
- *HasWeather* : décrit la possession d'un bulletin météo ; s'applique aux aéroports et aux points de cheminement
- *HasRunway* : décrit la possession d'une piste ; s'applique aux aéroports

Certaines relations sont naturellement inverses : par exemple, si un aéroport *Airport1* possède la piste *Runway1*, alors *Runway1* appartiendra à l'aéroport *Airport1*. Ces propriétés entre relations peuvent être explicitées dans *Protégé*.

Attributs

Les attributs correspondent à toutes les feuilles des extrémités des branches de la représentation graphique de l'architecture du système. Pour plus de clarté, nous les regroupons selon les objets possédant les propriétés décrites par l'attribut. Nous proposons ci-dessous quelques exemples d'attributs de chaque classe, l'ensemble des attributs étant donné dans la représentation graphique de l'ontologie.

✈ Aircraft

- *AircraftLength* : longueur de l'appareil
- *AircraftSpan* : envergure de l'appareil
- *AircraftFlightPhase* : phase de vol

✈ Airport

- *AirportICAOCode* : code OACI de l'aéroport
- *AirportName* : nom commun de l'aéroport
- *AirportCTRGround* : fréquence radio au sol

✈ Checklist

L'object *checklist* a trois attributs : des chaînes de caractères donnant le chemin vers les trois checklists de l'appareil.

- *CheckListApproach* : checklist de la phase d'approche
- *CheckListLanding* : checklist de la phase d'atterrissage
- *CheckListStart* : checklist du démarrage

✈ Runway

- *RunwayPCN* : caractérisation du revêtement de la piste
- *RunwayWidth* : largeur de la piste
- *RunwayLength* : longueur de la piste

✈ Waypoint

- *WaypointGPSLatitude* : latitude du point de cheminement
- *WaypointName* : nom du point de cheminement
- *WaypointPlannedAltitude* : altitude prévue du point de cheminement

✈ Weather

- *WeatherClouds* : type des nuages présents
- *WeatherTemperature* : température
- *WeatherOrientation* : direction du vent

Individus

La structure de l'ontologie est entièrement déterminée avec les classes, attributs et relations. Nous pouvons créer des individus qui sont de représentants d'une classe : ils possèdent les propriétés propres à leur classe, qui peuvent alors prendre une valeur donnée.

Par exemple, un individu *Airport1* aura un code OACI spécifique renseigné dans l'attribut *AirportICAOCode*.

Les individus sont utilisés dans la maquette finale pour tester l'implémentation de l'ontologie. Ils sont générés à partir des données de fichier texte grâce à un script python. Ce script Python charge l'ontologie initiale vide d'individus, lit les fichiers textes correspondant aux différentes classes et crée les individus correspondants, en leur associant les attributs et relations spécifiés dans ces mêmes fichiers texte. La maquette finale est décrite dans la partie 3.6.

3.5 Crédit d'un Jeu de Données Test

La finalité de notre projet est de pouvoir démontrer le fonctionnement de l'ontologie et des requêtes. Pour cela, il nous a été demandé de réaliser une maquette simulant un vol d'avion civil et grâce à laquelle il serait possible d'interroger l'ontologie comme un pilote et en temps réel. Pour ce faire, il nous a fallu créer une base de données permettant d'alimenter l'ontologie et de simuler un court vol en avion.

Nous avons donc décidé d'illustrer le fonctionnement de nos outils en simulant un vol d'une heure entre Toulouse et Paris en A320. Pour ce faire, nous avons procédé en 3 étapes :

- Définition du plan de vol
- Vol sur simulateur
- Post-traitement des données

3.5.1 Définition du Plan de Vol

Pour créer le plan de vol qui ressemble le plus à la réalité d'une navette entre Toulouse et Paris, nous avons décidé de simuler un vol au départ de Toulouse-Blagnac établi sous le code OACI, LFBO, à destination de Paris-Orly plus connu sous le code OACI, LFPO. Pour créer la route qui relie les deux aéroports, plusieurs sites internet nous permettent, soit de générer un plan de vol directement, soit de connaître les différentes routes aériennes et points de cheminement¹. Afin de simplifier ce processus, nous avons opté pour la première solution qui nous permet d'obtenir un plan de vol cohérent rapidement. Le site utilisé est le suivant <https://flightplandatabase.com/>. Le plan de vol latéral est affiché sur la figure 3.3.

1. <https://skyvector.com/>

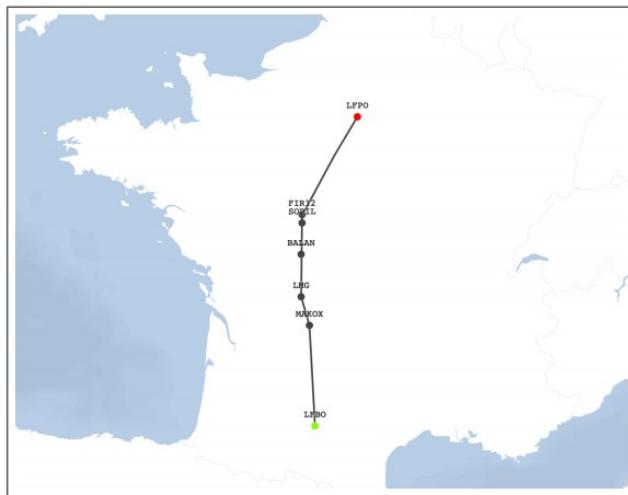


FIGURE 3.3 – Plan de vol au départ de LFBO à destination de LFPO

A propos du plan de vol vertical, le niveau de vol enregistré est le FL24. Ce niveau a été vérifié avec un vol de la compagnie Air France sur le site <https://www.flightradar24.com/> dans le but de se rapprocher au plus près de la réalité. La vitesse de l'avion en croisière est celle qui est procurée par les caractéristiques de l'avionneur. Ainsi, la vitesse nominale ou plus communément le Mach nominal en croisière est de 0.78.

Concernant les phases de décollage, de montée, d'approche et d'atterrissage, nous avons choisi de partir de la piste 14R et de se poser sur la piste 24. Ainsi, nous avons considéré les SID et STAR correspondant aux caractéristiques du vol : décollage vers le sud en direction du nord, ainsi qu'un atterrissage orienté vers l'ouest pour une approche par le sud.



FIGURE 3.4 – Plan de vol au départ de LFBO à destination de LFPO avec les STAR et SID

Le plan de vol complet est en annexe C de ce rapport.

3.5.2 Vol sur Simulateur

De manière à maîtriser tous les paramètres de vol, nous avons privilégié l'utilisation d'un simulateur plutôt que d'un site fournissant les données provenant d'ADS-B pour un vol LFBO-LFPO. Le simulateur choisi est FlightGear², simulateur gratuit, avec un large choix d'avions dont l'A320neo d'Airbus. Lors de la simulation, le plan a été soigneusement enregistré dans le FMC de l'avion pour activer le pilote automatique durant le vol. Ensuite, toutes les données de vol ont été sauvegardées dans un fichier .csv afin de les traiter et de les utiliser pour tester l'ontologie.



FIGURE 3.5 – Modèle A320neo sur FlightGear durant le vol LFBO-LFPO

Ainsi, en ayant obtenu les paramètres enregistrés par FlightGear nous avons partiellement le jeu données indispensable au fonctionnement de la maquette. Les données répertoriée au sein de l'ontologie et enregistrées directement par FlightGear sont les suivantes :

2. <https://www.flightgear.org/>

Navigation		Flight Parameters	
Aircraft - GPS coordinates	✓	Aircraft - Climb Speed	✓
Aircraft - Heading	✓	Aircraft - Reference Cruise - FL	✓
Airports - Name, ICAO code		Aircraft - Reference Cruise - MORA	
Airports - Radio Frequencies		Aircraft - Reference Cruise - Z_{opt}	
Airports - Runways		Aircraft - Reference Cruise - IAS	✓
Airports - GPS coordinates		Aircraft - Reference Descent - Speed	✓
Airports - Estimated Departure Time		Aircraft - Landing - Minimal Landing Distance	
Airports - Estimated Arrival Time		Aircraft - Specific Range (SR)	✓
Airports - Parking Spot		Aircraft - Live Speed - IAS	✓
Airports - ARFF index ³		Aircraft - Live Speed - V_z	✓
Airports - Weather - Wind (orientation & speed)	✓	Aircraft - Live Altitude - Altitude	✓
Airports - Weather - QNH	✓	Aircraft - Masses - Current Weight	✓
Airports - Weather - Temperature	✓	Aircraft - Maximum Achievable Range (MAR)	
Airports - Weather - Clouds - Type & ceiling	✓	Aircraft Systems	
Waypoint - Name	✓	Aircraft - Fuel - Current Volume	✓
Waypoint - Estimated Arrival Time	✓	Aircraft - Anti-icing (status & probes)	
Waypoint - Planned Altitude	✓	Flight Surfaces - Flap positions	✓
Waypoint - GPS coordinates	✓	Flight Surfaces - Spoiler positions	✓
Waypoint - Weather - Wind (orientation & speed)	✓	Flight Surfaces - Slats positions	✓
Waypoint - Weather - QNH	✓	Landing Gear - Positions	✓
Waypoint - Weather - Temperature	✓		
Waypoint - Weather - Clouds - Type & ceiling	✓		
Aircraft Characteristics			
Aircraft - ARFF index			
Aircraft - ACN table			
Aircraft - Span			
Aircraft - Length			
Aircraft - Maximum Landing Weight			
Aircraft - Fuel Capacity			

TABLE 3.2 – Données extractibles des enregistrements FlightGear

Les autres données qui ne peuvent pas être extraites de FlightGear sont obtenues ou

3. ARFF : (*Aircraft Rescue and Firefighting*) Service de sauvetage et de lutte contre l'incendie des aéronefs.

calculées en utilisant d'autres ressources comme nous allons le décrire dans la section suivante. Par exemple, les caractéristiques des aéroports avoisinant le trajet de l'avion sont obtenues sur le site du Service de l'Information Aéronautique (SIA)(3).

