Normalisation de l'apprentissage par travaux pratiques des systèmes d'exploitation

Standardization of learning by operating systems labs

Mohamed Taha Bennani, Mohamed Abidi Allagui, Yacine Ben Nacer
Unité de recherches : Méthodes et outils pour les systèmes informatiques complexes, Institut supérieur d'informatique et de mathématiques de Monastir, Tunisie

Résumé

L'enseignement des notions liées aux systèmes d'exploitation pose un problème d'homogénéité. En cours et travaux dirigés, les étudiants abordent les mécanismes, par exemple : gestion de mémoire, de processus, etc. du noyau. L'implémentation de ces mécanismes est difficile à traiter au niveau des travaux pratiques à cause du temps nécessaire à la compilation du noyau. Les étudiants restent confinés à la manipulation de ces mécanismes au niveau de l'interface utilisateur. Plusieurs simulateurs ont essayé de résoudre ce problème en proposant des solutions permettant de montrer l'exécution de ces mécanismes. D'autres ont essayé de faciliter leurs mise-en-œuvre en utilisant des micro-noyaux dont le temps de compilation est court. Ce papier présente un simulateur qui permet de montrer l'exécution des mécanismes du noyau, de donner la main aux étudiants pour les développer et de les intégrer dans cette plateforme afin de simuler leurs propres implémentations. Cette plateforme obéit aux exigences principales de la norme SCORM. Ce papier est constitué de trois parties : Normalisation et simulateurs, spécification et conception de la plateforme éducative.

Mots clés : SCORM, simulateur, travaux pratiques, systèmes d'exploitation.

Abstract

The courses and tutorials content on operating systems concepts is different from laboratories. This difference leads to some learning inconsistency. The kernel mechanisms are introduced in the courses and their related exercises are realized during the tutorials. As the kernel compilation last about forty five minutes, the implementation of the kernel mechanisms is not feasible. Thus, instead of implementing these mechanisms, the labs exercises handle the kernel concepts at the application programming interface. This shortcoming leads to the learning inconsistency. Several simulators have been implemented to show the execution of the kernel mechanisms. Others decrease the compilation time using micro-kernel based approach. These solutions handle the kernel modification which is suitable for the concepts learning. This paper presents a simulator that can show the performance of these mechanisms, to be implemented by students and integrated into this platform to simulate their own implementations. This platform follows the main requirements of SCORM standard. The outline of the paper contains three sections: Standard and simulators, specification and design of the learning platform.

Keywords: SCORM, simulator, labs, operating systems.

I. Normalisation et simulateurs

L'apprentissage est la pierre angulaire dans le transfert technologique. Il repose sur le concept d'objet d'apprentissage qui est "toute entité numérique ou non, qui peut-être utilisée, réutilisée ou référencée lors d'une formation dispensée à partir d'un support technologique" [1]. Plusieurs approches ont essayé de définir et de gérer ces objets d'apprentissages : LOM (*Learning Object MetaData*), SCORM (*Sharable Content Object Reference Metadata*), etc.

Dans LOM, les objets d'apprentissages sont indexés dans le but de les réutiliser dans des unités de plus haut niveau. Par exemple, une leçon peut être utilisée dans plusieurs cours [1]. L'approche SCORM [2] définit des objets à contenu partageable pouvant être mis en ligne facilement. Elle enrichit le standard LOM avec un nouveau modèle d'agrégation et un environnement d'exécution qui permettent de surveiller l'activité d'un apprenant dans un LMS (*Learning Management System*). Ce nouveau modèle se compose de trois niveaux : base, intermédiaire et supérieur. Le niveau de base est appelé "*asset*" est formé d'éléments qui ne communiquent pas avec le LMS. Les objets à contenus partageables "*Sharable Content Object, SCO*" se trouvent au niveau intermédiaire. C'est l'élément atomique dont l'utilisation pourrait être tracée, par exemple, une simulation a-t-elle été effectuée ? Contrairement à l'asset, le niveau intermédiaire est constitué d'éléments pouvant communiquer avec le LMS. Le niveau supérieur est une agrégation constituant le contenu d'un cours.

Dans notre papier, le simulateur de la gestion de la mémoire est constitué de deux objets d'apprentissages : le simulateur et l'environnement de développement intégré, du niveau intermédiaire. Le cours sur les systèmes d'exploitation, qui est un élément du niveau supérieur, est une agrégation de l'ensemble des simulateurs des mécanismes du noyau (voir la section §I.A) et des supports de cours et des travaux dirigés. Par ailleurs, l'environnement de développement intégré peut constituer un objet d'apprentissage dans le cours "Programmation C".

A. Simulation de la gestion de la mémoire

Les systèmes d'exploitation (*Operating System, OS*) ont pour rôle d'offrir une couche d'abstraction du matériel. Pour assurer ce niveau d'abstraction un OS fournit quatre services : la gestion de processus, la gestion de mémoire, la gestion de fichiers et la gestion des entrées/sorties. Avec les OS libres, le code des différents gestionnaires est accessible. Cependant, il est complexe et long. Par exemple, la version 0.02 de LINUX comportait dix-mille lignes de code. Étudier de telles sources relève de l'impossible au niveau des travaux pratiques.

Ce papier présente une partie d'une plateforme éducative (PE) dédiée à la simulation d'un système d'exploitation et la mise en œuvre de certaines de ces parties. Cette plateforme vise à aider les étudiants à comprendre les algorithmes mis en jeux dans un OS en les simulant. Aussi, elle permet d'intégrer du code dans le simulateur afin de vérifier les algorithmes développés par les étudiants. Pour assurer ces objectifs et rendre cet apprentissage simple, la PE est purement graphique. L'accessibilité de l'apprentissage est aussi assurée par la portabilité de notre solution. Elle est exécutable sur n'importe qu'elle machine cible. Il suffit d'avoir la machine virtuelle JAVA installée.

La partie développée dans ce projet traite la gestion de la mémoire (i.e. PE-GM). L'OS gère trois types de mémoires : la mémoire cache, la mémoire vive et la mémoire virtuelle. Dans le cadre de cette plateforme éducative, nous nous limitons à l'organisation de l'information entre la mémoire virtuelle et la mémoire vive. L'organisation de la mémoire vive peut conduire à des situations où une partie de la mémoire virtuelle doit être chargée au niveau de la mémoire vive. Dans ce cas, il faut choisir la partie de la mémoire vive qui sera remplacée. Cette sélection est réalisée par les algorithmes de remplacement de pages.

B. Travaux connexes

Il existe essentiellement trois simulateurs de systèmes d'exploitation : MOSS (*Modern Operating Systems Simulators*), BOCHS et SIMDEF (Simulateur de défaut de page).

1. MOSS

MOSS [3] est une application portable écrite en Java. Elle contient une collection des programmes java qui simule les systèmes d'exploitation. Elle a été conçue par des étudiants. MOSS permet de simuler la gestion des processus, la gestion de la mémoire et la gestion de système de fichier. Elle est simple à utiliser. Son interface intuitive permet une prise en main rapide.

