

Faire la Java au Lycée :



Travail d'Etude et de Recherche Master 1 STIC Informatique

ETUDIANTS

Chalmeton Sébastien
Ennabli Mohammed
Lebrun Thomas
Guillaume Jean-Michel
Richter Xavier

ENCADRANTS

Mongiat Laurent
Viéville Thierry

Table des matières

1	Introduction :	2
2	Réponse au cahier des charges :	3
3	Technologies utilisées :	5
3.1	Eclipse RCP :	5
3.2	Java RMI :	7
3.3	XML & DOM:	8
3.4	Déploiement d'applications Java :	8
3.5	ORPHY GTS 2:	9
3.6	API RXTX:	9
4	Fonctionnalités :	10
4.1	L'interface Graphique :	11
4.2	Éditeur de code JavasCool et Java :	12
4.3	Traductions des erreurs d'exécution :	14
4.4	Une boîte à outils :	14
4.5	Affichage du code source généré :	16
4.6	Préférences :	17
4.7	Fichiers de configurations:	18
4.8	Folders:	18
4.9	Exécution de programmes:	19
4.10	Arrêt du programme:	19
4.11	Tutoriels et manuel intégrés :	19
4.12	Un « inspecteur » de variables:	20
4.13	Un plugin pour Orphy :	21
5	« Interface Homme Machine » :	25
6	Problèmes rencontrés :	26
6.1	Différences entre besoins de bases et besoins réels:	26
6.2	Absence quasi-totale de la société Mirelec:	26
6.3	Bugs divers & variés:	27
6.4	Difficultés de développement rencontrées :	28
7	Organisation et déroulement du projet :	30
7.1	Comparaison des plannings :	30
8	Évolution possible de l'application :	31
9	Conclusion :	33
10	Lexique :	34
11	Bibliographie :	35

Remerciements :

Nous tenions à remercier nos encadrants pour nous avoir permis de réaliser ce TER et pour leurs dévouements, ainsi que pour leurs temps accordés à nous accueillir.

Nous remercions également Fabrice Huet et son équipe pour nous avoir reçus à l'INRIA pour nous présenter les fonctionnalités d'Eclipse en développement d'application.

Ainsi que Benjamin Cabbé pour nous avoir apporté son aide sur les grandes complexités du développement d'application RCP et de plugin Eclipse.

1 Introduction :

Le projet JavasCool est né de la rencontre de deux personnes, Thierry Viéville, ingénieur chercheur au sein de l'INRIA et Laurent Mongiat enseignant, professeur de physique-chimie et ayant à sa charge une classe de seconde MPI. Ce dernier enseigne alors à ses étudiants les rudiments de la conversion de signaux analogiques et numériques par le biais de programmes écrits en Turbo Pascal. C'est en présentant l'enseignement de cette discipline particulière à Monsieur Viéville qu'il a dû se soumettre aux moqueries de ce dernier qui considérait Pascal comme un langage de « vieux » et qui l'a fortement encouragé à utiliser des langages plus contemporains. Leur vînt alors l'idée de proposer un travail pour les étudiants de l'université de Nice Sophia-Antipolis désireux de répondre à l'appel (nous). Nous prenons alors connaissance du sujet:

« Mettre en place et évaluer un environnement de développement de programmation minimal pour permettre aux lycéen(ne)s de s'initier à la programmation et aux concepts informatiques sous-jacents. »

Derrière cette courte phrase se cache toute l'ambition de rendre accessible, intéressante, ludique et en harmonie avec l'enseignement de la MPI, l'introduction de la programmation informatique au lycée. Pour cela, nous devons produire un langage de programmation basé sur Java, enrichie de quelques macros et fonctions prédéfinies et dépouillé de certains mots clefs rendant ainsi l'écriture de programmes beaucoup plus intuitive pour les élèves de seconde. Ce langage doit être incorporé à notre environnement minimal de programmation et doit pouvoir être compilé, exécuté, et fournir quelques fonctionnalités supplémentaires que nous détaillerons dans la suite de ce papier.

2 Réponse au cahier des charges :

Lors de la pré-soutenance ayant eu lieu début Avril, le jury a émis de forts doutes vis-à-vis de notre capacité à suivre notre ambition pour la finalisation de notre projet. Il a fallu nous rendre à l'évidence, nous sommes encore de très jeunes programmeurs (nous qualifier de la sorte témoigne d'un manque d'humilité de notre part) et les personnes chargées de notre évaluation sont elles bien conscientes des difficultés auxquelles nous allons devoir faire face. Pour le développement de notre produit nous pensions alors nous appuyer sur les librairies de logiciels tels que DrJava et BlueJ, implémenter notre propre interface en SWING et notre propre moteur de plugins. Il en fut autrement. En suivant leurs conseils, nous nous sommes orientés vers un développement de plugins/RCP pour la plate forme Eclipse. Cette bien heureuse mésaventure n'a en fait affecté que le diagramme de gant du cahier des charges nous obligeant à reprendre la répartition des taches dans le temps. Pour le reste, nous faisons le détail point par point de ce que nous devons fournir:

- **Une installation simple:** Nous avons utilisé un logiciel permettant l'installation personnalisée de programmes quelque soit le système d'exploitation hôte, il s'agit de IzPack, projet Open-Source. L'installation de JavasCool est tout à fait simple et conventionnelle.
- **Un macro-langage:** Le langage que nous délivrons répond entièrement à la demande du client, et est paramétrable par le biais de fichiers de configurations écrits dans un langage spécialement défini pour la tache (nous appelons ce langage BML).
- **Un traducteur Macro-Langage-Java:** Nous avons offert la possibilité depuis un menu de produire l'affichage du code java produit.
- **Un IDE:** Les seuls points auxquels nous n'avons pas pu répondre sont la possibilité de paramétrer la taille de police de l'interface de notre logiciel, l'internationalisation des menus et la possibilité de modifier dynamiquement depuis le code l'interface de la console d'exécution (ce dernier concept s'avérant en fait plus que superflu pour Monsieur Mongiat).

- **Un plugin d'acquisition de mesure via Orphy:** Cette partie du produit étant à la base classé parmi les besoins « excitants » n'est pas totalement terminé et doit faire l'objet d'une poursuite d'étude. A l'heure actuelle, nous permettons l'acquisition de signaux provenant d'une unique sonde thermomètre, l'observation de ce signal par le biais d'un graphe, et la possibilité d'enregistrer l'acquisition. Par rapport à la demande il reste à implémenter le moyen d'effectuer des mesures depuis un programme en cours d'exécution, de rendre accessible ces méthodes par une boîte à outils, et d'autoriser l'acquisition de signaux depuis n'importe quelle sonde.

- **Produit compatible JDK1.5:** Suite au constat de la présence de la version 1.6 du JRE sur les machines censées accueillir le logiciel, nous avons totalement modifié notre code pour la compatibilité avec cette dernière. L'installation de JavasCool ne devait également nécessiter aucune autre installation en parallèle, mais nous n'avons pu faire autrement que d'obliger l'installation du JDK1.6 sans quoi la compilation du code aurait été impossible.

3 Technologies utilisées :

Cette section présente les différentes technologies utilisées pour la réalisation de notre application.

3.1 Eclipse RCP :

Eclipse est dans la plupart des esprits, un environnement de développement intégré qui a été initialement développé par IBM et dont la première version date de novembre 2001.

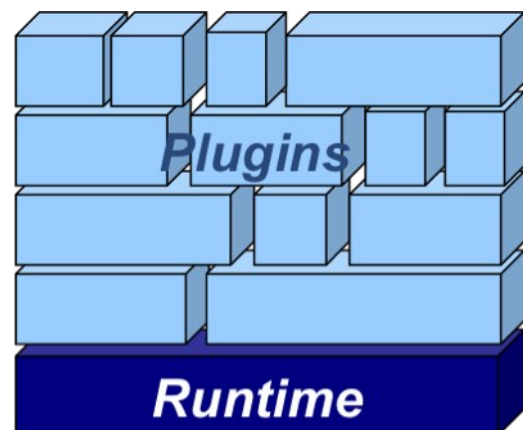
Mais la spécificité de cette application est que toute son architecture est basée sur un micro noyau aux fonctionnalités réduites et généralistes (runtime) sur lequel viennent se greffer des modules nommés plugins.

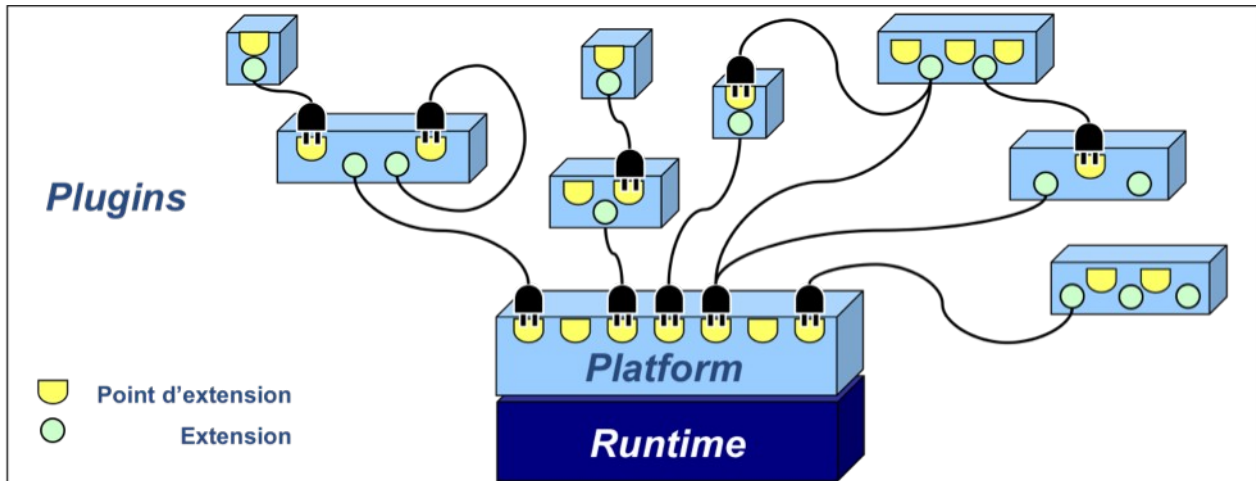
La plate-forme inclut un mécanisme pour les découvrir, les intégrer et les exécuter

Les informations nécessaires pour le déploiement d'un plugin sont décrites dans un fichier Manifest.mf. Il est chargé au démarrage avec le fichier "plugin.xml" alors que le plugin lui-même n'est chargé qu'au moment où son utilisation est requise.

Ce fichier xml définit les relations qu'établit le plugin avec les autres, elles sont de deux sortes :

- La dépendance : un plugin a besoin d'un autre pour fonctionner (simple dépendance de bibliothèques, comme entre 2 JARs)
- L'extension : un plugin répond à un point d'extension défini par un autre plugin. Un point d'extension est un contrat auquel le plugin apporte une implémentation (extension). Tout plugin peut définir lui-même des points d'extension auquel d'autres plugins peuvent contribuer.





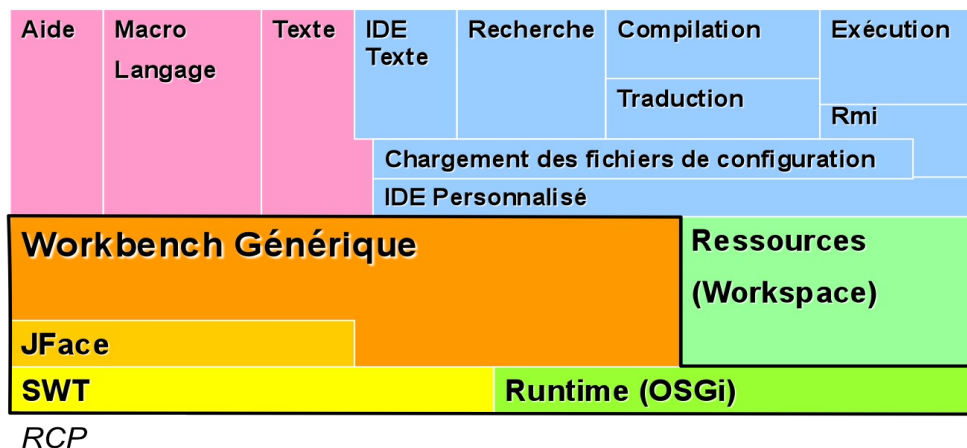
Cette conception a été souhaitée afin d'avoir une architecture modulaire et incrémentale qui offre de nombreux avantages :

- ⇒ Minimiser l'occupation mémoire et optimiser les performances
- ⇒ Mécanisme de chargement souple et incrémental basé sur l'architecture OSGI
- ⇒ Mise à jour transparente des plugins

Ainsi si on supprime tous les plugins liés à la notion d'environnement de développement, Eclipse devient une plate-forme de travail que l'on peut spécialiser aux activités de l'utilisateur et qui dispose de nombreux atouts :

- ◆ Support multi-plateformes
- ◆ Cadre de développement stricte et évolutif
- ◆ Composants graphiques riches via des bibliothèques intégrés (SWT & JFace)
- ◆ Intégrations d'éléments natifs (drag and drop, copy and paste ...)

Cette plate-forme générique, est appelée Eclipse Rich Client Platform (RCP). Elle est la base sur laquelle a été construite l'application JavasCool. Grâce à cela, il a été possible de reprendre de nombreuses fonctionnalités prédéfinies par des plugins écrits pour eclipse et d'avoir au final une architecture robuste (voir schéma ci-dessous).



3.2 Java RMI :

Le produit devait pouvoir permettre de décider des variables du programme que l'on désirait observer afin d'apprécier leur évolution. La première idée nous ayant plu est le chargement en mémoire centrale de la classe compilée dans un éditeur, et l'observation en temps réel de l'évolution des champs statiques de cette dernière.

Ce fut une bonne idée... jusqu'à ce que nous fîmes le constat que cette méthode là ne permettait pas l'implémentation de la fonctionnalité de stoppage du programme en cours d'exécution. En effet, à partir du moment où le programme écrit par les étudiants comportait des entrées/sorties et/ou des boucles infinies, il nous était impossible de stopper le thread de façon robuste et correcte. La seconde solution nous ayant donc séduits était de faire exécuter le programme directement par le système d'exploitation comme un programme traditionnel depuis lequel nous redirigerions les entrées/sorties. Là encore nous nous heurtions à un problème, de cette manière, nous n'avions aucun moyen d'observer dynamiquement l'évolution des champs de la classe... Lors de nos recherches sur les bibliothèques implémentant des solutions à nos besoins, nous avons étudié de près celle délivrée par DrJava autorisant l'évaluation de code java par le biais d'un interprète communiquant via Java RMI. La solution était là.

Nous avons donc fait le choix d'avoir recours à un service de type client/serveur, où le serveur aurait la tâche d'exécuter certains programmes et de permettre l'observation des champs de la classe principale en cours d'exécution. De cette manière, nous pouvions toujours stopper de manière correcte notre programme en tuant le processus serveur (un Thread ajouté à la JVM en tant que shutdownHook aurait la tâche de le signaler obsolète auprès du service de registre RMI-Registry, ainsi que de fermer toutes les ressources en cours d'utilisation), et également permettre toutes les autres fonctionnalités attendues. Ceci nous est vraiment apparu comme la marche à suivre car les mécanismes que nous procurait le JSE 5 étaient tout à fait obsolètes et ne nous permettait rien, et ceux fournis par le JDT d'Eclipse beaucoup trop puissants et importaient d'autres fonctionnalités que nous ne pouvions pas masquer puisque développées pour le mécanisme de plugin de cette plateforme (entre autre, l'éditeur Java était le choix par défaut du moteur de plugin pour l'édition de code java, ce qui ne permettait pas le chargement de notre propre éditeur).

3.3 XML & DOM:

Un des désirs forts de notre client était de pouvoir éditer lui même ses fichiers de configuration à l'aide d'une syntaxe à la java mais s'appuyant sur un modèle hiérarchique à la XML (de cette manière, nous passions outre le côté obscure de la syntaxe XML entachée par de nombreuses considérations commerciales). Nous avons donc écrit un basique compilateur de ce langage de configuration (désigné par BML pour bracket markup language), construisant un arbre DOM pour bénéficier de toute la robustesse qu'offrait les librairies déjà implémentés. Nous n'avons ainsi pas à nous soucier des possibles erreurs commises pendant l'édition des fichiers de configuration où l'ajout de propriétés pendant l'exécution du programme. Les configurations étaient par la suite rendues persistantes en traduisant directement l'arbre DOM en code BML. Nous avons toutefois laissé la possibilité à l'utilisateur de pouvoir observer les fichiers XML correspondant aux fichiers de propriétés BML (toujours dans l'optique d'apprentissage de technologies).

