Système multi-agents appliqué à la simulation de gestion de planning logiciel

Introduction

La gestion de projet est une tâche complexe. Plusieurs paramètres entre en compte tels que les facteurs humain, et matériel. Des outils de simulation permettent d'aider les prises de décision concernant les allocation de ressources au projet. Toutefois, des incertitudes découlent de l'apparition d'événements aléatoires possibles. Ainsi la projection des réalisations, et leurs délais exactes ne représente pas un résultat réaliste.

Une application de simulation doit pouvoir aider à la mise en place du planning d'avancement de projet. Cette application doit reproduire les incertitudes sur les tâches à réaliser, et leurs durées. Qui plus est, un gestionnaire de projet doit prendre en compte les modifications des délais des tâches pour adapter les allocations de ressources.

L'objectif ici, est de proposer une technique de simulation de planning dans l'environnement du développement logiciel. Cette technique repose sur la mise en place d'un système multi-agents, offrant une intelligence artificielle (I.A) dans la simulation.

Exemple

Système multi-agents

En informatique, un système multi-agents (SMA) est un système composé d'un ensemble d'agents, situés dans un certain environnement et interagissant selon certaines relations. Un agent est une entité caractérisée par le fait qu'elle est, au moins partiellement,autonome. Ce peut être un processus, un robot, un être humain, etc.

Y. Demazeau, en 1997, propose une mnémonique pour représenter les systèmes multi-agents intitulée *« Voyelles » :*

- (A)gents : qui sont-ils ?
- (E)nvironnement : comment est-il construit ?
- (I)nteractions : qui peut faire quoi ?
- (O)rganisation : modèle social, modèle de communication ?
- (U)tilisateur : quelle est notre place en tant qu'utilisateur de l'outil de simulation ?

Modèle : planification de projet

L'approche adoptée est dite de *« Bottom-up »*, nous cherchons à modéliser individuellement

chaque agent dans leur environnement dans le but d'observer des résultats au niveau macroscopique.

Unité de temps

L'unité de temps utilisée dans la simulation s'appelle *WorkUnit.* Cette unité permet d'unifier la mesure temporelle au cœur de notre simulation est correspond à un tour dans l'avancement. Cela est comparable par exemple à la valeur *one-man-week* répandue en planification logicielle. Une tâche nécessitant trois *WorkUnits* de travail, auquel on alloue un seul développeur sera réalisée au terme de trois itérations. Si toutefois, trois développeurs sont alloués à la tâche elle aboutira au bout d'un seul tour.

Simulation

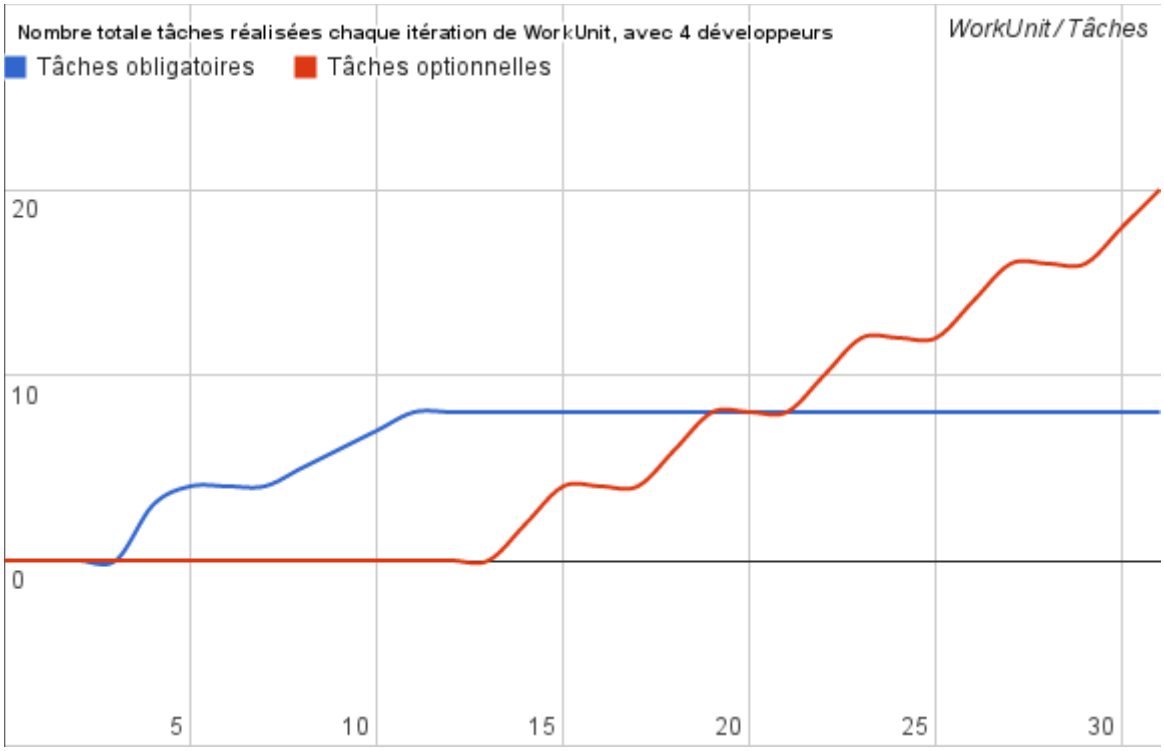
Un agent dans notre simulation est un développeur. Cet agent est attribué à une tâche. De plus, il dispose d'une capacité de travail, appelée *WorkUnit*. Chaque développeur allouer à une tâche disposera d'un capacité de un *WorkUnit*. En revanche lorsqu'il est alloué à une nouvelle tâche, cette capacité est divisée par deux, pendant un tour. Ceci modélise, le temps d'adaptation, ou apprentissage nécessaire à un développeur pour travailler sur une nouvelle implémentation.

