UNIVERSITETOULOUSE II – LE MIRAIL
M2 ISMAG
5 allées Antonio Machado
31058 TOULOUSE

THALES AIR SYSTEMS
105 Avenue du Général Eisenhower,
31100 Toulouse, France

Automatisation du portage d'un logiciel de déploiement de système ATC.

Rapport du stage effectué du 14 Avril au 26 Septembre 2014

Destinataires :

Mme Sophie EBERSOLD

Mme Estelle GAZAGNES

Thomas BOULOTTE

Promotion 2013/2014

CONFIDENTIEL

Page laissée blanche intentionnellement

Remerciements

Je tiens à remercier dans un premier temps, toute l'équipe pédagogique du département Mathématique-Informatique de l'Université Toulouse II – Le Mirail pour l'organisation de la partie théorique de cette formation.

Je remercie également :

- Mme Estelle GAZAGNES (ma tutrice) pour m'avoir permis de faire ce stage. Ainsi que pour le temps qu'elle m'a consacré tout au long de cette période, sachant répondre à toutes mes interrogations. Et sans oublier sa participation au cheminement de ce rapport et ces encouragements.

- Mr Christophe AUDIGIER, Responsable du service intégration pour le suivi du stage et de ce rapport. Pour les différents conseils et les réunions de suivi de stage qu'il a organisées, et qui m'ont permis de mieux m'organiser durant ce stage.

- Mr Gilles Blanc pour m'avoir aidé dans la recherche d'informations pour la rédaction de ce rapport.

- Mr Jérôme ANTIGNY pour m'avoir aidé à la rédaction de ce rapport.

- Mme Céline MARTIN, Mr Nicolas PASQUON, Mr Philipe LIMOUZY et Mr Vincent BOUHELIER pour leurs conseils et leur aide durant ce stage.

- Mr Georges GRUDOV pour son accueil et ses conseils.

- Ma tutrice pédagogique Mme Sophie EBERSOLD pour avoir suivi le déroulement de mon stage.

Je tiens aussi à remercier l'ensemble des membres des équipes Intégration Coflight et 4-Flight pour leur accueil, les échanges tout au long de mon stage, ainsi que pour la bonne ambiance qui règne dans les bureaux.

Et pour finir je remercie tous les stagiaires que j'ai pu rencontrer durant mon stage pour leurs aides et leur bonne humeur.

Merci à tous pour avoir fait de stage une expérience enrichissante et agréable.

Page laissée blanche intentionnellement

Sommaire

Table des illustrations

Introduction

Ce rapport présente le travail effectué lors de mon stage au sein de l'entreprise Thales Air Systems à Toulouse. Ce stage, sur une période de six mois du 14 avril 2014 au 26 septembre 2014, a pour but de nous familiariser avec le monde du travail en proposant des missions techniques concrètes en rapport avec ma formation. Ces missions permettent de mettre en pratique les connaissances acquises durant cette année de formation et d'accroître nos compétences techniques mais aussi humaines.

Thales est un groupe de renommée mondiale, présent dans 56 pays avec un effectif total de plus de 67000 salariés. Thales est spécialisé dans l'aérospatial, la défense et les technologies de l'information et est leader mondial des systèmes d'informations critiques.

Ce stage aura consisté principalement à faire évoluer un outil de configuration et de déploiement d'un centre de contrôle aérien, appelé COCO. Il est utilisé par les projets Coflight eFDP, 4-Flight et SESAR. Mon deuxième objectif aura été de mettre en place une documentation sur cet outil afin de le rendre plus facilement maintenable et plus facile d'utilisation.

Dans ce rapport je vais dans premier temps faire une présentation de l'entreprise dans laquelle je présente le groupe Thales en général, Thales Air Systems et l'équipe d'intégration système logiciel que j'ai intégrée pendant mon stage. Je vais ensuite vous présenter le contexte d'utilisation de COCO. Puis, je vous présenterai le travail réalisé durant ce stage.

Et pour finir je vous présenterai un bilan de mon stage avec tout ce qu'il m'aura apporté et tout ce que j'en ai retenu.

I. Présentation de l'entreprise

I. 1. Le groupe THALES

Thales est un groupe de renommée mondiale spécialisé dans l'aérospatial, la défense et les technologies de l'information. Il est leader mondial des systèmes d'informations critiques.

Le groupe est présent dans 56 pays avec un effectif total de plus de 67 000 salariés.

Figure 1: Répartition des effectifs du groupe Thales dans le monde

Avec un chiffre d'affaire excédant 14 milliards d'euros et un bénéfice net de plus de 600 millions d'euros, Thales est un des leaders mondiaux des équipements à destination de l'aéronautique, de l'espace, de la défense, de la sécurité et des systèmes de transport.

I. 1. a) Histoire du groupe

Le Groupe Thales voit le jour en 1893, lorsque la Compagnie Française Thomson-Houston (CFTH) est créée afin d'exploiter en France les brevets de la société américaine Thomson-Houston Electric Company, dans le domaine de la production et du transport d'électricité. En 1918, la Compagnie Générale de Télégraphie Sans Fil (CSF) est fondée dans le but de développer les techniques de transmission hertzienne.

En 1968, la CFTH devenue Thomson-Brandt et la Compagnie Générale de Télégraphie Sans Fil (CSF) 2 fusionnent pour donner naissance à Thomson-CSF, le premier acteur majeur de l'électronique en France. Pendant les années 70, le groupe poursuit son expansion internationale et diversifie ses activités (commutation téléphonique, semi-conducteurs silicium, imagerie, médicale…).

Thomson-CSF est nationalisé en 1982. Au cours de la période dans le secteur public, le groupe recentre ses activités dans le domaine de l'électronique professionnelle et de défense. Il fait également l'acquisition de plusieurs sociétés dont les activités militaires du groupe Philips (1989) et la prise de contrôle de Sextant Avionique ce qui eut pour conséquence une forte croissance des filiales internationales de la fin des années 80 aux années 90. Le groupe est finalement privatisé en 1998.

Le groupe profite d'une coopération entre les sociétés Aerospatiale, Alcatel et Dassault et du regroupement au sein Alcatel Space des activités spatiales des sociétés Alcatel, Aérospatiale et Thomson-CSF pour accroître son périmètre d'activité. Enfin, Alcatel et Dassault Industries deviennent actionnaires de la société qui consolide ainsi ses positions.

En 2000, le rachat de la société britannique Racal Electronics permet au groupe de prendre une part croissante dans les marchés civils des technologies de l'information et des télécommunications mobiles. La société s'organise alors autour de trois pôles : la défense, l'aéronautique et les technologies de l'information et des services. Le groupe est alors renommé Thales (Thomson racAL Electronics Systems).

Dans le contexte géopolitique et économique bouleversé par les attentats du 11 septembre 2001, Thales renforce son engagement dans les segments les plus technologiques de la défense (guerre info-centrée, interopérabilité des forces armées…). Thales développe également ses activités de maître d'œuvre et de services afin de mieux répondre aux attentes des Etats confrontés à la complexité croissante et à la haute technicité des programmes militaires.

I. 1. b) Domaines d'activités

Le groupe Thales dessert cinq marchés principaux à la fois dans les domaines civils et militaires : l'aéronautique, l'espace, la sécurité, la défense et les transports terrestres.

Figure 2: Les domaines d'activité de Thales

Les activités menées par le groupe sont décomposées en plusieurs domaines et sous domaines.

Figure 3: Les activités menées par Thales

I. 2. THALES et ses divisions

Le groupe Thales menant des activités diversifiées, il se décompose en 31 filiales dont certaines sont nommées dans la Figure 4.

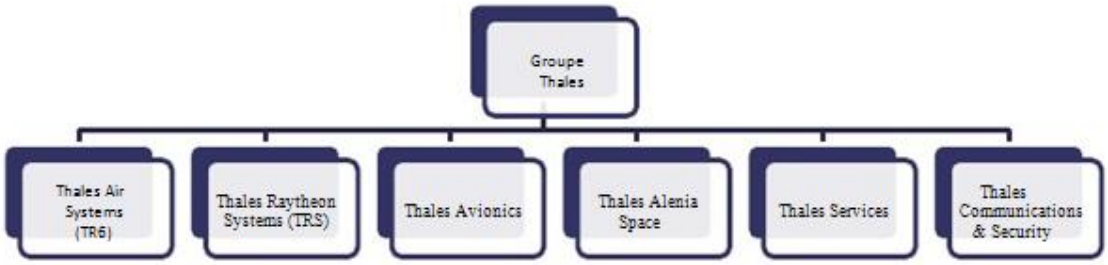


Figure 4: Les sociétés du groupe Thales

L'entité ATM dans laquelle j'ai été intégré appartenant à Thales Air Systems, il convient de s'attarder sur cette entité et évoquer succinctement les autres structures.

I. 2. a) THALES AIR SYSTEMS (TR6)

Thales Air Systems comporte trois entités :

I.2.a.1 AIR TRAFFIC MANAGEMENT (ATM)

Cette Division responsable de la gestion du trafic Aérien est implantée en Australie (Melbourne), Chine (Pékin), Allemagne (Stuttgart), Chessington en Angleterre, Arlington et Shawnee aux USA et Gorgonzola en Italie. Mais elle est principalement implantée en France (Rungis et Toulouse). Ses fonctions principales sont la réalisation de centres de contrôle, de systèmes de gestion de trafic aérien, de radars et de systèmes d'aide à la navigation. Elle propose également des solutions de gestion du trafic des aéroports au sol, en zone d'approche ainsi qu'un service de maintien des conditions opérationnelles.

Thales Air Traffic Management (ATM) Systems participe à plusieurs projets dont les principaux sont :

- COOPANS : Coopération industrielle entre Thales et les aviations civiles irlandaise (IAA), suédoise (LFV Group), danoise (Naviair), autrichienne (Astro Control) et croate (CCL). Dont l'objectif est de mutualiser les coûts liés au développement de nouvelles fonctionnalités des systèmes TopSky en Europe.

- LORADS III : Coopération industrielle entre les autorités de l'aviation civile de Singapour et Thales. C'est un projet portant sur la fourniture d'un nouveau système de contrôle du trafic aérien qui permettra d'assurer des niveaux de sécurité sans précédent et de promouvoir des approches plus respectueuses de l'environnement, en optimisant les itinéraires et en maximisant l'utilisation des niveaux de vol, avec à la clé, une réduction des coûts d'exploitation.

- Coflight : Système de gestion des plans de vol destiné à la France, l'Italie et la Suisse ; il remplacera les Flight Data Processing actuels, connus sous le nom de STPV (Système de Traitement de Plan de Vol) en France, et sera opérationnel pour les 20 ou 30 prochaines années.

- SESAR : Programme financé par EUROCONTROL et la Commission Européenne dont l'objectif est de définir un système de gestion du trafic aérien unique en Europe à l'horizon 2020.

- 4-Flight : Vise à développer une nouvelle génération de système ATM qui sera opérationnelle à horizon 2015-2020. Elle répondra aux exigences du règlement européen ciel unique. Elle permettra d'atteindre les objectifs définis dans le cadre du programme de R&D SESAR lancé par la Commission Européenne pour harmoniser l'ATM en Europe. Le système Coflight est un des composants de 4-Flight.

I.2.a.2 SURFACE RADAR (SRA)

C'est dans cette division implantée en France (Limours, Rouen, Fleury-les-Aubrais, Rungis) et aux Pays-Bas (Hengelo) que sont produits divers radars de tous types (radars de défense aérienne, de surveillance côtière et territoriale, de surveillance du trafic aérien etc.).

I.2.a.3 ADVANCED WEAPON SYSTEMS (ADW)

Située en France (Élancourt, Fleury, Rungis) et au Royaume-Uni (Basingstoke, Belfast), cette division est en charge des systèmes d'armes avancés de la Global Business Unit Land & Air Systems (GBU LAS).

I. 2. b) Autres filiales de THALES

I.2.b.1 THALES RAYTHEON SYSTEM (TRS)

Thales Raytheon Systems est le leader mondial des systèmes de commandement et de contrôle des opérations aériennes. Les produits réalisés sont des systèmes de commandement et de conduite des opérations aériennes ; radars de défense aérienne et de surveillance du champ de bataille.

I.2.b.2 THALES AVIONICS

Cette société se positionne en maître d'œuvre, architecte systèmes et fournisseur d'équipements et services dans le domaine des systèmes avioniques.
Thales Avionics est un partenaire industriel des avionneurs, des compagnies aériennes, des armées de l'air et des opérateurs, qu'ils soient civils ou militaires.

I.2.b.3 THALES ALENIA SPACE

Thales Alenia Space est un acteur majeur dans la construction de satellites en Europe et dans le domaine de l'infrastructure orbitale.
Il participe à de nombreux programmes comme la réalisation de satellites de Télécommunications, de télédétections, scientifiques, militaires et de navigation.

I.2.b.4 THALES SERVICE

Thales services est spécialisée dans les activités de sous-traitance informatique allant du conseil aux services informatiques.
Elle propose à ses entreprises clientes des solutions de simulation et d'entraînement, de l'infogérance et de la maintenance applicative, de l'informatique scientifique et technique, de l'informatique de gestion et de décision ainsi que des conseils sur la gestion de projet.

I.2.b.5 THALES COMMUNICATION & SECURITY

Thales Communications & Security est spécialisée dans les produits de systèmes d'information et de communications sécurisées pour les forces armées et de sécurité et dans les systèmes de sécurité urbaine, de protection des infrastructures critiques et des voyageurs.

II. Contexte général

Mon stage couvre trois projets qui sont en cours de réalisation. Le point commun entre ces trois projets est le projet Coflight qui est au cœur du système des deux autres projets.

II. 1. Coflight

A l'instar de TopSky dans les autres pays, Coflight vise à remplacer le système de traitement des plans de vol en France, en Italie et en Suisse. En effet Eurocontrol, l'organisme de gestion du trafic aérien en Europe, a défini plusieurs normes d'interopérabilité pour assurer une meilleure exploitation de l'espace aérien par les vols internationaux ; ceci dans une perspective d'uniformisation des procédures et des systèmes de gestion de l'espace aérien. Outre son interopérabilité à l'échelle européenne, ce système doit augmenter la capacité globale de contrôle de trafic aérien et offrir une maintenance améliorée et un haut niveau de fiabilité.

Coflight est destiné à accomplir les missions suivantes :

- Le service de traitement plan de vol pour le contrôle aérien civil et militaire
- La formation des contrôleurs et des superviseurs
- Les supports de test des nouvelles versions du logiciel ou de nouvelles fonctionnalités

Coflight permettra donc (en accord avec les exigences fixées par le programme de ciel unique européen et le programme SESAR) de traiter les opérations de contrôle aérien pendant les 20 prochaines années. En augmentant la capacité globale de contrôle de trafic aérien, tout en offrant une maintenance améliorée et un niveau de fiabilité très élevé.

Figure 5: Versions de Coflight

II. 1. a) L'organisation

L'importance des coûts d'un tel projet de remplacement du STPV (Système de Traitement Plan de Vol) en France et en Italie, ainsi que les besoins d'harmonisation européenne, ont conduit la DSNA (Direction des Services de la Navigation Aérienne) (France) et l'ENAV (Ente Nazionale di Assistenza al Volo) (Italie) à coopérer pour le développement d'un nouveau système répondant à ces missions.

Thales Air Systems détient 60% du projet, et Selex SI en détient 40%. Les deux ANSP (Air Navigation Service Provider) que sont la DSNA et l'ENAV ont été ensuite rejoints par le prestataire Suisse Skyguide. La Figure 6 résume les différentes parties prenantes dans le projet Coflight.

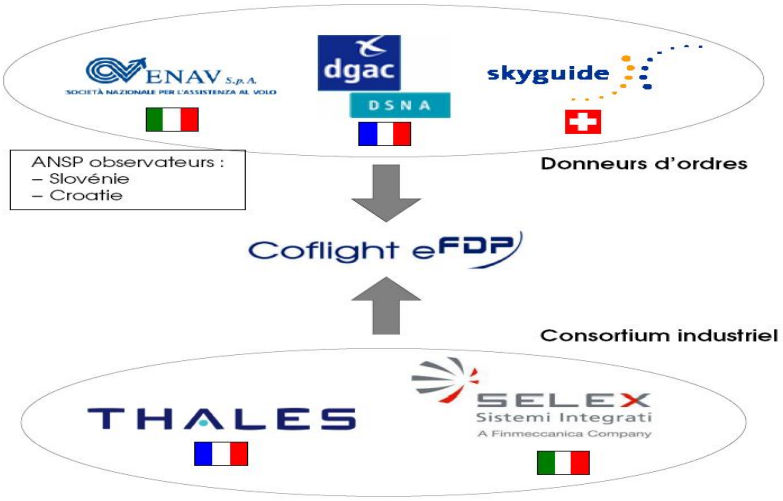


Figure 6: Parties prenantes dans Coflight

II. 2. 4-Flight

Le programme 4-Flight a été lancé en 2011 par la DSNA, le fournisseur français de services de navigation aérienne, qui a poursuivi cette collaboration après la coopération technologique réussie entre les deux parties dans le cadre du programme Coflight.

Ce programme consiste à remplacer le système de contrôle de trafic aérien actuel français par un système de nouvelle génération (**Voir annexe I**). Plus précisément, 4-Flight va interfacer les systèmes externes existants suivants :

- Surveillance et capteurs météo (radars SSR PSR, les radars Modes S, ADS-B, …),
- La communication avec les aéronefs,
- Centres de contrôle adjacents (civils ou militaires),
- Systèmes d'approche des aéroports,
- Les systèmes de Tour de contrôle des aéroports,
- Fournisseur des services Météo,
- Hyperviseur externe et panneau d'alarme,
- Système d'horloge externe (fournir la référence de temps pour le système),
- Etc.

II. 2. a) L'organisation

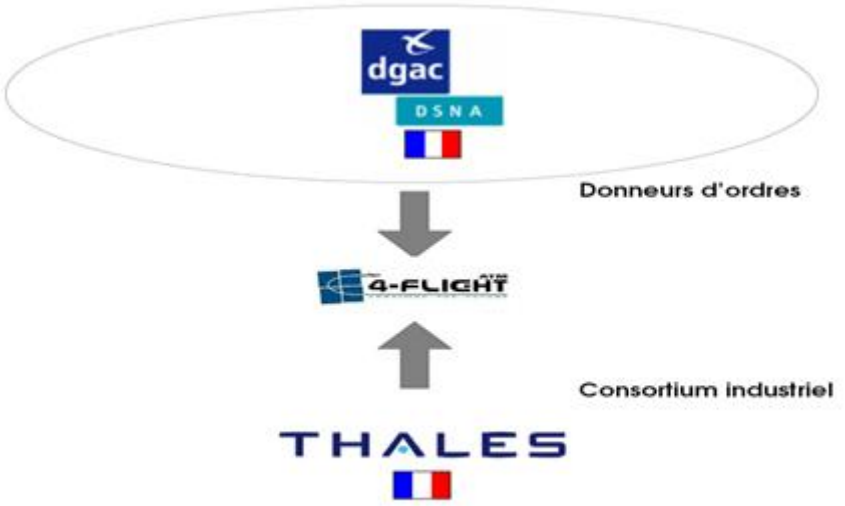


Figure 7: Parties prenantes dans 4-Flight

II. 3. SESAR

Contrairement aux Etats-Unis, l'Europe ne possède pas d'espace aérien civil unifié. C'est à dire un espace dans lequel la navigation aérienne est gérée au niveau européen. En outre, l'espace aérien européen est l'un des plus fréquentés au monde avec plus de 33 000 vols quotidiens, ce qui rend le contrôle du trafic aérien très complexe. Ainsi l'initiative SES (Single European Sky) a été élaborée pour surmonter la fragmentation de la gestion du contrôle aérien et la limitation des capacités de vols en organisant les services de l'espace et de la navigation aérienne sur un plan européen.

Pour développer la capacité technologique nécessaire, le programme SESAR (Single European Sky ATM Research) a été lancé en 2004. SESAR est un programme de R&D majeur pour développer le futur système ATM en Europe capable d'assurer la sécurité et la fluidité du transport aérien dans le monde entier au cours des 30 prochaines années. Il représente le pilier technologique du ciel unique européen. Il vise à développer une infrastructure de gestion du trafic aérien modernisée et de haute performance qui permettra le développement sûr, rentable et respectueux de l'environnement du transport aérien.

