

TME

MU4IN502- Ingénierie du Logiciel

Yann Thierry-Mieg

2019/2020

Supports en ligne : <https://pages.lip6.fr/Yann.Thierry-Mieg/IL/>

SUBOO : Sorbonne Université Build Order Optimizer

Dans un jeu de stratégie temps réel (STR) comme StarCraft II, le joueur doit trouver un équilibre économique entre la construction d'ouvriers (qui augmentent le revenu) la construction de bâtiments de technologie (qui débloquent de nouvelles unités) et la construction d'unités d'attaque (pour gagner la partie).

Le joueur démarre avec un Centre, une banque constituée de 50 PO (Pièces d'Or) et 5 ouvriers.

Le jeu considère deux types de ressources : l'or et la nourriture. A chaque instant le joueur dispose d'une banque constituée des PO qu'il a récolté et économisé, et un total de nourriture N qui correspond à la quantité maximum d'unités que le joueur peut construire avec les bâtiments qu'il possède.

Chaque unité du jeu possède des caractéristiques qui lui sont propres (cf tableau). On considère une version simplifiée du jeu avec un nombre limité de types d'unités.

Les ouvriers construisent des fermes pour augmenter la nourriture disponible, des casernes qui permettent de produire des soldats, et (en général un seul) Hall sans lequel la caserne ne peut pas construire des Boss. Quand ils ne sont pas mobilisés par une construction, ils produisent 1PO par ouvrier et par unité de temps. Les soldats et les Boss sont des unités d'attaque, le but du jeu est d'en construire beaucoup et vite.

Chaque unité ou bâtiment a un coût de construction (or et nourriture). Pour engager la construction d'une unité il faut posséder le coût indiqué dans la table en PO et en nourriture N. Ce coût en ressources est consommé dès que la production de l'unité est lancée. Mais l'unité démarre *en construction* pendant son « Tps Cstr ». Les unités coûtent donc de la nourriture même si elles sont encore en construction. Les bâtiments (Centre, Ferme, Caserne, Hall) ne coûtent pas de nourriture, mais les Fermes et Centre en construction ne contribuent pas de nourriture N.

Catégorie	Unité	Cout PO	Cout N	Tps Cstr	Prérequis	Mobilise	Fournit
Bâtiment	Centre	300		200	Ouvrier	Ouvrierx200	+6N
	Ferme	100		60		Ouvrierx100	+6N
	Caserne	150		100	Ouvrier,Ferme	Ouvrierx100	
	Hall	200		120	Ouvrier,Caserne	Ouvrierx120	
Unité	Ouvrier	50	1	30	Centre	Centrex30	1PO/s
	Soldat	100	1	50	Caserne	Casernex50	
	Boss	200	3	100	Hall,Caserne	Casernex100	

Certaines unités ont des prérequis, par exemple il faut une ferme pour produire une caserne, et il faut un Hall pour que la caserne puisse produire des Boss. Les prérequis sont satisfaits dès qu'une unité du type demandé est présente. De plus, chaque construction d'unité mobilise un ouvrier ou un bâtiment pendant une certaine durée.

On considère que les unités dans une partie peuvent être dans l'un des trois états suivants :

- *en construction* les unités nouvellement créées sont dans cet état pendant une durée déterminée par leur temps de construction. L'unité devient *disponible* à la fin de son temps de construction.
- *mobilisé* les bâtiments qui construisent une unité sont mobilisés, de même que l'ouvrier qui construit le bâtiment. L'unité redevient *disponible* quand elle finit sa mobilisation.
- *disponible* si une unité n'est ni en construction, ni mobilisée, c'est qu'elle est disponible. On considère que dans cet état les ouvriers génèrent naturellement 1PO par unité de

temps. Seules les unités *disponibles* peuvent être mobilisées pour une construction.

Pour se développer on peut construire de nouveaux ouvriers dans le Centre mais on sera vite limité par la nourriture N. Il faut donc construire des Fermes pour entretenir une population plus importante. Si on atteint le seuil de nourriture, on ne peut plus construire de nouvelles unités jusqu'à ce que la situation soit améliorée.

Un *état* d'une partie est caractérisé par la banque (PO disponible) et par les unités que le joueur possède à cet instant. Pour chacune des unités, le jeu connaît son état à cet instant et le temps restant avec qu'elle ne devienne disponible si elle ne l'est pas.