3.5.3 Post-traitement des Données

Comme décrit précédemment, grâce aux données enregistrées depuis la simulation sur FlightGear, nous avons pu extraire des paramètres nécessaires au test de l'ontologie. Néanmoins, il a fallu élaborer un script Python afin de traiter les données et de les afficher de la manière désirée. Avec ce script, nous avons également pu calculer certaines données manquantes à partir des données FlightGear comme le poids instantané ou le rayon d'action spécifique. Les formules considérées sont présentées dans l'annexe E. Finalement, le post-traitement s'est conclu par le formatage des données nécessaires selon les besoins du script de l'équipe code chargée de la réalisation de la maquette.

3.6 Maquette Finale

La finalité technique de notre projet est la réalisation d'une maquette. Cette dernière doit permettre de mesurer l'étendu du travail réalisé durant l'année. Ainsi, cette maquette doit être utilisable facilement par un utilisateur pas ou peu formé, inclure l'ontologie modélisant l'avion, le jeu de données test, des requêtes sur l'ontologie, et une interface utilisateur.

Le travail sur la modélisation de l'avion et sur l'ontologie a été détaillé précédemment dans les parties 3.2 et 3.3. Le jeu de données test a été présenté en partie 3.4. On va ici se concentrer sur l'implantation des requêtes, sur l'interface utilisateur et finalement sur la réalisation d'une maquette utilisable facilement réunissant tous ces éléments.

3.6.1 Les requêtes

Les requêtes sont codées en language SparQL (9), très bien intégré aux ontologies en OWL. Cependant, le reste de la maquette étant réalisée en Python, nous utiliserons le package *rdflib* (11) afin de charger l'ontologie et d'exécuter les requêtes SparQL dans un script Python. Nous présentons en annexe un tableau contenant la requête SparQL correspondant au numéro de requête défini précédemment dans ce rapport et le mot-clé à taper dans l'interface afin d'exécuter cette requête.

Comme nous pouvons le voir dans le tableau, certaines requêtes utilisent uniquement du code Python, d'autres simplement une requête SparQL et enfin certaines utilisent un mélange des deux. Il est impossible de présenter succinctement ces requêtes, nous présentons donc la partie la plus importante de la requête. Le code source peut être trouvé dans les codes Python qui seront présentés dans la partie suivante.

Certaines requêtes nécessitent un paramètre. Par exemple, on peut demander la météo d'un aéroport en particulier. L'utilisateur pourra alors préciser l'aéroport cible à la fin de la requête comme présenté dans le tableau en annexe. En l'occurrence, il pourra écrire `weather airport XXXX` où `XXXX` correspond au code OACI de l'aéroport.

Pour faire pointer les mots-clés de l'utilisateur vers les bonnes requêtes, nous utilisons une disjonction des cas sous forme de chaîne de `if`. Nous isolons certains mots clés de l'entrée qui sont susceptibles d'être utilisés (par exemple "weather" ou "fuel") puis nous affinons avec un second mot-clé. Au vu du nombre de requêtes dans la maquette, deux `if` suffisent à isoler une requête unique.

3.6.2 L'interface utilisateur

L'objectif principal de l'interface utilisateur est de rendre simple et rapide la sélection et l'exécution des requêtes définies dans la sous-partie précédente, et de présenter très clairement le résultat de ces dernières. Le produit final doit en effet faire gagner du temps à l'équipage, il faut donc réaliser une interface la plus réactive possible. L'interface est réalisée et utilisable uniquement en Anglais, langue de l'aéronautique.

Présentation générale

Finalement, notre choix s'est porté sur une interface graphique réalisée en langage Python basée sur du texte. Au total, l'utilisateur peut appuyer sur 3 boutons différents :

Start simulation Ce bouton permet de lancer la lecture du jeu de test . Après pression, le bouton disparaît et un texte affichant la progression de la simulation apparaît. L'ontologie est mise à jour à chaque étape de progression dans la simulation. Cette progression est présentée sous forme de temps de vol écoulé sur la durée totale du vol.

Start questions Ce bouton permet de lancer l'interface pour exécuter des requêtes. Après pression, le *focus* est automatiquement placé sur une zone d'insertion de texte -c'est à dire qu'à partir de maintenant, il suffit de taper un mot-clé et d'appuyer sur la touche *Entrée* pour exécuter une requête, la souris n'est plus nécessaire- et un cadre pour afficher le résultat de la requête apparaît. On peut poser un nombre illimité de question durant toute la durée de la simulation.

Quit Ce bouton permet de fermer la fenêtre d'interface graphique.

Cette interface est facilement portable sur tous les appareils pouvant exécuter du code Python, c'est le cas de tous les ordinateurs et de la majorité des appareils mobiles (téléphone portable, tablette, ...).

Apparence

L'interface présente une apparence très simple afin de favoriser la lisibilité. Nous présentons ici quelques captures d'écran prises sur Linux lors de l'exécution de la maquette finale.

Premier écran qui apparaît, avec simplement 2 des 3 boutons précédemment décrits disponibles. Nous ne pouvons pas poser de questions tant que la simulation n'est pas lancée. Nous pouvons appuyer sur le bouton *Start simulation* pour démarrer une simulation. Une fois la simulation lancée, on voit que le texte décrivant la progression est apparu.

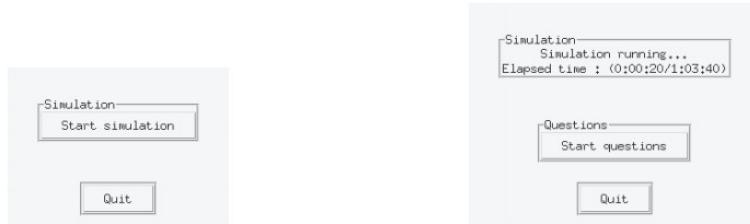


FIGURE 3.6 – Écran d'accueil puis apparition du bouton pour lancer l'interface de requêtage

Après pression sur le bouton pour lancer l'interface de requêtage (**Start questions**), le cadre pour taper les requêtes apparaît. On peut y taper directement sans utiliser la souris.



FIGURE 3.7 – Après avoir lancé l'interface de requêtage

Nous pouvons taper une requête dans le cadre, ici on demande par exemple l'aéroport le plus proche.



FIGURE 3.8 – Cadre d'entrée de texte

La réponse à la requête apparaît dans le cadre réservé sous le bouton *Submit*.



FIGURE 3.9 – Réponse à la requête

Nous voyons sur les captures d'écran la présence du cadre **Pilot Monitoring**. Ce cadre est décrit spécifiquement dans la partie 4 de ce rapport.

Requêteage

Comme décrit ci-dessus, l'exécution des requêtes se fait via la barre d'insertion de texte. Il suffit alors de taper un mot clé spécifique dans le tableau des requêtes de la partie ci-dessus et le paramètre correspondant à ce mot-clé si nécessaire. L'utilisateur peut alors appuyer soit sur la touche *Entrée* du clavier soit sur le bouton *Submit* de l'interface graphique pour exécuter la requête.

3.6.3 Réaliser une maquette simple et complète

Le défi principal de la maquette est l'intégration de tous les composants présentés jusqu'ici. C'est pourquoi nous avons choisi de coder toutes les différentes parties du projet dans le même langage de programmation : Python. Ainsi, la création et la mise à jour des entités de l'ontologie se fait à partir du jeu de test en utilisant le module Python owlready2 (7) pour manipuler l'ontologie. La lecture du jeu de test enregistré en format *csv* se fait grâce au module Python pandas (12). Le requêteage de l'ontologie est fait grâce au module Python rdflib, permettant d'exécuter des requêtes SparQL dans un script Python. L'interface graphique est réalisée grâce au module Python tkinter (5). Nous avons choisi ces modules car ils sont tous *open-source*, disponibles sur la dernière version de Python, toujours maintenus et sont les mieux documentés dans leurs domaines.

Nous allons voir ici la structure de la maquette finale et comment les différents composants communiquent entre eux. La structure globale est la suivante :

checklist Dossier avec les fichiers au format texte contenant les différentes checklists que l'utilisateur peut visualiser dans l'interface grâce à une requête.

csv Dossier contenant différents fichiers au format *csv* composant le jeu de données test. Nous le détaillons ci-dessous.

ontology Dossier contenant l'ontologie représentant l'architecture de l'avion, puis l'ontologie avec les entités créée à partir des fichiers *csv* et mise à jour au cours de la simulation.

python Dossier contenant les différents scripts Python.

readme.md Fichier Lisez-moi permettant de présenter le fonctionnement de la maquette à l'utilisateur.

Nous détaillons ici les deux dossiers les plus importantes de la structure de la maquette finale.