3.4 Déploiement d'applications Java :

Une condition de validation de notre projet était de fournir un moyen de déploiement simple et conventionnel pour un utilisateur lambda désireux de voir un jour notre logiciel tourner sur sa machine. Nous avons donc décidé d'avoir recours à un logiciel permettant via de simples spécifications écrites en XML de déployer n'importe quel software sur n'importe quel système d'exploitation. IzPack: <http://izpack.org/> est ce logiciel. Étant donné le caractère open-source du projet JavasCool, nous devions absolument n'avoir recours à des outils eux-mêmes open-source.

3.5 ORPHY GTS 2:

ORPHY GTS2 se présente comme un boîtier permettant d'effectuer des mesures et des commandes en tension (signal analogique) ainsi que sur des états logiques. Il possède un grand nombre d'entrées et de sorties, c'est une interface de mesures et de commandes, appelée plus communément centrale d'acquisition de données. Les performances de cette centrale autonome sont indépendantes des performances de l'ordinateur.

Le processus de mesure est commandé par un ordinateur à l'aide d'un jeu de commandes spécialement développées pour ORPHY GTS2. La communication entre l'interface et l'ordinateur se fait simplement par une liaison série.

ORPHY GTS 2 est beaucoup utilisé dans les lycées de France pour les TP de physique, principalement chez les secondes MPI.

ORPHY GTS II :



Caractéristiques techniques :

- 10 entrées analogiques unipolaires.
- 2 entrées analogiques différentielles.
- Convertisseur 16 bits.
- 10 entrées binaires.
- 5 prises DIN.
- 2 prises DB15.
- 6 fiches bananes.
- 1 connecteur d'extension.
- Connexion USB.

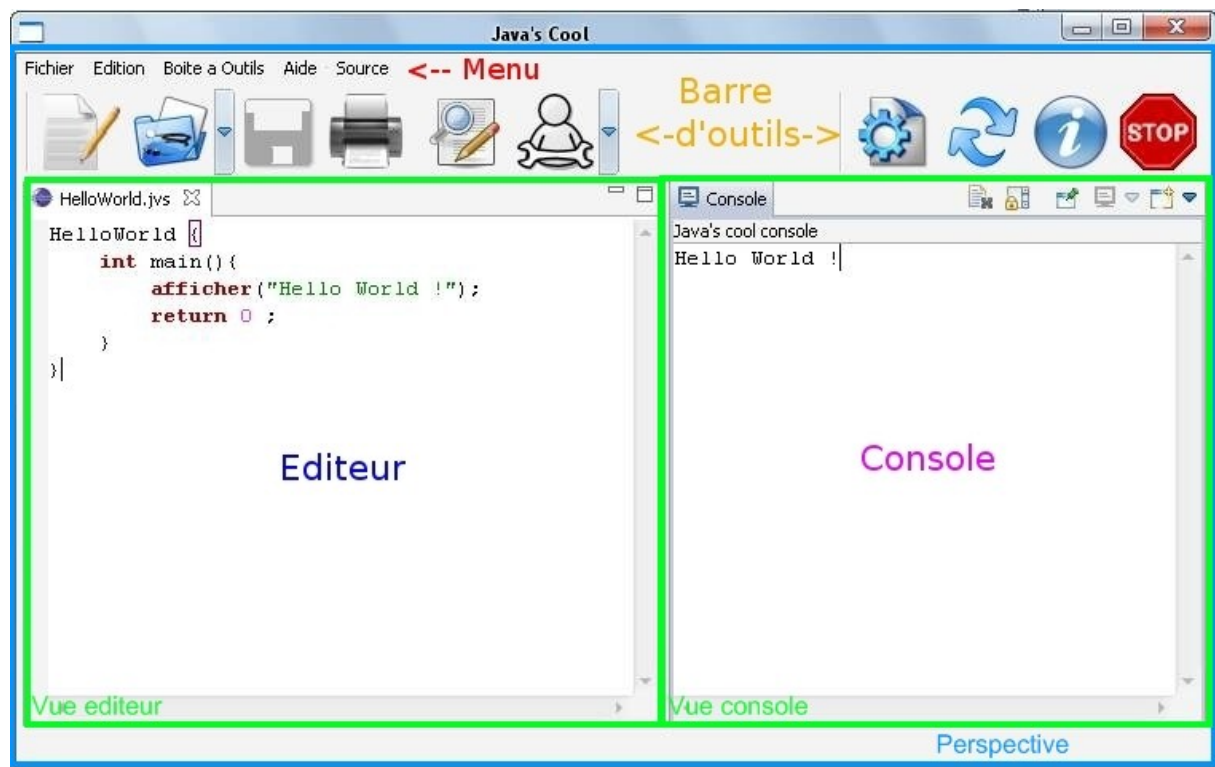
3.6 API RXTX

Pour la communication avec le matériel ORPHY nous utilisons L'API RXTX permettant la communication avec les ports séries et parallèles d'un ordinateur en utilisant Java.

Elle nécessite pour son utilisation l'installation de deux fichiers : « rxtxSerial.dll » et « RXTXcomm.jar » dans le dossier d'installation du jre de la machine.

4 Fonctionnalités :

Toutes les fenêtres faites par Eclipse RCP sont composées d'éléments communs. Ainsi la fenêtre dans sa globalité est appelée perspective, celle-ci est composée d'une barre des menus, d'une barre d'outils et de deux vues. L'une contenant l'éditeur de texte et l'autre la console



Ces informations vous aideront à mieux comprendre certains termes employés lors de l'explication des fonctionnalités.

4.1 L'interface Graphique :

Comme tout éditeur qui se respecte notre éditeur comprend toutes les fonctionnalités minimales tel que « copier », « coller », « imprimer », « sauvegarde », « chargement », « undo-redo », ...Mais également une console.

La console est un élément majeur. A la demande de notre client, celle-ci a été placée sur la droite. Il n'a pas été besoin de la créer de toute pièce. Nous avons simplement intégré la console d'éclipse dans notre interface utilisateur via le plugin « org.eclipse.ui.Console » et avons ajouté sa vue dans la perspective.

Ce qui a nécessité le plus de travail est la compréhension nécessaire à la redirection des flux d'entrées et de sorties. Nous avons mis un certain temps à bien cerner comment fonctionnait la console d'Eclipse car nous pensions au début avoir besoin de la redéfinir ou de l'étendre afin de pouvoir lui adresser des messages. Mais la solution est tout autre. Lors de l'ajout du plugin console d'Eclipse, celui-ci crée des points d'extensions parmi lesquels « MessageConsole ». En étendant cette classe, il est possible de récupérer les flots de la console et de redéfinir les flots standard par ces derniers. Mais ce que l'on a découvert plus tard, c'est qu'elle ne permettait pas de redéfinir le flot d'entrée. Ainsi après de longue recherche, nous avons découvert la classe « IOConsole » qui permet la redirection aussi bien du flot d'entrée que de sortie.

Nous avons ensuite rajouté des fonctionnalités personnelles à notre «éditeur» que nous allons vous présenter dans la suite de cette section.

4.2 Éditeur de code JavasCool et Java :

C'était le but premier de notre application que de permettre d'éditer du code JavasCool. Nous offrons également la possibilité à l'utilisateur d'éditer du code java.

Lorsque l'utilisateur développe du code JavasCool, l'application se charge de traduire ce code en Java. Cette opération est effectuée lorsque l'utilisateur choisit de compiler son code.

La traduction se déroule en plusieurs étapes; Tout d'abord on parcourt le code source JavasCool pour savoir quelles sont les macros-fonctions et les fonctions présentes dans les fichiers de configuration utilisés dans le code source, on récupère alors les imports nécessaires à l'aide des fichiers de configuration que l'on ajoute dans le fichier source java généré à partir du code JavasCool. Ensuite on ajoute devant chaque méthode et champs les mots clés « public static ». Et enfin on ajoute notre propre fonction « main » qui appelle la fonction « main » définie par l'utilisateur (si il y en a une) que l'on met dans in bloc « try-catch » de manière à pouvoir traduire les erreurs d'exécution comme expliqué précédemment.

Voici un exemple de code JavasCool suivi de sa traduction en java :

```
Tuto3{  
  
    String lirePrenom() {  
        println("entrez votre prenom :");  
        String prenom = readString();  
        return prenom;  
    }  
  
    int lireAge() {  
        println("entrez votre age :");  
        int age = readInt();  
        return age;  
    }  
  
    void main() {  
  
        String prenom = lirePrenom();  
        int age = lireAge();  
  
        println("prenom : " + prenom);  
        println("age : " +age);  
    }  
}
```

Texte 1 :Exemple de code écrit en JavasCool.

```
import static JavasCool.Macro.*;
import static JavasCool.Read.readString;
import static JavasCool.Read.readInt;
public class Tuto3{

    public static String lirePrenom(){
        println("entrez votre prenom :");
        String prenom = readString();
        return prenom;
    }

    public static int lireAge(){
        println("entrez votre age :");
        int age = readInt();
        return age;
    }

    public static void main(){

        String prenom = lirePrenom();
        int age = lireAge();

        println("prenom : " + prenom);
        println("age : " + age);
    }

    public static void main(String[] args){
        try{
            main();
        } catch (Exception e) {org.unice.JavasCool.util.erreur.ReThrower.log(e,
            "Tuto3.java", 4);}
    }
}
```

Texte 2 : Code source Java correspondant à la traduction du code JavasCool précédent.

4.3 Traductions des erreurs d'exécution :

Afin de faciliter la vie des lycéens comme de leurs professeurs, il fallait que les erreurs soient traduites. Pour cela, lors de la phase de traduction du code JVS en Java, il est ajouté un try-catch autour du main. Le catch contenant un appel vers une fonction « log » qui prend en paramètre une erreur, un nom du fichier java ainsi qu'un nombre qui correspond au décalage dû aux lignes qui ont été ajoutées lors de la traduction.

Avec l'erreur, il est possible de récupérer la classe d'erreur ainsi que tous les appels de fonctions concernées. Comme nous disposons du nom du fichier java et que nous savons que l'erreur ne peut provenir que du code source de l'utilisateur, seules les erreurs ayant un rapport direct avec le fichier de l'élève seront affichées. Ainsi l'utilisateur est toujours préservé de toutes les classes java sous-jacentes qu'il utilise sans le savoir.

La traduction s'effectue grâce au fichier « erreur.bml » qui contient la liste d'erreur ainsi que leur traduction. Lors du chargement des différents fichiers de configuration, une hash table est créée pour les erreurs avec pour clé, le nom de l'erreur java (par exemple : java.io.FileNotFoundException) et en second champs la traduction ("fichier non trouvé, vérifier le chemin du fichier). La fonction « log » ayant l'erreur en paramètre peut interroger la hash-map et afficher sa traduction.

4.4 Une boîte à outils :

La boîte à outils est un composant accessible par un menu et un groupement de boutons appropriés. Cette fonctionnalité est disponible uniquement lorsqu'on édite du code JavasCool.

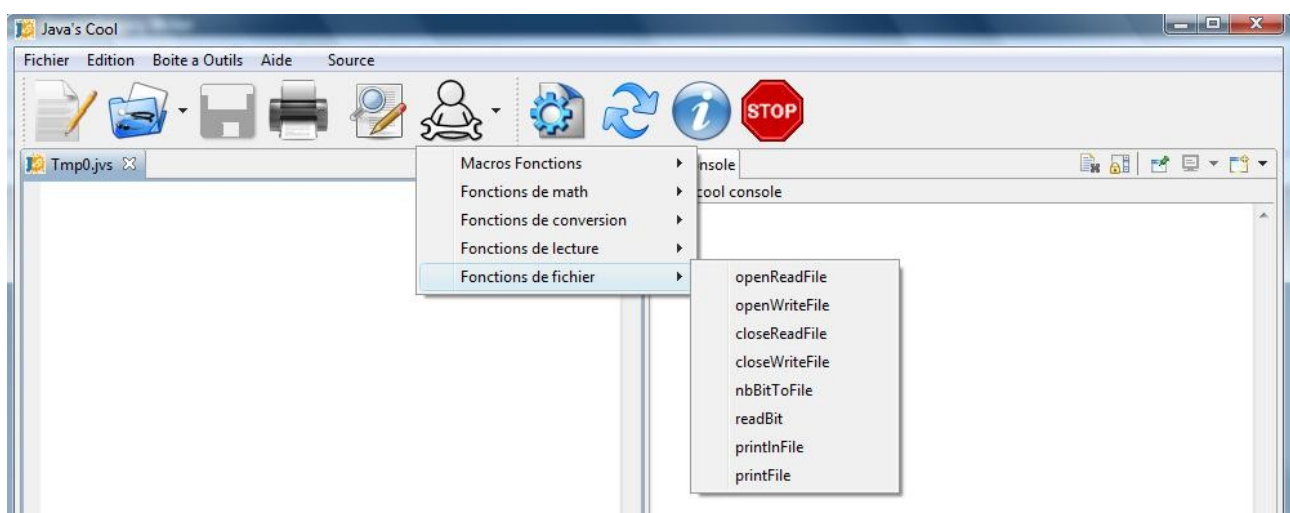


Figure 1: Utilisation de la boîte à outils à l'aide du groupe de boutons appropriés

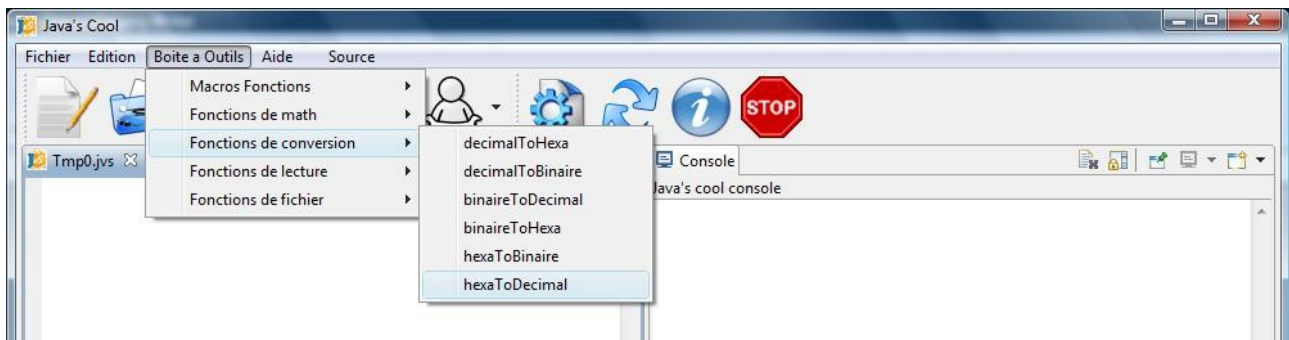


Figure 2: Utilisation de la boîte à outils à l'aide du menu approprié.

Cette boîte à outils regroupe des fonctions et des macros-fonctions organisées par types.

L'utilisation de la boîte à outils a été simplifiée au maximum pour les utilisateurs il leur suffit de choisir la fonction désirée et de cliquer dessus, automatiquement la signature de la fonction choisie s'insère dans l'éditeur courant à la position courante du curseur.

Le choix a été fait d'insérer la signature de la méthode de manière à ce que l'utilisateur voit directement le type de retour de la fonction choisie ainsi que les types des paramètres éventuels.

La boîte à outils est en étroite relation avec les fichiers de configurations. En effet toutes les fonctions et macros-fonctions présentes dans celle-ci sont définies dans le fichier de configuration.

La possibilité est offerte à l'utilisateur d'ajouter de nouvelles catégories dans la boîte à outils en créant simplement un nouveau « type » de fonction.

Voici un exemple de fonction définie dans le fichier de configuration, qui apparaîtront dans deux catégories (« fonction de math, « fonction de conversion ») dans la boîte à outils.

```
sqrt {  
  
    desc="racine carree";  
    import="java.lang.Math.sqrt";  
    signature="double sqrt(double a)";  
    type="math";  
}  
  
decimalToHexa {  
    desc="conversion decimal -> Hexadecimal";  
    import="JavasCool.Conversion.decimalToHexa";  
    signature="String decimalToHexa(int nb)";  
    type="conversion";  
}
```

Texte 3 : *Exemple de déclaration de fonctions dans le fichier de configuration approprié pour la boîte à outils*

4.5 Affichage du code source généré :

Pour les utilisateurs qui le souhaitent la possibilité est offerte d'afficher le code source Java généré par la traduction du code JavasCool en code Java. Il suffit de sélectionner dans le menu « source » le sous menu approprié.



Figure 3: *Menu permettant d'afficher le code source Java résultant de la traduction du code JavasCool.*

4.6 Préférences :

Notre éditeur de code est muni d'une coloration syntaxique des mots clés, différents types, ainsi que des macros-fonctions de manière à simplifier la lecture du code aux utilisateurs.

