El Bouzidi Aziz
Licence professionnelle SIL
Option qualité du logiciel

# Stage de fin d'études : Amélioration de la qualité du code d'un logiciel de réseau de Petri

Responsables : Catherine Tessier
Charles Lesire
Olivier Bonnet-Torrès

Université Paul Sabatier TOULOUSE III

ONERA

2005-2006

# *Remerciements*

Le stage présenté dans ce rapport a été effectué à l'ONERA-CERT, centre de Toulouse, dirigé par Monsieur Jean-Pierre Jung que je tiens à remercier pour son accueil.

Je remercie également Monsieur Patrick Fabiani directeur du département Commande des Systèmes et Dynamique du vol (DCSD) pour m'avoir reçu dans son équipe.

Je tiens à remercier Madame Catherine Tessier, ma responsable de stage, pour avoir facilité mon intégration au sein de ce groupe.

Je tiens aussi à remercier Charles Lesire et Olivier Bonnet-Torrès, co-responsables de stage, pour l'aide précieuse qu'ils m'ont apportée tout au long de ce projet.

Un grand merci également à toutes les personnes du département DCSD pour leur accueil et tout particulièrement les doctorants ainsi que les stagiaires pour leur bonne humeur quotidienne.

# Sommaire

# *Introduction*

Le programme de formation de la licence professionnelle Systèmes Informatiques et Logiciels s'articule autour de deux axes : une formation théorique dispensée à l'université et une formation pratique sous forme de stage professionnel de 4 mois.

Ce stage permet à l'étudiant d'acquérir une expérience de terrain durant laquelle il applique ses connaissances théoriques et acquiert de nouvelles connaissances plus « pratiques » qui lui seront utiles une fois entré dans la vie active.

Pour ma part, j'ai décidé d'effectuer mon stage à l'Office National d'Etudes et de Recherches Aérospatiales puisque la mission qui m'a été proposée me paraissait à la fois enrichissante et formatrice : j'ai intégré le Département Commandes des Systèmes et Dynamique du vol.

De part ces objectifs ce stage correspondait tout à fait à mes attentes professionnelles.

La mission qui m'a été confiée concernait la modification du code source d'un logiciel de modélisation des réseaux de Petri, pour en faire un outil délivrable présentant des gages de qualité.

Je vais donc m'attacher à vous présenter cette opération : sa méthodologie, ses principaux résultats et son bilan.

# I. Présentation de L'ONERA

## 1) L'ONERA :

L'ONERA, Office National d'Etudes et de Recherches Aérospatiales, est le premier acteur de la recherche aérospatiale en France. Etablissement public à caractère industriel et commercial, il a pour mission d'orienter et de conduire les recherches et de les valoriser pour l'industrie aérospatiale. Il gère aussi le premier parc européen de souffleries. 2000 personnes, dont plus de 1000 scientifiques sont repartis dans huit centres en France. Leur créativité se révèle dans les domaines porteurs (radar, optique, commande de systèmes…), sur des projets scientifiques internationaux (Very Large Telescope, propulsion pour les lanceurs de nouvelle génération, avions du futur comme l'aile volante ou les avions sans pilote…).


***Figure 1 :*** *les centres de l'ONERA en France*

Avec un budget annuel de 35M euros, l'ONERA assure annuellement 300 contrats d'études. Il participe aux programmes de recherche nationaux émanant du ministère de la Défense, du CNES… Il est impliqué dans les programmes européens et dans les activités de l'Agence Spatiale Européenne (ESA). Il participe à un certain nombre d'expériences internationales à caractère scientifique ou technologique, principalement dans le domaine spatial. L'ONERA propose également ses compétences aux industriels des secteurs de l'énergie, des télécommunications et des transports terrestres.

Les activités de l'ONERA reposent d'une part sur la Direction des Grands Moyens Techniques (GMT) et d'autre part sur 17 départements et un laboratoire mixte regroupés dans les quatre branches de la Direction Scientifique Générale (DSG) :

- Mécanique des Fluides et Energétique (MFE)
- Matériaux et Structures (MAS)
- Physique (PHY)
- Traitement de l'Information et Systèmes (TIS) dont fait partie le DCSD.

C'est au sein du département DCSD que s'est déroulé ce projet de fin d'études.

## 2) Le Département Commande des Systèmes et Dynamique du vol (DCSD) :

Le site de Toulouse créé en 1968 sous l'appellation Centre d'Etudes et de Recherches de Toulouse (CERT) est intégré à l'ONERA.

Son potentiel humain est de 420 personnes dont 280 ingénieurs et techniciens de recherche permanents et 90 ingénieurs doctorants pour une durée de trois ans. Il accueille également chaque année plus de 100 stagiaires en cours d'études.

Le centre de Toulouse compte 3 branches différentes divisées en 7 départements dont certains sont multi sites.

| Branche | Département |
|---|---|
| MFE | Modèles pour l'Aérodynamique et l'Energétique (DMAE) |
| PHY | ElectroMagnétisme et Radars (DEMR)<br>Environnement Spatial (DESP)<br>Optique Théorique et Appliquée (DOTA) |
| TIS | **Commande des Systèmes et Dynamique du vol (DCSD)**<br>Prospective et Synthèse (DPRS)<br>Traitement de l'Information et Modélisation (DTIM) |

Le DCSD a pour objectif la maîtrise des comportements des systèmes dynamiques complexes. Il a pour mission principale de développer des recherches pour maintenir une grande expertise en commande, en conduite de systèmes, en mécanique du vol, en analyse de qualités et en interfaçage homme-machine.

Le département est décomposé en 3 domaines de compétences (systèmes décisionnels, commande, dynamique du vol et identification des systèmes).

Ces trois domaines réunis dans un même département permettent de renforcer le degré d'efficacité, d'innovation et d'excellence dans cette spécialité.

# II. Contexte de l'étude

## 1) Les réseaux de Petri :

### a. Définition :

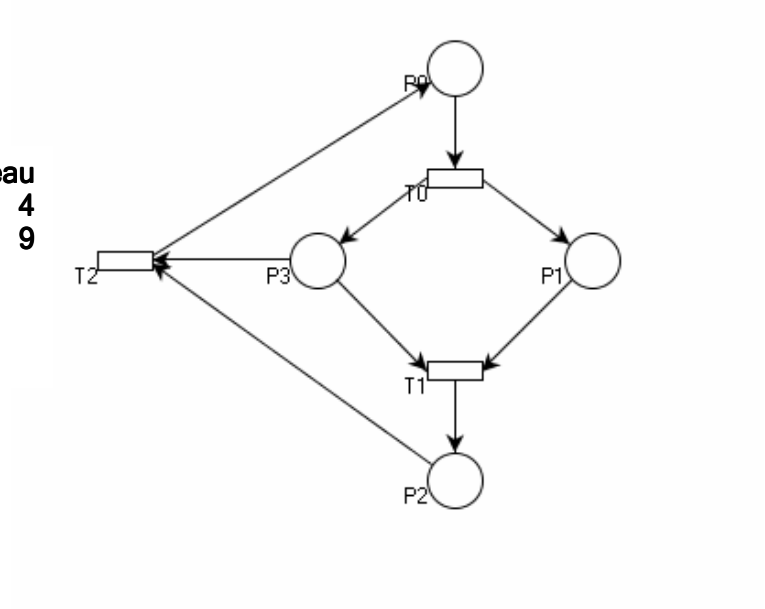
Un réseau de Petri est un modèle mathématique servant à représenter divers systèmes (informatiques, industriels). Il est défini par :

- un ensemble de places, notées graphiquement par des cercles ;
- un ensemble de transitions, notées graphiquement par des rectangles ;
- un ensemble d'arcs, notés par des flèches qui joignent les places aux transitions et les transitions aux places ;
- une distribution de jetons dans les places, symbolisés par des disques.