Dossier *csv*

Le dossier *csv* contient 7 fichiers, tous au format *csv*, contenant les données utilisées pour construire les entités de l'ontologie. Les 7 fichiers sont les suivants :

- *ACN.csv* : fichier permettant d'accéder à la classe ACN de l'avion en fonction de différents paramètres. C'est une table de correspondance entre des paramètres et la classe ACN.
- *aircraft.csv* : fichier contenant le jeu de test avec les données de l'avion. Chaque ligne du fichier correspond à un instant de la simulation. Certains paramètres vont donc évoluer avec le temps (par exemple la vitesse de l'avion) et d'autres sont statiques (par exemple vitesse de croisière de référence).
- *checklist.csv* : fichier contenant le chemin relatif de chaque fichier checklist (contenus dans la dossier checklist).
- *runway.csv* : fichier contenant les informations des différentes pistes des aéroports dans le jeu de test. Par exemple le numéro de la piste, sa longueur, ou encore l'objet météo associé (dont les informations sont enregistrées dans la fichier *weather.csv*).
- *waypoint.csv* : fichier contenant les informations des différents points de cheminement. Par exemple l'objet météo associé au point de cheminement ou encore le temps estimé d'arrivée à ce point de cheminement.
- *weather.csv* : fichier contenant les données météo pour les points de cheminement et les pistes.

Tous les fichiers ne seront lus qu'une fois puisque les données sont statiques hormis le fichier *aircraft.csv* où chaque ligne correspond à un instant de la simulation. Cela permet d'imiter un flux de données temps réel entrant comme si l'avion volait actuellement.

Dossier *python*

Le dossier *python* contient 3 fichiers permettant d'implémenter les différentes fonctionnalités de la maquette.

- *creating_entities.py* : Ce script contient les fonctions permettant de créer les différentes entités des objets de l'ontologie à partir des données contenues dans les fichiers *csv* en utilisant le module owlready2 et pandas. Une première fonction permet de créer toutes les entités et les propriétés correspondantes, une autre permet de mettre à jour l'ontologie en modifiant uniquement les informations ayant été modifiées (certains paramètres du fichier *aircraft.csv*). Des requêtes sur des propriétés de données sont également implémentées dans ce code afin de gagner en temps d'exécution pour certaines requêtes.

- *ontology_request.py* : Ce script contient les différentes requêtes écrites en SparQL appelées en Python sur l'ontologie grâce au module `rdflib`. Certaines requêtes utilisent des fonctions codées dans le premier script.
- *main.py* : Ce script réalise l'intégration des deux codes précédents et la création de l'interface graphique. Il synchronise la création et la mise à jour de l'ontologie, la progression de la simulation, le requêtage de l'ontologie et l'interaction de l'utilisateur avec l'interface graphique.

En pratique, l'utilisateur lance uniquement le script *main.py*, l'interface graphique apparaît alors et il peut alors interagir avec comme décrit précédemment. Le fichier `readme.md` permet à l'utilisateur de savoir quels aéroports et quels points de cheminement sont disponibles pour être requêtés.

3.7 Validation et Vérification

Après avoir implémenté l'ontologie, généré les données d'un vol test et finalisé les scripts Python permettant de créer les individus correspondant aux données, nous passons à la phase de vérification et de validation de la maquette.

3.7.1 Méthode de vérification

Nous devons vérifier que chacune des requêtes implémentées renvoie le bon résultat à chaque instant. Cependant, comme nous avons au total 190 pas de temps, nous faisons le choix de vérifier chaque requête à intervalles réguliers, dix fois lors d'une simulation complète. Il s'agit alors de reporter la réponse de chaque requête pour les temps suivants :

Indices	Temps
1	00 :20
2	06 :40
3	13 :00
4	19 :20
5	25 :40
6	32 :00
7	38 :20
8	44 :40
9	51 :00
10	57 :20

La simulation en temps réel durant plus d'une heure, nous modifions le code et fixons le temps entre deux indices à 1 seconde au lieu de 20 secondes.

D'autre part, nous calculons et reportons la réponse que devrait donner chacune des requêtes selon les données du vol pour chaque indice précisé plus haut. Ces réponses théoriques sont parfois immédiates à extraire des données (par exemple les bulletins météo ou les requêtes portant sur des paramètres directement accessibles dans les fichiers .csv) ; pour d'autres, il est nécessaire de mettre en place des routines de calcul. C'est notamment le cas des calculs de distances aux aéroports ou de vitesse d'atterrissage. Pour cela, nous avons utilisé Excel puisque les données, sous forme de fichiers .csv, y sont facile à manipuler.

Nous comparons enfin les réponses théoriques avec celles renvoyées par la maquette.

3.7.2 Erreurs mises en lumière lors de la phase de vérification

Si toutes les requêtes renvoyaient un résultat après la phase d'implémentation, la phase de vérification a permis de montrer que certaines ne renvoyaient pas un résultat *correct*. Par exemple, la fonction de calcul de distance contenait dans un premier temps des erreurs de conversions des angles : la requête portant sur l'aéroport le plus proche renvoyait cependant un résultat correct sur les premiers indices, mais renvoyait des erreurs plus tard dans la simulation. La vérification nous a permis d'identifier le problème et de le corriger.

Un autre problème touchait les données tests contenues dans les fichiers .csv : l'aéroport de Paris-Orly ne contenait aucune piste d'atterrissage à cause d'une faute de frappe qui attribuait les pistes à l'inexistant aéroport LFPP. La requête donnant l'aéroport le plus proche sur lequel l'appareil peut atterrir ne répondait donc jamais Paris-Orly, puisqu'en effet celui-ci ne répondait pas aux critères de longueur de piste. À nouveau, nous avons pu corriger ce problème.

3.7.3 Résultats et validation de la maquette

Les résultats théoriques et les réponses obtenues en requettant l'ontologie sont reportées dans un tableau Excel et comparés.

Après avoir effectué les quelques corrections mentionnées plus haut, nous constatons que chaque requête renvoie bien la réponse attendue, parfois à un arrondi près. **Nous validons donc la maquette.**

Chapitre 4

Travail annexé

Grâce aux entretiens pilotes que nous avons menés au début du projet, nous avons eu l'idée d'ajouter certaines fonctions à notre système. Ces fonctions vont au delà des exigences du client mais qui peuvent tout de même être profitables.

4.1 Pilote de Surveillance ou Pilot Monitoring

Dans un cockpit classique de vols commerciaux, deux pilotes doivent être présents. L'un, assis normalement sur le siège de gauche appelé le commandant de bord, et l'autre qui occupe normalement le siège de droite, le copilote.

Avant le début de chaque vol, le commandant de bord décide quel pilote sera directement responsable du pilotage de l'avion pour l'ensemble du vol ou pour des phases particulières de celui-ci, telles que la descente, l'approche et l'atterrissement, et il devient "*Pilot Flying*" ou "*Pilote aux Commandes*" pour ce vol ou la phase spécifiée de celui-ci. L'autre pilote est alors désigné par opposition comme le "*Pilot Monitoring*" (PM), "*Pilot Not Flying*" (PNF) ou en français "*Pilote de Surveillance*" et, dans ce rôle, il doit surveiller la gestion du vol et les actions de contrôle de l'aéronef du PF. Il doit également effectuer des tâches de soutien telles que les communications et la lecture des listes de vérification. Le manuel d'exploitation précise entièrement les rôles du PF et du PM/PNF, mais l'un des aspects les plus importants des tâches de tout PM/PNF est la vérification croisée des actions du PF. En effet, cette partie du rôle représente l'une des raisons les plus importantes pour lesquelles un équipage de conduite de deux pilotes est spécifié.

Ainsi, dans un futur hypothétique, il pourrait être fructueux, soit de remplacer à terme le PM par un système autonome, soit d'alléger les charges de travail de celui-ci, en ajoutant à notre système les fonctions de vérification croisée des actions du PF.

Nous avons implémenté les actions suivantes :

Scenario	Data	Data Source	Test	Alert
Monitoring Speed	Vz	Primary Flight Display	Vref - 500 < Vz < Vref + 500	CHECK SPEED
Monitoring Altitude	Vref	FMS	FL * 100 - 250 < Alt < FL * 100 + 250	CHECK ALTITUDE
Monitoring Altitude	Altitude	Altimeter	MORA < Alt	CHECK ALTITUDE
Monitoring Altitude	Reference Flight level	FMS		
Landing Gear Position	Altitude	Altimeter		
Landing Gear Position	MORA	FMS		
Landing Gear Position	Flight Phase		if flight phase == landing : landing gear position == 1	GEAR DOWN
Landing Gear Position	Landing Gear Position	Landing Gear System	if flight phase == climb : landing gear position == 0	GEAR UP
Landing Gear Position	Flight Phase			
Landing Gear Position	Landing Gear Position	Landing Gear System		

FIGURE 4.1 – Fonctions de PM implémentées dans notre système

L'implémentation d'une fonction *Pilot Monitoring* nous a paru particulièrement pertinente lors de l'étude de l'utilité de notre IA. En effet, les avionneurs imaginent de plus en plus l'avion civil du futur comme un avion avec un seul ou même sans aucun pilote. Etre capable de produire et de certifier ce type d'avion deviendrait alors un avantage de poids dans la stratégie commerciale de Dassault Aviation puisqu'ils auraient la capacité de vendre un avion dont les coups opérationnels seraient considérablement réduits. C'est en prenant en compte ces problématiques commerciales et la possibilité de déléguer les devoirs de surveillance du co-pilote, voire le pilotage complet de l'avion, à une IA que nous nous sommes rendus compte des enjeux sous-jacents de notre PIE.