2. BOCHS

BOCHS [4] est une application en source libre développée en C++ qui s'exécute sous la plupart des systèmes d'exploitation. Elle assure une simulation du microprocesseur Intel x86, de son périphérique d'entrée/sortie et du BIOS (*Basic Input Out Put System*). BOCHS offre une simulation fidèle aux implémentations réelles des mécanismes du noyau. Ceci conduit à la manipulation des structures de données avancées ne pouvant être manipulées que par des utilisateurs avancés.

3. SIMDEF

SIMDEF [5] (Simulateur de défauts de pages) est une application portable écrite en Java qui permet de simuler la gestion de la mémoire. Elle a été développée dans le cadre d'un projet de fin d'études. Elle propose une interface simple à utiliser intégrant deux types de simulations : matricielle et statistique. Dans le premier type de simulation, les résultats sont représentés graphiquement. Dans le second type de simulation, les résultats apparaissent sous la forme de chiffres et de pourcentages. SIMDEF implémente trois algorithmes de remplacement des pages. Pour chacun d'entre eux, elle dresse un graphique permettant de montrer l'évolution de l'occupation des cadres de pages en mémoire ainsi que le nombre des défauts de pages.

4. Critiques par rapport aux standards

Le *tableau 1* présente les avantages et les inconvénients des simulateurs actuels.

Tableau I : Évaluation des simulateurs existants

Simulateur / Critères	MOSS	BOCHS	SIMDEF
Accessibilité	Non	Non	Non
Adaptabilité	Oui	Oui	Oui
Durabilité	Oui	Oui	Oui
Interopérabilité	Oui	Oui	Oui
Réutilisabilité	Non	Oui	Non
Nombre d'algorithmes	1	1	3
Nombre de SCO	1	2	1

Nous retenons les propriétés préconisées par SCORM [2] pour l'évaluation d'un objet d'apprentissage : Accessibilité, adaptabilité, durabilité, interopérabilité, réutilisabilité. En outre, nous

introduisons deux autres critères : le nombre d'algorithmes et le nombre d'objets à contenu partageable (SCO). L'accessibilité dénote la capacité de repérer le composant d'apprentissage et d'y accéder à partir d'un site distant. Les trois simulateurs étudiés, dans leurs versions actuelles, doivent être installés sur la machine de l'étudiant pour être utilisés. Par conséquent, ils ne sont pas accessibles.

La possibilité de personnaliser le composant relève de son adaptabilité. Ce second critère touche essentiellement la modification de l'algorithme de remplacement de page à utiliser. Cette modification est nécessaire pour le besoin de l'étudiant et de l'enseignant. Le simulateur SIMDEF ne permet pas la modification de l'algorithme utilisé. Cependant, les simulateurs MOSS et BOCH offrent la possibilité à l'utilisateur de saisir son propre algorithme de remplacement de pages.

La durabilité dénote le pouvoir du composant à résister à l'évolution technologique sans nécessiter un nouveau développement. Cette faculté n'est pas assurée par BOCHS et SIMDEF. En revanche, MOSS représente une solution durable. En effet, les traitements réalisés par BOCHS reposent sur l'architecture du processeur sous-jacent. Par conséquent, une évolution architecturale nécessiterait un nouveau développement. Puisque SIMDEF n'est pas adaptable, la création d'un nouvel algorithme de remplacement de pages nécessiterait un nouveau développement. En revanche, le simulateur MOSS est ouvert et indépendant de l'architecture matérielle.

L'interopérabilité nous renseigne sur la possibilité d'utiliser le simulateur sur plusieurs plateformes. Tous les simulateurs traités sont interopérables. MOSS et SIMDEF ont utilisé le langage de programmation JAVA pour assurer cet aspect. En revanche, le simulateur BOCH est implémenté en utilisant le langage de programmation C. L'interopérabilité à été assurée suite au portage de l'application initiale sur plusieurs plateformes.

Un composant dont l'intégration dans d'autres contextes et dans des applications multiples est un composant réutilisable. Les simulateurs MOSS et SIMDEF ne sont pas réutilisables dans une autre agrégation de contenu autre que celle liée au cours du système d'exploitation. En revanche, le simulateur BOCHS peut être intégré dans le cadre d'un cours sur les architectures des ordinateurs.

Le sixième critère montre le nombre d'algorithmes implémentés par le simulateur. Nous distinguons trois algorithmes : FIFO (*First In First Out*), LRU (*Least Recently Used*) et optimal. Nous remarquons que les deux premiers simulateurs évalués n'implémentent qu'un seul algorithme. Le simulateur SIMDEF offre les trois types d'algorithmes de remplacement de pages.

Le nombre de SCO montre le degré de maturité du simulateur et son potentiel à utiliser des composants pouvant être intégrés dans d'autres agrégations. SIMDEF et MOSS offrent un simulateur formé uniquement d'assets. En revanche, BOCHS offre un simulateur de plateforme matériel et des mécanismes qui permettent de la gérer. Par conséquent, le simulateur de la plateforme représente un SCO pouvant être agrégé dans un cours d'architecture des ordinateurs. La simulation des mécanismes de gestion de la plateforme peut faire partie du cours systèmes d'exploitation.

Notre simulateur sera composé de deux SCO : un simulateur des mécanismes de gestion de la mémoire et un environnement intégré de développement. Ce dernier est nécessaire pour la simplification de la modification du cœur du simulateur. Il pourra être agrégé dans le cadre d'un cours sur la programmation C.

C. Proposition

Nous nous proposons de réaliser une plateforme éducative dédiée aux étudiants pour les assister dans les travaux pratiques en systèmes d'exploitation, particulièrement, la gestion de la mémoire. Les étudiants sont invités à simuler les algorithmes qu'ils ont déjà vus dans le cours. Les algorithmes offerts par la plateforme sont : LRU, FIFO et optimal. Dans le cas où l'étudiant voudrait proposer un autre algorithme et voir son comportement, il peut recourir à un environnement de développement

intégré (*Integrated Development Environment, IDE*) des algorithmes que nous intégrons dans la plateforme. Ceci aboutit à une solution adaptable.

Afin d'assurer l'interopérabilité de notre plateforme éducative, nous la développons à l'aide du langage de programmation JAVA. Puisque les étudiants de première année n'ont pas une connaissance du langage de développement utilisé ni des concepts orientés objet, nous adaptons notre environnement de développement pour qu'il supporte le langage de programmation C. Cette faculté est assurée par l'utilisation de l'interface native de java (*Java Native Interface, JNI*). Elle permet l'interfaçage du langage java avec d'autres langages de programmation. Ce choix permet de décomposer notre plateforme en deux parties : métier et présentation. La première est responsable de la mise en œuvre d'algorithmes de remplacement de pages. La couche présentation assure une utilisation simplifiée de la plateforme. Cette décomposition simplifie l'accessibilité à notre plateforme. En effet, l'utilisateur peut lancer l'application dans un navigateur en utilisant un script JAVA tout en laissant l'exécution au niveau du serveur d'hébergement. Aussi, l'utilisation d'une bibliothèque standard, JNI, met notre plateforme à l'abri de toute rupture technologique au niveau des langages de programmation. La réutilisabilité de notre plateforme éducative dans d'autres enseignements sera justifiée par le modèle de sa conception (voir la section §III).