### b. Représentation graphique :

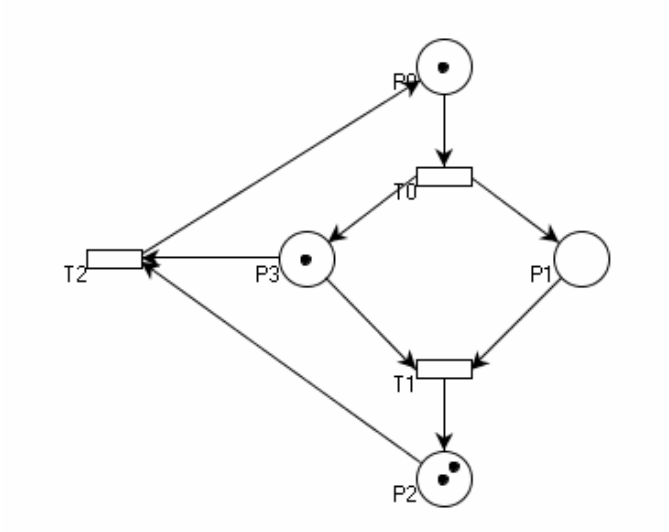
Les places Pi et les transitions Tj d'un réseau de Petri, en nombre fini et non nul, sont reliées par des arcs orientés. Un réseau de Petri est un graphe biparti alterné, c'est-à-dire qu'il y a alternance des types de noeuds : tout arc, qui doit obligatoirement avoir un noeud à chacune de ses extrémités, relie soit une place à une transition soit une transition à une place.

**Exemple de réseau de Petri comportant 4 places, 3 transitions et 9 arcs orientés.**




### c. Définition de l'état d'un réseau de Petri :

Pour définir l'état d'un système modélisé par un réseau de Petri, il est nécessaire de compléter le réseau de Petri par un marquage. Ce marquage consiste à disposer un nombre entier (positif ou nul) de marques ou jetons dans chaque place du réseau de Petri. Le nombre de jetons contenus dans une place Pi sera noté mi. Le marquage du réseau sera alors défini par le vecteur M={mi}.

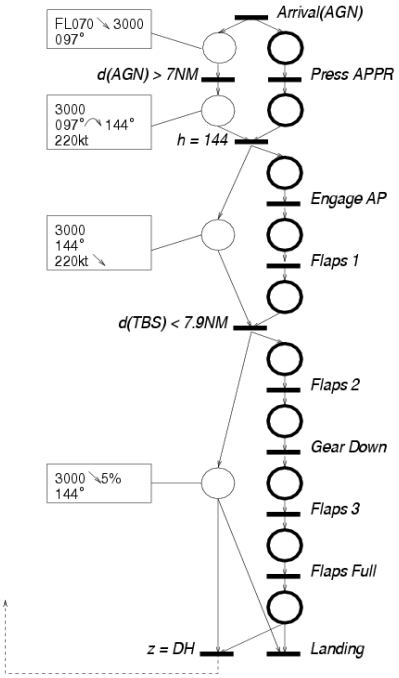



**Exemple de réseau de Petri marqué avec un vecteur de marquage M = {1,0,2,1}**

Sous la tutelle de Catherine Tessier, Charles Lesire et Olivier Bonnet-Torrès, doctorants du DCSD, utilisent dans le cadre de leurs travaux de thèses deux types de réseaux de Petri particuliers. Il s'agit des réseaux de Petri **particulaires** et des réseaux de Petri **modulaires**. Les parties suivantes décrivent brièvement le contenu des deux thèses.

### *d. Les réseaux particulaires :*

Dans le cadre de la thèse de Charles Lesire sur l'estimation numérico-symbolique pour le suivi d'activités hybrides le système avion-pilote est représenté par un réseau de Petri particulier.




L'avion suit un plan de vol découpé en segments. Son déplacement le long de ces segments correspond à une évolution continue des paramètres de vol (vitesse, cap, altitude...). Pendant ce temps, le pilote agit sur les systèmes pour mettre l'avion dans une configuration adaptée au segment en cours, et inversement, les systèmes envoient des informations au pilote.

Les **réseaux particulaires** permettent de représenter ce système avion-pilote ainsi que les différentes incertitudes sur les paramètres. La figure donne une représentation de la procédure d'approche sur l'aéroport Toulouse-Blagnac à l'aide d'un réseau particulaire.

### e. Les réseaux modulaires :

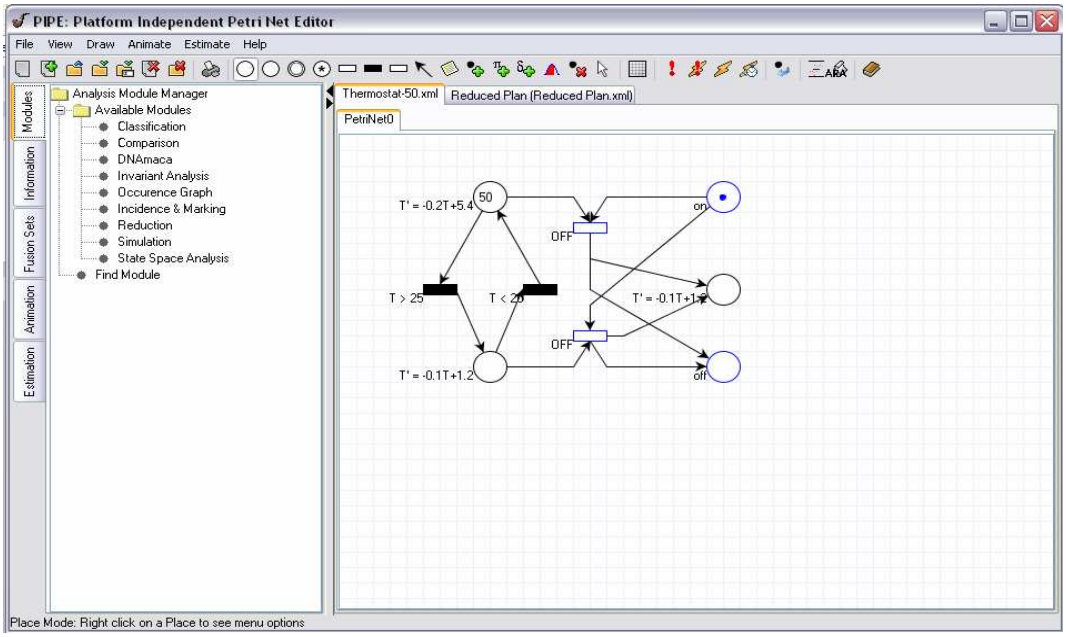
Dans le cadre de la mise en œuvre d'une équipe de robots, Olivier Bonnet-Torrès, propose une architecture décisionnelle qui permet une replanification des tâches lorsque l'autorité est dévolue à l'équipe. Ces recherches ont pour but de réparer le plus localement possible un plan devenu obsolète suite à un aléa (quelle qu'en soit sa nature), au niveau local ou global. Cette réparation locale est facilitée par la représentation du plan d'équipe par un réseau de Petri hiérarchique et modulaire.