L'utilisateur a la possibilité de modifier ces choix de colorations syntaxiques ainsi que de restaurer ceux par défaut à l'aide d'une page de préférence accessible par le menu préférence.

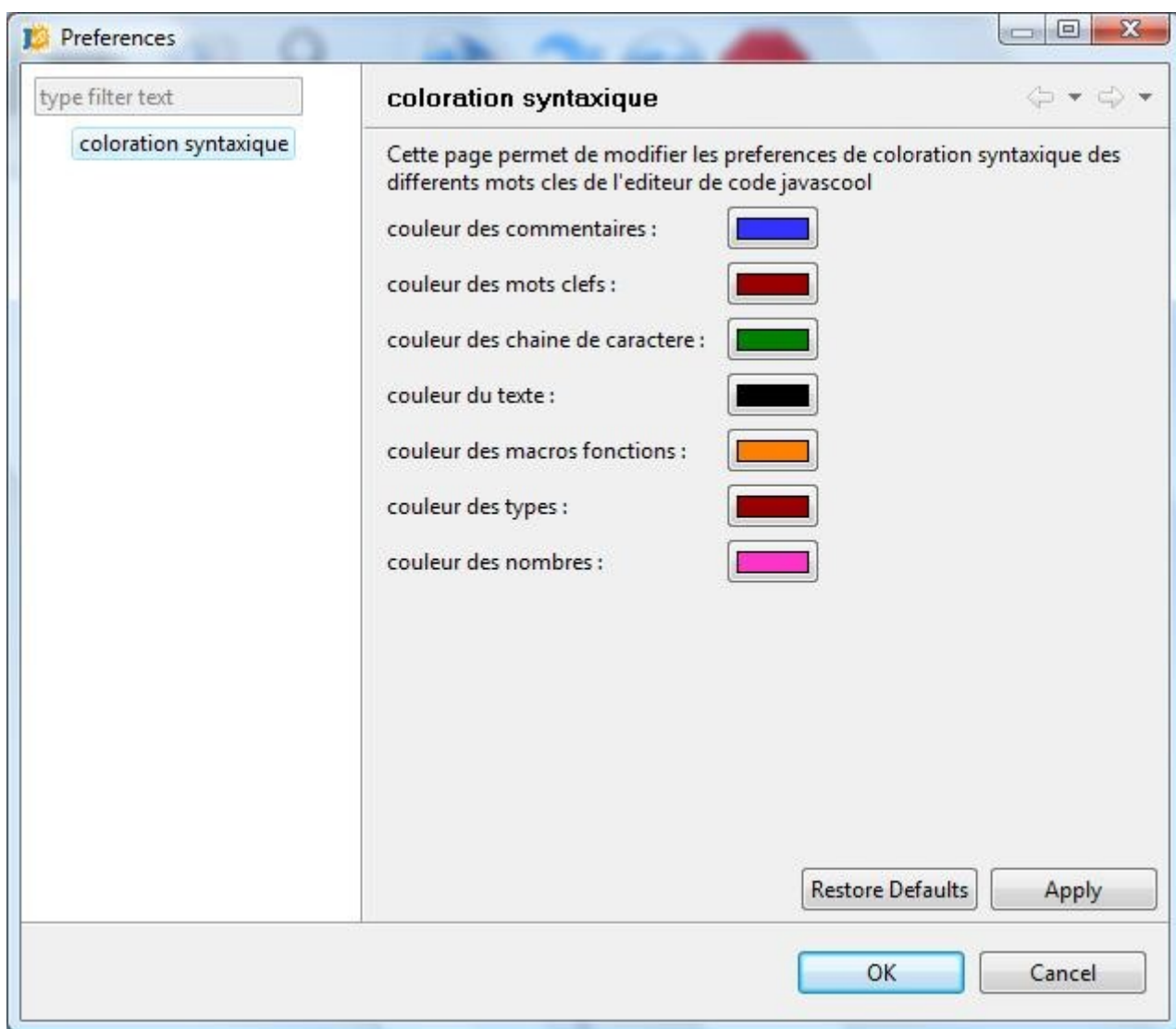


Figure 4: page de préférence permettant la modification de la coloration syntaxique de l'éditeur.

4.7 Fichiers de configurations:

Basés sur la syntaxe BML (Bracket Markup Language):

```
Tools{
  Folders{
    Folder{
      name='Mon Bureau';
      path='/home/user/Desktop/';
    }
  }
  Errors{
    Errors{
      name='java.io.FileNotFoundException';
      translate='Fichier non trouvé';
    }
  }
  .../...
}
```

S'appuyant en mémoire sur une représentation abstraite via arbre DOM, et persistant sous la forme BML et XML pour permettre à l'utilisateur de s'initier à cette syntaxe. Ces fichiers résident dans le répertoire home du user identifié sur la machine et permettent de conserver la configuration de l'espace de travail de l'utilisateur, de spécifier le degré d'aide du macro langage JavasCool, de spécifier des traductions d'erreurs... La configuration courante réside en mémoire centrale et est sauvegardée à chaque fois que l'utilisateur ferme son espace de travail JavasCool.

4.8 Folders:

Dans la plupart des lycées actuels, et c'est le cas de nos clients, l'architecture réseau visible pour un étudiant se résume à différents espaces de stockages que l'on peut interpréter différemment. Prenons pour exemple le nouveau lycée Valbonne Sophia-Antipolis:

- Un espace home où l'on retrouve tout ce qui concerne son espace de travail tel que ces préférences, son bureau...

- Un espace ressource où nous sommes en mesure de trouver tous ce dont nous avons besoin pour réaliser nos TP, devoirs...

- Un espace dépôt pour pouvoir déposer notre travail lors d'une évaluation par exemple.

Notre client voulait pouvoir rendre accessible ces différents espaces en un seul clic. Nous avons donc implémenté un mini gestionnaire d'espace de travail permettant d'ajouter/supprimer des répertoires et ainsi ouvrir/éditer/sauvegarder directement des fichiers JVS/Java depuis ces espaces.

4.9 Exécution de programmes:

Notre IDE JavasCool devait permettre une exécution de programme, mais pas tout à fait conventionnelle. En effet, il devait être possible de modifier son environnement de manière à voir évoluer les variables du programme en temps réel. Nous avons donc fourni ce service grâce à un mécanisme d'introspection sur les variables statiques de la classe principale du programme. Notre logiciel étant destiné aux secondes MPI, cela devait notamment permettre de voir l'évolution d'acquisitions réalisées à l'aide d'un matériel de mesure.

4.10 Arrêt du programme:

Nous avons ici à faire à de jeunes programmeurs qui n'ont pas à se soucier de la validité de leurs programmes dans le sens où, en seconde MPI, l'outil informatique/ordinateur est avant tout mis à la disposition des élèves pour leur permettre l'observation de traductions de signaux. On peut donc tout à fait imaginer des situations où l'on effectuerait une acquisition de signal dans une boucle infinie sans se soucier de la terminaison du programme. Il fallait donc autoriser l'utilisateur à stopper son programme de manière sécurisée pour la machine. Nous rappelons que notre outil est destiné à un fort degré d'interaction avec le système que cela soit en lecture de fichiers, de signaux ou autres. Il était donc indispensable de terminer un programme en libérant toutes ces ressources sous peine de voir les performances de la machine s'effondrer.

4.11 Tutoriels et manuel intégrés :

Pour répondre à la demande du client nous avons développé une dizaine de tutoriels de manière à apprendre aux utilisateurs à maîtriser l'application ainsi que le langage JavasCool.

Les tutoriels ont été développés en plus proche collaboration avec le client de manière à coller au plus près au programme scolaire des MPI.

Nous avons donc développé deux plugins pour notre application, qui permettent via notre application d'accéder directement à l'aide du menu « Aide » aux tutoriels et manuel.

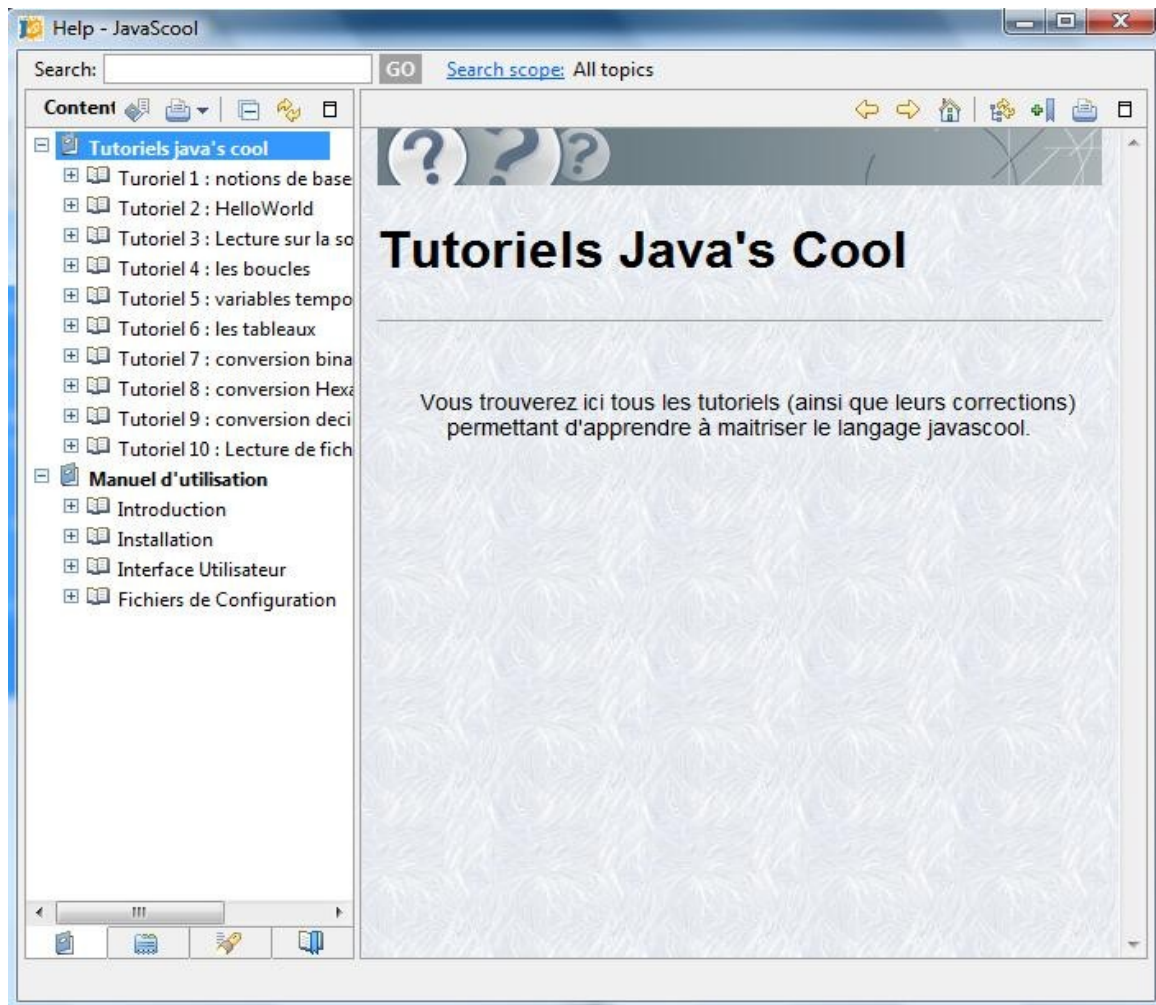


Figure 5: Page d'aide permettant d'accéder aux tutoriels et au manuel d'utilisation de l'application.

4.12 Un « inspecteur » de variables:

D'après la spécification du langage de programmation que nous avons établi, il est impossible d'instancier des objets, et les seules variables que l'on peut voir apparaître à l'extérieur de toute méthode seraient, en Java, qualifiées « public static ». Ceci nous permet de répondre à une partie du cahier des charges voulant permettre aux étudiants d'observer l'évolution en temps réel de ces variables. Pour cela, par l'activation d'une simple fonction, ils ont la possibilité d'afficher les valeurs des variables de la classe principale en exécution depuis une autre JVM sur leur machine.

4.13 Un plugin pour Orphy :

La volonté de création de ce plugin par nos encadrants vient du manque de fiabilité et de simplicité des logiciels actuellement disponibles, le but était donc d'intégrer à JavasCool un outil le plus simple et robuste possible.

Notre plugin ORPHY permet de relever des données sur les entrées analogiques différentielles où l'on a branché soit un thermomètre soit un voltmètre. Le choix du type d'entrée se fait dans l'interface et modifie l'affichage en conséquence.

S'agissant d'un plugin pour une application ECLIPSE RCP, la totalité de l'interface est codée en SWT pour une meilleure compatibilité.

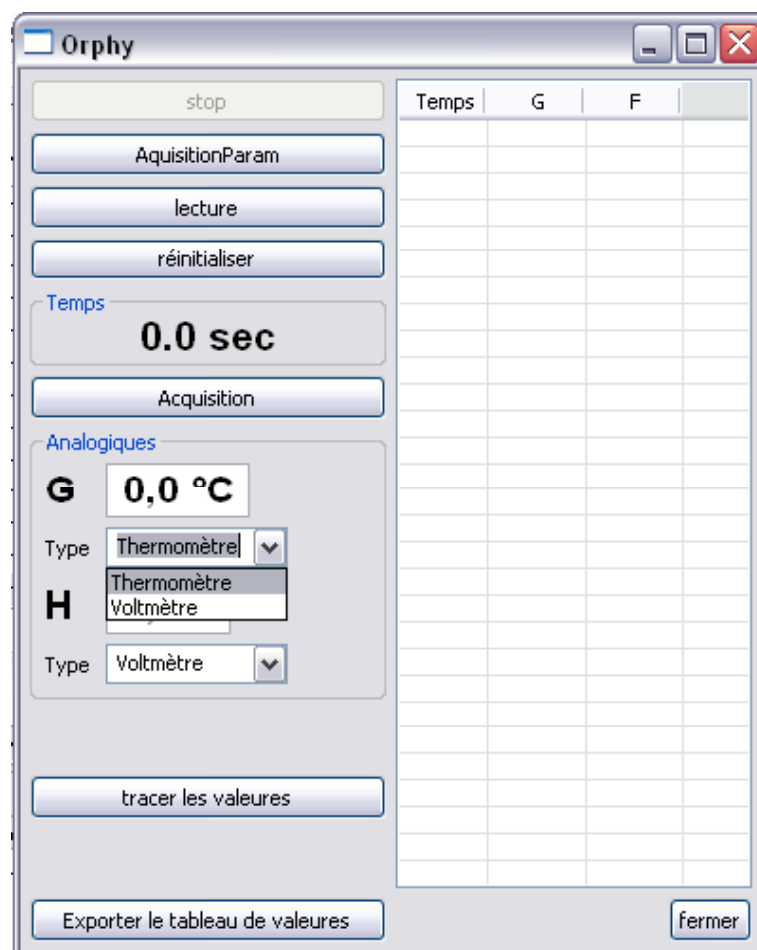
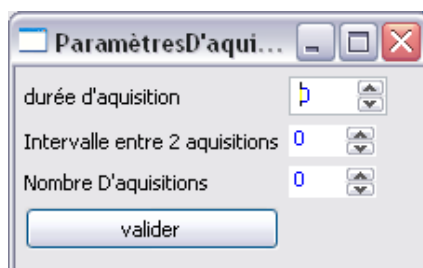
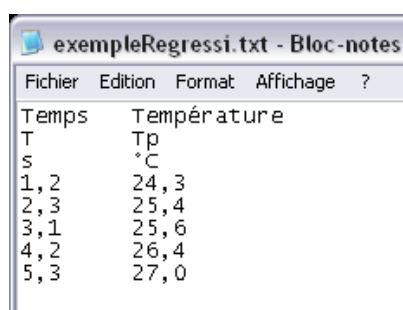


Figure 6: Interface graphique du plugin Orphys

L'acquisition des mesures et leur rangement dans le tableau peut se faire manuellement par le bouton acquisition, après avoir lancé le compteur, où elle peut se faire de manière paramétrée, en fonction de la durée, de l'intervalle de mesure ou du nombre d'acquisitions :

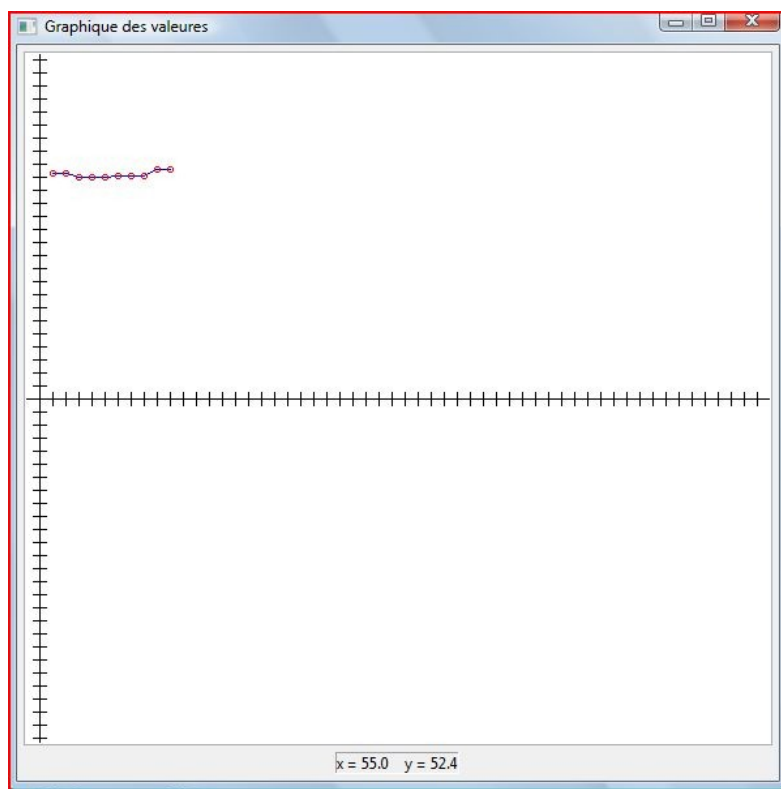


Le plugin permet l'exportation des données enregistrées sous format texte compatible avec le logiciel REGRESSI, cela permet à l'utilisateur d'étudier la courbe d'acquisition, voici un exemple d'un tel fichier :



Temps	Température
T	Tp
s	°C
1,2	24,3
2,3	25,4
3,1	25,6
4,2	26,4
5,3	27,0

Bien que les résultats aient pour but d'être étudiés dans un autre logiciel, nous avons intégré l'option de tracer les données sous forme de graphique simplifié, pour avoir un aperçu des résultats avant l'exportation :



Concernant la communication entre l'ordinateur et ORPHY, les pilotes fournis avec le matériel créent un port COM virtuel sur le port USB sur lequel est branché ORPHY. La communication se fait ensuite par envoi de bits correspondants à une commande, ORPHY répondant par un nombre de bits fonction du type de réponse.