Les objectifs de SESAR proviennent du cadre des directives SES:

- Augmentation de la capacité;
- Augmentation de la sécurité par un facteur de 10;
- La réduction des effets de vols sur l'environnement de 10%;
- La réduction des coûts de 50%.

En Juin 2010, les autorités européennes et américaines ont conclu un accord préliminaire sur l'interopérabilité entre leurs futurs systèmes de gestion du trafic aérien.

II. 3. a) L'organisation

SESAR est un programme tellement grand qu'il est décomposé en 315 projets différents. Cela implique, comme le présente la figure ci-dessous, qu'un grand nombre d'industriel soit impliqué au sein de celui-ci.

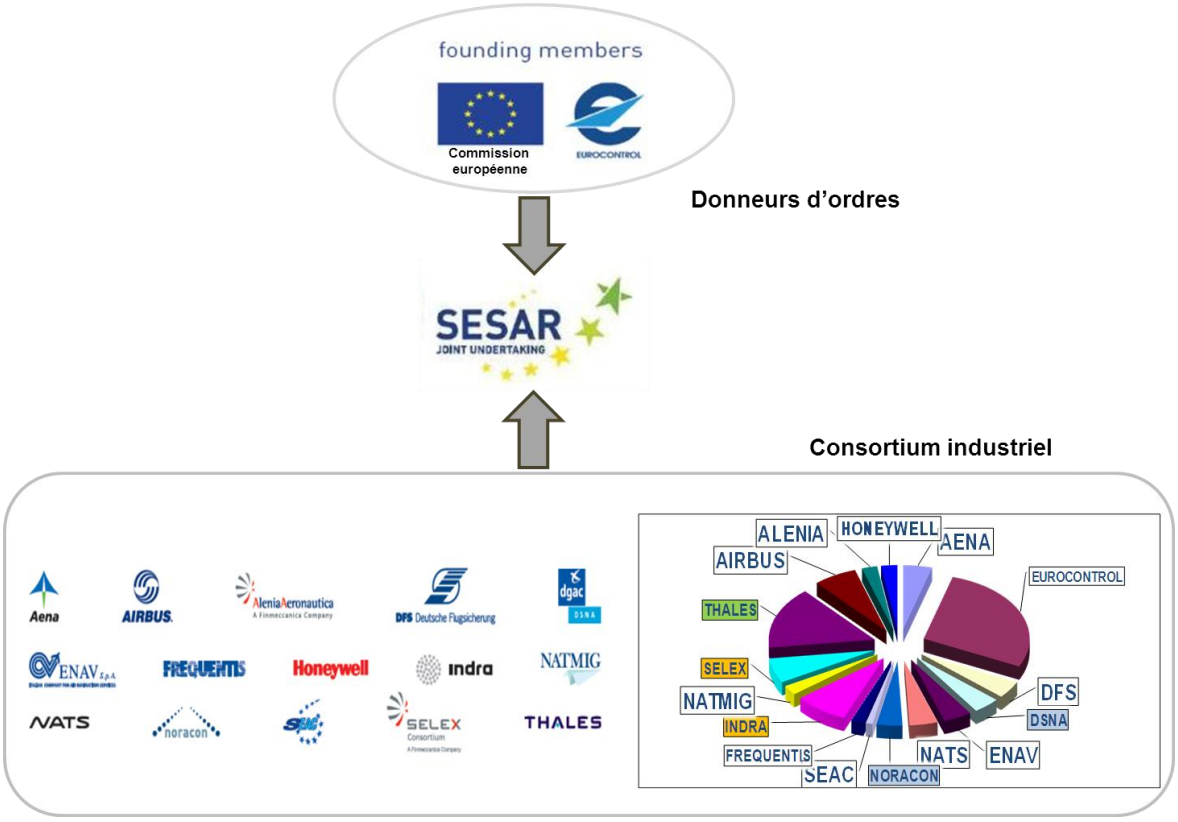


Figure 8 : Parties prenantes dans SESAR

II. 4. Présentation des équipes

Conformément à l'enchaînement des différentes étapes du cycle de développement en V, les équipes sur les différents projets sont déclinées de manière suivante :

- Equipe d'ingénierie systèmes : Située en amont du cycle de développement, cette équipe est chargée de l'élaboration de la spécification et de la conception du système
- Equipe safety : Cette équipe se charge de la sûreté de fonctionnement du système
- Equipe qualité : Cette équipe se charge de vérifier la qualité du système et sa robustesse
- Equipe Programme : Cette équipe s'occupe de tout ce qui est transverse au projet. C'est elle qui communique avec le client ou qui est en charge de la sauvegarde des données. On y retrouve d'ailleurs les deux équipes précédemment mentionnées.
- Equipe de développement : Cette équipe est chargée du développement des différents composants du système
- Equipe Intégration : C'est l'équipe au sein de laquelle j'ai effectué mon stage, et qui sera détaillée par la suite
- Equipe Validation : Cette équipe s'occupe de vérifier la conformité du système avec les spécifications avant sa livraison au client

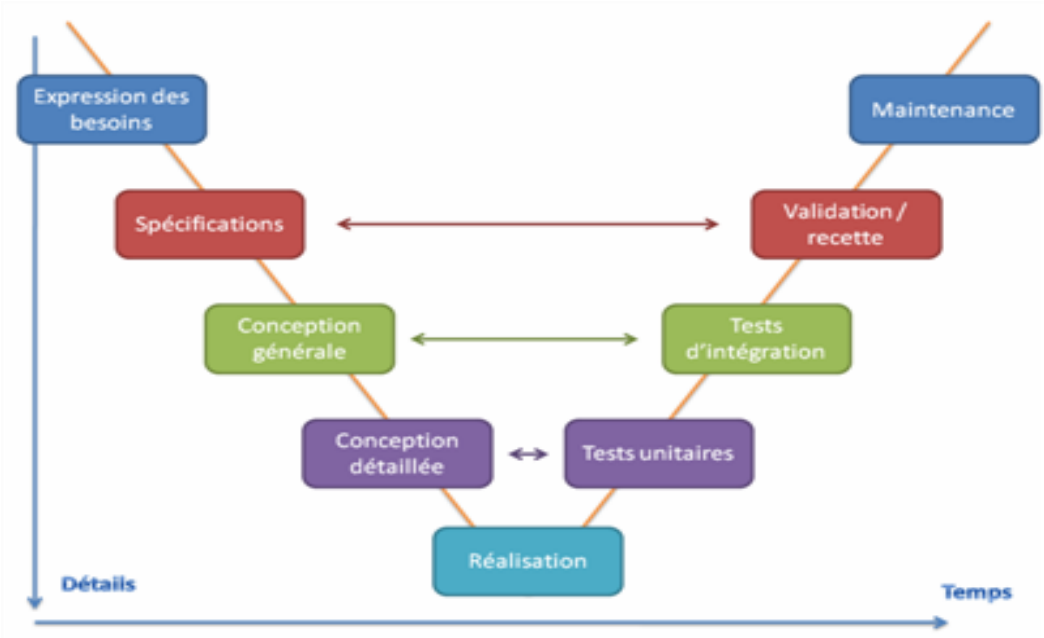


Figure 9 : Emplacement de la phase d'intégration dans le cycle en V

II. 4. a) L'équipe Intégration

L'équipe d'Intégration à Toulouse est multiprogramme et multi-activités. Elle est en charge des intégrations hardware et software ainsi que de la gestion de configuration système. Elle est constituée d'une vingtaine de personnes et la polyvalence de ces personnes est un atout majeur de celle-ci.

L'intégration système hardware est en charge du bon fonctionnement du matériel de tests pour les équipes systèmes (intégration, validation) mais aussi du matériel mis à la disposition du client lors de tests en usine. A Toulouse, ce parc de machines est de plus de 400 unités.

La gestion de configuration est en charge de la réception des livraisons venant des différentes équipes logicielles et de la construction de versions complètes pour les tests faits par les équipes systèmes (intégration, validation). La gestion de configuration système est aussi en charge des livraisons des logiciels aux clients.

L'intégration système logicielle reçoit les versions complètes de la gestion de configuration pour déployer l'ensemble du système et vérifier que tous les composants discutent correctement entre eux (**Voir Annexe II** : Présentation schématique des rôles de la pré-intégration et de l'intégration). Ensuite, la version complète vérifiée et donnée à l'équipe de validation pour faire des tests fonctionnels. L'intégration système démarre alors des tests de validation technique.

II. 4. b) Environnement matériel et logiciel

I.3.b.1 Environnement technique

Coflight et 4-Flight tournent sous Linux.

Pour ce stage j'ai utilisé des serveurs de tests Coflight et 4-Flight.Ce sont des machines HP ou DELL.

Techniquement, ces machines sont des quad-core avec16 ou 32Go de RAM et des disques de 144 Go jusqu'à 1 To.

I.3.2.2 Logiciels utilisés

Les postes de travail sont équipés du logiciel Windows XP et nous utilisons les logiciels bureautiques suivants :

- Microsoft Outlook afin de communiquer par mail entre les différentes personnes du groupe Thales ou de l'extérieur.
- MobaXterm afin d'effectuer des connexions en SSH sur les serveurs.
- Firefox pour accéder à Internet et l'Intranet.
- Microsoft Office 2010 pour tout ce qui concerne l'aspect bureautique.

III. Présentation du sujet

Après avoir présenté l'environnement du travail et le contexte général et technique, je vais maintenant présenter plus précisément l'objectif de mon stage.

Dans un premier temps, je présenterai le programme COCO, qui est le logiciel qu'on m'a chargé d'étudier et de modifier.

Puis je préciserai le travail que l'on m'a demandé d'effectuer durant toute la durée de ce stage.

Mon sujet de stage porte sur un programme appelé COCO (Cardamom Offline Configurator). Le logiciel est commun aux projets Coflight, 4-Flight et SESAR.

Dans ce chapitre, je vais vous présenter le Middleware CARDAMOM et le logiciel COCO qui permet de construire les fichiers de configuration pour déployer les logiciels à base de ce middleware.

III. 1. Présentation rapide de la base de Coflight (le modèle système)

Le modèle système sert de base de travail pour tous les développements du projet. Il définit tous les composants, toutes les interfaces (internes et externes) et toutes les données utilisées sur le projet. Ce modèle système est le cœur et la base des systèmes Coflight et 4Flight.

COCO se base sur ce modèle système comme tous les autres composants logiciels. Il utilise la liste de toutes les interfaces externes et internes. Il utilise également toutes les données liées au déploiement. Par exemple, le système Coflight est divisé en applications, puis en process, puis en composants comme ci-dessous :

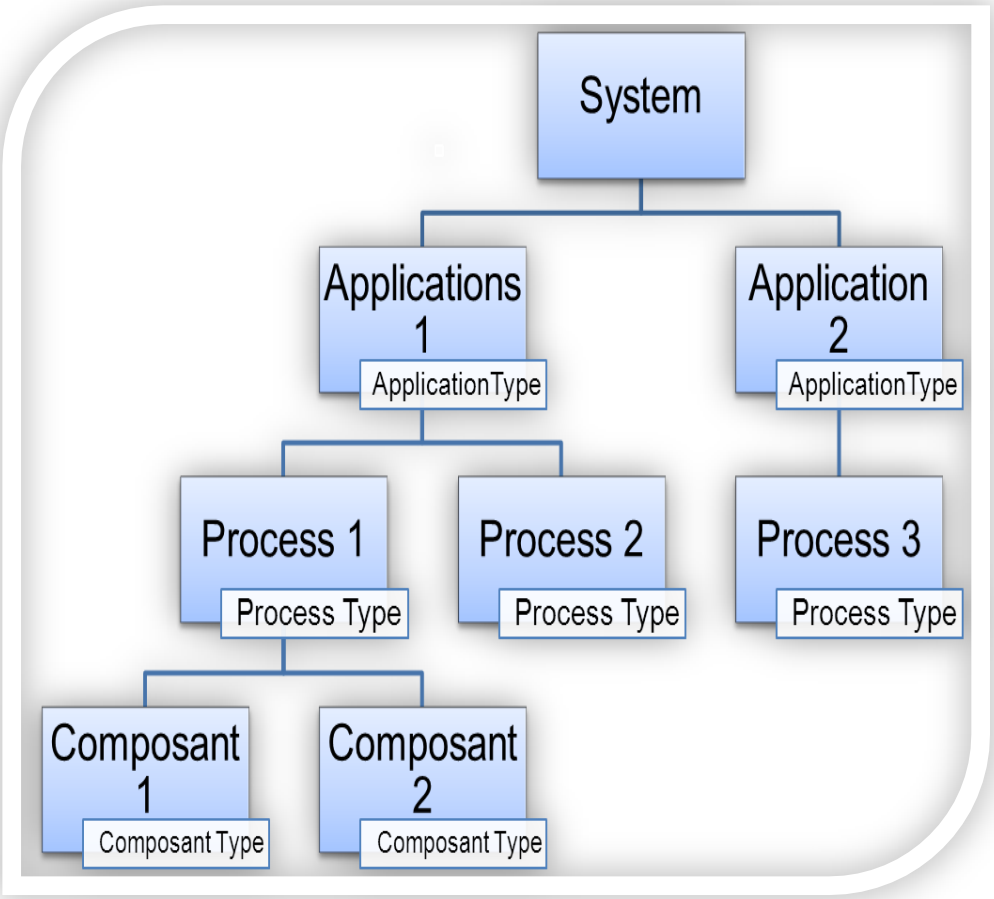


Figure 10: Architecture du system Coflight

COCO est dépendant de chaque modèle système qui évolue régulièrement et de manière conséquente tous les 3 mois environ.

III. 2. Présentation de CARDAMOM

CARDAMOM est le middleware utilisé sur les projets Coflight, 4Flight et Sesar. Comme le modèle système présenté précédemment, CARDAMOM est une des bases du système. Il est développé par le consortium industriel Thales/Selex. Ce logiciel est en charge de faire l'interface entre le logiciel développé et le système d'exploitation Linux.

La Figure 11 donne la position de CARDAMOM par rapport au CPR (Common PRoduct) (et ses applications) et les nœuds (machines) qui exécutent les composants du CPR.

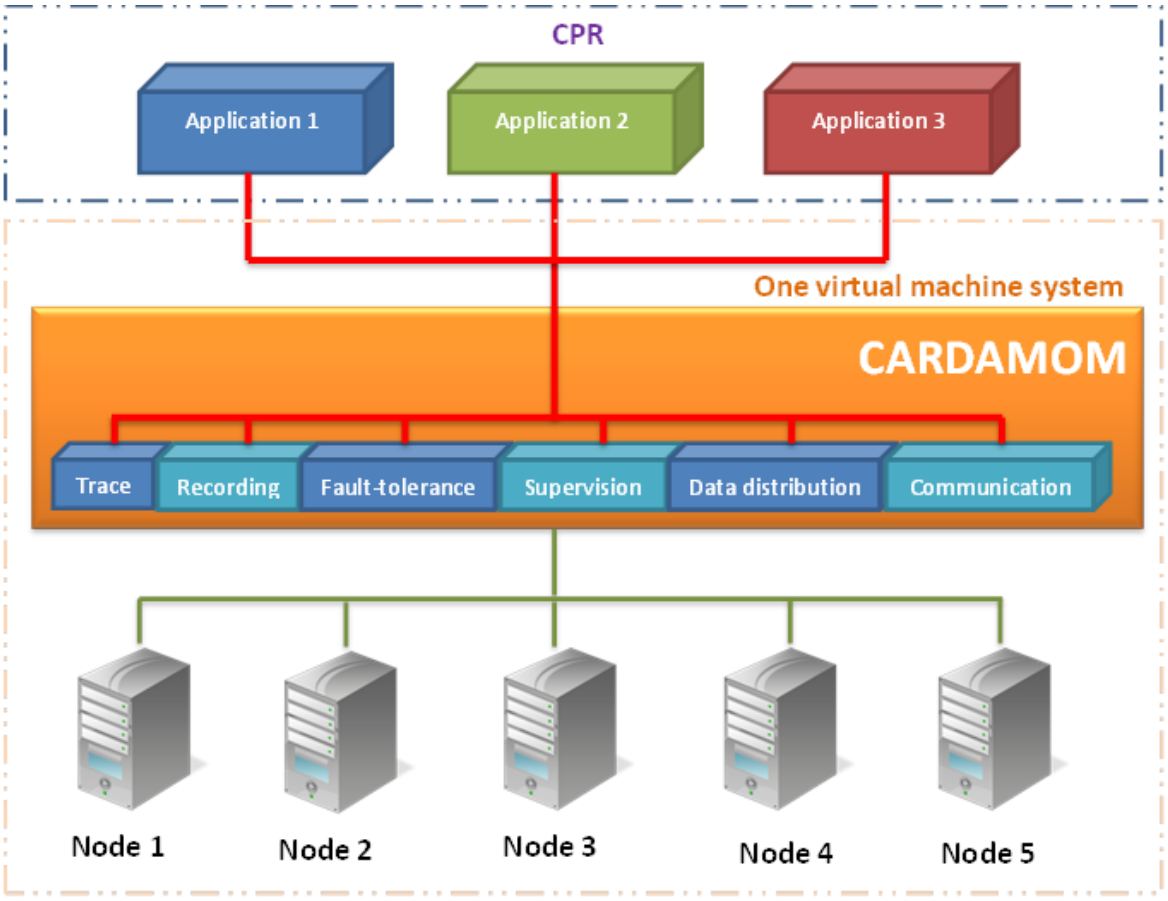


Figure 11: Vue d'ensemble de CARDAMOM

CARDAMOM est basé sur des standards d'architecture orientée objet et d'interopérabilité. Il utilise les mécanismes suivant, définis par une association appelée OMG (Object Management Group) :

- Au niveau de la couche métier, CARDAMOM utilise UML (standard OMG) et XML standard W3C (World Wide Web Consortium),
- Au niveau de la couche séparation, CARDAMOM utilise CCM (standard OMG), pour isoler la coucher métier des services techniques,
- Au niveau technique, CARDAMOM utilise CORBA (standard OMG).

CARDAMOM fournit ainsi un certain nombre de services décrits plus bas.

III. 2. a) Présentation de CORBA

CARDAMOM est basé sur l'architecture CORBA. CORBA est une architecture logicielle, pour le développement de composants. Ces composants, qui sont assemblés afin de construire des applications complètes, peuvent être écrits dans des langages de programmation distincts, être exécutés dans des processus séparés, voire être déployés sur des machines distinctes.

Plus précisément, l'objectif de CORBA est de simplifier le développement en fournissant un environnement qui, d'une part, masque les détails de l'utilisation des services offerts par les systèmes d'exploitation, et d'autre part, soit supporté par les différents systèmes d'exploitation actuels et à venir. Selon ces principes, la programmation distribuée deviendrait standardisée et interopérable, au sens où elle serait indépendante des exécutifs (système d'exploitation, couche matérielle de la machine hôte, langages de programmation…).

Tout ceci est résumé par la Figure 12 qui suit : plusieurs applications du CPR peuvent communiquer entre elles sans tenir compte des plateformes sur lesquelles s'exécutent leurs processus. On définit plus bas le concept de DDS introduit dans cette figure.