De manière générale, ces travaux de recherche nécessitent l'utilisation d'éléments plus spécifiques que ceux présents dans un réseau de Petri classique. Ainsi on notera la présence de places symboliques, numériques ou encore celle de transitions spécifiques.

## 2) PIPE :

PIPE (Platform Independent Petri Net Editor) a été développé par le département des sciences de l'Imperial College de Londres en 2002. Il s'agit d'un logiciel Open Source dont le code est disponible sur internet. Il est codé en Java et comme son nom l'indique présente l'avantage d'être multi plate forme.

Pour prendre en charge la modélisation et le jeu des réseaux de Petri particulaires et modulaires, de nombreuses modifications ont été apportés a la version de départ. Au terme de ces modifications Pipe possède un éditeur graphique complet, et un simulateur qui permet de jouer les réseaux de Petri. Divers outils, présentés sous forme de modules, sont également accessibles, tels que la fusion de places et de transitions ou encore le calcul des matrices d'incidence.




Exhost Pipe : version modifiée de Pipe

## 3) Objectifs du stage :

Le stage consiste à modifier le code prototype écrit en Java pour en faire un produit *open-source* délivrable avec des garanties de qualité. Entre autres, les éléments suivants nécessitent un développement conséquent :

- réduction du nombre de bogues sur l'interface fichiers,
- mise en conformité avec les standards XML et PNML des fichiers entrées/sorties,
- amélioration de la javadoc,
- prise en compte des modifications de Pipe2,
- améliorations mineures des fonctionnalités.

## 4) Outils utilisés :

Durant ces quatre mois j'ai eu l'occasion de découvrir et d'utiliser un certain nombre d'outils dont la mise en oeuvre est détaillée dans ce rapport :

- Eclipse
- Omondo
- Metrics
- PMD
- Cooktop 2.5
- Emma

# III. Travail réalisé

## 1) Analyse de Pipe

### a. Les évolutions de Pipe :

Pour la suite de l'étude nous distinguerons les différentes versions de la manière suivante :

PIPE1 : il s'agit de la version originale, elle ne permet de représenter que des réseaux classiques composés de places et de transitions très élémentaires. Le joueur de Petri est réduit à sa plus simple expression. En effet l'utilisateur doit cliquer lui-même sur les transitions qu'il désire franchir, celles-ci étant surlignées lorsqu'elles sont franchissables. Son interface graphique très intuitive présente de nombreux bogues.

PIPE2 : cette deuxième version est la seule actuellement disponible sur internet. Il s'agit en réalité de Pipe1 revu et corrigé, elle présente quelques fonctionnalités supplémentaires notamment au niveau de l'éditeur graphique.

PIPE4 ou Exhost-Pipe : développé par Olivier Bonnet-Torrès, Charles Lesire et Patrice Domenech, Pipe4 permet la représentation et le jeu des réseaux de Petri particulaires et modulaires. Cette version propose différents modes d'animation ainsi que des composants graphiques supplémentaires permettant de suivre l'évolution de l'état du réseau lors d'une animation.

Les copies d'écran ci-dessous présentent les objets graphiques disponibles sur la version de base, puis ceux proposées par la version modifiée.

Version de base : 

Version modifiée : 

### b. La structure générale :

Les classes du code de PIPE sont regroupées dans les « packages » DataLayer, Gui et Modules. Cette répartition des classes est très utile car elle permet une compréhension plus facile du code qui compte près de 110 classes à la base.

Le package DataLayer contient toutes les classes liées à l'objet réseau de Petri lui-même. En effet, on trouve les classes des places, transitions, jetons, arcs…

Le package Gui contient toutes les classes liées à l'interface graphique (menu « popup » lors d'un click sur les places ou les transitions…) et à l'aspect fonctionnel du logiciel (sauvegarde, ouverture d'un fichier, exportation sous forme d'image…)

Le package Modules regroupe l'ensemble des outils tels que le calcul des invariants ou des matrices d'incidence.

### c.Approche UML :

Afin d'aboutir à une meilleure compréhension du fonctionnement interne de l'application et d'avoir une vision globale des évolutions entre les différentes versions, un diagramme des classes simplifié a été généré à l'aide de l'outil Omondo. Etant donné la taille de l'application, du nombre de classes et de packages, le reverse engineering n'a concerné que le package Gui et DataLayer.

La figure ci-dessous propose un extrait simplifié du diagramme des classes du package DataLayer. Il a été choisi en raison du nombre important de modifications dont il a fait l'objet. Ainsi on remarquera l'ajout de la classe MCPN permettant la mise en place et l'animation de plusieurs réseaux simultanément. Mais aussi le changement du mécanisme de tir, au départ entièrement défini dans la classe DataLayer, il est désormais réparti entre l'ensemble des objets du réseau (Place, Transition, Token). La classe Transition prend en charge une partie importante du traitement en déterminant la nature des places en amont et en aval, et en leur appliquant les règles de franchissement nécessaires.

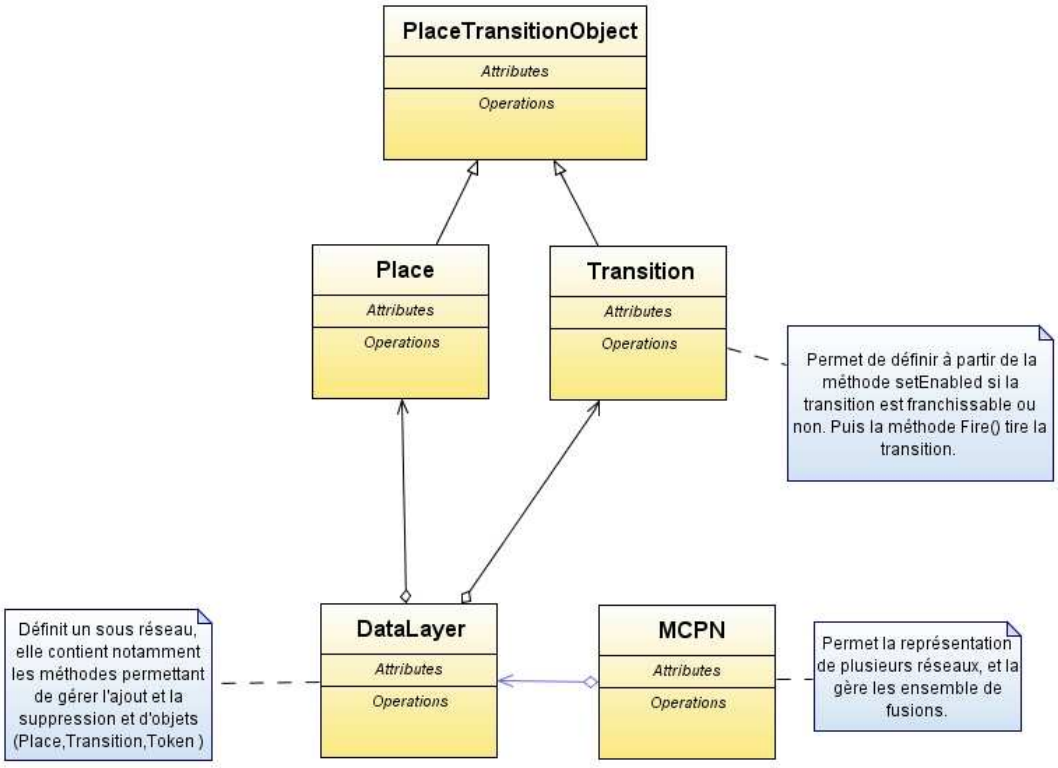



Figure 1 : Extrait du diagramme des classes de Pipe4

### d.Analyse quantitative et qualitative :

L'analyse quantitative permet d'obtenir des indicateurs sur la qualité des différentes versions de Pipe et de guider ainsi les modifications nécessaires.

