Projet informatique :

Interface Graphique Gnotime

Livrable

Tuteur : Denis Conan

Réalisé pour le 28/05/2017

Groupe : **Elise Castelli**

Hugo Duval

Grégoire Rabasse

Table des matières :

Introduction

Dans un monde professionnel où les projets et tâches peuvent s'accumuler, il est intéressant de mesurer le temps passé à travailler grâce au package Linux Gnotime. Ce dernier est un excellent outil qui permet de classer les différents travaux et de mesurer, en arrière-plan, le temps consacré à chaque tâche de chaque projet. La récolte de ces informations permet donc une optimisation du temps de travail.

Cependant, les données fournies par ce package sont brutes, et ne permettent pas de représenter visuellement la répartition du travail de l'utilisateur, et ne traite aucunement les durées mesurées. C'est là tout l'intérêt de notre programme. Grâce aux données brutes récoltées par Gnotime, l'utilisateur de notre logiciel a accès à de nouvelles informations pertinentes : des histogrammes, des camemberts et des courbes en fonction du temps, de son investissement pour ses différents projets par rapports aux autres. L'outil que nous proposons répond à des demandes qui peuvent être précises, comme par exemple le temps consacré aux différentes tâches du projet X sur une période Δt, représenté sur un diagramme camembert.

Ainsi, notre projet informatique se divise en deux parties importantes. La première consiste en un traitement des données en JAVA à partir des fichiers XML fournis par Gnotime. Il s'agit de lire les données et faire les calculs statistiques nécessaires et utiles que nous expliciterons plus dans le reste du livrable. La deuxième partie consiste à réaliser l'interface graphique qui affichera les graphes demandés.

Ce document comporte trois parties à la suite de la présente introduction : une présentation du cahier des charges, la description du développement proprement dit détaillant les différentes phases, et un manuel utilisateur.

Cahier des charges

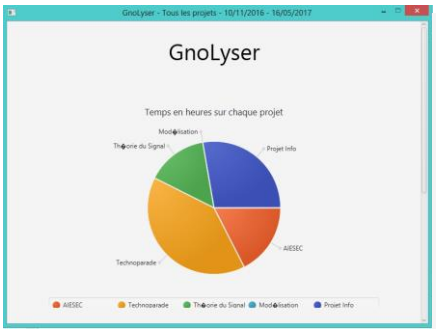
FONCTIONNALITÉS ET CONTRAINTES

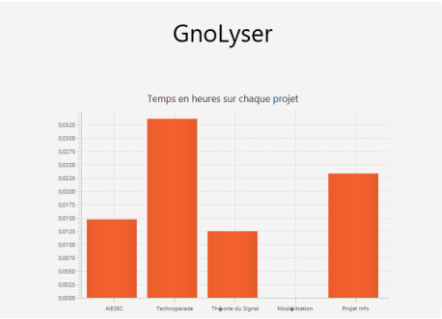
Les fonctionnalités de notre programme seront les suivantes :

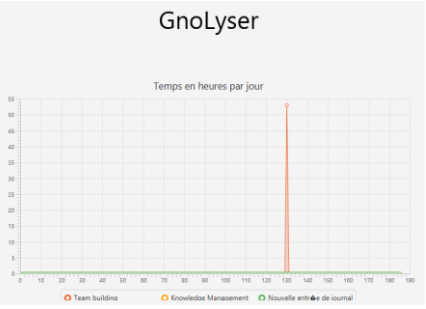
- donner un diagramme en camembert du temps passé sur chaque projet pour l'ensemble des projets,
- donner un diagramme en camembert du temps passé sur chaque tache pour l'ensemble des taches d'un projet,
- donner la courbe du temps passé sur un projet pour un intervalle de temps choisi,
- donner les courbes du temps passé sur tous les projets sur une période donnée,
- donner un diagramme en bâtons du temps passé sur chaque projet pour l'ensemble des projets,
- donner un diagramme en bâtons du temps passé sur chaque tache pour l'ensemble des taches d'un projet,
- apporter les détails statistiques en dessous du graphique,

Ces fonctionnalités sont implantées avec une interface graphique. Une première fenêtre s'affiche pour demander le(s) projet(s), un intervalle de temps et la figure souhaités.