Le jeu est mis à jour à chaque unité de temps ; à cette étape les ouvriers disponibles (qui récoltent donc) ajoutent 1 PO par ouvrier à la banque. On comptabilise les ressources en nourriture disponible à partir du décompte des bâtiments terminés (mobilisé ou non) présents.

On peut prendre à cet instant une décision qui consiste à produire une unité si les prérequis spécifiques à l'unité sont satisfaits, si la banque le permet, et que la limite de nourriture le permet. Le jeu boucle en interrogeant la prise de décision à chaque pas de temps.

Un ordre de construction (ou Build Order BO) est une séquence d'actions de construction. Un BO est réalisable si on peut l'exécuter en respectant les contraintes du jeu décrites au-dessus en attendant une durée arbitraire entre chaque action (i.e. les prérequis sont satisfaits). Son temps d'exécution (idéal) est calculé avec l'attente la plus courte possible entre chaque action.

Notre application devra construire le BO réalisable le plus rapide possible pour atteindre un objectif donné. L'objectif est défini par un certain nombre d'unités de chaque type (qu'on souhaite obtenir) et/ou une banque à atteindre.

Par exemple, obtenir 5 Boss et 12 attaquants le plus vite possible. Ou obtenir 2000PO en banque le plus vite possible.

On demande que SUBOO puisse supporter plusieurs versions du jeu : l'arbre de technologie et le prix des unités varient régulièrement au fil des versions du jeu. L'utilisateur doit pouvoir configurer un objectif, défini par un certain nombre d'unités de chaque type et une banque cible. Avec cet objectif, il peut obtenir un BO naïf (mais réalisable) qui permet de l'atteindre : seuls les éléments strictement nécessaires sont produits. Réciproquement, en fournissant un BO, l'utilisateur doit savoir s'il est réalisable et calculer sa durée idéale. S'il n'est pas réalisable, l'outil doit proposer une correction du BO (en ajoutant des actions). Enfin l'utilisateur doit pouvoir obtenir un BO optimisé, qui minimise le temps pour atteindre un objectif. L'utilisateur veut le meilleur BO possible, même si le calcul est coûteux. Enfin, l'outil devra pouvoir interagir avec une IA qui joue au jeu en temps réel. Dans ce cadre l'IA fournit régulièrement pendant la partie une observation de l'état actuel du jeu (au lieu de l'état initial) et un objectif qu'il faut optimiser rapidement (prise de décision court terme). Ces échanges doivent utiliser une API et non une IHM.

Organisation des TME

L'objectif des séances de TME est de construire une spécification et une implémentation du projet StoneHearth. Le travail sera collaboratif par équipes de 4 à 6 étudiants.

Au cours de la première séance de TME il vous est demandé de constituer des groupes, qui seront sauf cas exceptionnel figés pour le reste de l'UE.

Le travail de chaque étape sera réparti sur les membres de l'équipe. L'ensemble du travail du groupe sera présenté par l'équipe, à travers deux présentations (mini soutenances) en séances 5 et 10 de TME. Pour cela vous préparerez des transparents (powerpoint, keynote ou autre) qui seront présentés conjointement par les membres de l'équipe. Il est demandé qu'au cours de ces présentations tous les membres du groupe s'expriment.

- En séance 5, la présentation devra couvrir toute la phase d'analyse. On présentera donc les acteurs et cas d'utilisation du système, sa structure via le diagramme de classes métier, des ébauches d'écran d'interface, et des diagrammes de séquence de niveau analyse expliquant les responsabilités identifiées du système. On présentera également certains des tests de validation.
- Une présentation d'équipe, à la fin de l'ue (séance TME 10) couvrira la phase de conception. En 20 à 30 minutes (selon le nombre de groupes d'étudiants dans le groupe de TD) le groupe devra présenter les grandes lignes du travail réalisé, et dans la mesure du possible faire la démo des fonctionnalités opérationnelles. On présentera l'architecture retenue pour la solution (composants, interfaces, instanciation nominale), des tests d'intégration pour au moins deux composants, et les principales séquences d'interaction niveau composant. La conception détaillée des composants sera présentée à l'aide d'un diagramme de classes par composant.