Chapitre 5

Conclusion

L'ensemble du travail réalisé par notre équipe au cours de ces six mois s'est finalement concentré sur une tâche principale : la réalisation d'une interface d'assistance au pilotage d'avions civils lors de situations demandant une forte concentration des pilotes. La construction de cette interface au sein d'une maquette de démonstration aura demandé la réalisation de plusieurs tâches interdépendantes et résumées ci-dessous.

Nous avons commencé notre étude par une phase de définition des attentes client et des besoins auquel notre système devait répondre. Pour cela, nous avons mené une série d'entretiens avec des pilotes civils afin d'identifier différents cas d'utilisation de notre IA. Nous avons ensuite élaboré une liste d'une vingtaine de questions standard qu'un pilote pourrait être amené à poser à notre IA pendant un vol. La formulation de ces questions nous a alors menés à identifier les données nécessaires à la construction d'une réponse. Nous avons alors ordonné ces données à l'aide d'une ontologie pouvant être requêtée afin de récupérer un lot de données bien spécifique. Les questions identifiées précédemment ont alors été traduites en requêtes afin de pouvoir interroger l'ontologie. L'ensemble des travaux s'est conclu par la mise au point d'une maquette finale de démonstration permettant de simuler un court vol en A320neo entre Toulouse et Paris.

Le nombre de tâches à réaliser nous aura permis d'acquérir de nouvelles compétences en terme de planification, d'adaptation et de gestion des dépendances des travaux de chacun. En effet, notre projet nécessitait une forte communication entre l'équipe chargée de l'architecture et de l'identification des besoins et l'équipe code afin de ne pas prendre de retard sur les tâches accomm农田 à chacun. Notre chemin critique ne laissant pas beaucoup de marge, nous ne pouvions nous permettre de prendre du retard sur une tâche et ainsi impacter le travail du second groupe. Nous avons également dû gérer la sur-estimation des premières tâches en proposant une piste d'approfondissement du sujet avec l'aspect de *pilot monitoring*.

Cet approfondissement nous aura alors ouvert les yeux sur un cadre d'utilisation autre de notre IA. À terme, il est possible d'imaginer qu'une IA bien plus travaillée que la nôtre pourrait remplacer le co-pilote d'un avion civil. Une telle IA pourrait alors permettre à un avionneur comme Dassault Aviation de disposer d'un fort avantage stratégique et commercial. Il est alors possible d'imaginer une extension de notre PIE en proposant un nouveau sujet centré sur l'étude de l'impact technologique (ontologie, gestion des données à bord, machine learning, ...) et commercial des cockpit mono-pilote.