Il y a deux types de protocoles pour envoyer des commandes à ORPHY : rapide ou usuel, le premier ne contient pas toutes les commandes existantes, mais ne nécessite l'envoi que d'un caractère ASCII alors que la syntaxe d'une commande dans le second fait plusieurs caractères. Notre plugin n'utilisant pas toute la complexité du matériel nous avons décidé d'utiliser le protocole rapide pour gagner du temps.

Voici le tableau des commandes en protocole rapide de ORPHY GTS II :

PROTOCOLE RAPIDE D'ORPHY GTS 2			
n est le numŽro des E/S	EXEMPLES		
	Equivalence	code HEX/DEC	rŽponse de GTS2
mise a zŽro d'une sortie binaire Žquation dŽcimale : code = n	WRBIT 0 WRBIT 7	0/0 7/7	
mise Ā 1 d'une sortie binaire Žquation dŽcimale : code = 32+n	WSBIT 0 WSBIT 7	20/32 27/39	
rŽpŽtition de la derniŁre commande Žquation dŽcimale : code = 43	...	2b/43	...
lire une entrŽe analogique Žquation dŽcimale : code =64+n LSB puis MSB	WEA 0 WEA 11	40/64 4b/75	selon format 1 ou 2 octet(s) 1 ou 2 octet(s)
lire une entrŽe binaire Žquation dŽcimale : code =96+n	WEBIT 0 WEBIT 9	60/96 69/105	1 octet 1 octet
commander les 6 sorties binaires Žquation dŽcimale : code = 128 + Žtat des sorties ex : SB0=SB1=SB2=SB3=1 SB4=SB5=0 alors Žtat = 15	WSBLOC 15	8F/143	
lire les 10 entrŽes binaires Žquation dŽcimale : code = 192	WEBLOC	C0/192	2 octets
compte les changements d'Žtat pour les entrŽes fronts Žquation dŽcimale : code = 208 + n	WCPT 0 WCPT 2	D0/208 D2/210	2 octets 2 octets LSB puis MSB

La seule commande dont nous ayons eu besoin pour le moment est la lecture sur entrŁe analogique.

Exemple :

Une rŁcupŁration de donnŁe sur ORPHY pour l'entrŁe analogique diffŁrentielle 1 (qui correspond Ā l'entrŁe analogique 11), sur laquelle est branchŁ un thermomŁtre, se passe de la maniŁre suivante :

Envoi du code « 75 »(64 + 11) en dŁcimal vers ORPHY.

RŁcupŁration des 2 octets de rŁponse.

Conversion des 2 octets, alors sous forme hexadŁcimale, en forme dŁcimal pour avoir la valeur de la tension :

$$U = (\text{LSB} + \text{MSB} * 256) / 65535.0 * 20)$$

Multiplication de la tension par le coefficient correspondant au thermomŁtre :

$$T = U * 11$$

5 « Interface Homme Machine » :

Le logiciel devant être utilisé par des personnes n'ayant pas forcément de grandes connaissances en informatique, il est donc important que l'interface utilisateur soit simple à prendre en main et de plus agréable à l'œil pour augmenter l'envie de maîtriser le logiciel par les lycéens. Pour ce faire nous avons utilisé des icônes le plus standards possible, bien distincts, de bonne taille ainsi qu'une info-bulle qui permet d'avoir une brève description de la fonctionnalité d'un bouton lorsqu'on laisse le curseur dessus un certain temps.

De plus pour s'assurer que l'élève comprenne qu'une icône est cliquable lorsque son curseur passe dessus, ce dernier est mis en valeur. Par contre dans le cas où l'action n'est pas possible, l'icône est grisée.

On a aussi décidé de minimiser le nombre de bouton et de séparer par un délimiteur celles liées à l'écriture du code source et celles liées à son exécution (la séparation bien distincte avec un blanc entre n'a pas été possible). Cela n'étant pas autorisé par Eclipse RCP). Tout cela afin de ne laisser que le strict nécessaire et de donner un sentiment de simplicité et de maîtrise de l'interface.

Ainsi à une époque, il était question d'avoir un bouton « ouvrir » qui aurait permis à un élève de chercher un fichier sur le disque de manière classique en parcourant les répertoires du disque et un bouton « Ouvrir dossiers préférés » qui lui permettait de se rendre directement à certains dossiers pré-enregistrés. Mais lorsque les deux fonctionnalités furent implémentées et mises en place dans l'interface, il s'est avéré que cela n'avait pas réellement de sens, dans le cadre de l'usage de notre produit, d'offrir une ouverture classique des dossiers car les élèves sont confinés à des répertoires et qu'ils risquaient de se perdre dans les méandres des répertoires. Et de plus, ils n'auraient pas compris l'utilité de la fonction « ouvrir » qui aurait pu les déstabiliser car l'icône se situait avant celui de l'ouverture des fichiers préférés. Finalement, nous n'avons gardé que la seconde possibilité et elle a pris la place d'ouvrir.

Nous avons mis un point d'honneur à ce que les icônes soient disposées de manière à respecter l'ordre d'utilisation ainsi que leur fréquence d'utilisation. Par exemple pour les boutons qui ont attiré à l'exécution du programme, l'ordre pour le lancer est d'abord de le compiler puis de l'exécuter.



Une fois en cours, on peut inspecter les variables et si jamais il y a un problème, le bouton stop qui est le dernier recours peut être appelé.

D'autres fonctionnalités ont été ajoutées pour être sûr que l'élève ne commette pas d'erreur involontaire. Lorsque le programme se ferme alors que le code source a été modifié depuis la dernière sauvegarde, un message apparaît demandant s'il ne souhaite pas sauvegarder avant de quitter.

6 Problèmes rencontrés :

Comme tout projet, lors de la réalisation de notre application nous avons été soumis à certains problèmes de développement. Nous présentons ici ceux rencontrés et la méthode de résolution appropriée utilisée.

6.1 Différences entre besoins de bases et besoins réels:

Nous nous sommes fiés aux attentes du client en fonction de ce que nous avons pu apprendre de lui à propos de l'architecture réseau, existants logiciels et hardware auxquels notre outils devait se soumettre pour pouvoir fonctionner convenablement sur le site du lycée VSA (premier client de JavasCool). Nous avons donc rédigé le cahier des charges en fonction... Ce n'est qu'après une première période de codage débouchant sur un premier test en conditions réelles que nous nous sommes aperçus que bon nombre des informations que nous avions étaient erronées:

- La version de java présente se trouvait être JSE 1.6 contre JSE 1.5. Il se trouve que les outils permettant le développement de java ont énormément changé entre ces 2 versions.
- Le réseau devait se limiter à du tout local. Il apparaît en fait qu'un logiciel de partage de fichiers (Samba) est présent, accuse des lenteurs et rend notre méthode de gestion de documents obsolète.
- Les machines étaient à la base du Intel 32bits et sont en fait du AMD 64bits (les outils mis à notre disposition par le PDE d'Eclipse utilisent seulement des morceaux de codes natifs et nécessitent un paramétrage méticuleux notamment en ce qui concerne l'OS et l'architecture processeur de la machine d'accueil).

6.2 Absence quasi-totale de la société Mirelec:

Une des facettes intéressantes du projet JavasCool était le développement d'une interface de communication entre l'outil Orphy GTS de la société Mirelec. Il s'est avéré être une tâche non triviale que de pouvoir joindre des employés de la société pour avoir des renseignements concernant leur matériel ou leur emprunter un appareil pour nous permettre le développement d'un driver écrit en Java.

6.3 Bugs divers & variés:

SUN: nous avons été dans l'obligation de constater que certains mécanismes non communs mis à notre disposition par SUN dans sa JVM présentent de nombreux points à améliorer qui font tous l'objet de recommandation sur les « Bugs Reports » spécifiques:

- Dans la version JSE 1.6 maintenant disponible sur les principaux OS, Sun s'est autorisé à ne pas développer certaines bibliothèques graphiques n'autorisant plus par la même notre application à tourner sur les Systèmes d'exploitation ayant eu droit à ce « traitement de faveur » (il s'agit ici de MacOS X).
- La JVM ne prend pas en compte les terminaisons de processus lancés via un ProcessBuilder pour fermer les flux d'entrée/sortie. Les redirections de flux se font dans des Threads à part qui ne sont pas étiquetés « daemon » ce qui n'autorise pas la JVM exécutant le process à terminer tant que ces Threads n'ont pas terminé leur exécution.
- L'implémentation de la JVM n'est pas équivalente sur les différents Systèmes d'exploitation en ce qui concerne ces exécutions de processus directement par le système hôte (notamment pour ce qui est des entrée/sortie).
- Dans la version 1.6 les outils de compilation n'ont pas été intégrée au JRE (comme dans les version précédentes) mais au JDK. Nous sommes donc contraint d'installer le JDK 16 sur les machines souhaitant utiliser notre application.
- Eclipse PDE est victime de son succès et voit grandir l'intérêt qu'apportent les développeurs aux services qu'il expose. Il s'avère que le projet est conséquent et est sujet à des manques d'implémentations. De plus, les incompatibilités entre les différentes versions ne font que renforcer ce problème.
- L'outil d'aide au développement de plugin et à l'écriture de fichiers de spécifications est à ces balbutiements et présente de nombreuses failles ce qui nous a obligé à comprendre la forme et la signification de ces fichiers de manière à les éditer directement.
- Il n'existe pas encore réellement de communauté de programmeurs Eclipse PDE en France, les mailing-lists étrangères sont très obscures et la complexité des interfaces à implémenter pour voir son application fonctionner grâce aux technologies Eclipse font de cet outil une alternative compliquée. De plus, même si la documentation est complète, elle manque cruellement d'exemples.

- Pour un petit projet comme le nôtre, le fait de s'appuyer sur la technologie Eclipse RCP en a fait une application lourde, le Runtime-Core responsable du chargement de plugin est une mécanique éprouvée mais conséquente et nécessitant énormément de ressources.

L'utilisation de différentes JVM pour le fonctionnement de notre application a rendu difficile le partage d'informations. Le partage du serveur permettant l'exécution de programmes n'échappe pas à cette règle. Le seul moyen que nous avons trouvé pour nous soustraire à cette contrainte a été de n'autoriser qu'une seule instance de JavasCool à fonctionner à un instant donné.

6.4 Difficultés de développement rencontrées :

Lors du développement des pages de préférences nous avons rencontré des difficultés de développement pour la mise à jour automatique des préférences sur l'éditeur, ceci est dû au fait que nous sommes novices en développement d'application RCP et plugin Eclipse et du manque cruel de documentation sur internet à ce sujet.

Pour la traduction du code JavasCool et Java, nous avons rencontré de nombreuses difficultés en particulier, au début nous nous étions basés sur l'idée de remplacer dans le code les macros-fonctions et les fonctions qui se trouvaient dans nos fichiers de configurations par leurs traductions. Par exemple si l'utilisateur écrivait ceci :

```
void main(){  
  
    int a = max(3,5);  
    println(" a = "+a);  
}
```

nous aurions traduit ce code par :

```
void main() {  
  
    int a = java.lang.Math.max(3,5);  
    System.out.println(" a = "+a);  
}
```

Mais nous sommes aperçus que cette opération n'était pas robuste car nous passions par des expressions régulières pour effectuer ce traitement qu'il était facile de les faire « sauter ». Nous avons décidé d'écrire des classes contenant nos macros-fonctions et d'en rajouter l'import au début du code ce qui sensiblement plus robuste.

Pour l'exemple précédent nous obtenons donc :

```
import static JavasCool.Macro.*;
import static java.lang.Math.max;

public class Tmp34{

    public static void main(){

        int a = max(3,5);
        println(" a = "+a);
    }

    public static void main(String[] args){
        try{
            main();
        } catch (Exception e){
            org.unice.JavasCool.util.erreur.ReThrower.log(e,
                "Tmp34.java", 3);
        }
    }
}
```

Nous avons eu quelques difficultés à communiquer correctement avec ORPHY. Croyant tout d'abord qu'il s'agissait d'ORPHY GTS version I (branché par port série à l'ordinateur), nous avons dû revoir nos premières ébauches de fonctions et faire des recherches pour savoir s'il fallait communiquer avec ORPHY par une API USB quand nous avons vu qu'il s'agissait en fait d'ORPHY GTS II (branché exclusivement en USB). Les premiers temps de travail sur ORPHY n'ont pas été très faciles étant donné le peu de communication de la société créatrice de ORPHY concernant son utilisation. L'installation des drivers et de l'API RXTX (permettant la communication avec les ports séries) a présenté également quelques difficultés du fait des incompatibilités avec d'autres systèmes d'exploitation que Windows XP, nous avons finalement installé ou émulé XP sur nos ordinateurs.

7 Organisation et déroulement du projet :

La Réalisation du projet s'est déroulée en deux grandes phases :

- une première d'une durée de 8 semaines pendant laquelle nous avons effectué des recherches sur l'existant pouvant répondre à nos besoins, la prise de contacts avec nos encadrants, ainsi que la réalisation du cahier des charges (validé par nos encadrants, dont le client)
- Puis cette phase s'est suivie d'une seconde, de même durée durant laquelle nous étions consacrés à plein temps au développement de notre application.

Pour la deuxième phase nous nous sommes subdivisés en sous groupes et nous sommes répartis les tâches.

7.1 Comparaison des plannings :

lors de la première phase nous avons rédigé un planning d'organisation que nous devons suivre pour la seconde phase de notre projet.

Les deux plannings sont disponible en annexe.

On peut s'apercevoir facilement que les différences entre le planning que l'on devait suivre et la planning effectué les divergences sont grandes. On peut expliquer ces divergences de nombreuses manières. Notamment lors de la pré-soutenance on nous a indiqué l'existence du moteur Eclipse qui pouvait convenir à notre application. Nous avons donc du consacrer un certains temps à l'apprentissage de la technologie Eclipse ce qui n'était pas prévu préalablement.

Ensuite certaines tâches se sont avérées plus longues à réaliser que prévu et d'autres inversement plus courtes. Notamment la correction de « bugs » et la réalisation de l'interface graphique. Nous avons également effectué de plus nombreux tests en condition réelles que prévu (ce qui n'est pas un mal au contraire).

De plus il était difficile de prévoir à peu près au jour près les tâches que nous allions réaliser car nous ignorions les difficultés auxquelles nous serions confrontés.

