IMT Atlantique
Bretagne-Pays de la Loire
École Mines-Télécom

FR-PN – Réseau de Petri

INFO – MAPD

Objectifs du projet

- Dérouler un petit projet de développement logiciel en suivant un cycle en V
- Concevoir une solution orientée objet à un problème simple décrit dans un cahier de charges informel. La conception inclut un diagramme de classes UML et (au moins) un diagramme de séquence UML
- Justifier les choix réalisées lors de la conception d'une solution orientée objet à un problème simple
- Appliquer des bonnes pratiques de développement pour réaliser une implémentation Java d'un modèle UML donné (diagramme de classes et diagramme de séquence) et dont le domaine est connu
- Réaliser une critique constructive d'un code Java donné à partir d'un guide de bonnes pratiques de développement
- Analyser un code existant pour en déduire son fonctionnement dans l'objectif de l'intégrer à son propre code
- Utiliser le patron de conception *Adapter* pour intégrer son code dans un code existant

Travail attendu

Le Fil Rouge est un travail dans la durée que vous réaliserez en binôme. Les séances se suivent et sont calibrées (en principe) pour permettre de réaliser le travail demandé. Il pourrait être nécessaire de passer un peu de temps hors séance pour finir un travail, comme il peut être utile de passer un peu de temps hors séance pour approfondir une question, travailler une solution différente (une variante), etc.

Vous produirez des documents (brouillons, version définitive, variante), du code source, des réponses à des questions, etc. Il est important que vous ayez toujours à disposition l'ensemble de vos productions : sous forme numérique, sous forme papier, ou autre.

Il est essentiel d'avoir un référentiel de vos documents ; c'est un document qui dresse la liste de toutes vos productions : ce que c'est, son statut (brouillon, version, final), sa date de mise à jour, où on peut la trouver.

Le fil rouge est évalué en 3 phases :

1. Conception : les TD1-3,
2. La réalisation : les TP3-4, TD4
3. L'intégration : les TP11 ;15-16

Tout sera rendu sur l'espace *Moodle* de l'UE. Vous y trouverez les espaces de dépôt correspondants et les dates limites de dépôt.

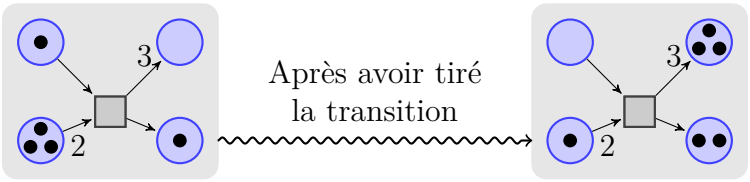
Réseau de Petri

Un réseau de Petri est un modèle mathématique servant à représenter divers systèmes (informatiques, industriels...). Il permet de modéliser et de vérifier le comportement dynamique des systèmes à événements discrets comme les systèmes manufacturiers, les systèmes de télécommunications ou les réseaux de transport. Vous pouvez voir un exemple très simple d'utilisation d'un réseau de petri à partir de la minute 1'30" de la vidéo `https://www.youtube.com/watch?v=EmYVZuczJ6k&frags=pl%2Cwn`.

Un réseau de Petri est composé de **places** (représentées par des cercles) qui peuvent contenir des **jetons** et de **transitions** (représentées par des carrés). Des **arcs** (porteurs d'une valeur) relient les places aux transitions (arcs sortants des places) et les transitions aux places (arcs entrants dans des places). Sans précision, la valeur d'un arc est par défaut 1.

Une transition est *tirable* lorsque les places qui y sont reliées par un arc entrant contiennent au moins autant de jetons qu'indiqué par la valeur de l'arc. Quand une transition est tirée (un *pas*), on enlève des jetons des places en amont de la transition selon les indications des arcs sortants (des places) et on ajoute des jetons dans les places en aval selon les indications des arcs y entrants. Le nombre total de jetons peut donc changer d'un pas à l'autre.

Le schéma qui suit montre un réseau de Petri avant que la transition soit tirée et après qu'elle a été tirée.



Dans ce schéma, on enlève des jetons aux places d'origine (ici à gauche) de la transition, et on ajoute des jetons dans les places d'arrivée (ici à droite) de la transition. Le nombre de jetons ajoutés ou enlevés est donné par le valeur sur l'arc (1 par défaut.) On a fait un pas (`step`).

À chaque pas, il peut y avoir plusieurs transitions *tirables* ; une seule est effectivement tirée après un choix non-déterministe.

Le but de ce projet est de concevoir et développer d'un outil de simulation de réseau de Petri.

L'enchaînement des exercices dans ce documenet est aligné sur les séances programmées. Il est bien sûr possible d'aller plus vite. Nous vous recommandons toutefois de ne pas aller trop vite, pour profiter des périodes de synthèses/synchronisation qui permettront de prendre du recul sur le déroulement de ce Fil Rouge.

Exercice 1 (*TD 1 : Conception initiale - diagramme de classe et de séquence*)

Objectifs

— Rappel de la notation UML (diagramme de classes)
— Identification des classes, des relations
— Raffinement avec attributs et méthodes
— Rappel : héritage, redéfinition
— Rappel de la notation UML (diagramme de séquence)
— Produire un diagramme de séquence
— Pour interagir, il faut connaître (attribut, paramètre, résultat de calcul)

Déroulement de la séance

1. Lire le descriptif des réseaux de Petri
2. Produire des diagrammes de classes en justifiant les choix
3. Produire des diagrammes de séquence en justifiant les choix

Travail attendu

— Un modèle UML (diagrammes de classes et de séquence),
— Des commentaires justifiant les choix que vous avez faits, en particulier si vous avez envisagé des variantes.