Une tâche, représente une itération dans l'avancement du projet. Un projet est une série hiérarchique de tâches, mises à la suite des autres, ou en parallèle. Ainsi une tâche se caractérise par un ensemble d'agents développeur qui lui sont attribués. Une tâche peut être configurée de sorte qu'elle accepte une quantité maximale de développeur.

L'environnement ensuite, comprend les agents développeurs, et les tâches. Il permet d'appliquer les stratégies du systèmes à ces entités instanciées.

Stratégies

La première stratégie sur laquelle repose la simulation est d'embaucher un employé. L'algorithme assigne un développeur libre à une tâche qui n'est pas terminée. Mais, les développeurs sont allouées en priorité au tâches obligatoires, si toutes les tâches obligatoires ne peuvent plus accueillir de développeurs, alors nous assignons les ressources aux tâches optionnelles, comme représenté sur le schéma suivant :



Dans cette simulation, le nombre de tâches effectuées augmentent au cours du projet, en priorisant les tâches obligatoires

La seconde stratégie est de mettre à jour la durée d'une tâche. A chaque itération dans la simulation, la durée de toutes les tâches restantes sont réévaluées. Ceci consiste à augmenter ou réduire la quantité restante de *WorkUnits.* Cette méthode tente de se rapprocher du monde réel ou l'estimation des délais est mise à jour au cours du projet.

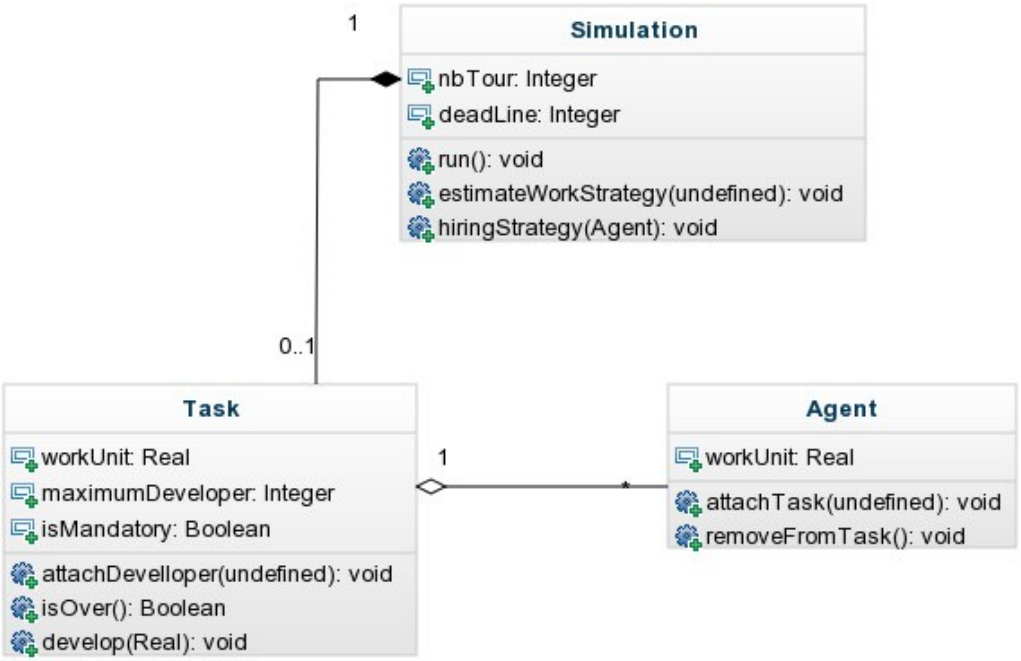
Implémentation

La durée ou *WorkUnit* est une valeur réelle. La durée d'une tâche est majorée ou minorée à chaque tour de la simulation de façon aléatoire via la méthode *Simulation.estimateWorkStrategy(Task task)*.