Annexe A

Gantt Chart détaillé

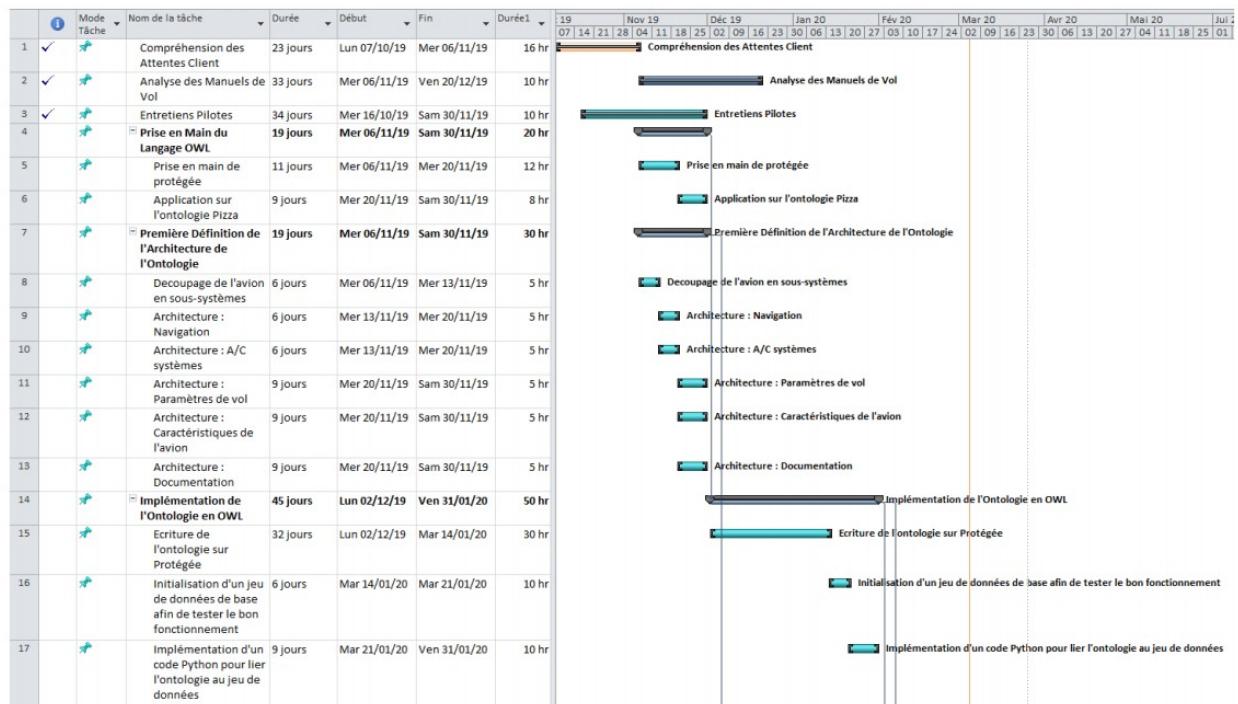


FIGURE A.1 – Gantt Chart détaillé - 1

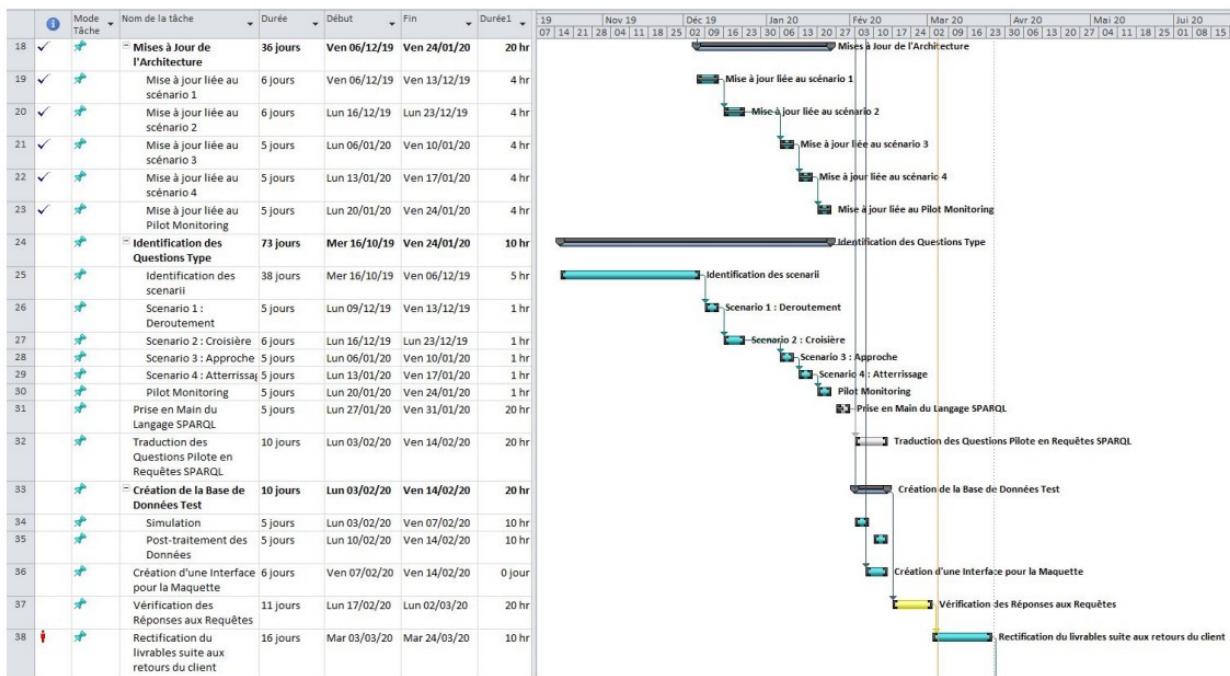


FIGURE A.2 – Gantt Chart détaillé - 2

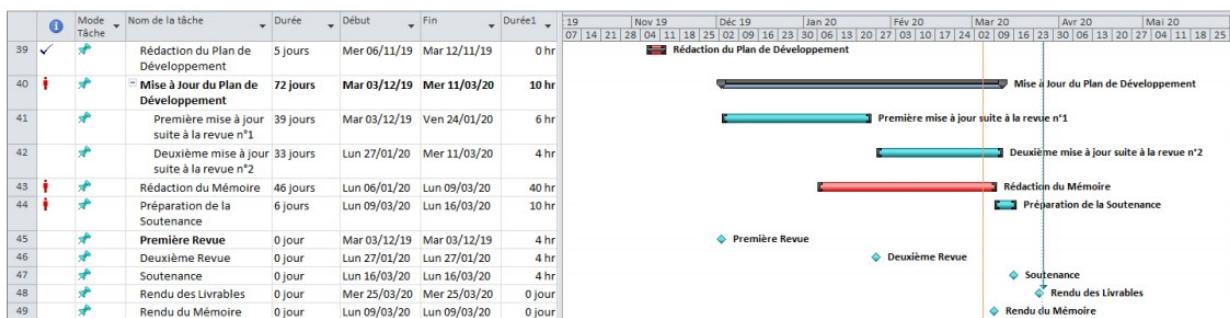


FIGURE A.3 – Gantt Chart détaillé - 3

Annexe B

Architecture Préliminaire de l'Ontologie

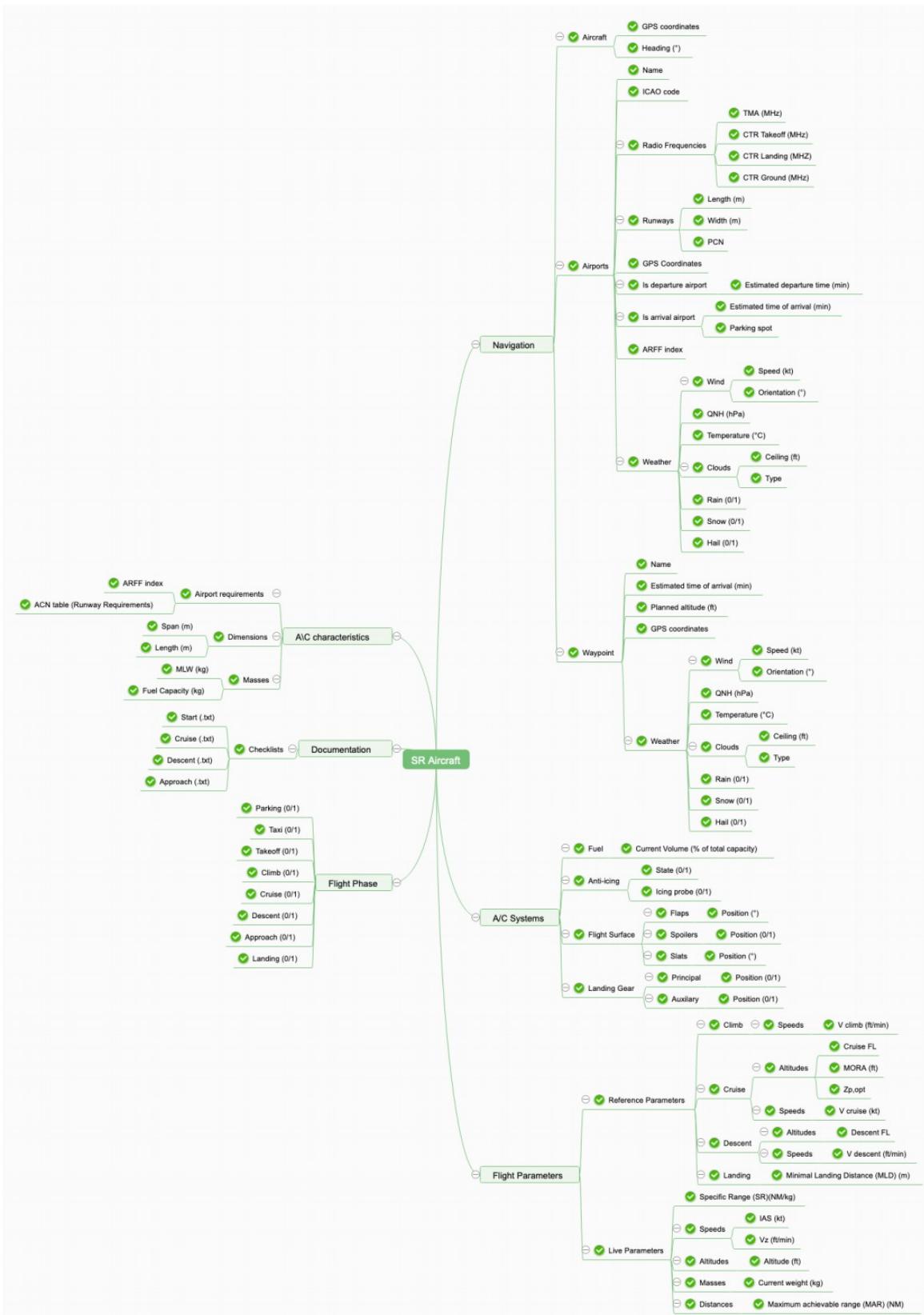


FIGURE B.1 – Architecture préliminaire de l'ontologie

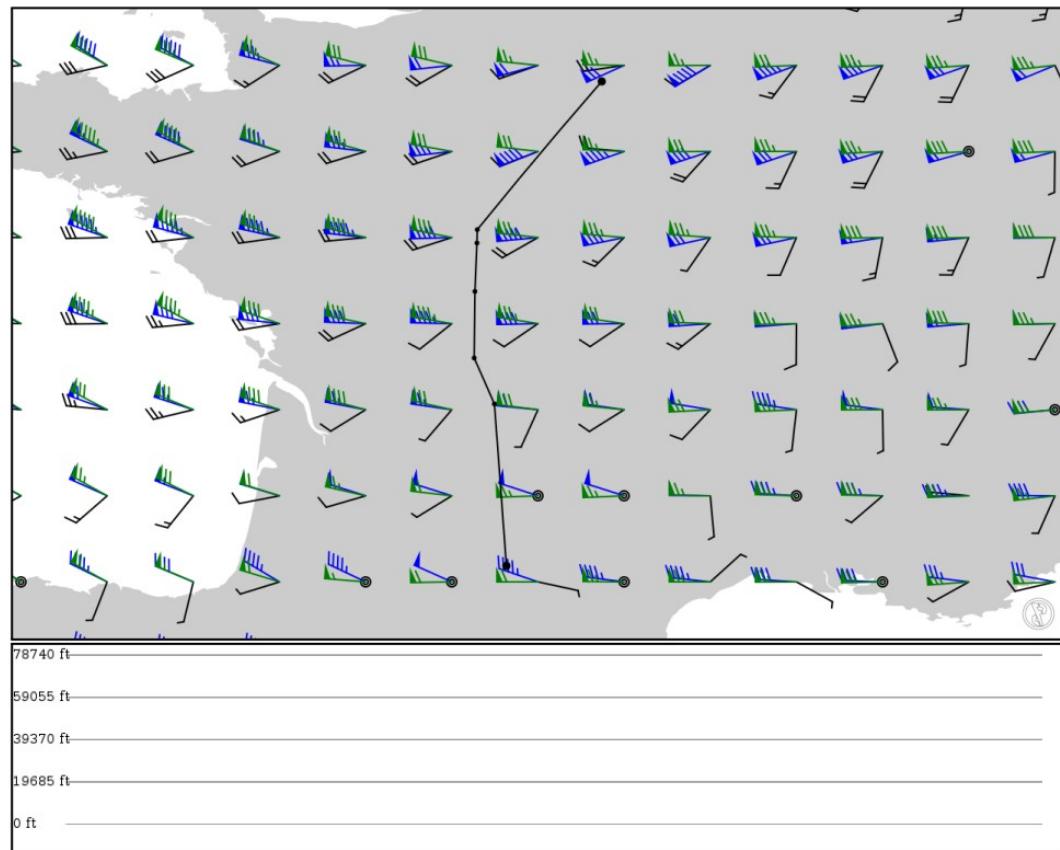
Annexe C

Plan de Vol Simulé

LFBO
Toulouse Blagnac

LFPO
Paris Orly

2020/02/13 1501Z
LFBO MAKOX G36 LMG A34 FIR12 LFPO
321.41 nm / 595.25 km



Notes

Basic altitude profile:

- Ascent Rate: 2500ft/min
- Ascent Speed: 250kts
- Cruise Altitude: 24000ft
- Cruise Speed: 420kts
- Descent Rate: 1500ft/min
- Descent Speed: 250kts

Options:

- Use NATs: yes
- Use PACOTS: yes
- Use low airways: yes
- Use high airways: yes

Route

Ident Type	Via	Lat Lon	Alt	Dist (nm)	Name
LFBO	-	43.62900 1.36397	0 ft 0 m	-	Toulouse Blagnac
MAKOK	-	45.33280	24,000 ft	102	-
FIX	-	1.23806	7,315 m		
LMG	G36	45.81590	24,000 ft	30	LIMOGES
VOR	AWY-LO	1.02558	7,315 m		
BALAN	A34	46.51610	24,000 ft	42	-
FIX	AWY-LO	1.03333	7,315 m		
SOPIL	A34	47.02580	24,000 ft	30	-
FIX	AWY-LO	1.05500	7,315 m		
FIR12	A34	47.16670	23,000 ft	8	-
FIX	AWY-LO	1.05833	7,010 m		
LFPO	-	48.72630 2.36701	0 ft 0 m	107	Paris Orly
APT	-				