➢ Informations sur la qualité du code :

Java impose une norme de documentation du code appelée javadoc. Par défaut le compilateur ne produit pas d'alerte lorsque la documentation est absente. Il est possible de modifier son paramétrage pour produire des avertissements signalant le code incriminé.
Une fois cette modification effectuée, la recompilation à l'aide d'Eclipse génère un nombre important d'avertissements.

La documentation de ce projet est donc problématique et nécessite un travail conséquent.

➢ Les bogues :

La prise en main de l'outil a été une étape importante pour la suite de l'étude, en effet celle-ci a permis de dresser la liste des bogues, afin de pouvoir les associer plus facilement à une zone du logiciel. Parmi les anomalies détectées on notera l'impossibilité de charger les modèles des versions 1 et 2, ainsi qu'un dysfonctionnement au niveau des boutons d'animations.

➢ Métriques logicielles :

Pour terminer cette analyse, nous avons calculé diverses métriques logicielles à l'aide du plug-in Eclipse Metrics.
Le résultat obtenu est le suivant :

| Métriques | PIPE 1 | PIPE 4 |
|---|---|---|
| Nombre de packages | 15 | 19 |
| Nombre de classes | 110 | 180 |
| Complexité cyclomatique moyenne/projet | 5 | 7 |

Tableau 1 : Dimension et mesure de risque

La complexité cyclomatique est une mesure permettant d'évaluer la complexité du logiciel étudié en tenant compte de l'imbrication des boucles et des conditions. Dans le tableau ci-dessus il ne s'agit que d'une valeur moyenne donnée à titre d'exemple. En réalité cette métrique a été utilisée a une échelle plus réduite et ce afin de localiser les éventuelles classes nécessitant des opérations de restructuration.

Le tableau ci-dessous présente les valeurs qui ont servi de référence pour tenir compte de cette mesure.

| Complexité cyclomatique | Niveau de risque |
|---|---|
| 1-10 | Programme simple sans véritable risque |
| 11-20 | Programme modérément complexe et risqué |
| 21-50 | Programme complexe et hautement risqué |

Tableau 2 : Mesures et risques associés

Source : Software Engineering Institute

Un grand nombre de classes de Pipe4 ont une complexité cyclomatique importante et se trouvent dans les packages Gui et DataLayer.
L'extrême est atteint dans la méthode loadPNML avec une complexité de 29.

Ces analyses permettent de planifier les tâches à réaliser et de leur attribuer un ordre de priorité :

- documentation du code,
- correction des bogues,
- simplification de certaines structures de codes.

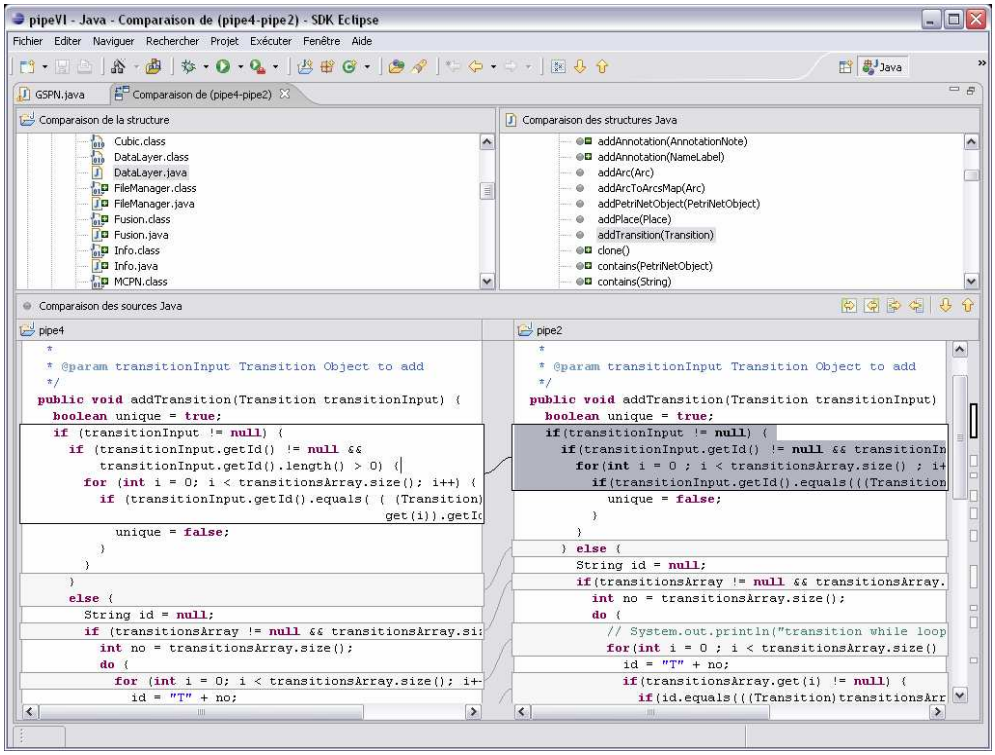
## 2) Vers PIPE5

### *a.La migration :*

Dans le but d'aboutir a une harmonisation des travaux menés parallèlement à la fois sur Pipe2 et Pipe4, il est apparu préférable d'utiliser la version 2 comme base de développement et d'y adapter les modifications de Pipe4.

Le modèle UML a permis avec l'aide de mes responsables de stages, de mettre en évidence les classes créées, modifiées et éventuellement celles supprimées (cf :annexes). Pour affiner la comparaison et réduire les erreurs d'appréciation, la migration s'est faite méthode par méthode. Ce travail a été effectué a l'aide du comparateur d'Eclipse qui s'est avéré être un outil puissant et simple à utiliser.

La capture d'écran illustre l'utilisation d'Eclipse : il s'agit de la comparaison de deux classes, les blocs de codes comportant des différences sont reliés par un arc




Comparateur d'Eclipse :

*b.Documentation et revues de codes :*

La javadoc absente ou mal structurée, a été entièrement revue et corrigée pour les packages Gui, DataLayer et Reduction. L'objectif de ce second travail est double, la documentation permettra :

- de comprendre la structure et le fonctionnement du programme ;
- d'adapter le programme à de nouvelles exigences (changement de spécifications, enrichissement des fonctionnalités attendues, amélioration de l'efficacité ou de la précision des calculs, …).