La simulation est une suite d'itérations non équitables. Lors de chaque itération, *Simulation.hiringStrategie(Agent agent)* permet d'allouer les employés aux tâches lorsqu'ils sont libres, en privilégiant les tâches obligatoires.

La simulation se termine lorsque la durée du projet est écoulée. Le résultat final de la simulation permet d'affirmer ou non que les tâches obligatoires (*MandatoryTask)* sont terminées. Les tâches optionnelles (*OptionnalTasks)* qui sont terminées sont également énumérées.

En revanche la durée maximum de la simulation est une valeur entières. Nous représentons la fin du projet à 20 *WorkUnits* par exemple.



Img 1: UML : Simulation multi-agents pour la planification de tâches à développer

L'exécution est basée sur un modèle probabiliste appelé méthode Monte-Carlo. La stratégie d'embauche (*Simulation.hiringStrategy)* s'applique sur les tâches, elles-mêmes triées aléatoirement. Ainsi aucune tâches n'est favorisée plus qu'une autre. Cependant l'algorithme prend en compte la hierarchie de tâche, de sorte à commencer par les tâches obligatoires (*Task.isMandatory).* Enfin, les développeurs sont également triés, alloués aléatoirement, sans privilégier l'un ou l'autre. Ceci permet de disposer d'une intelligence (I.A) équitable.

Utilisateur

L'outil développé permet de paramétrer chaque entité de la simulation. D'abord, il faut configurer la durée du projet. Ensuite, il faut définir le nombre de tâches obligatoires, respectivement optionnelles. Ces tâches sont configurées en entrant, le nombre de développeurs maximum qui peuvent y contribuer, et la durée de réalisation estimée.

Résultats

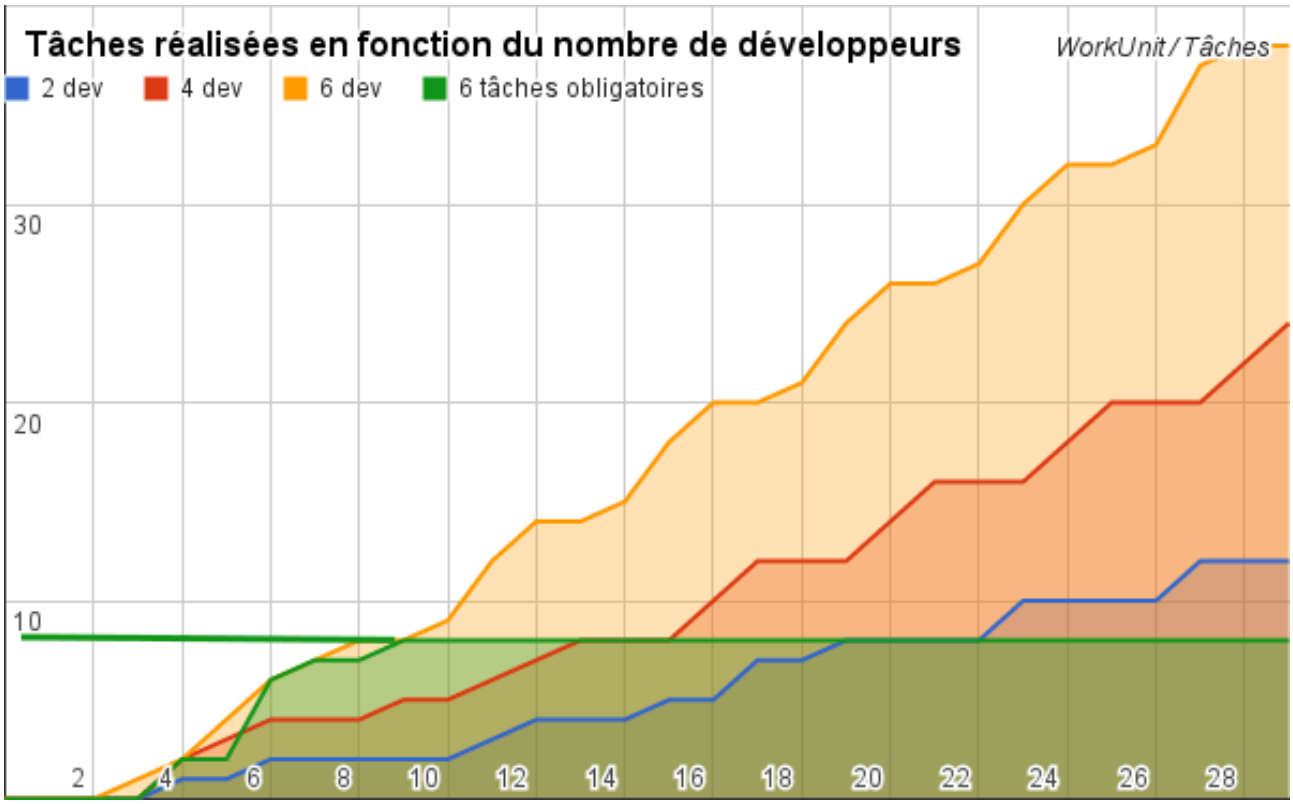
Comparaison de résultats de planification

Une première simulation permet de tester les délais nécessaires pour la réalisation d'un projet. Il est possible ici, de comparer les délais nécessaires en fonction du nombre de développeurs. Sur le schéma suivant, la ligne verte sert de seuil minimal du nombre de tâches à réaliser.