L'ensemble de ces éléments participera à la note de TME, comptant pour 10% de la note totale d'UE.

Pour la coordination du groupe, il est demandé de mettre en place rapidement un email de groupe, voire de créer un groupe sur un site communautaire (google group par exemple). Cela permet de disposer d'une zone où partager les documents facilement. Pour la gestion partagée des modèles et du code, une solution basée sur svn pour la gestion de version est mise en place à l'ARI.

Rendez-vous donc sur : <https://svn-trac.ufr-info-p6.jussieu.fr>

L'authentification sur ce site est la même que celle utilisée en salles TME. Après authentification, un Wiki donne les instructions à suivre afin de lister/créer/supprimer/modifier les projets svn/trac. Des vidéos tuto sont disponibles sur le site de l'ue pour vous guider dans la mise en place du projet.

Attention l'édition à plusieurs du fichier de modèle peut poser des soucis de conflits. Il est recommandé de bien faire attention aux éditions concurrentes du fichier.

La suite de ce document place quelques marqueurs pour vous guider au cours de votre travail ; l'objectif global est d'appliquer en groupe la méthodologie présentée en cours et TD à l'exemple du eCours.

NB : L'exemple utilisé les années précédentes (iSudoku) est présenté avec un corrigé détaillé sur la page de l'UE. Cet exemple à vocation à servir de référence pour votre travail.

TME1

Prise en main

L'objectif de ce TME est de présenter l'outil Rational Software Architect Designer et d'illustrer les différentes possibilités qu'il offre.

Notez qu'il vous sera demandé une très grande **autonomie** lors de l'utilisation de cet outil tout au long des TME. En cela, n'oubliez jamais d'utiliser l'aide (**touche F1**) avant de poser vos questions. Il y a une page d'accueil (« Help->Welcome ») qui peut être utile aux débutants. L'outil est disponible en version 9.6 sous linux (dans /usr/local/IBM/SDP/eclipse). Il faut un workspace, que vous placerez dans votre home. Attention, au premier démarrage, un certain nombre de fichiers de configuration sont créés ce qui ralentit le démarrage.

COMMENCEZ PAR IMPORTER LES PREFERENCES:
Fichier->Importer->General->Preferences et pointer le fichier
/usr/local/ibm/preferences.epf

Répétez cette procédure à chaque changement de Workspace)

Notons que cet outil est basé sur Eclipse, et qu'une familiarité avec cet IDE sera un sérieux avantage au cours des premières séances.

Partie 1 : Découverte d'une application aux travers des fichiers sources

Récupérez les sources d'une application java que vous avez écrite (d'au moins 500/1000 lignes), ou le projet exemple fourni « iSudoku » sur le site de l'UE.

A l'aide de l'outil, construisez un diagramme de classe par package.

Pour ce faire, utilisez la séquence :

- Sélection des classes à afficher
- Bouton droit->Visualize->Add to new diagram file->class diagram
- Déplacez/arrangez les classes
- Jouez un peu avec les filters, on peut le faire sur une multi sélection de classes:
 - Sélection d'une classe->Filters->Show/Hide Relationship puis décochez le « use » et cocher le « Collection »
 - Filters->show signature permet d'afficher la signature des opérations. Il faut par contre configurer le plugin Java pour avoir les types de retour via :Window->Preferences->Java->Appearance->show return type in signature
 - Filters->Stereotype and visibility style -> visibility style text permet de modifier l'affichage des visibilité

Voyagez entre le modèle et le code : double cliquez sur un attribut ou une méthode pour atteindre le source. Modifiez le source ou le modèle, l'autre est mis à jour de façon synchronisée.

Essayez de construire un diagramme de séquence, en sélectionnant une opération et en faisant Visualize->new diagram file->static sequence diagram. Choisissez une opération un peu complexe de préférence.

Essayez l'outil Software Analyzer en cliquant sur un projet Java. Bouton droit sur le projet->Software Analyzer, créez une nouvelle configuration (un peu comme le menu Run) et choisissez les règles Java dans la deuxième page.

