Ecole Polytechnique de l'Université François Rabelais de Tours
Département Informatique
64 avenue Jean Portalis
37200 Tours, France
Tél. +33 (0)2 47 36 14 14
www.polytech.univ-tours.fr

Projet Recherche & Développement
2015-2016

Simulation d'ateliers de fabrication

Introduction au principe de la simulation orientée processus

Tuteurs académiques
Christophe Lenté

Étudiants
Florian Montalbano (DI5)

26 septembre 2016

Liste des intervenants

Nom	Mail	Qualité
Florian Montalbano	florian.montalbano@etu.univ-tours.fr	Étudiant DI5
Christophe Lenté	christophe.lente@univ-tours.fr	Tuteur académique, Département infomatique

Avertissement

Ce document a été rédigé par Florian Montalbano susnommé l'auteur.

L'école polytechnique de l'université François Rabelais de Tours est représentée par Christophe Lenté susnommé le tuteur académique.

Par l'utilisation de ce modèle de document, l'ensemble des intervenants du projet acceptent les conditions définies ci-après.

L'auteur reconnait assumer l'entière responsabilité du contenu du document ainsi que toutes suites judiciaires qui pourraient en découler du fait du non respect des lois ou des droits d'auteur.

L'auteur atteste que les propos du document sont sincères et assument l'entière responsabilité de la véracité des propos.

L'auteur atteste ne pas s'approprier le travail d'autrui et que le document ne contient aucun plagiat.

L'auteur atteste que le document ne contient aucun propos diffamatoire ou condamnable devant la loi.

L'auteur reconnait qu'il ne peut diffuser ce document en partie ou en intégralité sous quelque forme que ce soit sans l'accord préalable du tuteur académique.

L'auteur autorise l'école polytechnique de l'université François Rabelais de Tours à diffuser tout ou partie de ce document, sous quelque forme que ce soit, y compris après transformation en citant la source. Cette diffusion devra se faire gracieusement et être accompagnée du présent avertissement.

Pour citer ce document :
Florian Montalbano, *Simulation d'ateliers de fabrication: Introduction au principe de la simulation orientée processus*, Projet Recherche & Développement, Ecole Polytechnique de l'Université François Rabelais de Tours, Tours, France, 2015-2016.

```
@mastersthesis{
    author={Montalbano, Florian},
    title={Simulation d'ateliers de fabrication: Introduction au principe de la simulation orientée proces-
        sus},
    type={Projet Recherche \& Développement},
    school={Ecole Polytechnique de l'Université François Rabelais de Tours},
    address={Tours, France},
    year={2015-2016}
}
```

Document mis en forme sous LaTeX
grâce à la classe de document `polytech.cls` V1.0 (2016/02/01)

Table des matières

Table des figures

Première partie

Partie Recherche

1 Présentation du sujet

Ce sujet a été proposé par Christophe Lenté, Enseignant chercheur et Maître de conférences à l'École Polytechnique de l'Université de Tours (EPU - Polytech'Tours) Laboratoire d'Informatique (LI).

1 Introduction

Le concept de simulation est apparu très tôt dans l'histoire de l'humanité avec par exemple les simulations militaires. L'objectif était de prévoir le résultat d'une bataille en fonction de divers stratégies pour déterminer la meilleure en terme d'efficacité ou de pertes. Cependant, les moyens de calculs étant limités, il n'était pas possible de tester toutes les possibilités dans un temps raisonnable.

De nos jours, la simulation est utilisée dans plusieurs domaines que se soit l'économie, les chaînes de production, ou même dans les jeux vidéos. Ce projet s'intéresse en particulier sur les ateliers de fabrications. Le principe de ces ateliers est de transformer une ou plusieurs matières premières en autre chose, en soumettant ces matières à des transformations réalisées en série par des machines. Par exemple, à partir de planches de bois, on peut fabriquer une table. Pour cela, il faut d'abord scier les planches puis les percer, ... Certains traitements peuvent être plus long que d'autres et certains doivent être réalisés dans un ordre précis. De plus, un traitement nécessite des ressources comme une machine et un employé ou alors pour assembler les morceaux de la table, il faut les quatre pieds plus le plateau.

Le problème est le suivant : Prenons un atelier de fabrication avec trois traitements. Les matières premières doivent arriver au traitement A puis passer par le traitement B et enfin se faire raffiner par le traitement C.
L'atelier dispose de 2 machines pour réaliser A, 3 machines pour B et 4 pour C. Il se trouve qu'il y a de l'attente entre certains traitements dû au manque de machines (ou à un traitement trop long).
On souhaite améliorer le taux de production (nombre de produits finis par heure). Cependant, on ne peut ajouter qu'une seule machine. Pour maximiser l'efficacité de l'investissement, on va chercher à voir quel est le meilleur choix pour améliorer au plus le taux de production.

Pour décider, on ne peut pas se permettre d'acheter les machines pour tester. On va *simuler* les différentes configurations : une avec une machine de type A, une avec un type B et une avec un type C. Avec si peu de configuration à tester, le procéder de simulation peut être réalisé "à la main", si on exécute la simulation sur une petite durée. Mais si on souhaite procéder sur du long terme ou avec des ateliers beaucoup plus compliqués, on va se diriger vers un traitement informatisé qui offrira la puissance de calcul nécessaire pour compléter ces simulations plus longues ou plus complexes.

Ce projet est axé sur l'étude d'une bibliothèque de Python, nommée SimPy, proposant des fonctions et des objets favorisant la réalisation de simulations informatisées. Il existe plusieurs types de simulation, je vais dans un premier temps, détailler le type qui nous intéresse, à savoir les simulations orientées processus. Ensuite, je vais proposer des algorithmes permettant de représenter les comportements des ateliers de fabrication. Puis, je vais décrire les principales fonctionnalités de SimPy et enfin, expliquer comment le problème de simulation des ateliers peut être abordé avec la bibliothèque SimPy.

2 Cahier des charges

Le résultat attendu pour ce projet est dans un premier temps une acquisition de connaissances dans le domaine des simulations et plus particulièrement dans les simulations orientées processus.

Ensuite, il faudra restituer ces connaissances en réalisant différents algorithmes de *processus* afin de pouvoir représenter des cas classiques de processus d'ateliers de fabrications. Á partir de ces processus "standards", il faut en étudier la possibilité de créer une simulation d'atelier et observer la *souplesse* de la combinaison de ces processus. Dans la finalité, si l'étude se révèle favorable, il pourrait être possible de représenter un grand nombre d'ateliers de fabrications "standards" en combinant ces différents processus "standards".

Après cette réflexion, il faudra se familiariser avec la bibliothèque python *SimPy* et en analyser les capacités à créer des simulations orientées processus. Puis, reprendre les algorithmes des processus "standards" recherchés pour les implémenter en Python en se basant sur SimPy.

2.1 Livrables

Á la fin du projet, il est attendu des algorithmes résolvant les cas de "base" d'un atelier de fabrication ainsi que leur implémentation à l'aide de SimPy.

Pour chacun de ces cas de base, il y aura donc un programme Python associé contenant le code pour effectuer la simulation du cas considéré.

3 Objectifs du projet

L'objectif du projet est de déterminer si il est possible de "standardiser" la construction de simulations orientée processus, c'est-à-dire proposer des *bases* et des *briques élémentaires* afin qu'une personne puisse "facilement" du moins d'une manière simplifiée créer une simulation d'un *atelier de fabrication*. Ensuite, dans la limite des capacités de SimPy et des algorithmes façonnés, en proposer une implémentation.

2 Ateliers de fabrication

Le titre de ce projet peut être divisé en deux parties : *simulation* et *atelier de fabrication*. Je vais commencer par définir la notion d'atelier de fabrication qui sera utilisée par la suite comme argument ou exemple pour mieux expliquer le principe de simulation et certains choix concernant l'orientation de l'étude.

1 Définition

Un atelier de fabrication est un ensemble de traitements et d'éléments nécessaires à leurs réalisations qui transforme des matériaux en produits raffinés.

1.1 Les traitements

Un traitement est une étape de la fabrication qui transforme l'état d'un matériau pour qu'il puisse continuer son raffinement. Exemple : Il faut faire fondre le verre pour pouvoir le mouler par la suite.
Chaque traitement nécessite une ou plusieurs ressources pour être réalisé.

1.2 Les ressources

Une ressource peut être une machine (perceuse, bras robotique articulé, four, ...), une personne, un véhicule, un conteneur... Pour effectuer un traitement, l'élément à traiter réquisitionne (retient) les ressources dont il a besoin, puis il les relâche lorsque son traitement est fini. Plusieurs ressources identiques peuvent être considérées comme une seule ressource. Chacune d'entre elle représente une *unité* et leur nombre caractérise la *capacité* de la ressource.
Dans la catégorie des ressources se trouve également les zones de stockage. Chaque zone possède sa propre capacité pour stocker un certain nombre de matériau.

1.3 Les files d'attente

Cette notion ne doit pas être confondue avec les zones de stockages même si leurs utilités sont très proches. Les traitements nécessitent un certain temps pour se réaliser et certains sont plus lents que d'autres. Il est donc possible à une étape de la fabrication, qu'un matériau soit prêt pour son prochain traitement mais que les ressources nécessaires ne soient pas disponibles. Pour ne pas bloquer la fabrication en amont, on peut utiliser des files d'attente dans lesquelles les matériaux peuvent attendre la libération des ressources. Il peut y avoir plusieurs files par traitement, et chaque file peut être de taille différente.

1.4 La topologie

Pour suivre une succession de traitement, les matériaux doivent être transportés de machine en machine et de stocks en stocks. Ce déplacement peut être automatisé (système de tapis roulant) ou manuel (un ouvrier doit déplacer à la main ou avec une machine les matériaux). Dans tous les cas, il peut exister un temps de trajet pour effectuer le déplacement et dans certain cas, le déplacement nécessite une ou plusieurs ressources. L'organisation spatiale de l'atelier à un rôle non négligeable dans les performances de celui-ci et pour avoir une meilleure représentation d'un atelier, il faut prendre en compte ce critère.

1.5 Exemple

Voici la schématisation d'un atelier de fabrication réalisant la construction de tables à partir de bois :

Sur le premier exemple, il y a 3 traitements, 2 files d'attentes et 2 zones de stockages.

Sur la seconde image 2, j'ai représenté un exemple de topologie d'un atelier : Il peut être compliqué de trouver une solution parfaite dans certains cas, il faudra alors faire des compromis et il n'est pas toujours évident de comparer ces compromis.

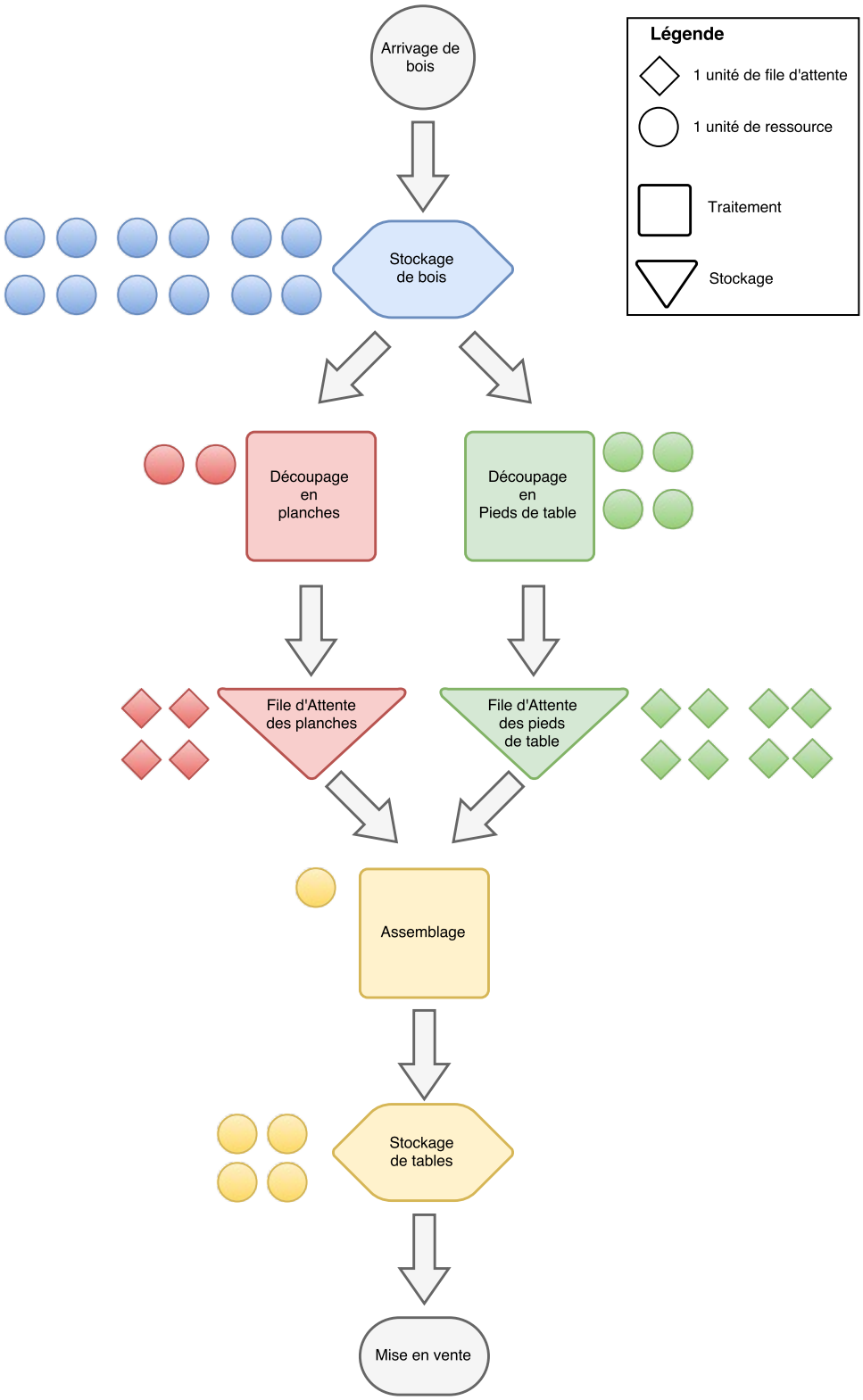


Figure 1 – *Exemple d'un atelier de fabrication*

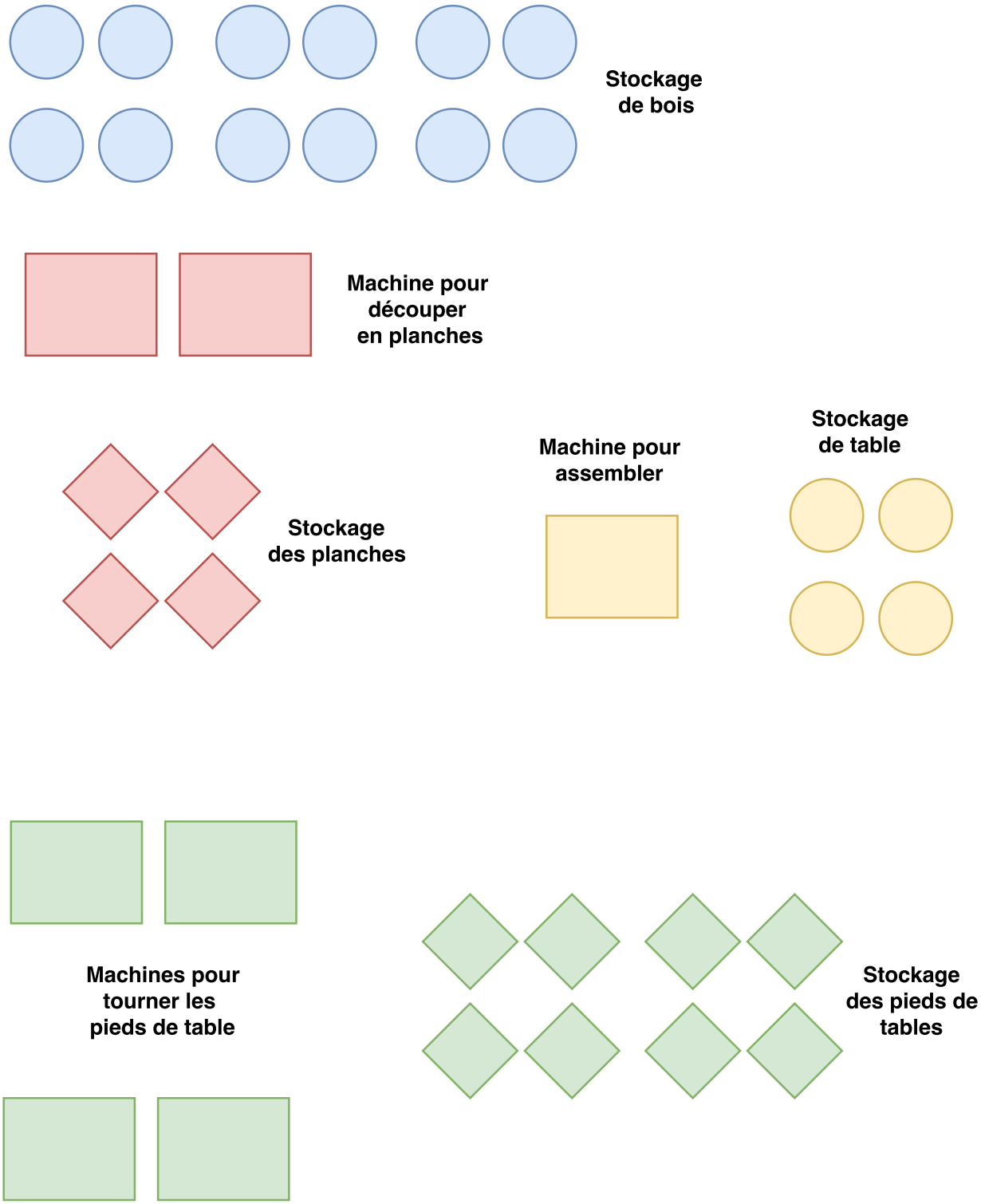


Figure 2 – *Exemple de topologie pour un atelier*

3 Qu'est-ce que la simulation de système

Le concept et la pratique de la simulation sont au coeur de ce projet, il devient alors nécessaire de bien définir ce principe.