Le but de cet exercice est de réaliser un début de conception du simulateur de réseau de Petri. Voici une liste d'exigences :

1. Un réseau de Petri est composé de places, de transitions et d'arcs.
2. Des arcs relient les places aux transitions et les transitions aux places.
3. Il ne peut y avoir qu'un arc dans le même sens entre une place et une transition.
4. Un arc possède un poids qui indique
   — pour un arc sortant d'une place, le nombre de jetons nécessaires pour le rendre tirable. Ce nombre de jetons sera enlevé à la place source si la transition est tirée,
   — pour un arc entrant dans une place, le nombre de jetons qui seront ajoutés à la place destination si la transition est tirée.
5. Un utilisateur peut choisir une transition pour la tirer.
6. Un arc a une valeur qui est un entier strictement positif.
7. Une place est vide ou contient des jetons.
8. Un utilisateur peut ajouter, supprimer des places, des transitions ou des arcs.
9. Des jetons peuvent être ajoutés ou enlevés à une place par un utilisateur.
10. Un utilisateur peut changer la valeur d'un arc.

Pour commencer, il peut être utile de vous asssurer que vous comprenez le fonctionnement de réseaux très simples :
— une transition avec un arc vers une place vide. Est-ce que la transition est tirable ? Si oui, que se passe-t-il si elle est tirée ? Est-elle à nouveau tirable ?

— une place vide avec un arc vers une transition ? Est-ce que la transition est tirable ? Et si on ajoute un jeton dans la place ? Est-ce que la transition devient tirable ? Si oui, que se passe-t-il si elle est tirée ? Est-elle à nouveau tirable ?
— deux places vide avec des arcs vers une transition. Mêmes questions que ci-dessus. Et si on rajoute un jeton dans une des places ? Et si on rajoute un jeton dans chacune des places ?

Par la suite, ces petits exemples simples peuvent fournir des tests de base pour l'implémentation et l'intégration.

▷ **Question 1.1 :**
Sans entrer dans les détails (ni attribut, ni méthode, mais avec rôles et cardinalités), proposez un diagramme de classes pour décrire un réseau de Petri.

▷ **Question 1.2 :**
Ajoutez méthodes et attributs dans le(s) diagramme(s) proposé(s). Ajoutez quelques commentaires explicatifs.

▷ **Question 1.3 :**
Y a-t-il des méthodes redéfinies ? Surchargée ?

▷ **Question 1.4 :**
Proposez un diagramme de séquence qui explique comment un pas est effectué (`fire`/`step()` d'une `Transition`) dans le cas simple avec un seul arc entrant dans la transition, et un seul sortant.

▷ **Question 1.5 :**
Dessinez en un autre généralisant à plusieurs arcs entrants et sortants.

Évaluation - rendu "formatif" sur moodle

Vous livrerez :
— Deux diagrammes de classes (variantes)
— Deux diagrammes de séquence (`fire` simple et multiple)
— Un diagramme de classes détaillé
— Des commentaires justifiant vos principaux choix (en particulier celui de la variante retenue)

Dans l'espace Moodle de l'UE vous trouverez l'espace de dépôt pour le livrable associé à cette activité avec le contenu attendu et la date limite de dépôt.

Le livrable sera évalué de manière formative par l'enseignant : il vous fera un retour sur son contenu dans les meilleurs délais.

Pour avoir un retour *formatif*, déposez sur moodle vos documents. L'enseignant qui vous suit vous fera un retour dans les meilleurs délais. La grille est disponible sur moodle.

Compléments

— Un éditeur en ligne : `https://apo.adrian-jagusch.de/#/Sample%20Net`
— Petri nets en 2'34" : `https://www.youtube.com/watch?v=EmYVZuczJ6k&frags=pl%2Cwn`

Exercice 2 (*TD2 : fin de la conception (extension)*)

Objectifs

- Faire évoluer un modèle UML pour tenir compte de nouvelles contraintes et/ou besoins fonctionnels
- Utiliser l'héritage et la redéfinition à bon escient dans un modèle orienté objet
- Faire des choix de conception sur la navigabilité des associations et les justifier
- Identifier les opérations de gestion pour une solution et utiliser la notion d'interface Java pour les inclure dans son modèle

Déroulement de la séance

1. Reprendre les diagrammes de classes précédents
2. Lire la description de l'extension des réseaux de Petri
3. Produire les diagrammes de classes
4. Proposer une interface (au sens Java)

Travail attendu

- Compléter le modèle UML (diagramme de classes et de séquence)
- Compléter les commentaires justifiant les choix que vous avez faits, en particulier si vous avez envisagé des variantes.

Le but de cet exercice est de revenir sur la conception car l'expressivité des réseaux de Petri peut être améliorée en définissant 2 autres types d'arcs sortants des places :

- Les arcs « zéro » qui ne sont actifs que quand la place source est vide.
- Les arcs « videurs » qui sont actifs dès qu'il y a un jeton dans la place source et qui enlèvent tous les jetons présents lorsqu'ils sont activés.

▷ **Question 2.1 :**
Dessinez un nouveau diagramme de classes (détaillé).

▷ **Question 2.2 :**
Quelles méthodes doit-on redéfinir ?

Avant de passer à la réalisation, on va faire des choix de conception sur les associations.

▷ **Question 2.3 :**
Indiquez les choix de navigabilité. Argumentez.

▷ **Question 2.4 :**
Réifiez la notion de PetriNet si cela n'a pas déjà été fait. Ajoutez une interface de service pour construire un réseau en ajoutant des places, des transitions et des arcs.