Partie 2 : Découverte d'une application au travers d'un modèle UML

Créez un nouveau projet de nature UML (New->Project->Modelling->Uml Project).
On utilisera le paramétrage par défaut avec un « General->Blank Package »

Créez un diagramme de use case, sélection du package dans le modèle -> Add Diagram->Use case diagram. Ajoutez un sous-système, quelques use case et un ou deux acteurs. Liez les acteurs aux use case par des « bidirectional association ».

Créez un diagramme de classes et au moins deux classes et une association entre les deux. Pour ajouter des opérations à la classe, utilisez le menu flottant, le rectangle ajoute des attributs, la roue crantée ajoute des opérations. On peut aussi quand on flotte sur une classe avec la souris utiliser les flèches qui apparaissent pour tirer des liens. Ces menus s'adaptent à l'objet sélectionné, sur une opération cela permet d'ajouter des paramètres par exemple.

Créez un diagramme de séquence. Tirez un de vos acteurs et une de vos classes, du modèle à gauche directement sur le diagramme. Ajoutez une invocation de l'acteur sur la ligne de vie (synchronous message). De nouveau le plus facile est d'utiliser le menu flottant avec les flèches. On notera que le système propose d'invoquer une opération existante ou d'en créer une à la volée.

Essayez de créer une transformation UML->Java ou Java->UML. Sélection d'un projet->Transform->New configuration, on choisira le mode « Conceptual » puis on sélectionne une source (e.g. un modèle ou package UML) et une cible (un projet Java existant ou qui sera créé à la volée). Utilisez le bouton « Run » pour lancer la transformation.

S'il vous reste du temps, créez d'autres types de diagrammes et explorez les possibilités offertes par l'outil.

TME2

Analyse de SUBOO

L'objectif de ce TME est de réaliser la phase d'analyse de l'application « SUBOO ».

Lancez RSA.

Créez un projet UML vierge:

- nom : eCours_NomEquipe.

Ce projet sera l'unique projet pour tous les TME (TME 2 à TME7)

Ce projet contiendra les phases d'analyse, de conception et de réalisation de code !

Q1 : Construisez un modèle nommé « analyse ». Sélectionnez ce modèle et construisez un nouveau diagramme de cas d'utilisation. Dans ce diagramme, définissez les acteurs et les cas d'utilisation de l'application.

Q2 : Sélectionnez le package « analyse » et construisez un nouveau diagramme de classes. Dans ce diagramme de classes, définissez les classes d'analyse de l'application. N'hésitez pas à utiliser beaucoup de notes pour commenter votre diagramme.

On les pose avec la palette du diagramme, à droite, dans l'onglet « UML common », ou via menu flottant : tirez un lien à l'aide des flèches flottantes depuis un objet à commenter du diagramme, terminez le geste dans le vide, « Create reference To->new element : Comment ».

TME3

Tests de validation

L'objectif de ce TME est de finir la phase d'analyse de « eCours » commencée lors du TME2. Les tests de validation seront écrits à l'issue de cette phase.

Lancez sur votre projet du TME2.

Q1 : Pour chaque cas d'utilisation, sélectionnez le cas d'utilisation et remplissez sa documentation, sous la forme d'une fiche détaillée. La documentation est accessible dans un des onglets de la vue Properties du cas d'utilisation (Window->view->Properties si elle n'est pas déjà visible). On dispose d'un éditeur RTF.

Il est également possible d'éditer vos fiches détaillées à l'aide d'un éditeur de texte de votre choix, puis de les placer simplement dans votre projet.

Essayez de générer la documentation de votre système, à l'aide du bouton Modeling->Publish, visible quand un modèle est sélectionné dans la perspective Modeling. Explorez les options et onglets disponible sur la page de publication ; cochez l'option de l'onglet « Properties » : Remove empty properties. Positionnez un répertoire de sortie raisonnable (attention ça crée toute une arborescence de fichier à partir de là).

Complétez un peu votre modèle pour améliorer sa documentation. On pourra aussi jouer avec les options du menu « publish ».

NB : Le suivi des besoins (requirements) se fait en principe dans la suite Rational à l'aide de l'outil « Requisite Pro » qui permet de lier les use case et autres artifacts du modèle à un modèle des requirements. Cet outil spécialisé est dédié au métier d'analyste du besoin, et s'intègre à RSA, mais n'est pas déployé sur vos machines. Il permet des fonctions avancées de suivi des besoins à travers le cycle de vie, génération de matrices de traçabilité entre autres. Cependant dans le cadre de ce module il paraît un peu « overkill », on se contentera donc des notes de documentation.