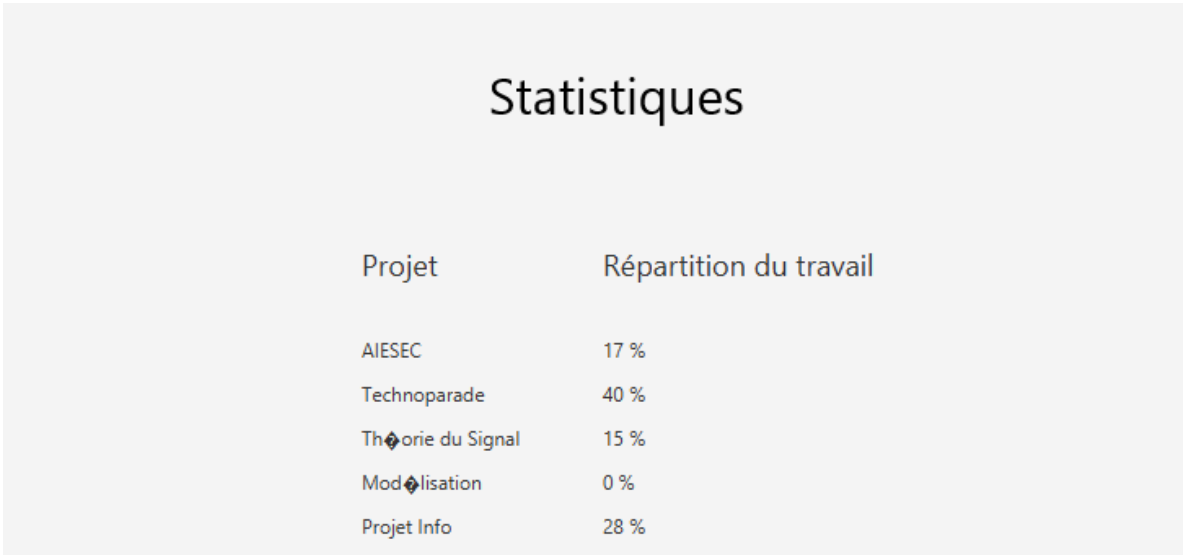
La figure s'affiche ensuite sur une deuxième fenêtre accompagnée des statistiques correspondantes.





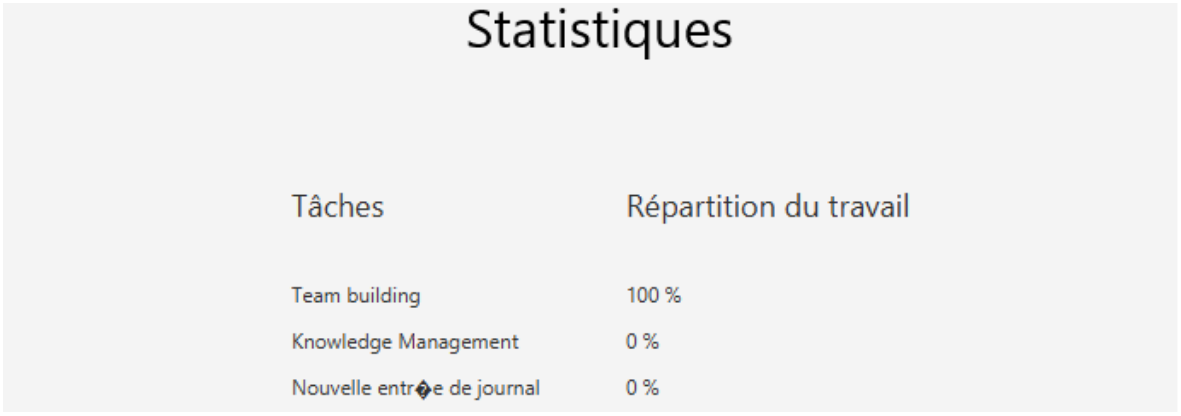


Des statistiques s'affichent en dessous des graphes :