Au final nous n'avons pas supprimé de tâches notamment en ce qui concerne la réalisation du plugin Orphy pour notre application que nous étions amenés à supprimer si l'on prenait du retard dans la réalisation du projet

Nous avons eu un réel aperçu de la difficulté de prévoir le temps de réalisation d'un projet et l'organisation que cela demande.

8 Évolution possible de l'application :

L'application développée étant une application RCP Eclipse, on peut donc créer de nombreux plugins pour notre application.

Dans cette section nous présentons les fonctionnalités qui pourraient être créées pour notre application.

- Arriver à utiliser le JDT d'Eclipse pour hériter de toute la robustesse, portabilité et aux avantages fournis par la plateforme.
- Transformer la grammaire du macro langage de JavasCool par le biais d'un vrai compilateur et y intégrer un mécanisme de listener pour le rafraîchissement des variables en observation par introspection (à l'heure actuelle, le rafraîchissement se fait périodiquement que les variables évoluent ou non).
- Porter le projet sur MacOS X. Actuellement notre projet dépend du jdk 1.6, or celui pour MacOSX a été développé non pas par Sun mais par eux mêmes et il semble qu'ils aient oublié de traduire certaines librairies.
- Internationaliser les menus.
- Intégrer un éditeur graphique pour les fichiers de configurations.
- Intégrer un éditeur/compilateur pour les fichiers de macros.
- Traduire les erreurs de compilation.
- Développer un système de mise à jour de l'application directement via l'interface, en utilisant le réseau internet. Ceci éviterait aux utilisateurs de télécharger et d'installer à la main les nouvelles versions de chaque plugin de notre application.

Le plugin ORPHY nécessiterait quelques améliorations pour pouvoir remplacer définitivement les anciens logiciels d'acquisition utilisés par notre encadrant, Monsieur Mongiat.

Tout d'abord, il faudrait implémenter le fait que le plugin puisse reconnaître automatiquement le port COM sur lequel ORPHY est connecté, même s'il est possible actuellement de le savoir grâce au tableau de configuration Windows et qu'il serait donc possible d'ajouter un choix de port paramétrable, cette option n'est pas assez simple pour une utilisation par des élèves de seconde.

Enfin, le but de ce plugin étant de remplacer des logiciels peu fiables qui entravaient le déroulement des TP, il faudrait rendre le plugin totalement robuste avec des séries de tests complets.

Concernant l'évolution du plugin, il y a deux choses qui pourraient le faire évoluer, premièrement permettre l'acquisition avec plus, voir la totalité du matériel de physique que l'on peut brancher à ORPHY, et secondement orienter l'application vers la possibilité de manipuler le matériel physique branché à ORPHY au moyen des sorties disponibles sur la centrale d'acquisition, pour permettre par exemple la création d'un thermostat par l'intermédiaire du code java.

9 Conclusion :

Nous avons choisi ce sujet de TER pour avoir une première idée de ce que représentait l'immersion totale dans une situation de travail fournisseur/client. Les dimensions intéressantes de notre projet étaient:

- Le logiciel à fournir avait un réel intérêt et il existait une vraie demande.
- Le logiciel permettrait l'initiation de jeunes étudiants aux rudiments de la programmation d'une manière assistée, ludique et intéressante. Chose nous ayant sûrement fait défaut pendant les premières années de notre cursus pré-bac.
- Nous devions mettre en pratique bon nombre des cours que nous avons suivis pendant ces dernières années. En plus, et cela n'est apparu que tardivement dans le projet, il nous aura permis de nous initier aux méthodes de programmations fournies par la plate forme Eclipse PDE, mécanisme appelé à devenir incontournable dans les années à venir.
- Le projet comprenait la totalité des différentes phases que nous serons amenés à rencontrer dans la suite de notre carrière, à savoir, étude du besoin, management de projet, test/validation du produit, déploiement et distribution du produit.

Nous avons donc la possibilité d'enrichir considérablement notre expérience de débutant par la réalisation de ce projet. La dimension « professionnelle » dont il fallait faire preuve nous a permis de quitter le cadre d'apprentissage et de développement scolaire pour nous immerger dans celui bien plus intéressant du développement en équipe avec les avantages et inconvénients induits. Nous avons du nous confronter à des bugs et problèmes bien particuliers et pour la plupart non résolus pour arriver à finaliser JavasCool.

Nous espérons seulement que ce software destiné à l'éveil de lycéens aux sciences informatiques puisse apporter autant à ces étudiants que ce que son développement nous a permis d'en apprendre sur nous et le rapport que nous avons à la programmation par le passé. Le support et l'extensibilité du produit étant assuré, il serait agréable de voir que son intérêt auprès des professeurs de seconde MPI est bien présent et ne se limite pas à nos clients.

10 Lexique :

MPI : *mesures physiques informatisées.*

DB15 : *Connecteur trapézoïdal à 15 broches, utilisé pour d'autres connexions que les liaisons série et parallèle.*

SWT : *toolkit d'IBM similaire à Swing de sun.*

RXTX : *api java pour la communication série.*

JRE : *Java Runtime Environnement, machine virtuelle java.*

COM : *port de communication.*

RMI : **Remote method invocation**, est une interface de programmation (API) pour le langage Java lui permet d'appeler des méthodes distantes. L'utilisation de cette API nécessite l'emploi d'un registre RMI sur la machine distante hébergeant ces objets que l'on désire appeler au niveau duquel ils ont été enregistrés.

Cette bibliothèque qui se trouve en standard dans Java J2SE, est une technologie qui permet la communication via le protocole [HTTP](#) (ou [IIOP](#), depuis la version 1.3 du JDK) entre des objets Java éloignés physiquement les uns des autres, autrement dit s'exécutant sur des machines virtuelles java distinctes. RMI facilite le développement des applications distribuées en masquant au développeur la communication client / serveur.

RCP : Le terme Plateforme client riche (en anglais Rich Client Platform) désigne un type de plate-formes de développement pour la réalisation d'applications, exemple : Eclipse..

11 Bibliographie :

Orphys :

- <http://christian.rellier.free.fr/orphy/ORPHYGTS.htm>

Librairie usb java :

- <http://www.apifinder.com>
- <http://www.icast.com>
- <http://www.java.sun.com>
- <http://www.javafr.com>

Librairie série java :

- <http://christophej.developpez.com/tutoriel/java/javacomm/>
- <http://java.sun.com/products/javacomm/>
- <http://www.labo-sun.com>
- <http://www.rx-tx.org>

Librairie SWT :

- <http://www.eclipse.org/swt/>
- <http://www.tutoriaux.com/forum/showthread.php?t=2768>

RMI :

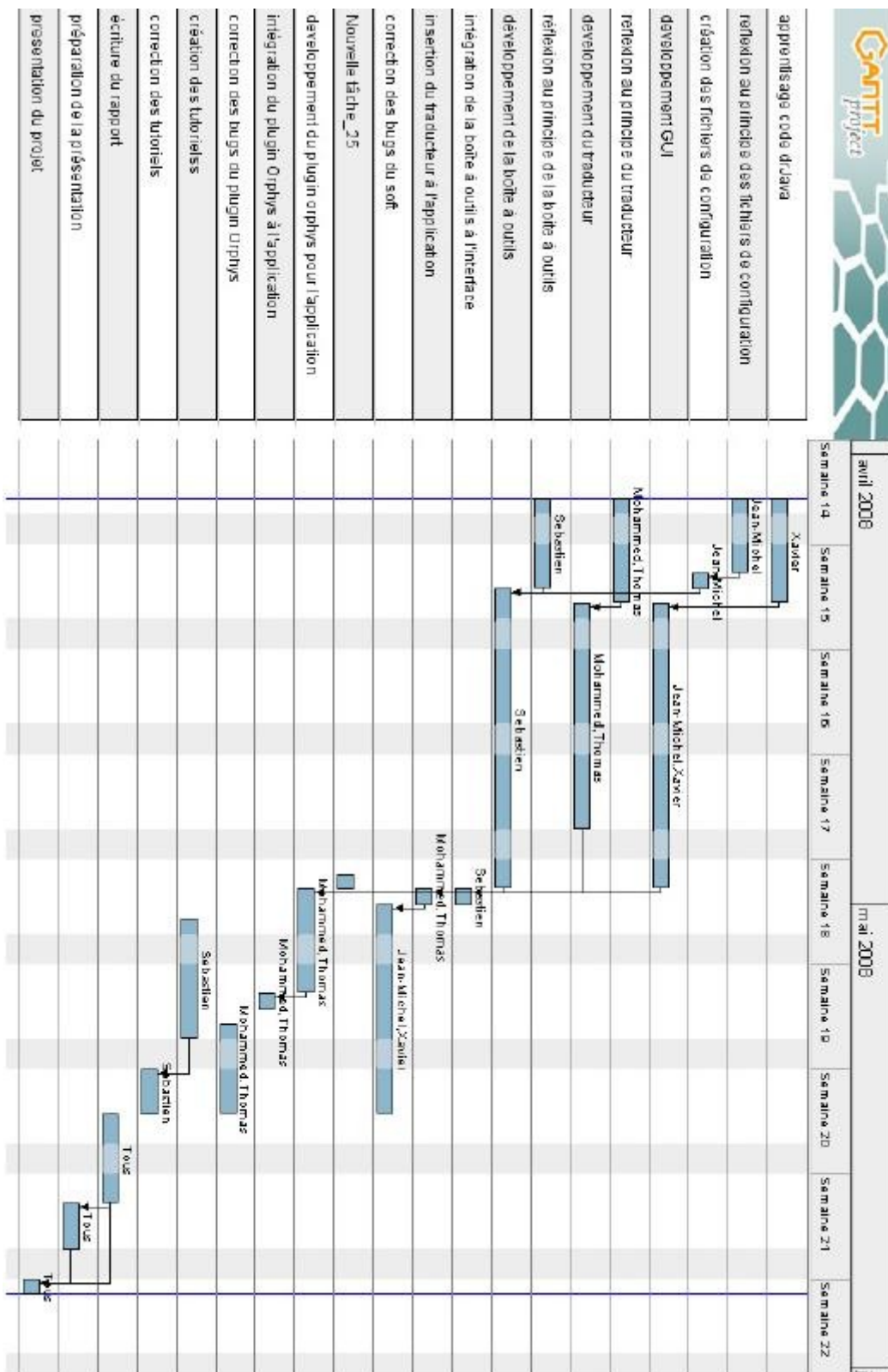
- <http://www-sop.inria.fr/oasis/Denis/ProgRpt/>

Plugins et RCP Eclipse :

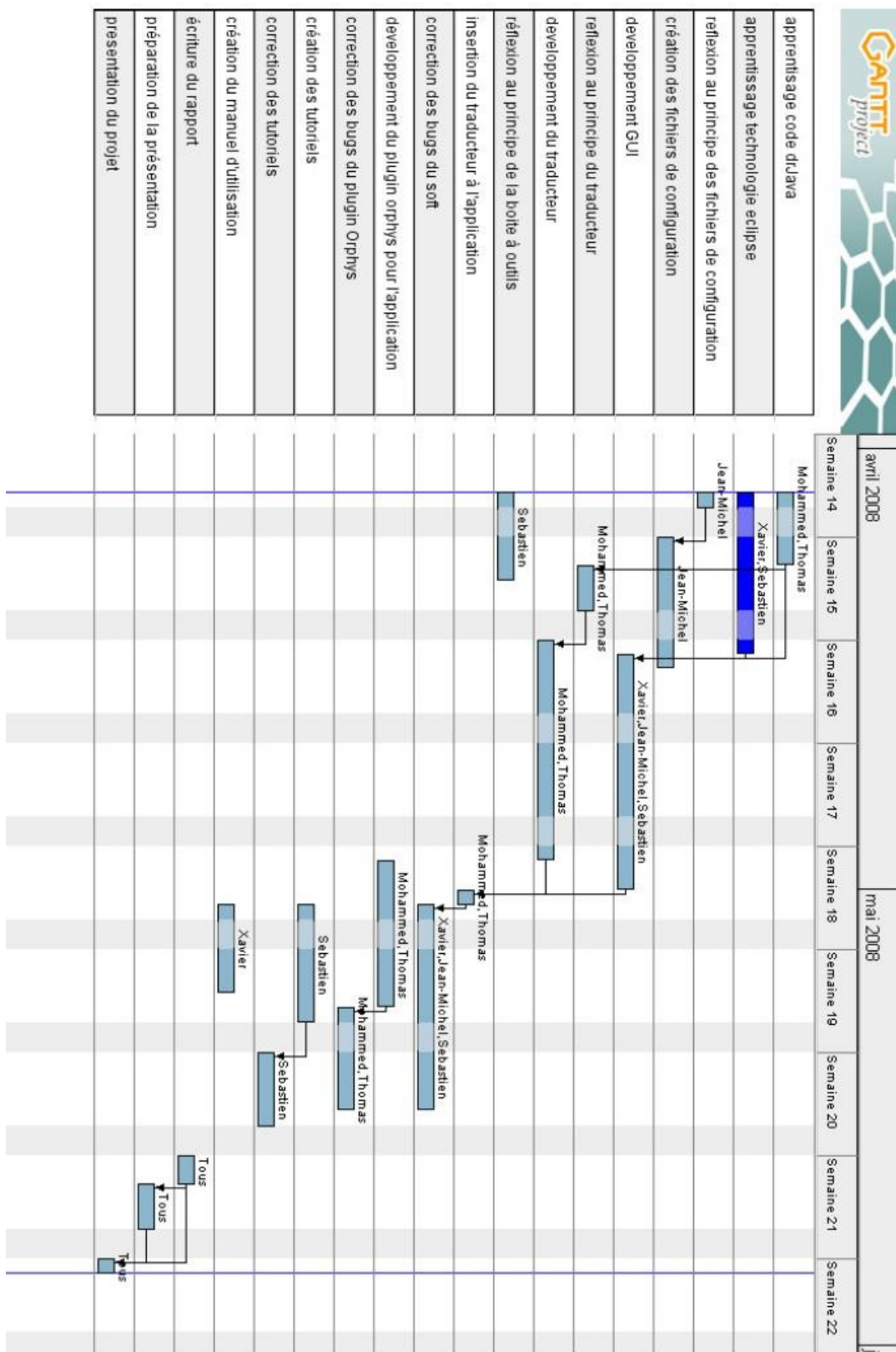
- <http://books.google.com/books?id=6Ob1ANNVcXcC&printsec=frontcover&hl=fr#PPA635,M1>
- <http://books.google.com/books?id=6Ob1ANNVcXcC&printsec=frontcover&hl=fr#PPP1,M1>
- <http://jmolieredeveloppez.com/tutoriel/java/eclipse/plugin/>
- <http://wiki.improve.fr/wiki/moni/>
- <http://www-igm.univ-mlv.fr/~dr/XPOSE2004/vforel/liens.html>
- http://www.cetic.be/internal.php3?id_article=224
- <http://www.developpez.net/forums/>
- <http://www.eclipse.org>
- <http://www.eclipsepluginsite.com/>
- <http://www.eclipsetotale.com/>
- <http://www.java2s.com/Code/Java/SWT-JFace-Eclipse/Thisclasscreatesacomplexttoolbar.htm>
- <http://www.labo-sun.com/resource-fr-articles-1235-0-eclipse-gui-introduction-a-eclipse-rcp.htm>
- <http://www.serli.com/blog/index.php/2007/12/08/7-coloration-syntaxique-avec-eclipse-rcp>
- <http://www.talient.fr/?q=node/12>
- http://www.vogella.de/articles/RichClientPlatform/article.html#editor_s1a

ANNEXE

Planning d'organisation prévu :



Planning d'organisation réel :



Évolution du travail du groupe :

Phase 1 : découverte du sujet, prise de contact avec les encadrants et développement du Cahier Des Charges :

Semaine 1 (du 04/02/2008 au 10/02/2008):

- prise de connaissance du sujet.
- rencontre entre les membres du groupe pour faire connaissance et définir chacun les attentes qu'il a du projet et pourquoi il a choisit ce sujet.

Semaine 2 (du 11/02/2008 au 17/02/2008):

Lors de cette semaine, nous avons étudié le sujet, puis nous avons pris contact par email avec nos encadrants afin de nous présenter et de décider d'une date de rendez-vous pour mieux connaître leurs attentes.

Semaine 3 (du 18/02/2008 au 24/02/2008):

Lors de cette semaine nous avons rencontré Thierry Vieville en petit groupe de 2 et 3 (nos emplois du temps ne nous permettant pas de tous y aller en même temps).

Pendant ce premier rendez-vous, nous nous sommes présentés de manière succincte et avons expliqué nos motivations concernant le choix de ce sujet. Thierry Viéville a ensuite expliqué la vision qu'il en avait ainsi que ses attentes. Nous avons partagé nos premières impressions ainsi que les idées qui pourraient répondre à cette demande.

