Un tempo de Reel sur un rythme de Java

Jean-Louis Dillenseger, Christine Toumoulin

Un tempo de Reel sur un rythme de Java*

Jean-Louis Dillenseger, Christine Toumoulin
jean-louis.dillenseger@univ-rennes1.fr
Adresses : IUT Rennes, dpt GEII, rue du Clos Courtel, Rennes

RESUME : Ce papier concerne l'enseignement d'un module sur les systèmes multitâches et systèmes temps réel, délivré à des étudiants de deuxième année de DUT Génie Électrique et Informatique Industrielle. Dans un premier temps, ce papier justifie l'utilisation d'un langage haut niveau comme Java pour illustrer les concepts d'un système temps réel, même si ce langage et le système d'exploitation ne sont pas strictement « temps réel ». Ce papier décrit ensuite les notions de cours enseignées. Ces notions sont illustrées lors de séances de Travaux pratiques, soit sur PC sous Windows, soit sur un robot Mindstorm de chez LEGO programmé en Java à l'aide l'API LeJos.

Mots clés : Systèmes temps réel, Java, Robot Mindstorm

Les titres auxquels vous avez échappé :
O reale tempus, O mores
Cavommavent avillavustraver lave tavemps ravéel avavavec davu Javavava et daves ravobavots LAVEGAVO

1 INTRODUCTION

Le programme pédagogique national (PPN) des IUT Spécialité "Génie Électrique et Informatique Industrielle" (GEII) [1] propose plusieurs modules complémentaires qui doivent *permettre à l'étudiant de compléter sa formation :*

- *Soit par approfondissement d'un domaine technologique (AT) dans le prolongement de la formation reçue dans le cœur de compétences,*
- *Soit dans un objectif de renforcement de ses compétences professionnelles (RCP),*
- *Soit à titre d'ouverture scientifique (OS).*

Notre département a choisi d'intégrer ces modules complémentaires dans le cursus des étudiants de deuxième année sous la forme de "parcours". Le parcours consiste à étendre le cœur de compétences par une série bien spécifique de modules complémentaires obligatoires afin d'approfondir un domaine technologique particulier. Parmi quatre parcours, nous proposons un parcours "Automatique, Informatique et Réseaux Locaux Industriels" (AIRLI). Ce parcours intègre, entre autres, deux modules complémentaires orientés vers l'informatique et l'informatique industrielle : le module MCII1 sur la Programmation Orientée Objet (POO) et plus récemment le module MCII2 sur les Systèmes multitâches, systèmes temps réel.
Le module MCII1 sur la POO est proposé depuis 2004. Le choix a été, dès le départ, de proposer un cours basé sur un langage permettant d'illustrer le plus simplement les concepts des langages orientés objet et les démarche de conception orientée objet, notamment UML. Le langage utilisé est Java sous l'environnement de programmation BlueJ sur des machines sous Windows. Les étudiants suivent 6h de cours magistraux et 7 séances de 3h de TD/TP. Les 5 premières séances portent spécifiquement sur les concepts de la programmation objet (objets, classes, héritage, surcharge, classe abstraite, interface), les deux dernières séances sont plus spécifique à l'interfaçage Homme Machine proposé par Java (création de fenêtres et d'actionneurs, réponses aux actions à l'aide de la gestion d'évènements –Listener, Event). Le module est évalué lors d'une huitième séance de TP de 2h, à l'aide d'un mini projet de complexité progressive (des notions les plus simples au début vers les plus complexes à la fin).
L'introduction du module MCII2 sur les systèmes multitâches et systèmes temps réel a été plus tardive dans notre département. Traditionnellement dans les départements GEII, la notion de temps réel est plutôt abordée par le biais de solutions matérielles sur la base de microcontrôleurs et des langages relativement bas-niveau pour rester le plus proche possible du matériel. Or ces langages bas niveau même associés à des librairies spécialisées en temps réel nous semblaient trop complexes et trop matériel-dépendants pour illustrer les notions générales du temps réels. Plus récemment sont apparus quelques ouvrages traitant des concepts temps réels à l'aide de langages de plus haut niveau comme ADA ou Java [2-3]. Nous avons donc choisi d'illustrer les notions de temps réel à l'aide du langage Java classique avec comme cibles applicatives, soit un PC sous Windows, soit un robot Mindstorm de chez LEGO sur lequel nous avons implémenté un firmware et une API Java (LeJos). La suite du papier portera, dans un premier temps, sur la justification de notre approche basée sur l'utilisation de Java, puis sur la description des différents concepts étudiés durant le cours.