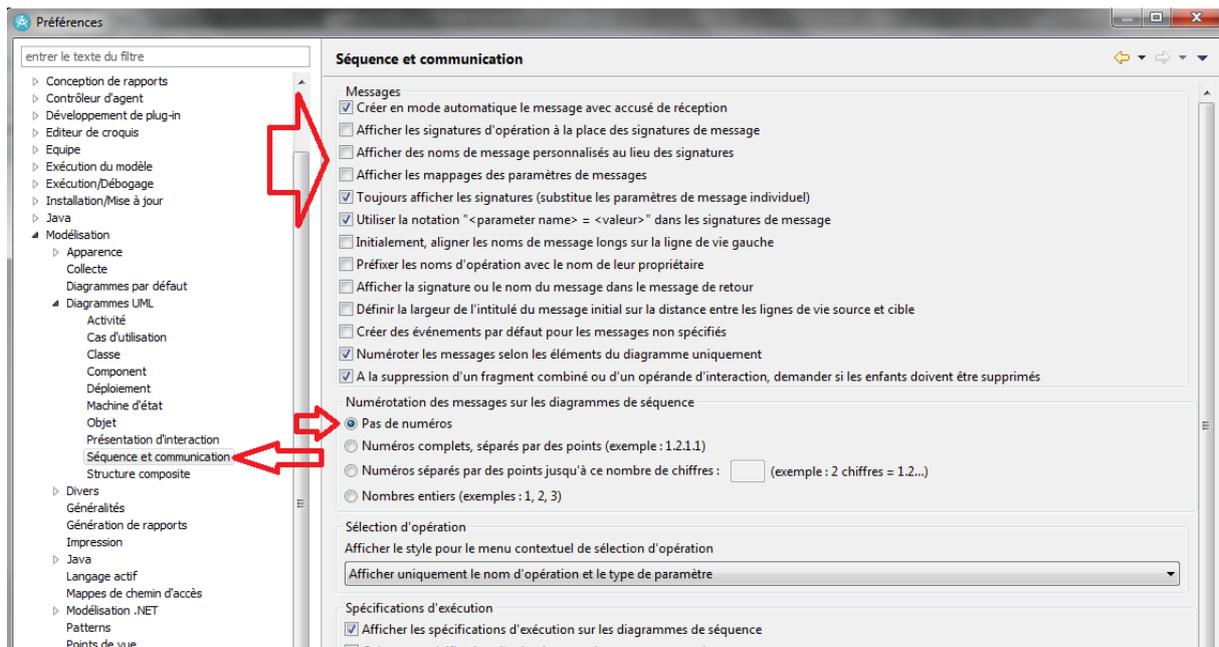
Q2 : Ouvrez le diagramme de cas d'utilisation. Pour chaque cas d'utilisation, sélectionnez le cas d'utilisation et construisez un nouveau diagramme de séquence. Définissez dans ce diagramme la séquence nominale correspondant au cas d'utilisation. N'oubliez pas de lier les instances appartenant à vos diagrammes de séquence avec les classes que vous avez déjà définies. On introduira au passage la classe représentant l'application et on enrichira ses opérations.

Note : pour construire la signature des opérations, plusieurs options sont possibles. Via la « Properties view » on peut éditer dans l'onglet « Parameters » les arguments de l'opération.

On ajoute les paramètres un à un et on édite les champs de la table. Une autre approche beaucoup plus commode consiste à utiliser la syntaxe textuelle, quand on édite le nom de l'opération : nomOperation ([in] param1 : Integer , [in] param2 : String) : Boolean.

La complétion/correction au cours de la frappe est disponible. Un clic sur la méthode quand elle est sélectionnée (attention pas un double clic) permet d'ouvrir l'édition textuelle.

NB : On positionnera les Préférences des diagrammes de séquence comme sur ce screenshot (attention surtout aux 5 premières options) ou en chargeant le fichier fourni
Fichier->Importer->General->Preferences : /usr/local/ibm/preferences.epf .



Q3 : Pour chaque cas d'utilisation, construisez les autres diagrammes de séquence définissant l'ensemble des comportements voulus par le cahier des charges de l'application.

