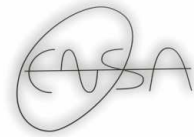


Université Mohammed Premier
Ecole Nationale des Sciences Appliquées
Oujda



Introduction à :

POSEIDON FOR UML

Réalisé par :

BENADDOU Mounir
MOUMEN Younes

Encadré par :

BOUCHENTOUF Toumi
BELLKASMI GHAOUTH Mohammed

Année universitaire 2004/2005

Sommaire

Introduction	3
Approche objet pour créer des solutions efficaces	4
Généralités	4
Définition du modèle	4
L'analyse objet	4
UML	5
Les vues UML	6
Vues statiques	6
Vues dynamiques	11
Les outils UML	15
Poseidon for UML	18
Introduction	18
Community Edition	18
Standard Edition	18
Professional Edition	19
Enterprise Edition	19
Embedded Enterprise Edition	20
Tableau comparatif	20
Installation et prérequis	22
L'interface	24
Cadre de navigation	24
Cadre des diagrammes	24
Cadre des détails	25
Cadre de vue d'ensemble	25
Fonctionnalités avancées	26
Génération du code	26
Reverse engineering	26
Round-trip engineering	26
Velocity Template Language	27
Génération de la documentation	27
Plug-ins	27
Profils	28
Intégrer Poseidon for UML à Eclipse	28
La perspective Java	30
La perspective UML	30
Cas d'utilisation	31
Présentation du problème	31
Diagramme de classes	31
Diagramme d'objets	35
Diagramme de collaboration	37
Diagramme d'états	38
Fonctionnalités avancées	39
Conclusion	42

Introduction

L'utilisation de plus en plus accrue d'UML dans le monde du développement logiciel nous a poussé à approfondir nos connaissances à son sujet. Il est difficile d'ignorer une conception UML lors du développement d'un projet basé sur une modélisation objet. Non seulement le temps de développement sera plus long mais aussi le maintien et la communication du projet aux autres seront moins évidents.

Ce document présentera quelques notions fondamentales sur la modélisation objet en général et UML en particulier. Puis il présentera l'un des outils UML les plus répandus : Poseidon for UML. Nous allons commencer par faire un tour d'horizon sur les différentes fonctionnalités de l'outil puis on procédera à l'étude d'un exemple à la manière d'un tutorial afin de mieux cerner son utilisation.

Le but de ce document est de fournir un moyen de prise en main facile de Poseidon for UML. Sans entrer dans les détails, nous allons essayer de guider un débutant pressé d'utiliser un outil UML à faire ses premiers pas et réaliser rapidement son projet.

Approche objet pour créer des solutions efficaces

Généralités

Définition du modèle

Un modèle est une abstraction de la réalité. L'abstraction est un des piliers de l'approche objet. Il s'agit d'un processus qui consiste à identifier les caractéristiques intéressantes d'une entité, en vue d'une utilisation précise. L'abstraction désigne aussi le résultat de ce processus, c'est-à-dire l'ensemble des caractéristiques essentielles d'une entité, retenues par un observateur.

Un modèle est une vue subjective mais pertinente de la réalité. Il définit une frontière entre la réalité et la perspective de l'observateur. Ce n'est pas "la réalité", mais une vue très subjective de la réalité. Bien qu'un modèle ne représente pas une réalité absolue, il reflète des aspects importants de la réalité et en donne donc une vue juste et pertinente.

Le caractère abstrait d'un modèle doit notamment permettre de faciliter la compréhension du système étudié en réduisant sa complexité. Un modèle représente le système étudié et reproduit ses comportements. Il réduit (décompose) la réalité, dans le but de disposer d'éléments de travail exploitables par des moyens mathématiques ou informatiques. La différence qui existe entre un modèle et la réalité peut être assimilée à la différence entre le numérique et l'analogique respectivement.

L'analyse objet

L'analyse objet a connu l'émergence de plus de 50 méthodes objet. Les plus connues entre elles sont : Booch, Classe-Relation, Fusion, HOOD, OMT, OOA, OOD, OOM, OOSE... Aucune de ces méthodes n'a su réellement s'imposer. La question fondamentale posée lors de l'élaboration de ces méthodes était comment structurer un système sans centrer l'analyse uniquement sur les données ou uniquement sur les traitements (mais sur les deux) ?

Mais depuis 1995, il y'a eu certains consensus sur les méthodes :

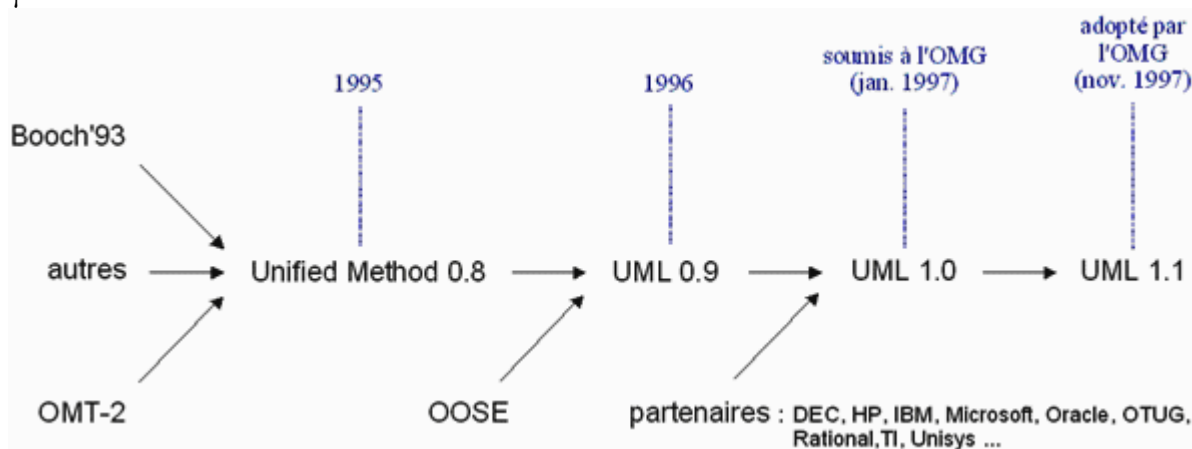
OMT : Issue du centre de R&D de General Electric. Elle fournit des vues statiques, dynamiques et fonctionnelles d'un système avec une notation graphique riche et lisible.

OOD : Définie afin de rationaliser le développement d'applications ADA, puis C++. Elle fournit des logiques et physiques du système et introduit le concept de package (élément d'organisation des modèles). Mais elle ne couvre pas la phase d'analyse dans ses 1^{ères} versions.

OOSE : Issue d'un centre de développement d'Ericsson, en Suède. Elle couvre tout le cycle de développement. Cette méthodologie repose sur l'analyse des besoins des utilisateurs.

UML

L'unification et la normalisation des méthodes (1995-1997) ont engendré UML (Unified Modeling Language), la fusion et synthèse des méthodes dominantes comme l'indique le schéma suivant :



UML est devenu aujourd'hui un standard incontournable. C'est le résultat d'un large consensus d'industriels, méthodologistes, ... et le fruit d'un travail d'experts reconnus. Il a l'avantage d'être issu du terrain ce qui lui apporte une richesse et une efficacité redoutables. UML est ouvert ; il est indépendant du domaine d'application et des langages d'implémentation.

UML évolue mais reste stable. De nombreux acteurs industriels centralisent et normalisent les évolutions d'UML au niveau international. Il inclut des mécanismes standard d'auto extension et grâce à la description de son méta modèle, UML gagne en maturité et en précision en restant stable.

UML n'est pas une méthode ou un processus. Si l'on parle de méthode objet pour UML, c'est par abus de langage. Une méthode propose aussi un processus, qui régit notamment l'enchaînement des activités de production d'une entreprise. Mais UML a été pensé pour permettre de modéliser les activités de l'entreprise, pas pour les régir. Un processus de développement logiciel universel est une utopie. Il est impossible de prendre en compte toutes les organisations et cultures d'entreprises. Un processus est plutôt adapté (donc très lié) au domaine d'activité de l'entreprise. Et même si un processus constitue un cadre général, il faut l'adapter de manière précise au contexte de l'entreprise.

UML cadre l'analyse objet en offrant différentes vues (perspectives) complémentaires d'un système, qui guident l'utilisation des concepts objets. A côté de ça UML offre plusieurs niveaux d'abstraction, qui permettent de mieux contrôler la complexité dans l'expression des solutions objets.

UML est aussi un support de communication. Sa notation graphique permet d'exprimer visuellement une solution objet. Son aspect formel limite les ambiguïtés et les incompréhensions et facilite la comparaison et l'évaluation de solutions. En plus, son indépendance (par rapport aux langages d'implémentation, domaine d'application, processus...) en font un langage universel.

UML comporte aussi des points faibles. Par exemple, la mise en pratique d'UML nécessite un apprentissage et passe par une période d'adaptation. Aussi, le processus (non

couvert par UML) est une autre clé de la réussite d'un projet. Or, l'intégration d'UML dans un processus n'est pas triviale et améliorer un processus est une tâche complexe et longue. Les auteurs d'UML sont tout à fait conscients de l'importance du processus, mais l'acceptabilité industrielle de la modélisation objet passe d'abord par la disponibilité d'un langage d'analyse objet performant et standard.

Les vues d'UML

Vues statiques

Cas d'utilisation

La conceptualisation a pour but de comprendre et structurer les besoins du client. Il ne faut pas chercher l'exhaustivité, mais clarifier, filtrer et organiser les besoins. Une fois identifiés et structurés, ces besoins définissent le contour du système à modéliser (ils précisent le but à atteindre) et permettent d'identifier les fonctionnalités principales (critiques) du système. Le modèle conceptuel doit servir d'interface entre tous les acteurs du projet.

Les cas d'utilisation sont la solution pour représenter le modèle conceptuel. Ils permettent de structurer les besoins des utilisateurs et les objectifs correspondants d'un système. Loin de présenter des solutions d'implémentation, les cas d'utilisation se limitent aux préoccupations réelles des utilisateurs. Leur rôle principal consiste à identifier les utilisateurs du système (acteurs) et leur interaction avec le système.

Eléments de base des cas d'utilisation :

Acteur : entité externe qui agit sur le système (opérateur, autre système...).

- L'acteur peut consulter ou modifier l'état du système.
- En réponse à l'action d'un acteur, le système fournit un service qui correspond à son besoin.
- Les acteurs peuvent être classés (hiérarchisés).

Use case : ensemble d'actions réalisées par le système, en réponse à une action d'un acteur.

- Les uses cases peuvent être structurés.
- Les uses cases peuvent être organisés en paquetages (packages).
- L'ensemble des use cases décrit les objectifs (le but) du système.

Exemple de cas d'utilisation :

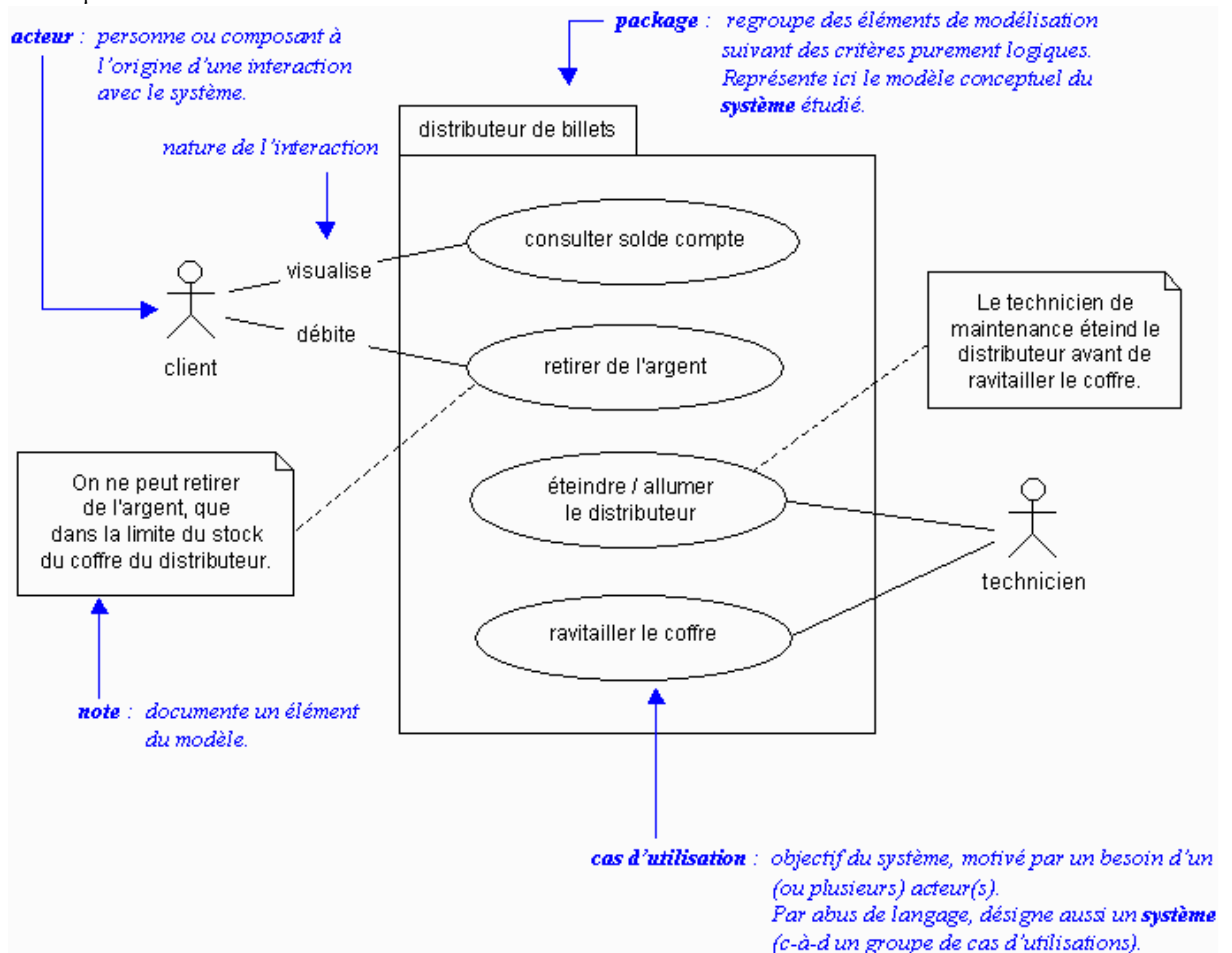


Diagramme d'objets

Ce type de diagramme UML montre des objets (instances de classes dans un état particulier) et des liens (relations sémantiques) entre ces objets. Il s'utilise pour montrer un contexte (avant ou après une interaction entre objets par exemple). Généralement ce type de diagrammes sert essentiellement en phase exploratoire, car il possède un très haut niveau d'abstraction.