Dans ce rapport, le terme simulation est employé pour parler de simulation de système.

1 Introduction à la simulation de système

Je vous propose de commencer par un peu d'étymologie. Le mot simulation vient du latin *simulo* signifiant *reproduire*, *copier*, *imiter* ou de *simulato* soit *celui qui représente*, *imitateur*. L'idée de la simulation est en effet de reproduire, et dans le cadre de ce projet, reproduire un système.
Reproduire un système consiste à prendre en compte tous les éléments qui interviennent *dans* celui-ci ainsi que les liens qui existent entre ces éléments. Á cela, il faut ajouter les comportements internes et externes régissant le système.

Cette reproduction se nomme "*modèle*" et va servir à étudier le fonctionnement d'un système.

Le dictionnaire *Webster's Collegiate Dictionary* proposait comme définition "*feindre*, *obtenir l'essence de quelque chose*, *sans la réalité associée*". Cette définition souligne un point important, **la simulation n'est pas une reproduction exacte du réel**. Si aux premiers abords cela semble être un défaut, dans les faits, c'est un avantage non négligeable. L'énorme avantage que cela apporte est que le vrai système n'a pas besoin d'être testé pour pouvoir être étudié.
Prenons l'exemple d'une attaque de missiles sur un pays, ce n'est pas un cas pouvant être testé "juste pour voir" comment le système fonctionnerait dans ce genre de situation. C'est cette qualité de la simulation qui en fait un outil très puissant et surtout très utile. Ce qui m'amène à la définition suivante :

La simulation est un outil aidant à la prise de décision face à un problème pour un système donné.

La simulation se classe dans la catégorie des outils comme le marteau. C'est-à-dire qu'il faut l'utiliser d'une certaine manière pour résoudre certains problèmes (un marteau n'est pas adapté pour visser par exemple).

En résumé, **la simulation est un outil pour aider à la résolution de problèmes. Elle se base sur la reproduction non exacte appelée *modélisation* d'un système, pour analyser le comportement de ce dernier dans des conditions difficiles voire impossibles à tester sur le vrai système.**

Pourquoi utiliser la simulation pour étudier un système ? Est-ce la seule manière de procéder ?

2 Pourquoi utiliser la simulation ?

Afin de répondre à cette question, je vais m'appuyer sur l'explication donnée par Averill M. Law [3]. Cette explication compare les différentes manières d'étudier un système. Ces manières sont placées dans un arbre qui permet de choisir la méthode la plus adaptée pour analyser un système.

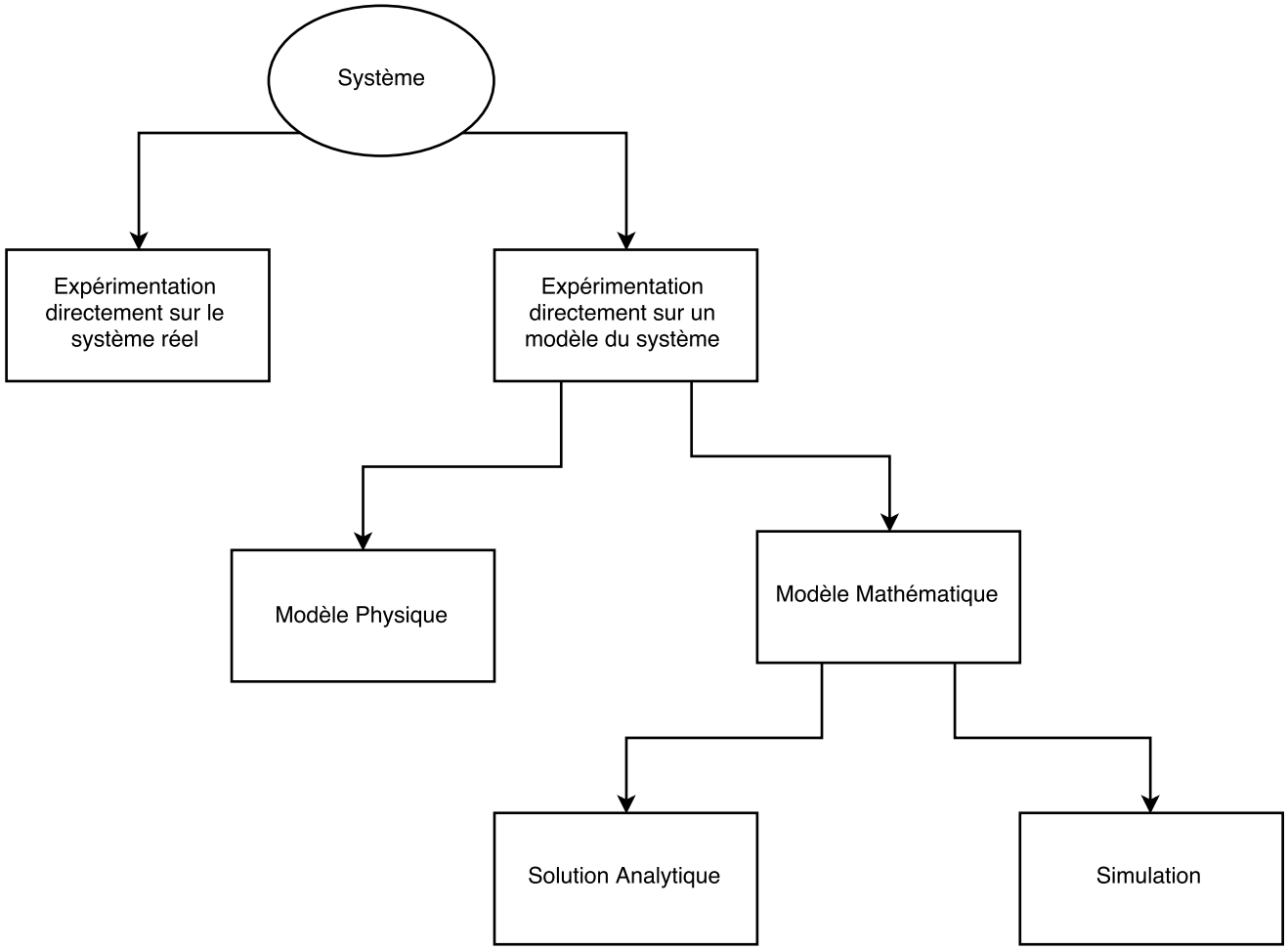


Figure 1 – *Moyens pour étudier un système d'après Law*

Cette figure est la version "traduite" de celle du livre de Law [3] (page 4).

Comment parcourir cette arbre ?
Il faut commencer par le haut. Le premier choix à effectuer est :

Expérimentation sur le système réel ou *Expérimentation sur un modèle*

La méthode la plus efficace pour étudier un système est de le réaliser. Cependant, cette option n'est faisable que si la mise en place du système est dans un premier temps *possible* et si elle n'est pas *trop coûteuse*. Il faut considérer que le système à étudier peut être une nouvelle configuration d'un système existant. Dans ce cas, la modification de l'existant peut être compliquée à mettre en œuvre.
Exemple : Tester une pièce sur un satellite en orbite. C'est une action très compliquée et très coûteuse qui ne peut pas permettre l'expérimentation sur le système réel.
Ou bien, le système n'existe pas et l'objectif est de trouver la meilleure configuration pour construire le système de manière optimale.

Exemple : La construction d'un habitat sur la Lune, il n'est pas possible de faire des tests de construction directement à l'extérieur de notre atmosphère.

Lorsque les contraintes de mises en place ou de modifications du système sont acceptables, il est préférable et plus efficace de se tourner vers l'*expérimentation directe sur le système réel*. Par exemple, essayer de nouveaux tracés de lignes pour organiser un parking. Mais lorsque les coûts ou la complexité sont trop important, il faut s'orienter vers l'*expérimentation sur un modèle du système*. Mais lorsque l'on utilise un modèle, il faut être très attentif à la qualité de celui-ci.

Pour cette deuxième option, il reste encore des choix à faire.

Modèle Physique ou *Modèle Mathématique*

Le modèle physique correspond à une reproduction matériel du système. Il peut être à l'échelle grandeur nature ou bien réduite. Ce n'est pas le système réel, c'est une reproduction du système qui est étudié. En exemple de modèle physique, pour préparer les astronautes à une mission de réparation du satellite *SolarMax*, une maquette du satellite et de la navette ont été utilisées pour simuler la réparation dans des conditions proches du systèmes [WWW6].
Ces modèles physiques sont pratiques dans certaines disciplines comme la mécanique ou le management d'un système mais la recherche opérationnelle emploi plus souvent les modèles mathématiques. Le principe de ces modèles est de représenter le système par un ensemble de règles (comme des équations). Un exemple de modèle mathématique sont les automates finis déterministes. Ils sont complètement expliqué par le 5-tuples : listes des états, alphabet, fonction de transition, état initial, liste des états d'acceptation.[WWW10]
Pour donner un exemple plus simple, prenons la représentation du volume d'une boîte. Le volume d'une boîte peut se traduire par Volume = Hauteur * Largeur * Longueur . C'est un modèle mathématique représentant le volume à l'intérieur d'une boîte. [WWW4]

Pour revenir sur la notion de qualité d'un modèle, le modèle mathématiques ci-dessus peut être affiné en prenant en compte l'épaisseur de la boîte. Une description plus approfondie de la qualité d'un modèle sera détaillé plus tard dans le rapport.

Enfin, il reste un dernier choix à effectuer si on travaille sur un *modèle mathématique*.

Solution Analytique ou *Simulation*

Le modèle étant construit, on connaît alors le comportement de celui-ci au travers d'équations et de règles logiques. Si à partir de ces informations on arrive à extraire une équation qui explicite une relation entre le point du modèle à étudier et le reste des données, on a utilisé une *solution analytique*.
Pour illustrer cette explication, je vais reprendre l'exemple de la boîte. On sait que

$$Volume = Hauteur * Largeur * Longueur$$

. Si on cherche la hauteur de la boîte à partir de laquelle le volume est supérieur à 100 mètres cubes sachant que la Largeur est fixée à 5 Mètres et la Longueur à 5 Mètres également. On va travailler depuis l'équation de départ pour obtenir

$$Hauteur >= \frac{Volume}{Largeur * Longueur}$$

et en remplaçant par les valeurs numériques :

$$Hauteur >= \frac{100}{5 * 5} \quad \Rightarrow \quad Hauteur >= 25m$$

La solution a été trouvée par analyse du modèle et résolution des équations. Cette méthode permet d'obtenir des résultats exacts. L'inconvénient provient du fait que si le système est complexe, la résolution des équations par l'analyse devient trop complexe et/ou trop longue. Dans ces cas là, on s'oriente vers la *simulation*. "C'est à dire, soumettre le modèle à différentes entrées numériques pour observer leurs influences sur les sorties du système" (Law [3] page 5). Cette phrase évoque la différence avec la méthode analytique. Le système d'équations n'est pas résolu, il est testé avec différentes valeurs. Cela engendre que les solutions trouvées peuvent être bonnes mais elles ne seront pas forcément la ou les meilleures.

Les ateliers de fabrications entre dans le cas de la simulation car on ne peut pas ajouter de machines pour "tester" la nouvelle configuration. Il faut doc passer par un *modèle du système*. La question du type de modèle (physique ou mathématique) obtient sa réponse de la même manière que pour le choix précédent. On ne peut pas se permettre de reproduire le système avec d'autres machines pour tester différentes configurations. On utilisera donc un *modèle mathématique*. Enfin, comme un atelier de fabrication prend en compte des arrivées de matières premières variables et des temps de traitement de machines tout aussi variable, il faudrait, pour obtenir une solution analytique, essayer toutes les combinaisons possibles d'horaires d'arrivées de matières premières, combinées aux quantités possibles et ajouter à cela toutes les possibilités de temps de traitement pour chaque machine.

Je vais maintenant essayer de couvrir tous les concepts importants de la simulation dans les sections suivantes.

3 Les composants d'une simulation

Une simulation est composée de plusieurs parties notoires.

En premier, il y a le **système** à simuler. Ce qui nous intéresse dans ce système est son évolution au cours de la simulation et en particulier ses **variables** "*globales*". Elles représentent tous les aspects du système comme l'occupation de chacune des files d'attente, le nombre de ressources utilisées,... De plus, elles contiennent les paramètres de la simulation, par exemple le temps nécessaire pour effectuer tel ou tel traitement.

Ensuite, il y a les "**accumulateurs statistiques**" ([2] page 27) dont le but est de surveiller les performances du système notamment, les temps d'attente moyens dans les files, la quantité de matériaux transformés, ... Ils sont tous initialisés à zéro au début. C'est sur ces variables que l'on va pouvoir estimer l'efficacité de la configuration du système simulé.

Pour le cas des variables, il y en a une qui se démarque des autres, l'**horloge de la simulation**. Elle permet de connaître "l'heure" où en est la simulation car dans la simulation, le temps ne s'écoule pas de façon continu. Il ne prend que les valeurs pour lesquelles il se produit un événement. La raison à cela est que calculer tous les instants peut être très long alors que s'il n'y a pas d'événement, les variables ne seront pas modifiées et donc tous les calculs seront inutiles. Une analogie simple pour mieux comprendre ce principe est de penser à un événement dans son agenda. Il n'est pas utile de se préparer pour l'événement toutes les heures jusqu'à la date de celui-ci. C'est identique pour la simulation sauf que dans ce cas, il est possible d'avancer dans le temps jusqu'à l'événement (si aucun autre événement ne se produit entre temps).

Dans le livre Simulation With Arena [2], le dernier élément présenté est "*Important mais parfois négligé*". Il s'agit du **démarrage** et de l'**arrêt** de la simulation.
Des conditions de démarrage doivent être explicitées, notamment par le biais des variables globales. Pour les conditions d'arrêt, le choix dépendra de l'objectif de la simulation. Par exemple, on peut déterminer une contrainte de temps (arrêt de la simulation au bout d'un certain temps), une contrainte de production(arrêt lorsque X items ont été fabriqués) ou même des contraintes plus spécifiques (arrêt lorsque Y ressources sont tombées en pannes). Bien sûre, il est possible de spécifier plusieurs conditions de fin.

4 Le système d'une simulation

Pour faire une simulation, il faut un système à simuler. Le système est l'élément le plus complexe de la simulation. Ce système possède un état appelé simplement *état du système*. C'est cet état qui va être suivi au long de la simulation. A tout instant de celle-ci, l'état du système renseignera la condition de tous les éléments du système.
Dans le contexte de ce projet, un *système* peut être défini comme suit :

Système Une "*agrégation ou un assemblage d'objets reliés par quelques interactions ou interdépendances*" [1].

Cette définition correspond correctement aux ateliers de fabrication. L'agrégation d'objets est représentée par le parc de machines de l'atelier et son personnel. En ce qui concerne les interactions, les différents éléments de l'atelier se communiquent les matériaux tout au long du raffinement de ceux-ci.

4.1 Les éléments d'un système

Le terme **élément** est assez vague lorsque l'on parle d'un système. Je vais donc préciser les notions que regroupe ce mot.

Un système se compose de trois types d'éléments. Le premier est la notion d'**entité**. Chaque entité possède des **attributs** et enfin, les entités subissent ou engendrent des **activités** qui font réagir le système.