LFBO

Region: FRANCE
 Timezone: EUROPE/PARIS
 Runways: 2

Elevation: 499 ft / 152 m
 Location: 43.629000 1.363970
 Magnetic Var: 0.834 E

METAR

LFBO 131430Z AUTO 28010KT 9999 FEW020/// SCT038/// BKN047/// //TCU 17/09 Q1018 TEMPO 28015G27KT BECMG NSC

TAF

TAF LFBO 131100Z 1312/1418 19005KT CAVOK BECMG 1312/1314 28012KT PROB30 TEMPO 1315/1317 28015G27KT PROB40 TEMPO 1

Frequencies

REC - 123.13 MHz - BLAGNAC ATIS	GND - 121.90 MHz - BLAGNAC GROUND
TWR - 118.10 MHz - BLAGNAC TOWER	APP - 121.10 MHz - BLAGNAC APPROACH
APP - 120.35 MHz - TOULOUSE APPROACH	APP - 123.85 MHz - TOULOUSE APPROACH
APP - 125.18 MHz - TOULOUSE APPROACH	APP - 129.30 MHz - TOULOUSE APPROACH
COM - 122.75 MHz - TOULOUSE BLAGNAC UNICOM	

Runways

Ident	Width	Length	Bearing (true) (mag)	Surface	Threshold Offset	Overrun Length
14R	148 ft 45 m	11,484 ft 3,500 m	142.99 142.16	ASPHALT	0 ft 0 m	295 ft 90 m
32L	148 ft 45 m	11,484 ft 3,500 m	323.01 322.18	ASPHALT	0 ft 0 m	194 ft 59 m
14L	148 ft 45 m	9,923 ft 3,025 m	143.00 142.17	ASPHALT	0 ft 0 m	30 ft 9 m
32R	148 ft 45 m	9,923 ft 3,025 m	323.02 322.18	ASPHALT	0 ft 0 m	0 ft 0 m

Approach Navaids

Runway	Type	Ident	Frequency	Range	Bearing (true) (mag)	Slope	Elevation
14L	DME	TG	108.90 MHz	18 nm 33 km	- -	-	528 ft 528 m
14R	DME	TBS	110.70 MHz	18 nm 33 km	- -	-	538 ft 538 m
32L	DME	TBN	109.30 MHz	18 nm 33 km	- -	-	535 ft 535 m
32R	DME	TD	108.35 MHz	18 nm 33 km	- -	-	554 ft 554 m
14L	LOC-ILS	TG	108.90 MHz	18 nm 33 km	143.02 142.18	-	494 ft 494 m
14R	LOC-ILS	TBS	110.70 MHz	18 nm 33 km	143.01 142.18	-	497 ft 497 m
32L	LOC-ILS	TBN	109.30 MHz	18 nm 33 km	322.99 322.15	-	487 ft 487 m
32R	LOC-ILS	TD	108.35 MHz	18 nm 33 km	322.70 321.86	-	486 ft 486 m
14L	GS	TG	108.90 MHz	10 nm 19 km	143.02 142.19	3.00	494 ft 494 m
14R	GS	TBS	110.70 MHz	10 nm 19 km	143.00 142.17	3.00	497 ft 497 m
32L	GS	TBN	109.30 MHz	10 nm	323.00	3.00	487 ft

Runway	Type	Ident	Frequency	Range	Bearing (true) (mag)	Slope	Elevation
32R	GS	TD	108.35 MHz	19 km 10 nm 19 km	322.17 323.02 322.19	3.50	487 m 486 ft 486 m

LFPO

Region: FRANCE
 Timezone: EUROPE/PARIS
 Runways: 3

Elevation: 291 ft / 89 m
 Location: 48.726300 2.367010
 Magnetic Var: 0.889 E

METAR

LFPO 131430Z 24023G35KT 200V260 9999 SCT040 SCT046 BKN086 11/04 Q1000 TEMPO 26030G50KT 2000 SHRA SCT020CB

TAF

TAF LFPO 131100Z 1312/1418 24018KT CAVOK TEMPO 1312/1314 24020G35KT SCT020TCU TEMPO 1314/1316 26030G50KT 2000 +SH

Frequencies

REC - 126.50 MHz - ATIS	REC - 131.35 MHz - ATIS
CLD - 121.05 MHz - PREFLIGHT	GND - 121.70 MHz -
GND - 121.82 MHz -	TWR - 118.70 MHz -
TWR - 120.50 MHz -	APP - 118.85 MHz -
APP - 123.87 MHz -	APP - 124.45 MHz -
DEP - 124.35 MHz -	DEP - 127.75 MHz -
DEP - 128.37 MHz -	

Runways

Ident	Width	Length	Bearing (true) (mag)	Surface	Threshold Offset	Overrun Length
06	148 ft 45 m	11,952 ft 3,643 m	61.78 60.89	ASPHALT	984 ft 300 m	197 ft 60 m
24	148 ft 45 m	11,952 ft 3,643 m	241.81 240.92	ASPHALT	0 ft 0 m	197 ft 60 m
08	148 ft 45 m	10,868 ft 3,313 m	74.36 73.47	CONCRETE	0 ft 0 m	1,050 ft 320 m
26	148 ft 45 m	10,868 ft 3,313 m	254.39 253.50	CONCRETE	1,427 ft 435 m	0 ft 0 m
02	197 ft 60 m	7,947 ft 2,422 m	18.31 17.42	CONCRETE	0 ft 0 m	482 ft 147 m
20	197 ft 60 m	7,947 ft 2,422 m	198.32 197.43	CONCRETE	0 ft 0 m	0 ft 0 m

Approach Navaids

Runway	Type	Ident	Frequency	Range	Bearing (true) (mag)	Slope	Elevation
06	DME	ORE	108.50 MHz	18 nm 33 km	- -	-	291 ft 291 m
24	DME	OLO	110.90 MHz	18 nm 33 km	- -	-	291 ft 291 m
26	DME	OLW	111.75 MHz	18 nm 33 km	- -	-	291 ft 291 m
02	LOC-ILS	OLN	110.30 MHz	18 nm 33 km	18.28 17.39	-	291 ft 291 m
06	LOC-ILS	ORE	108.50 MHz	18 nm 33 km	61.79 60.91	-	291 ft 291 m
24	LOC-ILS	OLO	110.90 MHz	18 nm 33 km	241.79 240.91	-	291 ft 291 m
26	LOC-ILS	OLW	111.75 MHz	18 nm 33 km	254.37 253.48	-	291 ft 291 m

Runway	Type	Ident	Frequency	Range	Bearing (true) (mag)	Slope	Elevation
08	LOC-LOC	OLE	108.15 MHz	18 nm 33 km	74.37 73.48	-	291 ft 291 m
02	GS	OLN	110.30 MHz	10 nm 19 km	18.59 17.70	3.00	319 ft 319 m
06	GS	ORE	108.50 MHz	10 nm 19 km	62.59 61.70	3.00	291 ft 291 m
24	GS	OLO	110.90 MHz	10 nm 19 km	242.59 241.70	3.20	291 ft 291 m
26	GS	OLW	111.75 MHz	10 nm 19 km	254.74 253.86	3.00	291 ft 291 m

Annexe D

Traduction des Requêtes Textuelles en Requêtes SparQL

Dans le jeu de données de test, les aéroports et les points de cheminement pouvant être requêtés sont les suivants :

Aéroports : LFPO, LFBD, LFBA, LFMT, LFBO.

Points de cheminement : SID_14R-3, SID_14R-6, SID_DEP-3, SID_DEP-4, SID_DEP-5, MAKOX, LMG, BALAN, SOPIL, FIR12, STAR_APP-5, STAR_APP-4, STAR_24-12, STAR_24-8, STAR_24-GS.

N°	Requête	Mot-clé interface
1	filename = onto.Checklist_0.CheckListApproach[0][1 :-1] with open(filename, 'r') as file : data = file.read().replace('\n', '') return data	approach checklist
2	eight = onto.DassaultJet.LiveCurrentWeight[-1]/1000 slats = onto.DassaultJet.AircraftSlatsPosition[-1] flaps = onto.DassaultJet.AircraftFlapsPosition[-1] increment = approach_speed_increment(slats,flaps) return 1.0125*weight+64.139+increment	approach speed
3	return nineteenth_request() - 5	landing speed
4	mld = onto.DassaultJet.ReferenceMLD[-1] slats = onto.DassaultJet.AircraftSlatsPosition[-1] flaps = onto.DassaultJet.AircraftFlapsPosition[-1] factor = landing_distance_factor(slats,flaps) return mld*factor	landing distance
5	SELECT ?winddir WHERE{ ?arrival_airport pie :AirportIsArrival 1 . ?weather_at_arrival pie :BelongsToAirport ?arrival_airport . ?weather_at_arrival pie :WeatherOrientation ?winddir .}	wind orientation at arrival
6	SELECT ?windspeed WHERE { ?arrival_airport pie :AirportIsArrival 1 . ?weather_at_arrival pie :BelongsToAirport ?arrival_airport . ?weather_at_arrival pie :WeatherSpeed ?windspeed .}	wind speed at arrival