Exemple :

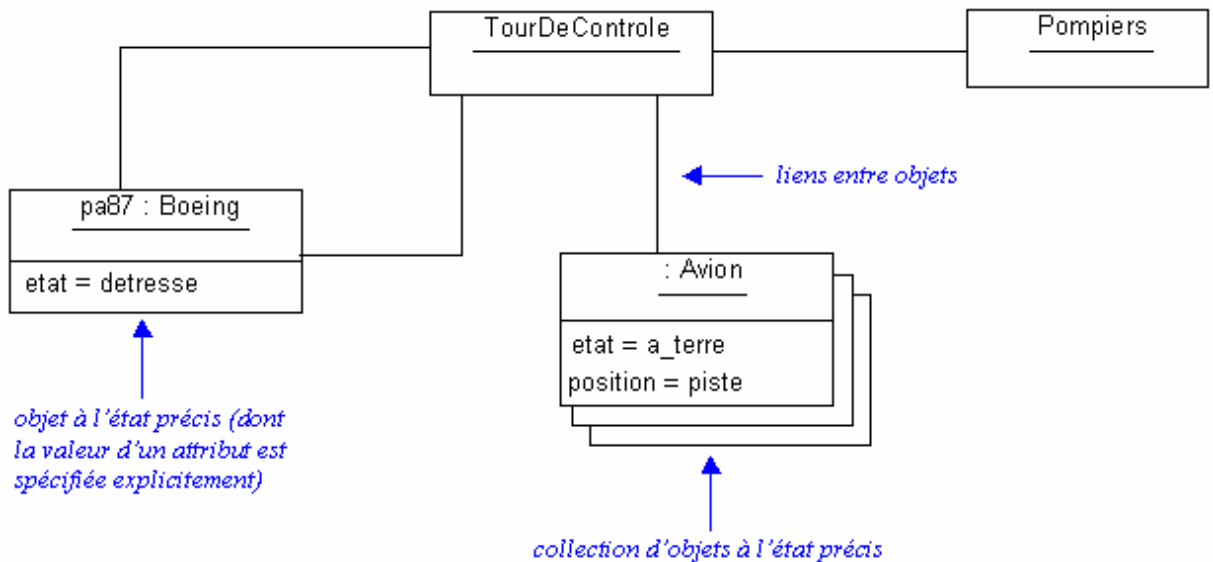


Diagramme de classes

Un diagramme de classes est une collection d'éléments de modélisation statiques (classes, paquetages...), qui montre la structure d'un modèle en faisant abstraction des aspects dynamiques et temporels. Pour un modèle complexe, plusieurs diagrammes de classes complémentaires doivent être construits. On peut par exemple se focaliser sur :

- o les classes qui participent à un cas d'utilisation,
- o les classes associées dans la réalisation d'un scénario précis,
- o les classes qui composent un paquetage,
- o la structure hiérarchique d'un ensemble de classes.

Pour représenter un contexte précis, un diagramme de classes peut être instancié en diagrammes d'objets.

Exemple :

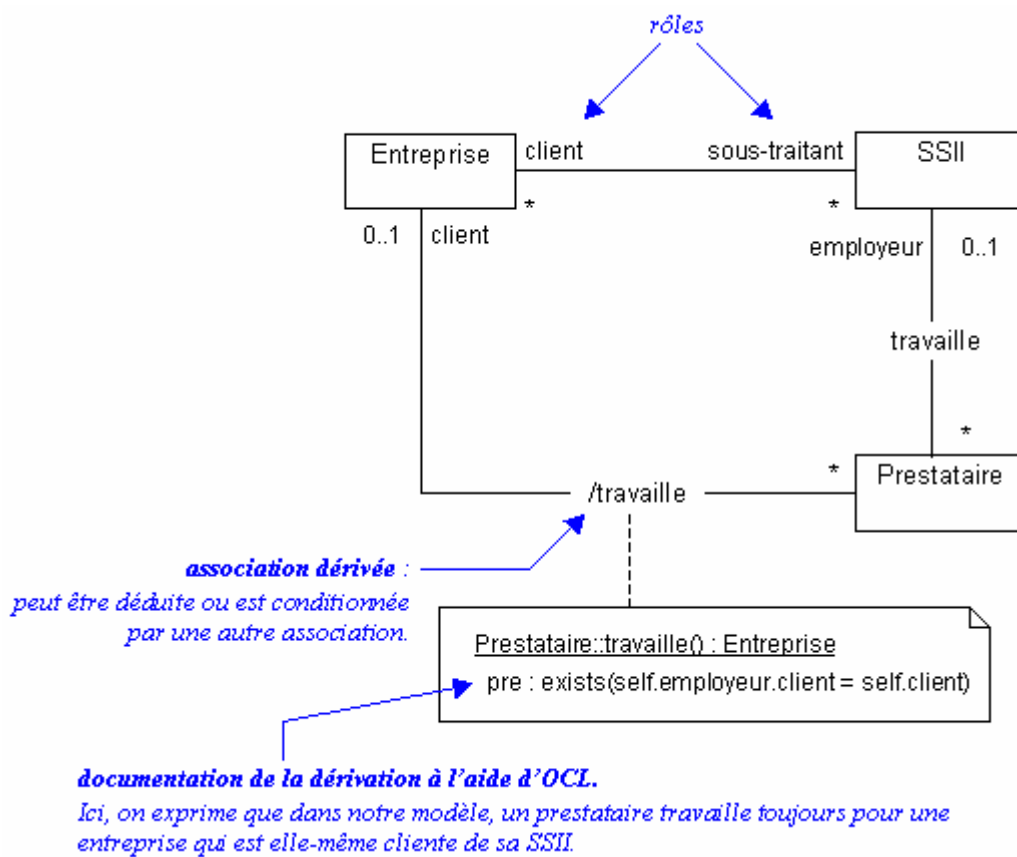
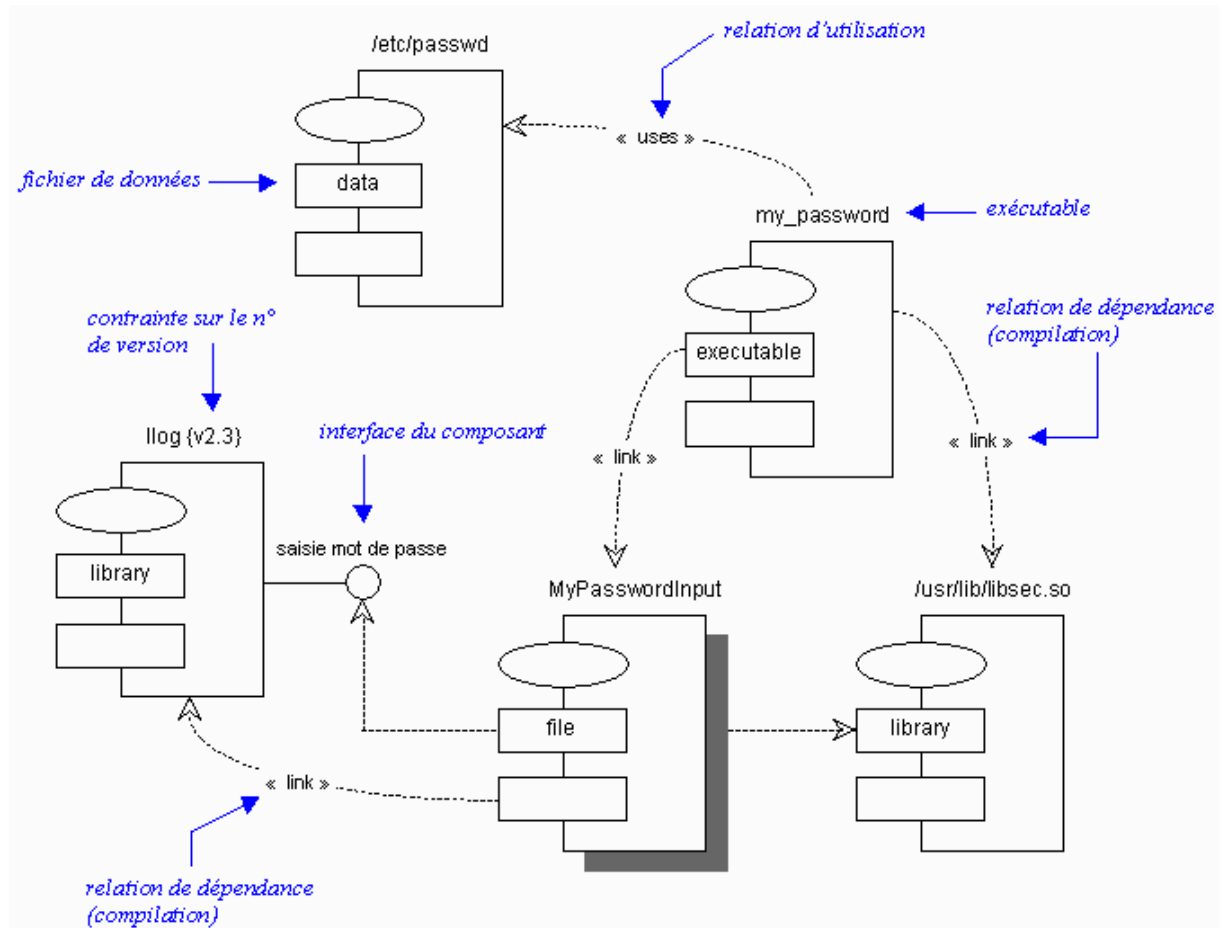


Diagramme de composants

Les diagrammes de composants permettent de décrire l'architecture physique et statique d'une application en terme de modules : fichiers sources, bibliothèques, exécutables, etc. Ils montrent la mise en oeuvre physique des modèles de la vue logique avec l'environnement de développement. Les dépendances entre composants permettent notamment d'identifier les contraintes de compilation et de mettre en évidence la réutilisation des composants. Ces derniers peuvent être organisés en paquetages, qui définissent des sous-systèmes. Les sous-systèmes organisent la vue des composants (de réalisation) d'un système. Ils permettent de gérer la complexité, par encapsulation des détails d'implémentation.

Exemple :

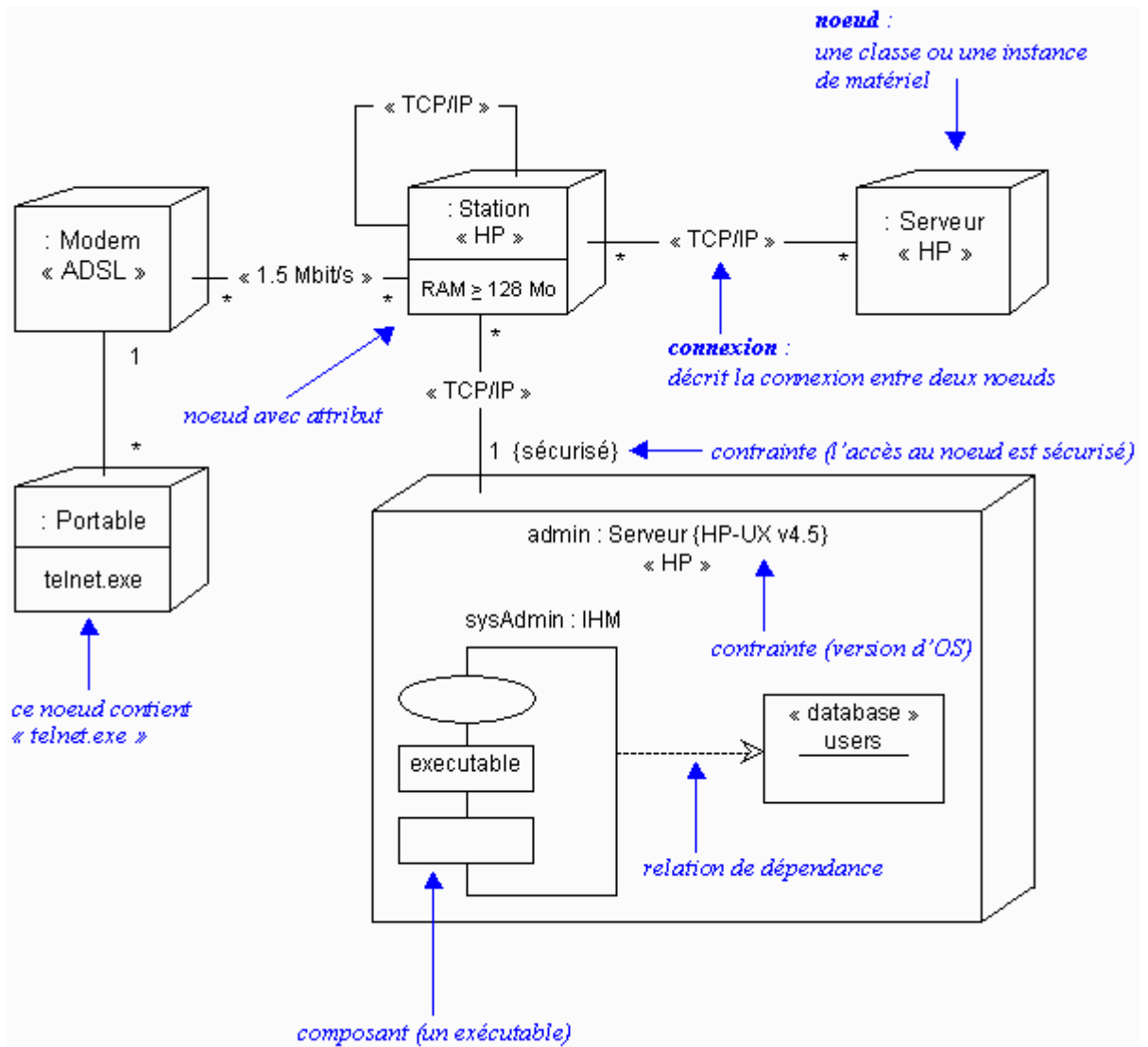


Exemple :

Diagramme de déploiement

Les diagrammes de déploiement montrent la disposition physique du matériel qui compose le système et la répartition des composants sur ce matériel. Les ressources matérielles sont représentées sous forme de nœuds. Ces nœuds sont connectés entre eux, à l'aide d'un support de communication. La nature des lignes de communication et leurs caractéristiques peuvent être précisées. Ces diagrammes peuvent montrer des instances de nœuds (un matériel précis), ou des classes de nœuds.

Exemple :



Vues dynamiques

Diagramme de collaboration

Les collaborations sont des interactions entre objets, dont le but est de réaliser un objectif du système (c'est-à-dire aussi de répondre à un besoin d'un utilisateur). Ils permettent de représenter le contexte d'une interaction, car on peut y préciser les états des objets qui interagissent.

L'élément de modélisation UML "collaboration", représente les classes qui participent à la réalisation d'un cas d'utilisation.

Exemple :

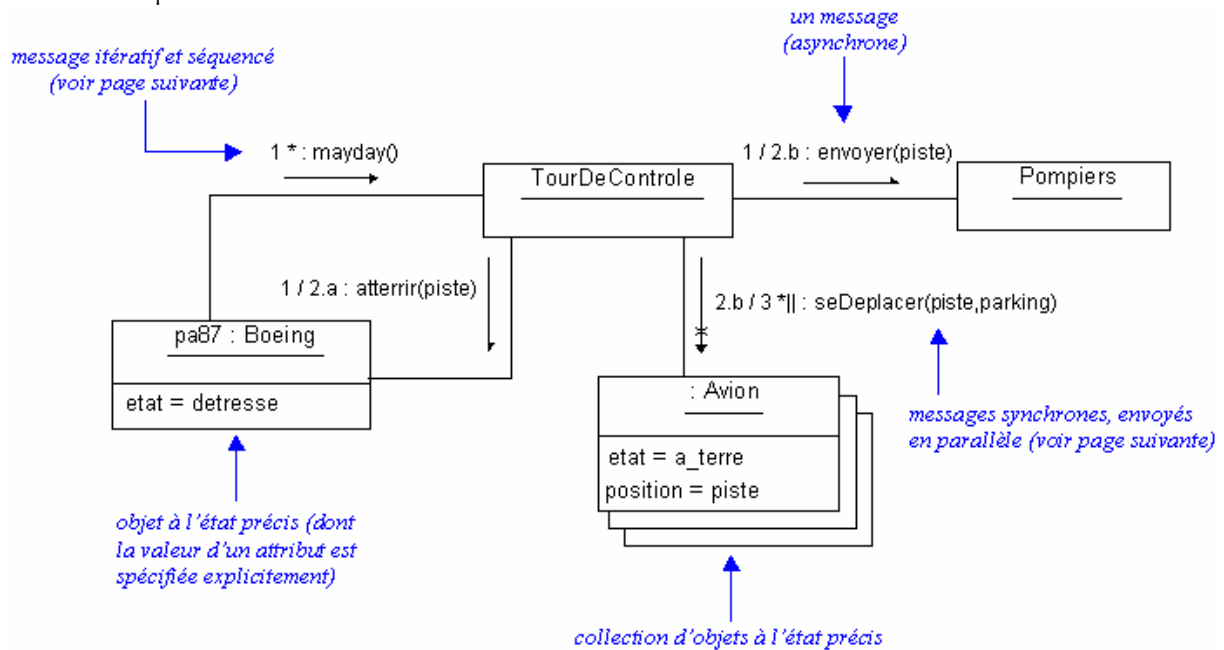


Diagramme de Séquences

Les diagrammes de séquences permettent de représenter des collaborations entre objets selon un point de vue temporel, on y met l'accent sur la chronologie des envois de messages. Contrairement au diagramme de collaboration, on n'y décrit pas le contexte ou l'état des objets, la représentation se concentre sur l'expression des interactions. Les diagrammes de séquences peuvent servir à illustrer un cas d'utilisation.