Statistiques

Projet	Répartition du travail
AIESEC	17 %
Technoparade	40 %
Th�orie du Signal	15 %
Mod�lisation	0 %
Projet Info	28 %



Statistiques

Tâches	Répartition du travail
Team building	100 %
Knowledge Management	0 %
Nouvelle entr�e de journal	0 %

Les contraintes techniques sont les suivantes :

- le langage Java,
- la lecture des fichiers XML,
- la création d'une interface graphique associée,
- l'utilisation de dépôts SVN.

Développement

ANALYSE DU PROBLEME ET SPECIFICATION FONCTIONNELLE

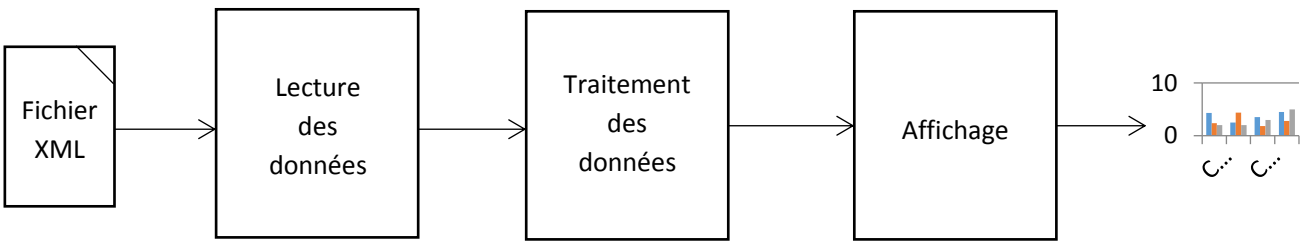
Comme nous avons commencé à l'expliquer précédemment, ce projet informatique est divisé en deux parties principales : une partie lecture/traitement des données extraites de Gnotime, et une partie mise en forme de ces données pour obtenir une représentation graphique plus conviviale des informations.

Pour cela, nous avons décidé de répartir le travail : Hugo et Elise travaillent sur la partie lecture/traitement des données, Grégoire sur la partie représentation graphique. John pourra, s'il réintègre le projet, travailler sur la vérification du code.

Ainsi, dans un premier temps, il faut s'occuper du traitement des données à l'aide du langage JAVA. Il s'agit de lire les fichiers XML à l'aide de bibliothèques. Il faut que notre programme soit capable de traiter les fichiers les plus conséquents, par exemple, les projets qui s'étendent sur plusieurs années sont divisés en sous projets, eux-mêmes comportant un certain nombre de tâches et ainsi de suite. C'est la raison pour laquelle les informations doivent être soigneusement triées pour que les graphes ne soient pas erronés.

Ensuite, une fois que le temps passé sur chaque tâche est récolté, il faut être capable de traiter ces données. Pour cela, il faut d'abord proposer des utilisations de ces statistiques, celles-ci ont déjà été décrites dans le cahier des charges. En fonction des statistiques que nous souhaitons afficher, il s'agira de mettre en place des algorithmes de calculs qui permettent de mettre en forme ces premières. Par exemple, si l'on veut afficher le diagramme en camembert de la répartition en termes de pourcentages des différentes tâches du Projet X, il faut sommer les heures passées ces tâches et comparer chaque durée au total du projet pour en déduire le pourcentage correspondant.

Enfin, la dernière étape sera de représenter ces données calculées sous une forme ludique avec, comme expliqué précédemment, des diagrammes par exemple, mais aussi des histogrammes.



CONCEPTION DETAILLEE

D'un point de vue informatique, notre programme se décompose en plusieurs modules :

- le premier : intro, affiche l'interface principale à l'utilisateur, et prend en compte les données que celui-ci rentre,
- le second : DOM récupère les informations du fichier XML fourni par Gnotime,
- le troisième : charts, affiche les résultats sur l'interface graphique.