Évaluation - préparation

Aucun livrable ne vous est demandé à la fin de cette séance. Afin de vous préparer, sachez que ...

Vous livrerez après la prochaine séance :
- Le diagramme de classes détaillé retenue
- Deux diagrammes de séquence
- Des commentaires justifiant vos principaux choix (document unique qui devrait contenir des extraits de variantes de diagrammes de classes)

L'évaluation a lieu à la fin du prochain TD.

Exercice 3 (*TD3 : Regards croisés et critiques sur la conception*)

Objectifs

- Comparer et critiquer des variantes de modèles de classes
- Identifier des conséquences des choix de conception sur la programmation
- Justifier des choix de conception

Déroulement de la séance

1. Introduction de l'organisation de la séance
2. Par binôme, travail d'analyse du/des modèle/s proposé/s par l'enseignant
3. Restitution de l'analyse réalisée par les binômes pour différents modèles

Travail attendu

- Une critique d'un modèle de classe (grille et commentaires)
- La version finale du modèle de Petri net

▷ **Question 3.1 :**
Commentez, critiquez le modèle de classes choisi.

Évaluation - à déposer sur moodle après TD3

Vous livrerez :
- Le diagramme de classes détaillé retenue
- Deux diagrammes de séquence
- Des commentaires justifiant vos principaux choix (document unique qui devrait contenir des extraits de variantes de diagrammes de classes - les vôtres ou ceux comparés)

Pour l'évaluation *sommative*, déposez sur moodle vos documents. Le livrable sera évalué par l'enseignant et contribue à la validation de compétences de l'UE. (CG 1, 4 & 10). La grille est disponible sur moodle.

Exercice 4 (*TP 3&4 : Implémentation*)

Objectifs

- Utiliser un IDE (type Eclipse) pour programmer et tester un programme Java
- Écrire un programme Java conforme à un modèle UML donné constitué d'un diagramme de classes détaillé et (au moins) d'un diagramme de séquence UML
- Utiliser les bonnes pratiques de programmation et de programmation objet pour écrire un programme Java

Déroulement de la séance

1. Installer Eclipse (si besoin)
2. Créer un projet Java (en relation avec un dépot git dans `https://gitlab-df.imt-atlantique.fr`)
3. Traduire la conception en Java

Travail attendu

- Un projet Java qui implémente des réseaux de Petri
- Une auto-évaluation de l'usage de bonnes pratiques

Pour le développement vous utiliserez l'environnement Eclipse.

Utiliser Eclipse (4.9) 2018-09 pour la compatibilité avec les outils utilisés. La version *Eclipse IDE for Java Developers* est là : `https://www.eclipse.org/downloads/packages/release/2018-09/r`

⚠On utilisera l'environnement Java 1.8 (`https://www.oracle.com/fr/java/technologies/javase/javase-jdk8-downloads.html`). Il faut donc que vous l'installez sur votre machine même si vous avez déjà une autre version de Java.

(À Plouzané, l'environnement de travail vous permet d'utiliser la commande `SETUP MAPD` pour permettre l'utilisation de java 1.8 et d'Eclipse adapté à MAPD. Pour lancer Eclipse, utilisez la commande `eclipse`.)

Une fois l'installation faite, il est possible que toutes les 5 minutes vous ayez un popup avec...

An internal error occurred during : "Polling news feeds". javax/xml/bind/JAXBContext.

Pour la cacher, désactivez « automatic news polling » dans les Préférences d'Eclipse (General>News).

Mise en place de Git et Eclipse

Afin d'être efficace dans le lancement de cette phase de réalisation, nous vous proposons une procédure de mise en place de votre environnement de travail à suivre en binôme (nous supposons que Eclipse et Git sont installés). :

1. Création-initialisation du dépôt Git (un seul des membres du binôme)
   (a) Connectez-vous au service proposé par l'école gitlab

(b) Créez un projet dédié au fil rouge, avec un dépôt Git associé
(c) Partagez l'accès au projet/dépôt avec votre binôme
(d) Récupérez l'URL du dépôt
(e) L'autre membre du binôme vérifie qu'elle a bien accès au dépôt sur l'interface web

2. Test de fonctionnement et configuration initiale (pour chaque membre du binôme)
   (a) Ouvrez un terminal et déplacez-vous dans un répertoire pertinent pour vous (créez une arborescence de fichiers si besoin)
   (b) Clonez le dépôt Git nouvellement créé : `git clone url_de_votre_dépôt`
   (c) Si vous ne l'avez pas déjà fait,
      — configurez le courriel associé à votre compte dans Git :
      `git config --global user.email "votre.email@imt-atlantique.net"`
      — configurez le nom associé aux commits, de sorte que vous soyez clairement identifiable (et donc en évitant les pseudonymes fantaisistes) :
      `git config --global user.name "Votre nom clairement identifiable"`
      — configurez le comportement par défaut de Git concernant les *rebase* lors d'un *pull* :
      `git config --global pull.rebase false` (nous vous déconseillons une autre configuration tant que vous ne maîtrisez pas le fonctionnement de Git)
   (d) Vous pouvez aussi configurer votre éditeur par défaut pour rédiger les commentaires. C'est particulièrement utile pour les personnes qui utilisent les machines GNU/Linux de l'école, la configuration par défaut pouvant être exotique :
      `git config --global core.editor nano` (nom de l'éditeur à adapter selon votre système, mais ce choix est sûr pour les machines de la famille Unix)
3. Création de projet Eclipse et intégration du dépôt Git (pour chaque membre du binôme)
   (a) Lancez Eclipse
   (b) Fermez tous les projets (pour se concentrer sur le fil rouge)
   (c) Créez un nouveau projet Java (*File > New > Java Project*)
      i. Dans la première fenêtre de l'interface de création (celle où vous devez donner un nom au projet), décochez la case « *Use default location* »
      ii. Cliquez sur le bouton qui vient de se dégriser (*Browse*)
      iii. Cherchez le répertoire de votre dépôt Git fraîchement cloné dans la fenêtre d'exploration du système de fichiers qui vient d'apparaitre, sélectionnez et validez-le
      iv. De retour sur la fenêtre de création, terminez votre configuration en cochant éventuellement des options selon vos goûts (création automatique de modules, *working sets*, etc.) et validez-la
4. Test de fonctionnement