L'ordre d'envoi d'un message est déterminé par sa position sur l'axe vertical du diagramme ; le temps s'écoule "de haut en bas" de cet axe. La disposition des objets sur l'axe horizontal n'a pas de conséquence pour la sémantique du diagramme.

Il serait intéressant de noter que les diagrammes de séquences et les diagrammes d'état transitions sont les vues dynamiques les plus importantes d'UML.

Exemple :

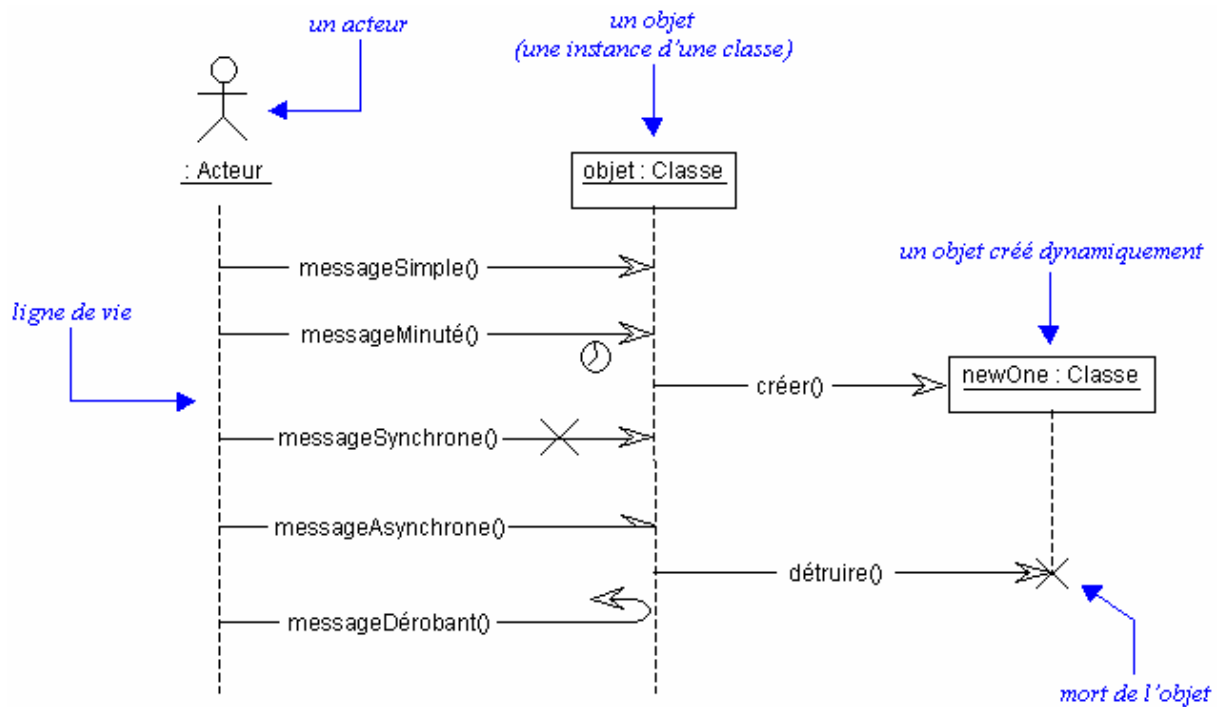


Diagramme d'états transitions

Ce diagramme sert à représenter des automates d'états finis, sous forme de graphes d'états, reliés par des arcs orientés qui décrivent les transitions. Ce qui permet de décrire les changements d'états d'un objet ou d'un composant, en réponse aux interactions avec d'autres objets/composants ou avec des acteurs.

Un état se caractérise par sa durée et sa stabilité, il représente une conjonction instantanée des valeurs des attributs d'un objet. Alors qu'une transition représente le passage instantané d'un état vers un autre. Elle est déclenchée par un événement. En d'autres termes : c'est l'arrivée d'un événement qui conditionne la transition.

En plus de spécifier un événement précis, il est aussi possible de conditionner une transition, à l'aide de "gardes" : il s'agit d'expressions booléennes, exprimées en langage naturel (et encadrées de crochets).

Exemple :

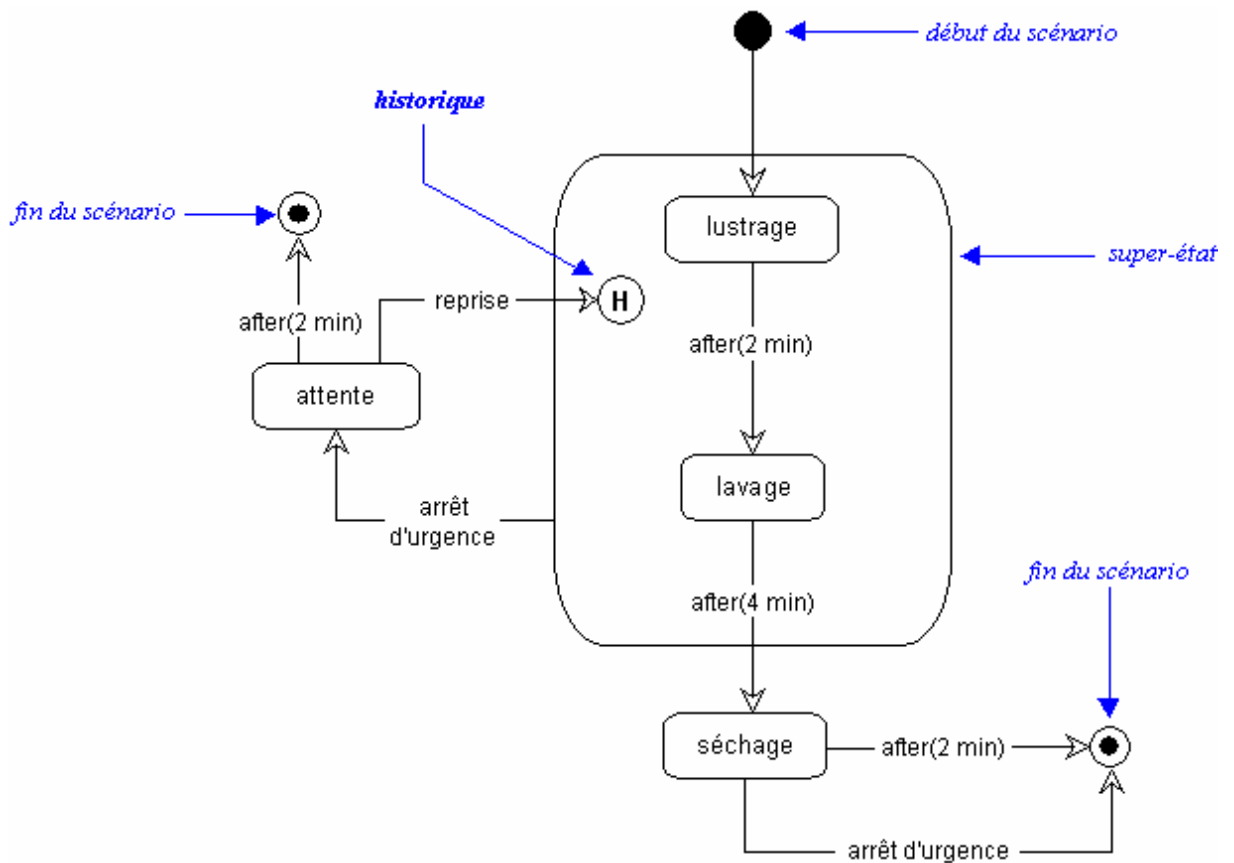
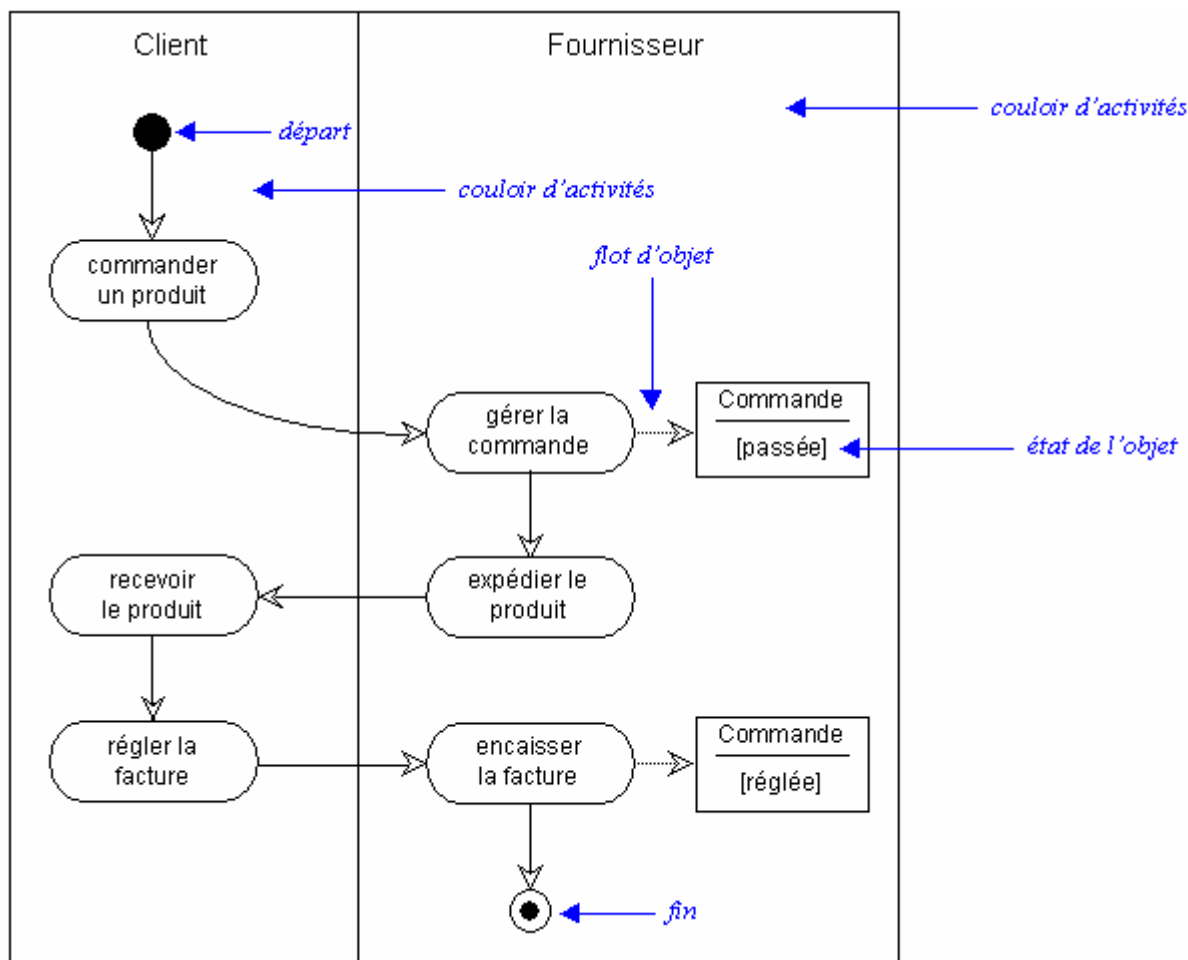


Diagramme d'activités

UML permet de représenter graphiquement le comportement d'une méthode ou le déroulement d'un cas d'utilisation, à l'aide de diagrammes d'activités (une variante des diagrammes d'états-transitions). Une activité représente une exécution d'un mécanisme, un déroulement d'étapes séquentielles. Le passage d'une activité vers une autre est matérialisé par une transition. Ces transitions sont déclenchées par la fin d'une activité et provoquent le début immédiat d'une autre (elles sont automatiques).

En théorie, tous les mécanismes dynamiques pourraient être décrits par un diagramme d'activités, mais seuls les mécanismes complexes ou intéressants méritent d'être représentés.

Exemple :



Les outils UML

Après la standardisation d'UML les outils de conception UML se sont vus multipliés. Grâce à sa sémantique simple et claire, l'édition des différentes vues à l'aide d'outils graphiques présente plusieurs avantages qu'on va discuter ultérieurement.

Voici quelques outils :

- Argo/UML (<http://argouml.tigris.org/index.html>) de l'université de Californie UCI (www.ics.uci.edu/pub/arch/uml)
- Prosa/om (www.prosa.fi/prosa.html) d'Insoft Oy
- Objectteering (<http://www.objectteering.com/>) de Softeam (www.softeam.fr)
- Paradigm Plus (www.platinum.com/products/appdev/pplus_ps.htm) de Computer Associates
- Rhapsody (www.ilogix.com/fs_prod.htm) d'I-Logix (www.ilogix.com)
- Rose 2000 (<http://www.rosearchitect.com/>) de Rational Software Corporation (www.rational.com)
- StP/UML (www.aonix.com/Products/SMS/core7.1.html) d'Aonix (www.aonix.fr)
- Visual UML (www.visualuml.com/products.htm) de Visual Object Modelers
- 4Keeps d'A. D. Experts (www.adexperts.com)

- Bold for Delphi (www.boldsoft.com/products) de BoldSoft
- COOL:Jex (www.cool.sterling.com/products/Jex/index.htm) de Sterling Software
- Elixir CASE (www.elixirtech.com/ElixirCASE/index.html) d'Elixir Tech.
- Ensemble Suite (www.ensemble-systems.com/products.html) d'Ensemble Systems
- Florist (www.qoses.com/products) de Qoses
- Framework de Ptech (www.ptechinc.com)
- Framework Studio (www.blueprint-technologies.com/products/index.html) de Blueprint Tech.
- GDPro d'Advanced Software Tech. (www.advancedsw.com)
- How de Riverton Software (www.riverton.com)
- Innovator (www.mid.de/e/innovato/index.htm) de Mid
- ISOA de Mega International (www.mega.com)
- JVision d'Object Insight (www.objectinsight.com)
- MagicDraw UML de No Magic (www.nomagic.com)
- Mesa/Vista Enterprise (www.mesasy.com/vista/vista_family.html) de Mesa
- MetaEdit+ de Metacase (www.metacase.com)
- Model Prototyper d'Objexion (www.objexion.com)
- Object Domain d'Object Domain Systems (www.objectdomain.com)
- ObjectGeode (www.verilogusa.com/products/geode.htm) de Verilog
- ObjectiF (www.microtool.de/obje.e) de MicroTOOL
- Opentool de TNI (www.tni.fr)
- ORCA (www.telelogic.com/solution/tools/orca.asp) de Telelogic
- Pragmatica de Pragmatix Software (www.pragsoft.com)
- Real-Time Studio Prof. d'Artisan (www.artisansw.com)
- Rocase (www.cs.ubbcluj.ro/rocase) de l'université de Roumanie Babes-Bolyai
- Rose RealTime de Rational Software Corporation (www.rational.com)
- RoZeLink d'Headway Software (www.headway-software.com)
- Select/Enterprise (www.princetonsofttech.com/products) de Princeton Softech
- Simply Objects (www.adaptive-arts.com/products.htm) d'Adaptive
- SoftModeler/Business (www.softera.com/products.htm) de Softera
- Structure Builder de Tendril Software (www.tendril.com/)
- System Architect (www.popkin.com/products/sa2001/product.htm) de Popkin
- Together/Enterprise (www.togethersoft.com) d'Object International
- UML Designer (www.software.ibm.com/ad/smalltalk/about/umldfact.html) d'IBM
- Visual Modeler (msdn.microsoft.com/vstudio/prodinfo/new.asp) de Microsoft
- Visual Thought de Confluent (www.confluent.com)

- Win A&D d'Excel Software (www.excelsoftware.com)
- WithClass de Microgold Software (www.microgold.com)
- Xtend:Specs (www.iconcomp.com/products/index.html) d'ICON

Computing

Et il y'en a encore plusieurs. Cette liste vous donne une idée sur le nombre important des outils disponibles. Un choix parmi cette panoplie n'est sûrement pas évident quoiqu'on peut classer ces outils en deux catégories : les outils libres et le outils payant. En essayant de porter notre attention sur les outils libres, Argo/UML s'est avéré un outil assez intéressant, populaire et très prometteur. Mais vu son abondant par les développeurs (ou presque). On a préféré nous pencher vers Poseidon for UML qui est une amélioration de Argo/UML, effectuée par les mêmes développeurs et qui comporte certains avantages qu'on discutera en temps utile.