TABLE D.1 – Requêtes

N°	Requête	Mot-clé interface
7	<pre>SELECT ?Ceiling ?Clouds ?QNH ?Hail ?Orientation ?Speed ?Temp ?Rain ?Snow WHERE { ?Airport pie :AirportIsArrival 1 . ?Airport pie :HasWeather ?Weather . ?Weather pie :WeatherCeiling ?Ceiling . ?Weather pie :WeatherCloudsType ?Clouds . ?Weather pie :WeatherQNH ?QNH . ?Weather pie :WeatherHail ?Hail . ?Weather pie :WeatherOrientation ?Orientation . ?Weather pie :WeatherSpeed ?Speed . ?Weather pie :WeatherTemperature ?Temp . ?Weather pie :WeatherRain ?Rain . ?Weather pie :WeatherSnow ?Snow .}</pre>	weather arrival
8	<pre>SELECT ?Name ?ICAO ?lat ?long WHERE { ?Airport pie :AirportICAOCode ?ICAO . ?Airport pie :AirportName ?Name . ?Airport pie :AirportGPSLongitude ?long . ?Airport pie :AirportGPSLatitude ?lat .}</pre>	nearest airport
9	<pre>SELECT ?Name ?ICAO ?lat ?long ?arff ?rwPCN ?runway_length ?runway_width WHERE { ?Airport pie :AirportICAOCode ?ICAO . ?Airport pie :HasRunway ?rw . ?rw pie :RunwayLength ?runway_length . ?rw pie :RunwayWidth ?runway_width . ?Airport pie :AirportName ?Name . ?Airport pie :AirportGPSLongitude ?long . ?Airport pie :AirportGPSLatitude ?lat . ?Airport pie :AirportARFFIndex ?arff . ?rw pie :RunwayLength ?runway_length . ?rw pie :RunwayPCN ?rwPCN .} ORDER BY DESC(?runway_length)</pre>	nearest airport land
10	<pre>FlightPhase = onto.DassaultJet.AircraftFlightPhase[-1] return [FlightPhase,onto.DassaultJet.LiveMAR[-1]]</pre>	MAR
11	<pre>onto = owl.get_ontology("../ontology/final-archi_entities.owl").load() return onto.DassaultJet.AircraftNextWaypointETA[-1]</pre>	waypoint eta XXXX
12	<pre>SELECT ?PlannedAlt WHERE { ?Waypoint pie :WaypointName ?WPname . ?Waypoint pie :WaypointPlannedAltitude ?PlannedAlt . FILTER regex(?WPname, """ + '"' + stest + '"' + """ , "i")}</pre>	altitude waypoint XXXX
13	<pre>SELECT ?ETA WHERE { ?Airport pie :AirportIsArrival 1 . ?Airport pie :AirportEstimatedTimeOfArrival ?ETA . }</pre>	arrival eta

TABLE D.2 – Requêtes

N°	Requête	Mot-clé interface
14	SELECT ?IcingProbe ?IcingState WHERE { pie :DassaultJet pie :AircraftIcingProbe ?IcingProbe . pie :DassaultJet pie :AircraftAntiIcingState ?IcingState .}	antiicing
15	SELECT ?Ceiling ?Clouds ?QNH ?Hail ?Orientation ?Speed ?Temp ?Rain ?Snow WHERE { ?Waypoint pie :WaypointName ?WPname . ?Waypoint pie :HasWeather ?Weather . ?Weather pie :WeatherCeiling ?Ceiling . ?Weather pie :WeatherCloudsType ?Clouds . ?Weather pie :WeatherQNH ?QNH . ?Weather pie :WeatherHail ?Hail . ?Weather pie :WeatherOrientation ?Orientation . ?Weather pie :WeatherSpeed ?Speed . ?Weather pie :WeatherTemperature ?Temp . ?Weather pie :WeatherRain ?Rain . ?Weather pie :WeatherSnow ?Snow . FILTER regex(?WPname, """ + '"' + stest + '"' + """", "i")}	weather way-point XXXX
16	SELECT ?Ceiling ?Clouds ?QNH ?Hail ?Orientation ?Speed ?Temp ?Rain ?Snow WHERE { ?Airport pie :AirportICAOCode ?ICAO . ?Airport pie :HasWeather ?Weather . ?Weather pie :WeatherCeiling ?Ceiling . ?Weather pie :WeatherCloudsType ?Clouds . ?Weather pie :WeatherQNH ?QNH . ?Weather pie :WeatherHail ?Hail . ?Weather pie :WeatherOrientation ?Orientation . ?Weather pie :WeatherSpeed ?Speed . ?Weather pie :WeatherTemperature ?Temp . ?Weather pie :WeatherRain ?Rain . ?Weather pie :WeatherSnow ?Snow . FILTER regex(?ICAO, """ + '"' + stest + '"' + """", "i")}	weather airpott XXXX
17	current = onto.DassaultJet.AircraftFuel[-1] total_capa = onto.DassaultJet.AircraftFuelCapacity[-1] return current/total_capa*100	fuel remaining
18	SELECT ?Lati ?Longi WHERE { ?Airport pie :AirportICAOCode ?ICAO . ?Airport pie :AirportGPSLatitude ?Lati . ?Airport pie :AirportGPSLongitude ?Longi . FILTER regex(?ICAO, """ + '"' + stest + '"' + """", "i")}	fuel airport XXXX
19	SELECT ?Lati ?Longi WHERE { ?Waypoint pie :WaypointName ?WPname . ?Waypoint pie :WaypointGPSLatitude ?Lati . ?Waypoint pie :WaypointGPSLongitude ?Longi . FILTER regex(?WPname, """ + '"' + stest + '"' + """", "i")}	fuel waypoint XXXX
20	weight = onto.DassaultJet.LiveCurrentWeight[-1]/1000 return optimal_altitude(weight)	optimal altitude

TABLE D.3 – Requêtes

Annexe E

Formules Utilisées pour le Post-traitement des Données

Rayon d'action spécifique :

Le rayon d'action spécifique est un paramètre indiquant la distance parcourue par un avion par unité de carburant consommée.

$$SR = \frac{TAS(kt)}{FF(kg/h)} \quad (\text{E.1})$$

Où TAS correspond à la vitesse vraie de l'avion et FF est la consommation de carburant instantanée de l'avion.

Calcul de la distance maximale atteignable :

Le paramètre distance maximale atteignable nous permet de connaître la distance maximale qu'un avion peut atteindre avec la quantité de carburant restante.

$$MAR = SR \times Fuel \quad (\text{E.2})$$

Où SR correspond au rayon d'action spécifique et $Fuel$ à la quantité de carburant restante.

Calcul de la vitesse indiquée (IAS) connaissant l'altitude et la vitesse vraie (TAS) :

Supposant que $\text{IAS} = \text{CAS} = \text{EAS}$, c'est à dire en ignorant les effets de positionnement, de calibrage et de compressibilité,

$$IAS = TAS \sqrt{\frac{\rho}{\rho_0}} = TAS \left(1 + 0.02 \frac{\text{Altitude}}{1000}\right) \quad (\text{E.3})$$

Approximation valable pour des altitudes suffisamment faibles pour rester dans les modèles linéaires de l'atmosphère.

Calcul de l'altitude de croisière optimale :

Pour obtenir l'altitude de croisière optimale, nous avons cherché dans un manuel de vol A320 et nous avons obtenu la courbe suivante à la vitesse nominale de croisière, Mach = 0.78.

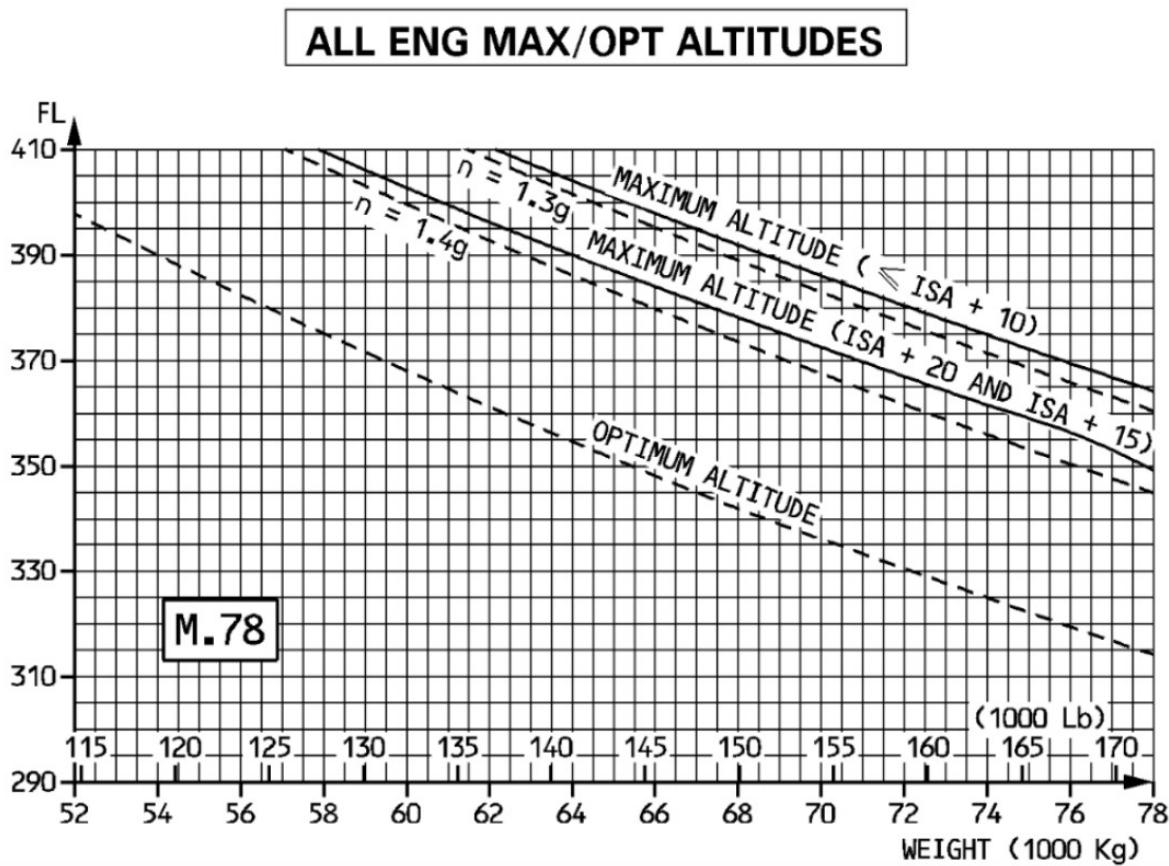


FIGURE E.1 – Courbe indiquant l'altitude optimale en fonction de la masse de l'avion(1)

Pour simplifier, nous avons linéarisé la courbe aux masses correspondant aux deux extrémités du vol, c'est à dire au décollage et l'atterrissage.