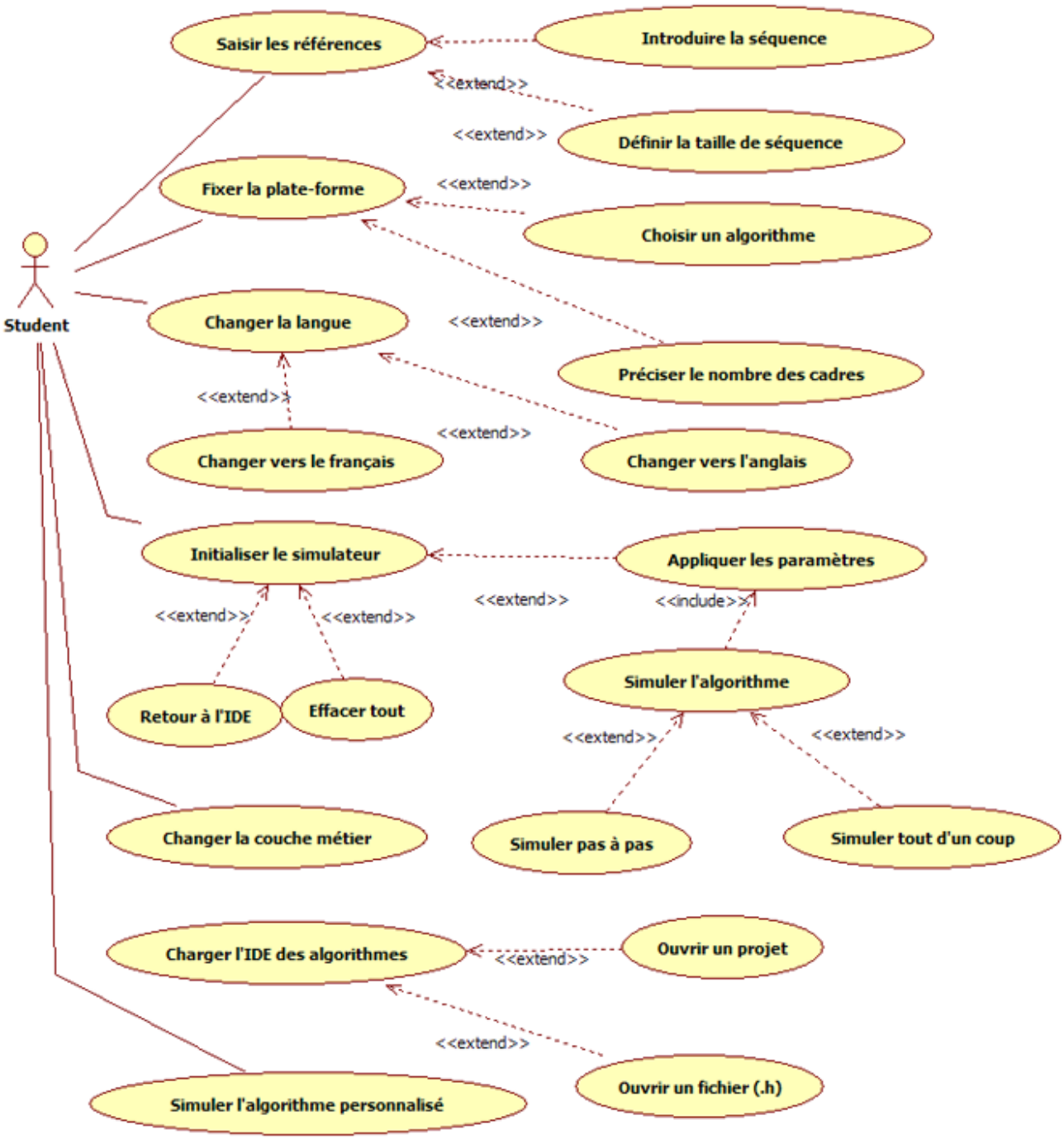
II. Spécification de la plateforme éducative

Notre plateforme éducative contient trois parties : un simulateur, un environnement de développement intégré et un installateur. Les deux premiers éléments représentent deux objets à contenu partageable. Dans la suite de cette section, nous détaillons la spécification de ces trois parties.

A. Spécification du simulateur

La *figure 1* présente la vue statique de la plateforme qui permet de simuler un algorithme graphiquement.

Figure 1 : Simuler un algorithme



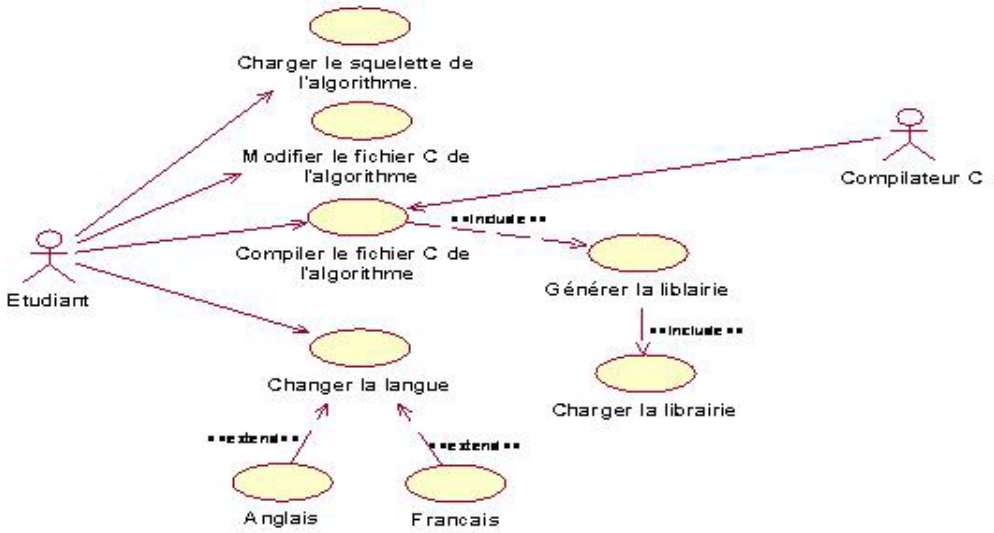
Le cas d'utilisation `Saisir les références` fait appel à deux autres cas d'utilisation : `Définir la taille de séquence` et `introduire la séquence`. La définition de taille représente la dimension du tableau de référencement à remplir. Ce dernier est affiché dynamiquement selon le choix de l'utilisateur. L'introduction de la séquence nécessite le remplissage de toutes les cases du tableau de référencement. Le cas d'utilisation `Fixer la plateforme` interagit avec les cas d'utilisation : `Préciser le nombre de cadres de page à utiliser dans le simulateur` et `choisir l'algorithme à simuler`. Le nombre des cadres de pages permet de fixer la taille de la mémoire vive. Pour choisir l'algorithme,