2 TEMPS REEL ET JAVA

2.1 Notions du temps réel

Les objectifs annoncés par le PPN sont la *compréhension et la maîtrise de l'organisation fonctionnelle d'une application en tâches parallèles coopérantes et la connaissance des principaux mécanismes de coopération entre tâches.* Le PPN précise également que *l'objectif pédagogique est de montrer que, moyennant la compréhension de la philosophie « multitâches », le travail du programmeur est grandement simplifié, il n'a plus à traiter qu'un problème à la fois, il voit le système*

* Une explication du titre est donnée en ligne sur le site j3eA.org

Article publié par EDP Sciences et disponible sur le site http://www.j3ea.org ou http://dx.doi.org/10.1051/j3ea/2013013

comme une machine virtuelle qui est entièrement disponible pour le programme qu'il est en train d'écrire. La mise en œuvre serait souhaitable par l'*utilisation d'un environnement de développement croisé en langage évolué (en principe C ou C++) avec les librairies d'un système temps réel* et par des *applications sur un exemple de robot simple*. La liste de mots clés suivant est également donnée : *tâche, thread, parallélisme, exceptions, ordonnancement, réentrance, sémaphores, partage de ressources.*

Pour être un peu plus complet que le PPN, nous sommes basé sur la définition de systèmes temps réels donnée par Alabau et Dechaize [4] : *une application temps réel constitue un système de traitement de l'information ayant pour but de commander un environnement imposé, en respectant des contraintes de temps et de débit (temps de réponse à un stimulus, taux de perte d'information toléré par entrée) qui sont imposées à ses interfaces avec cet environnement*. Si nous nous référons à cette définition, plusieurs éléments vont nous aider à caractériser un système temps réel. Ces éléments, même s'ils ne sont pas partagés par tous les systèmes, sont à prendre en compte pour la définition et la construction d'une application :

- La gestion des processus ou de tâches que doit effectuer le système. Rares sont les systèmes qui ne sont dédiés qu'à une tâche. Généralement ils doivent effectuer plusieurs tâches en parallèle. Le plus simple est alors de décomposer le système en tâches indépendantes (*systèmes multitâches*) qui fonctionnent en parallèle. Les tâches doivent communiquer entre-elles et doivent accéder à des *ressources communes*.
- La *gestion du temps*. Elle intervient à deux niveaux : 1) à un niveau applicatif où les tâches peuvent être périodiques (échantillonnage de signaux) ou répondre à des sollicitations externes (fonctionnement apériodique par interruptions par exemple) ; 2) à un niveau plus bas niveau un seul processeur (ou un faible nombre de processeurs) doit assurer par un *ordonnancement* temporel les différentes tâches afin de proposer un pseudo-parallélisme. Dans tous les cas, le système temps réel est censé garantir un contrôle temporel strict des actions quelles que soient les conditions.
- La sûreté de fonctionnement et fiabilité. Pour les systèmes temps réel (pouvant être autonomes ou embarqués), la sûreté de fonctionnement peut être un critère essentiel. Le concepteur d'un système temps réel, outre les erreurs inhérentes au développement d'un logiciel, devra essayer de renforcer la sécurité de fonctionnement de son système en anticipant les situations et les fonctionnements anormaux (délais trop important, division par zéro, etc.). La gestion de ces comportements non normaux par le système lui même s'appelle la gestion des *exceptions.*