Semaine 4 (du 25/02/2008 au 02/03/2008):

Lors de cette semaine certains étudiants de l'équipe (les autres ne pouvaient pas en raison de l'incompatibilité de leur emploi du temps avec celui des encadrants) ont rencontrés Laurent Mongiat (professeur de MPI au lycée Valbonne) pour connaître ses attentes exactes par rapport au sujet du TER. Après cette rencontre, il est apparu une grande divergence (du moins sur certains points) entre ses attentes et celles que nous avait définies Thierry Viéville. Nous avons donc pris rendez-vous avec les deux encadrants simultanément la semaine prochaine pour éclaircir leurs attentes.

Nous avons également commencé à rédiger le cahier des charges, mais n'avons pas pu en compléter de nombreux points en raison de l'ambiguïté qui règne autour de notre sujet.

Semaine 5 (du 03/03/2008 au 09/03/2008):

Lors de cette semaine nous nous sommes tous rencontrés (encadrants et étudiants) pour la première fois, de manière à éclaircir les zones d'ombres qui entouraient notre sujet. Lors de cette réunion nous avons délimité les limites exactes de notre sujet et les attentes précises du client (c-a-d Laurent Mongiat). Puis chacun de notre côté nous avons commencé à écrire les informations et idées qui nous paraissaient importantes pour le cahier des charges. La semaine prochaine nous (les étudiants) devons nous rencontrer pour réunir les informations de manière à rédiger le cahier des charges de manière précise, et également rencontrer Thierry Viéville, à qui nous espérons pouvoir montrer une première version de notre cahier des charges.

Semaine 6 (du 10/03/2008 au 16/03/2008):

Comme prévu nous avons rencontré nos deux encadrants du TER pour éclaircir le sujet et en définir les limites exactes de manière à pouvoir réaliser le cahier des charges du projet. Suite à ce rendez-vous nous avons chacun réalisé une partie du cahier des charges, nous nous sommes ensuite réunis pour fusionner les différentes parties du cahier des charges et effectuer une lecture complète. Nous avons ensuite soumis cette première version du cahier des charges à nos encadrants.

Semaine 7 (du 17/03/2008 au 23/03/2008):

Nous avons corrigé le cahier des charges en fonction des corrections apportées par nos encadrants. La version de notre cahier des charges est quasiment finale. Nous avons également contacté Christian rellier (<http://christian.rellier.free.fr/orphy/ORPHYGTS.htm>) qui a développé un driver en c++ pour la machine Orphy GTS de manière à savoir si il accepte que nous le traduisions en Java pour l'intégrer à notre soft. Actuellement nous sommes dans l'attente de sa réponse.

Semaine 8 (du 24/03/2008 au 30/03/2008):

Étant en période de partiels cette semaine nous ne travaillerons pas sur le TER.

Phase 2 : Travail à plein temps :

Semaine 1 (du 31/03/2008 au 06/04/2008):

Lors de cette semaine nous avons finalisé le cahier des charges et avons préparé la pré-soutenance de notre TER puis nous avons présenté la pré-soutenance du projet, après celle-ci, nous avons décidé de nous baser sur le moteur d'Eclipse et développement de plugins eclipse pour créer l'interface graphique de notre projet. Nous avons donc dû revoir un peu notre planning de manière à consacrer du temps à l'apprentissage au développement de plugin pour eclipse et d'applications RCP eclipse.

Semaine 2 (du 07/04/2008 au 13/04/2008):

Nous avons rencontré notre encadrant Thierry Viéville pour faire des choix d'implémentations par rapport à notre application. Nous avons profité de notre déplacement à l'INRIA pour rencontrer l'équipe de Fabrice Huet en particulier Johann Fradj qui nous a expliqué brièvement le fonctionnement du noyau d'eclipse. Ils nous ont montré également quelques applications qu'ils ont réalisées avec le noyau d'Eclipse. Ils nous ont également donné des liens et des sources pour nous documenter sur la librairie SWT d'Eclipse.

Nous avons appris le développement de plugin eclipse et d'applications RCP. Certains se sont consacrés à récupérer le code présent dans [DrJava²](#) qui pourrait nous être utile.

Nous avons commencé à développer notre interface graphique, à développer le traducteur de code en code Java, ainsi que la création des fichiers de configuration.

Semaine 3 (du 14/04/2008 au 20/04/2008):

Cette semaine nous avons continué à développer l'interface graphique de notre application, ainsi que les fichiers de configurations (ainsi que le code qui permet de récupérer les informations présentes dans ces fichiers) et également le traducteur de code source.

Semaine 4 (du 21/04/2008 au 27/04/2008):

Lors de cette semaine nous avons continué à développer l'interface utilisateur de notre application ainsi que les plugins qui y sont associés comme le traducteur de code, ou encore la traduction de la console, ... En fin de semaine nous sommes arrivés à une version quasiment Beta que nous présenterons la semaine prochaine à nos encadrants notamment à Laurent Mongiat (professeur de lycée). Nous nous sommes également renseignés sur l'interfaçage de la machine Orphy GTS2 avec notre application, en particulier la manière d'envoyer et recevoir des données via la machine et l'interface.

Semaine 5 (du 28/04/2008 au 04/05/2008):

Lors de cette semaine un groupe a développé le manuel d'utilisation de notre application, ainsi que les tutoriels pour maîtriser notre application ainsi que les rudiments du langage JavasCool. Un autre groupe s'est consacré à la réalisation du plugin Orphy pour notre application. enfin un dernier groupe a commencé à corriger les bugs de notre application. par rapport au temps de développement qui nous reste et le travail restant à effectuer nous ne sommes pas en retard.

Semaine 6 (du 05/05/2008 au 11/05/2008):

Lors de cette semaine nous avons essentiellement corrigé les bugs de notre application et développé le plugin pour le machine Orphy GTS2. Nous avons également tenté une installation de notre application sur les machines du lycée de valbonne et une utilisation de notre logiciel sur celle-ci. L'installation s'est bien déroulée en revanche nous avons rencontré des Bugs lors de l'utilisation que nous avons pu identifier et corriger.

Au niveau du planning de notre application nous sommes dans les temps, avec peut être un léger retard au niveau du plugin Orphy. La semaine prochaine nous allons retenter une installation et utilisation de notre application au lycée de valbonne. et nous espérons que notre encadrant Laurent Mongiat (professeur au lycée) pourra utiliser notre logiciel lors de sa séance de TP du vendredi 16 mai.

Semaine 7 (du 12/05/2008 au 18/05/2008):

Lors de cette semaine nous nous sommes déplacés plusieurs fois sur le lycée de Valbonne pour tester notre application sur les ordinateurs du lycée. Nous avons passé également cette semaine à corriger le BUGS de notre application. A la fin de la semaine la plupart des BUGS recensé à ce jours ont été corrigés. Parallèlement cette semaine nous avons continué à développer le plugin Orphy pour notre application.

Par rapport à notre emploi du temps nous sommes dans les temps. La semaine prochaine nous allons la consacrer à écrire le rapport ainsi qu'à préparer notre oral.

Semaine 8 (du 19/05/2008 au 25/05/2008):

Cette semaine entièrement été consacré à la réalisation du rapport du TER ainsi qu'a la préparation de de notre oral (préparation des transparents et entraînement).

Évolution du travail de Chalmeton Sébastien :

Semaine 1 (du 31/03/2008 au 06/04/2008):

Lundi, mardi, mercredi, jeudi : lors de ces 4 jours j'ai finalisé le cahier des charges avec mes camarades et nous avons également préparés les transparents pour la pré-soutenance. Nous nous sommes également entraîné pour la présentation.

Vendredi : j'ai passé l'oral de pré-soutenance avec mes camarades. suite à cet oral quelques échanges d'idées avec le jury nous ont permit de remettre en question (de manière positive) la plateforme que nous allons utiliser pour notre application.

Deux logiciels existants, semble correspondre à nos attentes : "Eclipse" et "DrJava", nous nous sommes donc divisés en deux groupes (un groupe travail sur le code source de DrJava

et un autre groupe travail sur Eclipse en particulier sur le coté RCP). Pour ma part je m'occupe de comprendre le coté RCP d'Eclipse.

Samedi : en suivant différents cours et différents tutoriels sur les RCP j'essaie de créer un RCP correspond à nos attentes. par exemple : <http://mbaron.developpez.com/eclipse/intro/>
http://blog.developpez.com/index.php?blog=12&title=forums_plugins_eclipse_et_applications_r&more=1&c=1&tb=1&pb=1

Dimanche : J'ai étudié les librairies SWT et Jface, et fait quelques tutoriels sur le développement de plugins pour Eclipse et d'application RCP.

Semaine 2 (du 07/04/2008 au 13/04/2008):

Lundi : Nous avons rencontrés notre encadrant "Thierry Vieville" pour faire des choix d'implémentations par rapport à notre application. Nous avons profité de notre déplacement à l'INRIA pour rencontrer l'équipe de "Fabrice Huet" en particulier "Johann Fradj" qui nous à expliqué brièvement le fonctionnement du noyau d'Eclipse. Ils nous ont montrés également quelques applications qu'ils ont réalisés avec le noyau d'Eclipse. Ils nous également donné des liens et des source pour nous documenter sur la librairie SWT d'Eclipse.

Mardi : Lors de cette journée j'ai développé le plugin de notre application correspondant à l'éditeur et comportant la coloration syntaxique. ce plugin sera à intégrer ultérieurement dans notre application RCP.

j'ai également commencé à développer le plugin correspondant à la boîte à outils (boîte contenant des raccourcis pour certaines fonction).

Mercredi : Ce jour, j'ai continué le développement du plugin correspond à la boîte à outils. j'ai décidé d'intégrer directement le plugin de la boîte à outils dans le plugin de notre éditeur de code JavasCool développé précédemment. c'est à dire que des deux plugin on n'en fera qu'un seul regroupant les deux fonctionnalités, cette décision à été prise par le fait que la boîte outils est un utilitaire propre à notre éditeur de code.

Jeudi : Aujourd'hui j'ai continué le développement du plugin Eclipse qui permet d'éditer du code JavasCool et la boîte à outils qui va avec.

Vendredi : Ce jour j'ai continué comme le jour précédent à développer l'éditeur et la boîte à outils. j'ai également essayé d'intégrer une version Beta du traducteur de code (code JavasCool vers code java) à notre éditeur pour la fonction compiler. nous avons également essayé d'inclure le plugin que j'ai développé à un RCP mais nous n'avons pas réussi à l'intégrer complètement.

Semaine 3 (du 14/04/2008 au 20/04/2008):

Lundi, Mardi, Mercredi, Jeudi, Vendredi : J'ai consacré ces jours à continuer à développer une partie de l'interface graphique de notre application. Nous avons également développés un RCP pour pouvoir y inclure les différents plugins que nous avons développés en parallèle.

Samedi, Dimanche : j'ai consacré ces deux jours à continuer à développer la boîte à outils de l'application, en particulier à apprendre à ajouter des boutons (comme celui de Run de Eclipse) dans la barre des boutons d'un RCP

Après ces deux jours j'ai quasiment terminé le développement de la boîte à outils.

Semaine 4 (du 21/04/2008 au 27/04/2008):

Lundi : Aujourd'hui j'ai terminé la boîte à outils de notre application, elle est maintenant complètement opérationnelle et d'après les quelques tests effectués semble stable, il sera tout de même nécessaire de vérifier de nouveau sa "stabilité" une fois l'application complètement développée. Je passe maintenant au développement des trois boutons : "compilation" "exécution" et "stop" (bouton d'arrêt du compilateur en cas de boucle infinie).

Mardi : ce jour, j'ai continué le développement des trois boutons : "compiler", "exécuter", "Stop"(arrêt du compilateur). notre application est maintenant capable de compiler du code source "java" et "jvs" écrit dans notre éditeur.

Mercredi, Jeudi, Vendredi : lors de ces trois jours je me suis documenté sur la manière de réaliser une page de préférence pour gérer dans notre application RCP les préférences tel que la coloration syntaxique, la taille d'écriture... Et je développe ces pages de préférences pour notre RCP.

Samedi, Dimanche : Lors de ces deux jours j'ai continué à développer la page de préférence pour les choix de coloration syntaxique et d'écriture de notre éditeur. Maintenant cette page de préférence est fonctionnelle, il n'y a plus qu'un petit "Bug" : pour que les préférences soient prises en compte il faut redémarrer l'application (les modifications ne se font pas à chaud).

Semaine 5 (du 28/04/2008 au 04/05/2008):

Lundi : Nous nous sommes réunis pour faire un point sur là où nous en étions, si nous étions en retard ou non, et nous avons attribué à chacun une nouvelle tâche. Pour ma part la tâche qui m'a été attribuée est la réalisation du manuel d'utilisation et des TPs/tutoriels de notre application. (tâche en commun avec Xavier Richter) Avec Xavier nous nous sommes répartis les tâches en deux : un qui fait les manuels et l'autre les tps/tutoriels.

J'ai donc commencé à réaliser les TPs/tutoriels.

Mardi : J'ai continué le développement des TPs/tutoriels correspondant à notre application

Mercredi : ce jour j'ai continué comme les jours précédents à écrire des Tps/tutoriel en relation avec notre application et à la demande de notre encadrant, j'ai également corrigé quelques bugs de notre application notamment sur le compilateur.

jeudi : Ce jour j'ai continué à développer des TPs/tutoriels pour notre application comme les jours précédents. Puis j'ai corrigé quelques bugs sur l'application notamment sur l'effacement de la console ainsi que sur les pages de préférences qui étaient inutiles et dont l'utilisateur avait accès.

Vendredi : Aujourd'hui nous avons rencontré nos deux encadrants pour leur présenter une version plus ou moins BETA de notre application, nous leur avons également demandé si notre application correspond bien à leurs attentes, ce qui est le cas. J'ai profité de cette réunion pour montrer à Laurent Mongiat une ébauche des tutoriels que j'ai développés pour notre application, pour savoir si au niveau pédagogique ils étaient correctes, il en est d'ailleurs ressorti qu'ils étaient trop élevés pour les jeunes lycéens.

Ensuite j'ai revu mes tutoriels pour bien les adapter aux attentes de Laurent Mongiat. avec l'aide de Jean-Michel j'ai mis notre projet en version jdk1.6 car notre encadrant s'était trompé sur la version qu'ils avaient à leur lycée. Heureusement il n'y avait pas beaucoup de modifications à apporter à nos sources. j'ai ensuite créé un nouveau plugin pour notre application qui permet d'avoir les tutoriels directement dans notre application à l'aide du menu "help".

Samedi, dimanche : Ces deux jours j'ai continué le développement des tutoriels et de leurs corrections, j'en ai créé un nombre de 9, il nous en était demandé un minimum de 5. J'ai également corrigé quelques Bugs de notre application notamment la sauvegarde automatique lors de la compilation ainsi qu'un problème de coloration syntaxique des commentaires

Semaine 6 (du 05/05/2008 au 11/05/2008):

Lundi : Aujourd'hui j'ai continué le développement des tutoriels avec leurs corrections, ils sont finis ils ne faut plus que j'aille les faire valider par notre encadrant Laurent Mongiat. J'ai d'ailleurs pris rendez-vous avec lui pour ce mercredi pour lui présenter ces tutoriels et essayer d'installer notre soft sur leur ordinateur, nous attendons confirmation de sa part pour ce rendez-vous. j'ai ensuite fait des recherches pour trouver le moyen de charger à "chaud" les modifications effectués sur la page de préférence de la coloration syntaxique.

Mardi : Ce jour, j'ai corrigé quelques bugs sur l'application, ainsi que chercher le moyen de changer à chaud les préférences choisit par l'utilisateur.

Mercredi : Aujourd'hui je me suis rendu avec Jean-Michel Guillaume au lycée de Valbonne pour tenter une installation de notre application sur les machines du lycée ainsi que la vérification du fonctionnement de l'application. Suite à cet essai l'installation s'est déroulée correctement mais au niveau de l'utilisation nous avons rencontré des bugs au niveau de la compilation et de l'exécution que nous allons donc nous efforcer de résoudre. De même j'ai discuté des tutoriels que je suis chargé d'écrire, il en ressortit que je devais en écrire un supplémentaire pour l'ouverture et la lecture de fichier texte.

Jeudi : Aujourd'hui avec Xavier Richter nous avons cherché à comprendre d'où vient le bug pour la compilation que nous avons eu lors du test avec Jean-Michel Guillaume au lycée de Valbonne. Nous avons réussi à identifier ce bug, il s'agit en fait du fait que la méthode utilisée pour la compilation n'est pas dans le JRE mais dans le JDK. Donc pour que notre application soit utilisable il faut installer sur la machine hôte le JDK 1.6.