Chacun de ces modules est conçu selon le modèle MVC : Modèle Vue Controleur. Le modèle se charge de récolter les informations demandées par le contrôleur, qui enregistre les commandes de l'utilisateur, qui sont affichées par la vue.

Dans le package DOM, les classes XMLDOM et GnotimeData permettent l'extraction des données, que l'on convertit du format XML en objets compréhensibles par JAVA. Plus précisément, la classe XMLDOM crée des objets appelés « Elements » sans distinguer les tâches et projets, et la classe GnotimeData récupère ces données pour les transformer en informations exploitables. Une grande difficulté de ce projet était la conversion de balises et nœuds d'un fichier XML en langage JAVA, et nous avons choisi de procéder ainsi : nous avons construit des classes Projet, Task et Intervalle pour définir les projets. Ainsi, un projet est constitué d'un nom, d'une durée et d'objets tâches ; une tâche est définie par un nom, une durée, et des intervalles (en effet, on peut la réaliser en plusieurs fois, c'est-à-dire en marquant des pauses, régulières ou non) ; et enfin, un intervalle est défini par un début et une fin.

Il est important de noter que nous avons défini une fonction « Intersection » dans la classe Intervalle, qui nous le verrons plus tard, permet d'ajuster les calculs statistiques à effectuer en fonction d'une période précisée ou non par l'utilisateur.

GnotimeData traite les informations données par les fonctions getProjectList, getTasksList et getIntervallesList de XMLDOM. GnotimeData permet de hiérarchiser les projets, tâches et intervalles comme expliqué précédemment, selon des listes d'objets distincts et bien définis qui nous permettent de travailler dans de bonnes conditions.

C'est sur cette définition d'objets que repose principalement notre programme.

Dans le package intro, on distingue trois classes : ControllerIntro, Main et Model. Nous passerons les détails du Main qui permet à l'utilisateur de lancer le programme. Les deux autres classes sont celles qui font fonctionner le programme à son lancement. Le contrôleur est le cœur du lancement du programme : il récupère la liste des projets grâce à la classe Model et sa fonction getProjectsNames, et l'affiche à l'utilisateur. Il prend aussi en compte les informations entrées par l'utilisateur et les traite, notamment, il transforme la date rentrée par l'utilisateur en « TimeStamp », notion de temps bien plus facile à gérer par l'ordiateur. L'affichage se fait grâce au fichier ui.fxml, que nous avons créé à l'aide de SceneBuilder.

Lorsque l'utilisateur appuie sur le bouton « Afficher », les classes et fichiers du package charts sont sollicités. Une fois de plus ce package est construit selon le modèle MVC. Lorsque l'utilisateur choisit d'obtenir des informations sur tous ses projets, le programme fait appel aux classes qui finissent par « All », et lorsqu'il choisit en particulier, ce sont celles qui finissent par « Each ». Le principe de fonction de ces deux « familles de classes est quasiment le même, la seule différence est que dans un premier cas, l'ordinateur doit récolter les informations qui concernent les différentes tâches d'un projet, et dans un second cas il ne considère qu'un ensemble de projets. Les contrôleurs « Area » sont sollicités par une demande de courbe en fonction du temps, les contrôleurs « Bars » par les histogrammes, et les contrôleurs « Pie » par le choix de diagramme camembert.

Ainsi, chacun des contrôleurs fait appel aux fonctions dont l'utilisateur a besoin, et définies dans les classes « Model ». C'est ce que l'on peut voir avec les fonctions « barsEach », « pieEach », « statsEach Names » et « statsEachDuree », ainsi que leurs jumelles s'appliquant à tous les projets dans le ModelChartsAll.

Que ce soit pour les histogrammes ou les camemberts, il faut récupérer les noms et durées de chaque tâche d'un projet, ou simplement de chaque projet (« barsEach » et « bars », en fonction de l'intervalle de temps demandée par l'utilisateur) puis les comparer et les afficher grâce à la vue et aux fichiers fxml.