2.2 Éléments de choix d'une solution logicielle

Ces éléments (et donc les langages informatiques pour les mettre en œuvre) peuvent être vus selon plusieurs niveaux. Classiquement, comme il a été dit dans l'introduction, le temps réel était plutôt abordé par le coté bas niveau, proche du matériel (microcontrôleur) avec des langages relativement bas-niveau (noyaux, gestion d'interruptions,…). Ce point de vue est (ou était) sans doute indispensable pour une gestion stricte du temps. Toutefois, ce type de solution à forte dépendance au matériel et les programmes associés (assembleur, voire C et C++) sont peu lisibles et peu flexibles. De surcroit certains des éléments (multitâches, gestion des exceptions) sont assez compliqués à mettre en œuvre et nécessitent l'emploi de librairies spécialisées. Actuellement, certains langages de haut niveau (Java, C#, Ada, …) propose, de manière native, de décrire simplement des processus complexes. Par exemple, le langage Java [5] a intégré, dès sa conception, des mécanismes de programmation distribuée (multithreading) et de sécurité (gestion des exceptions). De même, du fait de l'interfaçage graphique, des mécanismes de gestion d'évènements sont également disponibles. En regardant de plus près les différents paquetages de Java, "`java.util.concurrent`" propose une série de classes utilisables pour la gestion de ressources communes. De certaines classes natives, comme `Timer` et `TimerTask` permettent l'ordonnancement de tâches (tâche périodique où tâche lancée au bout d'un certain délai). Java permet donc de programmer, de manière très simple, la plupart des concepts haut niveau liés au temps réel. Par contre Javapas ne permet pas de faire du temps réel stricto sensu. En effet, la portabilité de Java repose sur le fonctionnement sur une machine virtuelle spécifique à chaque plate-forme ou couple machine/système d'exploitation. Le comportement temporel d'un programme sous Java sera donc différent en fonction de la machine cible. En outre, la gestion du temps n'est pas précise (la résolution temporelle est la milliseconde, les timers ne respectent pas strictement les échéances,…). De même la machine virtuelle ne propose aucun mécanisme de gestion prédictible de l'ordonnancement des tâches au niveau processeur et elle possède des mécanismes internes cachés non prédictibles (gestion de la mémoire, …). Il existe toutefois des solutions Java temps réel. Par exemple, Real-Time Specification for Java (RTSJ) est une proposition pour étendre les machines virtuelles Java avec des fonctionnalités temps réel. Certains concepteurs de logiciels[1] proposent des environnements de programmation Java temps réel (classes et machines virtuelles temps réel) qui peuvent être associés à des systèmes d'exploitation temps réels. Ces solutions comportent de nouvelles classes spécialisées dans l'ordonnancement des threads, la gestion mémoire, la synchronisation et le partage de ressources,

[1] PERC Ultra (Aonix), Java SE Real-Time (Sun), JamaicaVM (Aicas), Jrate (open source), …

la gestion des événements asynchrones, la gestion du temps et de l'horloge, …

2.3 Solution choisie

Nous considérons que l'objectif du module sur les systèmes multitâches et systèmes temps réel est avant tout une ouverture scientifique pour le public visé (étudiant de GEII). Ce public n'ayant pas pour vocation d'être ingénieur informaticien, il nous a semblé plus utile d'expliquer et d'illustrer globalement les différents concepts liés à un système temps réel (réactivité, multitâche, notion de gestion du temps – tâche ponctuelle ou périodique, gestion de la date – et sureté de fonctionnement) plutôt que d'entrer dans des détails liés à un système temps réel spécifique (gestion d'un OS et d'un noyau temps réel, gestion temps réel strict des tâches,…). Comme il a été rappelé dans l'introduction, le Java classique possède de manière native des classes qui répondent à la plupart des notions liées au temps réel. Nous choisissons donc la solution logicielle déjà utilisée dans le module MCII1 sur la POO, c'est-à-dire du Java classique, sous l'environnement de programmation BlueJ sur des machines sous Windows. Il est clair que Windows n'est absolument pas un système d'exploitation temps réel (impossible de descendre sous la milliseconde, aucune possibilité de gérer la machine virtuelle Java), mais nous ne souhaitons pas atteindre ce degré de précision. Dans notre cas, les avantages de Java sont multiples : langage et environnement connus et maitrisés, multitâche natif, langage de haut niveau, classes gérant les ressources communes, les exceptions, des timer et des schedulers, ….