Vendredi : Aujourd'hui avec Xavier Richter nous avons décidé de corriger les bugs du traducteur qui traduit le code JavasCool en fichier java. Nous avons identifié tellement de bugs que nous avons décidé de reprendre le code du traducteur à 0, nous nous sommes repartit les tâches en deux avec Xavier et avons commencé à réécrire le traducteur. J'ai également corrigé le bug sur la gestion de préférences pour que les modifications soient effectuées à "chaud", ceci marche correctement maintenant.

Samedi, Dimanche : Lors de ces deux jours j'ai continué à écrire le code correspondant au traducteur de code JavasCool vers java, j'ai également pris contact avec nos enseignants de manière à pouvoir retourner au lycée de Valbonne dès ce mardi pour effectuer des tests en conditions réelles. J'en ai profité pour corriger quelques bugs minimum de l'application que je trouve au fur et à mesure.

Semaine 7 (du 12/05/2008 au 18/05/2008):

Lundi : Nous rentrons dans les dernières phases de notre application, pour ma part aujourd'hui j'ai continué comme les trois jours précédents à corriger les bugs que je trouve sur l'application ainsi qu'à réécrire une partie du traducteur. Demain avec Jean-Michel nous nous rendons au lycée de Valbonne pour tester en conditions réelles notre application. Nous espérons que les bugs que nous avons identifiés lors de notre dernière tentative d'utilisation sur les plateformes du lycée auront été réglés de manière correcte et optimale.

Mardi : Aujourd'hui Jean-Michel et moi-même nous nous sommes rendu pour la journée au lycée de Valbonne pour installer le logiciel sur les ordinateurs du lycée pour voir si les bugs identifiés lors de la dernière installation ont bien été résolus, ce qui est d'ailleurs le cas. Au cours du test du logiciel nous avons rencontrés de nouveau bugs mais minimaux cette fois-ci. J'ai profité de la rencontre avec le professeur pour récupérer ses corrections vis à vis des tutoriels que j'ai écrit pour apprendre à programmer en JavasCool. J'ai ensuite corrigé quelques bugs en rapport avec notre traducteur. Ensuite j'ai corrigé des bugs présents dans le traducteur.

Mercredi, Jeudi : Lors de ces deux jours j'ai corrigé encore quelques bugs de notre traducteur, j'ai également corrigé quelques fautes apparaissant sur les tutoriels, puis j'ai écrit de nouvelles macros pour notre boîte à outils qui permettent la conversion d'un nombre dans différentes base(décimal, hexadécimal, binaire) suite à la demande de notre encadrant Laurent Mongiat.

Vendredi : Aujourd'hui je suis monté avec Thomas Lebrun au lycée de valbonne pour tester notre application sur leurs ordinateurs et faire une installation les machines du lycée. quant à Thomas il a montré le plugin Orphy développé pour notre application, Laurent Mongiat semble enchanté du résultat. J'ai également corrigé quelques BUGS identifiés sur notre application.

Samedi, Dimanche : j'ai passé ces deux jours à chercher des bugs sur notre application et à les résoudre, jusqu'à présent tous ceux identifiés ont put être résolus. j'ai également corrigé les erreurs qui s'étaient glissés dans les tutoriels et leurs corrections. Ainsi qu'à modifier quelques détails de pédagogies sur les tutoriels.

Semaine 8 (du 19/05/2008 au 25/05/2008):

lundi : Cette journée je suis monté avec Jean-Michel au lycée de Valbonne pour voir comment se comporte notre application sur les machines du lycée et pour voir si tous les bugs qui avaient été identifiés précédemment, ont correctement été résolus.

mardi, mercredi, jeudi : j'ai consacré ces trois jours à écrire une partie du rapport ainsi qu'a préparer les transparents pour l'oral.

Vendredi : lors de cette journée je suis monté avec l'équipe au lycée pour réaliser un TP avec les lycéen, sur notre application. puis nous nous sommes entraînés pour l'oral.

Évolution du travail de Ennabli Mohammed :

Semaine 1 (du 31/03/2008 au 06/04/2008):

Pendant cette semaine, on a cherché à comprendre le code de DrJava afin d'extraire les parties intéressantes pour notre projet. J'ai fait des recherches aussi sur le fonctionnement des parseurs de fichiers (`StreamTokenizer`). J'ai commencé à écrire le code du traducteur.

Semaine 2 (du 07/04/2008 au 14/04/2008):

Implémentation du traducteur: J'ai trouvé deux façons de procéder afin d'implémenter le traducteur qui traduit du code jvs (JavasCool) en code java. J'ai choisi d'utiliser `StreamTokenizer` afin de parser le fichier "jvs ». La traduction commence.

Semaine 3 (du 14/04/2008 au 20/04/2008):

Lors de la semaine passée, j'ai réussi à ajouter les déclarations de variable (public static). Quelques bugs ont apparu. Rapport des bugs: déclaration de variables à l'intérieur des méthodes comme étant public static. traduction des macros. Problème de parsing des nombres. Il n'existe pas un moyen de savoir si le nombre lu est un entier ou un double etc.

Semaine 4 (du 21/04/2008 au 27/04/2008):

La traduction des structures de données comme les tableaux n'a pas encore été effectuée. J'ai réussi à traduire les tableaux du fichier jvs en `ArrayList`. Petit bug trouvé lors de la traduction des tableaux d'entiers (int à la place de Integer). Je suis passé à la traduction des appels de tableaux et d'affectations. Travail réussi après quelques complications. La traduction de la méthode main est fait. Ceci est insuffisant car les exceptions qui seront produites par le programme ne pourront pas être traitées car la traduction ne comporte pas de bloc try..catch. Ajout des blocs try catch.

Quelques complications ont rendu le travail un peu difficile vue qu'il faudra distinguer la méthode main des autres fonctions. Lors des tests effectués sur cette partie, le traducteur a montré quelques bugs au niveau de la traduction (try....catch). Un autre problème a fait son apparition qui est le return utilisé dans les méthodes ou dans le main. J'ai choisi d'utiliser un cast juste avant l'instruction qui comporte le return. Ce choix n'est pas juste vu qu'il y a une possibilité qu'une erreur de programmation sera ignoré à cause du cast. Ce problème majeur a rendu l'utilisation du traducteur impossible pour notre application. Abandon du traducteur à l'aide du streamTokenizer. Il faudra trouver une autre solution.

Semaine 5 (du 28/05/2008 au 04/05/2008):

Après une courte réflexion, j'ai choisi d'utiliser les expressions régulières et un StringTokenizer afin de coder le traducteur. Au bout de 2 jours j'ai réussi à avoir une version +- stable du traducteur. Un problème est apparu. Le traducteur fonctionne sur mon ordinateur mais pas sur ceux de mes collègues. J'ai continué à travailler sur le traducteur et j'ai obtenu une version à 90% stable. Deux petits bugs. L'ajout de public static devant chaque déclaration de variable après une déclaration d'une ou plusieurs méthodes. Mes collègues n'arrivent toujours pas à faire marcher le traducteur dans l'application. Problème incompréhensible.

Semaine 6 (du 05/04/2008 au 11/04/2008):

Cette semaine, j'ai arrêté de travailler sur le traducteur. L'interfacage de la machine ORPHY GTSII a commencé. Après des recherches, plusieurs librairies ont été trouvées (javax.usb, javax.comm, JCommUsb payante). Un essai a été effectué à l'aide de la librairie javax.usb. On arrive à détecter la machine ainsi que toutes ces caractéristiques. Vu qu'une configuration usb de la machine par le constructeur n'a pas été prévue, il est impossible d'envoyer et de récupérer des informations sur le contenu des entrées sorties de la machine (Abandon de la librairie usb). Mon collègue Thomas a réussi à récupérer les valeurs d'une des sorties analogiques à l'aide de la librairie javax.comm qui permet de programmer des interfaces utilisant le port série. Un plugin eclipse a été notamment développé afin d'intégrer l'interfacage avec la machine dans l'application.

Semaine 7 (du 12/05/08 au 18/05/08):

L'objectif de cette semaine est de réaliser la détection automatique du port série sur lequel sera branchée Orphy et l'acquisition paramétrée sur la machine Orphy GTS II. Détection automatique: On a réussi à détecter le port automatiquement mais un problème avec la jvm est apparu. La jvm ne s'arrête pas après la détection du port. Comme ce problème n'a pas été résolu, on a abandonné l'idée de la détection automatique du port et on s'est contenté de demander à l'utilisateur de donner le port de communication sur lequel la machine a été branchée. Acquisition paramétrée: On a réussi l'ajout de l'acquisition paramétrée à l'interface d'Orphy ce qui va permettre à l'utilisateur de configurer le mode d'acquisition. L'interface d'Orphy est complètement fonctionnelle. L'objectif est atteint, arrêt du travail sur la machine Orphy.

Semaine 8 (du 19/05/08 au 23/05/08):

Cette semaine, on a commencé par l'écriture du rapport. J'ai fini l'écriture du manuel d'Orphy et on a préparé les diapos de la soutenance du projet. L'installation du logiciel sur les machines du lycée est prévu pour vendredi avec le professeur Laurent Mongiat.

Évolution du travail de Guillaume Jean-Michel :

Semaine 1 (du 31/03/2008 au 06/04/2008):

Nous avons profité du début de semaine pour travailler sur notre pré-soutenance de vendredi.

Semaine prochaine: Pendant la présentation nous avons eu droit à des remarques très intéressantes nous conseillant de revoir notre stratégie de travail car nous nous orientons vers une voie bien trop compliquée, il serait donc intéressant de reparler de ça avec nos encadrants en début de semaine pour peut-être s'organiser différemment.

Semaine 2 (du 07/04/2008 au 13/04/2008):

Pendant le weekend, nous nous sommes séparés en 2 groupes pour nous permettre de travailler en parallèle sur les différentes stratégies possibles pour notre projet. Après la réunion du Lundi 7 Avril avec Thierry Vieville, nous avons décidé de suivre ce schéma jusqu'à la fin de semaine. Mon travail personnel consiste à extraire le maximum d'utilitaires possibles des librairies mise à notre disposition par le logiciel Dr-Java...

Semaine prochaine: Enfin de semaine, nous sommes arrivés à la conclusion qu'il serait beaucoup plus profitable de revoir nos stratégies initiales et de nous orienter finalement vers un développement de logiciel RCP via l'outil eclipse. Mon travail de la semaine étant, concluant en un sens mais décevant dans un autre, a mis en évidence l'incompatibilité des librairies offertes par Dr Java avec nos attentes, ces librairies qui répondaient à des besoins beaucoup trop spécifiques, très peu extensibles, pas du tout indépendantes (et dont la plupart des sources provenant d'autres projets étaient manquantes)...La semaine prochaine je dois donc développer une première librairie qui permettra la lecture et l'interprétation des fichiers de configurations et l'exporter sous forme de plugin eclipse, et dans un second temps, m'atteler à la réalisation d'un autre plugin permettant la construction dynamique d'une boîte à outils de répertoires pour notre futur logiciel...

Semaine 3 (du 14/04/2008 au 20/04/2008):

J'ai tenté d'implémenter le plugin délivrant les outils de lecture de configuration le plus tôt possible car d'autres personnes en étaient dépendantes, j'ai donc commencé l'implémentation ce Lundi et l'ai fini mardi en soirée. J'ai commencé le développement du plugin pour répertoires Mercredi matin. Le plugin outils consiste en fait en la réalisation d'un menu et d'une toolbar permettant la modification de ces répertoires favoris, la navigation à travers ceux-ci, et la possibilité d'enregistrer les fichiers en cours de modification directement dans l'un d'entre eux. En soit, cela ne doit pas représenter quelque chose de difficile, mais l'implémenter à l'aide d'une api de plusieurs milliers de classes où certaines d'entre elles implémentent parfois une vingtaine d'interfaces s'avère être une tâche très dure. J'ai donc poursuivi l'implémentation de cette boîte à outils jusqu'à la fin de semaine.

Semaine prochaine: Le plugin répertoires n'est pas totalement terminé, il faut qu'il soit fini le plus vite possible pour pouvoir passer en phase de test sur des bêta versions en condition réelles.

Semaine 4 (du 21/04/2008 au 27/04/2008):

Lundi et Mardi matin, j'ai fini d'implémenter le plugin de management des dossiers favoris. À partir de Mardi après-midi je me suis attaché à l'implémentation de l'exécution de code depuis notre RCP. J'ai opté pour l'instanciation d'une nouvelle JVM faisant tourner un serveur via RMI, cela me permet d'avoir un bon répondant pour "killer" les programmes bouclants et permet également de répondre à la partie du cahier des charges qui voulait que l'on puisse permettre l'observation de variables internes au programme pendant son exécution.

Semaine prochaine: Cette dernière tâche est pour moi complexe, je ne maîtrise pas du tout la librairie RMI, je dois allouer énormément de ressources (notamment pour les redirections de flux I/O pour les programmes esclaves et serveurs), je dois donc m'assurer que aucune erreurs ne remonte jusqu'en haut de la pile d'exécution et dois également veiller à ne pas encombrer le système hôte avec tout un tas de process morts. Ceci doit être réalisé de façon méticuleuse, je passerai donc le début de semaine sur la production de code robuste et propre pour cette méthode particulière d'exécution de code. Nous devrions ensuite entrer dans une phase de développement particulière qui ne vise au déploiement du RCP, à son internationalisation, au développement du plugin Orphy, et aux tests...

Semaine 5 (du 28/04/2008 au 04/05/2008):

En début de semaine j'ai continué à travailler sur l'exécution de code depuis notre logiciel. Je n'ai aucun moyen de tester la portabilité de ce que j'ai produit, donc je cherche au maximum pour éviter à mes camarades de se retrouver avec des surprises une fois mon travail mis en commun sur le serveur de versions. Cela constitue une grosse masse de travail en terme de recherches car peux de conversations/tutos/docs traitent du sujet. A partir de Mercredi je suis tout seul à travailler sur le déploiement de notre logiciel, je dois pour cela "tunner" mon IDE pour lui permettre d'exporter notre travail sur les plateformes cibles.

Semaine prochaine: Nous sommes censé tenter une première installation en milieu de semaine au lycée, il faut tenter d'anticiper au maximum les problèmes que nous pourrions rencontrer. Je devrai ensuite entrer en pseudo phase test/développement pour mettre en évidence ce qui ne va pas, et éventuellement améliorer ce qui pourrait l'être...

Semaine 6 (du 05/04/2008 au 11/05/2008):

Nous avons finalement tenté l'installation en début de semaine et il s'avère que tout n'est pas comme il le devrait... En fait, nous devons au début développer pour du java 1.5 mais il s'avère que les machines sont équipées de la version 6. De plus, le réseau du lycée est géré par Samba et nous donne un peu de fil à retordre. Pour assurer l'exécution d'un programme et permettre les fonctionnalités décidées par nos encadrants, j'ai décidé d'utiliser une classe "serveur" et de l'interroger par RMI. Il se trouve que la lenteur de leur réseau et les réglages de bases que j'avais établi mènent à une application tournant au ralenti... Jusqu'à la fin de semaine je dois travailler sur ce problème et en même temps commencer l'internationalisation de l'application.

Semaine prochaine: Nous avons à nouveau une journée au lycée pour nous consacrer à l'installation du logiciel et aux problèmes que nous pourrions rencontrer. Nous devrions ensuite travailler sur les finitions au niveau des logs pour les catch d'exceptions et les derniers détails d'interface.

Semaine 7 (du 12/04/2008 au 18/05/2008):

Ce mardi nous avons réussi une première installation sur les postes du lycée VSA. Je me consacre maintenant entièrement à la correction de bugs et tests. Je termine également les derniers détails liés à l'installation et la distribution (Licence GNU GPL).

Semaine prochaine: Entièrement consacrée à l'écriture du rapport.

Semaine 8 (du 19/04/2008 au 23/05/2008):

Tout le début de la semaine est consacré à la rédaction du rapport et à la présentation. Jeudi nous préparons une version définitive du logiciel pour l'installation sur les machines du lycée VSA le lendemain. Vendredi, nous procédons donc à l'installation ainsi qu'à une séance de TP pour constater l'utilisation du programme.

Évolution du travail de Lebrun Thomas :

SEMAINE 1 :

Pré-soutenance du cahier des charges, une nouvelle orientation du projet à été proposée par le jury concernant la structure de départ de l'application. Nous avions prévu d'étendre l'API DrJava, nous allons maintenant envisager d'utiliser eclipse RCP, nous avons créé 2 groupes pour le weekend, un chargé d'étudier l'option eclipse, et le second l'option DrJava.