Poseidon for UML

Introduction

Poseidon est directement basé sur Argo/UML mais beaucoup plus développé. Il fournit davantage d'options et connaît plus de stabilité. Argo/UML, par contre, est open source et dédié à la recherche, l'étude des architectures et l'extensibilité.

Poseidon for UML est délivré en différentes éditions. Toutes les éditions sont basées sur la même source et portent ainsi la même version. De nouvelles versions apparaissent deux fois par an. Ces éditions offrent différents outils et fournissent différents niveaux de support.

Community edition :

C'est la version de base. Distribuée librement, c'est la première entrée dans le monde d'UML pour les développeurs individuels. Toutefois, son utilisation doit être non commerciale. Elle contient tous les diagrammes d'UML et tous les éléments de diagrammes implémentés. Vous pouvez créer, sauvegarder, charger des projets, explorer les modèles existants, échanger des modèles, générer du code java, exporter vos diagrammes vers différents formats ...

La community edition fournit tous les éléments nécessaires pour apprendre UML à un niveau non professionnel. Par contre, il existe certaines restrictions. Certains des outils qui sont disponibles dans les versions commerciales ne sont pas inclus dans la version libre. Ces outils sont intéressants dans la perspective d'augmenter la productivité, mais pas nécessaires pour créer des modèles UML.

La community edition contient les avantages suivants :

- Totalemment écrits en java, donc indépendants de la plateforme.
- Support des 9 diagrammes UML.
- Respect du standard UML2.
- XMI 1.2 est supporté comme un format de sauvegarde standard. XMI 1.0, 1.1 et 1.2 peuvent être chargés.
- Les diagrammes peuvent être exportés vers gif, ps, eps et svg.
- Support des formats jpeg et png.
- Copier, couper, coller.
- Certains diagrammes supportent le glisser déplacer.
- Localisation en anglais, français, allemand, espagnol, et chinois simplifié.
- Forward engineering for java.
- Installation et mises à jours simples avec Java Web Start.
- Support de annuler/refaire.

Standard edition

C'est l'outil extensible de base pour le professionnel. Il est fourni avec tous les avantages de la community edition en plus de d'outils de productivité comme l'impression, le glisser déplacer vers le système Windows (copier des diagrammes vers Word ou Powerpoint) et un zoom complet. Grâce à un mécanisme de plug-ins, vous

pouvez choisir depuis un ensemble en constante évolution de plug-ins qui vous permet des extensions futures à ses fonctionnalités. Un support e-mail est aussi fournit avec cette édition.

UMLdoc, le générateur de documents HTML, vous permet d'exporter vos modèles vers un format HTML. Il est similaire à Javadoc, mais inclut toutes les informations d'un modèle UML avec ses diagrammes ; c'est pour cette raison qu'il a été appelé UMLdoc.

La standard edition a les avantages suivants sur la community edition :

- Forward et Reverse engeneering pour Java.
- Le mécanisme des plug-ins pour charger et libérer des plug-ins depuis les partenaires technologiques de Gentleware, même en runtime.
- Impression confortable avec mise en page.
- Copie direct vers le clipboard Windows.
- Génération d'une documentation HTML avec UMLdoc
- Support du bureau d'aide Gentleware via des e-mails.

Professional edition

C'est la principale version de Poseidon for UML. Pour répondre aux besoins du développeur professionnel, Gentleware a construit un mécanisme de génération de code très intéressant avec un ensemble d'outils de productivité. Cette édition contient le round-trip engineering, importation JAR et la génération de documents HTML.

L'une des fonctionnalités les plus intéressantes de Poseidon for UML est la possibilité de générer du code. L'édition professionnel vous donne un accès total à cet avantage. Le mécanisme de génération de code est basé sur la template technology, où la template définit la syntaxe du code généré. Ce code peut être Java, C++, XML, HTML ou tous ce que vous voulez. Les informations du modèle, comme les noms des classes et des méthodes sont fournit par Poseidon for UML. Cette édition vous donne l'accès à l'API et aux templates. En tant que développeur vous pouvez éditer et changer ces templates, même en cours d'exécution, et configurer ainsi la forme du code généré vous-même.

Le round-trip engineering sophistiqué pour Java vous permet de lire du code Java existant et en générer le modèle correspondant ou de synchroniser continuellement votre code avec le modèle. Vous pouvez changer le code générer ou modifier le modèle sans jamais perdre la cohérence entre les deux. Avec la fonctionnalité d'importer des fichiers JAR, vous pouvez lire des librairies existantes et les utiliser dans vos modèles.

La Professional Edition contient les avantages suivants par rapport à la Standard Edition :

- Génération de code basée sur les Templates avec un accès total.
- Round-trip engineering pour Java.
- Importation JAR pour inclure des librairies existantes.
- Importation des fichiers Rational Rose (.mdl).

Enterprise Edition

Le travail en équipe est supporté par cette édition. Se caractérise par le contrôle de version, support Multi-Utilisateur, architecture client serveur, et beaucoup d'autres

avantages qui pourraient être nécessaires pour l'ingénierie orientée modèle dans un environnement de développement hautement collaboratif.

L'Enterprise Edition est fournit en deux composantes, serveur et client. Tous les deux se caractérisent par une installation facile. Les transmissions entre client et serveur sont sécurisées par ssh, méthode standard de cryptage et d'authentification. Afin de supporter l'aspect communication dans une modélisation collaborative, le client intègre aussi une messagerie instantanée basée sur le protocole Jabber.

L'Entreprise Edition inclut les additions suivantes par rapport à la Professional Edition :

- Un environnement de modélisation collaborative basée sur une architecture client-serveur.
- Blocage exclusif de parties du modèle et gestion des conflits.
- Manipulation du projet par le serveur.
- Transmission sécurisée entre client et serveur avec SSH.
- Installation facile pour le serveur et le client.

Embedded Enterprise Edition

Elle a été conçu spécialement pour le développement des systèmes intégrés. Afin de répondre aux besoins des ingénieurs des systèmes intégrés, cette version contient la plupart des particularités de la Professional Edition avec une optimisation du code généré pour ANSI C et C++ ainsi que la capacité de travail en équipe de l'Enterprise Edition. Le générateur de code a été spécialement conçu afin de répondre aux critères dictés pour les systèmes intégrés, comme les contraintes des ressources mémoire et performances. Il supporte la génération automatique du code des diagrammes d'état d'UML aussi bien que les diagrammes de classes.

Tableau comparatif

Le tableau qui suit donne une comparaison claire entre les différentes éditions :

Caractéristique	CE	SE	PE	EmbEE	EE
	Communi ty Edition	Standard Edition	Professio nal Edition	Embedded Enterprise Edition	Enterpris e Edition
Installation simple avec WebStart	✓				
Support UML 2.0	✓	✓	✓	✓	✓
Les 9 diagrammes	✓	✓	✓	✓	✓
Forward Engineering Java	✓	✓	✓	✓	✓
XMI Supporté	✓	✓	✓	✓	✓
Indépendant de la plateforme	✓	✓	✓	✓	✓

Export vers GIF, JPG, PNG, PS, EPS, SVG, WMF	✓	✓	✓	✓	✓
Copier/Couper/Coler	✓	✓	✓	✓	✓
Support OCL	✓	✓	✓	✓	✓
Impression	✓	✓	✓	✓	✓
Annuler/Refaire	✓	✓	✓	✓	✓
UMLdoc (Export HTML)	*	✓	✓	✓	✓
Reverse Engineering Java		✓	✓	✓	✓
Support des Plug-Ins		✓	✓	✓	✓
Import JAR			✓	✓	✓
Import MDL			✓	✓	✓
Templates de codes modifiables			✓	✓	✓
Génération C++			✓	✓	✓
Intégration à l'IDE Eclipse			✓		
Génération ANSIC				✓	
Génération C#			✓		✓
Génération CORBA IDL			✓		✓
Génération Delphi			✓		✓
Génération Perl			✓		✓
Génération PHP			✓		✓
Génération SQL DDL			✓		✓
Génération VB.net			✓		✓
Round Trip Engineering Java			✓		✓

Plusieurs licences simultanées			✓	✓	✓
Modélisation en équipe synchronisée				✓	✓
Contrôle de version				✓	✓
Blocage des éléments				✓	✓
Communication sécurisée				✓	✓
GoVisual Autolayout Plug-In par Oreas		(✓)	(✓)+		
yWorks Autolayout Plug-In par yWorks		(✓)	(✓)+	(✓)+	(✓)+
AndroMDA Plug-In par M. Bohlen		(✓)	(✓)+	(✓)	(✓)
Requirements Engineering Plug-In par oose.de		(✓)	(✓)	(✓)+	(✓)+
b+m EJB Plug-In par b+m Informatik AG		(✓)	(✓)	(✓)+	(✓)+

Notations:

✓ Inclut à cette édition

(✓) Non inclus à cette édition, peut être acheté séparément.

* Disponible de façon externe à <http://www.uml4oc.org>

+ Licence disponible.

Installation et pré-requis

Poseidon for UML a été entièrement écrit en Java ce qui le rend indépendant de la plateforme. Il marche à peu près sur n'importe quel PC moderne. Afin de démarrer Poseidon for UML avec succès vous aurez besoin de ce qui suit :

- Java Runtime Environment ou Java Development Kit. JDK 1.4 ou supérieur est requis pour Linux, Mac OS X et les plateformes Windows. Poseidon for UML ne peut s'exécuter avec JDK 1.3 ou autre version antérieure.

- Un ordinateur avec mémoire et CPU raisonnables. Pour la mémoire, 512 MB est recommandée. Pour le CPU, un Pentium II ou équivalent est un minimum recommandé.
- Un système d'exploitation spécifique n'est pas requis. Poseidon for UML a été testé sur Windows 98, 2000, NT et XP, sur Linux SuSe 6.x, 7.x, RedHat, et MacOS X. Il a été développé et testé sur Linux. Toutefois, sur les plateformes Windows, il connaît des performances supérieures dues à un environnement Java plus rapide.

Le moyen le plus facile d'installer Poseidon for UML est d'utiliser InstallAnywhere pour votre plateforme. Si vous avez déjà une version Java récente installée, vous pouvez télécharger l'installateur pour votre plateforme qui n'inclut pas Java. Si vous n'avez pas Java installé ou si vous n'êtes pas sûr de la version, téléchargez l'installateur qui inclut Java.

On va vous demander de spécifier le chemin d'installation et l'emplacement des raccourcis. Un espace disque libre de 20 MO est requis.

Ceci est valable pour toutes les éditions sauf pour l'Entreprise Edition et l'Embedded Enterprise Edition où l'on doit installer au moins un serveur et un client.

Les variables d'environnement JAVA_HOME et POSEIDONxx_HOME, où xx représente la version de Poseidon for UML, peuvent être modifiées afin de spécifier l'emplacement du JRE et l'emplacement des fichiers personnels de configuration et de log respectivement.

Il faudrait noter que l'emplacement du JRE doit contenir un SDK complet de Java afin de permettre à Poseidon for UML de générer ou compiler du code.

Pour utiliser Poseidon for UML vous avez besoin d'une clé de licence valide. C'est une chaîne de caractères qui contient des informations cryptées. Pour obtenir cette clé il suffit de suivre un simple processus de enregistrement. Une fois que vous avez la clé il suffit de la copier vers l'application.

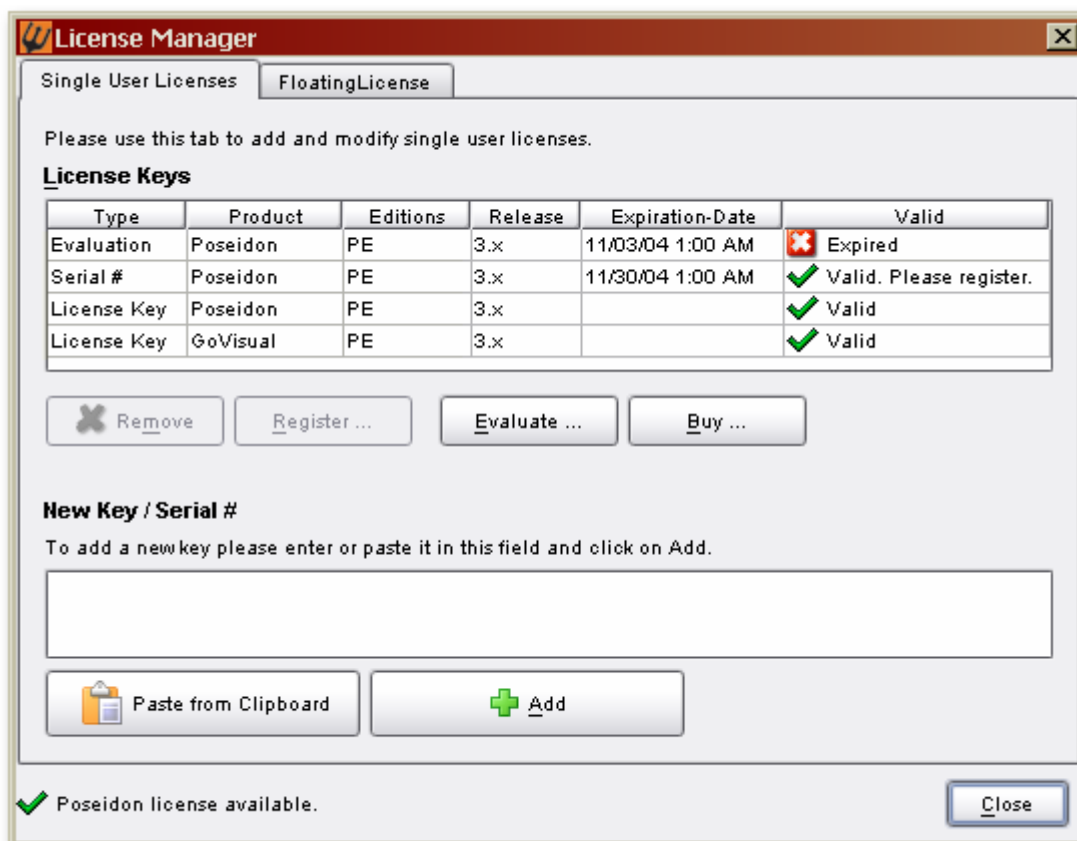
Il y'a plusieurs types de clés :

Evaluation Key : Fournie avec une copie d'évaluation d'une édition commerciale. Cette clé place une limite de temps et de fonctionnalités à l'usage de l'application.

Serial Number : Fourni avec la Community Edition et les copies achetées des éditions commerciales. Le numéro de série est un identificateur unique utilisé pour enregistrer l'utilisateur avec la copie spécifique de l'application afin de recevoir la Licence Key.

Licence Key : Fourni avec la Community Edition et les copies achetées des éditions commerciales. Ces clés deviennent valides après que les données de l'enregistrement aient été reçues par Gentleware. Une fois cette clé mise en place, le processus d'enregistrement est terminé.

Voici un aperçu de la fenêtre de gestion des licences :



L'interface

L'espace de travail de Poseidon est composé de 5 parties. En haut de la fenêtre, il y'a le menu principal et une barre qui fournit un accès aux fonctions principales. En dessous, il y'a 4 cadres :

Cadre de navigation :

On commence par explorer ce cadre qui est situé en haut à gauche. Il est utilisé pour accéder à toutes les parties principales d'un modèle en présentant les éléments du modèle sur différentes structures arborescentes. Par exemple, l'arbre peut être trié par noms des éléments en ordre alphabétique, par type d'élément ou par nom de diagramme. Le moyen classique de les organiser est par packages. Poseidon for UML utilise cette structure en packages comme arbre de navigation par défaut, comme le font la plupart des outils UML. Ces structure arborescentes sont appelées vues. Ceci présente un point fort de Poseidon for UML, permettant une énorme flexibilité pour la navigation.