Vitesses d'approche pour différentes configurations d'hypersustentateur :
Pour connaître les vitesses d'approche pour les différentes configurations d'hypersustentateur en cas de pannes, nous avons utilisé les informations figurant dans un manuel de vol d'A320. La vitesse d'approche en configuration full est la suivante(1) :

$$V_{app_conf_full} = 1.025W(Weights(T))kt + 64,139kt + 5kt + \Delta V_{ref} \quad (\text{E.4})$$

Avec ΔV_{ref} l'incrémentation de vitesse à ajouter en fonction des configurations d'hypersustentateur données par la figure suivante.

A320 FAMILY	FAILURE		FLAPS LEVER POSITION FOR LDG	Δ VREF APPR SPD INCREMENT	MULTIPLY LDG DIST (CONF FULL) BY			
					DRY	WET (b)	CONTA (b)	
FLAPS/ SLATS	FLAPS and SLATS at zero			1	60 (APPR) 50 (THRESHOLD)	2.40*	2.10*	2.10*
	FLAPS < 1	S < 1	3	45	2.30*	2.00*	2.00*	
		S ≥ 1	3	25	1.95*	1.60*	1.60*	
	1 ≤ FLAPS < 2	S < 1	3	30	1.85*	1.70*	1.60*	
		S ≥ 1	3	15	1.50*	1.45*	1.35*	
	2 ≤ FLAPS < 3	S < 1	3	25	1.70*	1.55*	1.50*	
		S ≥ 1	3	10	1.40*	1.35*	1.25*	
	FLAPS = 3	S < 1	3	25	1.65*	1.55*	1.50*	
		1 ≤ S ≤ 3	3	10	1.35*	1.30*	1.25*	
		S > 3	3	5	1.30*	1.25*	1.20*	
	FLAPS > 3	S < 1	NOT ALLOWED					
		1 ≤ S ≤ 3	FULL	10	1.30*	1.30*	1.20*	
		S > 3	FULL	5	1.25*	1.25*	1.15*	

FIGURE E.2 – Vitesses d'approche en fonction de la configuration d'hypersustentateur(1)

Distance maximale d'atterrissage :

La distance maximale d'atterrissage, en fonction de plusieurs paramètres tels que la masse de l'avion, est donnée par la figure suivante, obtenue dans le manuel de vol :

HYDRAULIC SYSTEM											
The Reference Distance (REF DIST) considers : Sea Level (SL), ISA, no wind, no slope, no engine reverse thrust, manual landing ⁽¹⁾ , maximum manual braking, VAPP = VREF+ Δ VREF without APPR COR.											
Corrections on Landing Distance (m)				WGT ⁽²⁾	SPD	ALT	WIND	TEMP	SLOPE	REV	OVW
FAILURE	FLAPS LEVER for LDG	Δ VREF	REF DIST (m) for 66T	Per 1T above 66T	Per 5kt	Per 1000ft above SL	Per 5kt TW	Per 10°C above ISA	Per 1% Down Slope	Per Thrust Reverser Operative	If OVW PROC applied
G SYS LO PR	FULL	0	1 730	+ 60	+ 140	+ 90	+ 260	+ 80	+ 90	- 70	+ 550
	3	6	1 910	+ 60	+ 160	+ 100	+ 280	+ 100	+ 110	- 90	+ 660

(1) Automatic Landing correction: add 170m - (2) Weight correction: subtract 10m per 1T below 66T

FIGURE E.3 – Distance d'atterrissage en fonction des conditions de vol(1)

De plus, la distance d'atterrissage en fonction des configurations d'hypersustentateur

est délivrée par la précédente figure E.2.

Calcul de l'ACN :

L'ACN est un index caractérisant la capacité d'un avion à atterrir ou décoller d'une piste donnée et caractérisée par un index PCN. Pour le calculer, nous avons utilisé la formule suivante (4) :

$$ACN = ACN_{min} + (ACN_{max} - ACN_{min}) \frac{M - M_{min}}{M_{max} - M_{min}} \quad (\text{E.5})$$

Dans cette formule M_{min} correspond à la masse à vide opérationnelle de l'avion et M_{max} à sa MTOW. Les valeurs de ACN_{min} et ACN_{max} sont quant à elle renseignées dans une table ACN propre à chaque avion. Elles se lisent respectivement sur la deuxième et première ligne de la table ACN de l'A321 présentée figure E.4. Pour sélectionner les colonnes de lecture, il faut se référer au PCN de la piste sur laquelle on souhaite atterrir. Le PCN se présente sous le format suivant : 27/F/A/W/T. Seuls les 3 premiers caractères nous intéressent.

- 27 est le PCN à comparer à l'ACN calculé
- F fait référence au type de chaussée : souple (Flexible) ou rigide (Rigid)
- A classifie la résistance de la piste
 - A : Résistance élevée
 - B : Résistance ultra-faible

On choisit donc les colonnes avec les deux lettres correspondant au PCN.

Exemple : A321-100		CLASSES (catégorie de résistance du sol support)							
		Chaussées souples				Chaussées rigides			
	Masse de calcul	A	B	C	D	A	B	C	D
Masse maximale au roulage	83 400 kg	45	48	53	59	50	55	57	59
Masse à vide opérationnelle	47 000 kg	23	24	26	30	26	28	29	31

FIGURE E.4 – Tableau ACN A321

Pour finir, si on trouve $ACN < PCN$, l'avion peut atterrir à masse M sur la piste désignée.

Caractéristiques utiles de l'A320neo pour le calcul des paramètres précédents :

A320neo	
Length	37.57m
Wingspan	35.80m
MTOW	79t
MLW	66t
ARFF	6(10)
Max. Charge Utile	20t
Masse à Vide Opérationnelle	44.3t
Vitesse	Cruise : Mach 0.78 (450 kn ; 833 km/h) Max. : Mach 0.82 (473 kn ; 876 km/h)
Position Volets	0 ;10 ;15 ;20 ;35 (deg)
Position Becs	0 ;18 ;22 ;27 (deg)

TABLE E.1 – Caractéristiques de l'A320neo (13)

Bibliographie

- [1] Airbus. *A319/A320/A321 Flight Manual*, 2006.
- [2] Dassault Aviation. Langages gestion de connaissances.
- [3] DGAC-DSNA. *Service de l'Information de l'Aéronautique*. <https://www.sia.aviation-civile.gouv.fr/>, 2019. [Online ; accessed Feb-2020].
- [4] DGAC-STAC. *ACN/PCN*. <https://www.stac.aviation-civile.gouv.fr/fr/chaussees-aeronautiques/portance/acn-pcn>, 2017. [Online ; accessed Jan-2020].
- [5] Python Software Foundation. *Tkinter Documentation*. <https://docs.python.org/2/library/tkinter.html>, 2020. [Online ; accessed Feb-2020].
- [6] Matthew Horridge. *A Practical Guide To Building OWL Ontologies Using Protégé 4 and CO-ODE Tools Edition 1.3*, March 24, 2011.
- [7] Jean-Baptiste Lamy. *Owlready 0.2 Documentation*. <https://pythonhosted.org/Owlready/index.html>, 2015. [Online ; accessed Jan-2020].
- [8] *Protégé Desktop User Documentation*. <https://protegewiki.stanford.edu/wiki/Protege4UserDocs>. [Online ; accessed Nov-2019].
- [9] Eric Prud'hommeaux. *SparQL Documentation*. <https://www.w3.org/TR/rdf-sparql-query/>, 2008. [Online ; accessed Feb-2020].
- [10] Airbus SAS. *ICAO FAA Airfield Rescue and Fire Fighting Categories for Airbus Aircraft*. 18th of June 2019.
- [11] RDFlib team. *rdflib Documentation*. <https://rdflib.readthedocs.io/en/stable/apidocs/rdflib.html#id1>, 2012. [Online ; accessed Jan-2020].
- [12] Pandas Development Team. *Pandas Documentation*. <https://pandas.pydata.org/pandas-docs/stable/>, 2020. [Online ; accessed Nov-2019].
- [13] Wikipedia. *Airbus A320*. https://fr.wikipedia.org/wiki/Airbus_A320, 2020. [Online ; accessed Feb-2020].