   Si vous maîtrisez la perspective Git de Eclipse pour interagir avec votre dépôt Git, n'hésitez pas à l'utiliser. Si vous ne vous sentez pas très confiant, nous vous recommandons de gérer l'interaction avec Git dans le terminal (ce qui est plus simple pour se rapprocher du TP et des tutoriels « *IDE agnostic* » en ligne). Le test de fonctionnement consiste en la création d'un changement et de son partage :
   (a) À la racine de votre dépôt, dans Eclipse ou hors Eclipse, un seul des membres du binôme crée un fichier `.gitignore` contenant au moins les quatre lignes suivantes (vous le complèterez par la suite si besoin) :
   ```
   .project
   .classpath
   ```

```
bin/
*.class
```

(b) Dans le terminal (ou avec n'importe quel autre outil si vous le maîtrisez), placez-vous à la racine de votre dépôt Git et partagez le changement avec les quatre commandes suivantes :
`git add .gitignore` (construction du changement)
`git commit -m "creation of .gitignore"` (création-enregistrement du changement)
`git pull` (récupération d'éventuels changements venant d'ailleurs, inutile la première fois si vous avez suivi strictement la procédure et que votre binôme n'a pas effectué de *commit* entre temps ; commande néanmoins présente dans cette procédure pour que cela devienne une habitude avant chaque `git push`)
`git push` (diffusion du changement vers le dépôt Git distant)

(c) L'autre membre du binôme vérifie que le changement a bien eu lieu en tapant sur sa propre machine, à la racine de son dépôt, la commande `git pull`

5. Vous pouvez maintenant travailler sur votre fil rouge et interagir avec Git via le terminal, la perspective Git de Eclipse ou tout autre outil qui vous convient le mieux. Une manière de vous assurer que vous avez compris l'utilisation de Git est de créer votre premier fichier Java et de le partager avec votre binôme.

▷ **Question 4.1 :**
Implémentez la conception que vous avez finalisée. Quelle stratégie de développement suivez-vous ?

Évaluation - préparation

Aucun livrable ne vous est demandé à la fin de cette activité. Par contre, vous devrez :

— avoir dans le projet Redmine du binôme (ou dans l'espace de partage du code) votre projet Eclipse
— avoir commencé à compléter la grille d'auto-évaluation de l'usage de bonnes pratiques de développement. Elle est disponible dans la partie « Fil rouge » de l'espace Moodle de l'UE. Une fois complétée, vous devez la déposer dans votre espace de partage de documents sur Nextcloud. Elle vous sera utile pour connaître quelques éléments clés lors du développement d'un programme Java et en tenir compte pour la suite du projet (et au delà).

Exercice 5 (*TD4 : Relecture*)

Objectifs

On fait une pause sur les productions (modèles, code). On fait relire par un autre groupe ce qu'on a fait et on évalue le travail d'un autre groupe.

— Relire, commenter, critiquer le code d'un autre groupe.

Déroulement de la séance

On part des livrables précédents : modèle UML et code source. Travail en binôme.

1. Élaborer une grille d'analyse en commun

2. Distribuer la fiche de remarques
3. Faire une relecture croisée du code source
4. Échanger les grilles dans les 10 dernières minutes avec les développeurs originaux

Travail attendu

On fournira à un autre binôme la grille de relecture de son code.

Livrable attendu :
— Un rapport de relecture
Ce livrable sera utilisé pour améliorer le code.

Évaluation - retour "formatif" après TP5 IDL

Le livrable à déposer sur moodle devra contenir :
— Le projet exporté contenant le code source commenté implémentant votre modèle
— Un README (décrit le contenu et comment lancer le code et/ou les test ; des notes sur le lien entre le code et les modèles de conception [conforme sinon, description des modifications])
— Le dossier `src` avec les sources commentés
— Le dossier `tests` avec les sources des tests
— La grille d'auto-évaluation d'usage des bonnes pratiques
— Le rapport de relecture, complété par les *décisions* finales

Pour avoir un retour *formatif*, déposez sur moodle les attendus. L'enseignant qui vous suit vous fera un retour dans les meilleurs délais. La grille est disponible sur moodle. (CG 2, 4 & 10).

Compléments

Des adresses où trouver des règles de codage Java :

officielles `http://java.sun.com/docs/codeconv/html/CodeConvTOC.doc.html`
simples `http://www.bruno-garcia.net/www/Java/guide.html`

Exercice 6 (*TP 7 : PNE analyse statique*)

Objectifs

1. Citer des métriques de qualité logicielle (au moins 3) et expliquer leur intérêt
2. Observer des mesures de qualité de code sur un logiciel de taille moyenne et identifier des actions permettant d'améliorer les valeurs observées
3. (optionnel [1]) Installer des plugins dans Eclipse

1. À Brest, l'environnement de travail avec `SETUP MAPD` propose un Eclipse adapté à MAPD, avec les plugins pré-installés

Déroulement de la séance

1. Découvrir la notion de qualité logicielle et quelques métriques de qualité logicielle
2. Importer le logiciel PNEditor dans Eclipse et en faire une première utilisation
3. Découvrir le code source de PNEditor
4. (Optionnel) Installer le plugin Eclipse STAN et l'utiliser
5. Observer des mesures de qualité de code avec STAN et les analyser

Travail attendu

— Un environnement Eclipse étendu

Les exercices qui suivent sont destinés à vous introduire le domaine de la qualité logicielle et des métriques et à vous apprendre à collecter de l'information (des métriques de qualité) d'une application que vous ne connaissez pas. C'est un exercice essentiel, mais compliqué. Il faut sélectionner, trier, ne pas chercher à tout comprendre.