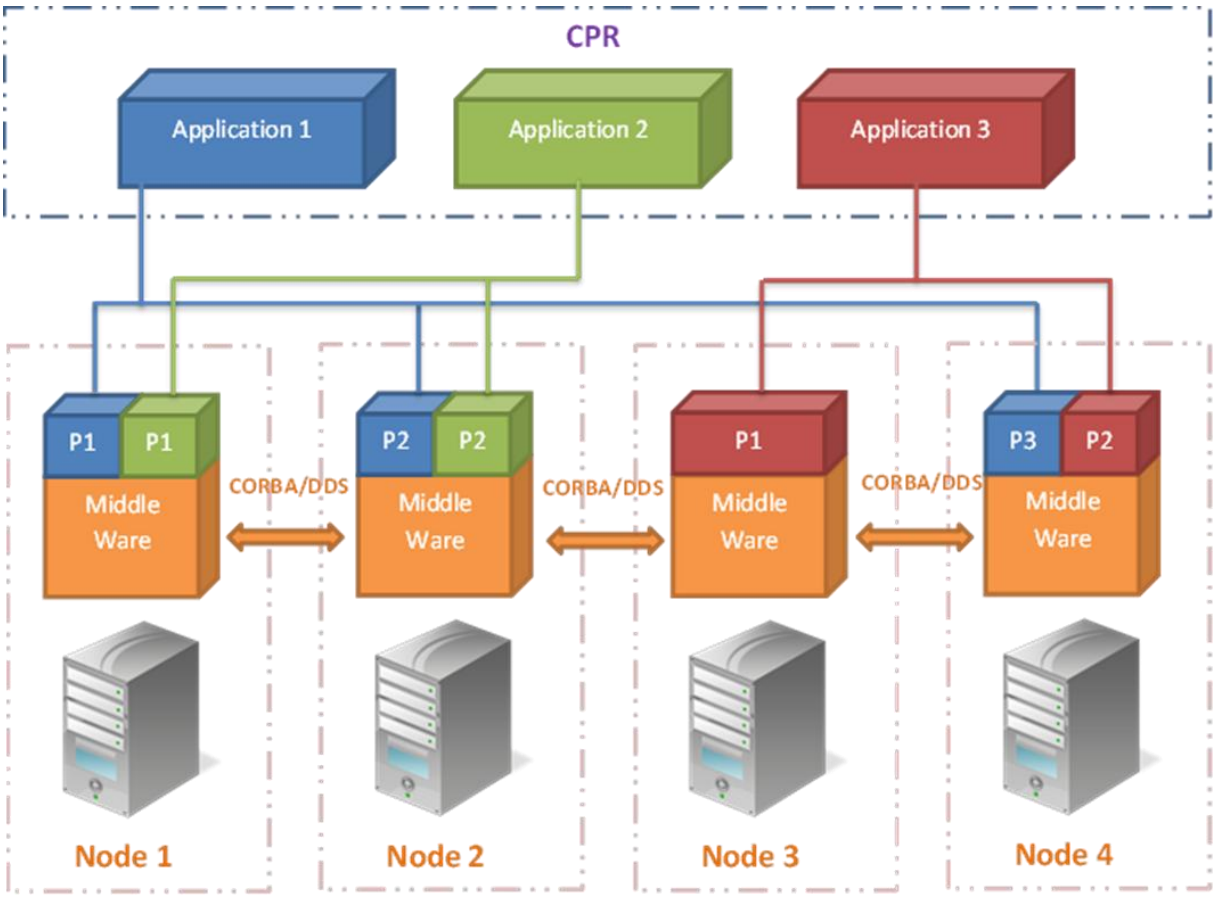


Figure 12: Communication entre applications via CORBA

III. 2. b) Services du middleware CARDAMOM

Outre les avantages fournis par CORBA (décrits ci-dessus), CARDAMOM accentue ses services sur la performance, l'évolutivité et la sécurité. Il offre plusieurs services, comme :

- La *fault-tolerance* (tolérance aux fautes) : ce service propose de basculer entre composants en cas de défaillance du composant primaire afin d'éviter une interruption de service. Cela nécessite donc aux composants d'avoir deux (ou plus) instances pour prendre le relais en cas de panne.
- Le *load balancing* (équilibrage de charge) : ce service permet de répartir les tâches à un ensemble de serveurs. La disponibilité des serveurs de traitement (*processing servers*) est vérifiée et les requêtes ne sont envoyées qu'aux serveurs en mesure de les traiter.
- Le service de distribution de données (DDS) : il s'agit d'un standard OMG qui réalise la communication et la distribution des données entre des clients (récepteurs) et des serveurs (processus traitant l'information). Il fonctionne de manière répartie et en temps réel pour délivrer la bonne information, au bon endroit et au bon moment. Il facilite donc la communication entre tous les acteurs qui interviennent lors de l'exécution des applications. La distribution des informations n'est pas impactée par le nombre de clients. Ce service de DDS est divisé en Topic. Chaque Topic contient les messages concernant un même sujet.

III. 3. Présentation de COCO2

COCO2 est un logiciel offline permettant de générer tous les fichiers de déploiement et de configuration d'un système utilisant le middleware CARDAMOM. Il est en réalité la seconde version d'un autre logiciel nommé COCO.

En effet COCO devenant de plus en plus complexe et de ce faite de moins en maintenable il fut entièrement refait dans le but de le rendre plus simple.

COCO2 utilise différents langages pour cette génération :

- Shell linux
- Java
- XML
- XSLT

COCO2 utilise plusieurs fichiers XML comme fichiers d'entrées pouvant contenir les spécifications du hardware (les machines disponibles et leurs types), les capacités dont on a besoins ex : le process X utilise une machine du type Y). Dans ces fichiers se trouve en

particulier toute la configuration des applications avec leurs options, leurs interfaces avec des définitions de toutes les connexions.

Ces différentes définitions et les configurations des interfaces rendent COCO2 complètement dépendant du modèle système et du middleware CARDAMOM.

Pour créer une configuration Coflight (**Voir Figure 14**), COCO2 prend en entrée un fichier XML de schéma de déploiement, fournit par l'utilisateur final, dans lequel est spécifié sur quel nœud doit être déployé chaque process.

Puis il utilise d'autres fichiers XML qui sont quant à eux fournis avec COCO2 et qui représentent les différentes configurations pour Coflight :

- Le system model : ce fichier est généré à partir du system model et il contient toutes les informations de celui-ci comme quel process utilise quel composant ou encore quelles sont les interfaces internes et externes d'un composant.
- Le fichier d'intégration : ce fichier décrit tout ce que ne vient pas du system model comme les arguments exécutions de chaque process ou la taille et la configuration des DDS. Il contient également différentes informations pour configurer le middleware CARDAMOM.
- Le fichier de connexion : ce fichier permet de particulariser certaines connexions entre différents process. Par exemple, certains process doivent être déployés sur un même nœud. Les connexions entre ces deux process sont alors spécifiquement locales (Voir Figure 13).

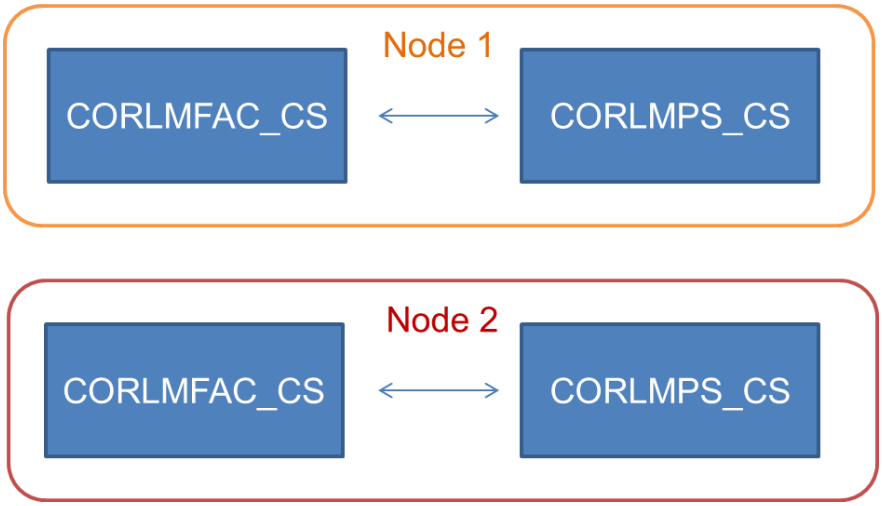


Figure 13: Connections intra node

COCO2 permet de configurer le middleware CARDAMOM ainsi que tous ses services décrits sur la **Figure 11** en fonction du déploiement demandé par l'utilisateur. Il configure aussi chaque élément du système, Opensplice, les applications, les processes, etc.

Sur une plateforme de référence Coflight par exemple, COCO2 génère environ 440 fichiers. Son utilisation est permanente et primordiale. La réactivité des corrections est donc très importante sur un projet.

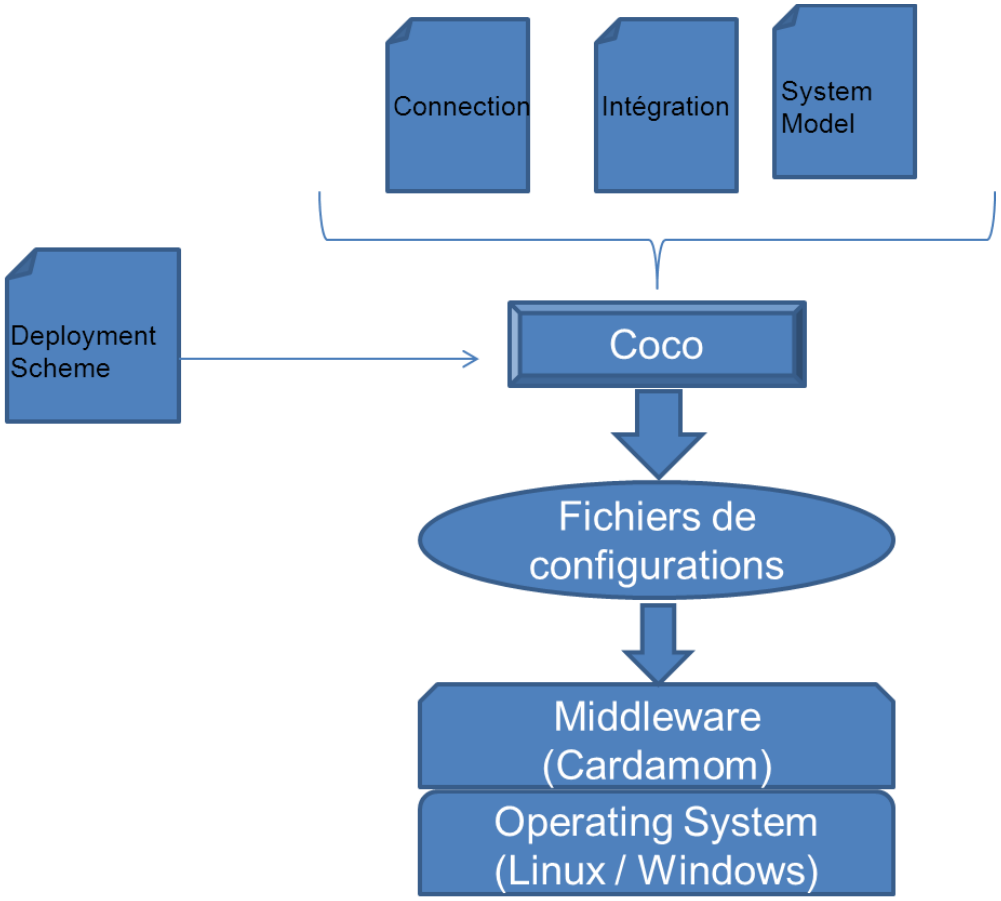


Figure 14: Fonctionnement de COCO2

III. 4. Justification du stage

COCO2 est une application très complexe qui demande une connaissance non seulement du système dans sa globalité mais aussi une connaissance particulière du middleware. C'est un logiciel qui est tout le temps utilisé. Et la rédaction d'une documentation approfondie est alors devenue indispensable.

COCO2 est utilisé sur plusieurs projets qui ont plusieurs versions en parallèle. Cette multiplicité des versions logicielles complique la maintenance de l'application. La politique produit chez Thales Air Systems est importante. COCO2 a été développé dans ce cadre-là. Mais les spécificités des différents projets nous amènent à revoir la conception et la maintenance de COCO2.

Actuellement, pour la prise en compte d'un nouveau Coflight dans l'environnement 4-Flight ou SESAR, toutes les modifications sont faites à la main. COCO2 est complexe et utilise des fichiers de très grande taille (plus de 8000 lignes pour le fichier d'intégration). Cette mise à jour des fichiers assez fastidieuse est alors source d'erreurs possibles. Actuellement cette mise à jour manuelle est estimée à une semaine de travail.

III. 5. Travail demandé

Pour pallier à ces différents aspects il m'a été proposé de remplir les exigences suivantes :

1. Adapter COCO2 pour faciliter le passage d'une configuration Coflight à une configuration 4-Flight de manière la plus simple, automatique et rapide possible.
    a. Analyse de l'existant
    b. Propositions d'améliorations
    c. Mise en œuvre des modifications
2. Faire une documentation de COCO2 afin de permettre aux futurs utilisateurs de mieux appréhender le logiciel pour y apporter des modifications par exemple.
    a. Documentation du code
    b. Guide d'utilisation

III. 6. Les exigences

Dans le cadre du stage certaines exigences ont été définit lors de la première réunion :

1. Les modifications apportées doivent être le plus génériques possible car d'autres projets vont aussi intégrer Coflight (le projet SESAR avec différentes maquettes en cours et futures)
2. Les fichiers de configurations générés par COCO2 après application de mes modifications doivent être identiques à ceux générés par la version actuelle de COCO2. Cela afin d'assurer une non régression du programme.

IV. Gestion de projet

Maintenant que j'ai présenté le projet, je vais m'intéresser à la gestion du projet afin de voir l'évolution du projet tout au long du stage.

IV. 1. PBS

Dans le but de respecter les exigences citées précédemment, et de bien mener le projet, un ensemble de livrables matériels et logiciels doivent être livrés tout en suivant une planification respectant les objectifs du stage.

Ces livrables sont organisés selon le PBS (Project Break down Structure) suivant :

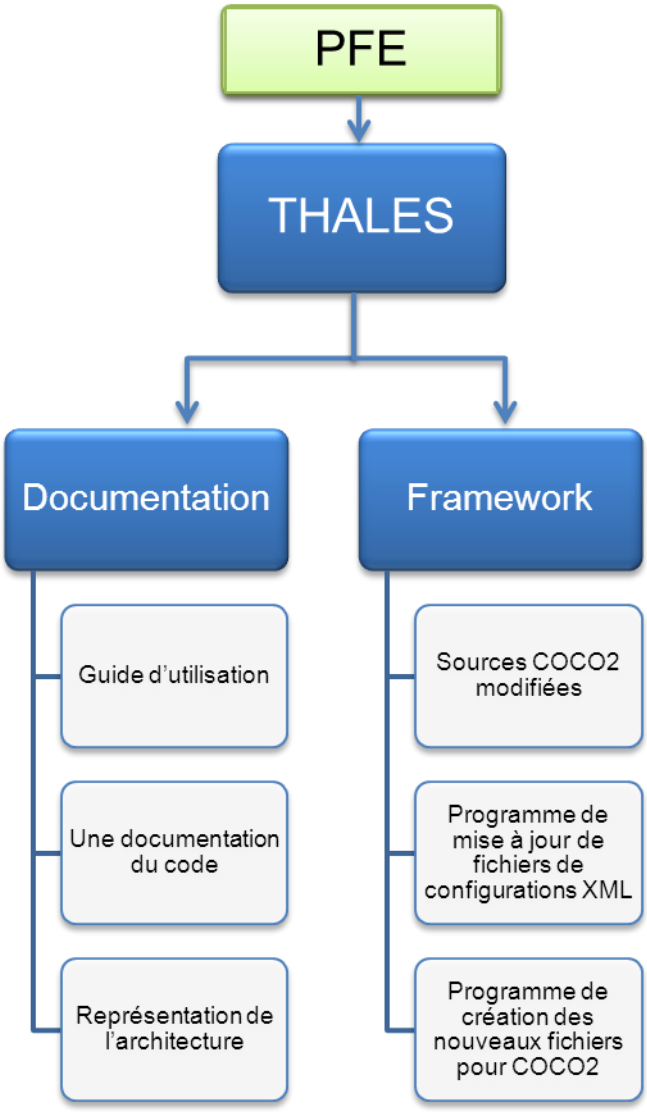


Figure 15: Project Break down Structure

IV. 2. Planning du stage

Le planning ci-dessous (Figure 16) représente les différentes tâches réalisées durant mon stage avec les temps consacrés à chacune d'elles. Ainsi nous pouvons voir que mon stage contenait 3 principales parties qui seront détaillées dans le chapitre suivant

Nom	Durée	Date de dé...	Date de fin
Portage du logiciel COCO2	101	14/04/14	05/09/14
Prise en main du sujet et appropriation du contexte	11	14/04/14	28/04/14
Lecture de la documantation de COFlight	6	14/04/14	21/04/14
Formation sur CoFlight	0	29/04/14	29/04/14
Phase d'analyse	11	22/04/14	07/05/14
Etudes des fichiers XML utilisés par COCO2	5	22/04/14	28/04/14
Etudes des programmes JAVA et Shell	6	29/04/14	07/05/14
Proposition d'une solution	0	09/05/14	09/05/14
Première phase de coception et de réalisation	11	09/05/14	23/05/14
Développement de la solution des plugin pour les process	11	09/05/14	23/05/14
Phase de documentation	26	26/05/14	30/06/14
Création des diagrammes des classes	17	26/05/14	17/06/14
Début de la ducumentation du code	9	18/06/14	30/06/14
Seconde phase de développement et de conception	34	01/07/14	19/08/14
Développement de la soution des plugin pour les applications	11	01/07/14	16/07/14
Test sur platforme de la nouvelle solution	7	17/07/14	25/07/14
Développement du programme de mise à jour	16	28/07/14	19/08/14
Rédaction du rapport	19	11/08/14	05/09/14

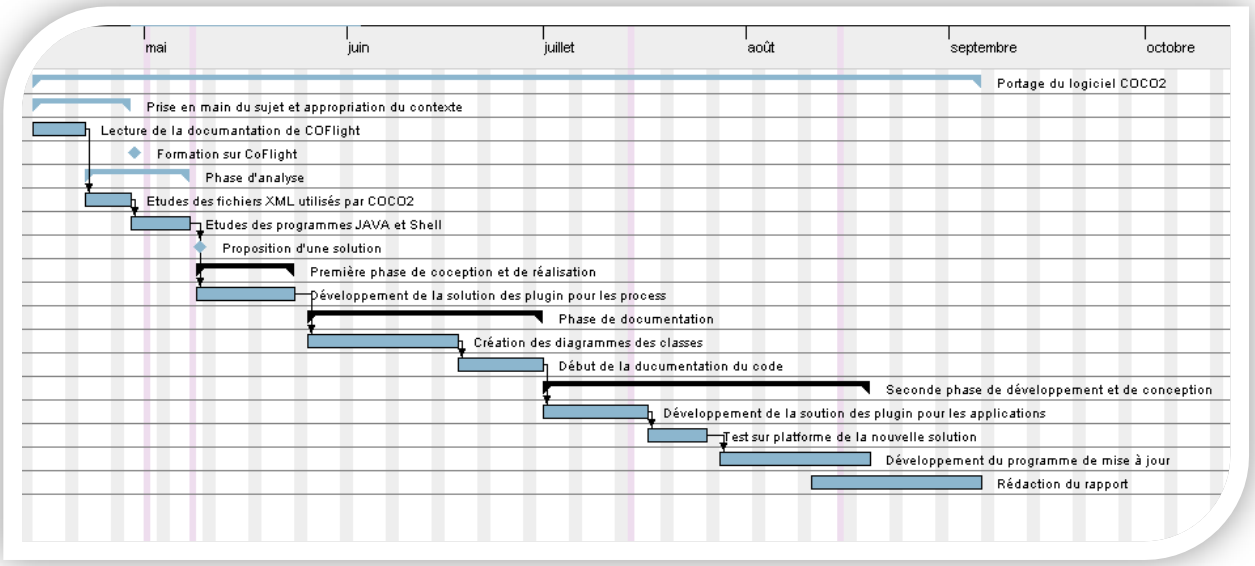


Figure 16: Diagramme de Gantt

V. Réalisation

V. 1. Analyse de l'existant

Au début du stage je me suis familiarisé avec COCO2 et tout son environnement. Pour cela j'ai donc commencé par lire la documentation existante. J'ai assisté à une formation sur Coflight pour avoir une vue d'ensemble du logiciel et en avoir une première approche. En effet aux premiers abords il est assez compliquer d'en appréhender le fonctionnement.

Une fois le principe global du logiciel COCO2 compris je me suis lancé dans la compréhension du code de celui-ci en essayant de voir les différents enchaînements entre les sous-programmes constituant COCO2.

Comme le montre la Figure 17ci-dessous COCO2 fonctionne de la manière suivante.

1er Phase : Compilation

Delivery.sh et build.xml utilisent Apache Ant pour compiler et rassembler les classes Java dans un Jar. Ces classes représentent tous les types d'objets utilisés par Coflight. Par exemple on y retrouve la classe représentant une application.