Comme le suggère le PPN, une cible applicative présentant un système réactif permet de mieux illustrer les concepts des systèmes temps réel. Parmi les robots pédagogiques, l'ensemble LEGO Mindstorms NXT[2] a été l'un des plus utilisés. Afin de programmer les LEGO NXT sous Java, nous utilisons LeJos NXJ[3], un environnement de programmation gratuit Java des LEGOs NXT et qui contient une machine virtuelle portée sur la brique, une interface de programmation (Application Programming Interface ou API) permettant de lire les données des capteurs et de piloter les actionneurs ainsi que des outils sur PC permettant de communiquer et de flasher les briques NXT. De plus, l'API Lejos et les outils de communications sous PC peuvent, être intégrés dans l'environnement de programmation BlueJ[4].

Nous avons décidé de garder la même configuration de robot LEGO pour toute la série de Travaux Pratiques. C'est le robot mobile classique (voir fig 1) avec, comme actionneurs : 2 roues motrices et un gyrophare ; et en capteurs : un capteur photosensible qui mesure l'intensité lumineuse captée ou réfléchie par un objet (pour du suivi de ligne), un capteur de son (micro) et un capteur de distance basé sur les ultrasons. Il dispose également de 4 boutons sur sa face avant et d'un écran LCD que nous utiliserons pour écrire du texte sur 7 lignes de 15 caractères.

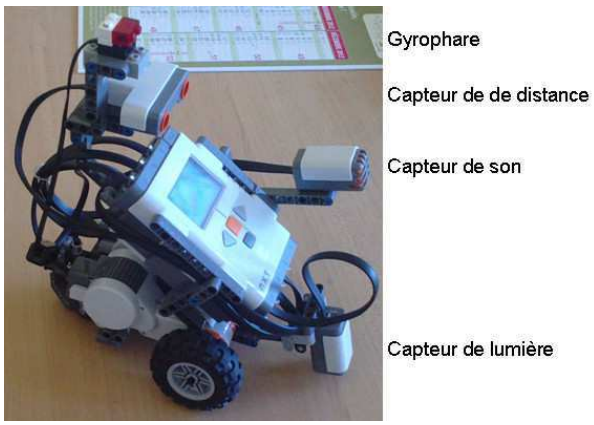

fig 1 : Robot mobile LEGO Mindstorms™ NXT.

2.4 Déroulement de l'enseignement

L'enseignement est décomposé en 6h de cours magistraux et 7 séances de 3h de TP. Le public se compose d'un groupe de 24 étudiants, tous ayant suivi le module de Programmation Orientée Objet basé sur le langage Java.

2.4.1 Cours magistral

L'idée générale du cours magistral est d'aborder les différentes notions du temps réel du point de vue le plus généraliste possible et ensuite d'illustrer certains des points à l'aide de Java.

Ainsi, la première partie du cours généraliste comporte la définition du temps réel, une classification des systèmes temps réel (temps réel dur, ferme ou mou) et une description des éléments caractéristiques d'un système temps réel (gestion du temps, décompositions en sous-systèmes concurrents, sûreté de fonctionnement et niveau de conceptualisation du système).

Cette partie généraliste est suivie par une réflexion avec les étudiants sur le choix d'une cible et d'un langage de programmation. Cette partie intègre la justification de notre choix d'utiliser le Java classique sur des cibles non temps réel pour l'illustration de certains concepts temps réel. Un petit complément Java sur les machines virtuelles et sur les variables et méthodes statiques leur est également apporté.