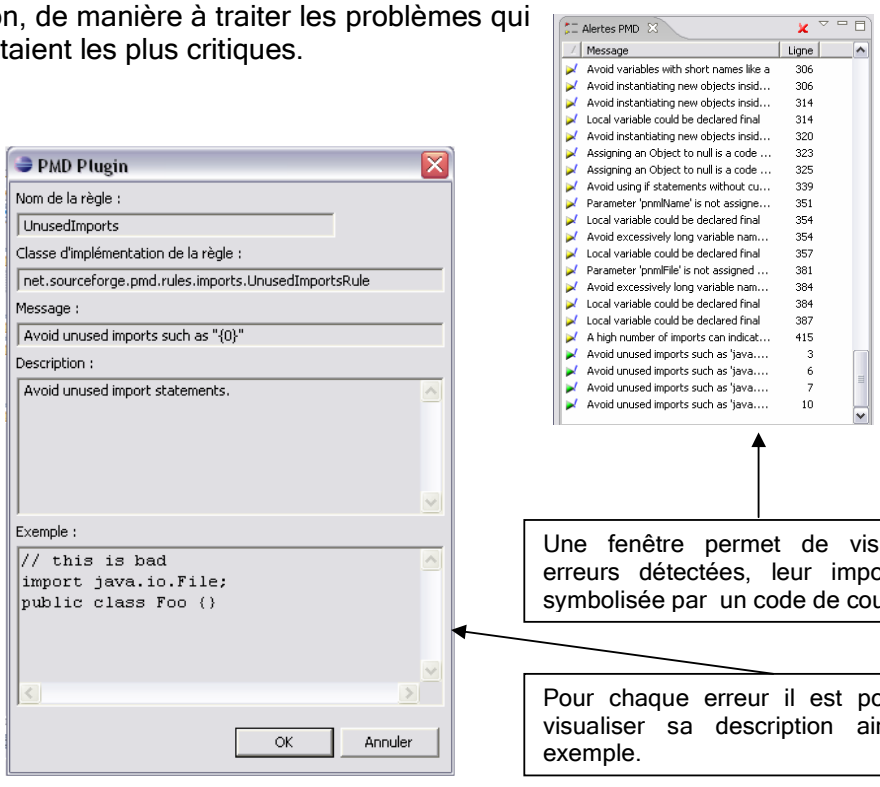
La revue de code, part importante du travail accompli, a été réalisée à l'aide de PMD. Cet outil gratuit a permis l'analyse et la correction du code.

PMD est un plugin d'Eclipse, il repose sur un certain nombre de règles auxquelles sont confrontées toutes les lignes de code source qui composent le logiciel. Ces règles sont fournies en standard avec l'outil et peuvent être paramétrées.

Parmi l'ensemble des problèmes détectés l'accent a été mis sur les points suivants :

- code mort : variables ou paramètres inutilisés, importations de packages inutiles ;
- gestion des exceptions : clauses catch sans contenu ;
- documentation javadoc : vérification de la présence de documentation et de la syntaxe.

Il est à noter que l'utilisation de cet outil génère un nombre important d'erreurs lorsqu'il est exécuté pour la première fois. D'autant que certaines règles peu importantes peuvent générer beaucoup de bruit. J'ai donc consacré du temps à la configuration, de manière à traiter les problèmes qui selon moi étaient les plus critiques.



### c.Ajout de fonctionnalités :

Outre l'amélioration du code quelques structures ont été modifiées, on notera ainsi la fusion de certaines classes (AnimationHistory 1et 2) ou encore l'intégration de quelques fonctionnalités mineures au niveau de l'interface. Ces dernières concernent notamment l'éditeur graphique.

## 3) Mise en conformité avec les standards XML et PNML des fichiers entrées/sorties :

La première partie de ce travail a consisté à définir la structure des documents XML générés lors de la sauvegarde, afin de permettre d'une part, de cerner le problème lié au chargement des modèles crées a partir de la version originale, d'autre part de faciliter la mise en place d'un standard tenant compte à la fois des travaux réalisés par les doctorants, et de la norme déjà existante (PNML).

### a.Sauvegarde et chargement :

Lors de la sauvegarde d'un modèle sous Pipe, une feuille de style présente dans le répertoire XSL est appliquée afin de le rendre conforme à la norme PNML. Le même procédé est utilisé pour le chargement, à la différence que la feuille de style permet dans ce second cas de définir une structure semblable au modèle objet. L'impossibilité de visualiser les documents réalisés avec les versions 1 et 2 à partir de Pipe4, était liée à l'absence de certains objets. La solution a donc consisté à instancier par défaut les objets manquants. Ces modifications se sont traduites par l'ajout de méthodes dans la classe FileManager du package DataLayer.

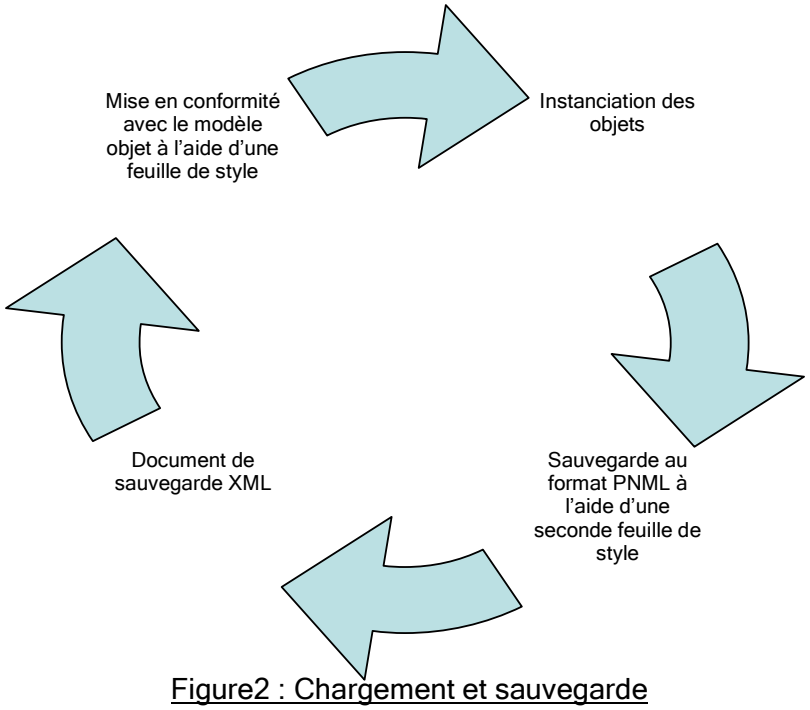



Figure2 : Chargement et sauvegarde

*b.Définition d'un standard XML :*

PNML pour Petri Net Markup Language est un standard XML mis en place dans le but de faciliter les échanges des modèles Petri. Il est disponible à l'adresse suivante : http://www.informatik.uni-hamburg.de/TGI/PetriNets/, et propose une notation et une structure type pour les documents XML.

Comme on a pu le voir précédemment, dans sa version modifiée Pipe est capable de manipuler des éléments spécifiques aux réseaux de Petri particuliers. Par conséquent une nouvelle structure basée sur le PNML a été créée, et un modèle de document PNTD (Petri Net Document Type Definition) mis a disposition (annexe 1). Cette DTD définit la structure d'un document, les éléments et attributs qui y sont autorisés, et le type de contenu ou d'attribut permis. Il fait la différence entre un document bien formé et un document valide : le premier répond aux exigences de la spécification, tandis que le second se conforme strictement aux règles établies par la PNTD.

La PNTD a deux objectifs le premier est de proposer une grammaire plus complète pour la notation XML. Le second est de promouvoir l'utilisation de Pipe comme outil de modélisation. En effet tous les documents valider par le modèle proposé sont compatibles avec Pipe4.