Cadre de diagrammes :

Comme les diagrammes sont le centre d'intérêt d'UML, le cadre de diagrammes est par conséquent l'espace de travail principal. C'est l'emplacement principal pour construire et éditer des diagrammes qui composeront tous vos modèles. Comme le cadre de navigation peut afficher plusieurs vues, le cadre de navigation utilise des onglets pour ouvrir des espaces de travail additionnels.

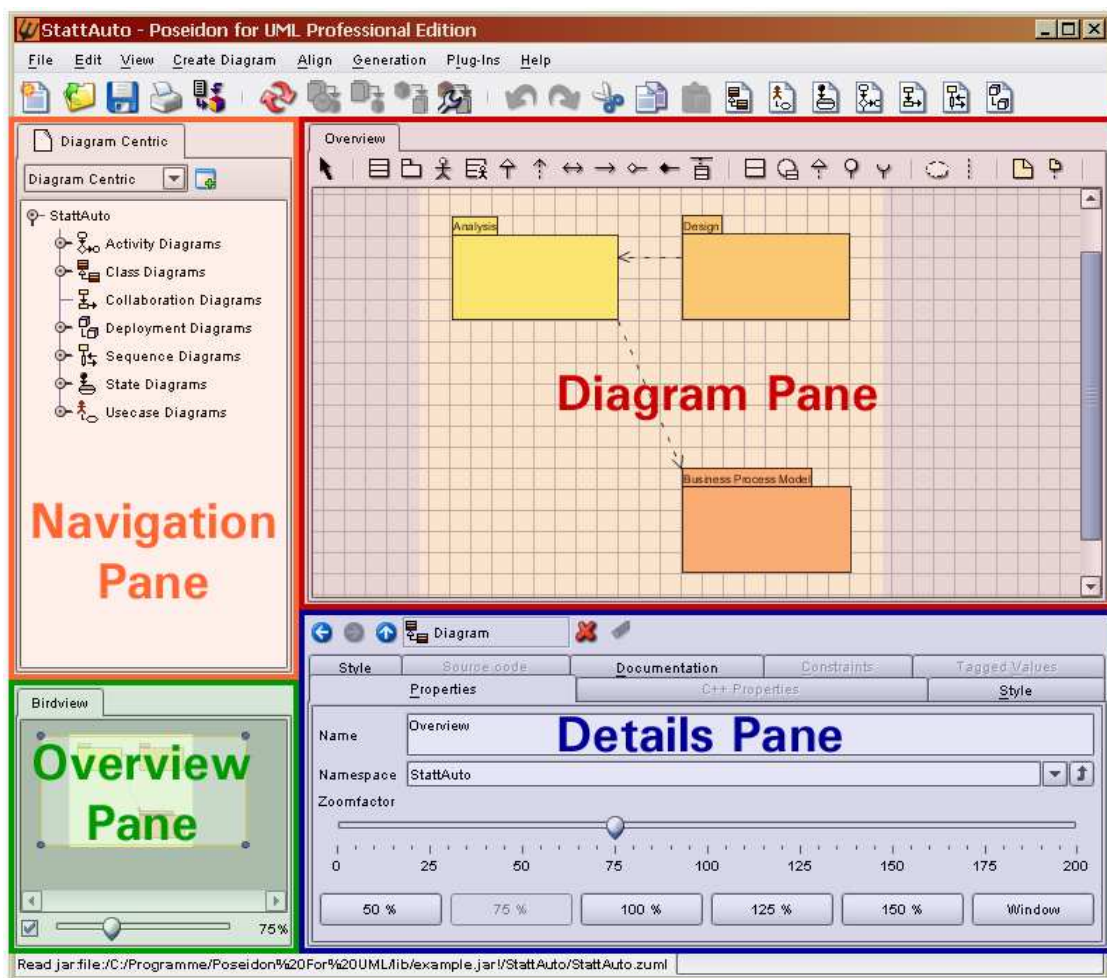
Cadre des détails :

Plusieurs choses interviennent dans un modèle hormis les figures représentant les éléments et les connexions entre eux. Si toutes ces informations étaient affichées dans le cadre des diagrammes, les modèles seraient rapidement illisibles. Le cadre des détails organise et présente toutes ces informations via des onglets.

Le cadre de vision d'ensemble :

Plus le diagramme est grand, plus il est difficile de garder une vue sur tous ces éléments, surtout s'ils sont hors de la zone de vision immédiate. Ce cadre, situé en bas à gauche, fournit une vue d'ensemble des éléments, ce qui représente un sommaire graphique du diagramme en cours d'affichage dans le cadre des diagrammes.

Aperçu de l'espace de travail de Poseidon for UML:



Fonctionnalités avancées

Génération du code

Poseidon for UML fournit un framework très puissant et flexible pour la génération du code, basé sur un mécanisme de templates. Il est utilisé pour le moment pour générer une variété de types de code comme HTML et Java, mais il est assez flexible pour générer n'importe quel type de langages de programmation, ou autres sorties, comme XML.

La génération du code Java est généralement basée sur les classes du modèles et autres informations contenues dans les diagrammes de classes. Additionnellement, Poseidon for UML peut générer des méthodes d'accès aux attributs pour chaque classe.

Par défaut, les associations entre classes dans UML sont bidirectionnelles ; ce qui fait qu'ils permettent la navigation entre les classes dans les deux sens. Pour la plupart des langages de programmation orientés objet, ceux-ci doivent être transformés en des associations unidirectionnelles séparées. Le code pour gérer les associations bidirectionnelles ainsi que unidirectionnelles est aussi généré automatiquement.

Reverse Engineering

Les ingénieurs en génie logiciel courent toujours au problème de devoir reconcevoir un projet existant qui ne contient que du code source. C'est là où le reverse-engineering entre en jeu ; un outil analyse le code existant à partir duquel il génère automatiquement un modèle et un ensemble de diagrammes de classes.

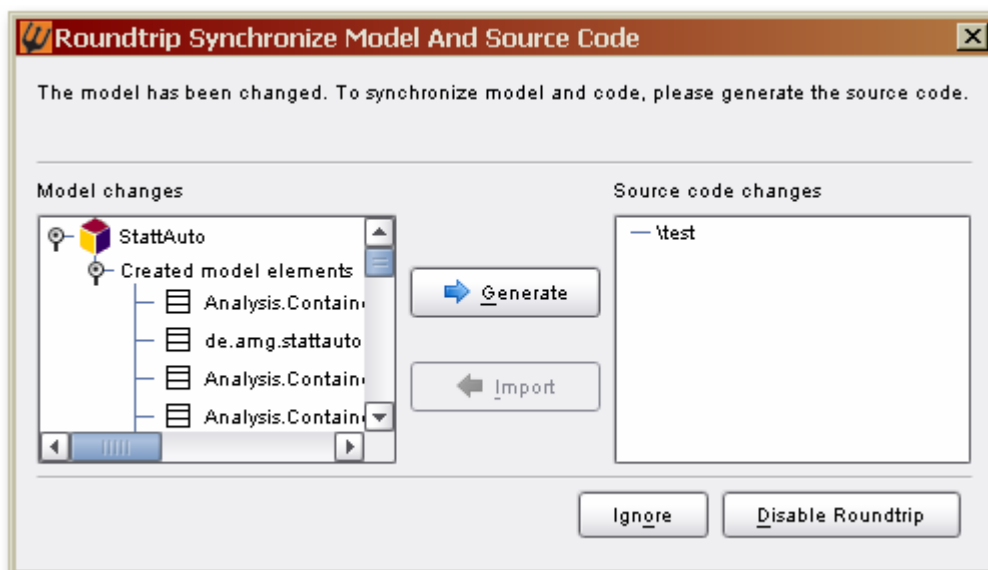
Poseidon for UML peut faire ça pour les programmes Java, là où le code source est disponible dans un état où il peut être compilé sans erreurs. Quant à la fonction d'importation de fichiers JAR (disponible uniquement dans la Professional et Enterprise Edition), il est même possible de recréer le modèle à partir de classes Java compilés.

Round-trip Engineering

Générer du code et exécuter le reverse engineering sur ce même code n'effectue pas le round-trip engineering. Reverse-engineering génère un nouveau modèle à partir du code existant, mais ne le reconnecte pas à un modèle existant. Cette caractéristique est disponible uniquement dans la Professional et Enterprise Edition. C'est l'un des outils les plus sophistiqués de Poseidon for UML.

Vous pouvez générer un modèle UML à partir de votre code existant, changer le modèle, re-générer le code, changer le code et ainsi de suite. Tous les fichiers Java générés sont suivis, pour que les changements effectués avec un éditeur externe soient importés vers le modèle Poseidon de votre projet. Vous pouvez utiliser votre éditeur Java favoris pour éditer les corps des méthodes, ajouter et supprimer des méthodes et des variables membres ; Poseidon garde trace de tous les changements.

Voici la fenêtre de synchronisation entre le code source et le modèle :



Velocity Template Language

La génération du code dans Poseidon for UML est basée sur le Velocity Template Language (VTL). Velocity est un moteur de templates open source développé comme une partie du projet Apache/Jakarta. Originellement conçu pour être utilisé dans les applications Web basées sur les servlets, il a aussi prouvé son utilité dans d'autres domaines d'application incluant la génération du code, le formatage et la transformation du texte.

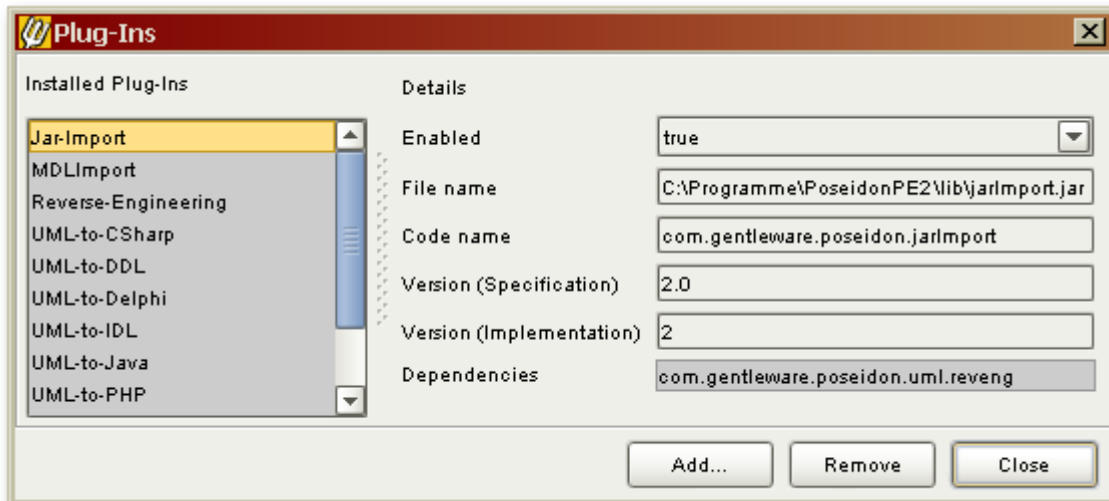
La génération de la documentation

Pour générer la documentation HTML correspondante à votre modèle vous avez besoin de UMLdoc qui est disponible dans toutes les éditions de sauf la Community Edition. Toutefois, les utilisateurs de n'importe quelle édition incluant la Community Edition peuvent utiliser le service UMLdoc Online. Le 'Look and feel' du document généré est très similaire à Javadoc. Poseidon for UML vous permet de spécifier les informations Javadoc directement dans votre modèle (dans l'onglet documentation). Cette information, comme les commentaires des classes ou des méthodes, est incluse dans le code. Mais à l'instar de UMLdoc, Javadoc fournit une vue uniquement sur le code, pas sur le modèle. Par exemple, vous ne pouvez pas voir les diagrammes. Avec UMLdoc, vous avez les mêmes informations que Javadoc en plus de tous les diagrammes de votre modèles.

Les Plug-ins

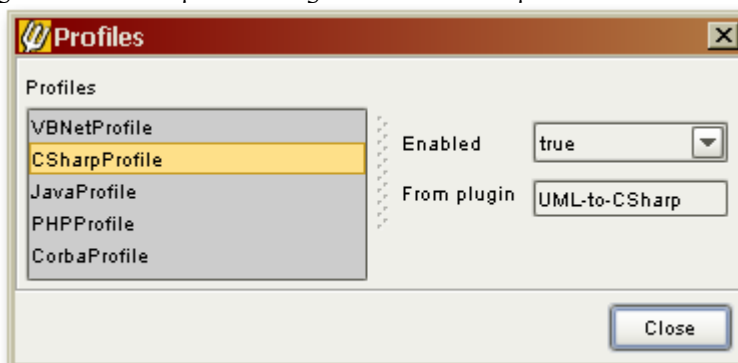
Avec l'interface des plug-ins de Poseidon, il est possible d'ajouter des fonctionnalités supplémentaires ne figurant pas parmi celles implémentées dans le corps du produit. La Standard Edition vient avec ce support de plug-ins. Les équipes de développements de Gentleware ainsi que des partenaires travaillent sur des plug-ins qui englobent les besoins spécifiques des concepteurs et développeurs.

Le gestionnaire de plug-ins fournit une interface facile pour installer, gérer et désinstaller des plug-ins :



Profiles

Les profils étendent UML à travers l'utilisation des stereotypes qui sont la plupart du temps dépendants du langage. Ils fournissent une notation graphique et un vocabulaire spécifiques. Par exemple, les types des variables et des opérations changent dépendamment du profil (et ainsi du stéréotype) utilisé. Il y'a un profil associé à chaque plug-in de langage. Voici un aperçu du gestionnaire de profils :



Intégrer Poseidon à Eclipse

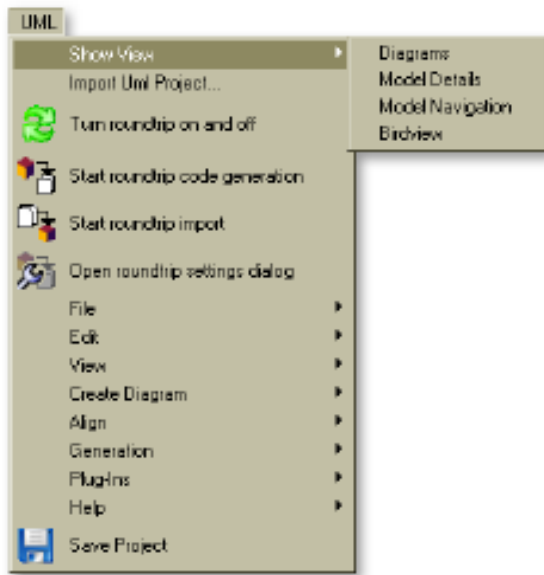
Il faut s'assurer que vous avez Eclipse bien installé et qu'il n'est pas en cours d'exécution.

Dans le menu 'Plug_Ins', sélectionnez 'Eclipse-Bridge'

Et sélectionnez votre Eclipse 'installation directory', puis cliquer 'install'



Le menu UML dans Eclipse



- ◊ **Show View** – ouvrir le cadre de poseidon choisi.
- ◊ Diagrams
- ◊ Model Details
- ◊ Model Navigation
- ◊ Birdview
- ◊ **Import UML Project...** – ouvrir une fenêtre pour ouvrir un projet poseidon dans le projet eclipse en cours.
- ◊ **Turn Roundtrip On and Off** – activer/désactiver roundtrip engineering.
- ◊ **Start Roundtrip Code Generation** – générer le code source à partir du modèle UML.
Quick Key - Ctrl-Alt-G
- ◊ **Start Roundtrip Import** – mise à jour du modèle UML à partir du code source.
Quick Key - Ctrl-Alt-I
- ◊ **Open Roundtrip Settings Dialog** – ouvrir la fenêtre d'options.
- ◊ **Poseidon Menus** – permet un accès à tout les options du menu de Poseidon.
- ◊ File
- ◊ Edit
- ◊ View
- ◊ Create Diagram
- ◊ Align
- ◊ Generation
- ◊ Plug-Ins
- ◊ Help
- ◊ **Save Project** – sauvegarder le projet UML en cours. Il ne sauvegarde pas le projet Eclipse en entier.

La perspective Java

Les perspectives en Eclipse sont une collection d'onglets et de cadres, qui facilitent l'accès aux options et fonctions disponibles.