La dernière partie du cours est plus axé sur l'illustration de certains éléments caractéristiques d'un système temps réel à l'aide de Java. Cette partie traite chronologiquement :

- De la gestion des erreurs d'exécution à l'aide des mécanismes de captation et de traitement d'exception de Java.
- Des systèmes multitâches et de la programmation concurrente à l'aide des threads. Nous introduisons dans ce chapitre les notions de pseudo parallélisme et donc de noyau de système d'exploitation. Nous faisons également un parallèle entre le schéma général d'exécution d'une tâche et celui d'un thread

[2] http://mindstorms.lego.com/en-us/Default.aspx
[3] http://lejos.sourceforge.net
[4] http://www.bluej.org/

sous Java. Nous présentons la classe `Thread` et l'interface `Runnable` utilisée dans les mécanismes multitâches. Les caractéristiques de `Thread` propres à un système temps réel (priorité, mécanismes de synchronisation, de contrôle ou de communication entre threads,…) sont détaillées.

- Du partage de ressources et donc de l'exclusion mutuelle. Certains mécanismes d'attente active et passive sont décrits par leur principe (algorithme de Peterson, verrou, sémaphores). Par contre, il est bien précisé que Java fournit des classes qui gèrent directement ces aspects (`Lock` et `Semaphore`). De même la notion de moniteurs (et de section indivisible ou atomique) est également décrite. En effet, Java permet de gérer de manière très simple les ressources communes en rendant une section ou une méthode indivisible à l'aide du mot clé `synchronized`.
- De l'ordonnancement et de gestion du temps. Nous ne parlons pas ici d'ordonnancement du système d'exploitation (lié au noyau) jugé trop complexe ou trop spécifique au système. Cette partie aborde plutôt les notions d'appréhension et de mesure du temps (date, délais), les traitements périodiques par délais ou par minuterie (classes `Timer` et `Timertask` en Java) et un petit rappel de traitements asynchrones (notion `Listener` et d'`Event` en Java).

2.4.2 Travaux Pratiques

L'illustration du cours précédent se fait à l'aide de sept séances de Travaux Pratiques de trois heures chacune. Le taux d'encadrement est de douze étudiants pour un enseignant. Il y a un PC par étudiant. Par contre nous n'avons que deux robots Mindstorms par groupe de douze. D'un point de vue chronologique, le contenu des Travaux Pratiques est bien synchronisé avec les notions vues en cours. Il est à noter que le module sur les systèmes temps réel suit de très près le module sur la POO. Les étudiants ont donc encore en mémoire le langage Java et l'outil de développement.

Une séance de Travaux Pratiques commence habituellement par un rappel du cours sur les notions qui seront vues lors de la séance. L'enseignant discute également avec les étudiants de la façon dont la notion générale spécifique au temps réel a été adaptée en Java. Par contre, l'enseignant peut rester assez vague sur la syntaxe précise, le but avoué est de forcer les étudiants à utiliser la documentation des API Java et LeJos[5].

La chronologie des sept séances, est la suivante :