l'utilisateur accède à une liste déroulante qui contient quatre choix : FIFO, LRU, Optimal et WS-Clock. Le cas d'utilisation `Choisir la langue` offre deux options : changer la langue vers le français ou changer la langue vers l'anglais. Le changement de la langue est fait dynamiquement. Ainsi l'utilisateur ne doit pas choisir une langue d'utilisation dès l'installation de la plateforme. Il peut à tout moment modifier la langue des composants textuels utilisés. Le cas d'utilisation `Initialiser le simulateur` fait appel à trois cas d'utilisation : `Appliquer les paramètres`, `effacer tout` et `retour à l'environnement de développement intégré`. Les deux derniers cas d'utilisation permettent successivement la réinitialisation de tous les champs de saisie et la navigation entre les deux interfaces principales de simulation et d'implémentation. Le premier cas `Appliquer les algorithmes` est une tâche nécessaire avant de commencer la simulation. Cette dernière peut être orchestrée par l'utilisateur qui, manuellement, demande l'avancement dans le traitement du scénario ; c'est le mode pas à pas. Aussi, la simulation peut être réalisée d'une manière automatique, sans intervention humaine ; c'est le mode résultat. Le cas d'utilisation `changer la couche métier` permet à l'utilisateur de changer la librairie de contrôle du simulateur. La librairie est un programme C, c'est-à-dire, un algorithme de remplacement de page, encapsulé dans un fichier de librairie dynamique. Il suffit de charger un nouveau fichier à l'aide de l'explorateur intégré à la plateforme. Le cas d'utilisation `Charger l'IDE` permet de passer d'un environnement de visualisation graphique vers un environnement d'édition et de développement des algorithmes. Pour passer à l'IDE (i.e. Environnement de Développement Intégré), l'utilisateur doit activer le menu fichier et choisir de créer un nouveau projet ou ouvrir un fichier ".h" en édition. Le dernier cas d'utilisation réalisable par un acteur primaire est `Quitter la plateforme`. Pour activer ce cas, l'utilisateur peut faire appel au menu Fichier ou bien le bouton de fermeture de la fenêtre. Il peut l'activer aussi à partir des raccourcis clavier.

B. Spécification de l'environnement de développement

La *figure 2* ci-dessous présente le diagramme de cas d'utilisation permettant d'implémenter un algorithme.

Figure 2 : Environnement de développement intégré



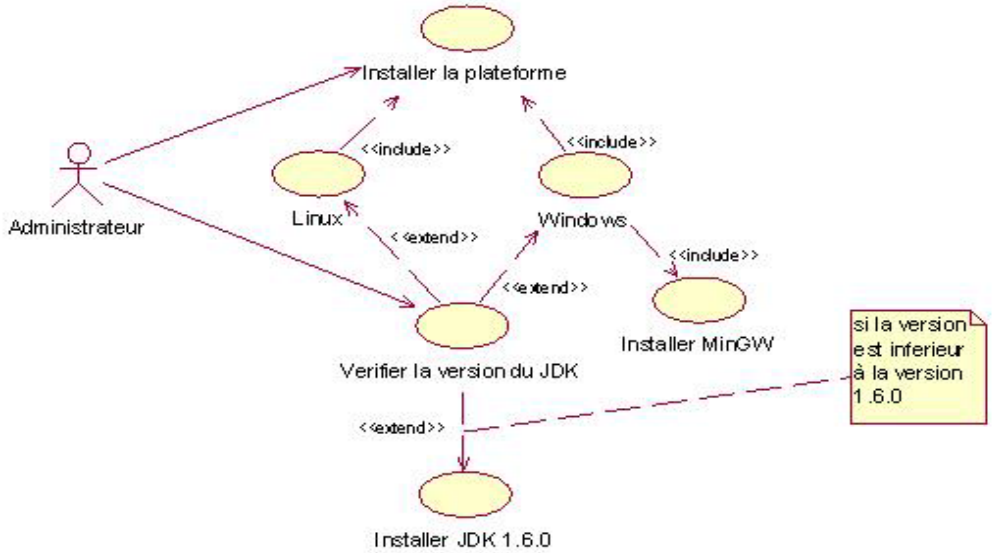
L'environnement de développement intégré est décrit à l'aide des cas d'utilisation suivants : `Charger le squelette`, `modifier le fichier C de l'algorithme`, `Compiler`

`le fichier C de l'algorithme`, `générer la librairie`, `charger la librairie` et `changer la langue`. Le premier cas d'utilisation `Charger le squelette` permet de charger le squelette. Ce dernier se compose des en-têtes des fonctions assurant le service que l'algorithme doit fournir ainsi que les variables globales nécessaires à la modélisation des pages, de la table des pages et des cadres de pages. Le deuxième cas d'utilisation `modifier le fichier C de l'algorithme` nécessite la connaissance du langage de programmation C. Dans ce cas d'utilisation, l'étudiant doit implémenter les fonctions initialement chargées dans la zone d'édition. Les fonctions à implémenter sont responsables de la détection d'un défaut de page et du calcul de la page à remplacer. Le troisième cas `Compiler le fichier C de l'algorithme` fait appel au compilateur C qui est différent en fonction du système d'exploitation utilisé. Le cas d'utilisation `générer la librairie` permet de générer un fichier objet partagé ".dll" dans le cas du système d'exploitation Windows ou bien ".so" dans le cas du système d'exploitation LINUX. Cette librairie permet de faire le lien entre le code écrit en C et l'interface JAVA qui le représente. Evidemment, le simulateur dispose de sa librairie initiale, que nous avons développée. Cependant, si l'étudiant veut changer le code des fonctions mentionnées plus haut, nous devons générer une nouvelle librairie qui correspond à son code et la charger dans le simulateur. Le chargement de la libraire se fait par le biais d'un script permettant de changer la librairie initiale du simulateur par celle nouvellement créée par l'étudiant. Comme dans le cas du simulateur, l'environnement de développement intégré peut être adapté à l'exécution pour changer de langue. Ceci est réalisé par le cas d'utilisation `changer la langue`.

C. Spécification du système d'installation

La *figure 3* ci-dessous présente le diagramme de cas d'utilisation relatif au système d'installation de la plateforme au niveau du serveur.