TME4

Q1 : A l'aide des diagrammes de séquence et des fiches détaillées des cas d'utilisation, élaborer l'ensemble des tests de validation. On saisira les tests de validation dans les notes de documentation accompagnant les use case qu'ils valident ou dans un document séparé.

Générez encore votre documentation. Selon les groupes de TD, on peut vous demander de rendre ce travail pour contribuer à votre note CC.

TME5

Cette séance est dédiée à la présentation de l'analyse réalisée. On divisera la séance par le nombre de groupes, chacun faisant une présentation de son travail.

On s'attend à trouver dans cette présentation :

- une présentation rapide des membres du groupe, et de l'organisation mise en place (svn...)
- une présentation des acteurs et cas d'utilisation identifiés
- un zoom sur une ou deux fiches détaillées de cas d'utilisation
- la description des principales classes métier identifiées
- quelques diagrammes de séquence de niveau analyse
- quelques tests de validation.

Il est fortement recommandé de faire une répétition en groupe avant la présentation. Tous les membres du groupe devront s'exprimer.

Les transparents utilisés au cours de la présentation seront également donnés au responsable de TD par mail (format pdf).

TME6

Conception

Q1 : Construisez un package « conception » par copie de votre modèle d'analyse. On prendra soin de travailler dans cette copie, pour préserver l'état de votre spécification à la fin de l'analyse.

Q2 : Proposez une découpe en composants. Pour chaque composant, introduisez un package puis réalisez les diagrammes de classes du composant en découpant les classes métier.

Q3. Dans le modèle de « conception », réalisez un diagramme présentant tous les composants et leurs interfaces requises/offertes.

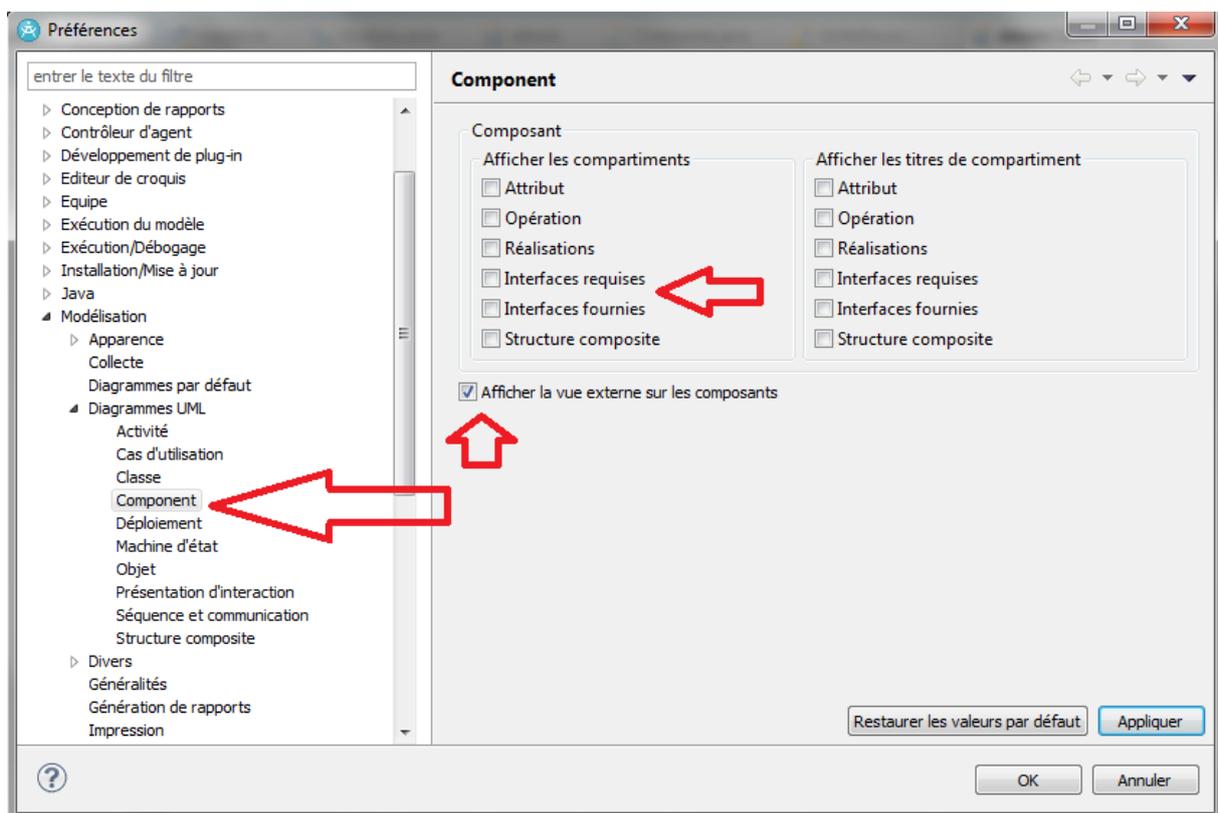
On prendra soin de distribuer les responsabilités de l'application sur des interfaces nouvellement introduites.