Voici quelques exemples pour chacune de ces notions [1] :

Entité : un objet d'intérêt dans le système comme une voiture dans une simulation de station d'essence, un client dans un magasin, un message sur un réseau, une machine dans un atelier

Attribut : une propriété d'un objet comme la vitesse d'une voiture, la distance entre deux points, une longueur, une priorité, le statut d'un crédit, le capital d'un compte

Activité : un événement, une suite d'actions qui entraîne un changement dans le système comme transmettre un message, conduire d'un point à un autre, déposer de l'argent sur un compte, transformer un matériau

Avec ces trois notions, on peut représenter complètement un système. Cependant, il est nécessaire de caractériser le système choisi avec les propriétés présentées dans les paragraphes suivants.

4.2 Types de système

Tous les systèmes n'ont pas les mêmes propriétés. Dans cette section, je vais énumérer différentes caractéristiques que possèdent les systèmes.

4.2.1 Statique ou Dynamique

Un système peut être **statique** ou **dynamique**. La différence se situe dans le fait que pour des systèmes dynamiques, les interactions influençant le système entraînent des changements au sein de la composition ou du fonctionnement de celui-ci.

Statique Le temps n'a pas d'influence sur les entités du systèmes, aussi bien sur leurs attributs que leur nombre.

Dynamique Les entités peuvent évoluer au fil de la simulation. Certaines peuvent être retirées pendant une période de temps, d'autres voir leurs performances modifiées.

Pour donner un exemple, prenons le cas d'une usine. Des matériaux arrivent dans l'usine, des machines transforment les matériaux et ceux-ci sortent de l'usine une fois raffinés. Dans un système statique, les machines sont incassables et inusables. Pour la version dynamique, les machines peuvent s'user et donc tomber en panne.

Les systèmes statiques sont plus simples à étudier mais les systèmes dynamiques sont plus courants.

4.2.2 Discret ou Continu

Un système peut aussi être considéré *discret* ou *continu*.

Discret : Les variables du système changent *instantanément* de valeur lors du passage d'un point temporelle à un autre.
Exemple : une montre à affichage digital qui passe de minute en minute (de 16h04 à 16h05 par exemple) représente un système discret.

Continu : La valeur d'une variable dépend de la valeur du temps de la simulation.
Exemple : Pour une montre à aiguilles il est possible de calculer la position de chaque aiguille en connaissant le temps de la simulation. Si on considère qu'il y a 60 positions possibles pour une aiguille et que la position 0 représente midi alors à chaque instant on a :
Position de la petite aiguille = (5 * Heure) modulo 60 et Position de la grande aiguille = Minute
Á 16h04, la petite aiguille est à la position (5 * 16) modulo 60 = 20 (qui correspond à la position du chiffre 4 sur un cadran) et la grande aiguille est sur la position 4.

Il n'est pas nécessaire que toutes les parties d'un système réagissent de façon discrètes ou continues pour considérer le système comme tel. En fait, rare sont les systèmes qui sont discrets ou continus en tout point.[3]

4.2.3 Orientée événements ou orientée processus

Cette caractéristique est plutôt liée à la simulation elle-même mais elle trouve tout de même sa place dans cette partie.
Le choix de cette caractéristique entraîne un choix entre deux manières de représenter une simulation.

Orientée événements Dans ce mode de vision, les entités (voir définition 4.1) sont des *gestionnaires d'événements*. C'est à dire que se sont des événements qui peuvent en plus créer ou annuler d'autres événements. La simulation reçoit au début une liste de tous les événements qui vont se produire au fil du temps. Certains événements peuvent être fixer la date d'apparition d'autres événements.

Orientée processus Pour ce paradigme, une entité est constituée d'une suite d'événements. Á son lancement, la simulation reçoit la liste d'apparitions des entités.

Pour citer un exemple, je vais me baser sur un exemple de simulation d'arrivées et de départs d'avions dans un aéroport (event-oriented [WWW2] slides 8 et 13, process-oriented [WWW3], slide 4 et 7).

Cas *orienté événement* :
Trois événements sont définis (Arrivée dans l'espace aérien, Atterrissage, Départ). Au début, on programme toutes les arrivées en indiquant l'heure à laquelle se produit événement. Puis on lance la simulation. Dans cette exemple, lorsque l'événement "Arrivée dans l'espace aérien" se produit, si la piste d'atterrissage est libre, un événement d'atterrissage est programmé et la piste est considérée utilisée. Sinon, aucun événement n'est généré et l'avion "attend son tour" dans les airs. En fait, il faut qu'un événement "Atterrissage" soit crée pour qu'il puisse se poser.
Lorsqu'un événement d'"Atterrissage" se produit, si il reste des avions dans les airs, un nouvel événement d'Atterrissage est crée et programmé puis un événement de "Départ" est programmé. Sinon, seul l'événement de Départ est crée.
Enfin, lorsque l'événement "Départ" se produit, aucun autre événement n'est généré.

Cas *orienté processus* On défini le processus que va suivre un avion : Arrivée, Atterrissage, Départ. Au début de la simulation, on choisi la date de création de tous les avions. Puis on lance la simulation.
Lorsqu'un avion est crée, il arrive dans l'espace aérien et il attend que la piste soit libre.
Une fois la piste libre, il se pose, et se prépare pour le départ.
Á l'heure de son départ, il décolle.

Ces deux orientations représentent le même système mais la représentation des entités est différentes. Dans le premier cas, les entités n'existent que par les événements en jeu dans la simulation. Si il y a un événement "Atterrissage" programmé, cela signifie qu'il reste au moins 1 avion dans les airs.
Pour le second cas, les entités sont des processus qui comportent les précédents événements. Ainsi, on peut connaître l'avancement d'une entité (un avion dans notre cas) dans son processus en regardant l'action qu'elle est en train d'effectuer. Cette vision de la simulation nécessite des primitives de synchronisation pour gérer les différents processus (attente d'une ressource, attente de la fin d'un calcul, reprise d'un événement, ...).

En conclusion, la version orientée événement est plus simple à mettre en oeuvre puisqu'elle limite les besoins en primitives de synchronisation. Ce point la rend aussi plus rapide.
La version orientée processus permet de simplifié le développement du modèle et la mise à jour de celui-ci car les entités sont décrites par une suite d'actions permettant de suivre le principe d'encapsulation des données. De plus, cette représentation devient intuitive une fois mise en place. Cependant, il faut gérer la synchronisation des entités ce qui entraîne de l'"overhead" [1] à cause des changements de contexte.

4.2.4 Déterministe ou Stochastique

La dernière caractéristique concernant les systèmes est le comportement des interactions.

Déterministe Si il est possible de complètement prévoir le résultat d'une action(d'un événement) de la simulation en fonction des données d'entrées alors cette action est *déterministe*.

Stochastique Si les résultats d'une action ont la capacité de varier **aléatoirement** alors elle est dite *stochastique*.

Un système peut posséder à la fois des tâches déterministes et stochastiques.

4.2.5 Type de système utilisé dans le cadre du projet

Pour simuler le fonctionnement d'ateliers de fabrication, nous allons nous intéresser à des systèmes qui seront **dynamiques**(du moins en parti), **discrets** , **orientés processus** et **stochastiques**.

De façon plus concise, le sujet étudie les **simulations discrètes orientées processus**.

Le côté **dynamique** provient de l'usure potentielle de l'équipement qui doit être remplacé, entraînant des modifications dans la composition du système.
L'aspect **discret** correspond bien car les états intermédiaires d'un matériau pendant son raffinement ne sont pas intéressant pour étudier les ateliers de fabrication. Ce qui est plus pertinent est de savoir si le matériau est transformé ou non.
La vision **orientée processus** est très intuitive pour un atelier de fabrication. On peut s'imaginer pour simplifier que l'atelier est constitué d'un tapis roulant qui passe par plusieurs machines.
Enfin, la qualification du système de **stochastique** est dû à l'utilisation de lois de probabilités pour représenter les temps d'arrivées et de traitements des matériaux dans la chaîne de fabrication des ateliers.

Ces choix permettent une représentation à la fois **assez complète** (on introduit du dynamisme et de l'aléatoire pour se rapprocher de la réalité) et assez concise (on utilise une représentation discrète et orientée processus).

4.2.6 Vitesse de calcul d'une simulation

Il est possible de choisir entre deux approches pour dérouler une simulation :

1. L'*overhead* désigne un traitement à effectué en plus de l'essentiel.

Temps Réel

Le *Temps Réel* est utilisé lorsque la simulation incorpore des interactions humaines extérieures, ou alors pour surveiller l'évolution d'un algorithme au cours du temps. L'un des avantages est que le déroulement de la simulation peut être surveillé comme il le serait en réalité. L'un des inconvénients est que si la simulation se déroule sur plusieurs mois ou plusieurs années, le temps réel n'est pas envisageable si on doit prendre une décision rapidement.

Rapidement Que Possible (As Fast As Possible)

Dans le cas du *Aussi Rapidement Que Possible*, il est tout à fait possible de dérouler des simulations qui s'écoulent sur plusieurs années en un temps raisonnable mais il faut un bon système de *feedback* pour analyser ce qui s'est passé dans la simulation car on ne peut pas "suivre humainement" le déroulement. Il faudra se baser sur les résultats et les sorties de l'exécution, d'où un peu plus de post-traitement des résultats.
Comme il est possible de choisir le *pas* de la simulation, il faut faire attention à ne pas mettre n'importe quelle valeur. Si le pas est trop grand, certaines actions seront exécutées dans le même pas (donc "en même temps"), ce qui peut entraîner des erreurs ou introduire des approximations dans la simulation ainsi qu'une perte de précision. Par exemple, prendre un pas trop petit ralentira la simulation. Si on prend l'évolution de la population d'une ville, fixer le pas à *une heure* renseignera beaucoup d'entrées peu utiles, c'est-à-dire où il ne se passe pas d'événement d'intérêt.

4.3 Comment construire une bonne représentation d'un système ?

La représentation du système, ou modélisation, est une part très importante de la simulation. Cependant, la construction de cette représentation est une tâche analogue à la conception d'un logiciel, il n'existe pas d'algorithme ou de méthode pour passer d'un système à sa représentation. C'est une tâche assez complexe qui est pourtant critique dans la réussite de la simulation. Mais il existe tout de même des lignes directrices à suivre et quelques indicateurs de qualité.

Dans l'introduction, j'explique qu'une simulation ne doit pas être une copie conforme du système réel. La raison à cela est que les systèmes peuvent être très compliqués et cela entraîne la complexité de la simulation associée. Le problème qui va survenir se lie à la compréhension de la simulation par une personne extérieure à sa construction. Il est tout à fait possible que la personne construisant la simulation soit différente de la personne l'utilisant. De même, plusieurs personnes peuvent être amenées à travailler sur une même simulation. Il faut donc qu'elle soit suffisamment simple pour pouvoir être utilisée par la suite. Cependant, la simulation ne doit pas être trop simple non plus, car le risque de ne pas prendre en compte une des spécificités essentielles du système augmente.
Le travail de représentation du système consiste donc à extraire les éléments caractéristiques du système réel pour les retenir dans la simulation sans trop détailler. Cela peut se traduire par augmenter le niveau d'abstraction de certaines parties du système ou bien par regrouper plusieurs éléments en un seul.

L'*état du système* recense la situation de chaque entité, attribut et activité. Pour trouver tous les éléments qui composent un systèmes, il faut dans un premier temps repérer les objets/concepts intéressants puis leur rattacher leurs propriétés et enfin, trouver les actions engendrées par ces objets.

C'est à ce moment que la définition des *limites* du système doit être effectuée. Cela consiste à déterminer ce qui est interne au système et ce qui est externe. Par exemple, une usine possède des machines pour raffiner les matières premières. Ces machines sont internes au système. Par contre les fournisseurs de matières premières sont externes au système de l'usine. Ce qui est situé en dehors des limites du système se nomme *l'environnement*. Une activité qui se déroule à l'intérieur du système est qualifiée d'*endogène*. Si elle provient de l'environnement mais qu'elle a un impact sur le système, elle est dite *exogène*. Un système qui ne connaît aucune activité exogène est dit *fermé*. Sinon, il est *ouvert*.

Une question est soulevée dans le livre de G.Gordon [1] : Si les résultats d'une activité endogène sont aléatoires à un moment donné, cela signifie que l'on ne connaît pas son *état* dans l'état du système. Il

faudrait alors considérer cette activité comme *exogène*. Cependant, l'aléatoire peut se mesurer et être contrôler (en partie) notamment si les données aléatoires suivent une loi de *probabilité*. Le constat suivant est alors défini, si les apparitions d'une telle activité sont sous contrôle du système, alors elle sera considérée *endogène*. Sinon, si les occurrences de l'activité sont indépendantes du système, elle devient *exogène*.
L'exemple utilisé pour illustrer est très pertinent, je vais donc le citer :
Dans le cadre d'une usine, le temps nécessaire pour qu'une machine finisse sa tâche peut varier mais il peut être assimiler à une loi de probabilité. La tâche effectuée par une machine est alors *endogène*.
Cette usine a besoin d'électricité pour faire fonctionner ses machines. Il peut arriver qu'une coupure de courant survienne de temps en temps. Ces coupures seront considérées comme des activités *exogènes* car le système n'a pas d'influence sur ces coupures ni sur les intervalles entre chacune d'elles.

La prochaine section propose un exemple de système avec le repérage de ses éléments.

4.4 Exemple simple de système

L'exemple que je vais détailler concerne un atelier de fabrication très simple avec une machine et un ouvrier qui redémarre la machine toutes les heures. Cet atelier transforme du verre en petit grain de verre.

Entité : Le verre qui arrive dans l'atelier.
Attributs : On connaît l'âge du verre qui arrive et sa couleur.
Ressources : La machine et l'ouvrier **Indicateurs Statistiques** : Le poids de verre transformé.
Processus : Traiter un arrivage de verre, Redémarrer la machine.

Voici un exemple d'éléments qui peuvent être extraits. Cependant, sont-ils tous utiles ? et sont-ils tous présents ? Pourquoi avoir pris en compte l'âge du verre et sa couleur ?

4.5 Le choix des éléments pertinents

Le point qui va permettre de répondre à ces questions est la définition du *problème* que doit résoudre la simulation. Si le problème possède une question sur la moyenne d'âge du verre traité, l'attribut est pertinent et même nécessaire.
Avant de commencer à modéliser le système de la simulation, il faut donc **déterminer le ou les problèmes** que la simulation doit aider à résoudre.

5 Premières étapes pour créer une simulation

Á ce stade du rapport, je vais proposer un début d'ordre (inspiré de [5]) pour créer une simulation, afin de résumer les précédentes sections.

1. La première étape consiste à **définir le problème**. Il faut pouvoir répondre aux questions suivantes après cette définition : Pourquoi étudier ce problème ? et Quelles sont les questions auxquelles on veut pouvoir répondre ?
2. Après ce premier point, il faut commencer à **cerner les limites du système** à étudier. C'est à cette étape que le fonctionnement du système doit être assimilé dans sa totalité.
3. Une fois le système délimité, il faut en déterminer les **entités**, les **attributs** et les **activités**. Puis, les trier pour ne garder que les plus pertinents face au problème défini.

5.1 Exemple complété

Reprenons l'exemple ci-dessus. Je vais suivre les étapes vues ci-dessus pour créer un début de conception de simulation.
La question à laquelle je souhaite répondre est : Quelle est la proportion de verre de couleur rouge traitée en 1 journée (24 Heures) ?

Le système se contente de réceptionner le verre et de le transformer. Il ne gère pas le transport de celui-ci ni son stockage.

Entité : Le verre qui arrive dans l'atelier.
Attributs : La couleur du verre.
Ressources : La machine et l'ouvrier.
Indicateurs Statistiques : Le poids total de verre transformé, le poids de verre de couleur rouge transformé.
Processus : Traiter un arrivage de verre, Redémarrer la machine.

Si on considère que les arrivages ont 100% de chances d'être à l'heure sans problème, ces deux processus sont alors internes à l'entreprise et ne sont pas dépendant de l'environnement extérieur. La simulation travaille donc sur un système **fermé**.

Avec toutes ces informations, la simulation possède son squelette mais il manque encore les muscles pour la faire avancer.

6 Les données d'entrée de la simulation

Après ces premières étapes, nous avons construit le moteur de la simulation. Maintenant, il faut préparer le carburant et pouvoir estimer si le moteur fonctionne correctement.