1. Méthodes et variables `static` - prise en main du robot - Gestion d'évènements.
   Après quelque petits exercice sur les variables et les méthodes `static` de Java, l'étudiant doit prendre en main l'API et le flux de programmation du robot Mindstorms. Pour cela il lui est demandé de piloter le robot à l'aide des boutons de la face avant (par exemple, un appui sur le bouton droit fait tourner le robot à droite, ...) et d'afficher sur l'écran LCD les valeurs lues par les différents capteurs.
   La dernière partie de la séance porte sur la gestion d'évènements. L'objectif est de lancer et d'arrêter le robot après un clap sonore. L'API du robot possède un mécanisme d'écouteur des capteurs (`SensorPortListener`) qui permet de lancer une méthode dès que la valeur mesurée par les capteurs écoutés a changé. Ce mécanisme est très proche de la gestion des évènements de l'interfaçage Homme Machine proposé par Java et vu par les étudiants lors du module sur la POO..
2. Exceptions.
   Cette séance étudie les mécanismes de gestion des erreurs d'exécution par un programme Java. Un premier sujet porte sur un calcul de moyenne. Les étudiants doivent imaginer les mécanismes de captation et de gestion des erreurs de frappe (symbole à la place d'un nombre, valeur aberrante). Les différents mécanismes proposé par Java : captation (`try catch`), remontée (`throw` et `throws`) et création de ces propres exceptions sont étudiés.
   Ces mécanismes sont également utilisés lors de la réalisation d'une pile FILO avec des exceptions dues à une pile vide ou trop pleine.
3. Programmation concurrente - threads Java.
   Dans un premier temps, les trois mécanismes permettant de faire des threads sous Java (classe `Thread` et 2 mécanismes d'imbrication de l'interface `Runnable` et la classe `Thread`) sont explorés sur une application où 3 tâches en parallèle incrémentent une variable interne. L'affichage régulier de ces variables permet de bien visualiser le parallélisme des tâches mais également le comportement temporel entre tâches. Plusieurs exécutions successives de cette application montrent le coté imprédictible du séquencement de threads Java sous Windows (et donc les limites du temps réel du couple Java/Windows). Cette application permet également de mettre en avant l'effet des priorités que l'on peut attribuer aux threads.
   Finalement, un petit système multitâche est programmé dans le robot Mindstorm. En parallèle d'une première tâche liée aux moteurs qui permet de piloter un petit parcours du robot (avance pendant 5s, repos pendant 5s, recul pendant 5s et arrêt), un deuxième thread est affecté à la supervision du robot (affichage de la distance parcoure par le robot et gestion du bouton arrêt) et un troisième thread contrôle le gyrophare (clignotement à 1 Hz). Il est à noter que ces trois tâches incluent des notions de gestion de temps périodiques par la mise au repos des threads durant un temps défini (méthode `sleep(duree)` de la classe `Thread`).
4. Synchronisation et communication entre threads.
   Plusieurs techniques de synchronisation et de communications entre tâches sont explorées durant cette séance :

[5] http://docs.oracle.com/javase/6/docs/api, http://lejos.sourceforge.net/nxt/nxj/api/index.html

Le mécanisme « fork/join » est étudié dans une petite application sur ordinateur où une organisation bien définie de tâche est à réaliser. Les threads Java intègre de manière native ces mécanisme de lancement (`start()`) et d'attente de fin d'exécution (`join()`).

Des mécanismes de communication entre threads sont utilisés dans une application où le robot mobile doit suivre une ligne au sol. Différentes tâches en parallèle, communiquant entre-elles, vont gérer les capteurs, les déplacements et la supervision du robot. Les tâches communiquent entre-elles par des mécanismes d'accesseurs et modificateurs d'éléments mémoire.