Vous tracerez dans un fichier vos réponses, observations et commentaires au fil de l'eau.

▷ **Question 6.1 :**
Reformuler avec vos propres mots la définition de qualité logicielle donnée sur Wikipedia (`https://en.wikipedia.org/wiki/Software_quality`).

Dans cette activité, et dans l'UE en général, nous nous intéressons plus particulièrement à la qualité structurelle des logiciels et des métriques associées. Ces métriques ont été regroupées par le CISQ (« *Consortium for IT Software Quality* ») dans cinq domaines (considérés comme les 5 caractéristiques structurelles les plus importantes pour un logiciel) : fiabilité, sécurité, maintenabilité, taille et efficacité. Vous trouverez dans la partie 3 de `https://en.wikipedia.org/wiki/Software_quality` des informations sur des métriques qui contribuent à qualifier les 5 domaines.

Dans un premier temps, vous allez découvrir le logiciel sur lequel vous allez observer ces métriques : un éditeur de réseaux de Petri, PNEditor [2].

Récupérez le code de l'éditeur qui est disponible dans l'espace Moodle de l'UE.

— Vérifiez que vous utilisez bien Java 1.8.
— Importez le projet (*File > Import > General > Existing Projects into Workspace...*) à partir du fichier `mapd-pne-code-project.zip` récupéré de l'espace Moodle.

▷ **Question 6.2 :**
Trouvez la classe `Main`, puis lancez l'application. Utilisez-la pour décrire un réseau avec quelques places, quelques transitions et quelques arcs. La figure 1 présente un exemple d'exécution de PNEditor.

▷ **Question 6.3 :**
Identifiez les répertoires, les paquetages, les classes principales. Trouvez l'endroit où la fenêtre principale est créée. Essayez de reconnaître les éléments de l'interface homme-machine (IHM).

Astuces de navigation dans Eclipse :

⚠ Quand un identifiant est sélectionné, on peut voir sa déclaration en utilisant la touche F3 (ou menu contextuel>Open Declaration)

2. Adaptation pédagogique de `http://www.pneditor.org`

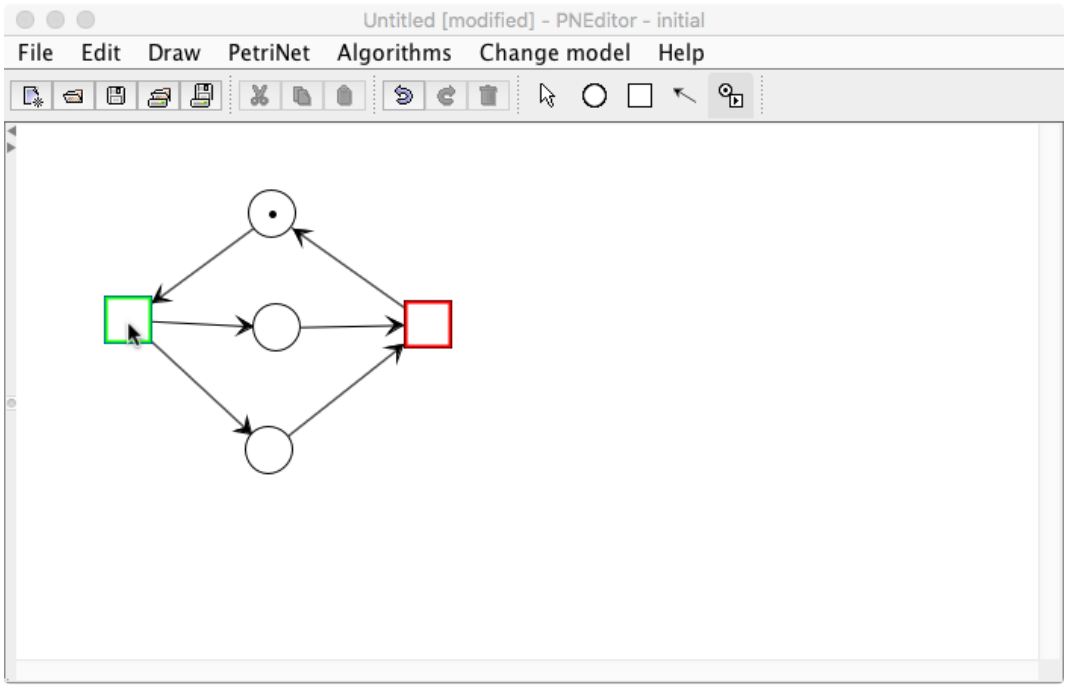


FIGURE 1 – Exemple d'interface lorsque PNEditor est exécuté

⚠ Pour voir qui appelle/utilise un identifiant, on peut utiliser la touche Ctrl-Opt-H (ou menu contextuel>Open Call Hierarchy)

Couplage et métriques. Vous allez maintenant vous intéresser à la qualité du code de PNEditor. Pour cela, vous allez utiliser l'outil d'analyse statique de code STAN [3]. Cet outil permet de visualiser les dépendances entre les paquetages, classes et méthodes et de calculer quelques métriques de qualité de code.