Le carburant d'une simulation correspond à ses données d'entrées. Ce sont ces données qui vont initialiser l'état de départ du système et définir les lois de probabilités des activités à occurrences aléatoires avec leurs paramètres. Dans l'exemple précédent, il faut déterminer les règles d'arrivées du verre : Est-ce que l'arrivage est régulier (ex : 1 par heure), est-ce qu'il suit une loi de probabilité (Uniforme, Exponentielle) ? Si c'est une loi, quelles sont ses paramètres ?

6.1 Déterminer les lois de probabilités d'un système

Le choix des lois de probabilités et de leur(s) paramètre(s) est très important pour la conception d'une simulation fidèle au système.

Pour choisir les bonnes lois, il faut étudier et comprendre le comportement des événements à caractériser. Pour cela, si le système existe déjà, on peut utiliser des relevés de données pour voir si la distribution de ces données s'approche d'une loi en particulier. Si le système n'existe pas, il faut créer un relevé de données à partir des caractéristiques du futur système et de son environnement. S'il n'est pas possible de déterminer un seul comportement pour certains événements, il peut être intéressant de tester le système avec différents comportements pour anticiper les difficultés de chaque comportement avant la réalisation du vrai système.

6.2 Exemple de détermination de loi de probabilité

Pour savoir si une distribution de valeurs suit une loi de probabilité, il faut effectuer un test du χ^2 (Khi Deux).

Voici un exemple :
Il y a 300 arrivages de verre dans la journée et je voudrais savoir si la couleur du verre qui arrive suit une loi uniforme. Le verre peut être Bleu, Verre ou Rouge (donc 1 chance sur 3 d'avoir du Bleu en théorie).

Sur 1 journée, on recense la couleurs du verre qui arrive :

Couleur du verre	Bleu	Vert	Rouge
Nombre recensé	95	112	97

Puis on applique le test du χ^2 sur ces données :

Pour cela, on pose l'hypothèse suivante :
H_0 : la couleur des bouteilles de verre, de fonction de répartition F, suit une loi uniforme de fonction de répartition connue F_0. On a donc : $F = F_0$

On se fixe un taux d'erreur $\alpha = 5\%$. Ce taux signifie que si l'on rejette H_0 il y a 5% de risque que l'on se soit trompé.

L'étape suivante consiste à déterminer la répartition théorique suivant la loi choisie (ici loi uniforme). On a autant de chance que la couleur d'une bouteille soit bleue, rouge ou verte.

Voici le tableau de répartition théorique :

Couleur du verre	Bleu	Vert	Rouge
Nombre Théorique	100	100	100

On va maintenant calculer la somme des écarts entre les valeurs pratiques et les valeurs théoriques.

$$T = \sum_{i=[Bleu,Vert,Rouge]} \frac{(n_{ipratique} - n_{itheorique})^2}{n_{itheorique}}$$

Ce qui donne avec les valeurs de l'exemple :

$$T = \frac{(95-100)^2}{100} + \frac{(112-100)^2}{100} + \frac{(97-100)^2}{100}$$

$$T = 1.78$$

On regarde ensuite dans la table du χ^2 la valeur-p correspondant à $\alpha = 5\%$ pour 2 degrés de liberté : on a 3 classes et on n'estime aucun paramètre pour la loi uniforme (ie : la moyenne, l'écart type) donc le degré de liberté de l'exemple est 3 - 1 = 2.
La table donne $valeur_p = 5.99$.

On a donc : $P(T < 5.99) = 1 - \alpha = 0.95$

On peut donc accepter l'hypothèse H_0 au risque de 5 %. En conclusion, il est acceptable de considérer que la couleur du verre qui arrive suit une loi uniforme au risque de 5 % .

6.3 Choix des lois pour l'exemple

Dans cet exemple, on va considérer que les arrivages se produisent suivant la loi exponentielle d'espérance $\lambda = 0.5 Heure : \xi(0.5)$ et que chaque arrivage possède 1 Tonne de verre unicolore.

Le traitement du verre suit une loi constante de paramètre 0.1 Heure. Dans cette simulation, on considère que le temps de traitement du verre est largement inférieur à l'intervalle de temps séparant deux arrivages.

Enfin, la machine s'arrête suivant une loi exponentielle de paramètre 4 Heures et l'ouvrier peut la remettre en marche immédiatement.

7 Évaluation du modèle

La simulation est construite, il faut maintenant procéder à une évaluation du modèle établi. Comme le rappelle le livre *Computer Simulation Techniques* "il n'y a que peu de bénéfices à continuer une simulation [dont le modèle est mal établi] en l'informatisant car cela reviendrait tout simplement à 'simuler notre propre ignorance' " [4].

Cependant, comment procéder pour estimer la justesse du modèle à ce niveau du développement de la simulation ?

Il est important de noter qu'à ce niveau du développement de la simulation, ce test de justesse ne concerne que la faisabilité et l'intégrité de l'implémentation future de la simulation ainsi que le choix des **caractéristiques de fonctionnement** (operating characteristics). La vérification de la pertinence des données générées par la simulation s'effectue dans une étape ultérieure.

Les questions auxquelles il faut pouvoir répondre correctement sont les suivantes [4] (page 36-37) :

- Est-ce que toutes les variables choisies sont pertinentes ? Une variable est caractérisée ainsi si elle participe positivement à l'augmentation de la justesse du comportement des variables endogènes du système.
- Est-ce que toutes les variables exogènes qui pourraient agir significativement sur les variables endogènes du systèmes sont recensées et incorporées dans le modèle ?
- Toutes les relations entre les variables endogènes et exogènes sont-elles correctement établies ?
- Les estimations des différents paramètres des caractéristiques de fonctionnement du système sont-elles proche de la réalité ? (par exemple, estimateurs biaisés, non biaisés, ...)
- Les estimations des paramètres des caractéristiques de fonctionnement du modèle sont-elles proches de la réalité ? (par exemple, le choix des lois de probabilités pour la génération de certaines données)
- Est-ce que, après applications numériques "à la main", les valeurs théoriques des variables endogènes se comportent de façon similaire à leur équivalent en situation réelle ?

Pour continuer la conception de la simulation, il faut avoir répondu favorablement à toutes les questions ci-dessus. Dans le cas contraire, il faut reprendre les étapes précédentes de la conception pour pallier aux réponses négatives.

8 Écriture du programme informatique de la simulation

Jusqu'à présent, nous avons décrit les étapes pour créer une simulation. Tout est désormais prêt pour traduire cette simulation en programme informatique. Pour se faire, on peut soit créer la simulation de

toute pièce en intégrant une gestion de l'avancement du temps et de l'ordre d'exécution des processus. Ou alors on peut utiliser des logiciels qui implémentent déjà ces contraintes de temps et d'exécution et qui propose de construire une représentation graphique de notre simulation : c'est le cas d'Arena [WWW5].

Dans le cadre de ce projet, nous allons utiliser une bibliothèque Python nommée SimPy [WWW7]. Elle propose une gestion déjà implémentée des bases de la simulation (horloge, queue de processus). Cette bibliothèque sera présentée en détail plus tard dans le rapport.

Pour ce qui est des principaux éléments à prendre en compte pour implémenter une base de simulation, vous pouvez vous reporter à la section suivante. Cependant, cette organisation et ces algorithmes ne sont valables que pour des simulations orientées processus seulement.

9 Algorithmes de base d'une simulation orientée processus

Cette section décrit des algorithmes qui peuvent être utilisés pour mettre eb place une simulation orientée processus.

9.1 Représentation d'une Simulation

Une simulation se compose des éléments suivants :

- Une **liste des processus** à exécuter trier du processus le plus urgent au moins urgent. Un processus est plus urgent qu'un autre si son heure d'exécution est plus tôt. Les processus sont donc classés en ordre d'exécution.
- L'**Heure** de la simulation.
- Le **processus en cours d'exécution**.

Pour pouvoir utiliser la liste des processus dans la simulation, il faut être capable d'insérer un processus au bon endroit dans la liste et de retirer un processus de la liste. Cette liste est représentée sous forme de Heap dans Simpy.

Au début de la simulation, il doit y avoir au moins un processus, sinon la simulation se terminera immédiatement.

9.2 Représentation d'un Processus

Un processus est une suite d'action qui peut se dérouler pendant la simulation. Par exemple, :

Arriver dans une file d'attente, Attendre son tour, Lorsque c'est notre tour réaliser notre action, Attendre la fin de l'action, Quitter le système.

Chaque processus à exécuter doit être ajouté à la liste des processus en fonction de leur **Heure d'exécution**. Cette heure correspond au moment de la simulation où le processus tentera de s'exécuter. J'utilise le verbe "tenter" car il y a un autre requis. En effet, chaque processus possède une **condition d'exécution**. Elle représente la concaténation des conditions dans lesquels la simulation doit se trouver pour que le processus puisse poursuivre son exécution. En quelque sorte, l'heure d'exécution est un cas spécial de condition d'exécution.
Pour résumer, un processus nécessite deux requis pour pouvoir s'exécuter : son heure doit être arrivée et ses conditions d'exécutions doivent être réalisées.

9.3 Algorithme général

Voici une version synthétique de l'algorithme que suit une simulation orientée processus :

```
Data : ListeProcessus : une liste de processus à réaliser avec leur Heure de départ.
Result : La simulation est terminée → Heure de la simulation ≥ Heure à atteindre
.
while not SimulationFinie() do
    Processus = ProchainProcessus(ListeProcessus);
    if Processus == null et ListeProcessus.Taille > 0 then
        ReplanifierProcessus() ;
        AvancerTemps(1) ;
    else
        AvancerTemps(Processus.HeureDebut);
        ReprendreProcessus(Processus);
    end
end
```

Algorithme 1 : Algorithme général de la simulation orientée processus

9.4 Détection de la fin de la simulation

Pour configurer la fin de la simulation, plusieurs choix sont possibles. On peut déterminer un certain nombre de pas à effectuer avant de stopper la simulation, on peut attendre la réalisation d'une condition (ou d'un événement) ou on peut laisser la simulation s'exécuter jusqu'à l'épuisement de tous les processus de sa liste.

```
Data : Heure de la simulation, Heure de fin de la simulation, condition(s) d'arrêt(s)
Result : Retourne Vrai si Heure de la simulation ≥ Heure de fin de la simulation, Sinon
         Retourne Faux
.
```

Algorithme 2 : SimulationFinie : Algorithme déterminant la fin de la simulation

La simulation se déroule tant qu'elle n'a pas atteint sa (ou ses) condition(s) d'arrêt. Cet arrêt est implémenté par l'ajout d'un processus dans la liste des processus qui se réalisera au moment configuré. Dans ses instructions, ce processus termine la simulation et les processus situés après dans la liste ne seront pas exécutés.

9.5 Sélection du processus à exécuter

Le principe de l'algorithme est le suivant, on essaye de récupérer le processus le plus imminent (dont l'heure d'exécution est la plus tôt) et dont les conditions d'exécutions sont réalisées. Si la liste est vide, il n'y a pas de processus à exécuter (fin de la simulation ?) on retourne donc **Null**. Sinon, on retourne le processus à exécuter.

D'une façon assez intuitive, on peut imaginer que si le processus ne rempli pas ses conditions d'exécutions, il est reprogrammé pour le prochain pas de la simulation. Si il peut s'exécuter, il est retiré de la liste et réalise ses instructions. On fait de même pour tous les processus programmés pour le même pas, ainsi, on exécute tous les processus.
Mais, il y a un cas qui n'est pas correctement pris en compte avec cette méthode.

Considérons P1, processus qui possède une ressource et qui doit la libérer à l'instant **t**. P2, processus souhaitant accéder à la ressource que possède P1.

Premier scénario : ListeProcessus = [P1, P2]

```
Data : ListeProcessus : une liste de processus à réaliser avec leur Heure de départ.
Result : Le prochain processus à exécuter : Soit P le processus retourné,
         P.ConditionExecution = Vrai et ∀ Pi ∈ ListeProcessus \ P , P.HeureDebut ≤
         Pi.HeureDebut ou Null si aucun processus ne peut s'exécuter
.
if ListeProcessus.Taille ≤ 0 then
    return null ;
else
    Processus = ListeProcessus[0];
end
trouvé = ListeProcessus.ConditionExecution ;
i = 0 ;
while trouvé = Faux et i < ListeProcessus.Taille do
    if ListeProcessus.ConditionExecution = Vrai then
        trouvé = Vrai ;
    else
        trouvé = Faux ;
    end
    i++;
    Processus = ListeProcessus[0];
end
if trouvé = Faux then
    return null ;
else
    ProcessusARetourner = ListeProcessus[i] ;
    ListeProcessus ← ListeProcessus \ ListeProcessus[i] ;
    return ProcessusARetourner ;
end
```

Algorithme 3 : ProchainProcessus : Algorithme déterminant le prochain processus à exécuter

Á l'instant t :
P1 s'exécute et libère la ressource
P2 récupère la ressource
Passage à l'itération suivante

Second scénario : ListeProcessus = [P2, P1]
Á l'instant t :
P2 essaye de récupérer la ressource mais elle est toujours occupée par P1
P1 libère la ressource
Passage à l'itération suivante

Á l'instant t+1 :
P2 récupère la ressource

On observe que selon l'organisation de la ListeProcessus, le résultat de la simulation peut varier et introduire des erreurs dans les résultats.

Pour résoudre cette difficulté, il n'y a pas de solution générique. Cependant, il y a deux pistes que l'on

peut considérer.

L'affectation de priorité à chacun des processus est une bonne réponse à ce problème s'il est possible de classer tous les processus selon une priorité pertinente (exemple : tous les processus libérant des ressources devraient s'exécuter au plus tôt pour éviter le problème précédent).

Une autre possibilité consiste à boucler sur les processus à exécuter au même moment jusqu'à ce qu'aucun d'entre eux ne réussisse à s'exécuter. Un des inconvénient est que dans le pire des cas, la complexité est en O(!n) avec n le nombre de processus devant s'exécuter à un moment de la simulation. Ceci peut entraîner un ralentissement de l'exécution de la simulation.

Enfin, il est possible de combiner les deux méthodes s'il n'est pas évident de classer tous les processus par ordre de priorité et si la simulation est trop ralentie.

Dans cet algorithme, on ne replanifie pas les processus qui n'arrivent pas à s'exécuter. Cette replanification est effectuée dans l'algorithme principale 9.3.

Il faut maintenant expliquer comment un processus peut être amené à s'**arrêter** pendant son exécution.

9.6 Algorithme d'un Processus Lambda

Chaque processus possède son propre algorithme, je vais donc présenter un algorithme qui possèdera des instructions spéciales qui déclenchent des mécanismes propres à la simulation orientée processus. Ces instructions sont Attendre(Délai) et AttendreQue(Conditions).

Data : Rien
Result : Toutes les instructions ont été exécutées
.
instruction1 ;
instruction2 ;
Attendre(10) ;
instruction3 ;
instruction4 ;
AttendreQue(Ma Condition = Vrai)
instruction5 ;

Algorithme 4 : Algorithme d'un Processus Lambda

9.6.1 La fonction Attendre(Délai)

L'instruction Attendre(Délai) indique que le processus doit arrêter son exécution pendant le nombre de pas **Délai**. Voici l'algorithme de cette fonction :

Lorsque le processus commence son exécution (ou la reprend), il est retiré de la liste des processus. Lorsque dans ses instructions il rencontre un appel à Attendre(Délai), il va se reprogrammer dans la liste suivant sa nouvelle Heure d'exécution (Planifier(Processus)). Puis, il interrompt son exécution (Quitter(Processus)) et l'algorithme de la simulation procède au choix d'un nouveau processus à exécuter. Á la reprise de son exécution, le processus repars à la ligne suivant l'appel à Attendre(Délai).

Data : Délai : Le nombre de pas de simulation pendant lesquels le processus sera à l'arrêt.
Result : Le processus arrête son exécution et il est correctement replanifié.
.
Processus.HeureExécution = Simulation.Heure + Délai ;
Planifier(Processus) ;
Quitter(Processus) ;

Algorithme 5 : Attendre : Algorithme pour arrêter l'exécution d'un processus pour un certain nombre de pas de la simulation.

9.6.2 La fonction AttendreQue(Conditions)

L'autre fonction capable d'interrompre un processus est la fonction AttendreQue(Conditions). Son fonctionnement est assez identique à Attendre(Délai). Voici son algorithme :

Data : Conditions : La concaténation des conditions d'exécution du processus.
Result : Le processus est arrêté, son exécution est correctement replanifiée et
Processus.ConditionExecution = p.
.
Processus.HeureDebut = Simulation.Heure + 1 ;
Processus.ConditionExecution = p ;
Planifier(Processus) ;
Quitter(Processus) ;

Algorithme 6 : AttendreQue : Algorithme pour arrêter l'exécution d'un processus jusqu'à ce que sa condition d'exécution devienne Vrai.