5. Gestion de ressources communes.
   Les problèmes liés aux ressources critiques sont illustrés par une petite application où 3 threads en parallèle lancent chacun 400.000 fois une ressource commune (une méthode d'un objet commun, `static`, qui a pour but d'incrémenter 10 fois la même variable). Les étudiants constatent que la valeur finale est largement inférieure au résultat souhaité (entre 4 et 6 millions au lieu de 12 millions). Il est alors demandé aux étudiants de gérer cette ressource commune à l'aide d'un mécanisme d'attente active (l'algorithme de Peterson [6] qui leur est donné) et de trois mécanismes d'attente passive (le verrou à l'aide de la classe `ReentrantLock`, le sémaphore à l'aide de la classe `Semaphore` et de moniteur sous Java à l'aide du mot clé `synchronized`). Les étudiants vérifient la justesse de la solution et également l'impact sur le temps de traitement de ces différentes solutions. Ils constatent également que l'utilisation d'un langage haut niveau simplifie énormément ce traitement (plus particulier par l'emploi du mot clé `synchonized`)
   L'accès à une ressource commune est également illustré sur un sujet mettant en œuvre le robot Mindstorms. Les étudiants doivent gérer un robot qui fonce à la vitesse maximale tout droit vers un mur. À l'aide du capteur à ultrasons, le robot mesure sa distance par rapport au mur. Lorsqu'il est à moins de 6 cm du mur, le robot doit ralentir pour s'arrêter à 3 cm du mur. Dans un premier temps, le système est constitué de deux tâches en parallèles, l'une qui mesure la distance par rapport au mur et donne une consigne de vitesse, l'autre qui gère les roues en fonction de la vitesse souhaitée. La communication entre ces deux tâches se fait à l'aide d'un objet tampon qui reçoit et stocke la consigne de vitesse et qui doit intégrer des mécanismes de gestion de ressources communes. La vitesse angulaire des deux moteurs n'est jamais strictement identique, (même avec la même consigne). Nous demandons alors aux étudiants de prévoir un mécanisme de synchronisation des roues. L'idée est de mesurer le déplacement angulaire sur chaque roue et de corriger cette dissymétrie par des consignes de vitesse adaptées. Ces consignes sont transférées à l'objet tampon, ce qui justifie ainsi les mécanismes de gestion de ressource commune.
6. Cadencement et ordonnancement.
   Plusieurs petits exercices permettent d'illustrer ces notions liées à une synchronisation entre threads ou à la gestion du temps :
   - La synchronisation entre thread est vue à l'aide d'un exemple basé sur le modèle Producteurs/Consommateurs. Un thread « producteur » transmet un texte à un thread « consommateur » au travers d'un tampon circulaire. Cet exercice classique permet de montrer l'intérêt des sémaphores.
   - Les tâches périodiques. L'objectif est de lancer toutes les deux secondes une tâche qui met en moyenne 500 ms pour s'exécuter. Le cadencement est géré par des temps d'attente entre deux tâches. Or le temps d'exécution de la tâche est relativement aléatoire (entre 470 et 530 ms), les étudiants doivent prévoir un temps d'attente adapté au temps d'exécution mesuré pour chaque instance de la tâche.
   – Ordonnancement. Celui-ci est vu sous la forme d'une application simulant un réveil. Deux tâches, l'une périodique (incrément de l'horloge toute les minutes) et l'autre ponctuelle (sonnerie du réveil à l'heure choisie), doivent être ordonnancées à l'aide des classes `Timer` et `TimerTask` de Java.
7. Tâche périodique - Préhension entre Threads.
   Ce dernier TP propose une synthèse des notions vues précédemment. Dans un premier temps, il est demandé de réaliser un robot qui change de trajectoire toute les 5s. Ce programme met en œuvre la `Timer` de LeJos. Cette classe, différente de celle du Java classique, lance un évènement (`timeOut`) qui sera scruté par un écouteur (`TimeListener`). Dans un second temps, il est demandé d'inclure un mécanisme de gestion d'obstacles qui prendra la main sur le fonctionnement normal en fonction des indications du capteur de distance.

2.4.3 Évaluation

L'évaluation se fait sous la forme d'un mini-projet de 2h qui demande de réexploiter, de manière progressive, les différentes notions apprises. Chaque test suit un petit scénario permettant d'intégrer les différentes notions. En 2012 par exemple, le thème était le contrôle d'une centrale nucléaire[6]. Il était donné aux étudiants plusieurs classes (chargées en mémoire) qui permettait de simuler et de contrôler le fonctionnement d'une centrale : `CapteurTemperature` qui permet de mesurer la température du cœur, `Pompe` qui contrôle la tension et donc le débit des pompes de refroidissement du cœur, `EcranControle` une interface graphique qui permet d'afficher la température du cœur, la tension de commande et le débit des pompes et éventuellement des messages et `GroupeElectrogene` qui permet de gérer éventuellement un groupe électrogène de secours. Deux autres classes, cachées des étudiants, simulaient

[6] Fukushima mon amour !!!

des lois reliant la température du cœur au débit des pompes et ce débit à la tension de commande des pompes. Les exercices, de difficulté progressive, concernaient :