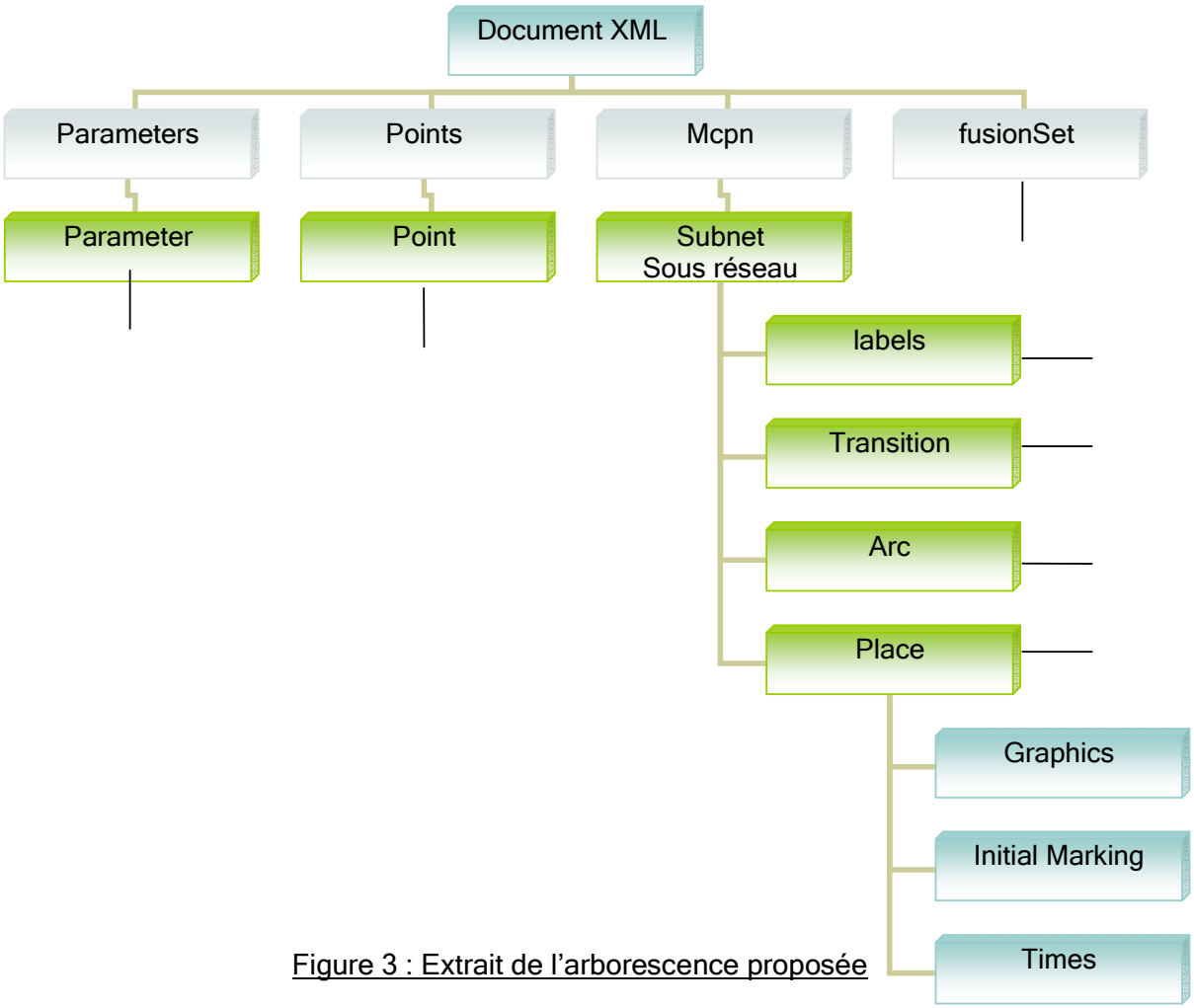



Figure 3 : Extrait de l'arborescence proposée

## 4) Restructuration :

Les analyses présentées précédemment ont permis d'orienter les opérations de restructuration. Le paragraphe suivant décrit les étapes ayant permis de simplifier le code de la classe FileManager. Celle-ci gère la sauvegarde et le chargement des fichiers.

*Refonte de loadPNML() :*

- Description :

Cette méthode effectue un traitement que l'ont peut décomposer de la manière suivante :

- chargement du fichier,
- mise en forme à l'aide d'un document XSL,
- instanciation des objets.

- Points faibles :
  - Nombres de ligne de code importante, 302 au total.
  - Profondeur d'imbrication des blocs de code forte
  - Complexité cyclomatique élevé (29)

- Solution :

L'extraction d'une méthode consiste à sélectionner un bloc de code consistant, dont l'exécution peut se faire de manière indépendante, et à le transformer en une méthode. Cette opération a deux objectifs :

- l'extraction rend la méthode source moins complexe et plus lisible, les méthodes extraites sont privées car le détail d'implémentation n'est pas visible de l'extérieur ;
- le bloc de code constitue un élément réutilisable au sein de la classe.

- Application :

Le processus détaillé dans la figure 3 est assez simple. Les paramètres de la méthode source nécessaires à l'exécution de la méthode extraite deviennent aussi des paramètres de cette dernière. Les variables locales utilisées exclusivement par la nouvelle méthode sont extraites en même temps que le bloc. Enfin la dernière opération consiste à déplacer le bloc de code de la méthode source pour le placer dans la méthode extraite et remplacer le vide ainsi formé par un appel à la méthode créée.

Eclipse permet d'automatiser cette opération. Les variables locales externes au bloc à extraire sont transformées systématiquement en paramètres.