On retrouve ici les fonctions Planifier(Processus) et Quitter(Processus) qui ont le même comportement que pour la fonction Attendre(Délai). Dans cet algorithme, la condition d'exécution du processus courant est modifiée au profit d'une nouvelle. Cependant, on ne peut pas savoir l'heure à laquelle cette condition deviendra Vrai. On doit donc reprogrammer l'exécution du processus pour le prochain pas de la simulation afin qu'il puisse de nouveau essayer de s'exécuter.

Enfin, on pourrait avoir une fonction AttendreQue(Conditions, Délai) qui reprogrammerait les processus non pas pour le pas suivant mais après un certain délai, comme pour la fonction Attendre(Délai).

Data : t : Le nombre de pas de simulation qu'il faut avancer
Result : Soit H l'heure de la simulation avant en entrée, Simulation.Heure = H + t
.
if $t > 0$ **then**
  Simulation.Heure = Simulation.Heure + t ;
else
  Erreur : La simulation ne peut pas reculer
end

Algorithme 7 : AvancerTemps : Algorithme pour faire avancer l'heure de la simulation

Data : Processus : Le processus à reprendre.
Result : Les prochaines lignes de codes à traiter sont celles de Processus.
.

Algorithme 8 : ReprendreProcessus : Algorithme pour reprendre (ou commencer) l'exécution d'un Processus

10 La vérification

Après la réalisation de la simulation informatisée, il faut une fois de plus procéder à une **validation** de la cohérence du travail effectué. C'est une étape qui peut s'avérer très difficile suivant la complexité de la simulation. Le livre *Computer Simulation Techniques* [4] décrit cette vérification comme un cas d'un problème plus général : "Comment valider un modèle ou une hypothèse ?". Pour répondre à cette question, il faudrait pouvoir trouver une réponse à "Á partir de quoi peut-on considérer que l'on a validé une hypothèse ?" et à "Quels critères devraient être utilisés pour assurer la validité d'une hypothèse ?". Malheureusement, le livre renseigne que les réponses sont de l'ordre philosophique et que pour le moment il n'y a pas de réponse retenue à l'unanimité.

Si la réponse exacte n'est pas encore disponible, on peut tout de même observer certains points pour estimer le niveau de validité de notre simulation.

Dans un premier temps, si il existe des données, des informations sur le comportement "attendu" du modèle, on peut les utiliser pour les comparer avec les résultats obtenus avec la simulation. Il suffit de reproduire les conditions dans lesquelles les données ont été recueillies.

Par contre, s'il n'y a pas de données, on peut essayer de prédire le comportement du système à simuler et le comparer avec la réalité.

S'il n'y a ni donnée, ni système existant, il va être assez difficile de déterminer si le modèle est juste.

Le livre [4] (chapitre 8, p310-319) propose plusieurs paradigmes pour tester la validité d'un modèle : le **sysnthétique a priori**, l'**ultra empirisme**, le **multistage verification** et les **économies positives**. Toutes ces méthodes sont plutôt orientées pour des vérifier des modèles économiques mais certains modes de pensée peuvent être appliqués à d'autres modèles. Ensuite, on retrouve la validation par la comparaison de "données historiques" et par "prévision"(forecast).

L'inexistence d'une méthode *parfaite* pour vérifier n'est cependant pas une excuse pour ne pas vérifier le modèle. En effet, si le modèle est faux alors la simulation s'exécutera sur des erreurs et ne pourra donner que des résultats erronés (même s'ils semblent cohérents).

11 Création des expériences

Dans cette phase, il faut choisir les différentes expériences que la simulation va devoir réaliser. Pour chaque expérience, il est nécessaire de déterminer quelles variables seront des **facteurs**(factors), c'est-à-dire les variables que l'on va "contrôler" pendant la simulation (ou alors juste au début) et quelles variables seront des **réponse**(responses). Cette deuxième catégorie représente les variables à observer.

Par exemple (inspiré de [4]), prenons une simulation sur l'étude de l'évolution du nombre de visiteurs dans un zoo. Le problème est le suivant : Est-ce que le nombre de visiteurs augmente si le nombre de girafes augmente ? Si on modifie le nombre de girafes pour observer l'évolution du nombre de visiteurs alors la variable "nombre de girafes" est un **facteur** et la variable "nombre de visiteurs" est une **réponse**. Le facteur peut être choisi arbitrairement ou bien aléatoirement. Il peut y avoir plusieurs facteurs dans

une simulation ainsi que plusieurs réponses. Cependant, les résultats de la simulation dépendront des facteurs et des réponses choisies, ce qui implique qu'un choix mal étudié des facteurs et des réponses peut entraîner des conclusions fausses dans la dernière phase de la simulation : l'**analyse des données simulées**.

Le second point important de cette phase est de réduire l'impact de l'aléatoire sur la simulation. Cet aléatoire est introduit par les lois de probabilités qui régissent les variables aléatoires de la simulation. Pour atténuer l'effet de cet aléatoire, une bonne solution est d'exécuter plusieurs fois la même expérience puis de faire une analyse en prenant en compte les différents résultats de chaque exécution.

12 Analyse des données simulées

La dernière phase de la simulation consiste à analyser et interpréter les données résultant de la simulation.

L'approche pour analyser ces données semblent identiques à celle pour analyser les données d'entrées mais il y a des différences majeurs qui vont venir compliquer cette analyse :

- L'aléatoire introduit dans les simulations est plus complexe que pour les données d'entrées.
- Le dynamisme des modèles construits par la simulation rend étalement l'analyse plus difficile que pour un jeu de données statique.
- Le nombre de paramètres qui évoluent au sein de la simulation est très important et l'impact relatif de chacun de ces paramètres entre eux est quasiment impossible à déterminer.

En dépit de tous ces points qui complexifient l'analyse de données résultant d'une simulation, celle-ci reste possible en utilisant des outils adéquats.

4 Conception des processus de base

Dans ce chapitre, je vais présenter les différents cas de base qui constitueront les "briques" pour construire les simulations d'ateliers de fabrication. Puis, je proposerai un algorithme pour chacun de ces cas.

1 Présentation des cas de base

Voici les différents cas à étudier :

1.1 Le cas élémentaire

Figure 1 – *Cas élémentaire d'atelier de fabrication*

Dans ce cas très simple, il s'agit d'une machine M1 qui transforme des éléments qui arrivent selon un intervalle régulier ou bien une loi de probabilité définie. Ces éléments subissent ensuite un traitement d'une durée là aussi constante ou aléatoire et enfin, après transformation, ces éléments sont placés dans un stock de sortie.

On peut concevoir un processus CréerProduit qui récupère un élément du stock d'entrée puis qui le transforme pour enfin le placer dans le stock de sortie.

1.2 Deux machines en série et un stock intermédiaire limité

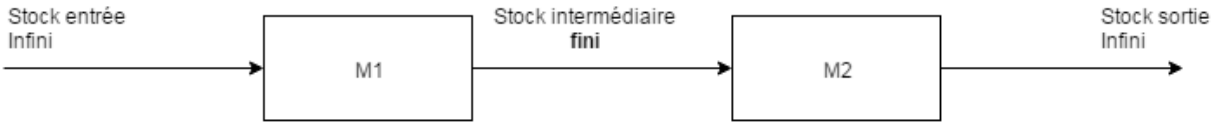


Figure 2 – *Cas d'atelier de fabrication avec deux machines reliées par un stock intermédiaire*

Un cas qui est assez simple également. Cependant, il y a une remarque à noter. Dans le cas élémentaire, il était clair que notre simulation ne permettait la création que d'un seul processus. Mais dans ce cas là, on peut en imaginer trois :

- Le "nouveau" processus **CréerProduit** qui prend un élément du stock de départ, le transforme, attend qu'il y ait de la place dans le stock intermédiaire, place l'élément dans ce stock, ensuite, transformation par M2 et placement dans le stock de sortie.
- Le processus **UtiliserM1** consistant seulement à prendre un élément du stock d'entrée, le transformer et le placer dans le stock intermédiaire dès qu'il y a de la place pour le faire. Ce processus pourrait exister si par exemple, le traitement par M1 est assez long et on aimerai remplir le stock intermédiaire pour anticiper une possible arrivée de commandes et ainsi gagner du temps.
- Le processus **UtiliserM2** qui fonctionne de la même manière que *UtiliserM1*. Si il est possible de remplir le stock intermédiaire sans demander de traitement par M2 par la suite, alors il faut définir un processus pour pouvoir utiliser ces éléments sans forcément relancer un processus *CréerProduit*.

Ces processus semblent intéressants mais pour simplifier les explications et pouvoir concevoir des algorithmes assez simple pour un début de recherche, je vais laisser de côté la possibilité de créer ces processus "UtiliserM1" et "UtiliserM2". Mais attention, je ne les utiliserai pas en tant que **processus** mais par contre, en tant que **fonction**, je vais expliquer qu'ils sont très utiles.

Reprenons notre processus **CréerProduit**. En fait, pour créer un produit, il faut d'abord *utiliser* la machine *M1* puis la *utiliser* la machine *M2*. Donc, les algorithmes "UtiliserM1" et "UtiliserM2" vont être inclus dans l'algorithme de "CréerProduit". En programmation, cela reviendrait à avoir une fonction *CréerProduit* qui exécuterait un appel à chacune des fonctions pour utiliser les machines.

Pour représenter ce cas, on a donc un processus qui correspond à l'appel successif de l'utilisation de M1 et de l'utilisation de M2.

1.3 Le désassemblage

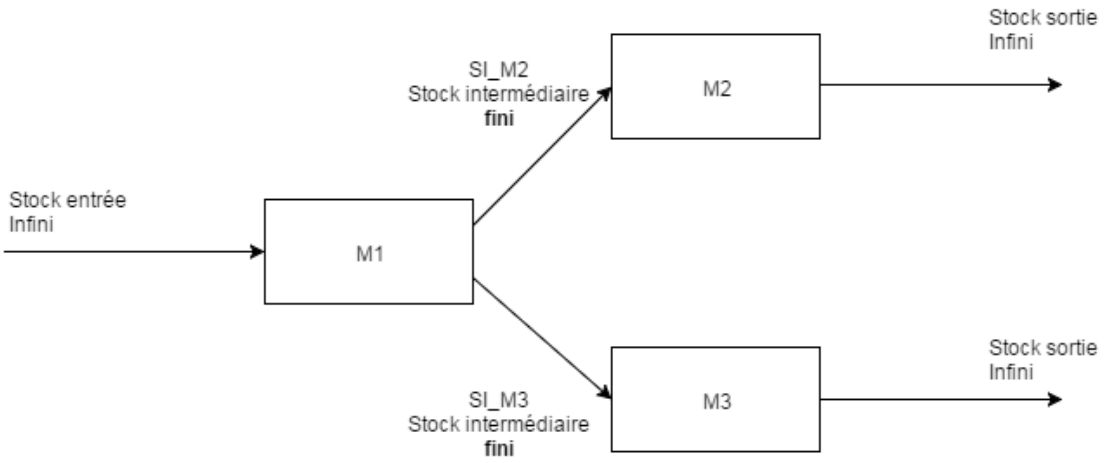


Figure 3 – *Cas de désassemblage*

Ce schéma représente un cas de **désassemblage**. Il peut aussi représenter un cas de **choix de gamme**, c'est-à-dire que l'on peut à partir d'une transformation par la machine M1 choisir si on veut créer un produit passant par M2 **ou** par M3 (c'est un ou exclusif dans ce cas). Dans le cas du désassemblage, une transformation par M1 entraînera une transformation par M2 **et** par M3 mais pas sur le même élément. Par exemple, si M1 consiste à démonter une ampoule, M2 traitera le verre et M3 le filament.

Là encore, je vais procéder à une simplification. Je vais supposer que si on demande de désassembler un élément par M1 alors il y aura exactement suffisamment de matières pour effectuer les autres traitements.

En concret, lorsque l'on désassemble par M1, on obtient juste assez d'élément pour utiliser M2 et pour utiliser M3. La raison de cette simplification est que dans le cas normal, il peut y avoir beaucoup de cas différent à traiter, notamment, si à chaque fois on désassemble "trop" d'élément, au bout d'un moment on va arriver à la limite de capacité du stock intermédiaire, et ce stock ne peut être vidé à cause de la première simplification.

Pour ce cas, si on se place dans un désassemblage, on a un processus qui est constitué des actions suivantes : prendre un élément dans le stock d'entrée, le désassembler avec M1, placer les résultats dans leur queue respective, programmer un traitement par M2 et un traitement par M3, lorsque qu'un traitement se fini, il place le résultat dans son stock de sortie.

Si on est dans le cas d'un choix de gamme alors on a deux processus. Le premier est en fait le cas de deux machines avec un stock intermédiaire en considérant M1 et M2 et le second le même cas en considérant M1 et M3.

Enfin, si l'atelier possède une configuration qui permet de choisir entre un désassemblage et un choix de gamme, alors il y a trois processus qui sont la concaténation du processus du désassemblage avec les deux du choix de gamme.

Dans ce cas, on peut voir l'apparition de la notion de parallélisation des traitements. En effet, il paraît normal de pouvoir faire fonctionner la machine M2 et M3 en même temps. Cette parallélisation est cruciale et elle aura un impact important sur les algorithmes de ces processus.

1.4 Assemblage

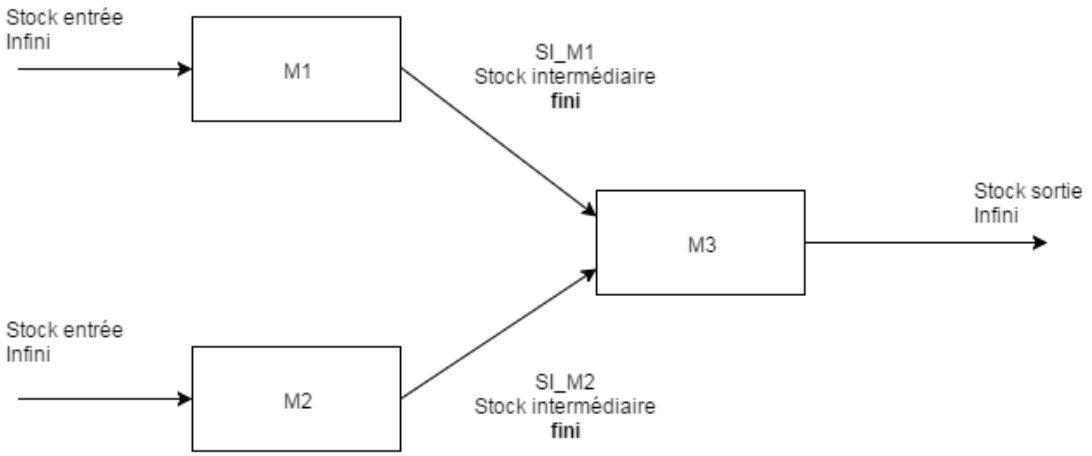


Figure 4 – *Cas d'assemblage*

Le cas d'assemblage comme pour le désassemblage peut représenter deux stratégie : le cas d'assemblage simple, il faut un élément dans chaque stock intermédiaire pour pouvoir employer la machine M3 et le cas de choix de gamme où on peut choisir le pré-traitement du produit avant de le finaliser. Par exemple, pour un tournevis, le manche peut être en bois de chêne (M1) ou en bois d'hêtre (M2) mais possèdera toujours une tige en fer (M3).

Dans le cas de l'assemblage, on a un seul processus qui réalise les actions suivantes : prendre un élément de chacun des stocks d'entrée et les faire transformer par leur machine respective avant de les placer dans les stocks intermédiaires. Puis, lorsqu'il y a de la matière dans chacun des stocks, lancer le traitement de M3. Une fois le traitement fini, le résultat est placé dans le stock de sortie.
Un point très important qui revient ici, est le lancement parallèle d'une demande de traitement par M1 et par M2.

Dans le cas d'un choix de gamme, on retrouve une fois de plus le cas de deux machines avec un stock intermédiaire par les couples (M1,M3) et (M2,M3). Il n'y a pas de grand changement par rapport au cas précédent à part pour l'ordre d'exécution des machines.

Enfin, on peut aussi imaginer un atelier permettant de faire à la fois assemblage et choix de gamme avec ces trois machines. L'explication est identique à celle du cas précédent.