1. La création d'une classe et méthode `static` d'envoi d'un message vers le siège de la compagnie. Vérification de la notion de méthode `static`.
2. La protection de cette classe accessible de plusieurs endroits par un mécanisme de gestion de ressources communes (le mot clé `synchonized` était la solution la plus simple). Vérification de la notion de gestion de ressources communes.
3. La régulation de la centrale à l'aide d'une classe héritant de Thread et dont le rôle est de maintenir la température du cœur du réacteur autours de 300°C. L'idée est de mesurer la température du cœur toute les secondes et d'ajuster la tension des moteurs des pompes (le cahier des charges de la loi de régulation était donné). Vérification de la notion de thread et de tâche périodique souple.
4. Une supervision de la centrale à l'aide d'un thread dont le rôle est de recueillir toute le 2s et d'afficher sur l'écran de contrôle la température du cœur, la tension de commande de la pompe et le débit de la pompe. La supervision doit également envoyer, toutes les 4s, la température du cœur vers le siège de compagnie à l'aide de la classe développée dans la première question. Vérification de la notion de système multitâche et communication entre threads.
5. Simulation d'un tremblement de terre avec arrêt de l'alimentation électrique principale des pompes. Il est demandé une gestion asynchrone du problème : la classe `Pompe` possède un mécanisme d'évènement qui se lance lorsque la tension est nulle. Les étudiants doivent prévoir un mécanisme de capture de cet évènement qui, le cas échéant, lancera le groupe électrogène de secours et enverra un message vers le siège de la compagnie. Vérification de la notion de gestion d'évènements asynchrones.

L'évaluation porte essentiellement sur la compréhension ou non de la notion sous-jacente de l'exercice et également sur la structure du code écrit par les étudiants. Le cas échéant, l'enseignant peut aider les étudiants qui bloquent lors d'une étape nécessaire à la suite du projet.

En 2010, première année du module, le thème portait sur le dépouillement d'un vote de plus de 10.000.000 électeurs avec des notions de dépouillement en parallèle, d'accès à l'urne commune et de communication entre dépouilleurs. En 2011, le thème était la gestion d'un parking aux places limitées avec deux entrées et deux sorties et des véhicules arrivant de manière aléatoire.

2.5 Discussion

Généralement les notions les plus simples (exception, multitâche, et même gestion de ressources commune) sont toutes bien comprises avec des moyennes de l'ordre de 13-14/20 lors des contrôles.

Le robot Mindstorms n'a été introduit que lors de l'année universitaire 2011-2012 Nous avons constaté un supplément d'intérêt lors des séances de TP avec des appropriations du robot au deçà de ce qui était demandé.

Une évaluation informelle est effectuée à la fin du module sous la forme d'une discussion avec les étudiants. Le module a été globalement apprécié et le contenu a été jugé comme intéressant et très instructif. Certaines remarques ont permis d'améliorer le contenu. Ainsi, la première année, le contenu était considéré comme trop dense par les étudiants et a été allégé (nous traitions des flux de données et de canaux de communication entre threads). Selon les étudiants, l'introduction du robot, outre son aspect ludique, à permis de rendre plus concrets certains des aspects d'un système temps réel embarqués.

3 CONCLUSION

Nous avons essayé de démontrer que la plupart de notions liées à des systèmes temps réels (gestion du temps, décomposition en tâches simple s'exécutant en parallèle, sûreté de fonctionnement,…) pouvaient parfaitement être illustrées à l'aide d'un langage haut niveau, Java dans notre cas, s'exécutant sur des systèmes d'exploitation classique. Ces différentes notions sont également mises en œuvre dans une application embarquée réactive simple, un robot LEGO Mindstorms. Les évaluations et les retours des étudiants montrent une bonne compréhension de ces notions.

Bibliographie

[1] "Organisation des études conduisant au DUT, spécialité « Génie Électrique et Informatique Industrielle », programme pédagogique national", *http://media.enseignementsup-recherche.gouv.fr/*, 2008.

[2] L. Zaffalon, "Programmation Concurrente et Temps Réel avec Java", *Presses Polytechniques et Universitaires Romandes, 2007.*

[3] A. Burns, A. Wellings, "Real-Time Systems and Programming Languages, Ada 2005, Real-Time Java and C/Real-Time POSIX", *Addison Wesley, 2009.*

[4] M. Alabau and I. Dechaize, "Ordonnancement temps réel par échéance", *Technique et Science Informatiques,. 11(3), pp. 59–123, 1992.*

[5] Java™ Platform, Standard Edition 6, API Specification *http://docs.oracle.com/javase/6/docs/api*

[6] G. L. Peterson, "Myths about the mutual exclusion problem," *Inf. Process. Lett., 12(3), pp. 115–116, 1981.*