Figure 3 : Système d'installation de la plateforme



Le système d'installation de la plateforme est décrit à l'aide de quatre cas d'utilisation : `Installer la plateforme`, `installer MinGW`, `vérifier la version du JDK` et `installer JDK 1.6.0`. Le premier cas d'utilisation `installer la plateforme` possède deux options : Exécuter l'installateur sous WINDOWS ou bien sous LINUX. Pour faire l'installation de la plateforme, l'administrateur doit vérifier la version de la machine virtuelle installée. Dans le cas où la version est plus ancienne que 1.6.0, il doit faire l'installation de la version nécessaire. Il est

obligatoire dans le cas de l'installation sous WINDOWS d'installer la version minimale de développement et de compilation de GNU sous WINDOWS, c'est -à-dire, le compilateur MinGW. Ceci est primordial pour la compilation et la génération des librairies de dynamique (*DLL*).

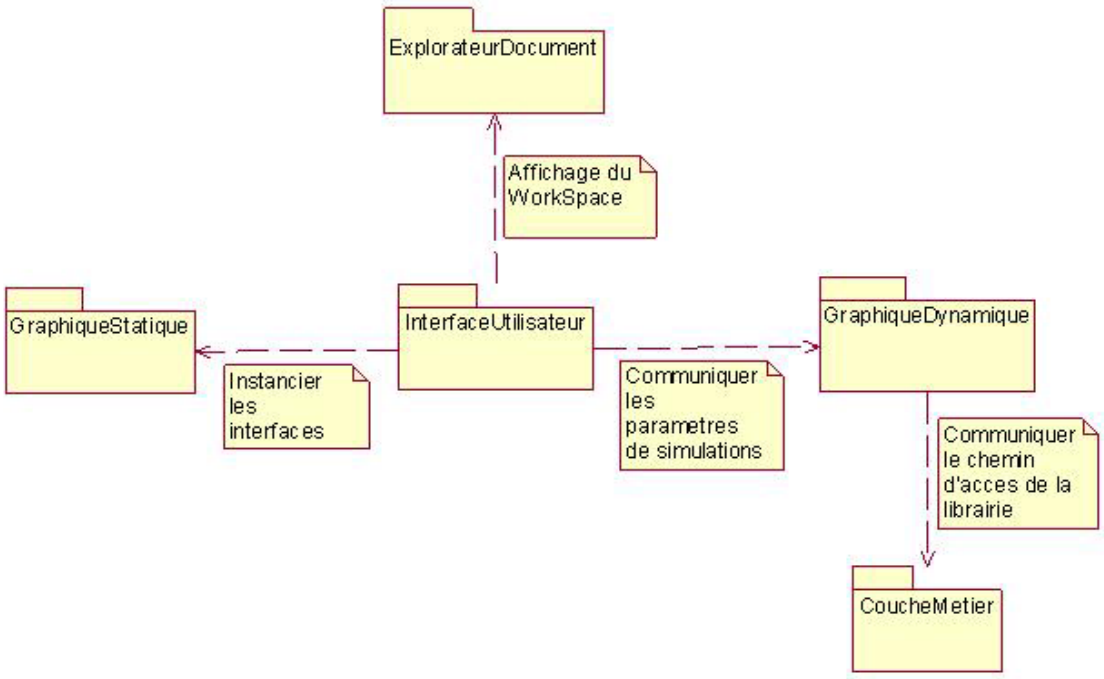
III. Conception de la plateforme éducative

La conception de notre plateforme éducative se fonde sur une décomposition fonctionnelle et architecturale des différentes parties. Ceci permet la réutilisation des composants dans d'autres plateformes.

A. Vue générale

La *figure 4* présente les différents groupes logiciels (paquets) permettant de montrer les principaux composants de notre plateforme.

Figure 4 : Paquetages de la plateforme éducative



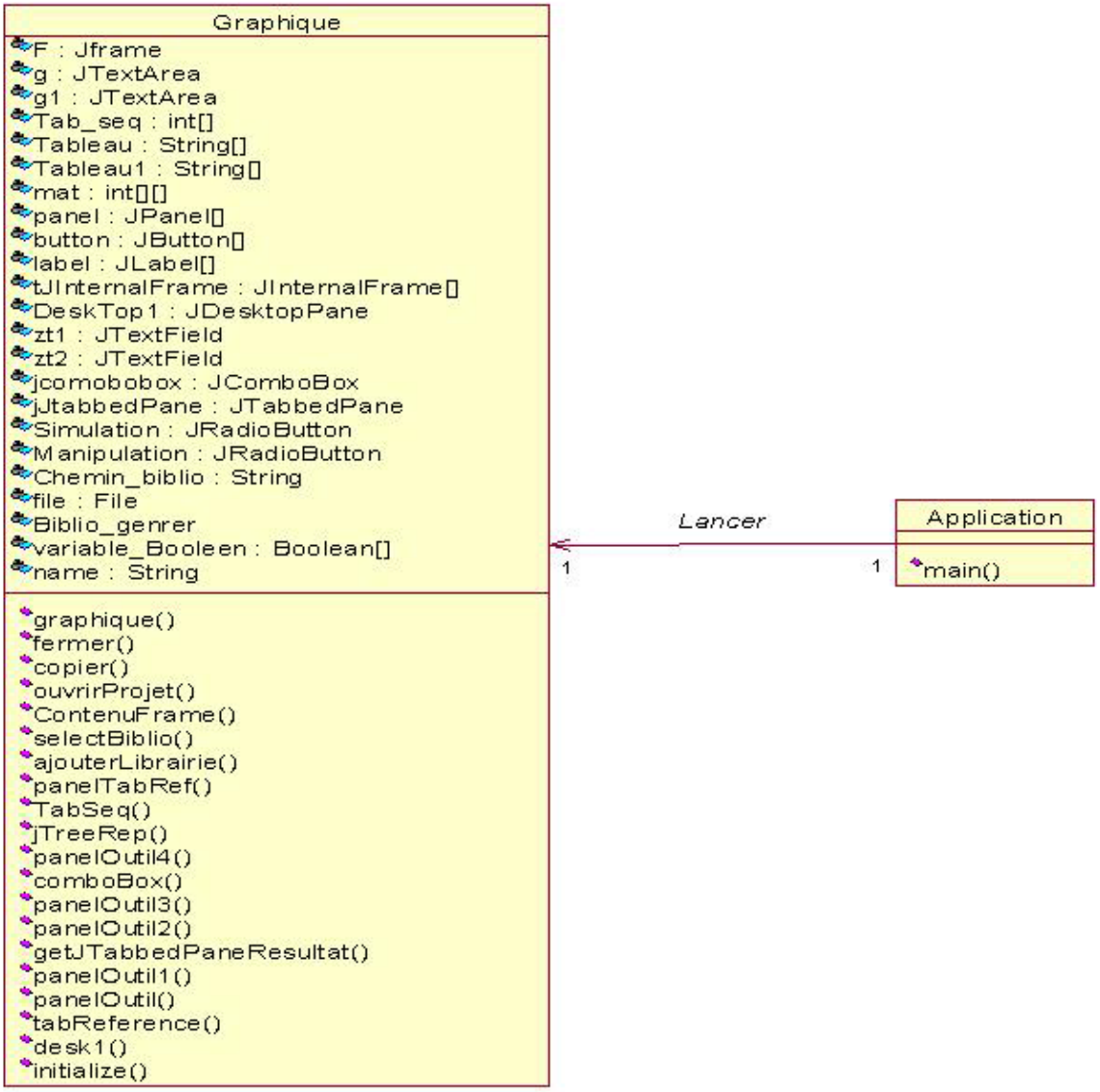
Les cinq paquetages utilisés sont : `InterfaceUtilisateur`, `ExplorateurDocument`, `GraphiqueStatique`, `GraphiqueDynamique` et `CoucheMetier`. Nous remarquons que tous les paquetages sont en relation directe avec `InterfaceUtilisateur`. Ce paquetage permet la génération des éléments graphiques responsables de la simulation et de l'implémentation d'un nouvel algorithme de gestion de mémoire. Il entre en interaction avec tous les paquetages si l'on excepte le paquetage `CoucheMetier`. Le paquetage `ExplorateurDocument` permet d'accéder aux dossiers de travail. Il assure deux fonctionnalités : `enregistrement de travail` et `ouverture de travail`. Le deuxième paquetage `Graphique statique` offre les composants graphiques non dynamiques à la simulation. Nous avons deux éléments graphiques statiques : la barre de menu et les conteneurs des fenêtres. Le troisième paquetage