La perspective de Java est une perspective standard qui a été désignée pour le développement du code Java. Parmi d'autres choses incluses on trouve : un navigateur et un éditeur Java. N'importe quel cadre de Poseidon peut être ajouté à cette perspective via UML.

La Perspective UML

La Perspective UML est la manière recommandée pour travailler avec Poseidon dans Eclipse

Parce qu'elle a été désignée pour garder l'interface connue de Poseidon.

Pour ouvrir la perspective UML, cliquer le bouton 'Open UML Perspective' à partir de la barre d'outils ou du sélecteur de perspectives dans le coin droit en haut.

Cas d'utilisation

Présentation du problème

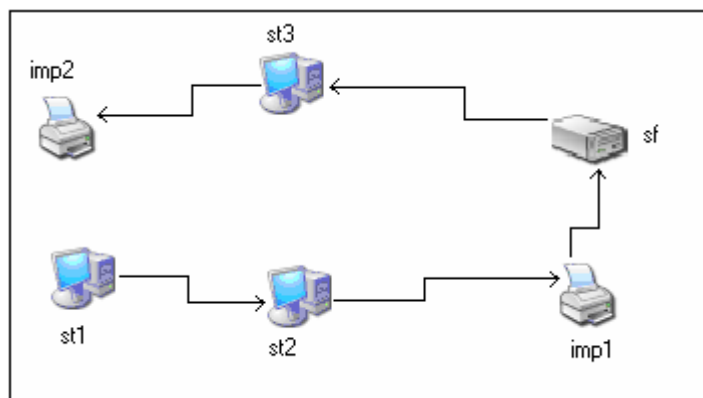
On va essayer de suivre les étapes de création d'un exemple en forme d'un tutorial afin de mieux voir les différentes fonctionnalités de Poseidon for UML.

Il s'agit de concevoir une application permettant de simuler un réseau informatique. Nous allons partir du principe qu'un réseau est constitué d'un ensemble de nœuds reliés les uns aux autres. Dans un premier temps, nous allons simplifier le problème en considérant que chaque nœud du réseau est relié à un seul autre nœud. Un nœud est identifié dans le réseau par une adresse unique et possède également un nom. Au sein du réseau, un nœud reçoit des paquets d'informations et re-expédie éventuellement ces paquets si ceux-ci ne le concernent pas.

Dans la réalité, un réseau informatique contient généralement des nœuds de différents types (station de travail, passerelle, imprimante, terminal, serveur fichier, etc.). Pour refléter au mieux cette réalité au niveau de notre simulation, nous allons représenter les types de nœuds suivants :

- Station de travail
- Imprimantes
- Serveurs de fichiers

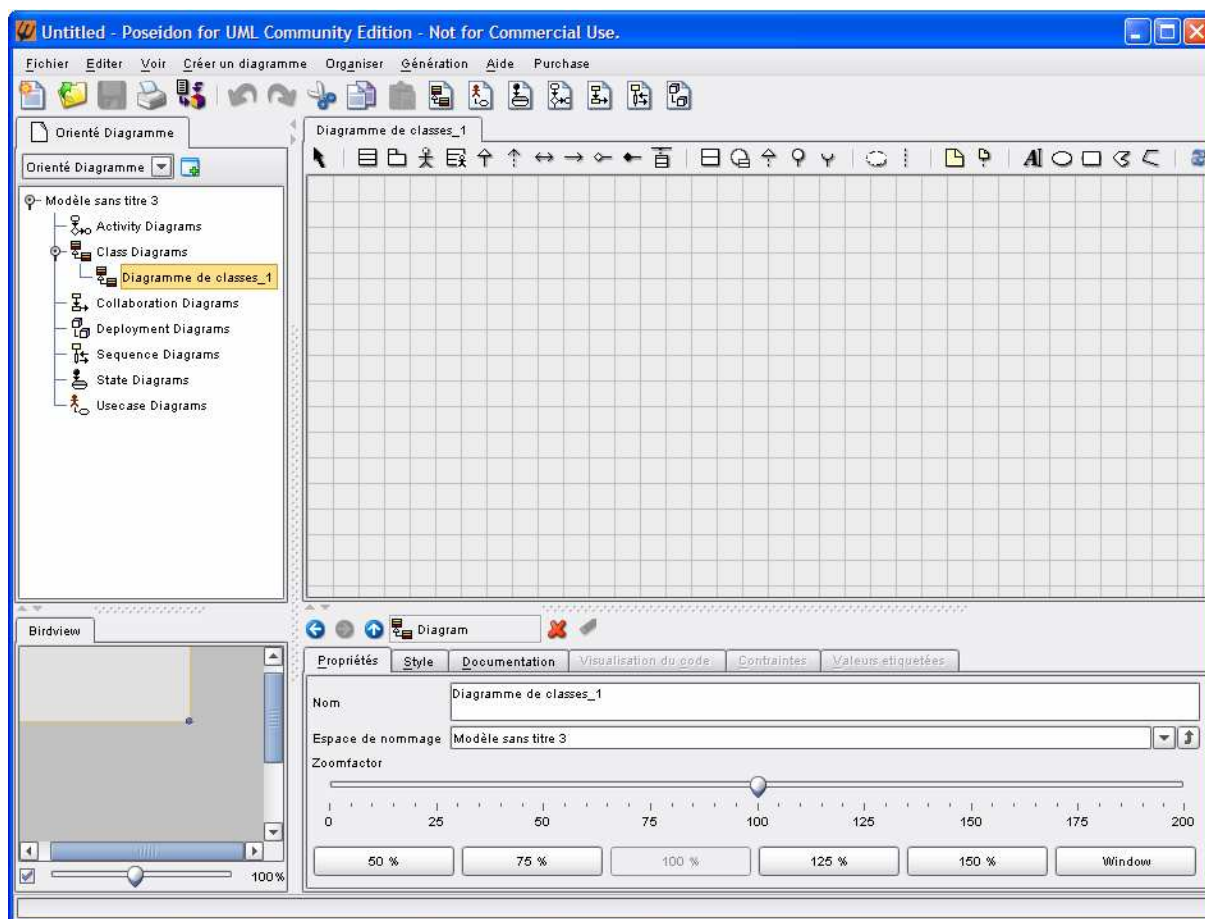
Un exemple de réseau basé sur ces informations est présenté ci-dessous :




Chaque type de nœud va se distinguer par la manière dont il traite les paquets qui lui sont destinés. Nous préciserons le comportement relatif à chaque type de nœud au fur et à mesure de l'élaboration de l'application.

Diagramme de classes

On commence tout d'abord par créer un nouveau projet. *Fichier* → *Nouveau projet*. Ceci est la fenêtre classique d'un projet vide.

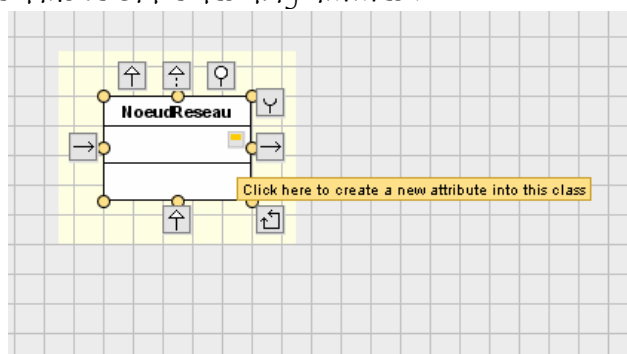



Renommez le modèle 'Modèle sans titre x' et le diagramme de classes 'Diagramme de classes_1' qui a été créé par défaut. Dans cet exemple il n'est pas nécessaire d'utiliser des paquetages, donc on crée directement l'ensemble des classes.

Appuyez sur le bouton  dans la barre en haut du cadre des diagrammes puis une deuxième fois à l'intérieur du cadre. Vous venez de créer une nouvelle classe. Entrez le nom de la classe : 'NoeudReseau' puis validez par Entrer.

Vous remarquerez que le cadre des détails affiche les informations de la classe.

On va ajouter un attribut à cette classe. Mettez le curseur de la souris au dessus de la classe dans le cadre des diagrammes :

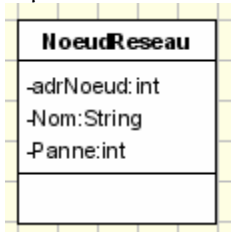


Le symbole  apparaît. Appuyez dessus et tapez 'adrNoeud : int' puis appuyez sur un espace vide du cadre. C'est l'une des méthodes de création d'attributs. Vous pouvez aussi appuyer sur le même symbole dans le cadre des détails.



Sélectionnez l'attribut nouvellement créé. Le cadre des détails affichera cette fois des informations sur l'attribut.

Le bouton  vous permet de remonter vers la classe. Alors que  vous affiche l'attribut suivant.

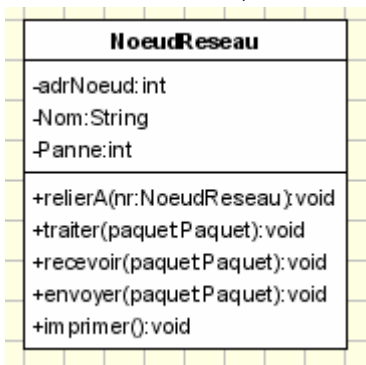
Ajoutez les attributs jusqu'à ce que votre classe devienne comme suit :



Appuyez maintenant sur le bouton  Afin d'ajouter une méthode. Donnez lui le nom 'relierA'. Puis appuyez sur  afin d'ajouter un paramètre à cette méthode. Donnez lui le nom 'nr' avec le type 'NoeudReseau (from simulation d'un réseau)'.


Vous trouverez les mêmes boutons de navigation  et  pour revenir à la méthode et parcourir les attributs respectivement.

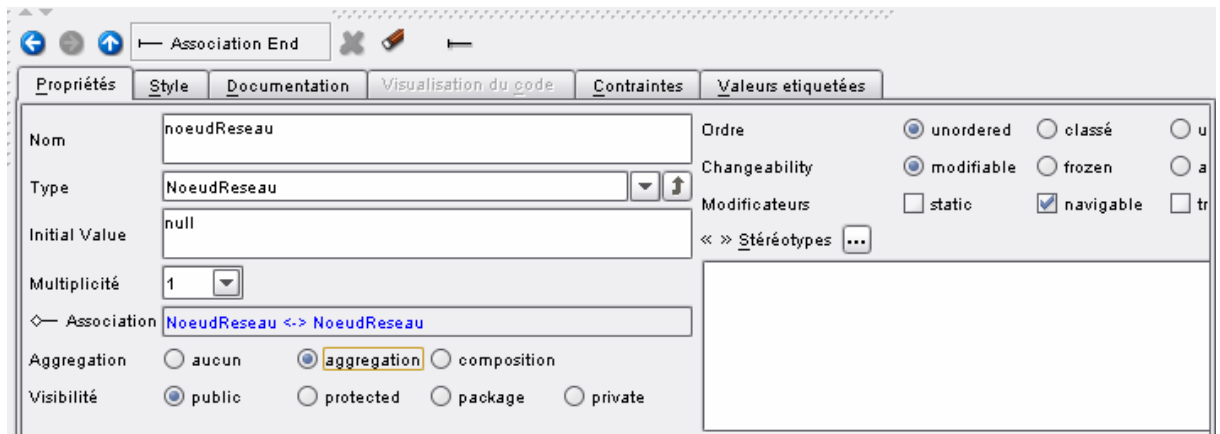
Entrez les méthodes suivantes :



Remarquez que le 'Paquet' n'est pas reconnue comme type par le logiciel. C'est une classe qu'on va entrer ultérieurement. Poseidon for UML vous laisse entrer des éléments pas encore définis. Quand vous voudrez spécifier le type de 'paquet', tapez 'Paquet' dans le champ Type. Une liste déroulante va apparaître. Choisissez 'New class' et Poseidon va la déclarer comme une classe du modèle jusqu'à ce que vous la définissiez.

On va voir maintenant plus en détails quelques options sur les attributs et les méthodes. Les attributs sont private et les méthodes sont public par défaut. Sélectionnez tous les attributs de la classe en maintenant la touche Ctrl enfoncée puis appuyez sur le bouton radio protected dans le champ Visibilité.

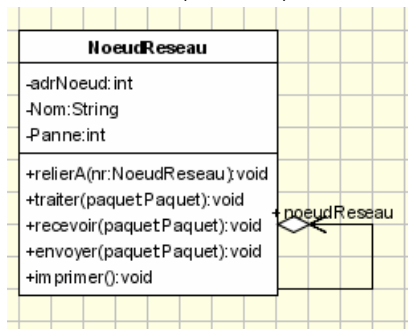
Maintenant on veut ajout un attribut 'noeudReseau' qui est de type de la classe en cours. La meilleure façon de faire est de l'ajouter sous forme de relation. Déplacez le curseur au dessus de la classe. Vous verrez apparaître le symbole  qui vous permet de créer une relation récursive. Appuyez sur un bout de la relation créée et donnez lui un nom, une valeur initiale 'null' et activez agrégation. Le cadre des détails devrait ressembler à ceci :



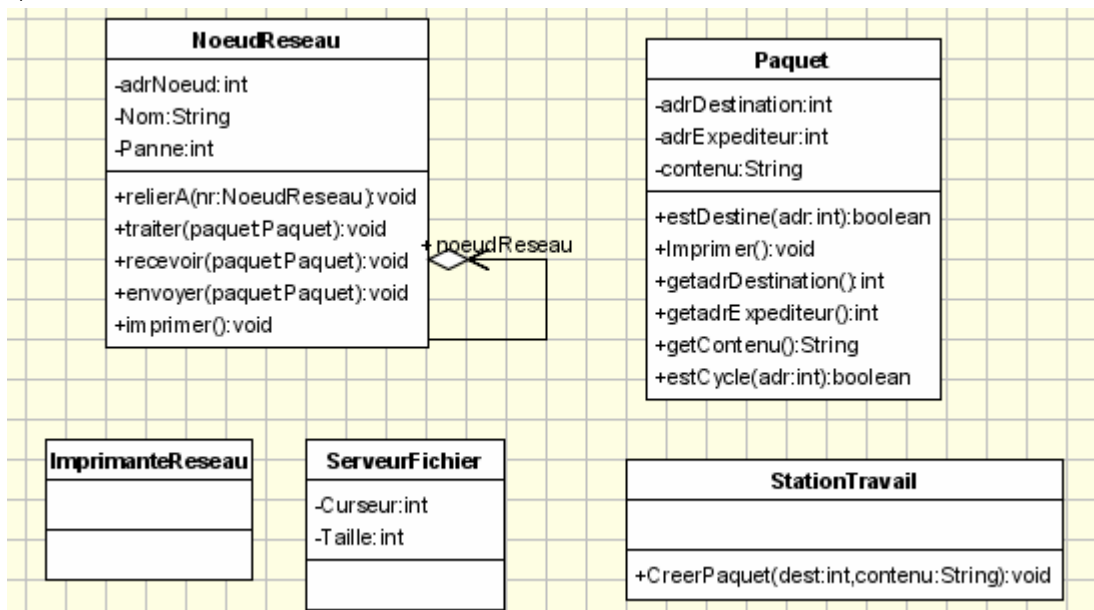
Sélectionnez l'autre bout et décochez navigable.

Vous venez de créer une agrégation à sens unique portant le nom de 'noeudReseau' au sein de la même classe. Ce qui revient à créer un attribut ayant le type 'NoeudReseau'. Mais cette méthode est plus claire et formelle, quoique moins évidente que la simple création d'attribut.

Votre classe devrait désormais ressembler à ceci :





Vous pouvez maintenant créer les autres classes du modèle avec les mêmes procédures :



Les fonctions `Paquet::getadrDestination`, `Paquet::getadrExpediteur` et `Paquet::getContenu` sont des fonctions d'accès simple à des attributs. Vous pouvez l'indiquer à Poseidon for UML pour qu'il puisse en prendre considération lors d'opérations ultérieures comme la génération de la documentation. Pour ce faire, il suffit de sélectionner ces méthodes une à une et sélectionner l'attribut correspondant dans le champ `Accessed` attribut.