Outre le fait d'ajouter les classes compilées, build.xml ajoute au Jar tous les fichiers nécessaires à la génération d'une configuration.

Cette compilation permet aussi de créer un COCO Stand-alone qui permet la génération des fichiers COCO sans être dans l'environnement de compilation.

2ième phase : Lancement de la génération

Durant cette phase Cocospv.sh met en place les variables de l'environnement pour la génération et lance coco.sh.

3ième phase : Génération

Coco.sh va lire les fichiers XML d'entrées. Et pour chaque schéma de déploiement fournit par l'utilisateur, il va créer tous les fichiers nécessaire au déploiement sur les machines. La génération des fichiers de déploiement et de configuration est faite durant cette phase.

4ième phase : Résultat de la génération

A ce moment-là, tous les fichiers de déploiement et configurations pour CARDAMOM sont créées.

5ième phase : Appel au « packaging »

Cocospv.sh appelle spvfactory.sh pour « packager » tous les fichiers générés afin d'être déployés ensuite.

6ième phase : Génération du résultat final

Spvfactory.sh récupère tous les fichiers de configuration et les rassemble dans SPV (Software Package Version). Lequel est par la suite installé sur les plateformes.

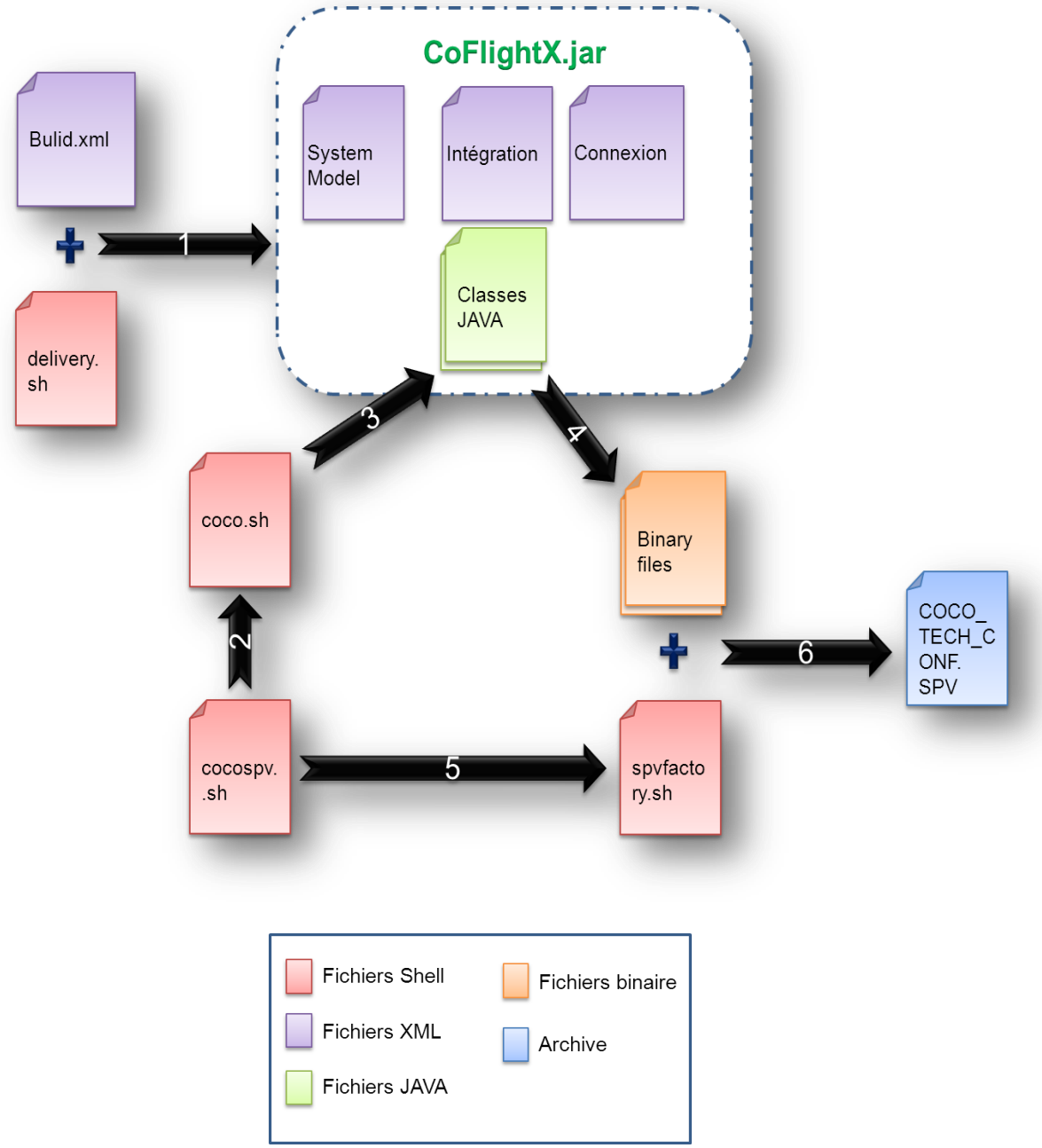


Figure 17: Schéma de fonctionnement simplifié de COCO2

V. 1. a) Modélisation de l'architecture de COCO2

Une fois le fonctionnement principal de COCO2 compris je me suis lancé dans la compréhension du code JAVA présent qui est le cœur de métier de celui-ci. J'ai donc fait du « reverse engineering» en lisant et réalisant le diagramme des classes du code JAVA.

V.1.a.i Visual Paradigme

Visual Paradigm (Figure 18) est un logiciel de création de diagrammes dans le cadre d'un projet de programmation. Tout en un, il possède plusieurs options permettant une large possibilité de modélisation en UML. En effet ce logiciel offre de nombreux outils pour créer différents types de schémas comme les diagrammes d'exigences et de cas d'utilisation. Il possède aussi un bon nombre de navigateurs permettant la personnalisation chaque élément.

Visual Paradigm sert aussi à l'analyse et à la manipulation de code en générant des codes sources en divers langages comme le Java ou C++ à partir du modèle créé. Et inversement, il permet de produire un modèle à partir de codes sources.

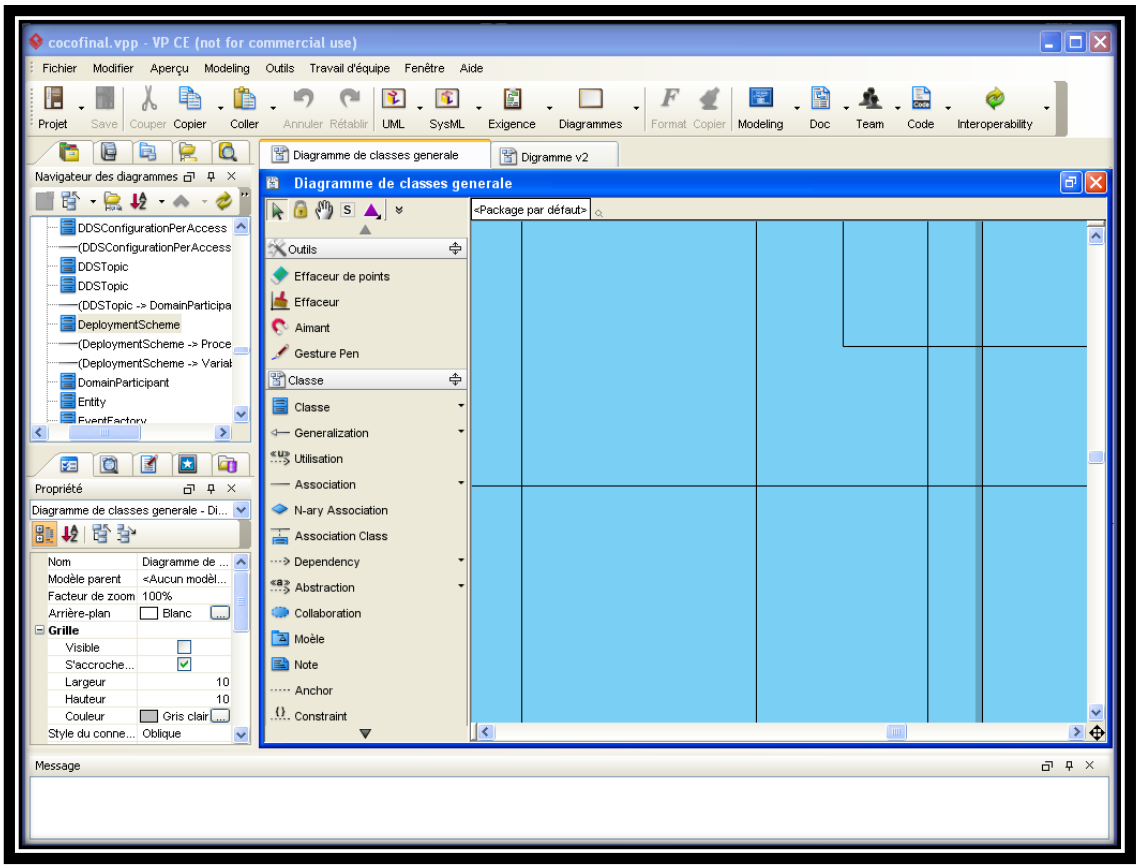


Figure 18: Outil Visual Paradigm

V.1.a.ii Diagrammes des classes

Pour la réalisation des diagrammes de classes j'ai d'abord utilisé l'analyse de code de Visual Paradigm pour générer les classes JAVA avec leurs attributs et leurs fonctions. Car le code de COCO2 est composé de 74 classes java dont certaines font plus 1550 lignes.

J'ai ensuite relu le tout code en suivant le programme principal comme fil conducteur afin de pouvoir représenter les différents liens entre les classes. J'ai ainsi rajouté les associations multiples ou les associations de compositions. Ceux-ci ne pouvant pas être généré par Visual Paradigm car cela nécessite une compréhension du fonctionnement du programme.

Le diagramme final était tellement complexe, dû aux nombres importantes de classes présentes et d'associations, que j'ai décidé de le séparer en plusieurs parties (**Voir Figure 19 et Annexe III** : Diagramme de classes complet**, Annexe IV** : Diagramme du système model**, Annexe V** : Diagramme des applications**, Annexe VI** : Diagramme des connexions CORBA**, Annexe VII** : Diagramme de déploiement CSCI). Afin de permettre à celui-ci d'être beaucoup plus lisible pour les futurs utilisateurs. J'ai donc découpé le diagramme en cinq sous diagrammes en fonction de la partie de l'architecture qu'ils servent à représenter :

- Les connexions CORBA
- Le déploiement des CSCIs
- Le System Model
- Les applications
- Les connexions CSCIs

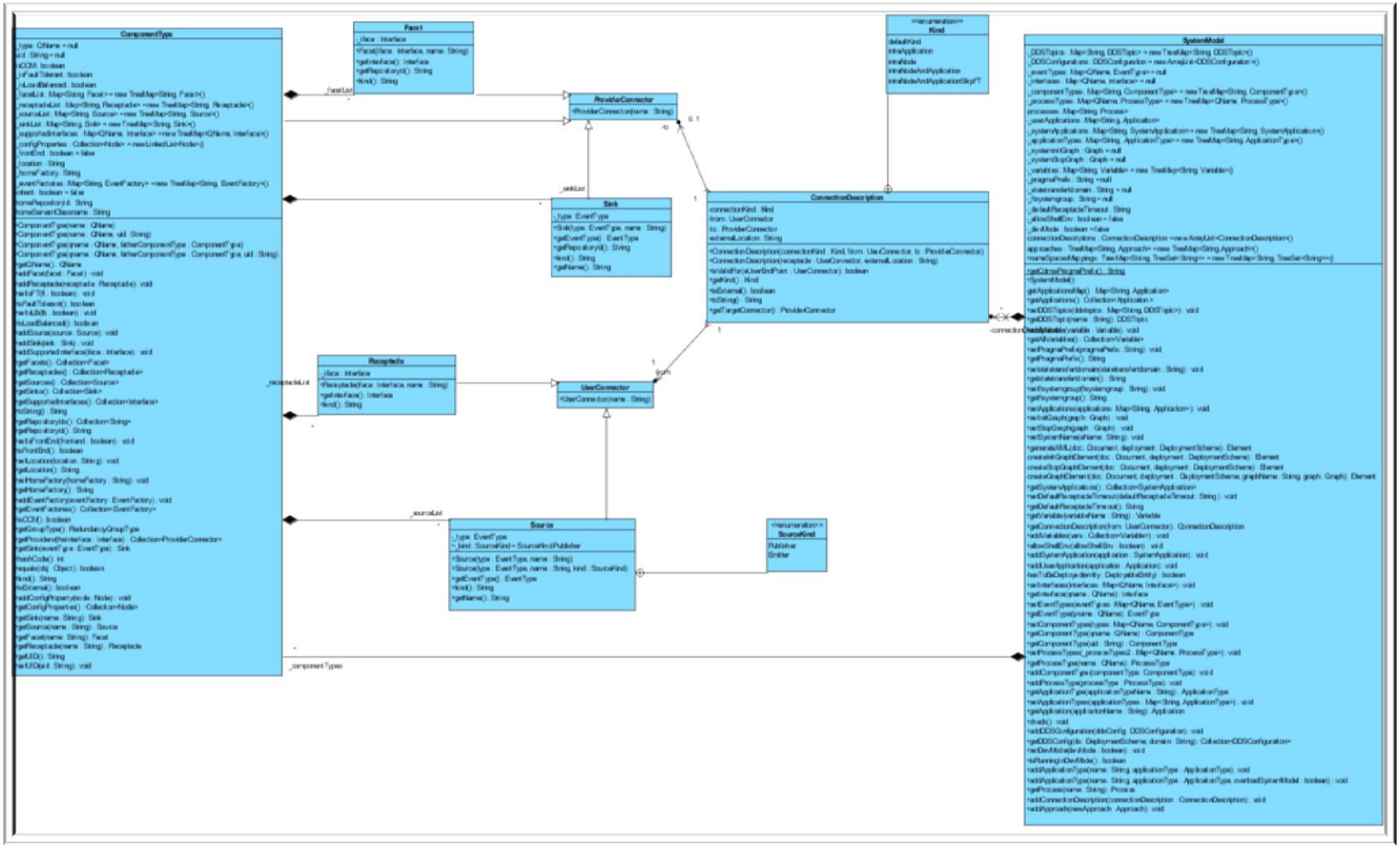


Figure 19: Diagramme de classe des connexions CSCI

Cette première phase d'étude de l'existant m'a permis par la suite de mieux appréhender les modifications et leurs impacts sur l'ensemble du logiciel COCO2 en ayant une idée précise de l'architecture de celui-ci.

V. 1. b) Conclusion

Cette partie correspond aux phases d'analyse et d'une partie de la phase de documentation du planning qui à durée au total 28 jours. Cette étape a été complexe car j'ai dû, en discutant beaucoup avec les ingénieurs, comprendre tous les aspects techniques de COCO2. Elle a été aussi l'occasion pour moi, d'apprendre à utiliser un nouveau logiciel de modélisation UML (Visual Paradigm).

V. 2. Propositions d'améliorations

Au tout début de mon stage il m'a été demandé de proposer une ou plusieurs solutions afin de rendre le plus automatique le passage de la création d'une configuration pour un Coflight à un Coflight pour 4-Flight.

Pour réaliser cela j'ai effectué une comparaison entre l'environnement de création d'un simple Coflight et celle d'un Coflight pour 4-Flight. J'ai pu ainsi remarquer que les fichiers d'entrées étaient les mêmes mais que certains fichiers XML recevaient des modifications. Par exemple l'ajout ou le changement de valeurs pour des éléments. Pour résoudre ce problème j'ai trouvé deux solutions.

Les solutions :

La première était de faire un dossier spécial pour 4-Flight contenant les fichiers modifiés pour 4-Flight. Et dire à COCO2 d'utiliser ces fichiers si le projet est un 4-Flight. Cette solution n'a pas été retenue car elle avait un énorme inconvénient. En effet en dupliquant ces fichiers la maintenance devenait compliquée, obligeant à garder à jour plusieurs fichiers du même type. Avec toutes les versions à venir (sans oublier SESAR), cette solution était donc impossible.

La deuxième était de mettre en place un système de mise à jour sans avoir à aller dans chacun des fichiers pour y apporter les modifications. C'est cette solution qui a été gardée et qui sera décrite dans les chapitres en dessous. Car contrairement à la précédente

une seule version des fichiers est gardée. De plus cette solution apportée comme avantages :

1. Une rapidité de mise en œuvre.
2. Une diminution des sources d'erreurs.

Durant cette comparaison j'ai aussi remarqué que la différence principale venait du fichier d'intégration que COCO2 prend en fichier d'entrée pour générer les fichiers de configuration. En effet lors du passage vers 4-Flight j'ai pu remarquer que :

- Certaines applications sont ajoutées ou supprimées afin de permettre à 4-Flight de proposer plus de fonctionnalités.
- Certains processus avaient été rajoutés ou supprimés à l'intérieur d'applications.
- Et de plus les modifications des applications entraînaient des changements au niveau du fichier de connexion.

Ces modifications de processus ou d'applications dans le fichier d'intégration sont très longues et comme tout est manuel des erreurs peuvent être commises. Il est donc nécessaire de trouver une solution adaptée.

Les solutions :

Pour pallier à ces problèmes la première solution était d'utiliser le système de mise à jour pour enlever ou rajouter les applications et les processus. Malgré le fait que cette solution permette de pouvoir voir de manière rapide les différents ajouts et suppression. Cette solution n'apporte aucun gain de temps par rapport à la méthode actuel.

La deuxième solution, qui a été retenue, était de mettre en place un système de plugin pour permettre de rajouter ou d'enlever des applications et des processus de manière simple et transparente. Les avantages de celle-ci sont :

1. Aucune modification de fichier n'est nécessaire.
2. Aucune erreur n'est possible car tout est géré par COCO2 automatiquement.

V. 3. Cahier des charges

Après la validation de ces solutions nous avons établi le cahier de charge suivant, en plus des exigences mentionnées au paragraphe III. 6.

V. 3. a) Objectifs

Le premier objectif est de diviser de manière judicieuse le fichier integration.xml tout en garantissant le fonctionnement actuel de COCO2. Ceci pour permettre à l'utilisateur de simplement déplacer des fichiers pour mettre à jour les applications.

Quant au système de mise à jour des fichiers, celui-ci devra être un programme pouvant être lancé en amont de COCO2. Avec ce système, l'utilisateur devra pouvoir apporter toutes les modifications suivantes aux fichiers de configuration XML:

1. L'ajout d'éléments
2. La suppression d'éléments
3. La modification d'éléments

V. 3. b) Langage de l'outil

La langue des outils est l'anglais

V. 3. c) Langage de programmation

Aucune contrainte n'a été donnée. Mais je devais essayer de rester le plus cohérent par rapport aux langages déjà utilisés :

1. JAVA
2. XML
3. XSD
4. XSLT
5. Shell script

V. 4. Développement des améliorations

V. 4. a) Analyse du fichier d'intégration

Afin de mettre en place la solution de plugin, j'ai analysé l'impact d'un ajout ou d'une suppression de process et d'application au sein du fichier d'intégration.

Pour modéliser ce système le fichier d'intégration, voir Figure 20, définit d'abord les variables de l'environnement. Puis il définit les propriétés du système avec les graphes d'initialisation et d'arrêt du système. Ces graphes correspondent à l'ordre de lancement ou d'arrêt des différentes applications car certaines applications sont dépendantes entres elles.

Ensuite le fichier d'intégration définit les types de chaque application et des process qui les composent.

Nous retrouvons après la même arborescence pour la définition des applications, des process et des composants. Avec pour chaque application l'ajout de leurs propres graphes d'initialisation et d'arrêt.