1.5 Désassemblage / Assemblage

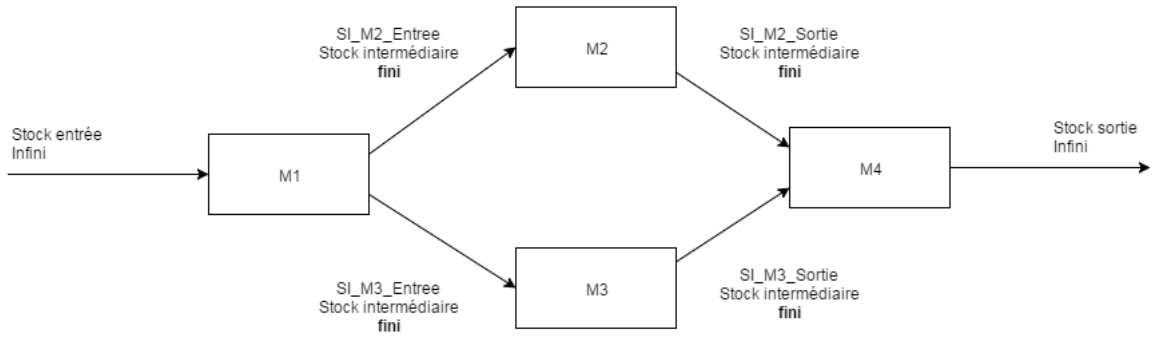


Figure 5 – *Cas de désassemblage suivi d'un assemblage*

Ce cas représente un Désassemblage suivi d'un Assemblage. Cette chaîne peut correspondre à un atelier qui recycle des ampoules : elles sont démontées en M1, puis le verre est nettoyé en M2 et le filament remplacé en M3, enfin, le tout est remonté en M4.

Il est aussi possible de voir ce cas comme un choix de gamme ou la gamme est déterminée par le traitement intermédiaire. Par exemple, un atelier qui prépare des ordinateurs, l'unité centrale est assemblée en M1, puis on a le choix entre deux types de carte graphique (M2 ou M3) puis on referme l'unité centrale en M4.

Pour ce qui est des processus, pour le désassemblage / assemblage, il n'y a toujours qu'un seul processus. Ce processus est en fait comme le nom du cas l'indique la fusion entre le processus d'assemblage et le processus de désassemblage. Attention à ne pas effectuer un simple copier/coller car il ne faut exécuter qu'une seule fois le traitement de M2 et de M3.

Dans le cas du choix de gamme, on a deux processus qui sont en fait l'extension du *cas avec deux machines et un stock intermédiaire* qui devient *cas avec trois machines*.

1.6 Plusieurs machines pour un traitement

Il est possible que plusieurs machines (ressources) soient allouées pour la même exécuter le même traitement. Dans ce cas, elles possèdent les mêmes caractéristiques. En particulier elles suivent la même loi de génération du temps de traitement.

D'un point de vue orienté processus, ce cas est en fait identique au cas élémentaire. La différence se percevra au niveau de l'algorithme du processus lors de la tentative d'acquisition d'une machine. Au lieu d'avoir une seule machine, on aura une certaine limite définie.

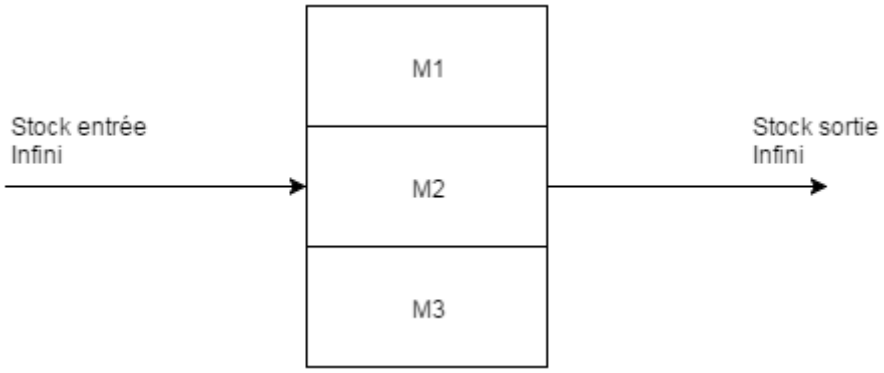


Figure 6 – *Cas de plusieurs machines disponibles pour un même traitement*

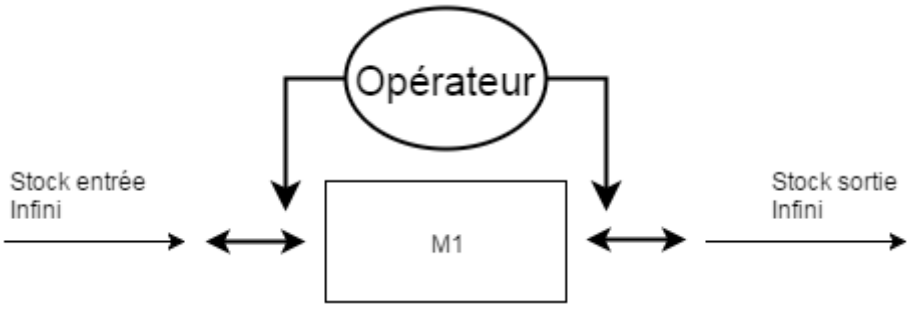


Figure 7 – *Cas de manipulation des stocks par un opérateur*

1.7 Traitement nécessitant un opérateur

Dans ce cas là, il n'y a qu'une seule machine mais il faut qu'un opérateur (humain ou robotisé) récupère un élément du stock d'entrée pour le placer dans la machine. Ensuite, après la fin du traitement de la machine, l'opérateur déplace le résultat vers le stock de sortie.

La création d'un produit peut être décrite par un seul processus. Le processus consiste en la réalisation des actions suivantes : Récupération d'un élément du stock d'entrée par l'Opérateur puis celui le place dans M1. M1 effectue son traitement, l'opérateur récupère le résultat et le déplace dans le stock de sortie.

C'est le cas le plus simple introduisant la notion d'Opérateur.

1.8 Traitements nécessitant le même opérateur

Dans cette situation, l'opérateur doit interagir avec les machines pour qu'elles puissent accomplir leur traitement. Par exemple, l'opérateur peut avoir à vérifier que l'élément à traiter est bien en place ou bien à sélectionner un mode de traitement spécifique pour la machine ou encore recalibrer la machine.

Ce cas nécessite la définition d'un processus par machine. Chacun de ces processus sera semblable à celui du *cas élémentaire* à la petite différence qu'avant de lancer le traitement de M1, il faudra attendre que l'opérateur soit disponible puis qu'il règle la machine. Si l'opérateur doit intervenir au début et la fin du traitement de chaque machine, les processus seront identiques au cas *traitement nécessitant un opérateur*.

2 Algorithmes des processus liés aux cas de base

Dans cette section, je vais exposer les algorithmes pour les deux premiers cas.

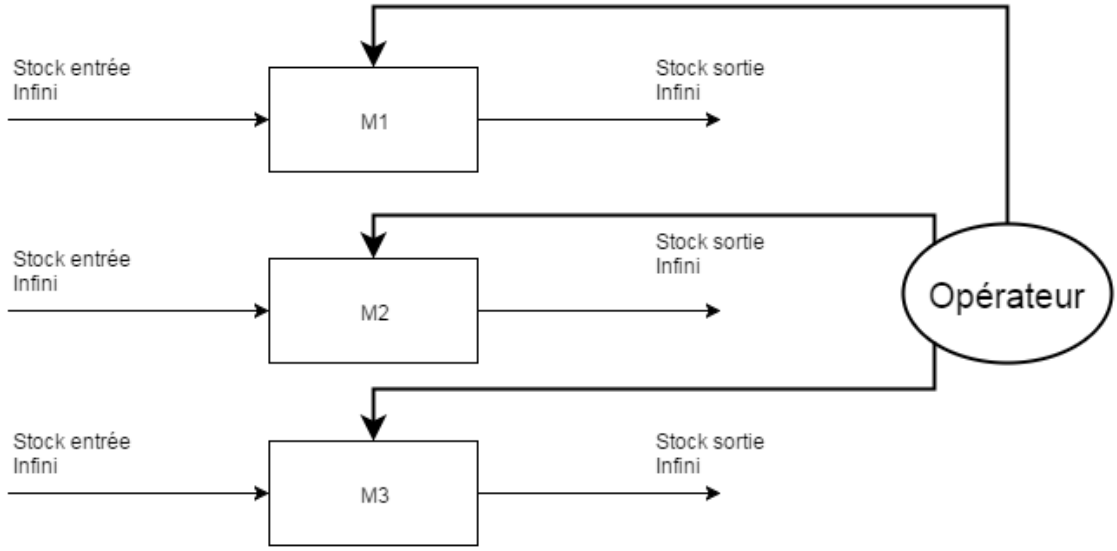


Figure 8 – *Cas de réglages de plusieurs machines par le même opérateur*

2.1 Le cas élémentaire

Data : T_M1 : le temps de traitement de M1, N : le nombre de produit fabriqués
if *M1 est occupée* **then**
  AttendreQue(M1 soit libre) ;
end
Occuper(M1) ;
Attendre(T_M1) ;
Libérer(M1);
$Q_{StockSortie} \leftarrow Q_{StockSortie} + 1$;

Algorithme 9 : Algorithme du cas élémentaire

C'est un algorithme très simple qui gère l'accès limité à une machine (ressource). Dans la simulation, lorsque l'algorithme d'un processus s'exécute, il continu jusqu'à ce qu'il rencontre une instruction qui le suspend (comme la fonction AttendreQue() ou bien qui l'arrête (la fin de l'algorithme). Comme un seul processus peut s'exécuter à la fois, si la condition *M1 est libre* est vrai alors, on exécutera forcément l'instruction *Occuper(M1)* qui mettra à jour la disponibilité de la machine. On assure ainsi que l'accès à la ressource est bien réglé.
Après la réalisation du traitement, il ne faut pas oublier de libérer la machine et de mettre à jour le stock de sortie.

2.2 Deux machines en série et un stock intermédiaire limité

Á cause de la capacité du stock intermédiaire qui est limitée, on remarque que cet algorithme ne correspond pas à deux algorithmes du cas de base placé l'un après l'autre. Par contre, une fois que M2 est accessible (libre), on se retrouve dans un état équivalent à celui de l'algorithme de base lorsque l'on a accédé à la machine.

2.3 Le désassemblage

Certaines étapes de cet algorithme nécessitent plus d'explication. Je vais commencer par décrire le principe de cet algorithme puis je vais donner plus de détails sur les nouvelles instructions.

Data : T_M1 : le temps de traitement de M1
T_M2 : le temps de traitement de M2
N : le nombre de produit fabriqués, Q2 : le nombre d'élément en attente dans la queue intermédiaire pour accéder à M2

```
1  if M1 est occupée then
2  |  AttendreQue(M1 soit libre) ;
3  end
4  Occuper(M1) ;
5  Attendre(T_M1) ;
6  if Q2 est pleine then
7  |  AttendreQue(Q2 ne soit plus pleine) ;
8  end
9  Libérer(M1) ;
10 if M2 est occupée then
11 |  Q2 ← Q2 + 1 ;
12 |  AttendreQue(M2 soit libre) ;
13 |  Q2 ← Q2 - 1 ;
14 end
15 Occuper(M2) ;
16 Attendre(T_M2) ;
17 Libérer(M2) ;
18 Q_StockSortie ← Q_StockSortie + 1 ;
```

Algorithme 10 : Algorithme du cas Deux machines et un stock intermédiaire limité

Data : test

```
1  if M1 est occupée then
2  |  AttendreQue(M1 soit libre) ;
3  end
4  Occuper(M1) ;
5  Attendre(T_M1) ;
6  M2_statut ← NON_DÉMARRÉ ;
7  M3_statut ← NON_DÉMARRÉ ;
8  M2_statut ← Exécuter(Bloc(M2), M2_statut) ;
9  M3_statut ← Exécuter(Bloc(M3), M3_statut ;
10 AttendreQue(M2_statut != NON_DÉMARRÉ et M3_statut != NON_DÉMARRÉ) ;
11 Libérer(M1) ;
12 AttendreQue(M2_statut == FINI et M3_statut == FINI ) ;
13 Q_StockSortie ← Q_StockSortie + 1 ;
```

Algorithme 11 : Algorithme du cas de désassemblage

Au début de l'atelier, il n'y a qu'une machine, les étapes 1 à 5 sont identiques à celles du cas de base. Une fois que la machine M1 a fini son traitement, elle ne peut pas être libérée tout de suite. Il faut que les résultats du désassemblage puissent accéder à leur queue respective (une pour M2 et une pour M3). Comme ces queues sont de taille limitée, il est possible que l'une d'entre elle ou même que les deux se retrouvent pleines. Et rien ne dit qu'elles se libèreront au même moment. Il devient alors nécessaire

d'exécuter ces tâches en parallèle.

2.3.1 Gestion des tâches parallèles

C'est ce que les lignes 8 et 9 de l'algorithme essayent de symboliser. Lorsque l'on appelle l'instruction **Exécuter**, on va créer un autre fil d'exécution. Ceci permet de ne pas bloquer l'exécution du processus en cours. Ainsi, on peut lancer un fil d'exécution par "branches" du désassemblage.

Maintenant, il faut surveiller le statut de ces fil d'exécution pour déterminer l'avancement de ces instructions. La machine M1 ne pourra être libérer qu'une fois que tous les résultats du désassemblage ont réussi à accéder à leur queue respective. Comme on ne peut pas anticiper ce moment ni l'ordre dans lesquelles ces accès vont s'effectuer, on délègue cette tâche aux fils d'exécution. On associe à chaque fil une variable pour surveiller son statut. Ainsi, lorsque dans le fil d'exécution, on arrive à accéder à la queue suivante, on met à jour le statut et le processus principal peut alors mettre à jour la disponibilité de la machine. La variable de statut doit être commune au processus créateur (le processus père) et au processus crée (le processus fils).

Quelles valeurs peuvent prendre ces variables de statut ?
Á ce stade de la conception, j'en discerne au moins 4 :

- NON_DÉMARRÉ : l'accès à la queue pour le prochain traitement n'est pas encore réalisé. La machine initiatrice de ce fil d'exécution est donc toujours **occupée** par au moins cet élément.
- DANS_LA_QUEUE : l'élément à traiter est en attente dans la queue du prochain traitement. La machine est libérée de cet élément.
- EN_TRAITEMENT : Le traitement de l'élément est en cours.
- FINI : L'élément a fini de subir son traitement et il est disponible dans la queue suivant le traitement.

Lorsqu'il n'y a qu'un seul traitement à réaliser, la description de chaque état semble satisfaisante mais il est possible que cela ne soit pas le cas.

2.3.2 Notions de blocs

C'est pour prévoir ces cas que j'ai parlé de **Bloc** dans les paramètres de l'instruction Exéctuer. Voici une représentation d'un atelier plus compliqué pour mettre en avant cette notion de Bloc.

Dans cet atelier, on retrouve au début notre situation de désassemblage (M1 -> M2 + M3). Il va donc y avoir une création de Threads (fils d'exécution). Ces créations sont schématisées par des flèches pleines. En tout, 4 Threads seront crées en plus du Thread "initial".

Les blocs associés à ces Threads seront les suivants :

- Le bloc A [M3, M6]
- Le bloc B [M2, M4, M5, M7]
- Mais aussi le bloc C [M4]
- et le bloc D [M5]

Un bloc peut donc lui-même contenir des sous blocs. Une question se pose : Comment déterminer la zone d'influence des blocs ?

Sur les différents cas que j'ai étudié, j'ai pu constater que des **blocs sont nécessaires** lors d'un "**désassemblage**" dans l'atelier. Il va y avoir **un bloc par branches** résultant de ce désassemblage.(par exemple lors du traitement M8, deux branches se créent donc deux blocs seront nécessaires lors du désassemblage).

Un bloc se termine lorsqu'il procède à un assemblage avec un élément d'un autre bloc.(par exemple, lorsque M4 et M5 respectivement le bloc C et le bloc D se rejoignent, on sort de la zone d'influence des blocs.