Si vous n'êtes pas à Brest sur une machine de l'école, installez le plugin STAN. Pour cela :

- Dans Eclipse Help>Install New Software... ajouter (Add) et remplir avec Name : STAN et Location : `http://update.stan4j.com/ide` puis suivre les consignes.
- Relancer Eclipse.

Puis, sous Eclipse, sélectionnez le projet (le nom du projet) et, à l'aide du menu contextuel, vous pouvez faire `Run As > Structure Analysis`

Lorsque vous lancez l'exécution de PNEditor avec STAN, vous vous retrouvez avec une fenêtre qui ressemble à celle de la figure 2 :

- La liste des paquetages (et classes) du projet Eclipse se trouve à gauche. En sélectionnant un élément ou un autre les résultats affichés seront différents.
- En haut à droite, une vue graphique du contenu du paquetage ou classe sélectionné à gauche ou une vue de la répartition des valeurs d'une métrique.
- À droite, une liste de métriques pour l'élément sélectionné à gauche. Sélectionner une métrique peut apporter des informations sur la répartition des valeurs. Dans ce cas, elles sont affichées en haut à droite.
- Les dépendances entre paquetages, classes ou méthodes au centre.
- Des informations en bas (« *Violations* »). Selon les seuils définis dans les préférences de

3. « *STAN - Structure Analysis for Java* »

STAN [4], certaines alertes sont levées.

STAN permet de visualiser quatre types de métriques :

- métriques de taille (« *Count* ») : nombre de librairies, paquetages, classes, classes/classe, méthodes/classe, etc. mais aussi le nombre de lignes de code (ELOC, ELOC/Unit)
- métriques de complexité (« *Complexity* ») : en particulier la complexité cyclomatique (CC) d'une méthode (`https://fr.wikipedia.org/wiki/Nombre_cyclomatique`)
- métriques de R. Martin (`https://www.future-processing.pl/blog/object-oriented-metrics-by-robert-martin/`)
- métriques de Chidamber et Kemerer (`https://www.aivosto.com/project/help/pm-oo-ck.html`) : en particulier CBO (nombre de classes auxquelles une classe est couplée) et LCOM (cohésion des méthodes d'une classe) pour mesurer le couplage dans le code

▷ **Question 6.4 :**
Utilisez STAN pour obtenir des métriques pour différents paquetages et classes. Puis, une fois que vous avez compris le fonctionnement, utilisez le document `https://stan4j.com/papers/stan-whitepaper.pdf` pour expliquer pourquoi certaines dépendances entre paquetages sont indiquées en rouge et en quoi cela peut être problématique. Visualiser par exemple la composition du paquetage `org.pneditor` pour un exemple de dépendance en rouge puis `org.pneditor.editor` pour un exemple de plusieurs de ces dépendances.

On va s'intéresser maintenant aux métriques proposées par STAN. Sélectionnez le paquetage `org.pneditor.editor.gpetrinet`.

▷ **Question 6.5 :**
Existe-t-il des cycles dans les dépendances entre les classes du paquetage ? Et avec d'autres paquetages ? (voir l'onglet « *Couplings* »).

Regardez l'onglet « *Metrics* », en particulier celles de Robert C. Martin. La valeur pour *D* est signalée en jaune. Donnez une explication en vous basant sur les informations données dans `https://www.future-processing.pl/blog/object-oriented-metrics-by-robert-martin/`. Qu'en concluez-vous sur ce paquetage ?

▷ **Question 6.6 :**
Sélectionnez l'onglet « *Pollution* ». Quels sont les métriques pour lesquelles le paquetage a des avertissements ? Quel(s) est(sont) le(s) artifact(s) (classes, méthodes) qui sont à l'origine de ces alertes ?

▷ **Question 6.7 :**
Faites le même exercice d'analyse de qualité de code pour le paquetage (cf. vue STAN) `org.pneditor.editor`.

Exercice 7 (*TP10 : Rétro-ingénierie*)