Et pour finir on retrouve la définition de la DDS (voir paragraphe III. 2. b), avec leurs différents Topics, et les process pouvant écrire dedans.

```
<?xml version="1.0" encoding="UTF-8" ?>
<IntegConf xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation="../integration.xsd">
  <Vars>
    <Var Name="COFLIGHT_HOME" Value="/opt/coflight" />
  </Vars>
  <System/>
  <ApplicationTypes>
    <ApplicationType Name="FDPS">
     <ProcessType Name="cfl.CPR.PCs.FDPS.FDPS_PS_CS" Csci-name="FDM" isRecorded="false" MinOccur="1">
        <ComponentType qname="cfl.CPR.PCs.FDPS.FDML.FDM" Location="${FDM_HOME}/lib/java/libFDM.jar">
            <Home-factory>com.coflight.cpr.fdps.fdm.framework.mainExecutor.FDMHomeMainExecutor.createCCMFDMHome</Home-factory>
        </ComponentType>
     </ProcessType>
    </ApplicationType>
  </ApplicationTypes>
  <Applications>
    <Application Name="FDPS" Max-nbr-of-restart-attempt="3" Force-manual-start="AUTO" Force-manual-start-persist="TRUE">
        <Managed-process Name="FDPS.FDPS_PS_CS" Host-name="${HOST}" Port="${FDPS_PS_PORT}" Type="COMPONENT-SERVER" Autoended="F
            <Ccm-component Name="FDPS.FDM" Redundancy-group-name="FDMLBGroup" Redundancy-group-type="LB_GROUP_TYPE">
              <Property Name="isPrimary" Type="Short" Value="0" Mode="RW" />
              <Property Name="group" Value="FDMLBGroup" Mode="RO" Type="String" />
            </Ccm-component>
        </Managed-process>
    </Application>
  </Applications>
  <DDS/>
</IntegConf>
```

Figure 20: Fichier d'intégration simplifier

Le fichier d'intégration est très complexe et très important (8000 lignes pour Coflight). L'exemple de fichier ci-dessus me permet d'expliquer plus facilement mon exemple suivant.

Dans cet exemple, je veux ajouter un process dans une application. La procédure est :

1. Trouver l'application type
2. Ecrire la définition du type du process que l'on veut ajouter (et le type des composants si il y en a)
3. Trouver l'application
4. Ecrire la définition du process que l'on veut ajouter (et des composants si il y en a)
5. Modifier les graphs d'initialisation et d'arrêt de l'application.

V. 4. b) Découpage du fichier d'intégration

Pour mettre en place la première solution exposée dans le cahier des charges j'ai d'abord commencé par séparer le fichier d'intégration. Dans le but de mettre en place ce système, j'ai décidé de séparer le fichier en 4 parties présentées ci-dessous :

1. Un fichier ne contenant que des définitions des variables du système.
2. Un fichier contenant les informations du système avec ses propriétés.
3. Un ensemble de fichiers correspondant chacun à un process et à son type.
4. Un ensemble de fichiers correspondant chacun à une application et à son type.

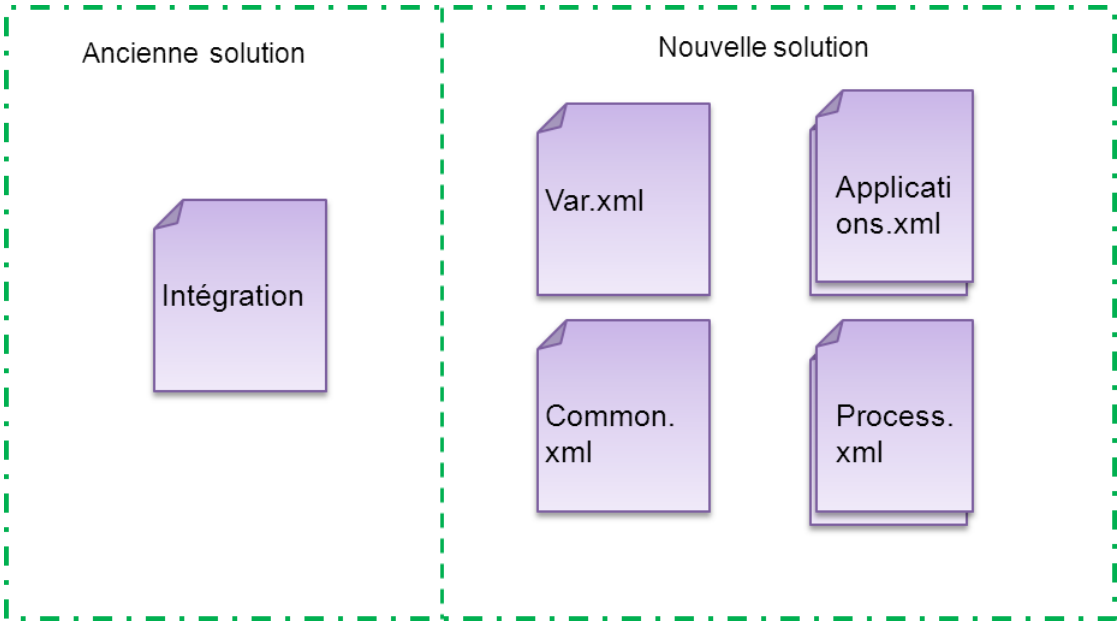


Figure 21: Nouvelle solution

Ce fichier étant très long (plus de 8000 lignes environ) cela m'était donc impossible de le traiter à la main.

Le fichier d'intégration étant écrit en XML, j'ai donc opté tout naturellement pour l'utilisation de la technologie XSL (XSLT) afin de pouvoir réaliser cette division automatiquement (**Voir Annexe VIII** : XSLT Pour enlever les process du fichier d'intégration).

XSL :

XSL spécifie les règles de présentation d'un document XML en utilisant XSLT, qui utilise le vocabulaire de formatage, pour décrire comment le document peut être transformé en un autre document. Et cela permet de transformer des documents XML en d'autres documents XML en décrivant les règles pour transformer un arbre source en un arbre résultat. Ces règles modèles sont exprimées dans un document XML qui est appelé feuille de styles. Une règle modèle est constituée de deux parties : un motif qui sert à identifier des nœuds de l'arbre source et un modèle pouvant être instancié afin de constituer une partie de l'arbre résultat. Ceci permet à une feuille de styles d'être applicable à une large catégorie de documents ayant des structures d'arbres source similaires.

Pour séparer le fichier d'intégration j'ai alors développé un programme JAVA totalement indépendant de la génération faite par l'utilisateur. Ce programme utilise des fichiers XSL, décrit dans les paragraphes suivants, ce qui m'a permis de traiter de manière efficace et rapide le fichier d'intégration.

Dans un premier temps, j'ai écrit le fichier XSL permettant de créer les fichiers XML contenant les variables et celui contenant les informations du système. La génération de ces fichiers ne demandait pas de traitements spécifiques. En effet ces deux fichiers ne font que parcourir le fichier d'intégration et recopier les nœuds dont ils ont besoin.

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
    <xsl:output method="xml" indent="yes"/>

    <xsl:template match="/" >
        <my:IntegConf xsi:schemaLocation="http://www.omg.org/Integration/ integration.xsd" xmlns:xsi="

            <xsl:text>&#xA;  </xsl:text><xsl:copy-of select="IntegConf/statetransfertdomain" />
            <xsl:text>&#xA;  </xsl:text><xsl:copy-of select="IntegConf/ftsystemgroup" />
            <xsl:text>&#xA;  </xsl:text><xsl:copy-of select="IntegConf/default-receptacle-timeout" />
            <xsl:text>&#xA;  </xsl:text><xsl:copy-of select="IntegConf/System" />
            <xsl:text>&#xA;  </xsl:text><xsl:copy-of select="IntegConf/ComponentTypes" />
            <xsl:text>&#xA;  </xsl:text><xsl:copy-of select="IntegConf/ProcessTypes" />
            <xsl:text>&#xA;  </xsl:text><xsl:copy-of select="IntegConf/DDS" />

        <xsl:text>&#xA;</xsl:text></my:IntegConf>

    </xsl:template >

    <xsl:template match="text()"></xsl:template>

</xsl:stylesheet>
```

Figure 22: XSL de création du fichier XML du system

Une fois ces deux fichiers finis et valides, je me suis lancé dans la création des différents fichiers process. Pour réaliser cela, l'utilisation de XSL et de JAVA après réflexion, c'est révélé peu performante. Car pour récupérer le nom de chaque process et les mettre en relation avec leur type, il me faut obligatoirement parcourir le fichier d'intégration.

J'ai donc implémenté dans mon programme JAVA à l'aide de l'api DOM, un parseur qui parcourt tout le fichier d'intégration afin de récupérer les différents types et process afin de créer un fichier pour chacun d'eux.

La première difficulté à ce niveau-là a été de mettre dans les fichiers XML de process toutes les données pour que COCO2 puisse les traiter correctement. En effet COCO2 a besoins de certaines informations sur l'application, comme son nom ou savoir si l'application est déjà définie dans le système model. J'ai donc trié les différents attributs des applications pour ne garder que celles étant essentielles.

Et la deuxième difficulté a été de regrouper dans le même fichier les process et leurs types. En effet certaines correspondances entre process et types sont uniquement définis dans le système model. Donc pour résoudre ce problème, il m'a fallu d'abord parser le système model pour récupérer les types déjà définis. Puis je les stocke dans une Map

(tableau permettant l'accès aux valeurs à partir de clés) et les réutilise lorsque je parse le fichier d'intégration.

Une fois les fichiers de process réalisés j'ai adapté le parseur précédemment créé pour qu'il puisse aussi générer les fichiers XML d'application sur le même principe de parcours du fichier d'intégration.

Pour finir j'ai adapté le fichier de validation XSD utilisé pour l'ancien fichier d'intégration afin de l'adapter aux fichiers common, var, et application. Et j'ai écrit un nouveau fichier XSD pour valider les fichiers de process (**Voir Annexe IX** : XSD de validation des fichiers de process).

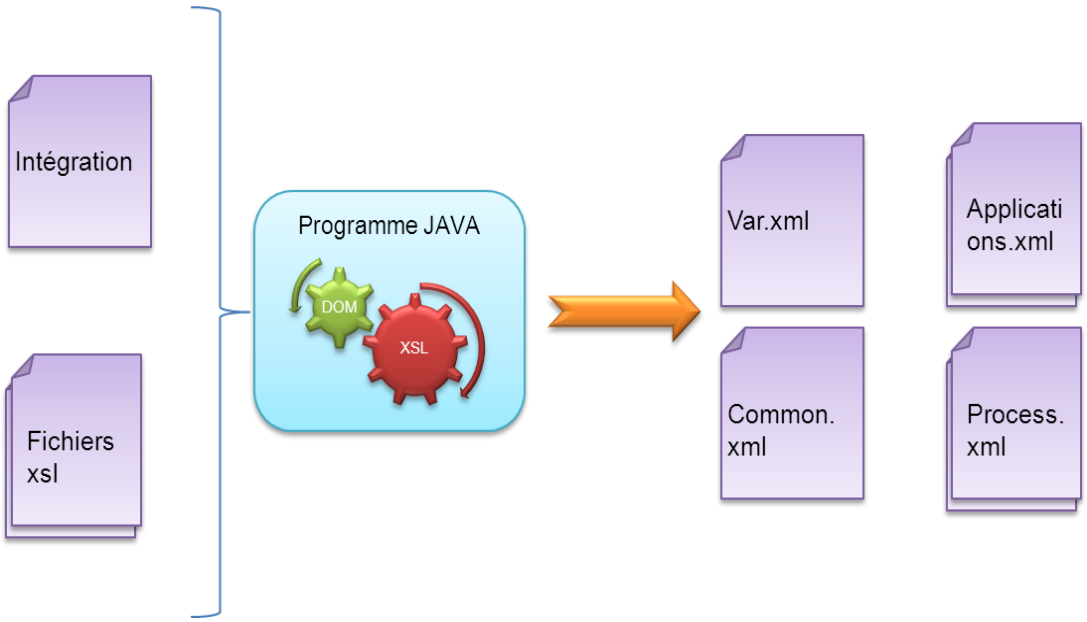


Figure 23: Fonctionnement du programme de séparation du fichier d'intégration

V. 4. c) Modification de la génération

Une fois la partie de séparation du fichier terminé, j'ai modifié le programme de COCO2 pour qu'il puisse utiliser les différents fichiers générés à la place du simple fichier d'intégration. Là plusieurs problèmes sont apparus :

1. L'accès aux fichiers de configuration contenus dans le Jar se fait en entrant en dur leurs noms. Mais vu que les fichiers d'applications et de process changent cela devient impossible.

2. Avec la nouvelle architecture adoptée les types des process sont déclarés dans le même fichier que leurs process. Mais comme plusieurs process peuvent avoir le même type il faut s'assurer qu'un même type possède la même définition dans chaque fichier.

Pour pallier à ces différents problèmes j'ai :

1. Créé une classe Java qui récupère dans un Jar une liste de fichier à partir d'une partie de leurs chemins d'accès sous lesquels ils se trouvent. Cela permet ainsi de ne pas être obligé de connaître les noms exacts des fichiers recherchés.
2. Créé une classe Java pour me permettre de comparer des nœuds XML avec leurs attributs et de manière récursive en comparant leurs nœuds fils.
3. Modifié le programme de parsing du fichier d'intégration pour pouvoir lire séparément les fichiers des applications de ceux des process.
4. Vérifier la cohérence des types de process entre les fichiers.

Et pour finir, j'ai ajouté dans le programme principal la mise à jour des graphes d'initialisation et d'arrêt du système et des applications pour qu'ils s'adaptent en fonction des éléments présents dans le système. Ainsi l'ajout ou la suppression d'une application ou d'un processus n'entraine aucune modification d'autres fichiers.

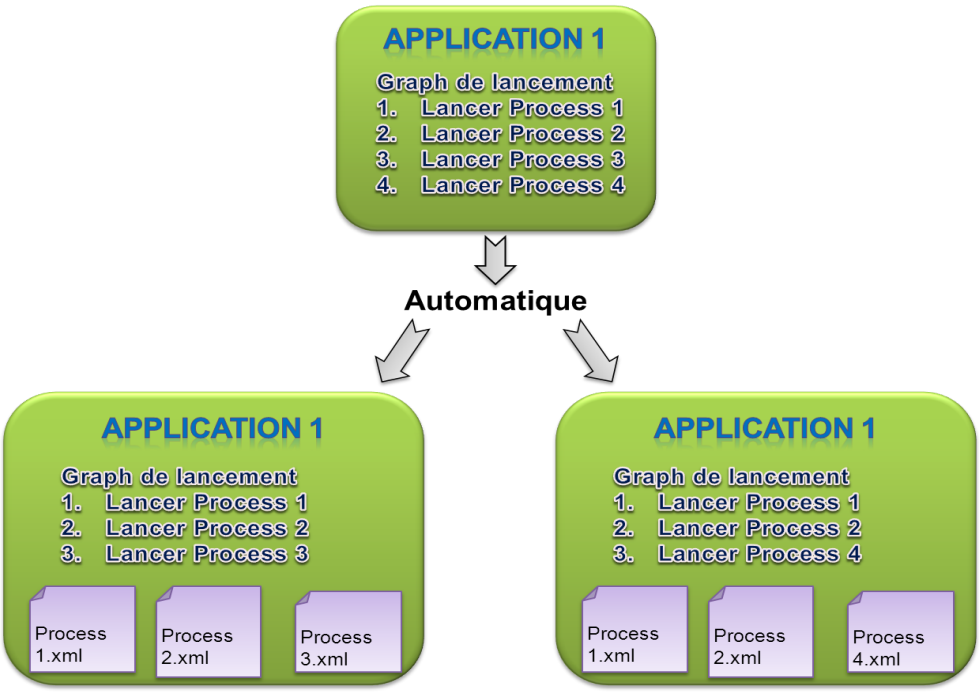


Figure 24: Fonctionnement du système plugin

V. 4. d) Système de mise à jour des fichiers XML automatiques

Le but de ce système est de pouvoir générer des fichiers de configuration de COCO2 adapté pour 4-Flight ou SESAR de manière quasi automatique. Afin de permettre cela sans avoir à modifier les fichiers un par un, j'ai opté pour la mise en place d'un programme Java. Celui-ci ne prenant en entrée qu'un seul fichier XML (**Voir Figure 25**). Ce fichier regroupe toutes les modifications à apporter aux différents fichiers utilisés par COCO2.

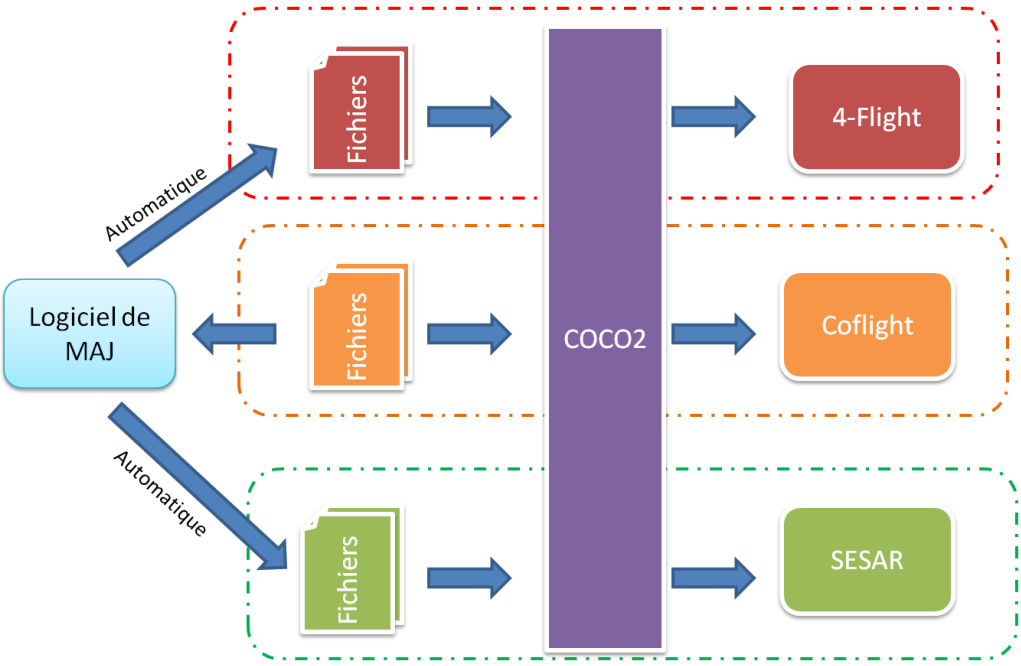


Figure 25: Système de mise à jour

Cette configuration permet donc en même temps de répondre au besoin en économie de temps car il n'y plus qu'un seul fichier à modifier pour impacter tous les autres. Et il permet aussi de réduire le taux d'erreurs pouvant être commises. Il est en effet bien plus facile de vérifier un unique fichier que plusieurs.

Dans le cadre de la réalisation de cet outil j'ai utilisé le langage Java et pour le fichier de mise à jour j'ai opté pour un XML.

L'utilisation du Java à la place d'autres langages, comme Python par exemple, a été une évidence pour moi. Je voulais garder une cohérence entre mon outil et tout l'environnement COCO2.

Quant au format XML, c'est le format de fichier qui m'est apparu être le plus performant pour le stockage et l'utilisation d'informations concernant les modifications. Un autre point est que cela me permet de réutiliser l'api DOM de Java que j'ai découvert durant ce stage. L'un des points les plus importants est que ce fichier sera modifié manuellement. Ainsi la forme d'arborescence d'un fichier XML permet à l'utilisateur de pouvoir lire ce fichier de manière rapide et plus agréable. L'utilisation de XML permet aussi, par rapport à un fichier texte par exemple, de pouvoir vérifier que ce fichier correspond bien au format décider en le validant par XSD.

.

V4.d.1 Architecture du fichier XML

Pour mettre en œuvre cette solution, j'ai d'abord réfléchis à la forme du fichier contenant les modifications à apporter. Après quelques réflexions je me suis arrêté sur l'arborescence de la Figure 26.

J'ai conçu cette architecture dans le but de pouvoir accéder à un élément dans un fichier XML le plus facilement possible. En essayant de trouver à chaque étape les informations pouvant être redondantes entre éléments de mêmes niveaux, afin de le regrouper dans un élément parent. Cela fait gagner du temps à l'utilisateur et permet d'avoir un fichier plus compréhensible en étant moins chargé.

Après quelques essais je me suis arrêté sur l'architecture suivante qui contient :

1. L'élément racine nommé « Maj » qui contient l'endroit où se trouve le fichier XSD pour le valider
2. Des éléments « Directory » correspondant aux dossiers dans lesquels se trouvent les fichiers à modifier.
3. Des éléments « File » correspondant au fichier à modifier
4. Des listes d'éléments « Add », « Delete », « Update » qui correspondent aux différentes actions à faire dans un fichier.
5. Des listes d'éléments « Element » correspondant à l'élément ou les éléments spécifiés par le « path » (voir chapitre suivant)

Plus spécifiquement les éléments « Element » changent en fonction de la modification à apporter. Ainsi lors d'un ajout « Element » contient une partie d'arborescence qui sera ajoutée aux éléments pointés.

Lors d'une suppression « Element » ne contient rien. Lors d'un update « Element » peut contenir des éléments « Attribute » ou « Del », qui permettent respectivement d'ajouter ou de supprimer un attribut. Et il peut contenir un élément « Text » permettant de remplacer le texte de l'élément.

Le principe détaillé est expliqué dans le chapitre suivant.