`GraphiqueDynamique` contient tous les éléments qui changent d'état lors de la modification des paramètres de simulation. Ce paquetage contient : les cadres de pages de la mémoire, le tableau de séquence et l'interface d'interaction entre la couche présentation et la couche métier. Le dernier paquetage `CoucheMetier` contient les classes responsables de la communication entre la plateforme implémentée dans le langage de programmation JAVA et la librairie de contrôle implémentée en C.

B. Interface d'utilisateurs

La *figure 5* montre les éléments de conception statique liés à l'interface de l'utilisateur. Cette décomposition permet la réutilisation de l'interface utilisateur dans d'autres solutions.

Figure 5 : Classes de l'interface d'utilisateurs

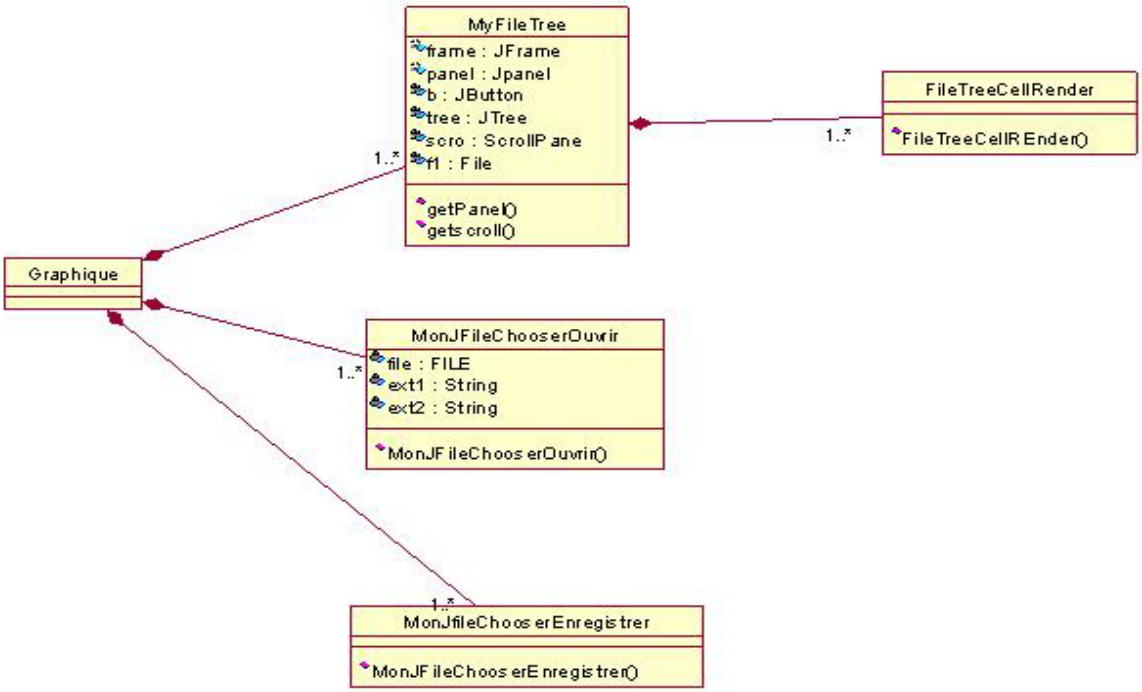


Les classes `Application` et `Graphique` se trouvent dans le paquetage `InterfaceUtilisateur`. La première classe permet de lancer la plateforme. Elle représente le processus d'initialisation. Elle contient une seule méthode `main()` qui permet d'instancier la plateforme et de lui fournir les paramètres nécessaires pour ajuster l'affichage en fonction du système d'exploitation sous-jacent.

C. Gestion des documents

La *figure 6* présente les composants nécessaires à la manipulation des fichiers liés à l'implémentation des concepts des systèmes d'exploitation.