Q4. Elaborez les diagrammes de séquence correspondant aux cas d'utilisation de l'application (ceux définis en analyse). On reste sur une granularité « inter-composants ».

On enrichira au fur et à mesure les interfaces et les diagrammes de classe des composants de l'application élaborés à la séance 4.

Q5. Introduisez un composant « eCours » muni du stéréotype <<subsystem>>. Définissez sa structure interne en instanciant et en connectant les composants que vous avez défini, selon la configuration nominale envisagée.

NB : Dans les Préférences des diagrammes de composant, sélectionnez « Vue externe » comme sur ce screenshot.



TME7

Conception Détaillée

Q1 : A partir des diagrammes de classe partiellement établis pour les composants, complétez les à l'aide de Façade pour vous rapprocher d'une structure implémentable en Java. On évitera cependant de s'enfoncer trop profondément dans la conception détaillée.

NB : Si la conception n'est pas encore satisfaisante, itérez pendant une séance sur les questions des TME 6 et 7. Le sujet 8 est le dernier qui demande encore de modéliser. La séance 9 seront consacrées à améliorer les spécifications et modèles déjà construits, et à attaquer la réalisation. Les projets de TME seront évalués au cours de la séance 10 (voir avec le chargé de TD pour les modalités).

TME8

Tests d'intégration

Q1 : A partir des séquences d'interaction inter composants, définissez les tests d'intégration. Construisez le document des tests d'intégration. Définissez en particulier les jeux de données nécessaires à ces tests.

Q2 : Construisez les classes/composant bouchons permettant un développement « en parallèle » de chacun des sous-modules.

Q3 : Représentez sur des diagrammes de structure interne des configurations de test d'intégration des composants.

TME9

Vers la réalisation

Q2 : Configurez et lancez une transformation UML vers Java, pour les package qui correspondent à vos composants.

Q3 : En travaillant avec les diagrammes « Vue sur le code » et en développement Java pur, implémentez un maximum des cas d'utilisation identifiés.

NB :

1. il peut y avoir des divergences entre votre code et le modèle de conception, essayez de les identifier, mais il n'est pas demandé de chercher à maintenir à tout prix la cohérence entre les deux.
2. La notation cherche à évaluer votre capacité à spécifier et à concevoir une architecture, pas vraiment votre niveau en Java. Le « produit fini » n'est donc pas le facteur prédominant dans la note de TP.

TME 10

Soutenance

Cette séance est de nouveau dédiée à une présentation de votre projet.

On s'attend à trouver dans cette présentation :

- La description des composants identifiés et de leurs interfaces requises/offertes
- Quelques diagrammes de séquence de niveau interaction inter-composant
- Un diagramme donnant l'instanciation nominale des composants
- La présentation de quelques tests d'intégration : le test lui-même ainsi que la structure à mettre en place pour le réaliser (bouchons, testeurs...)
- Un ou plusieurs diagrammes de classe de niveau conception détaillé modélisant la structure interne de composants
- Selon l'avancement du code, un ou plusieurs diagrammes obtenus par synchronisation/reverse à partir du code
- Eventuellement une petite démo des fonctions offertes si la réalisation est aboutie.

Les transparents utilisés au cours de la présentation seront également donnés au responsable de TD par mail (format pdf).

Un rapport rédigé comme celui du corrigé du Sudoku est un plus, référez-vous aux consignes du chargé de TD pour plus de détails.