```xml
<?xml version="1.0" encoding="utf-8" standalone="no"?>
<Maj  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://www.omg.org/Integration/ integration.xsd">
    <File Path="M:/xml/Applications/CONFTL.xml">

        <Add unique="true" checkExist="true">

            <Element Path="ApplicationTypes/ApplicationType [Name=CONFTL]" >
                <Poperty jjj="ddddd"/>
            </Element>

            <Element Path="" >
                <ApplicationType Name="YOP"/>
            </Element>

        </Add>


        <Delete>
            <Element Path="Property [Mode=RO]" unique="true" />
        </Delete>

        <Update unique="true" >

            <Element Path="Stop-graph/Graph-element-root [Name=CONFTL.CONF_TA_CS]">
                <Attribute Name="Name" Value="22" />
                <Attribute Name="rrrrrrr" Value="valeur" />
                <Attribute Name="Asupprimer" Value="essai" />


                <Text>
                    texte !!
                </Text>

                <Del Name="Asupprimer"/>
            </Element>

        </Update>

    </File>
</Maj>
```

Figure 26: Fichier de mise à jour

V4.d.2 Fonctionnement du système

Une fois la structure du fichier de mise à jour bien définie, j'ai développé un programme qui prend en entrée ce fichier de mise à jour et les fichiers de configuration utilisés par COCO2 pour produire une configuration 4Flight ou SESAR.

Ce programme suit la démarche suivante :

1. Il va chercher dans le dossier « Directory » le fichier de nom « Name »
2. Il va ensuite effectuer les différentes commandes présentes qui peuvent être soit l'ajout (**Voir Annexe X** : Fonction Java d'ajout d'un élément dans un fichier XML), la suppression ou la mise à jour d'éléments.
3. Si l'attribut checkXSD est mentionné il va alors valider le fichier qu'il vient de modifier avec ce XSD.

A ce moment-là un problème est survenu. En effet je devais trouver un moyen de dire au logiciel sous quel élément il devait faire un ajout ou quel élément devait être supprimé ou mis à jour. Pour le résoudre j'ai inventé un type de « path » qui permet d'indiquer le chemin vers un ou plusieurs éléments. Ce path est construit ainsi :

```
Nom_element1 [Attribut1=Valeur, Attribut2=Valeur]/Nom_element2/ ...
```

Ainsi je parcours tous les éléments du fichier XML et je les compare à element1 et s'ils correspondent je recherche alors dans leurs éléments enfants un élément correspondant à element2 et ainsi de suite. Ceci me permet donc en fonction des attributs donnés de cibler un ou plusieurs éléments du fichier.

J'y ai ensuite rajouté deux options :

1. checkExist : Qui retourne une erreur si l'élément correspondant existe déjà.
2. unique : Qui retourne une erreur si le path entré peut correspondre à plusieurs éléments.

V. 4. e) Conclusion

Cette étape correspond aux deux phases de développement du planning qui ont durées 45 jours. Durant celle-ci j'ai pu, en discutant avec me tutrice, orienter mes choix sur la manière de séparer le fichier d'intégration. J'ai aussi pu approfondir mes connaissances en XSL et XSD, comme la mise en place de namespace en XSD ou résoudre des problèmes de "path" en XSL. Pour le système de mise à jour, j'ai dû mettre en avant mon autonomie et mes compétences techniques (en Java, XML, XSD), afin de pouvoir concevoir et réaliser ce programme du début à la fin.

V. 5. Gestion de configuration

Tout au long de ce stage j'ai suivi un process strict de gestion de configuration. La gestion de configuration permet de sauvegarder et de tracer chacun des changements que j'ai effectués sur le logiciel. Cela autorise de toujours pouvoir revenir à un environnement stable.

V. 5. a) Principe

Le principe de la gestion de configuration est le suivant : je dispose en local d'un dossier appelé *sandbox*. Cette sandbox contient l'ensemble de mon projet, à savoir les sources du logiciel, les fichiers de données, les scripts de compilation et d'installation, etc. Dès que je considère qu'un progrès notable a été effectué sur le logiciel, que ce progrès est testé et sans bug, je peux « commiter » (i.e. publier) l'ensemble de mes données modifiées.

Il existe par ailleurs un répertoire situé au cœur du système appelé *repository*. Le repository est la copie exacte de mon dossier sandbox. C'est dans ce repository, disponible de tous les intégrateurs, que l'on compile le logiciel et que les lanceurs du logiciel sont créés. Il suffit de mettre à jour le repository pour que tous les commits effectués depuis la (ou les) sandbox soient pris en compte.

Les intégrateurs n'ont alors qu'à mettre à jour leurs sandbox pour recevoir toutes les modifications effectuées.

V. 5. b) Avantages et inconvénients du système

Ce système permet deux choses : d'une part, il permet une gestion des versions du logiciel. En effet, à chaque commit est associé un tag qui permet de définir un état d'avancement de l'ensemble des fichiers et dossiers de la sandbox. Si on constate qu'une erreur est faite, on peut toujours obtenir cet état par l'intermédiaire de ce tag et revenir à un état précédent.

D'autre part, il permet de gérer la concurrence des changements pouvant être opérés par plusieurs développeurs. On est ainsi assurés que tous les changements seront pris en compte sans risque de perte de données liée à la concurrence.

Par contre, ce système de gestion de configuration nécessite des connaissances avancées : utilisation de commandes, gestions de version principales (dites en *trunk*) et de versions divergentes (dites en *branch*), etc.

V. 5. c) Généralisation au système Coflight

Tout le système Coflight suit cette gestion. Tous les fichiers de données du système sont « commités » au moindre changement, garantissant que chaque utilisateur travaille sur le système avec des données à jour.

Dès qu'une modification majeure (un ensemble de fichiers système, un ou plusieurs composants, etc. – les équipes d'intégration et de validation se mettent d'accord sur ce point) est opérée, l'ensemble des données du système sont figées et un nouveau BUILD est créé. La création d'un nouveau BUILD nécessite la recompilation de l'ensemble du système (i.e. de tous les composants et outils). La Figure 27 présente la décomposition des versions en BUILD. Les BUILD se suivent à l'intérieur d'une même version de Coflight.

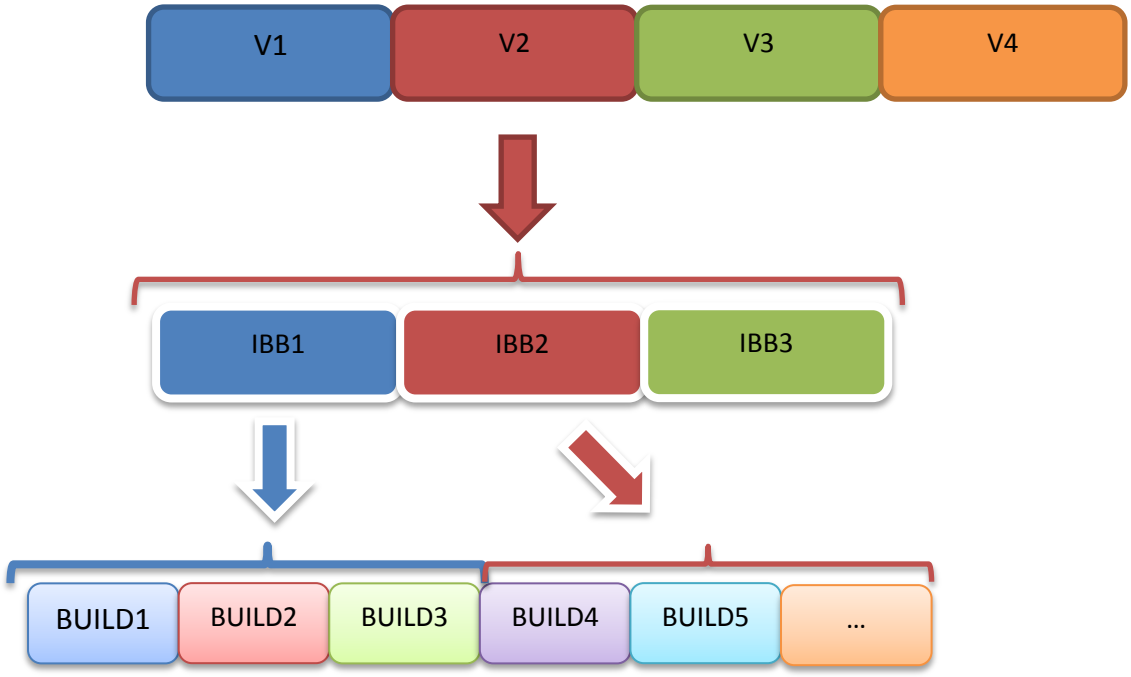


Figure 27 : Découpage des versions du système en BUILD

Durant mon stage ma tutrice m'a fourni une branche d'expérimentation où j'étais le seul à travailler dessus. Ceci m'a permis de pouvoir apporter mes modifications et d'effectuer mes tests sans pour autant impacter l'équipe d'intégration.

V. 6. Documentation

Pour permettre à de futurs utilisateurs de pouvoir plus facilement maintenir et utiliser le logiciel COCO2 j'ai réalisé la documentation de son code source. Cette documentation permet une meilleure lisibilité du code COCO2. Et elle facilite l'analyse de problèmes rencontrés en suivant le cheminement des actions.

V. 6. a) Choix du logiciel

Afin d'effectuer la documentation du logiciel COCO2 j'ai eu le choix entre deux outils. Ces outils déjà utilisés sur le projet Coflight sont Javadoc et Doxygen.

Javadoc :

Javadoc est un outil fourni par Sun avec le JDK pour permettre la génération d'une documentation technique à partir du code source.

Cet outil génère une documentation au format HTML à partir du code source Java et des commentaires particuliers qu'il contient.

Cette documentation contient :

- une description détaillée pour chaque classe et ses membres publics et protected par défaut (sauf les classes internes anonymes)
- un ensemble de listes (liste des classes, hiérarchie des classes, liste des éléments deprecated et un index général)
- une navigation entre ces différents éléments.

L'intérêt de ce système est de conserver dans le même fichier le code source et les éléments de la documentation qui lui sont associés. Il propose donc une auto-documentation des fichiers sources de façon standard.

Doxygen :

Doxygen est un outil de génération automatique de documentation à partir de code source annoté. La documentation est extraite directement des sources, ce qui rend beaucoup plus facile de tenir la documentation en accord avec le code source.

Doxygen permet de :

1. générer une documentation web (HTML) ou un manuel de référence en LATEX à partir du code documenté. Il supporte aussi les sorties en MS-WORD, PDF et en page UNIX.
2. extraire la structure du code. Ainsi il permet d'afficher les graphs de dépendance et les graphs d'héritage automatiquement.

Pour fonctionner Doxygen utilise des balises que l'on met en commentaire dans le code pour générer la documentation. Ainsi une fois lancé il parcourt les fichiers de code et lorsqu'il rencontre une balise qu'il connait il génère la documentation correspondante.

Solution retenu :

Le choix que j'ai retenu est le logiciel Doxygen. Mon choix c'est porté sur ce logiciel car, contrairement à Javadoc, il permet de supporter plusieurs langages de programmations. Et comme vu précédemment COCO2 est composé de plusieurs parties écrites avec langages différents. Ce qui permet au final d'avoir une unique documentation pour tout le logiciel.

V. 6. b) Réalisation

Pour réaliser cette documentation, j'ai modifié tous les fichiers de COCO2 pour implémenter les balises Doxygen (Voir Figure 28). Ainsi plus de 2000 lignes de commentaires ont été ajoutés durant cette phase dans le code source de COCO2.

Cette action étant manuelle, elle m'a pris beaucoup de temps pour lire chaque partie de code et comprendre leur utilité.