Paramètres de simulation :

- délais maximale de 30 *WorkUnits*
- 8 tâches obligatoires, avec une estimation respective de 5 *WorkUnits* à la création, et pouvant accueillir 3 développeurs maximum

- 40 tâches optionnelles, avec une estimation respective de 3 *WorkUnits* à la création, et pouvant accueillir 3 développeurs maximum



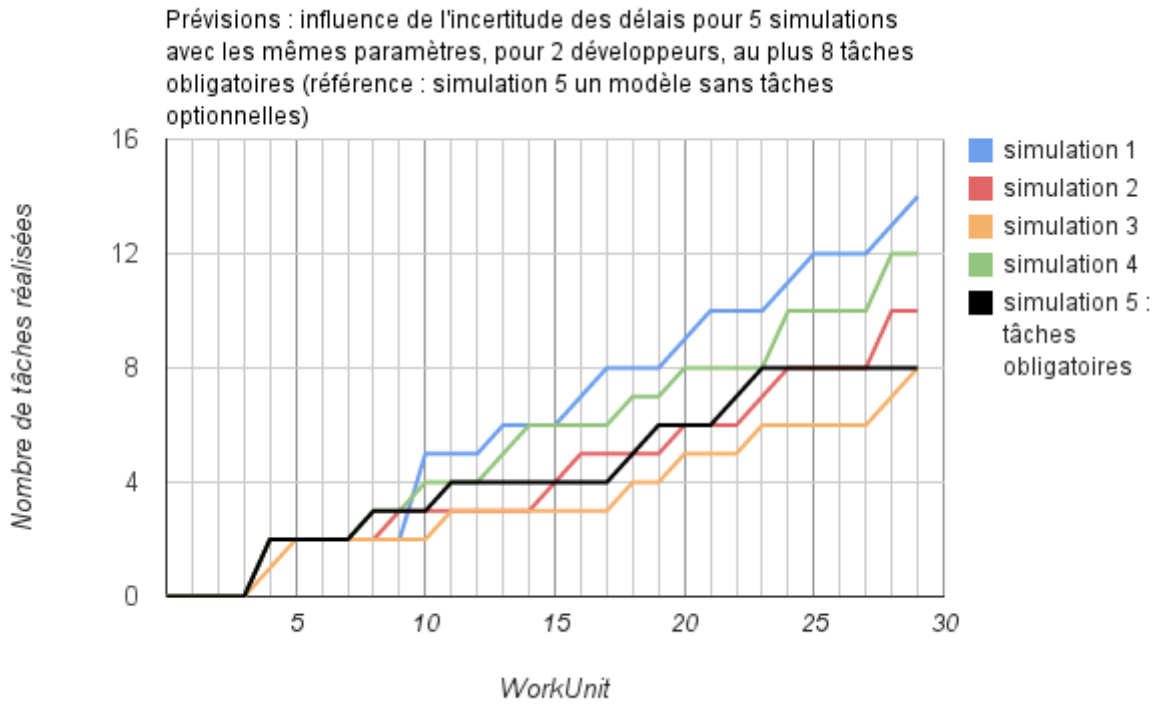
Interprétation :

Chaque simulation permettent de terminer les tâches obligatoires. Nous observons qu'avec 6 développeurs les tâches obligatoires sont terminées au bout de 9 *WorkUnits*. En revanche, avec 2 développeurs, elles se terminent au bout de 20 *WorkUnits*.

Influence de l'aléatoire

A chaque itération de la simulation, c.a.d lorsqu'un *WorkUnit* arrive à son terme, les délais sont mis à jour de façon aléatoire pour simuler les incertitudes dans l'avancement d'un projet réel.

Sur le schéma suivant, 5 simulations ont été exécutées avec les mêmes paramètres.



Interprétations :

Le nombre de tâches réalisées à un instant *t* n'est pas régulier. Ainsi, l'utilisateur (un chef de projet) devra être vigilant sur les délais estimés. En effet, nous constatons que sur la 3ème simulation n'aboutit qu'au terme du 29e *WorkUnit.*

Conclusion

SMA : permet une simplification des modèles proposés

3 qualités fondamentales : adaptatif, autonome, emergence