Objectifs

1. Utiliser un outil de retro-ingénierie pour générer le diagramme de classes d'un logiciel

4. Il est possible de modifier ces seuils en allant dans le menu d'Eclipse Window>Preferences>STAN>Metrics

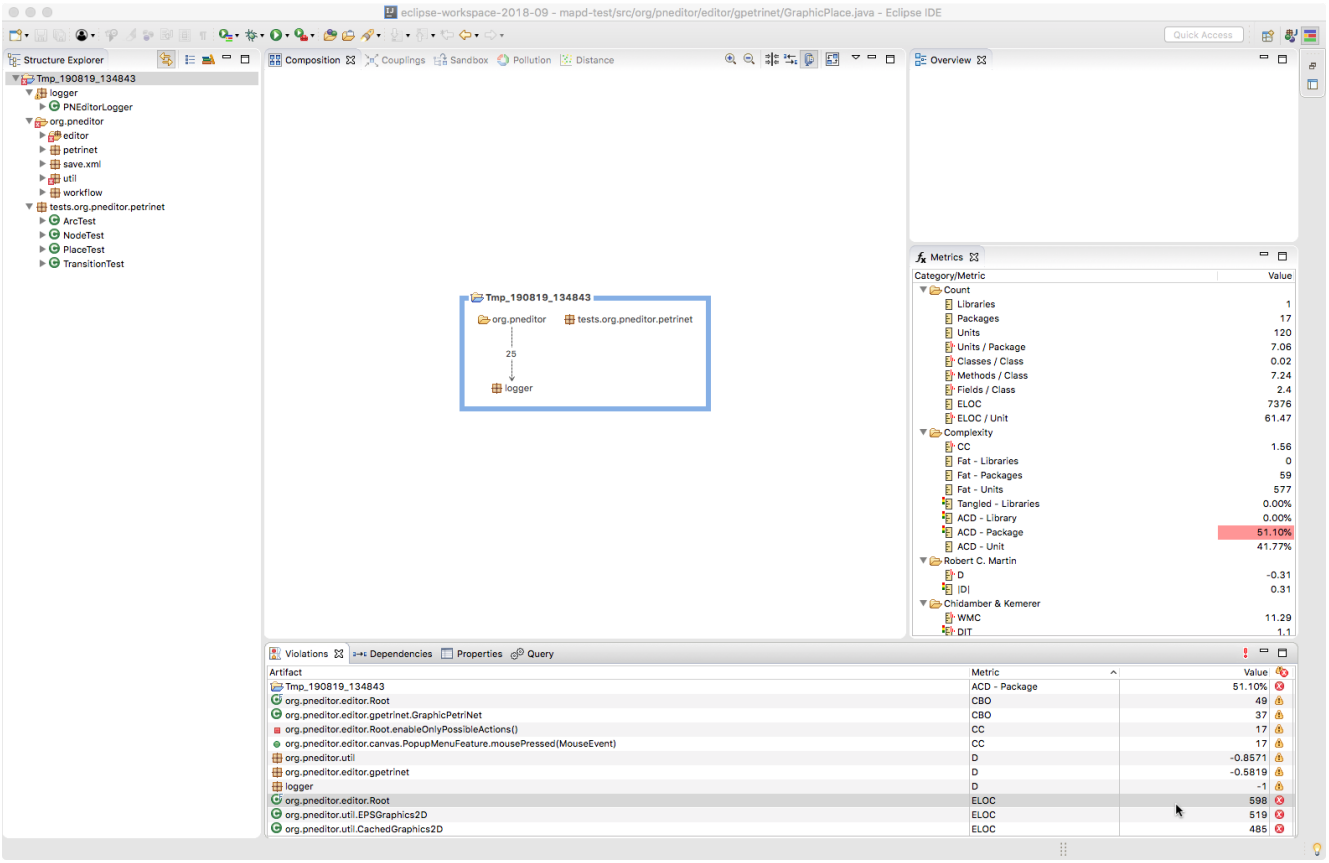


FIGURE 2 – Résultat de l'exécution de STAN sur le PNEditor

2. Comparer la rétro-ingénierie avec la conception initiale ; comparer, expliquer

Déroulement de la séance

1. (Optionnel) Installer plugin Eclipse ObjectAid et produire des diagrammes de classes

Travail attendu

— Un environnement Eclipse étendu
— Vos observations et une analyse critique du code de votre projet (1 page max)
— Le diagramme de classes de rétro-ingénierie de votre projet

Vous allez maintenant reconstruire un diagramme UML à partir du code source. On fait de la rétro-ingénierie. Pour ce faire vous allez utiliser le plugin Eclipse ObjectAid. Installez-le à partir du code disponible sur Moodle. (Help>Install New Software…) puis Add, Local ou Archive. (Si vous utiliser le SETUP MAPD, le pluggin ObjectAid est déjà installé.)

▷ **Question 7.1 :**
Ajoutez au projet Eclipse un répertoire (« *Folder* ») `diagrams` puis un diagramme de classes ObjectAid (New>Other>ObjectAid UML Diagram>ObjectAid Class Diagram). L'ajout des classes dans le diagramme se fait par *drag and drop*.

Commencez par le diagramme du paquetage `org.pneditor.petrinet` : l'interface et les quatre classes abstraites. Ajoutez ensuite quelques classes du paquetage

`org.pneditor.editor.commands` (`PasteCommand`, `AddPlaceCommand` par exemple) puis les classes ou interfaces `Root`, `UndoManager`, `Command`, `GraphicPlace`, `AbstractPlace`.

▷ **Question 7.2 :**
Ajoutez un autre diagramme de classes ObjectAid pour votre modèle de réseau de Petri. Ajoutez vos classes, interfaces pour retrouver votre modèle.

▷ **Question 7.3 :**
Comparer le modèle de votre implantation avec votre modèle de conception. Expliquer et justifiez les différences. Vous commenterez en particulier vos choix de navigabilités.

▷ **Question 7.4 :**
Pour finir, faites une analyse avec STAN de votre solution. Commentez.

Évaluation - préparation

Le livrable à déposer après le TP16 :
- Le diagramme de classes de rétro-ingénierie de votre implantation *finale* et l'explication des différences (ou pas) avec la conception initiale.
- Un commentaire de l'analyse statique de votre code avec STAN.

Compléments (pour votre culture générale ...)

Il existe de nombreux outils d'analyse statique de code. Par exemple :

- SpotBug (`https://spotbugs.github.io`) qui cherche des erreurs classiques en Java. Pour installer son plugin Eclipse lisez : `https://spotbugs.readthedocs.io/en/latest/eclipse.html`

  Pour analyser, lancer dans le menu du projet Spot Bugs>Find Bugs. Puis observer les résultats ; le nombre de détection apparait entre parenthèses. Il y en a très peu dans pneditor...
- PMD (`https://pmd.github.io`) qui cherche des défauts connus dans le code. Les défauts sont décrits par des règles qui décrivent la « forme » du défaut. Pour installer son plugin Eclipse lisez : `https://pmd.github.io/latest/pmd_userdocs_tools.html#eclipse`

  Dans les Preferences>PMD, pour configurer les règles activez « Gérer les règles globalement » puis groupez par « Langages » pour ne garder que Java, puis par « Jeu de règles » pour choisir le type d'analyse.