SEMAINE 2 :

Nous avons rencontré notre encadrant de TER Thierry Vieville, il a approuvé l'utilisation de eclipse, nous avons par la même occasion rencontré l'équipe de Fabrice Huet a l'inria qui nous a fait une démonstration des possibilités d'Eclipse. Prises de contact avec Christian Rellier(une des personne créatrice d'ORPHY) pour avoir des précisions sur l'utilisation d'ORPHY, début de programmation d'ébauches de fonctions en java avec l'api java.commx. Tentative de prise de contact avec l'entreprise créatrice d'Orphy : Microlec, afin d'emprunter un ORPHY GTS pour faire des test... aucune réponse...

SEMAINE 3 :

Documentation sur la communication de matériel relié à un ordinateur par un port série, développement des pilotes pour ORPHY, mais impossible de tester sans exemplaire de l'outil avec nous. J'ai commencé à travailler sur la technologie d'Eclipse concernant le développement du rcp et de plugin, il s'agit principalement de faire des tutoriels et de me renseigner au près des autre membres qui ont déjà commencés à apprendre la technologie. Début de travail sur le traducteur JVS permettant la traduction du code jvs en java. Début du développement du plugin d'inspection dynamique des variables.

SEMAINE 4 :

Cette semaine j'ai continué le développement du plugin permettant l'inspection dynamique de variables du fichier jvs au cours de l'exécution, celui-ci est mis en suspend à cause d'un changement au niveau du code de la compilation du programme, l'accès aux variables sera modifié. J'ai commencé à tester le traducteur jvs vers java, plusieurs bugs apparaissent que nous devons corriger. objectif proche : récupérer un ORPHY afin de pouvoir commencer l'intégration de l'interfaçage avec le matériel de mesure physique le plus rapidement possible.

SEMAINE 5 :

Dans les premiers jours le plugin d'inspection des variables a été terminé avec jean-michel, nous sommes aussi allés le lundi emprunter un Orphy gts2 à monsieur Mongiat au lycée de valbonne, ce n'est pas la version d'orphy que nous pensions, nous sommes donc un peu hésitant sur la démarche pour pouvoir communiquer avec l'appareil. Les recherches pour ma part s'orientent principalement sur l'utilisation de l'API javax.comm (API de contrôle des ports série grâce à java). Étant dans l'impossibilité d'installer une partition windows sur mon ordinateur j'ai tenté d'utiliser l'API sur MAC OS X, mais des problèmes concernant la détection des ports COM sont apparus. J'ai donc continuer en utilisant Windows xp par émulation, là j'ai pu installer le driver d'Orphy fourni par Laurent Mongiat, et commencer à utiliser l'api. Le dernier problème est de trouver le code de communication "rapide" pour accéder aux ports db15 sur lesquels est branché le thermomètre que monsieur Mongiat nous a aimablement prêté pour nos testes.

SEMAINE 6 :

La connexion avec le matériel Orphy a pu être faite grâce à l'API javax.comm, nous avons commencé le développement du plugin eclipse permettant la récupération des données et leur interprétation. Le problème actuel concerne la détection du port COM adapté, ainsi que l'adaptation de l'environnement du plugin aux différents matériels de mesures utilisés sur l'Orphy. Pour le moment l'interface est adaptée à l'utilisation d'un thermomètre, étant donnée qu'il s'agit du matériel emprunter, il faudra plus d'information sur le matériel utilisé en TP pour finir le plug-in de ce point de vue là.

SEMAINE 7 :

Cette semaine a été consacré à la finition du plugin Orphy. En début de semaine j'ai amélioré l'interface graphique, puis en fin de semaine nous sommes allés avec Sébastien Chalmeton voir Monsieur Mongiat au lycée de valbonne, là j'ai pu lui montré l'avancement d'Orphy, et connaître ses attentes exactes concernant le plugin, c'est-à-dire : permettre une acquisition paramétrable et automatique, pouvoir exporter les donnée sous fichier texte avec un format exploitable par regressi et reconnaître automatiquement le port COM utilisé par ORPHY pour simplifier l'utilisation du logiciel aux élèves. Le weekend à été consacré à l'implémentation des nouvelles directives et à la création d'un splash-screen et d'un icône pour notre application. J'ai eu quelques démêlés avec la programmation par thread concernant le plugin Orphy. Le plugin Orphy n'a pas pu être totalement finalisé et ne pourra pas être utilisé dans l'immédiat, nous nous sommes mis d'accord avec monsieur Mongiat de lui terminer pour qu'il puisse l'utiliser à sa futur rentrée. Il manque principalement des tests complets afin de trouver tous les bugs possibles pour avoir un plugin le plus robuste possible. La semaine prochaine sera donc entièrement consacré à l'écriture du rapport, la préparation à l'oral et à la mise en forme du code si besoin(ajout de commentaire et javadoc).

SEMAINE 8 :

Cette semaine a été consacrée principalement à l'écriture du rapport et à la présentation orale. Nous avons chacun écrit les partie correspondantes à notre travail, puis nous l'avons mis en commun afin d'harmoniser le tout. Nous avons aussi finalisé l'application dans ses derniers détails. Ce vendredi nous allons aller au lycée de Valbonne pour présenter le logiciel aux élèves puis à Monsieur Viéville l'après-midi.

Évolution du travail de Richter Xavier :

Semaine 1 (du 31/03/2008 au 06/04/2008):

Lundi, mardi, mercredi, jeudi :

- Finalisation du cahier des charges
- Écriture des diapositives
- Entraînement pour la présentation
- Répartition des rôles

Vendredi :

- Oral de pré-Soutenance où la technologie RCP a été évoqué par monsieur Huet
- Le groupe se scinde en deux parties, la première essaye d'explorer et de réutiliser DrJava dont le code source est très complexe mais dont le logiciel est très proche en terme de fonctionnalité du résultat final que nos encadrants souhaitent. Un second groupe dont je fais partie, cherche des informations et des tutoriels sur RCP

Quelques résultats de Recherche :

- <http://beuss.developpez.com/tutoriels/eclipse/plug-in/editor/bases/>
- http://wiki.eclipse.org/RCP_Text_Editor_Examples
- <http://wiki.improve.fr/wiki/moni/tutoriels>
- <http://www.vogella.de/articles/RichClientPlatform/article.html>

Weekend :

- Lecture de tutoriels
- Compréhension des structures de base d'une application rcp :
action, editor, view, perspective
- Début de prise en main de rcp.
- Création d'une première application rcp avec des boutons, des actions mais impossible de mettre en place un éditeur

Semaine 2 (du 07/04/2008 au 13/04/2008):

Lundi : Rencontre à Sophia avec notre encadrant **Thierry Viéville** concernant les choix d'implémentation. Après deux heures, il en ressort que RCP peut être une bonne solution. Il semble emballé par le côté 'tout plugin'. Il définit le langage qui sera utilisé dans les fichiers de configuration et explique de nouvelles idées.

- Rencontre avec Mr Huet et son équipe. Ils nous expliquent leur travaux et Jonathan Fradj nous donne des liens ainsi que des informations utiles sur la conception de plugin eclipse.

Mardi :

- Nouvelle Répartition des rôles au sein de l'équipe, je suis en charge de la création d'une console et de la traduction des erreurs.
- Première implémentation basique qui consiste à un try-catch dans lequel s'effectue la traduction. Implémentation non rcp.
- Recherche de documentation sur la console eclipse et du moyen pour la remplacer
- Recherche peu concluante

Mercredi :

- Téléchargement du projet pro-active via le svn, pour essayer de comprendre comment fonctionne leur console.
- Continuation de la console peu de résultat
- Écriture d'un mail de demande d'information à Mr Fradj qui nous répond rapidement
- Début de piste grâce au code qu'il nous fourni

Jeudi :

- Continuation du plugin console, l'orientation a changée suivant les conseils de mr fradj, il ne s'agira pas de remplacer ou d'étendre la console Eclipse mais de s'interfacer et de redéfinir les flots de sortie et d'erreurs.
- Bonne approche mais résultat peu concluant à cause d'un bug. La console apparaît mais l'écriture ne s'y effectue pas.

Vendredi :

- Continuation du travail sur la console
- Révélation : l'affichage ne s'effectue pas car le test consistait à un projet crée sous le workbench de d'Eclipse test. Celui-ci en tant que projet java, envoyait ses informations à la console java d'Eclipse sans passer par ma classe Console.
- Création d'un client RCP avec la console

Weekend Continuation dans la création du rcp

- Recherche d'ouvrage sur rcp :
 - <http://books.google.com/books?id=6Ob1ANNVcXcC&printsec=frontcover&hl=fr#PPP1,M1>
 - <http://www.lavoisier.fr/cgi-bin/couverture.cgi?target=O23ZR2V2KXRODX>
-

Semaine 3 (du 14/04/2008 au 20/04/2008):**Lundi :**

- Finition de la console au sein du rcp
- Mise en place de l'éditeur dans le rcp, recherche du bug quant à l'impossibilité d'ouvrir un fichier jvs (extension de notre langage)
- Résolution du bug néanmoins il manque encore des fonctionnalités comme sauvegarder, mise en avant des parenthèses, indentation, etc ...

Mardi :

- Mise en place de toutes les fonctionnalités de base (sauvegarder, charger, undo, redo, revert etc ...)

Mercredi :

- Mise en place de l'indentation automatique
- Début de mise en place du bracket closer

Jeudi & Vendredi :

- Mise en place du bracket closer automatique ainsi que de la mise en valeur des brackets
 - Fusion des autres parties au sein d'un même rcp afin de fonder un CVS
 -
-

Semaine 4 (du 21/04/2008 au 27/04/2008):**Lundi :**

- Jour entier passé à essayer de remettre le curseur d'écriture en fin d'indentation après avoir inséré le bracket fermant
- Aucun résultat probant tâche reportée à plus tard

Mardi :

- Création du bouton new (1ere partie) pour que l'élève d'un simple clique puisse avoir une page vierge. La première version pensée, consistait à la création d'un fichier puis à son ouverture de manière automatique au démarrage via une implémentation de IStartup qui permet d'effectuer une tâche au moment du lancement de l'application. Cette première version n'a pas fonctionné malgré les recherches pour résoudre les conflits. De plus la création du fichier s'effectuait de manière physique et n'obligeait pas l'élève à 'enregistrer sous' à la première sauvegarde. Recherche d'une meilleure solution, de nombreux tests d'implémentation mais aucun ne fonctionnait.

Mercredi :

- Création du bouton new (2nd partie) recherche avancée dans la mailing list eclipse pour trouver une solution viable. Un post contenait un début de réponse, il se basait sur la classe NonExistingFileEditorInput. Cette classe est interne donc non accessible et non invoquable. Recopie de la classe et du code donné en exemple dans le post.

Après implémentation, cela ne fonctionnait pas. L'éditeur souhaitait un document provider, or comme nous ouvrons des fichiers en dehors de tout workspace, notre éditeur ne contenait plus aucun document provider. A son ajout, new marchait mais open file ne fonctionnait plus. Après recherche, il s'avère qu'eclipse différencie les fichiers contenus dans le workspace (IEditorInput) des fichiers en dehors du workspace (FileStoreEditorInput) et ne renvoie pas la même classe de fichier. NonExistingFileEditorInput implémente IEditorInput qui est compatible avec le document provider mais pas FileStoreEditorInput. Impossible de caster l'une en l'autre ou de trouver une classe qui permette le passage de l'un à l'autre. J'ai eu l'idée de transformer la création du « new NonExistingFileEditorInput » en « new FileStoreEditorInput » et cela a fonctionné.

- Mise en place de l'auto-ouverture de l'éditeur au démarrage de l'application.

Jeudi :

- La console ne prenait pas en compte les appels de type system.in car elle étendait MessageConsole. Or cette classe n'est pas faite pour avoir un flot d'entrée. Réécriture de la console pour qu'elle étende IOConsole.
- Redéfinition du flot d'entrée. Après test, freeze de l'application. Il s'est avéré que l'appel à system.in se faisait dans le thread de l'UI, hors cela gèle l'application dans l'attente de la réponse de l'utilisateur. Après création d'un thread spécifique, cela fonctionnait mais System.in ne renvoie d'un byte.
- Recherche sur internet d'une meilleure solution, il semblait ne pas exister de fonction qui permette à un lycéen de faire de la saisie facilement.
- Création d'un premier scanf à la C : dans cette version je souhaitais avoir l'équivalent d'un scanf avec les %i,%s selon les types mais aussi avoir la possibilité de faire scanf("%i%s%d",i,j,k); en affectant bien entendu les valeurs saisies dans i,j,k. Cela s'avère impossible en Java à cause du passage par copie des types primaires. Impossibilité de forcer ce passage par référence.
- Demande d'aide sur developpez.com afin de savoir si il y a un moyen de faire cela.

Vendredi :

- Une seule réponse à la question posée sur le forum indiquant d'utiliser la classe Scanner. Obligation d'écrire une fonction par type de donnée.
 - Solution finalement retenue : création d'une classe Singleton Input qui sous traite à scanner. La classe Input sert à faciliter la vie aux lycéens en évitant les erreurs de saisie par exemple :
-

Semaine 5 (du 28/04/2008 au 04/05/2008):**Lundi :**

- Mise en place des icônes pour le design de l'application
- Configuration des icônes pour qu'elles soient placées au bon endroit
- Bug sur les icônes undo et redo : les icônes customs disparaissent quand l'éditeur apparaît pour reprendre leur apparence originale
- Bug non résolu
- Réunion avec le groupe pour discuter des nouvelles tâches et de la présentation de vendredi avec Laurent Mongiat

Mardi :

- Écriture du guide utilisateur
- Impossibilité de lancer l'application suite à des modifications liées à la partie compilation qui oblige à être en 1.6 en attendant d'être porté pour 1.5

Mercredi :

- Suite de l'écriture du rapport avec toujours impossibilité de faire des screen shoots

Jeudi : Férié**Vendredi :**

- Rendez-vous avec nos encadrants au lycée de Valbonne. Lors de cet entretien je m'aperçois que certains éléments que j'avais incorporés ont disparu de la base cvs p-e sous un mauvais commit.
- Après vérification il s'avère que les ordinateurs ne sont pas sous Java 1.5 comme il nous l'avait été indiqué mais en 1.6
- Mise à jour de mon JDK Mac pour passer en 1.6 (JDK sorti mardi) après cette mise à jour impossible de compiler le projet faute d'une obscure erreur de jar SWT-Carbon non trouvé
- Recherche d'une solution

Weekend :

- Obligé de passer sous windows pour avancer car aucune solution à l'erreur de compilation sous mac os X lié à Apple
 - Remise en place des éléments qui ont disparu du cvs
 - Amélioration de l'ergonomie de l'interface (icônes etc ..) et on a supprimé le open file classique pour ne laisser que l'ouverture par dossier préféré
-

Semaine 6 (du 05/05/2008 au 11/05/2008):**Lundi :**

- Création des pages html pour le guide utilisateur
- Finalisation du guide et insertion
- Recherche de solutions aux bugs (écriture dans la console, port déjà utilisé lors du rechargement de l'application)

Mardi & Mercredi :

- Recherche de solutions aux bugs
- Tentative de correction du traducteur JVs->java : refuse de fonctionner

Jeudi :

- Aujourd'hui avec Chalmeton Sébastien nous avons cherché à comprendre d'où vient le bug pour la compilation que nous avons eu lors du test avec Jean-Michel Guillaume au lycée de Valbonne. Nous avons réussi à identifier ce bug, il s'agit en fait du fait que la méthode utilisée pour la compilation n'est pas dans le JRE mais dans le JDK. Donc pour que notre application soit utilisable il faut installer sur la machine hôte le JDK 1.6.
- De plus la variable d'environnement doit être mise dans le classpath avant celle de windows, sinon il en résulte une erreur "ClassNotFoundException"

Vendredi :

- Réécriture du traducteur avec Chalmeton Sébastien, Sébastien s'occupe de transformer les macros, par introspection je dois réécrire le fichier java en ayant identifié les méthodes de l'élève et en ajoutant les mots public static etc Après de longue recherche et une demande d'aide sur le site développement pour savoir si l'introspection pouvait permettre la lecture du code d'une méthode, il s'est avéré que ce n'était pas possible. (voir : <http://www.developpez.net/forums/showthread.php?t=545290>). Néanmoins cette discussion aura permis de trouver une idée pour la traduction des macros que Jean-Michel aura eut en même temps (les grands esprits se rencontrent).

Weekend :

- Absence pour cause de concours jusqu'à mardi
-

Semaine 7 (du 12/05/2008 au 18/05/2008):

- Mercredi, jeudi, vendredi : debug & finolage du guide utilisateur
-

Semaine 8 (du 19/05/2008 au 25/05/2008):

- Lundi : Écriture du rapport
- Mardi : Écriture du rapport
- Mercredi : Écriture des transparents
- Jeudi : Oral blanc avec le groupe
- Vendredi : Voyage vers Valbonne pour tester l'application en conditions réelles