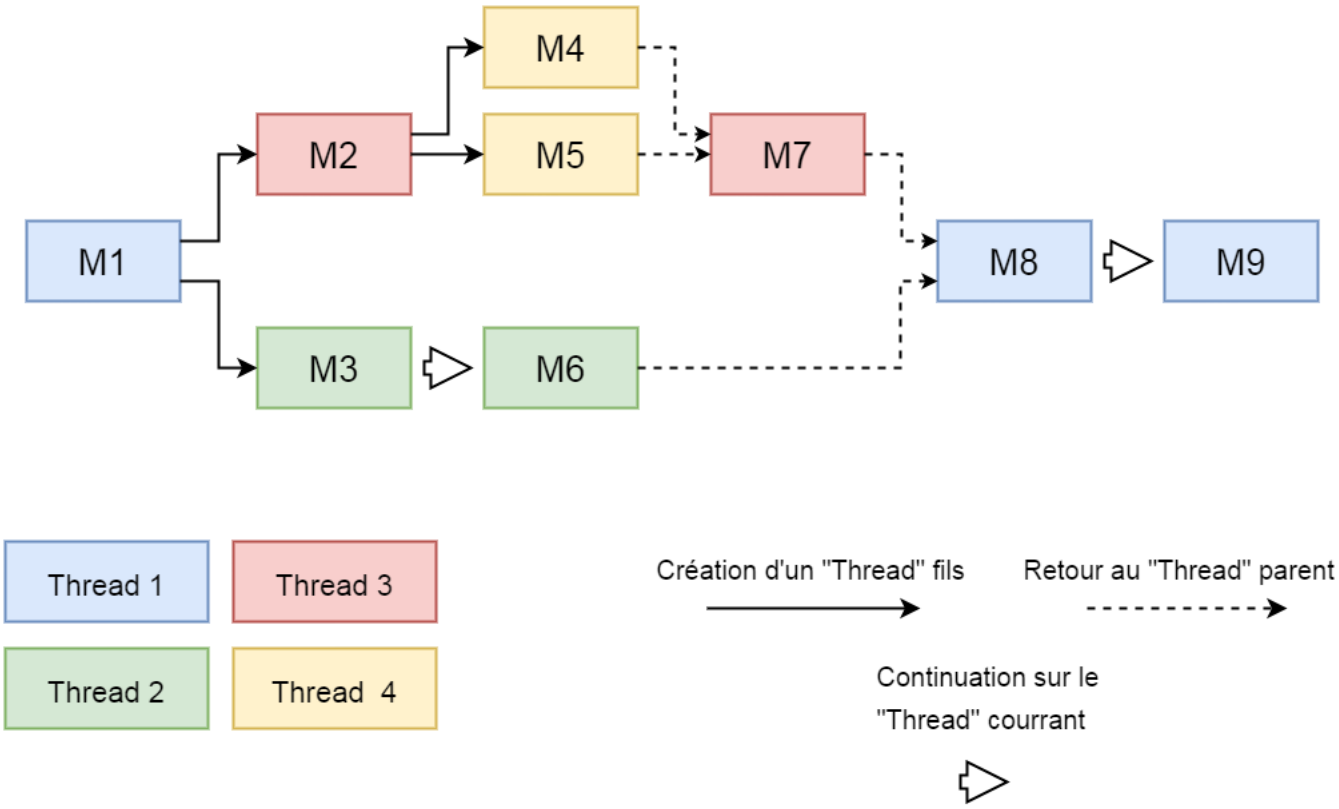


Figure 9 – *Exemple d'atelier plus complexe avec ses différents "Thread" (fil d'exécution)*

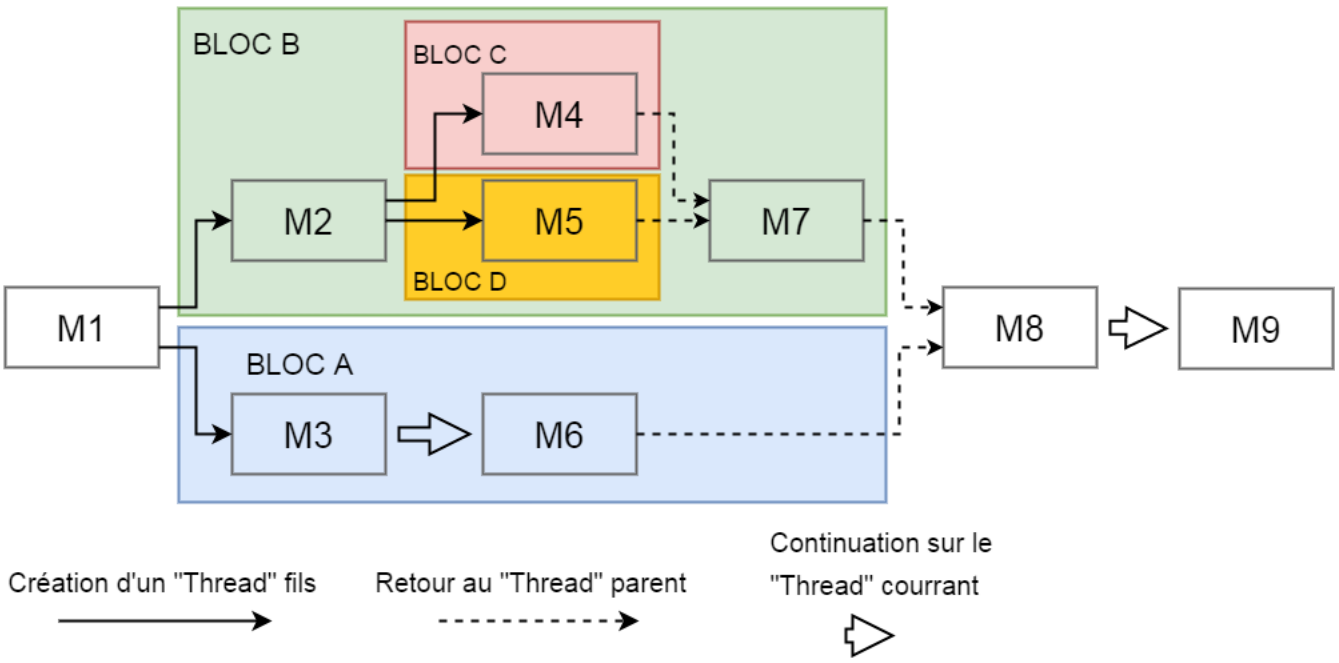


Figure 10 – *Découpage en "blocs" de l'atelier*

Les variables de statuts présentées plus haut manquent désormais de précision. Le Thread initial ne peut pas savoir précisément où en sont chacun de ses blocs. Par contre, il sait lorsqu'il peut libérer la ressource sur laquelle il travaillait (les statuts de tout ses blocs sont passés au stade DANS_LA_QUEUE), et il sait lorsqu'il doit reprendre la suite des traitements (statut FINI pour tous les blocs crées). Pour débuter, cela semble répondre aux besoins essentiels.

Tout ces points semblent être également valide pour les sous blocs.

2.3.3 Algorithme d'un bloc

Un bloc correspond à une série d'instructions du même ordre que celles déjà vues pour le cas de base. Je vais reprendre l'exemple du désassemblage "simple" pour continuer.(Les algorithmes pour décrire le cas "compliqué" seront explicités dans une future section)

L'algorithme proposé plus haut nécessite l'exécution du Bloc(M2). Ce bloc contient toutes les instructions pour récupérer une partie du désassemblage, l'insérer dans la queue d'attente de M2 et le placer dans la queue du stock de sortie. Sans oublier de mettre à jour sa variable de statut.

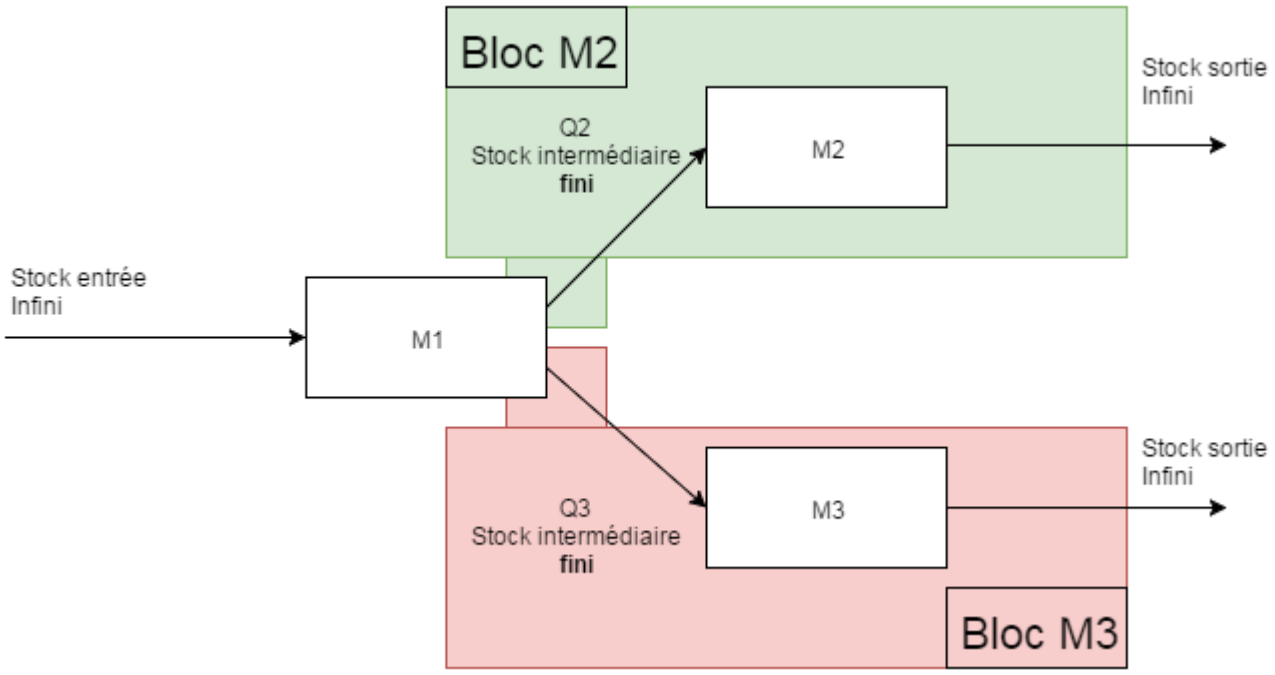


Figure 11 – *Découpage en "blocs" du désassemblage*

Data : Q2 : Nombre d'élément en attente dans la queue Q2 pour accéder à M2
T_M2 : temps de traitement de la machine M2
$M2_{statut}$: la variable de statut du bloc M2

```
1  M2_statut ← NON_DÉMARRÉ
2  if Q2 est pleine then
3  |  AttendreQue(Q2 accessible) ;
4  end
5  M2_statut ← DANS_LA_QUEUE ;
6  if M2 est occupée then
7  |  Q2 ← Q2 + 1 ;
8  |  AttendreQue(M2 libre) ;
9  |  Q2 ← Q2 - 1 ;
10 end
11 Occuper(M2) ;
12 M2_statut ← EN_TRAITEMENT ;
13 Attendre(T_M2) ;
14 Libérer(M2) ;
15 Q_StockSortie ← Q_StockSortie + 1 ;
16 M2_statut ← FINI ;
```

Algorithme 12 : Algorithme du bloc M2 pour le désassemblage

Á la fin cet algorithme, la variable de statut est mise à la valeur *FINI* et le bloc d'instruction termine son exécution. Le bloc C de l'exemple est identique au bloc B aux différences que la machine considérée est M3, son temps de traitement est T_M3 et sa variable de statut $M3_{statut}$. Suivant les ateliers, ce ne sera pas toujours le cas.

2.4 L'assemblage

Data : $M1_{statut}$: la variable de statut du bloc M1
$M2_{statut}$: la variable de statut du bloc M2
Q_{M1M3} : Nombre d'élément en attente dans la queue Q_{M1M3} pour accéder à M3
Q_{M2M3} : Nombre d'élément en attente dans la queue Q_{M2M3} pour accéder à M3

```
1  M1_statut ← NON_DÉMARRÉ ;
2  M2_statut ← NON_DÉMARRÉ ;
3  M1_statut ← Exécuter(Bloc(M1), M1_statut) ;
4  M2_statut ← Exécuter(Bloc(M2), M2_statut ;
5  AttendreQue(M1_statut == FINI et M2_statut == FINI ) ;
6  if M3 est occupée then
7  |   AttendreQue(M3 soit libre) ;
8  |   Q_M1M3 ← Q_M1M3 − 1 ;
9  |   Q_M2M3 ← Q_M2M3 − 1 ;
10 end
11 Occuper(M3) ;
12 Attendre(T_M3) ;
13 Libérer(M3) ;
14 Q_StockSortie ← Q_StockSortie + 1 ;
```

Algorithme 13 : Algorithme du cas de l'assemblage

Cet algorithme ressemble à une version inversée du cas de désassemblage.
Mais attention, il y a un point très important, lignes 8 et 9 il faut faire la mise à jour du nombre d'élément dans les queues nécessaires au lancement du traitement de M3.

Pour ce qui est des algorithmes des blocs, il y a une différence également car les queues d'attente de M3 sont à capacité limitée. Il faut donc gérer cela dans leur algorithme.

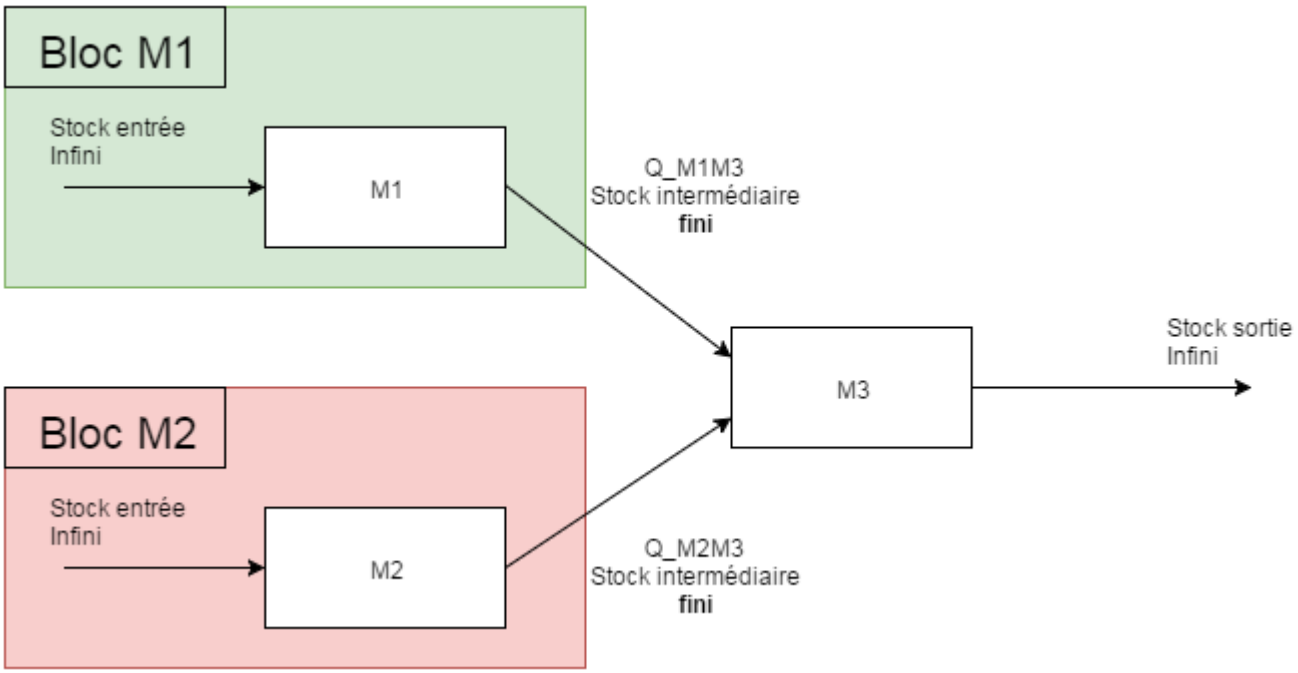


Figure 12 – *Découpage en "blocs" de l'assemblage*

Data : T_M1 : temps de traitement de la machine M1
$M1_{statut}$: la variable de statut du bloc M1
Q_{M1M3} : Nombre d'élément en attente dans la queue Q_{M1M3} pour accéder à M3

```
1  M1_statut ← NON_DÉMARRÉ
2  M2_statut ← DANS_LA_QUEUE ;
3  if M1 est occupée then
4  |  AttendreQue(M1 libre) ;
5  end
6  Occuper(M1) ;
7  M1_statut ← EN_TRAITEMENT ;
8  Attendre(T_M1) ;
9  if Q_M1M3 est pleine then
10 |  AttendreQue(Q_M1M3 accessible) ;
11 end
12 Libérer(M1) ;
13 Q_M1M3 ← Q_M1M3 + 1 ;
14 M1_statut ← FINI ;
```

Algorithme 14 : Algorithme du bloc M1 pour l'assemblage

Le point critique de cet algorithme est la ligne 13 qui met à jour le nombre d'élément en attente dans la queue et qui assure que lorsque la variable de statut indique *FINI* alors on a au moins cet élément dans la queue.

Le bloc M2 est identique en terme de structure à celui de M1.