  Pour analyser, lancer dans le menu du projet PMD>Analyser le code avec PMD. Puis observer les résultats dans la perspective PMD associée.

Exercice 8 (*TP 11-TP 15&16 : Intégration*)

Objectifs

- Écrire un programme Java conforme à un modèle UML donné constitué d'un diagramme de classes UML

— Utiliser un IDE (type Eclipse) pour programmer et tester un programme Java qui respecte les bonnes pratiques de programmation générales et objet
— Écrire un programme Java qui intègre un composant logiciel existant

Déroulement de la séance

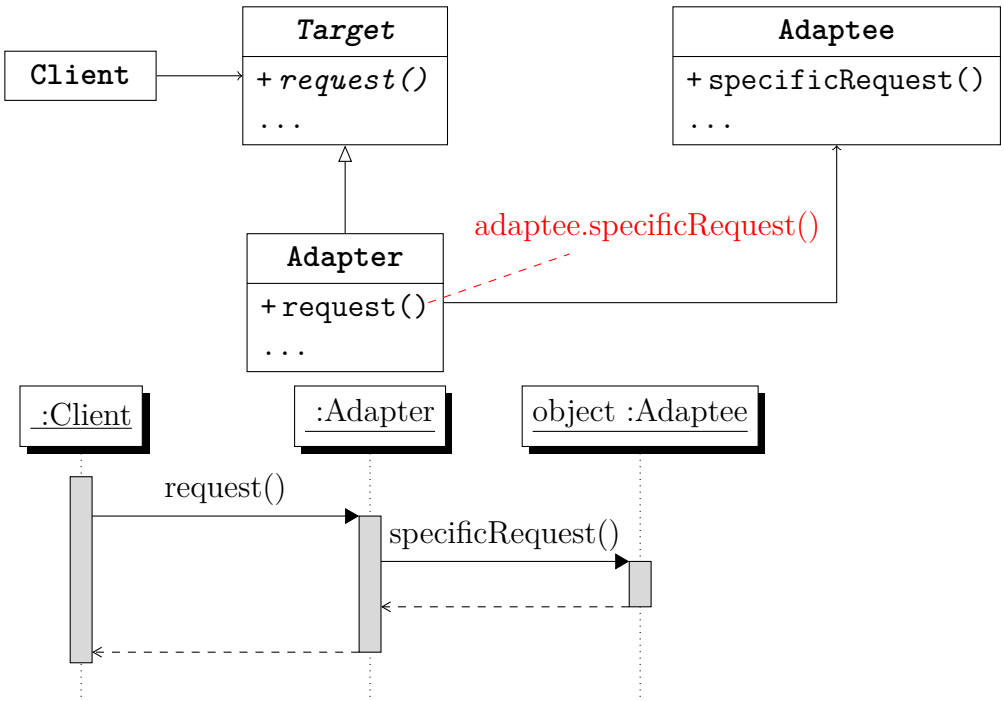
1. Découvrir le *pattern Adapter*
2. Ajouter du code au PNE dans le répertoire `org.pneditor.petrinet.models.<votreNom>` [5]
3. Réaliser l'adapteur dans le package `org.pneditor.petrinet.adapters.<votreNom>`
4. Utiliser le modèle via le PNE

Travail attendu

— Le PNE qui fonctionne avec votre modèle.

Vous allez intégrer le code écrit pour l'implantation du modèle de réseau de Petri (TP 3&4) avec le code de PNE.

Le code existant s'attend à une certaine interface avec son modèle qui n'est vraisemblablement pas celle que vous avez définie. Dans cette situation, un *pattern* classique est *adapter* dont voici une description simplifiée :



`Target` est l'interface attendue par le client (PNE) ; `Adaptee` est votre interface. Il faut écrire la classe `Adapter` qui « traduit » les requêtes.

▷ **Question 8.1 :**

1. **Identifiez la classe/interface qui joue le rôle de `Target` dans le PNE.**
2. **Identifiez dans ce même package les classes abstraites qui seront vos cibles pour l'adaptation, la rétro-ingénierie vous donnera les indications nécessaires.**

5. Alternative ; on pourrait utiliser l'adapteur pour rediriger vers le bon package aussi...

3. **Vous devez comprendre la correspondance entre les différents types d'objets du PNE (regardez le type de l'arc) et ceux de votre modèle.**

▷ **Question 8.2 :**
Implantez et testez.

⚠**Pour être accessible par le menu « Change model », la classe d'adaptation du réseau de Petri *doit* s'appeler `PetriNetAdapter` et être dans le package `org.pneditor.petrinet.adapters.<votreNom>`**

▷ **Question 8.3 :**
Effectuez la rétro-ingénierie de votre adaptation en utilisant ModelGoon (ou ObjectAid).

Évaluation - à déposer sur moodle après TP16

Le livrable à déposer sur Moodle après le TP16 :
- Le diagramme de classes de rétro-ingénierie de votre implantation *finale* et l'explication des différences (ou pas) avec la conception initiale.
- Un commentaire de l'analyse statique de votre code avec STAN.
- Le projet exporté contenant le code source commenté implémentant votre modèle, l'adaptateur et le PNE
  - Un README (décrit le contenu et comment lancer le code et/ou les test ; des notes sur le lien entre le code et les modèles de conception [conforme sinon, description des modifications])
  - Le dossier `src` avec les sources commentés
- La grille d'auto-évaluation d'usage des bonnes pratiques

Pour l'évaluation *sommative*, déposez sur moodle vos documents. Le livrable sera évalué et contribue à la validation de compétence de l'UE. (CG 2, 3, 4 & 10). La grille est disponible sur moodle.