Les classes `ImprimanteReseau`, `ServeurFichier` et `StationTravail` héritent toutes de la classe `NoeudReseau`. Il faudra l'indiquer dans le modèle.

Appuyez sur le symbole  afin de créer des relations d'héritage. Vous le faites en cliquant sur le bouton gauche de la souris et en maintenant enfoncée d'abord sur la classe fille et en lâchant au dessus de la classe mère.

La classe `ServeurFichier` contient l'attribut '`paquet: Paquet[]`'. Pour les créer, la meilleur façon est de créer une relation d'agrégation en configurant les 2 bouts comme fait précédemment. Sauf que cette fois vous devrez utiliser le symbole  à la manière de l'héritage et choisir une multiplicité *.

Enregistrez votre modèle. Il doit ressembler à cette apparence à la fin :

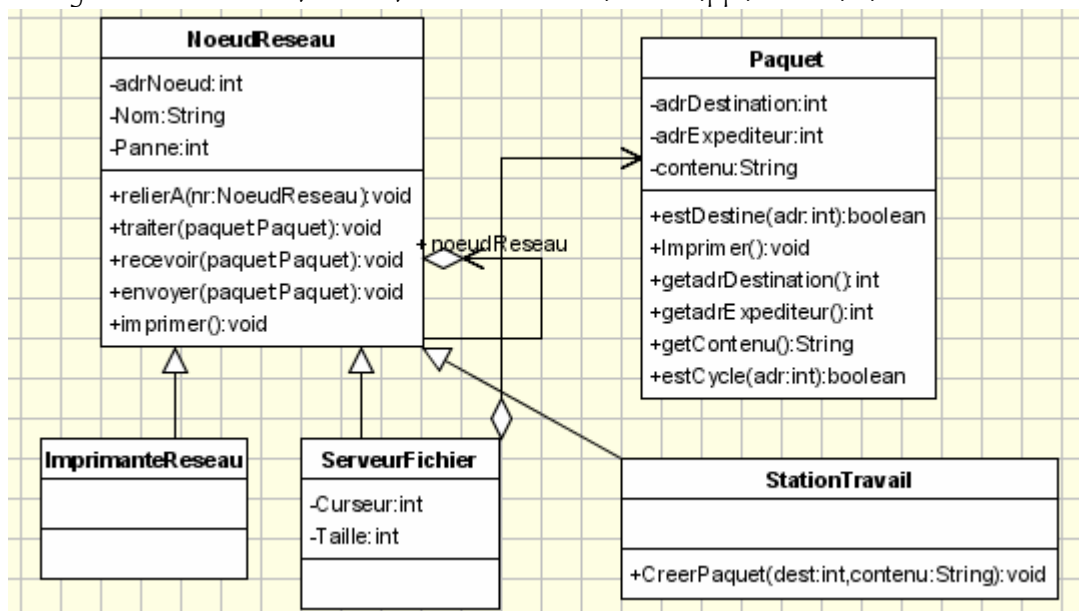


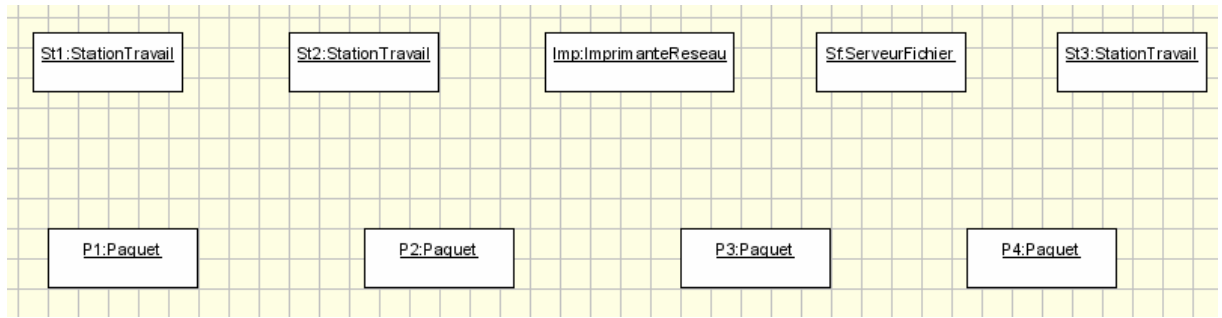


Diagramme d'objets


Maintenant on veut préciser l'état des objets lorsque la simulation débute. Pour cela, on utilise un diagramme d'objets.

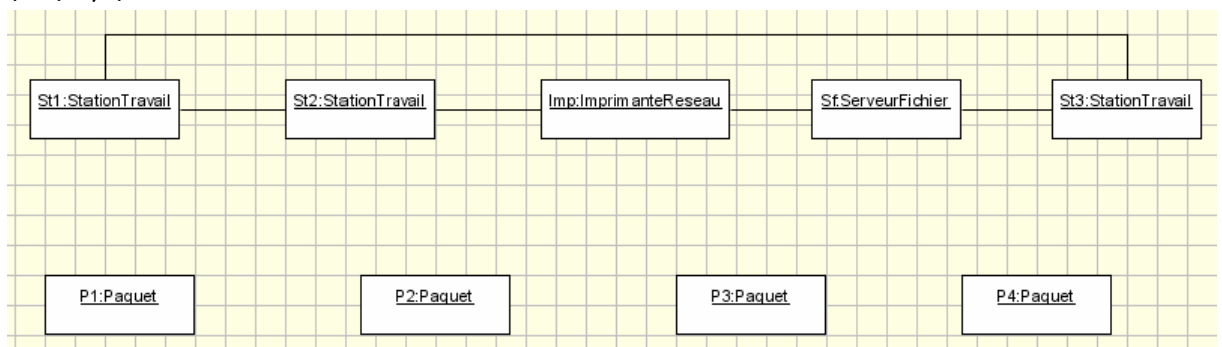
On commence par créer un diagramme d'objet. Pour cela, cliquez sur le symbole  situé dans la barre d'outils principale. Un nouveau diagramme d'objet (i.e. de déploiement pour Poseidon for UML) apparaît dans le cadre de navigation. Renommez le si besoin est.

On crée d'abord les nœuds réseaux. Pour créer un objet utilisez le bouton  appuyez sur le cadre des diagrammes puis remplissez le nom et le type comme vous avez fait précédemment avec les classes. Entrez les objets comme suit :

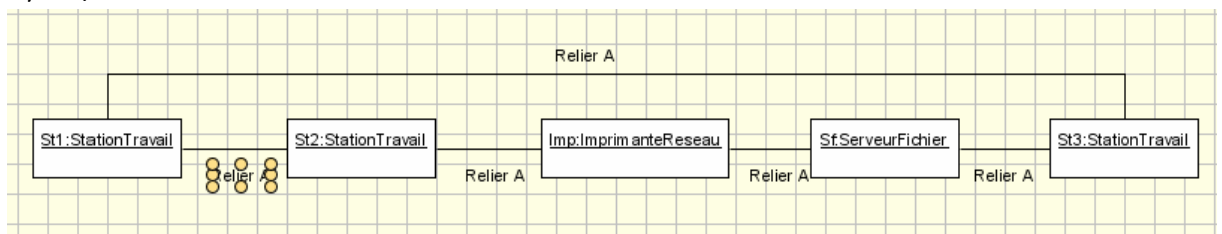


C'est l'ensemble des objets créés au début de la simulation. Il faut maintenant indiquer les relations qui existent entre ces objets. Tout d'abord les nœuds réseaux sont reliés entre eux.

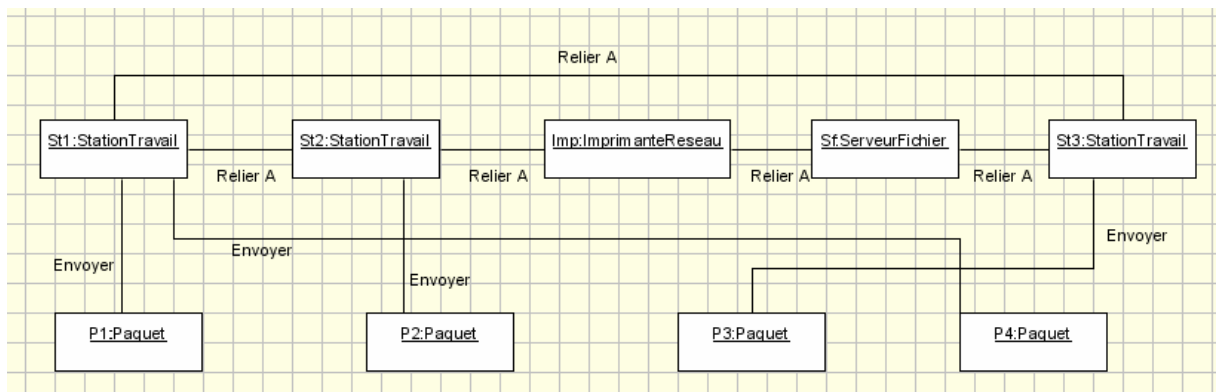
Utilisez le bouton  afin d'indiquer une relation. Les relations entre les nœuds se présentent ainsi :



Poseidon for UML permet d'inclure des éléments externes au modèle afin d'enrichir sa compréhension. Dans notre cas, on peut utiliser du texte **A** afin d'indiquer la nature des relations :




Finalement on rajoute les relations d'envoi de paquets et notre diagramme est complet :



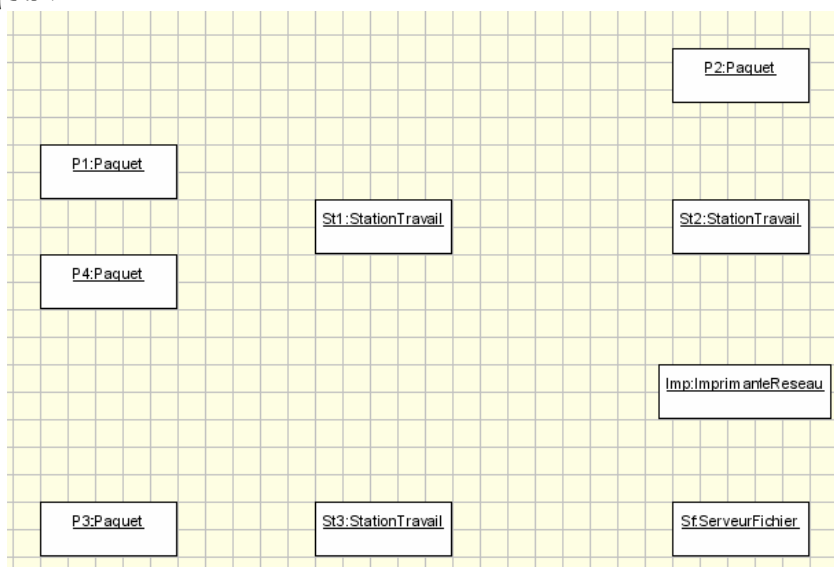
Remarquez que pour les relations, on peut créer des points de contrôle pour modifier leur forme simplement en cliquant et tirant dessus.



Diagramme de collaboration

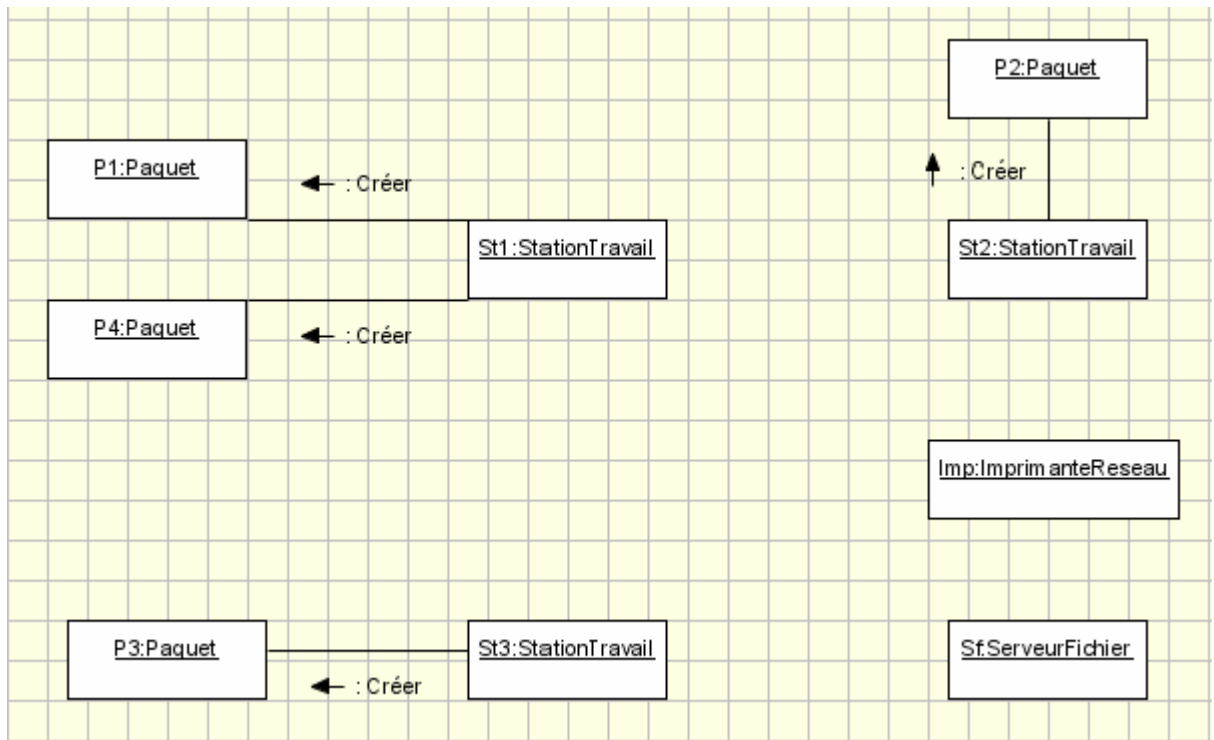
Afin de mieux comprendre les interactions entre objets, on va utiliser un diagramme de collaboration. Vu que le facteur de temps ne nous intéresse pas dans ce programme, nul besoin de réaliser un diagramme de séquences.

Utilisez le bouton  afin de créer un nouveau diagramme de collaboration et renommez le.

On commence par créer les objets de la même manière qu'avec le diagramme d'objets :



On introduit les messages de création des paquets. Utilisez le bouton  afin de créer une relation puis déplacez le curseur sur la relation créé. Vous verrez apparaître le symbole . Appuyez dessus afin de créer un stimulus que vous pouvez modifier dans le cadre des détails afin de modifier le sens ou autres paramètres.