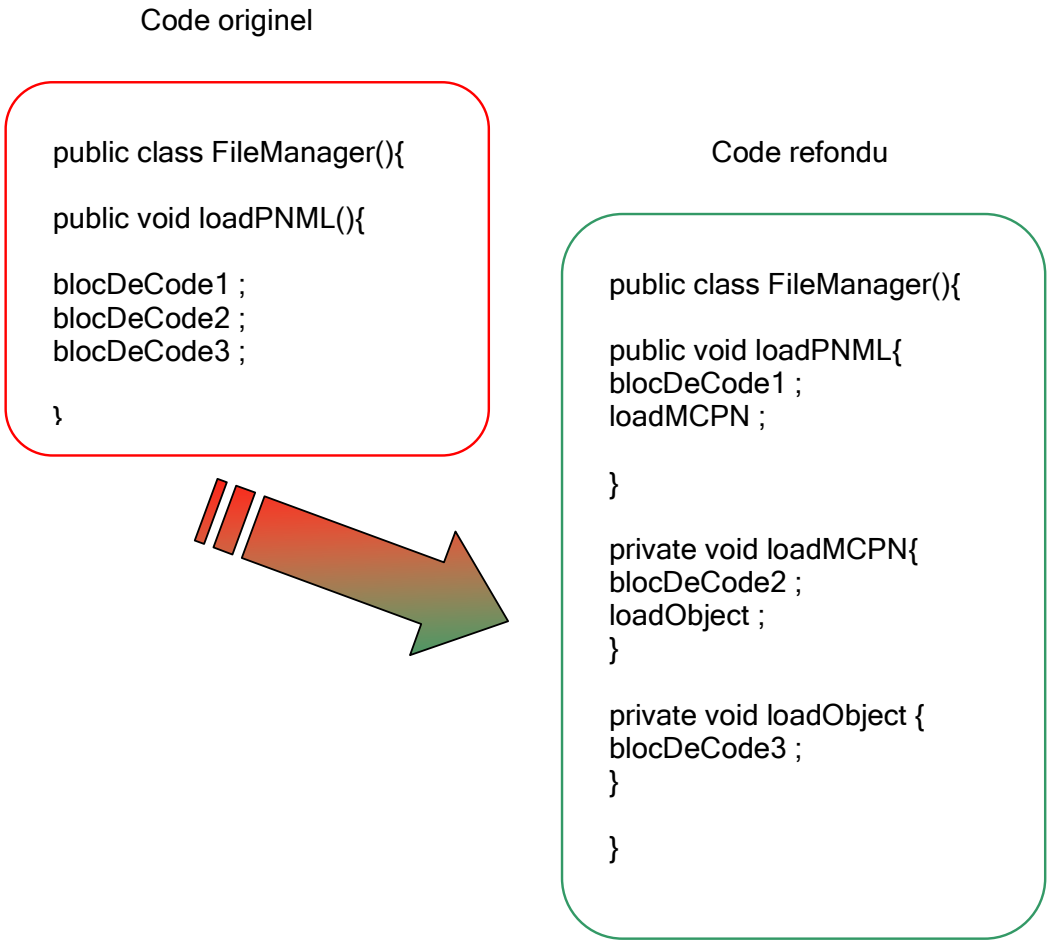



Figure 4 : Extraction d'une méthode

➢ Bilan :

Au terme de cette modification on note la création de quatre méthodes, une diminution des lignes de code par méthode et surtout la réduction de la complexité du code qui passe de 29 à 13.

La technique que nous venons de voir est une parmi d'autres utilisées afin de faciliter les modifications dont Pipe est susceptibles de faire l'objet. Au vu de la durée du stage et du temps restant, toutes les opérations de restructurations n'ont pu être mises en place.

## 5) Analyse de couverture des tests :

La dernière partie de mon stage a été consacrée à la réalisation des jeux de tests, pour valider le travail effectué et juger de l'aspect fonctionnel de Pipe5. Cette dernière étape s'est déroulée en deux parties. La première a permis de définir des jeux de test couvrant un maximum de fonctionnalités, et la seconde à réaliser une analyse de couverture. Celle-ci est nécessaire à toute campagne de tests car elle permet de visualiser le code exécuté ou non par ces derniers. Dans le cadre de cette analyse j'ai utilisé l'outil EMMA. Il est gratuit, et présente l'avantage de pouvoir générer un rapport complet et détaillé.

Fonctionnement d'EMMA :

Après avoir installé Emma, on lance le .jar à partir de ce dernier. Lors de la simulation sur Pipe Emma évalue l'exécution pour chaque branche de code. Une fois la simulation terminée, un document d'analyse est généré avec un tableau de vue d'ensemble de la couverture et un listing détaillé du code, portant les comptes pour l'exécution de chaque méthode. Ce rapport délivre des informations immédiates sur l'efficacité du test. En effet la connaissance de l'architecture de Pipe et le détail du rapport permettent de définir les fonctionnalités non prises en compte, et ainsi de compléter les jeux de test.

La capture d'écran illustre le type de statistiques produites par Emma. Le pourcentage indique le taux de code exécuté à différents niveaux d'agrégation (packages, classes, méthodes…).

[all classes][pipe.dataLayer]

COVERAGE SUMMARY FOR SOURCE FILE [FileManager.java]

| name | class, % | method, % | block, % | line, % |
|---|---|---|---|---|
| FileManager.java | 100% (1/1) | 50% (7/14) | 61% (979/1604) | 62% (216,7/349) |

COVERAGE BREAKDOWN BY CLASS AND METHOD

| name | class, % | method, % | block, % |
|---|---|---|---|
| class FileManager | 100% (1/1) | 50% (7/14) | 61% (979/1604) |
| FileManager (): void | | 0% (0/1) | 0% (0/11) |
| FileManager (File): void | | 0% (0/1) | 0% (0/5) |
| getDOM (): Document | | 0% (0/1) | 0% (0/5) |
| getDOM (String): Document | | 0% (0/1) | 0% (0/34) |
| savePNML (File, MCPN): void | | 0% (0/1) | 0% (0/381) |
| savePNML (String, MCPN): void | | 0% (0/1) | 0% (0/7) |
| setMCPN (MCPN): void | | 0% (0/1) | 0% (0/4) |
| FileManager (String): void | | 100% (1/1) | 28% (12/43) |
| getDOM (File): Document | | 100% (1/1) | 33% (17/51) |
| newPnml (Node, int, NodeList, NodeList, NodeList, Element, Node, Node, MCPN, ... | | 100% (1/1) | 85% (282/332) |
| oldPnml (String, MCPN): void | | 100% (1/1) | 90% (518/575) |
| loadPNML (String): MCPN | | 100% (1/1) | 95% (124/130) |
| clearPNML (): void | | 100% (1/1) | 100% (8/8) |
| pnmlType (NodeList): boolean | | 100% (1/1) | 100% (18/18) |

Extrait d'un rapport généré pour la classe FileManager :

# IV. Bilan

Afin de mieux appréhender le bilan de mon stage au sein du DCSD, nous aborderons tout d'abord les principaux résultats et implications obtenus. Puis, dans une partie plus personnelle, je vous présenterai une analyse critique de ce stage : ses points positifs et ses points négatifs.

## 1) Principaux résultats et implications :

Tout d'abord, il convient de préciser que les principaux objectifs ont été atteints.
En effet Pipe5 présente des gages de qualité aussi bien d'un point de vue fonctionnel, que structurel. Les jeux de tests et la javadoc permettent ainsi de juger du travail accompli.

## 2) Analyse critique du stage :

Afin de vous présenter une analyse critique, je vais à présent me pencher sur les aspects positifs et négatifs de mon stage.

- Les aspects positifs :

J'ai beaucoup apprécié le degré d'autonomie dans lequel je travaillais. En effet, toute l'équipe me faisait confiance et me laissait mener les opérations du début jusqu'à la fin. Bien sûr, en cas de problème, chaque membre de l'équipe était disponible et je pouvais toujours compter sur quelqu'un pour m'épauler. Ces conditions d'autonomie et les responsabilités qui m'ont été confiées m'ont semblé essentielles. Elles m'ont donc permis de travailler dans des conditions semblables à celles qui m'attendent à la sortie de ma licence.

- Les aspects négatifs :

Il faut tout d'abord préciser que la durée de mon stage m'a paru insuffisante au vu du travail à réaliser. Le souci durant le stage était d'une part d'identifier les tâches à réaliser mais aussi de définir leur ordre de priorité.

Enfin, la configuration des postes de travail et l'impossibilité de procéder à l'installation de nouveaux logiciels m'ont privé de l'utilisation d'outils, tel que CVS.

# Conclusion

Ce stage a la particularité de m'avoir immergé dans le monde de la recherche, un domaine dont j'ignorai le fonctionnement. En effet il s'agit de s'instruire en permanence tout en évoluant avec le travail effectué. J'ai d'ailleurs mieux compris ses objectifs et ses liens avec les différents domaines de l'industrie.

Je remercie tous les membres de l'équipe qui ont su se montrer disponibles et attentifs malgré leurs emplois du temps chargés. C'est ainsi que j'ai pu tout au long de ces quatre mois de stage acquérir une expérience du travail en équipe. Mais également étudier les différents aspects du refactoring et les outils facilitant sa mise en oeuvre. L'analyse d'un code de taille conséquente m'a permis d'appliquer mes connaissances dans plusieurs domaines dont celui de la programmation objet mais également de la représentation UML. J'ai saisi la très grande utilité de cette notation lors de la réutilisation d'un code existant.