En revanche, pour le calcul des courbes en fonction du temps, les calculs sont plus complexes, on le voit lorsqu'on exécute le programme sur une longue durée pour tous les projets. Cela vient du fait que le temps passé sur chaque projet ou tâche est calculé jour par jour, et non à l'aide des durées fournies par les fichiers XML.

TESTS

Pour tester notre programme, nous avons créé deux fonctions tests sur une classe JUnit : testgetTasksNames et testgetTasksDuree. Nous avons ainsi testé sur des projets simples que ces méthodes renvoyaient bien les noms et durées des tâches, et le résultat était positif.

```
@Test
	public void testgetTasksNames() {
		Intervalle i1 = new Intervalle(1,5);
		Intervalle i2 = new Intervalle(5,7);
		Intervalle i3 = new Intervalle(7,9);
		Intervalle i4 = new Intervalle(9,15);

		ArrayList<Intervalle> listeIntervalles1 = new
ArrayList<Intervalle>();
		listeIntervalles1.add(i1);
		listeIntervalles1.add(i3);

		ArrayList<Intervalle> listeIntervalles2 = new
ArrayList<Intervalle>();
		listeIntervalles1.add(i2);
		listeIntervalles1.add(i4);

		Task t1 = new Task("Tache1", 6, listeIntervalles1);
		Task t2 = new Task("Tache2", 8, listeIntervalles2);

		ArrayList<Task> listeTaches = new ArrayList<Task>();
		listeTaches.add(t1);
		listeTaches.add(t2);

		Projet p1 = new Projet("Projet", 14, listeTaches);

		if (Model.getTasksNames(p1).get(0) != "Tache1" &
Model.getTasksNames(p1).get(1) != "Tache2"){
			fail("mauvais noms");
		}
	}
```

```
@Test
	public void testgetTasksDuree(){
		Intervalle i1 = new Intervalle(1,5);
		Intervalle i2 = new Intervalle(5,7);
		Intervalle i3 = new Intervalle(7,9);
		Intervalle i4 = new Intervalle(9,15);

		ArrayList<Intervalle> listeIntervalles1 = new
ArrayList<Intervalle>();
```

```
            listeIntervalles1.add(i1);
            listeIntervalles1.add(i3);

            ArrayList<Intervalle> listeIntervalles2 = new
ArrayList<Intervalle>();
            listeIntervalles1.add(i2);
            listeIntervalles1.add(i4);

            Task t1 = new Task("Tache1", 6, listeIntervalles1);
            Task t2 = new Task("Tache2", 8, listeIntervalles2);

            ArrayList<Task> listeTaches = new ArrayList<Task>();
            listeTaches.add(t1);
            listeTaches.add(t2);

            Projet p1 = new Projet("Projet", 14, listeTaches);

            if (Model.getTasksNames(p1).get(0) != "Tache1" &
Model.getTasksNames(p1).get(1) != "Tache2"){
                  fail("mauvaise durée");

            }
      }
```

Manuel utilisateur

Le projet comporte :

- des fichiers sources XML,
- l'application,
- des tests unitaires pour des parties de l'application,

PRODUCTION DE L'EXECUTABLE

Le programme s'exécute grâce au bouton « Run Main ».

REALISATION DES TESTS

Pour chaque fichier test réalisé on va voir apparaitre le résultat de celui-ci : « Runs », « Errors » ou « Faillure ».

UTILISATION

Lors du lancement de l'application, une fenêtre s'affiche à l'écran. On peut ainsi grâce à un menu déroulant rentrer le projet qui nous intéresse, ou tous les projets. On peut aussi choisir un intervalle de temps et une forme de graphe : camembert, histogramme ou courbe. Il suffit alors d'appuyer sur le bouton « afficher » pour voir apparaitre une fenêtre avec le graphe et les statistiques. Si l'on veut comparer plusieurs graphes on peut appuyer de nouveau sur le bouton avec des données différentes. Une autre fenêtre va ainsi s'afficher.

Conclusion

Ce rapport décrit les étapes du développement d'une application sur la gestion du temps des projets en cours.