```
/**
 * @brief Check xml against xsd
 * @details If the \a file does not match with \a xsd an error is raised.
 * @see MyResourceResolver
 * @param file URL xml file that will be check.
 * @param xsd URL xsd file of this xml.
 *
 * @throws SAXParseException
 * @throws Throwable
 */
private static void checkXSD(URL file, URL xsd) {
```

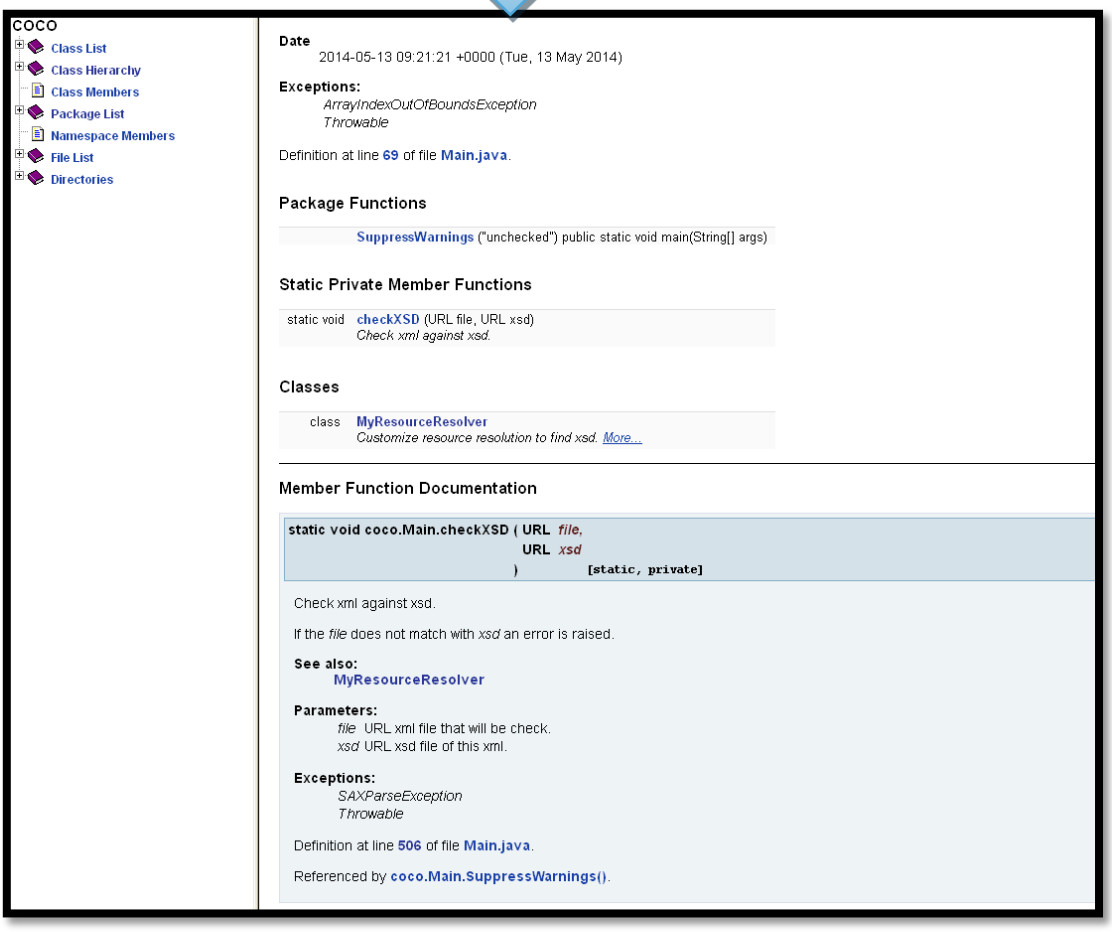


Figure 28: Documentation Doxygen

V. 6. c) Conclusion

Cette phase de 9 jours, est la dernière partie de la phase de documentation. Durant cette phase, j'ai beaucoup discuté avec les ingénieurs pour comprendre les fonctionnements de CADAMOM. J'ai pu ainsi décrire chaque classe de COCO2. J'ai aussi pu découvrir un nouveau logiciel de documentation, ne connaissant jusque-là que la Javadoc.

Conclusion

Le but de ce stage était d'automatiser le plus possible le logiciel COCO2. Afin de faciliter le passage des fichiers de configuration d'un Coflight à ceux d'un 4-Flight ou d'un SESAR.

Concernant les attentes du stage, les objectifs fixés au début du stage concernant les différents programmes ont été atteints dans leur totalité. Toutefois, vu le nombre important de classes dans COCO2, il reste encore quelques classes à documenter. Mais le plus gros du travail étant fait, le temps qui me reste pour finir le stage suffit largement pour achever le travail nécessaire.

Ce stage a parfaitement répondu à mes attentes car je souhaitais découvrir le domaine de gestion du contrôle aérien. Il m'a ainsi permis de découvrir un univers que je ne connaissais pas mais pour lequel j'y ai porté un grand intérêt.

Pendant ces 6 mois j'ai pu avoir un aperçu d'un projet de très grande envergure et de ce qu'est le métier d'intégrateur.

Durant ce stage j'ai pu mettre en pratique mes connaissances théoriques mais aussi pratiques acquises durant ma formation ce qui a joué un rôle important dans le bon déroulement du stage. En effet j'ai pu utiliser le langage JAVA, pour la première fois dans un cadre professionnel, que j'ai appris lors de ma formation. J'ai aussi pu approfondir mes connaissances en XSLT et XSD.

Il m'a fallu aussi pour mener à bien les différentes parties du stage mettre en avant mes capacités d'analyses afin de comprendre le fonctionnement d'un programme complexe comme COCO2.

J'ai aussi fait preuve d'autonomie tout au long du stage, pour ne pas perturber le travail de ma tutrice ou de l'équipe. Même si ils n'hésitaient pas à prendre du temps pour répondre à mes questions lorsque c'était nécessaire.

Pour finir je dirais que j'ai trouvé ce stage très intéressant et enrichissant puisqu'il m'a permis d'attirer mon attention sur l'importance de la gestion de configuration ou sur l'utilisation de logiciel sous licence. Ces notions peuvent apparaitre comme inutiles lors de projets personnels, mais elles prennent toutes leur importance pour des projets commerciaux aussi importants.

Ce stage m'aura aussi permis de faire des rencontres très enrichissantes tant sur le plan humain que sur le plan professionnel. J'ai pu ainsi rencontrer des gens avec des expériences et des profils différents. Cela m'a permis de pouvoir envisager d'autres perspectives pour mon avenir professionnel.

Table des abréviations

Sigle	Signification
ADS-B	Automatic Dependent Surveillance - Broadcast
ANSP	Air Navigation Service Provider (Prestataire de services de la navigation aérienne)
ATC	Air Traffic Control
ATM	Air Traffic Management
CCM	CORBA Component Model
COCO	Cardamom Offline Configurator
COOPANS	Cooperation between ANSP
CORBA	Common Object Request Broker Architecture
CPR	Common Product
CSCI	Computer Software Configuration Item
DDS	Middleware Data Distribution Service
DGAC	Direction Générale de l'Aviation Civile (ANSP Français)
DOM	Document Object Model
DSNA	Direction des Services de la Navigation Aérienne
eFDP	Eurpean Flight Data Processing
ENAV	Ente Nazionale di Assistenza al Volo (ANSP Italien)
IOP	InterOPerability
OS	Operating System
OMG	Object Management Group
PBS	Project Break down Structure
PCV	Package Configuration Version
PFE	Projet de Fin d'Etudes
PSR	Primary Surveillance Radar
SES	Single European Sky
SESAR	Single European Sky ATM Research
SPV	Software Package Version
SSR	Secondary Surveillance Radar
STPV	Système de Traitement de Plan de Vol
UML	Unified Modeling Language
W3C	World Wide Web Consortium
XML	Extensible Markup Language
XSD	XML Schema Definition
XSLT	eXtensible Stylesheet Language Transformations

Glossaire

Terme	Définition
Evénements (Source/Sink)	Messages asynchrones entre composants.
Middleware	Logiciel qui crée un réseau d'échange d'informations entre différentes applications informatiques. Il assure la communication entre les applications quels que soient les caractéristiques matérielles, les protocoles réseau ou les systèmes d'exploitation impliqués.
Node	Machine physique capable d'exécuter un OS (Windows/Linux) et un middleware).
Réceptacle/Facet	Interface permettant aux composants de communiquer de manière synchrone en se connectant les uns aux autres.
Parser	Parcourir un document
Reverse engineering	Activité qui consiste à étudier un objet pour en déterminer le fonctionnement interne ou sa méthode de fabrication.
Process	Programme d'un logiciel identifié par l'OS.
CCM	Instance d'un composant CORBA.
PCV	PCV permet de rassembler les librairies et les exécutables (SPV) nécessaires à l'exécution d'un système sur une machine ou un ensemble de machine
SPV	Librairies et exécutables d'un CSCI nécessaire pour les installer sur une machine
OMG	Organisme se chargeant de standardiser et de promouvoir le modèle objet sous toutes ses formes.
W3C	Organisme de standardisation chargé de promouvoir la compatibilité du World Wide Web (HTML et XML par exemple)
Plugin	Paquet qui complète un logiciel hôte pour lui apporter de nouvelles fonctionnalités.

Sources documentaires

Livre :

Titre	Auteur	Edition
Ant: précis & concis	**Stefan Edlich**	O'Reilly, 2002
CPR – System/Subsystem description for Coflight eFDP	**THALES et SELEX ES**	

Sites Internet :

Intranet THALES

http://fr.wikipedia.org/

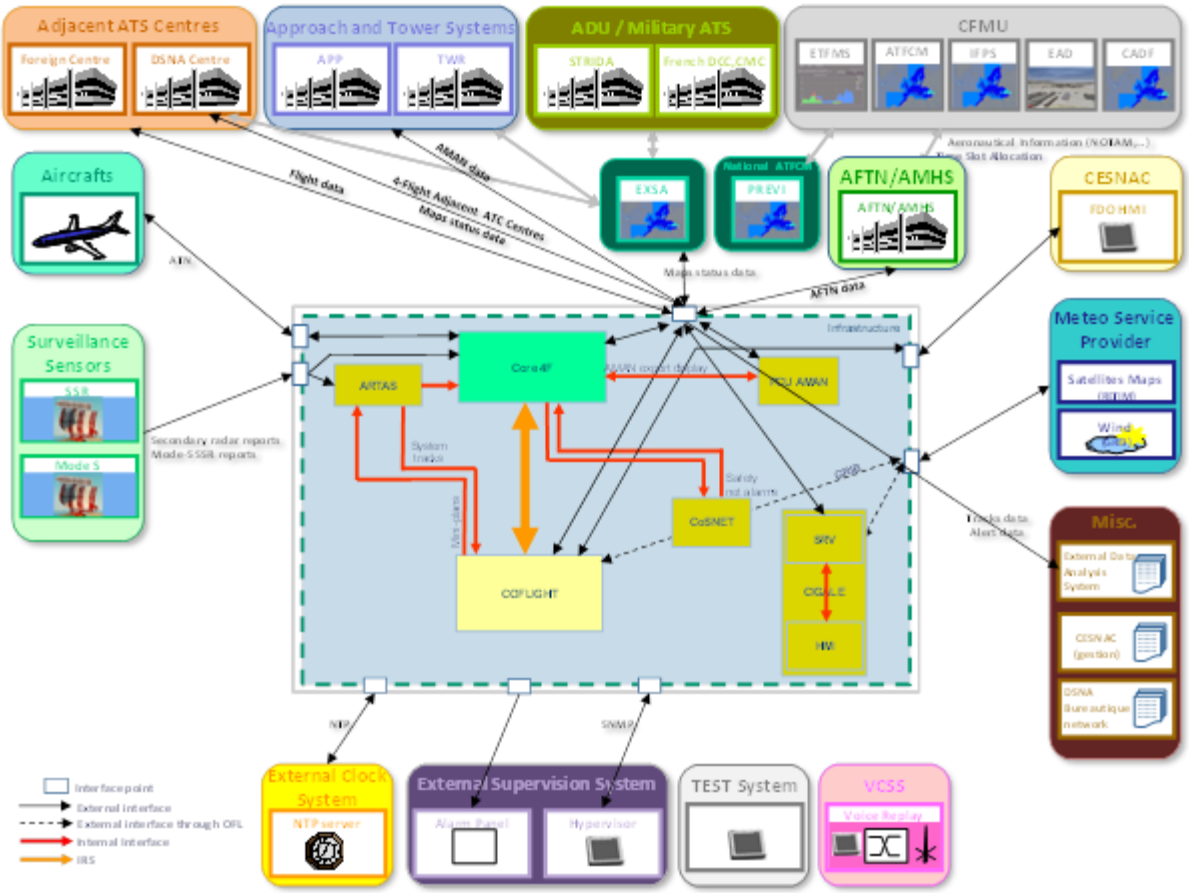
http://java.developpez.com/

http://stackoverflow.com

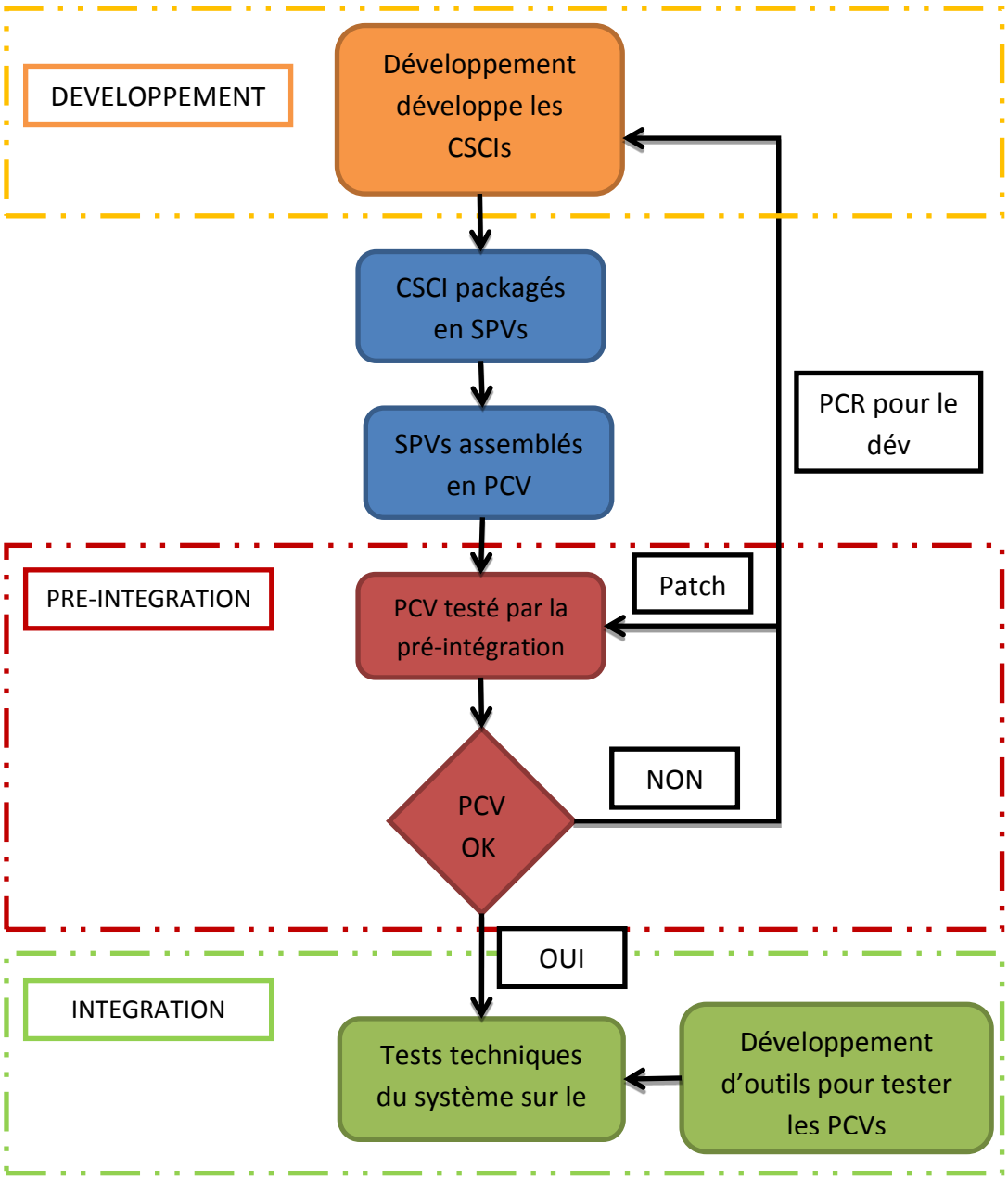
https://www.google.fr

Annexes

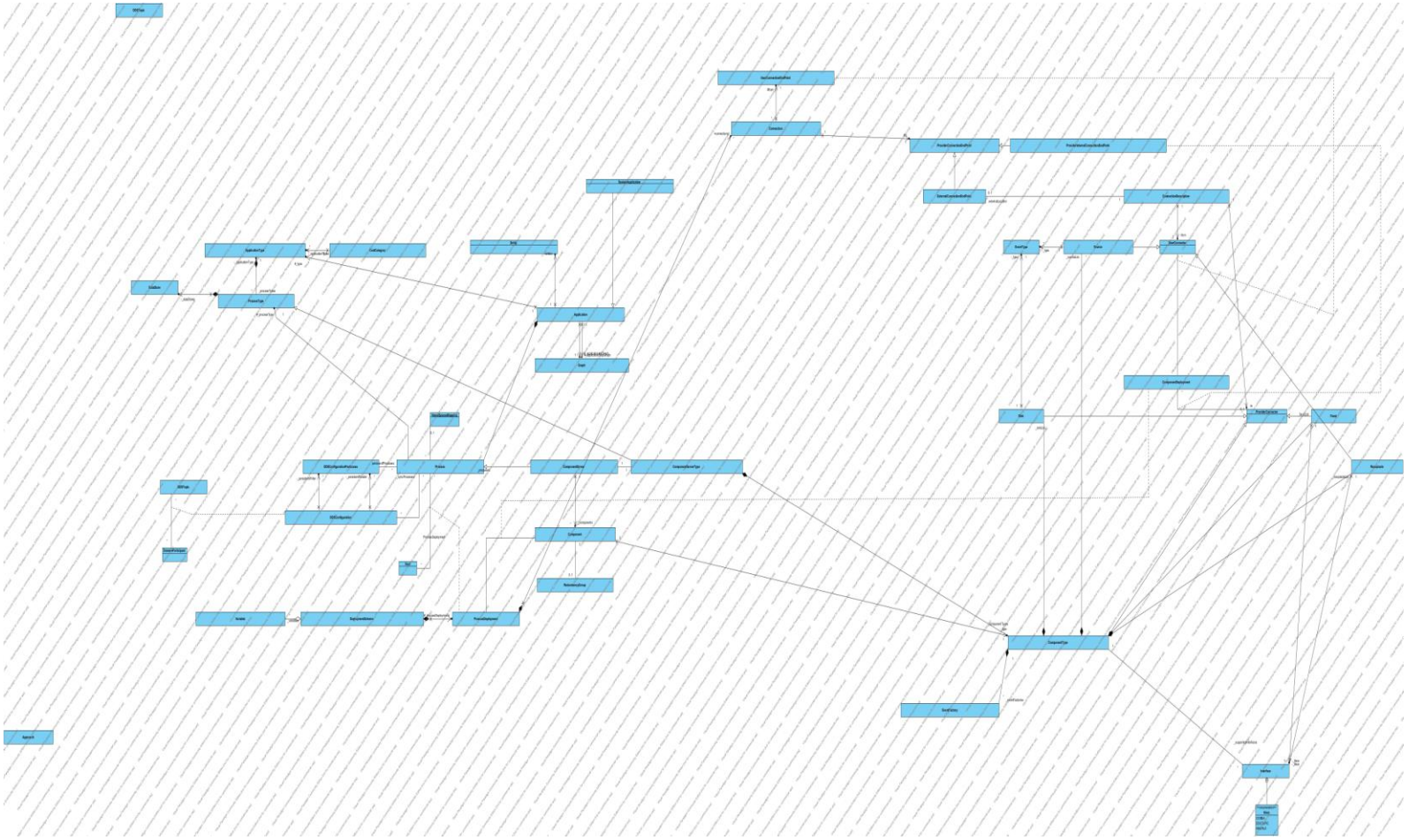
Annexe I : Vue interne du système 4-Flight



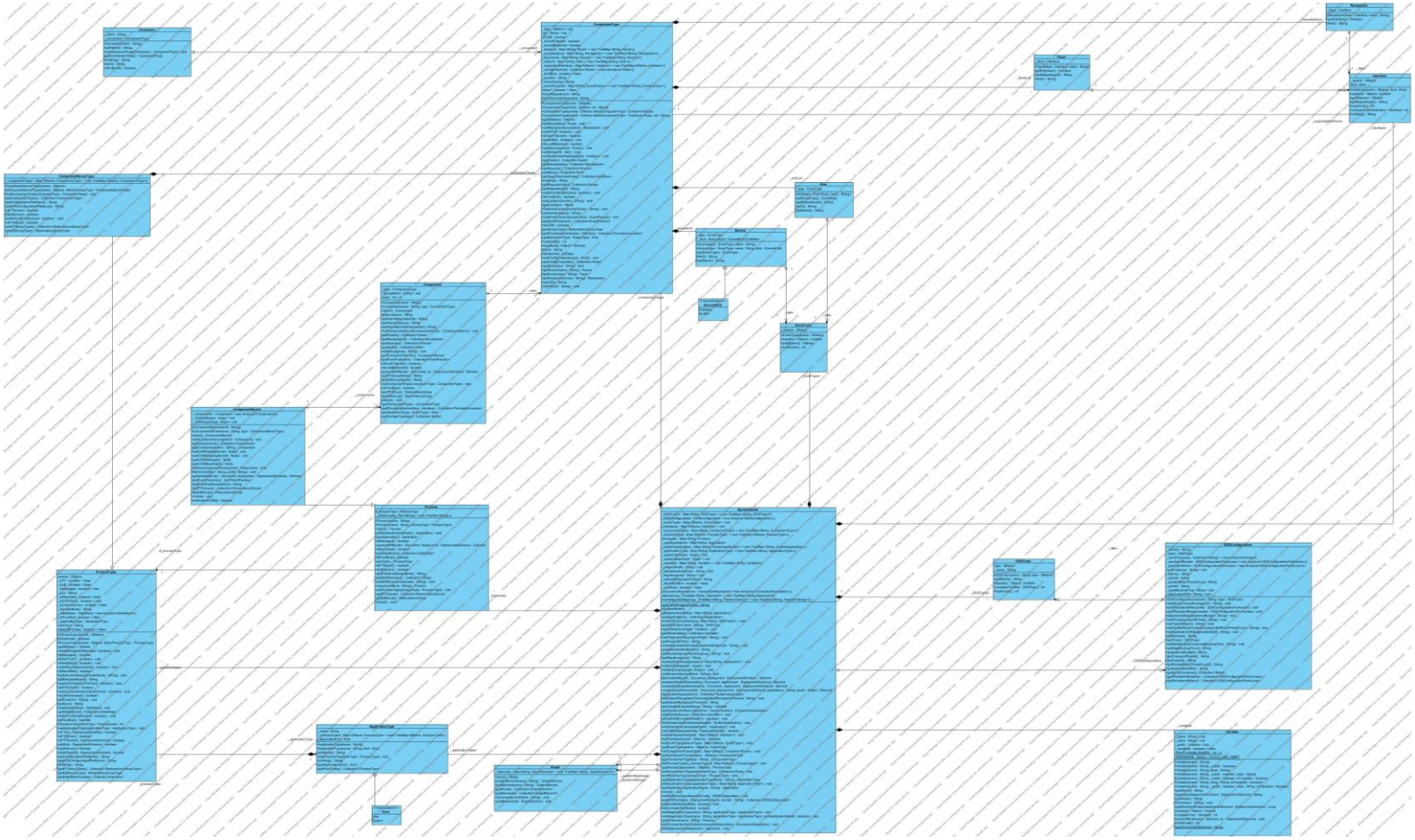
Annexe II : Présentation schématique des rôles de la pré-intégration et de l'intégration



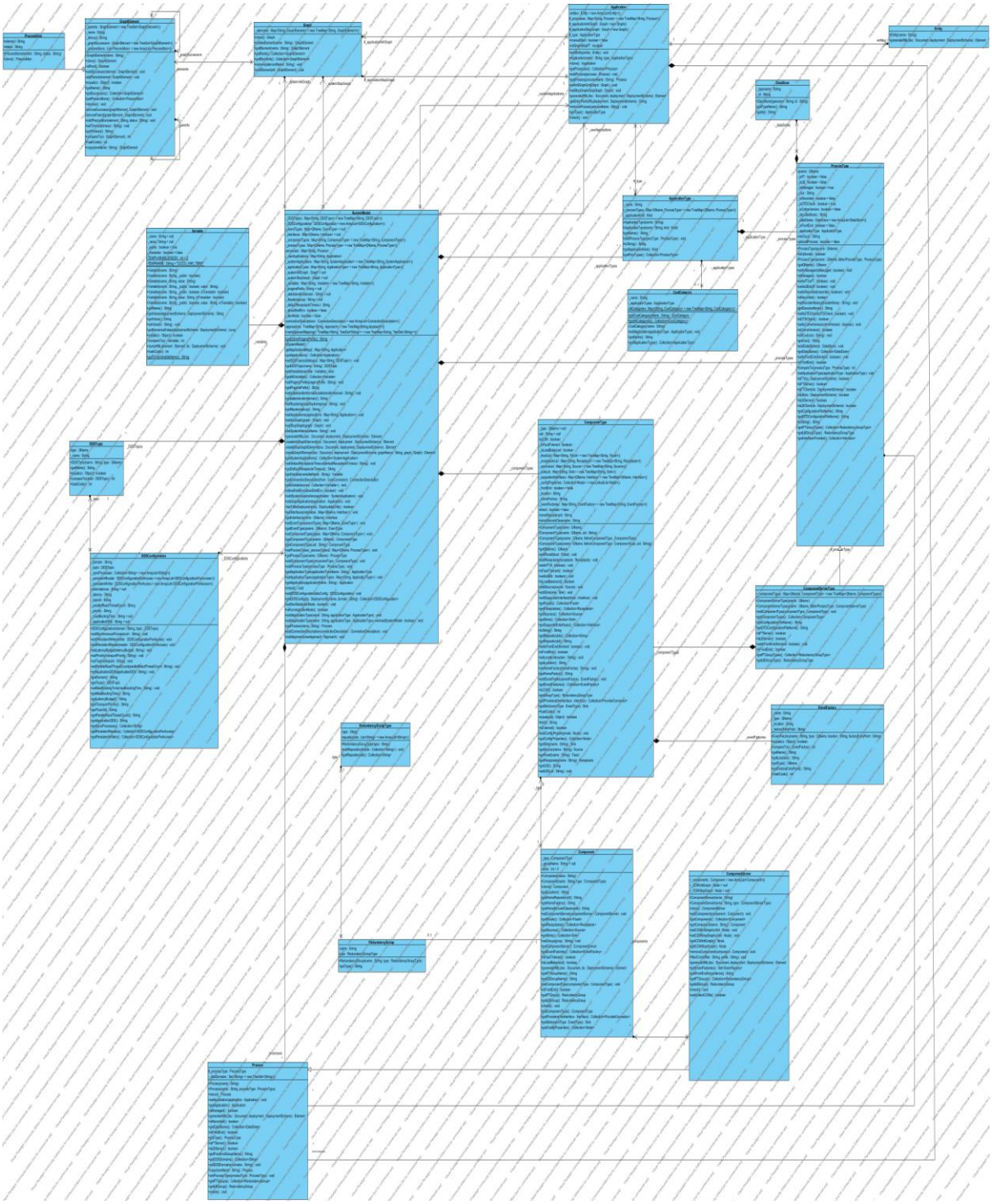
Annexe III : Diagramme de classes complet



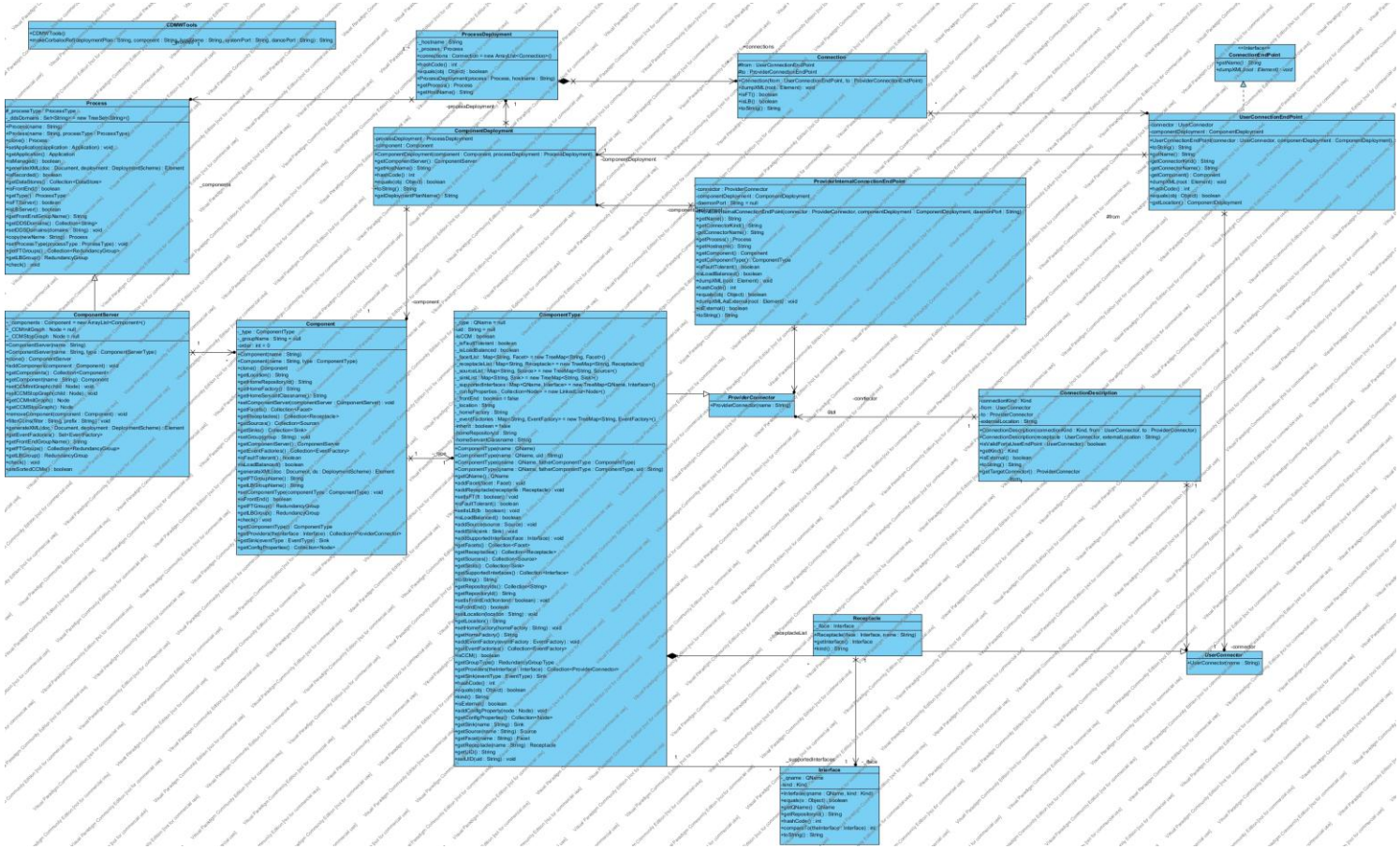
Annexe IV : Diagramme du système model



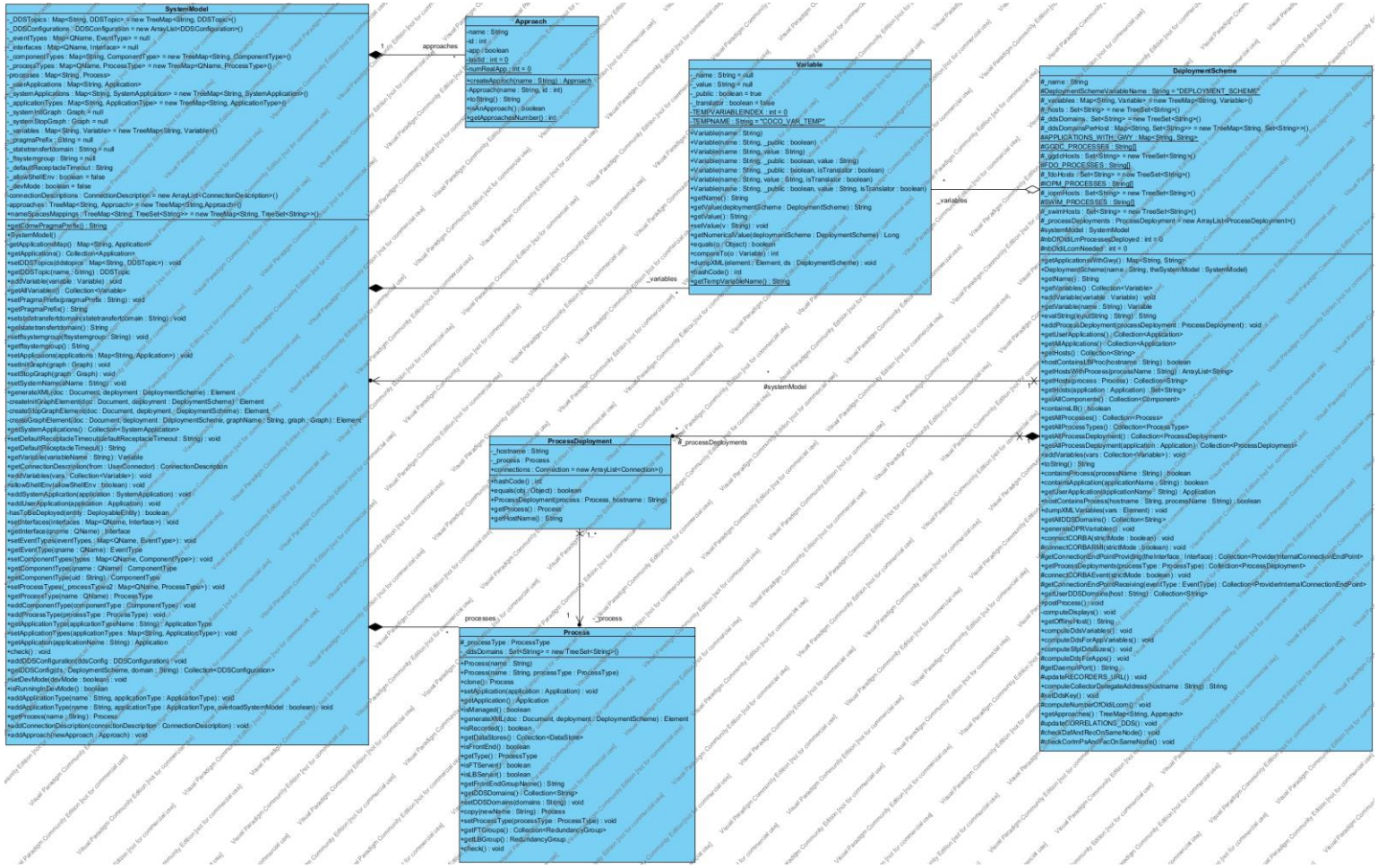
Annexe V : Diagramme des applications



Annexe VI : Diagramme des connexions CORBA



Annexe VII : Diagramme de déploiement CSCI



Annexe VIII : XSLT Pour enlever les process du fichier d'intégration

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
    <xsl:output method="xml" indent="yes"/>
    <xsl:template match="/" >
        <my:IntegConf xsi:schemaLocation="http://www.omg.org/Integration/ integration.xsd" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:my="http://www.omg.org/Integration">

            <xsl:text>&#xA;  </xsl:text><xsl:copy-of select="IntegConf/statetransfertdomain" />
            <xsl:text>&#xA;  </xsl:text><xsl:copy-of select="IntegConf/ftsystemgroup" />
            <xsl:text>&#xA;  </xsl:text><xsl:copy-of select="IntegConf/default-receptacle-timeout" />
            <xsl:text>&#xA;  </xsl:text><xsl:copy-of select="IntegConf/System" />
            <xsl:text>&#xA;  </xsl:text><xsl:copy-of select="IntegConf/ComponentTypes" />
            <xsl:text>&#xA;  </xsl:text><xsl:copy-of select="IntegConf/ProcessTypes" />

            <xsl:text>&#xA;  </xsl:text><ApplicationTypes>
                <xsl:for-each select="IntegConf/ApplicationTypes/ApplicationType">
                    <xsl:text>&#xA;    </xsl:text>
                    <xsl:copy>
                        <xsl:apply-templates select="@*"/>
                        <xsl:for-each select="node()[local-name()!='ProcessType']">
                            <xsl:choose>
                                <xsl:when test="self::text()">
                                </xsl:when>
                                <xsl:otherwise>
                                    <xsl:text>&#xA;      </xsl:text>
                                    <xsl:copy-of select="." />
                                </xsl:otherwise>
                            </xsl:choose>
                        </xsl:for-each>
                        <xsl:if test="./*[local-name()!='ProcessType']">
                                <xsl:if test="./* != text()">
                                    <xsl:text>&#xA;    </xsl:text>
                                </xsl:if>
                        </xsl:if>
                    </xsl:copy>
                </xsl:for-each>
            <xsl:text>&#xA;  </xsl:text></ApplicationTypes>

            <xsl:text>&#xA;  </xsl:text><Applications>
                <xsl:for-each select="IntegConf/Applications/Application">
                    <xsl:text>&#xA;    </xsl:text>
                    <xsl:copy>
                        <xsl:apply-templates select="@*"/>
                        <xsl:for-each select="node()[local-name()!='Managed-process' and local-name()!='Unmanaged-process'] " >
                            <xsl:choose>
                                <xsl:when test="self::text()">
                                </xsl:when>
                                <xsl:otherwise>
                                    <xsl:text>&#xA;      </xsl:text>
                                    <xsl:copy-of select="." />
                                </xsl:otherwise>
                            </xsl:choose>
                        </xsl:for-each>
                        <xsl:if test="./*[ local-name()!='Managed-process']">
                            <xsl:if test="./*[local-name()!='Unmanaged-process']">
                                <xsl:if test="./* != text()">
                                    <xsl:text>&#xA;    </xsl:text>
                                </xsl:if>
                            </xsl:if>
                        </xsl:if>
                    </xsl:copy>
                </xsl:for-each>
            <xsl:text>&#xA;  </xsl:text></Applications>
            <xsl:text>&#xA;  </xsl:text><xsl:copy-of select="IntegConf/DDS" />

        <xsl:text>&#xA;</xsl:text></my:IntegConf>

    </xsl:template >

    <xsl:template match="text()"></xsl:template>

    <xsl:template match="@* | node()">
        <xsl:copy>
            <xsl:apply-templates select="@* | node()"/>
        </xsl:copy>
    </xsl:template>

</xsl:stylesheet>
```