On introduit ensuite les transitions et liens entre les nœuds. Ce qui donne finalement :

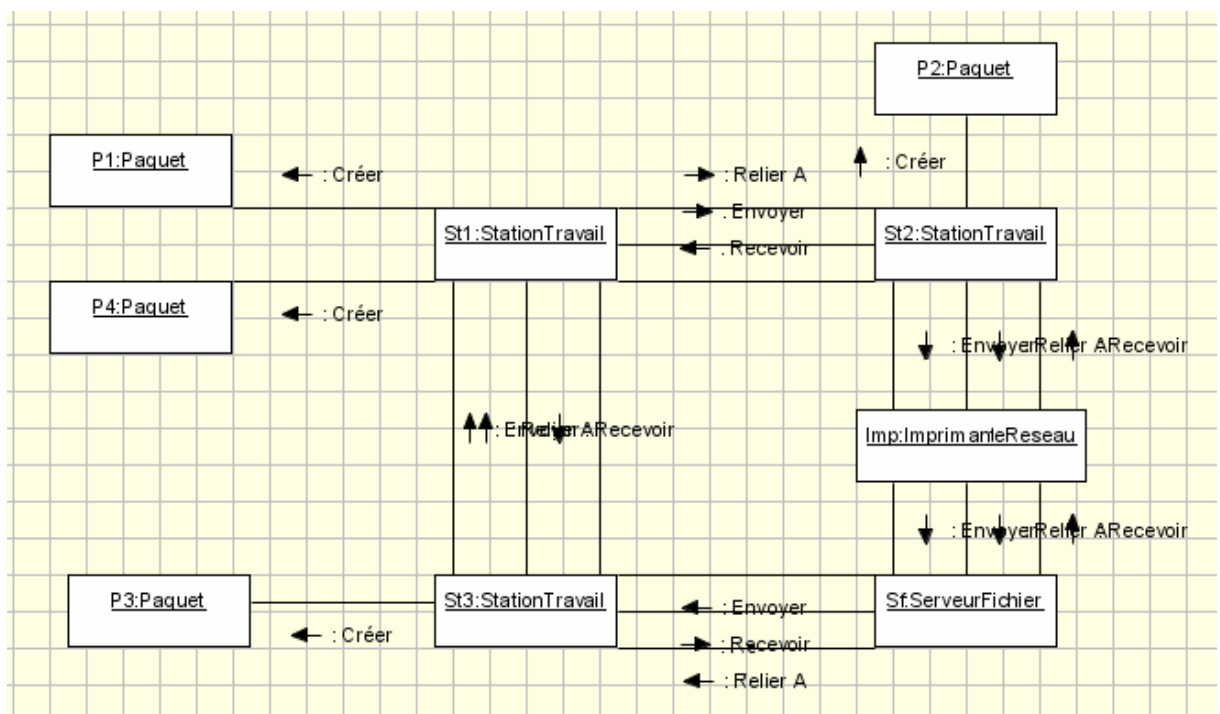





Diagramme d'états

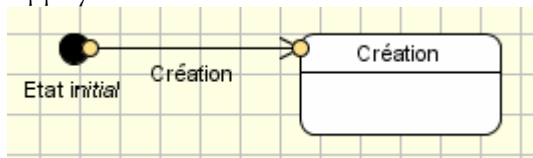
Nous allons nous intéresser aux états des paquets, car ces eux qui donnent vie au réseau. Nous allons donc commencer par la création de ces derniers jusqu'à leur

traitement. Le diagramme d'états nous fournira une excellente vue des différents états des paquets.

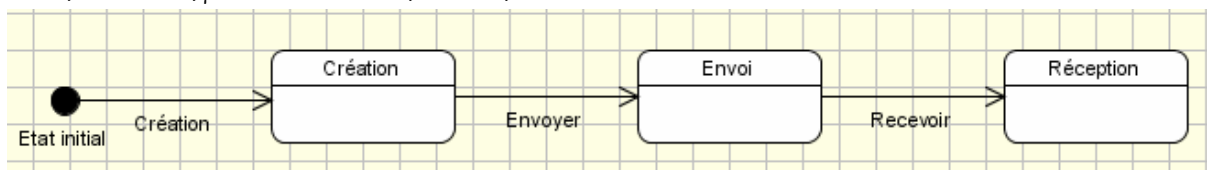
Créez tout d'abord un nouveau diagramme d'état à l'aide de  et renommez le.


On commence par l'état initial. Créez le avec le bouton  et donnez lui un nom.




En maintenant le curseur au dessus de l'état initial vous verrez apparaître le bouton . Appuyez dessus afin de transiter vers un nouvel état et donnez le nom 'création' à cet état. Appuyez ensuite sur la transition et nommez la 'construire'. Vous obtenez :



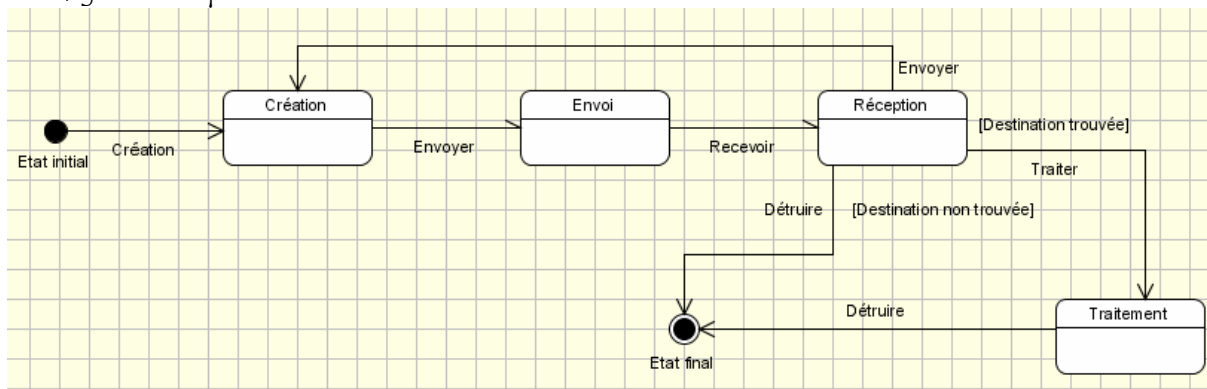
De la même façon créez les états suivants :



Depuis la réception on doit indiquer que le processus d'envoi peut recommencer. Créez une transition de Réception vers Envoi à l'aide de .

Créez un état final  et un état 'Traitement'. On veut créer deux transitions conditionnées de l'état Réception vers l'état final puis vers et vers l'état 'Traitement'. Créez les deux transitions puis appuyez sur le bouton  [A] Garde  Afin d'ajouter une condition à chaque transition.

Ajoutez finalement une transition de Traitement vers l'état final. Vous devez avoir à la fin un diagramme qui ressemble à ceci :

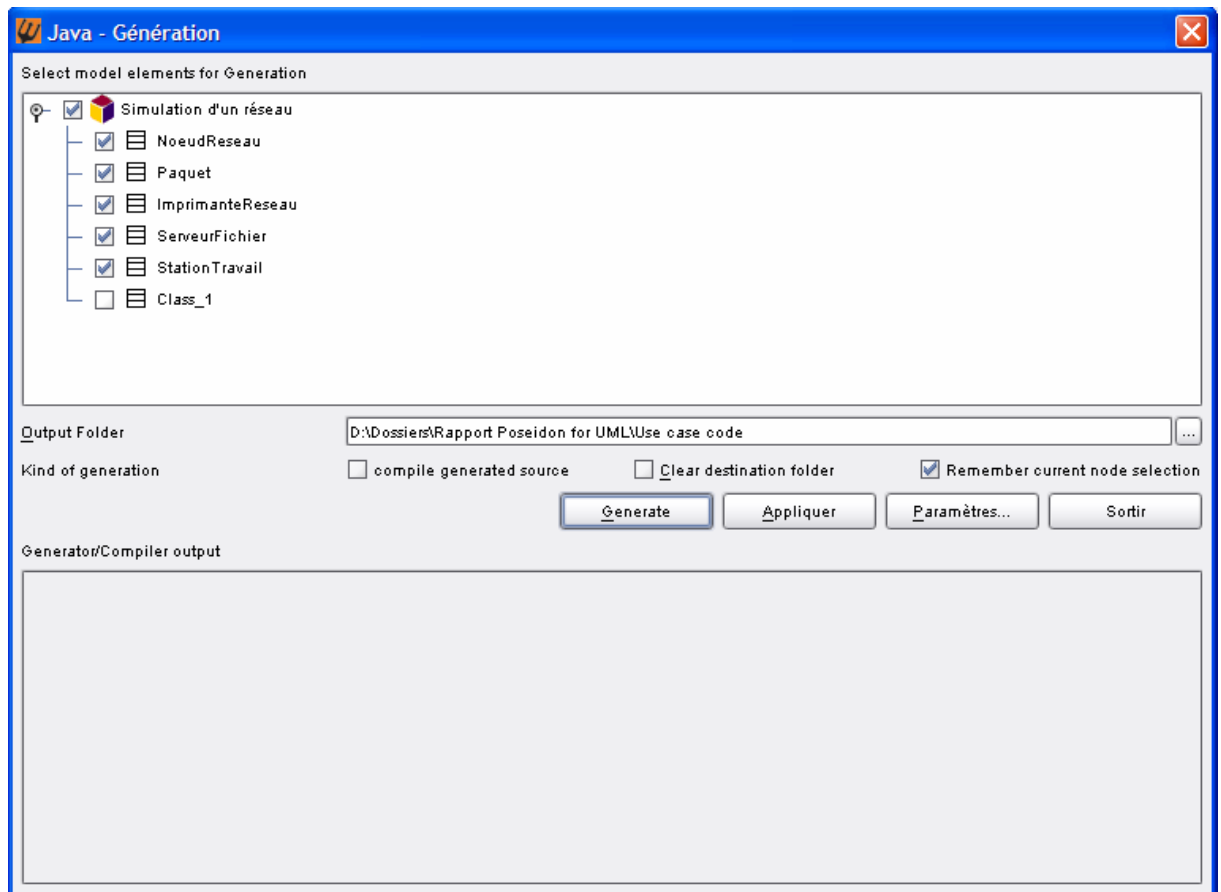


Ceci termine la conception de notre programme.

Fonctionnalités avancées

On peut maintenant tester certaines fonctionnalités avancées de Poseidon for UML.

On veut générer le code source java correspondant à notre diagramme de classe. Il suffit d'appuyer sur 'Java...' dans le menu 'Génération'. Une boîte de dialogue s'affiche vous permettant de choisir l'ensemble des classes à partir desquelles le code java va être généré :

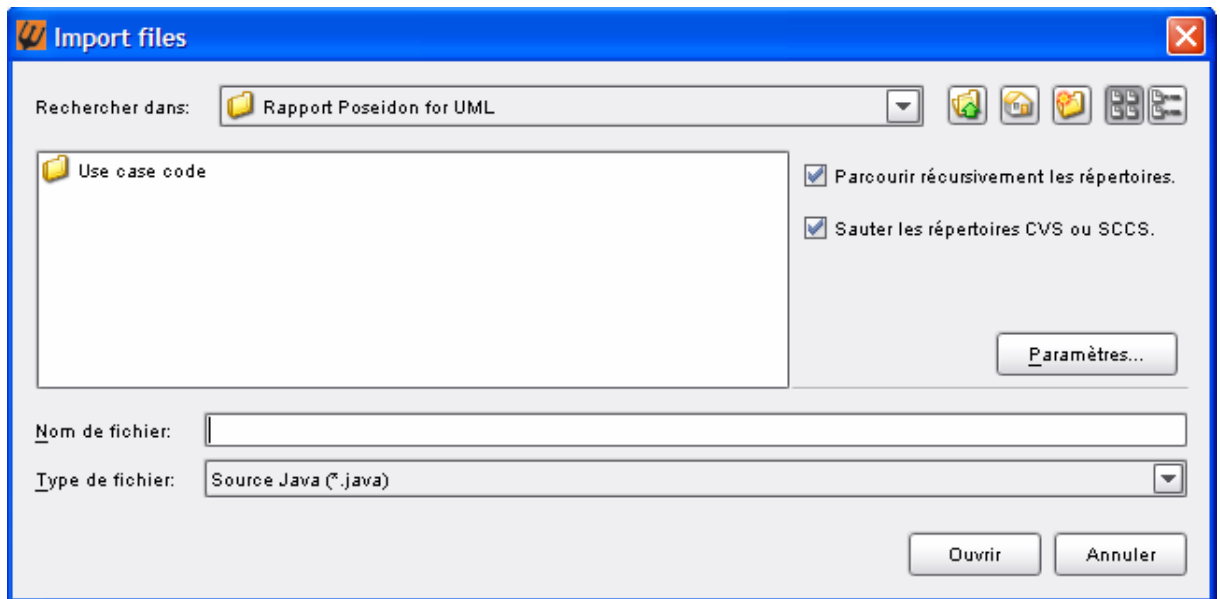


Indiquez un répertoire vide et appuyez sur le bouton 'Générateur'. Vous pouvez cocher 'Compile generated source' afin de générer directement du bytecode (il faut installer un SDK Java complet et indiquer son chemin dans 'Paramètres' pour accéder à cette fonctionnalité).

Le code de chaque classe sera généré dans un fichier séparé. Les associations seront transformées en variables ou objets selon leur multiplicité et leur navigabilité. Visualisez le code généré afin de mieux comprendre le mécanisme.

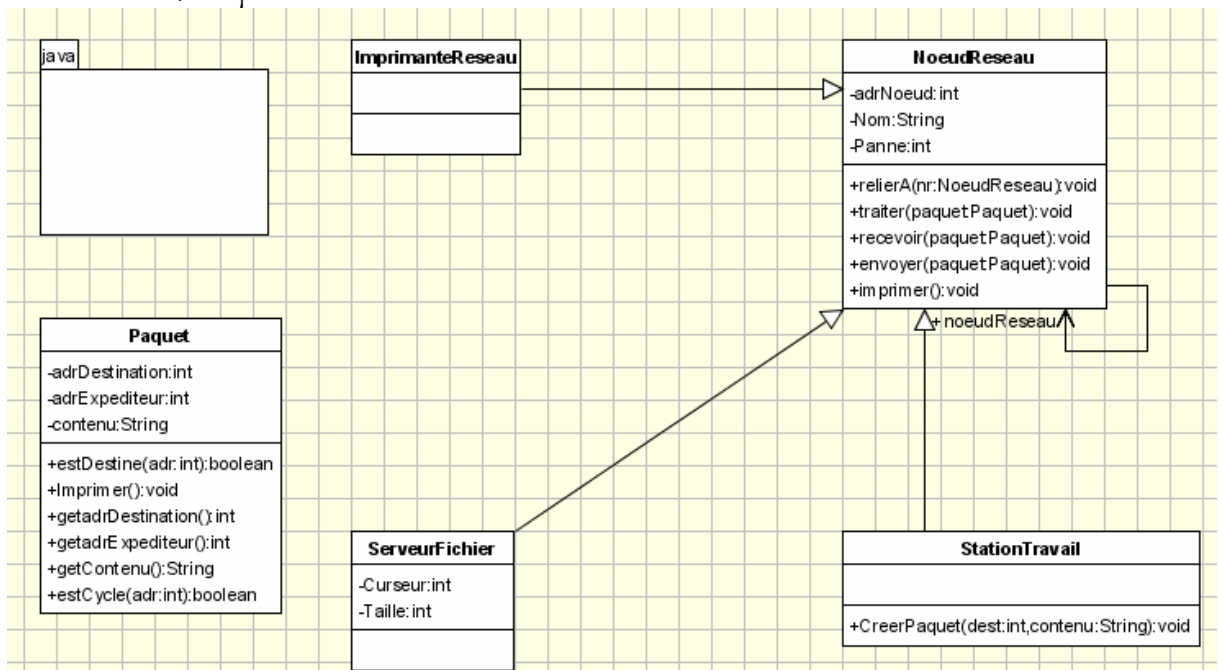
On peut maintenant faire l'opération inverse en générant un modèle à partir des fichiers générés.

Ouvrez un nouveau projet puis sélectionnez 'Fichier->Import files'. Une boîte de dialogue vous demandant les fichiers à importer s'affiche. Donnez le chemin du répertoire où vous avez mis vos fichiers précédemment générés puis appuyez sur ouvrir :



Cochez la racine de l'arborescence qui vous sera proposée puis validez. Le résultat de cette importation est que vous allez retrouver exactement le même modèle d'avant. Ceci montre l'efficacité du mécanisme de Reverse-engineering de Poseidon for UML.

Remarquez que pour afficher toutes les informations d'une classe, vous devez appuyer dessus avec le bouton droit de la souris puis cocher 'Show->All'. Le diagramme généré ressemble à ce qui suit :



Conclusion

Après notre travail de recherche, nous recommandons Poseidon for UML comme outil de modélisation UML. Non seulement il présente toutes les fonctionnalités dont un utilisateur pourra en avoir besoin mais son architecture ouverte et son énorme flexibilité en font un outil de choix. En plus de l'utiliser, il est possible d'étudier son API afin de le personnaliser pour vos propres applications en créant des plug-ins. Ces plug-ins peuvent aussi être proposés à Genteware afin de les commercialiser.

Grâce à son énorme contribution dans le milieu Open Source et ses expériences passées sur les outils UML, Genteware a su développer un outil libre et performant au grand bonheur des ingénieurs en développement logiciel.

Références

Le site web <http://uml.developpez.com>

Le guide d'utilisateur de Poseidon for UML

Le site officiel <http://gentleware.com>

Le document **Le langage de modélisation objet UML** par le Département Informatique de l'Institut Universitaire de Technologie de l'Université Bordeaux 1 O. Guibert