Cette application a été développée en Java, elle comporte deux parties : une sur la lecture, récupération des informations d'un fichier XML et des calculs statistiques et la deuxième sur l'interface graphique permettant à l'utilisateur de récupérer les informations qu'il souhaite connaitre.

Cette application pourrait encore être approfondie avec des calculs statistiques plus pertinents ou en confrontant simplement deux projets sur différents critères.

Annexe

PLAN DE CHARGE PREVISIONNEL

	Plan de charge prévisionnel					
			Charge en H / Participant			
DESCRITION DE L'ACTIVITE	Charge en %	Charge en H	John AO	Elise CASTELLI	Hugo DUVAL	Grégoire RABASSE
TOTAL	100	200	50	50	50	50
Gestion de projet	28%	56	12	15	15	14
Réunion de lancement	2%	3	0	1	1	1
Planning prévisionnel et Suivi d'activités	2%	4	1	1	1	1
Réunions de suivi	13%	26	5	7	7	7
Rédaction	8%	15	4	4	4	3
Outils collaboratifs (svn, etc.)	4%	8	2	2	2	2
Spécification	**3%**	**6**	**0**	**2**	**2**	**2**
Définition des fonctionnalités	3%	6	0	2	2	2
Conception préliminaire	**10%**	**20**	**2**	**6**	**6**	**6**
Définition d'un modèle de données	4%	7	1	2	2	2
Définition des fonctionnalités	3%	6	0	2	2	2
Définition des modules	4%	7	1	2	2	2
Conception détaillé	28%	56	18	13	13	12
Définition des classes	2%	4	1	1	1	1
Définiton des méthodes	6%	12	3	3	3	3
Définition des tests unitaires	5%	10	8	1	1	0
Auto-formation	12%	23	5	6	6	6
Maquettage des interfaces	4%	7	1	2	2	2
Codage	14,0%	28	8	6	6	8
Codage des classes	2%	4	0	1	1	2
Codage des méthodes	8%	16	2	4	4	6
Codage des tests unitaires	4%	8	6	1	1	0
Integration	9%	18	6	4	4	4
Intégration des modules	4%	8	2	2	2	2
Tests d'intégration	5%	10	4	2	2	2
Soutenance	8%	16	4	4	4	4

Préparation de la soutenance	6%	12	3	3	3	3
Soutenance	2%	4	1	1	1	1

PLAN DE CHARGE FINAL

	Plan de charge				
DESCRITION DE L'ACTIVITE	Charge en %	Charge en H	Elise CASTELLI	Hugo DUVAL	Grégoire RABASSE
TOTAL	100	163	50	50	63
Gestion de projet	26%	43	15	15	13
Réunion de lancement	2%	3	1	1	1
Planning prévisionnel et Suivi d'activités	2%	3	1	1	1
Réunions de suivi	13%	21	7	7	7
Rédaction	6%	10	4	4	2
Outils collaboratifs (svn, etc.)	4%	6	2	2	2
Spécification	**4%**	**6**	**2**	**2**	**2**
Définition des fonctionnalités	4%	6	2	2	2
Conception préliminaire	**13%**	**21**	**6**	**6**	**9**
Définition d'un modèle de données	5%	8	2	2	4
Définition des fonctionnalités	4%	6	2	2	2
Définition des modules	4%	7	2	2	3
Conception détaillé	25%	41	12	12	17
Définition des classes	2%	4	1	1	2
Définiton des méthodes	5%	8	1	1	6
Définition des tests unitaires	2%	4	2	2	0
Auto-formation	11%	18	6	6	6
Maquettage des interfaces	4%	7	2	2	3
Codage	16,0%	26	7	7	12
Codage des classes	3%	5	1	1	3
Codage des méthodes	10%	17	4	4	9
Codage des tests unitaires	2%	4	2	2	0
Integration	9%	14	4	4	6
Intégration des modules	4%	7	2	2	3
Tests d'intégration	4%	7	2	2	3
Soutenance	7%	12	4	4	4
Préparation de la soutenance	6%	9	3	3	3
Soutenance	2%	3	1	1	1

DIAGRAMME DE CLASSE