Annexe IX : XSD de validation des fichiers de process

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:Deployment="http://www.omg.org/Deployment" xmlns:integ="http://www.omg.org/Integration" xmlns="http://www.omg.org/Process"
 elementFormDefault="unqualified" targetNamespace="http://www.omg.org/Process">
    <!--
COCO_COPYRIGHT_TO_BE_REPLACED
-->
    <!--
  $Author: Thales $
  $Date: 2014-01-17 10:25:53 +0000 (Fri, 17 Jan 2014) $
  $HeadURL: file:///opt/INTEG_SVN/COCO2/branches/Experimentations/coco/code/java/resources/main/xsd/integration.xsd $
  $Revision: 4149 $
-->
    <xs:import namespace="http://www.omg.org/Deployment" schemaLocation="Deployment.xsd" />
    <xs:import namespace="http://www.omg.org/Integration" schemaLocation="integration.xsd" />

    <xs:element name="IntegConf" type="T_IntegConf">
        <xs:unique name="Process">
            <xs:selector xpath="./Processes/Managed-process | ./Processes/Unmanaged-process| ./Applications/Application/Managed-process | ./Applications/Application/Unmanaged-process"/>
            <xs:field xpath="@Name"/>
        </xs:unique>
        <!-- <xs:unique name="Component">
            <xs:selector xpath="./Components/Component | ./Processes/Managed-process/Ccm-component |./Applications/Application/Managed-process/Ccm-component"/>
            <xs:field xpath="@Name"/>
        </xs:unique> -->
    </xs:element>

    <xs:complexType name="T_IntegConf">
        <xs:sequence maxOccurs="1">
            <xs:element name="ApplicationTypes" type="T_ApplicationTypes"/>
            <xs:element name="Applications" type="T_Applications"/>
        </xs:sequence>
    </xs:complexType>

    <xs:complexType name="T_ApplicationTypes">
        <xs:sequence maxOccurs="1">
            <xs:element name="ApplicationType" type="T_ApplicationType"/>
        </xs:sequence>
    </xs:complexType>


    <xs:complexType name="T_ApplicationType">
        <xs:sequence>
            <xs:choice maxOccurs="1">
                <xs:element name="ProcessType" type="integ:T_ProcessType" />
            </xs:choice>
        </xs:sequence>
        <xs:attribute name="Name" type="xs:string" />
        <xs:attribute name="InSystemModel" type="xs:boolean" />
        <xs:attribute name="Kind">
            <xs:simpleType>
                <xs:restriction base="xs:string">
                    <xs:enumeration value="User" />
                    <xs:enumeration value="System" />
                </xs:restriction>
            </xs:simpleType>
        </xs:attribute>
        <xs:attribute name="OverloadSystemModel" type="xs:boolean"></xs:attribute>
    </xs:complexType>


    <xs:complexType name="T_Applications">
        <xs:sequence maxOccurs="1">
            <xs:element name="Application" type="T_Application"/>
        </xs:sequence>
    </xs:complexType>


    <xs:complexType name="T_Application">
        <xs:sequence>
            <xs:sequence maxOccurs="1">
                <xs:choice>
                    <xs:element name="Managed-process" type="integ:T_Managed-process" minOccurs="0" maxOccurs="unbounded" />
                    <xs:element name="Unmanaged-process" type="integ:T_Unmanaged-process" minOccurs="0" maxOccurs="unbounded" />
                </xs:choice>
            </xs:sequence>
        </xs:sequence>
        <xs:attribute name="Name" type="xs:string" use="required" />
        <xs:attribute name="InSystemModel" type="xs:boolean" use="required" />
    </xs:complexType>

</xs:schema>
```

Annexe X : Fonction Java d'ajout d'un élément dans un fichier XML

```
// let assume that racine and nodeToAdd belong to the same document object
public static int add(Node racine, String NodePath, Node nodeToAdd, boolean unique, boolean checkExist) throws Exception {
    int nbrowadded = 0;
    if (nodeToAdd.getNodeType() != Node.TEXT_NODE) {
        ArrayList<Node> listnode = new ArrayList<Node>();

        if (NodePath != null && NodePath.equals("")) {
            NodePath = nodeToAdd.getNodeName();

            listnode = DOMUtils.find(racine, NodePath, false);
            if (listnode.size() == 0) {
                throw new Exception("Not able to find node with same name or path :" + NodePath + ". Please check the node name or enter an node path");
            }
            Node parent = listnode.get(0).getParentNode();
            // System.out.println(listnode.get(0).getNodeName() +
            // " kkkkkkkkkkkkkkkkkkkkk" + NodePath);
            Node child = parent.getFirstChild();

            // Check if if node to add already exist
            boolean find = false;
            while (child != null) {
                if (DOMToolsLike.compareNodes(nodeToAdd, child, true, false, false)) {
                    find = true;
                    if (checkExist) {
                        throw new Exception("Node already exist with path : " + getNodePath(child));
                    }
                    System.out.println("Node already exist with path : " + getNodePath(child));
                }
                child = child.getNextSibling();
            }
            if (!find) {

                Node nd = nodeToAdd.cloneNode(true);
                parent.appendChild(nd);
                nbrowadded++;
                String st = getNodePath(nd);
                System.out.println("Node " + st.split(nodedelmiter)[st.split(nodedelmiter).length - 1] + " added");
            }

            for (Iterator iterator = listnode.iterator(); iterator.hasNext();) {

                Node currentParent = ((Node) iterator.next()).getParentNode();
                if (!DOMToolsLike.compareNodes(parent, currentParent, true, false, false)) {
                    if (unique) {
                        throw new Exception("The same type of node (" + NodePath + ") have different parents.\n Parent1 : " + getNodePath(parent) + "\n Parent2 : " + getNodePath(current
                    }
                    System.out.println("The same type of node (" + NodePath + ") have different parents.\n Parent1 : " + getNodePath(parent) + "\n Parent2 : " + getNodePath(currentParer
                    parent = currentParent; Node child2 = parent.getFirstChild();
                    // Check if if node to add already exist
                    find = false;
                    while (child2 != null) {
                        if (DOMToolsLike.compareNodes(nodeToAdd, child2, true, false, false)) {
                            find = true;
                            if (checkExist) {
                                throw new Exception("Node already exist with path : " + getNodePath(child2));
                            }
                        }
                        child2 = child2.getNextSibling();
                    }
                    if (!find) {
                        Node nd = nodeToAdd.cloneNode(true); parent.appendChild(nd); nbrowadded++; String st = getNodePath(nd);
                        System.out.println("Node " + st.split(nodedelmiter)[st.split(nodedelmiter).length - 1] + " added");
                    }
                }
            }
        } else {
            listnode = DOMUtils.find(racine, NodePath, unique);
            if (listnode.size() == 0) (throw new Exception("Not able to find node with same name or path :" + NodePath + ". Please check the node name or enter an node path");)
            for (Iterator iterator = listnode.iterator(); iterator.hasNext();) {
                Node parent = (Node) iterator.next(); Node child = parent.getFirstChild();
                boolean find = false;
                while (child != null) {
                    if (DOMToolsLike.compareNodes(nodeToAdd, child, true, false, false)) {
                        find = true;
                        if (checkExist) {
                            throw new Exception("Node already exist with path : " + getNodePath(child));
                        }
                        System.out.println("Node already exist with path : " + getNodePath(child));
                    }
                    child = child.getNextSibling();
                }
                if (!find) {
                    Node nd = nodeToAdd.cloneNode(true);
                    parent.appendChild(nd);
                    nbrowadded++;
                    String st = getNodePath(nd);
                    System.out.println("Node " + st.split(nodedelmiter)[st.split(nodedelmiter).length - 1] + " added");
                }
            }
        }
    } else {
        throw new Exception("Can't add text type node. See update function.");
    }
    return nbrowadded;
}
```

Page laissée blanche intentionnellement

Résumé

Coflight est un système de nouvelle génération de système de traitement de plans de vol, visant à replacer ceux utilisés actuellement en France, en Italie et en Suisse.

4-Flight et SESAR sont quant à eux deux autres projets se déroulant en parallèle de Coflight. Le premier consiste à remplacer le système de contrôle aérien français par un système de nouvelle génération. Et le deuxième est le développement d'une infrastructure de gestion du trafic aérien au niveau européen.

Le point commun entre c'est projet est Coflight qui est au cœur des deux autres systèmes.

COCO2 permet de configurer Coflight et d'autres composants utilisant CARDAMOM. COCO2 est un « produit » qui a un code commun à tous les projets. Seuls ses fichiers d'entrées sont modifiés en fonction des environnements différents.

L'objectif de ce stage est de faciliterla prise en compte d'un nouveau Coflight dans des environnements différentsen automatisant la modification des fichiers de configuration. Cette automatisation est nécessaire pour augmenter la productivité de l'équipe d'intégration et pour diminuer les risques d'erreurs.

Dans ce rapport, je présente tout l'environnement au sein duquel le stage s'est déroulé. Enfin, j'expose les différentes phases d'analyse et de réalisation du projet. Avec les problèmes et contraintes rencontrés ainsi que leurs solutions retenues.

Dans la conclusion, j'analyse l'impact de ce stage sur mon enrichissement personnel et professionnel.

Mots-clés

CARDAMOM, Coflight, 4-Flight, SESAR, CORBA, intégration, Java, middleware, scripts Shell, XML, XSD, XSLT, Doxygen, Visual Paradigm.

Page laissée blanche intentionnellement

CONFIDENTIEL