Figure 6 : Classes de l'explorateur du document

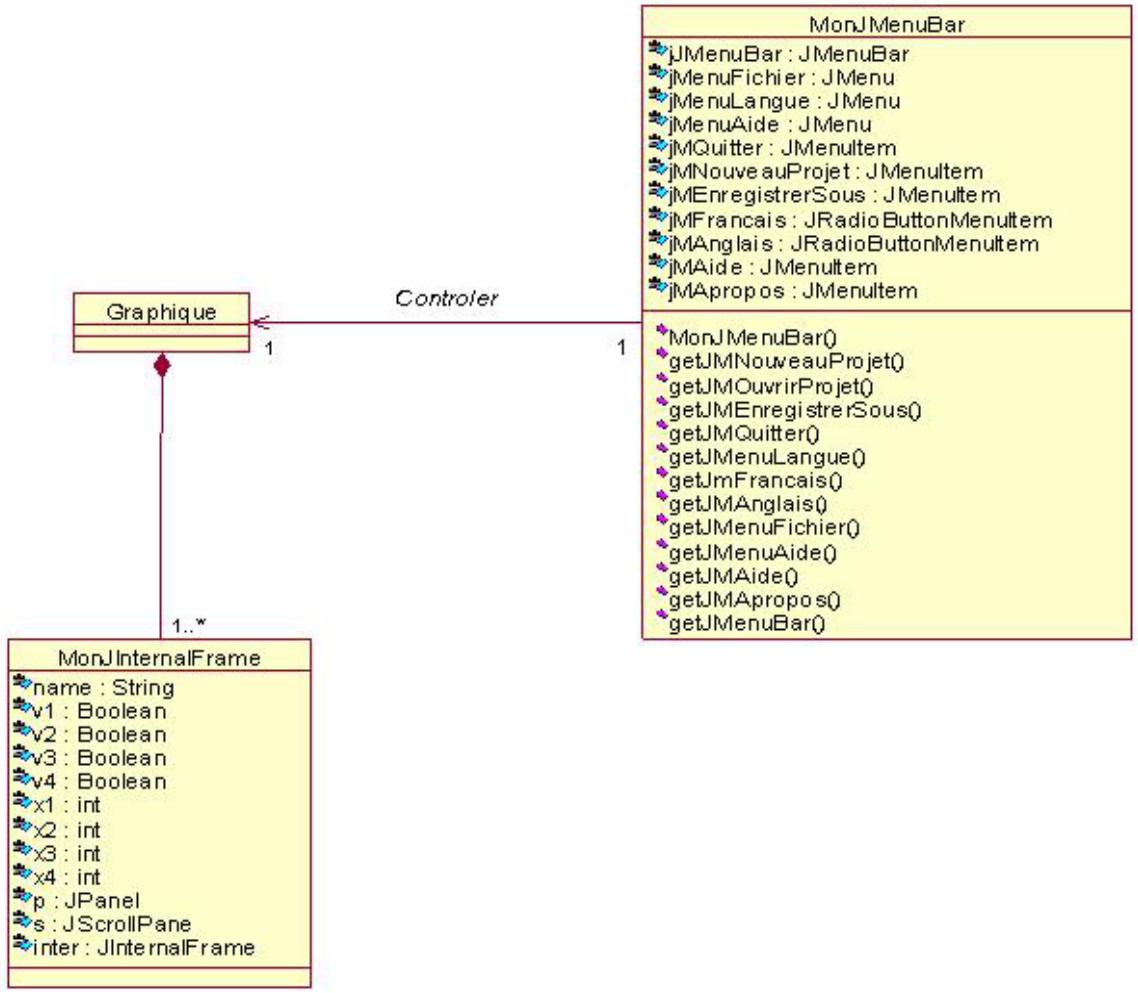


Le paquetage `ExplorateurDocument` contient cinq classes : `Graphique`, `MyFileTree`, `FileTreeCellrender`, `MonJFileChooseOuvrir`, `MonJFileChooserEnregistrer`. Les deux classes `FileTreeCellRender` et `MyFileTree` permettent d'obtenir une vue générale sur l'ensemble des fichiers à manipuler par l'utilisateur. Elles notifient chaque création ou destruction d'un fichier utilisateur. Les deux classes `MonJFileChooseOuvrir` et `MonJFileChooserEnregistrer` permettent respectivement l'ouverture d'un fichier de librairie à éditer par l'utilisateur et son enregistrement à fin de conserver son contenu pour de futures utilisations par la plateforme.

D. Gestion des éléments graphiques statiques

La *figure 7* introduit les éléments nécessaires à l'affichage des composants statiques de la plateforme éducative.

Figure 7 : Classes de l'interface statique

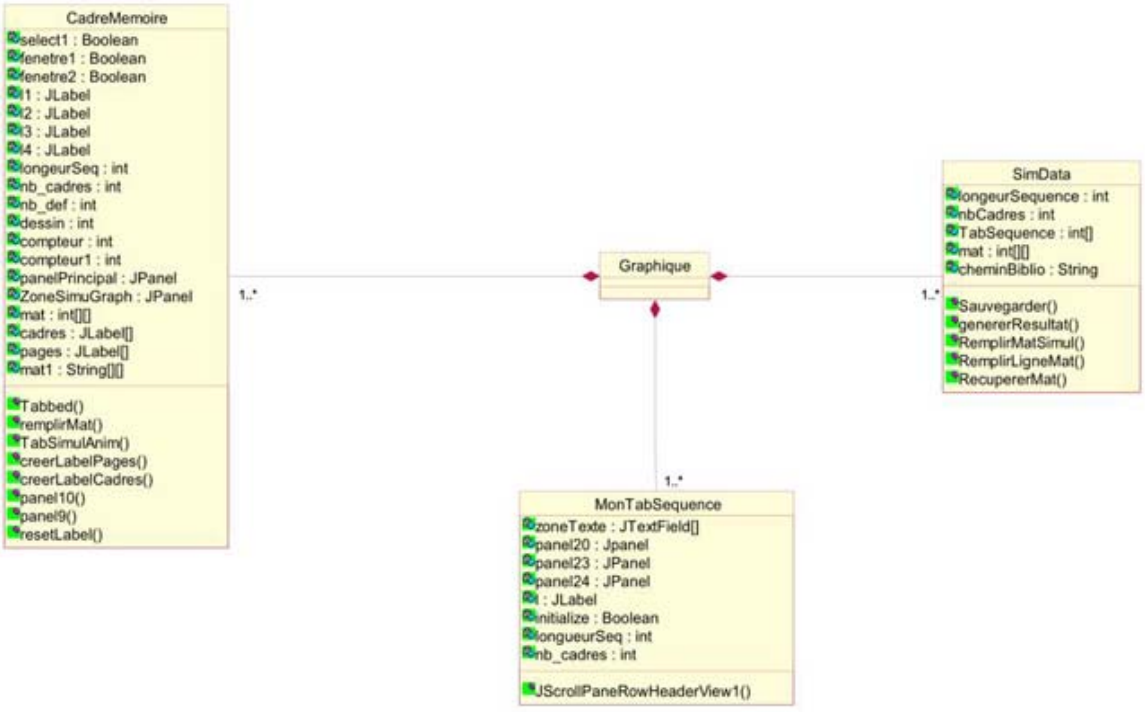


Le diagramme de classe du paquetage `GraphiqueStatique` contient trois classes : `Graphique`, `JmenuBar` et `MonJInternalFrame`. La première classe a été déjà introduite dans le début de cette partie. La deuxième classe permet d'avoir la barre de menu de l'interface graphique de la plateforme. Elle est composée de trois menus : `Menu Fichier`, `Menu Langue` et `Menu aides`. La troisième classe permet de positionner un objet graphique en précisant une largeur, une hauteur et une position d'insertion. Un objet de type `JInternalFrame` se comporte comme une fenêtre, mais celle-ci ne peut s'afficher que dans une fenêtre principale. La classe `Graphique` est une fenêtre principale qui contient plusieurs instances de la classe `MonJInternalFrame`.

E. Gestion des éléments graphiques dynamiques

La *figure 8* introduit les classes nécessaires à la prise en compte de l'évolution de la plateforme et qui assurent leurs affichages.