Cette séquence au sein de l'ONERA a aussi été l'occasion d'aborder de nouvelles connaissances dans le domaine des réseaux de Petri qui m'étaient encore totalement inconnue. Cela m'a permis de découvrir l'utilisation de ces réseaux ainsi que leurs éventuelles applications au domaine de l'aéronautique et à celui de la robotique.

En conclusion ce stage m'a réellement apporté en maturité professionnelle et en assurance. Il est pour moi un réel tremplin afin de me lancer dans la vie active.

# *Références*


- Programmer en java, Claude Delannoy, Edition Eyrolles
- Du Grafcet aux réseaux de Petri, René David Hassane Alla, Edition Ermes
- Lesire C., Tessier C., « Réseaux de Petri particulaires pour l'estimation symbolico-numérique », FAC'05.
- Bonnet-Torrès O., Tessier C., « Architecture décisionnelle pour la replanification dans une équipe d'agents », JFSMA'04.


Sites internet :


- http://sitedudeveloppeur.net
- http://www.petri-net.net/htdocs/
- http://java.sun.com/j2se/1.5.0/docs/api/
- http://www.informatik.uni-hamburg.de/TGI/PetriNets/
- http://petri-net.sourceforge.net

# Annexes

## Extrait de la PNTD :

<!ELEMENT pnml ( parameters*, points, net+ ) >

<!ELEMENT parameters ( parameter*) >
<!ELEMENT parameter EMPTY >

<!ELEMENT points (point*)>
<!ELEMENT point EMPTY >

<!ELEMENT net ( subnet+, fusionset* ) >

<!ELEMENT subnet (labels*, place+, transition+, arc+ ) >

<!ELEMENT place ( times, graphics, name,marking?,equation? ,initialMarking) >

<!ELEMENT initialMarking ( value, graphics ) >

<!ELEMENT transition ( times, graphics, name, orientation, rate, timed , condition?, effect?) >

<!ELEMENT condition EMPTY >

<!ELEMENT orientation ( value ) >

<!ELEMENT rate ( value ) >

<!ELEMENT timed ( value ) >

<!ELEMENT times EMPTY >

<!ELEMENT name ( value, graphics ) >

<!ELEMENT arc ( graphics, inscription, arcpath+ ) >

<!ELEMENT inscription ( value, graphics ) >

<!ELEMENT arcpath EMPTY >

<!ELEMENT value ( #PCDATA ) >

<!ELEMENT graphics ( offset | position )* >

<!ELEMENT offset EMPTY >

<!ELEMENT position EMPTY >

<!ELEMENT fusionset ( object+ ) >

<!ELEMENT object EMPTY >

<!ELEMENT labels ( text ) >

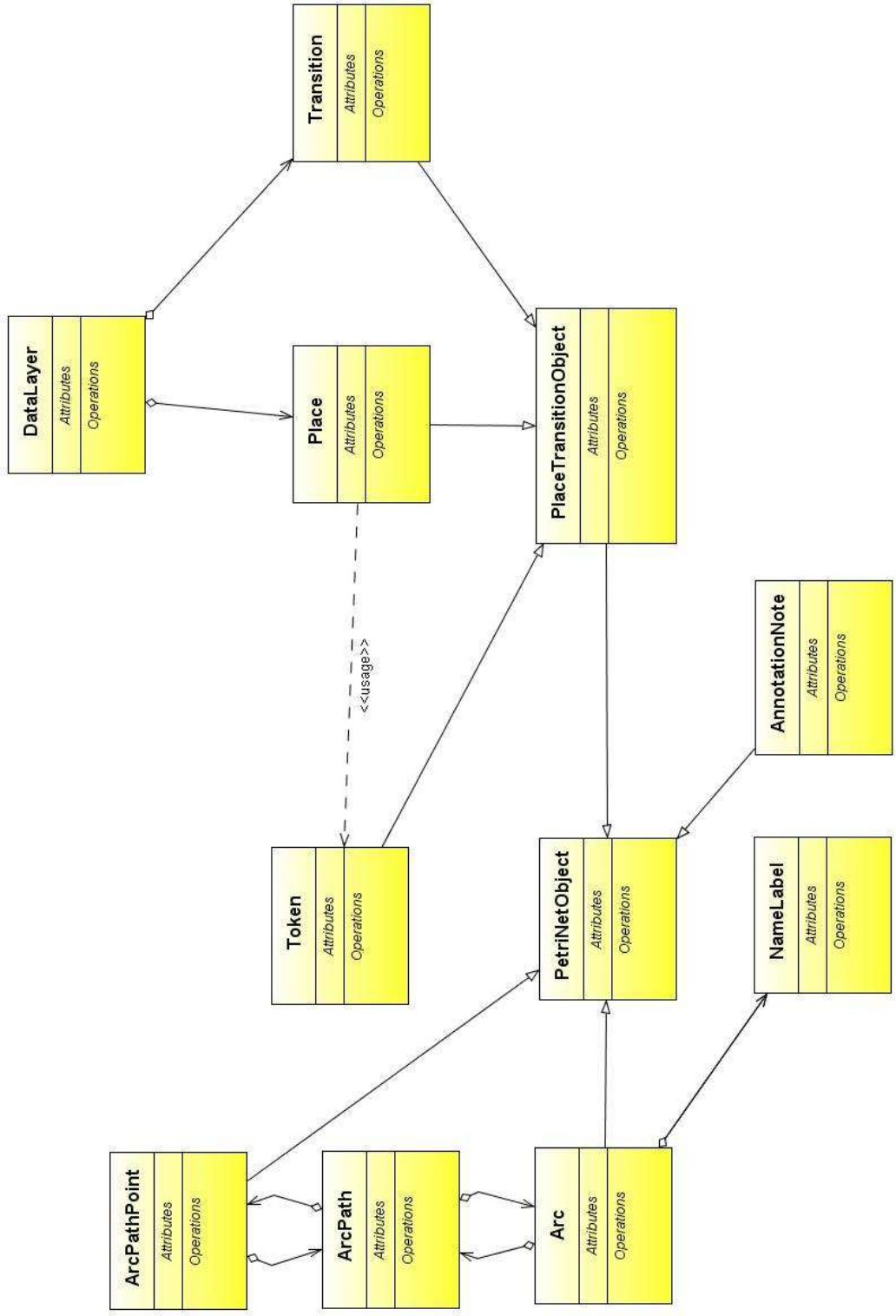
<!ELEMENT text ( #PCDATA ) >

<!ELEMENT effect EMPTY >
<!ELEMENT marking (token*)>
<!ELEMENT equation ( parameter* ) >

Notations:

| | | |
|---|---|---|
| ANY | L'élément peut contenir tout type de données | |
| EMPTY | L'élément ne contient pas de données spécifiques | |
| #PCDATA | L'élément doit contenir une chaîne de caractères | |
| + | L'élément doit être présent au minimum une fois | A+ |
| * | L'élément peut être présent plusieurs fois (ou aucune) | A* |
| ? | L'élément peut être optionnellement présent | A? |
| \| | L'élément A ou l'élément B peuvent être présents | A\|B |

## Pipe1 : Diagramme des classes du package DataLayer



## Pipe4 : Diagramme des classes du package DataLayer