Figure 8 : Classes de l'interface dynamique

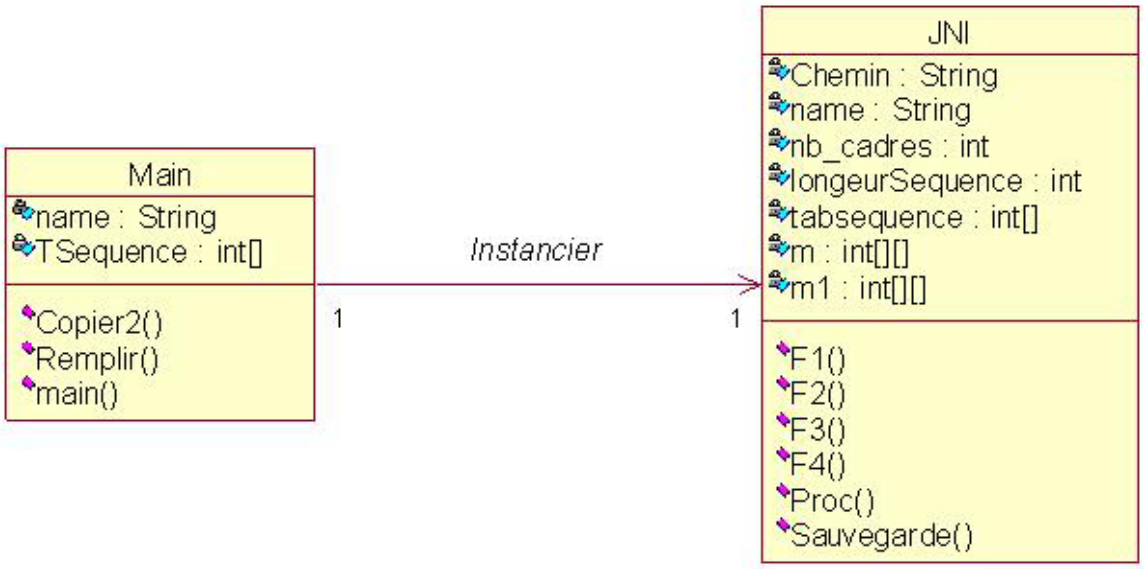


Le paquetage `GraphiqueDynamique` contient quatre classes : `Graphique`, `MontabSequence`, `SimDate` et `CadreMemoire`. Outre la classe principale `Graphique`, les trois autres classes représentent les éléments qui changent leurs états lors de l'utilisation du logiciel. Ainsi leurs diagrammes d'état/transition présente un changement dépendant à la fois de l'intervention de l'utilisateur et de la couche métier de la plateforme.

F. Classes métier

La *figure 9* présente les classes utilisées pour la prise en charge de la modification du cœur du simulateur par l'étudiant.

Figure 9 : Classes de la couche métier



Les classes du paquetage `CoucheMetier` sont : `Main` et `JNI`. La première classe permet d'avoir une instance d'un objet JNI. Ce dernier contient la définition des méthodes natives de la librairie à appeler par le programme JAVA. Ces classes retournent le résultat des algorithmes de gestion de la mémoire.

IV. Conclusion

Cette plateforme éducative a été réalisée au sein de l'institut supérieur d'informatique et de mathématiques de Monastir dans le but de développer une application d'apprentissage. Elle facilite l'assimilation des algorithmes de remplacement de pages introduits dans la partie relative à la gestion de la mémoire du cours de systèmes d'exploitation. La plateforme offre un environnement de développement intégré dédié au langage de programmation C. Cet environnement de développement intégré au simulateur donne l'occasion aux étudiants d'exécuter leurs propres algorithmes de gestion de mémoire. Par conséquent, ces deux parties du simulateur que nous avons proposé forment deux objets d'apprentissages à contenus partageables.

Ce papier se développait en trois parties : Normalisation et simulateurs, spécification et conception de la plateforme éducative. Dans la première partie, nous avons présenté les normes LOM et SCORM, ainsi que, les différents simulateurs utilisés dans l'enseignement des concepts liés aux systèmes d'exploitation. Nous avons aussi développé les avantages et les inconvénients de chacun par rapport aux propriétés préconisées par le standard SCORM. Nous avons spécifié les différentes tâches qu'un étudiant pourrait réaliser. Ceci est essentiel, non seulement pour mieux comprendre les concepts vus dans le cours et les travaux dirigés, mais aussi pour les mettre en pratique. Aussi, nous avons introduit la spécification du système d'installation de la plateforme éducative dans la deuxième partie de ce papier. La troisième partie était consacrée à la conception de la plateforme. UML (*Unified Modelling language*) a été utilisé comme langage de modélisation de notre projet. Nous avons commencé par la vue d'ensemble de notre plateforme qui a été introduite par un diagramme de paquetage. Cette décomposition est nécessaire pour avoir une solution réutilisable. Nous avons détaillé les classes qui composent les différents paquetages. Ces classes traitent les aspects graphiques statiques et dynamiques de manière séparée. Aussi, la gestion des fichiers ainsi que l'interface des utilisateurs étaient séparées pour une meilleure décomposition. Enfin, nous avons présenté les classes de la couche métier qui permettent l'adaptabilité de notre plateforme.

Nous pouvions envisager de rajouter de nouvelles fonctionnalités pour évaluer systématiquement le travail des étudiants. Il s'agit de juger le travail effectué par l'étudiant en lui attribuant une note préalable de Travaux pratiques. Cette évaluation serait possible par l'utilisation de deux éléments : le fichier des traces d'exécution et le fichier d'implémentation. Les premiers permettent de comparer le scénario d'exécution réalisé par l'étudiant à celui fourni par le simulateur. L'analyse du fichier d'implémentation permet d'évaluer un travail qui n'a pas abouti jusqu'à l'exécution. Dans ce dernier cas, nous devrions développer un système d'analyse sémantique du contenu du fichier d'implémentation. Une fonctionnalité de supervision par l'enseignant de TP pourrait être ajoutée. Cela nécessiterait une programmation répartie dans laquelle la partie client reste la même par contre la partie serveur, côté enseignant, doit être entièrement créée. Elle permettra de visualiser ce que l'étudiant a réussi à faire en temps réel.

Références bibliographiques

Bertin, G. (2011). Sharable Content Object Reference Metadata (SCORM). Récupéré en mai 2011 du site : http://www.enssib.fr/bibliotheque-numerique/document-1232

Bérubé, S. (2011). Simulateur de défauts de pages. Récupéré en mai 2011 du site http://www.cours.polymtl.ca/inf2610/simulateurs/Simdef/index.html.

Lawton, K. (2011). BOCHS. Récupéré en Mai 2011 du site http://bochs.sourceforge.net/

Ontoko, R., Reeder, A. & Tannenbaum A. S. (2011). Modern Operating Systems Simulator (MOSS). Récupéré en mai 2011 du site : http://www.ontko.com/moss/

Pernin, J.-P. (2011). LOM, SCORM et IMS-Learning Design : ressources, activités et scenarios. Récupéré en mai 2011 du site : http://www.enssib.fr/bibliotheque-numerique/document-1232