2.5 Le désassemblage/assemblage

Data : $M2_{statut}$: la variable de statut du bloc M2
$M3_{statut}$: la variable de statut du bloc M3
Q_{M2M4} : Nombre d'élément en attente dans la queue Q_{M2M4} pour accéder à M4
Q_{M3M4} : Nombre d'élément en attente dans la queue Q_{M3M4} pour accéder à M4
T_M1 : temps de traitement de la machine M1
T_M4 : temps de traitement de la machine M4

1 **if** *M1 est occupée* **then**
2 | AttendreQue(M1 soit libre) ;
3 **end**
4 Occuper(M1) ;
5 Attendre(T_M1) ;
6 $M2_{statut} \leftarrow$ NON_DÉMARRÉ ;
7 $M3_{statut} \leftarrow$ NON_DÉMARRÉ ;
8 $M2_{statut} \leftarrow$ Exécuter(Bloc(M2), $M2_{statut}$) ;
9 $M3_{statut} \leftarrow$ Exécuter(Bloc(M3), $M3_{statut}$;
10 AttendreQue($M2_{statut}$!= NON_DÉMARRÉ et $M3_{statut}$!= NON_DÉMARRÉ) ;
11 Libérer(M1) ;
12 AttendreQue($M2_{statut}$ == FINI et $M3_{statut}$ == FINI) ;
13 **if** *M4 est occupée* **then**
14 | AttendreQue(M4 soit libre) ;
15 | $Q_{M2M4} \leftarrow Q_{M2M4} - 1$;
16 | $Q_{M3M4} \leftarrow Q_{M3M4} - 1$;
17 **end**
18 Occuper(M4) ;
19 Attendre(T_M4) ;
20 Libérer(M4) ;
21 $Q_{StockSortie} \leftarrow Q_{StockSortie} + 1$;

Algorithme 15 : Algorithme du cas de désassemblage/assemblage

Dans cet algorithme, on peut observer la présence de suites d'instructions déjà rencontrées (au nom des machines près). Les lignes 1 à 12 sont identiques à celle de l'algorithme de désassemblage qui est la première étape de cet exemple. Les lignes 13 à 20 correspondent à la fin de l'algorithme d'assemblage (ligne 6 à 13).

Dans cette correspondance, on peut discerner un début de logique de base pour construire des ateliers. Avec une association des différents cas de base, on pourrait bien concevoir plus simplement les algorithmes des ateliers de fabrications.

Observons maintenant la constitution de l'algorithme du bloc M3 :

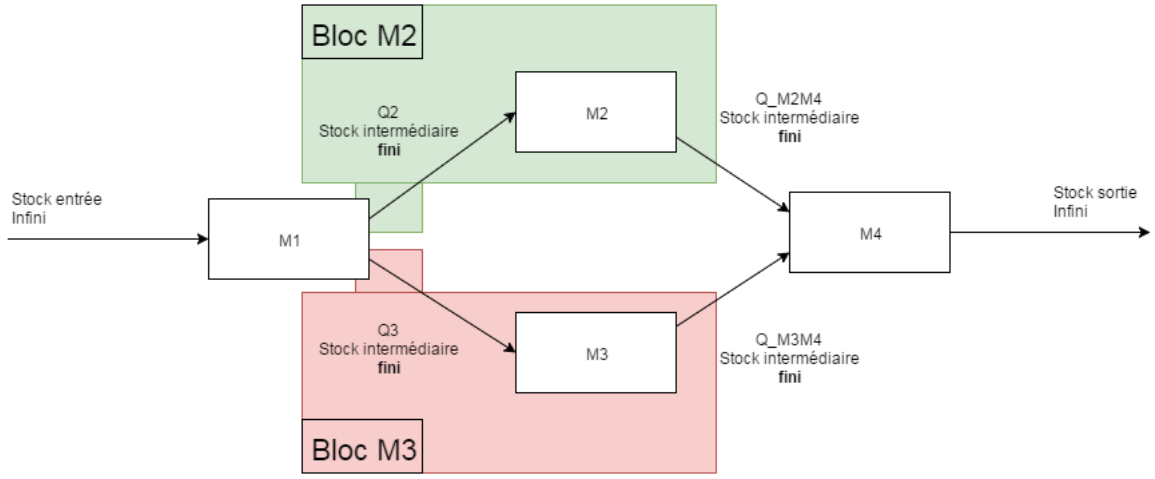


Figure 13 – *Découpage en "blocs" du désassemblage/assemblage*

Data : Q3 : Nombre d'élément en attente dans la queue Q3 pour accéder à M3
T_M3 : temps de traitement de la machine M3
$M3_{statut}$: la variable de statut du bloc M3
Q_{M3M4} : Nombre d'élément en attente dans la queue Q_{M3M4} pour accéder à M4

```
1  M3_statut ← NON_DÉMARRÉ
2  if Q3 est pleine then
3  |  AttendreQue(Q3 accessible) ;
4  end
5  M3_statut ← DANS_LA_QUEUE ;
6  if M3 est occupée then
7  |  Q3 ← Q3 + 1 ;
8  |  AttendreQue(M3 libre) ;
9  |  Q3 ← Q3 - 1 ;
10 end
11 Occuper(M3) ;
12 M3_statut ← EN_TRAITEMENT ;
13 Attendre(T_M3) ;
14 if Q_M3M4 est pleine then
15 |  AttendreQue(Q_M3M4 accessible) ;
16 end
17 Libérer(M3) ;
18 Q_M3M4 ← Q_M3M4 + 1 ;
19 M3_statut ← FINI ;
```

Algorithme 16 : Algorithme du bloc M3 pour le désassemblage/assemblage

Dans ce bloc, on constate le même effet que pour l'algorithme : les lignes 1 à 13 sont identiques à celles du bloc M2 pour le désassemblage 2.3.3 et les lignes 14 à 19 correspondent aux dernières lignes (9 à 14) du bloc M1 pour l'assemblage 14.

5 La bibliothèque SimPy

SimPy (à ne pas confondre avec SymPy) est une bibliothèque Python qui offre des fonctionnalités pour réaliser des simulations.

1 L'*Environment*

L'objet **Environment** de SimPy s'occupe de gérer l'horloge de la simulation ainsi que l'ordonnancement des processus. Il est possible d'accéder à l'heure de la simulation à tout moment ainsi qu'au contenu de l'ordonnanceur.

```
from simpy import Environment

env = Environment()

# Accès à l'heure de la simulation
env.now(initial_time=0)

# Le processus en cours d'exécution
env.active_process

# Le prochain processus programmé
env.peek()
```

Ici la simulation démarrera à l'instant t = 0, mais on peut passer en paramètre l'heure à laquelle on souhaite commencer.

Cet environnement est très complet et permet de s'abstraire de la gestion précise de l'ordonnanceur.

Pour démarrer une simulation, il suffit d'appeler l'instruction suivante :

```
# Simuler pendant 100 pas
env.run(until=100)

# Simuler jusqu'à l'épuisement de la queue de processus
env.run()
```

On peut choisir d'arrêter la simulation à une date précise ou alors jusqu'à l'épuisement de tous les processus à exécuter.

La bibliothèque propose également de la simulation temps réel avec la classe **RealTimeEnvironment**.

Pour soumettre des événements ou des processus, on utilise aussi l'objet *Environment*.

```
env.schedule(event, priority=1, delay=0)
```

Avec cette instruction, on programme l'événement **event** avec une priorité égale à 1 (plus la priorité est basse, plus l'événement est important) et l'événement s'exécutera dans *delay* pas.

```
env.process(generator)
```

Cette instruction programme le générateur dans la file d'exécution de la simulation. Cette notion de générateur est développée dans la Section 2.

2 Generator

Les Generators (Générateur en français) représentent un patron de conception non spécifique à SimPy mais qui revient beaucoup dans cette bibliothèque. Ce patron consiste à déclarer des fonctions qui pourront se comporter comme des *iterateurs* [WWW9]. Voici un exemple pour illustrer ce concept :

```
# Definition du générateur
def my_generator():
  number = 0
  while(true):
    number = number + 1
    yield number

###########

my_first_number = next(my_generator())
# vaut 1

my_second_number = next(my_generator())
# vaut 2
```

En fait, une fois appelé, un générateur s'exécute jusqu'à l'apparition d'une instruction **yield**. Ce yield agit comme un **return** car il renvoie la valeur précisée juste après (ici *number*). Au prochaine appel de la fonction, l'exécution reprend à l'instruction suivant ce yield et continue jusqu'à être de nouveau interrompu ou bien arriver à la fin de ses instructions.

Cette explication permet d'introduire la notion de **Process** dans SimPy.

3 Process

Un **Process** dans SimPy peut être vu comme une suite d'instructions pouvant être interrompu à certains moments pour attendre la réalisation (ou non) d'une ou plusieurs conditions. Par exemple, on peut attendre une certaine heure de la simulation ou bien l'apparition d'un événement.
Cette description correspond avec la définition des générateurs. Ici, SimPy s'occupe de rappeler les Process mis en attente sur un *yield*. Plus précisément, l'instruction yield doit être suivie d'un **Event** SimPy. Le processus est mis en pause et reprendra son exécution lorsque l'événement se sera réalisé. L'utilisation de yield permet de rajouter le process dans la liste des **callbacks** de l'événement. Un **callback** est une fonction ou un traitement qui doit être exécuté à la fin d'une fonction.

Voici un exemple de Process basique :

```python
# Définition du processus
def my_process():
  print("Hi, I'm a little process.")

  time = 0
  for i in range(10):
    time = time + 1

  print("I'm tired now, I'm going to rest a little")

  yield env.timeout(10)

  print("Waking up. Finishing the job")
  for j in range (10):
    time = time - 1

############

# Création de l'environnement
env = Environment()
# Planification du processus
env.process(my_process)

# Lancement de la simulation
env.run()
```

Á noter que les instructions du processus ne s'exécutent pas avant d'avoir lancé la simulation. Lors de la planification du processus dans la simulation, le processus est soumis à la file d'exécution. Étant le seul et premier processus de la simulation, il s'exécutera à l'instant t = 0. Il sera interrompu lors de l'instruction

```python
yield env.timeout(10)
```

La présence du yield précise que le processus sera interrompu. La condition pour reprendre le processus est d'attendre la réalisation de l'événement *env.timeout(10)* qui consiste simplement à attendre 10 pas de simulation. Au bout de 10 pas, le processus pourra reprendre son exécution et la finir.

4 Event

La dernière notion de SimPy à introduire est la notion d'événement (déjà aperçue plus haut). C'est ici que la séparation entre simulation orientée processus et orientée événement se floute. En fait, un processus peut faire appel à des événements pour réaliser son exécution. Il semble tout à fait raisonnable de penser qu'un processus de création pour une gamme d'un atelier peut être régulé par l'arrivée de certains événements. L'exemple le plus simple est lorsque l'on utilise une des ressources d'un atelier, l'événement "l'utilisation de la ressource est terminée" se produira de manière assez évidente (à moins d'une situation spéciale de type "arrêt de la simulation avant").
Ce qu'il faut retenir, c'est que conceptuellement parlant, les simulations sont orientées processus mais qu'au niveau de l'implémentation on peut utiliser des objets Event pour améliorer la lisibilité et les performances de l'application. De même, un processus peut se baser sur les événements dans ses instructions. L'important est de toujours considérer une gamme comme un processus (une suite d'instruction) et non comme un enchaînement plus ou moins lié d'événements.

Un Event est un objet qui peut **succeed** (**réussir**). On peut choisir le moment où cet événement devient réalisé de cette façon :

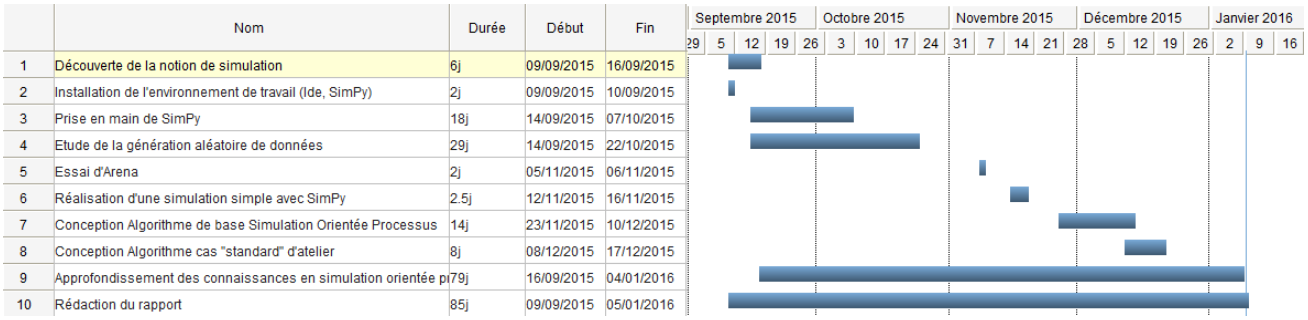
```python
# Création de l'environnement
env = Environment()
# Création d'un événement
```

```
4 my_event = env.event()
5
6 # On "réalise" l'événement
7 my_event.succeed()
```

De plus, les **Process** de SimPy sont aussi considérés comme des événements, ce qui permet d'appeler l'instruction *yield* avec un Process. Le Process en cours attendra la fin de l'exécution du Process "yield" avant de reprendre son exécution.

6 Planification des étapes de la recherche

Pour représenter les différentes phases de recherches par lesquelles je suis passé, j'ai choisi de construire un diagramme de Gantt.



	Nom	Durée	Début	Fin
1	Découverte de la notion de simulation	6j	09/09/2015	16/09/2015
2	Installation de l'environnement de travail (Ide, SimPy)	2j	09/09/2015	10/09/2015
3	Prise en main de SimPy	18j	14/09/2015	07/10/2015
4	Etude de la génération aléatoire de données	29j	14/09/2015	22/10/2015
5	Essai d'Arena	2j	05/11/2015	06/11/2015
6	Réalisation d'une simulation simple avec SimPy	2.5j	12/11/2015	16/11/2015
7	Conception Algorithme de base Simulation Orientée Processus	14j	23/11/2015	10/12/2015
8	Conception Algorithme cas "standard" d'atelier	8j	08/12/2015	17/12/2015
9	Approfondissement des connaissances en simulation orientée pr	79j	16/09/2015	04/01/2016
10	Rédaction du rapport	85j	09/09/2015	05/01/2016

Figure 1 – *Diagramme de Gantt des phases de recherches du projet*

Pour commenter ce diagramme, il y a eu deux phases "majeures" : une phase de découverte des notions de simulations et de prise en mains des outils et une phase de réflexion sur les algorithmes qui seront implémentés dans la phase de développement.

Ces deux phases évoluant en parallèle de deux fils "conducteurs", d'une part la rédaction du rapport et d'autre part la constante familiarisation avec les principes de la simulation.

Au début de ce projet, j'avais anticipé les deux fils conducteurs mais je m'étais attendu à plus de difficultés pour la prise en main (première phase) et par contre à moins d'écueils pour la seconde phase. Notamment, je me suis éloigné d'une solution optimale avec le maximum de souplesse pour les ateliers à simuler afin de pouvoir valider l'idée principale du projet et de pouvoir en prouver sa faisabilité lors de la phase de développement.

Conclusion de la partie Recherche

Il y a vraiment beaucoup d'aspects différents à prendre en compte pour réaliser des simulations même lorsque l'on limite le cadre de ces simulations aux ateliers de fabrications. Une des difficultés de ce projet est qu'il n'est pas envisageable d'apporter une solution à tous les problèmes dans le temps imparti par le projet. Il faut donc sélectionner les concepts importants à valider, c'est-à-dire la création d'algorithmes pour les "briques" élémentaires d'un atelier de fabrication et vérifier leur souplesse et leur facilité d'utilisation (de combinaison). Mais même à ce niveau ci, il m'a fallu poser des contraintes sur les possibilités offertes par les ateliers de fabrication (ex : l'impossibilité de créer un processus démarrant au milieu de la chaîne de l'atelier).

J'ai procédé à cette simplification certainement un peu tard dans le projet, ce qui entraîne un déséquilibre dans l'écriture du rapport qui laisse beaucoup de place aux explications sur les simulations (explications de plus non exhaustives au vu de la complexité du sujet) et moins sur les algorithmes qui ont été revus de façon simplifiée pour ne pas bloquer la partie développement du projet.

Deuxième partie

Partie Développement

7 Développement des solutions présentées

La suite de ce rapport détaille le déroulement de la **partie développement** du projet. Le principe est de partir des propositions de la partie *Recherche* pour proposer une implémentation. Pour ce projet, l'implémentation est en langage python.

L'objectif de cette partie est d'avoir à la fin du développement un programme python pouvant s'exécuter. Ce programme doit permettre de réaliser une simulation d'un atelier de fabrication contenant des aspects "intéressants".

1 Création des processus de base

La première étapde du développement est l'implémentation des cas de bases. Ensuite, il faudra déterminer si ces cas de base peuvent se combiner pour constituer ensemble un nombre intéressant d'ateliers différents.

Les cas très simples et linéaires ne posèrent pas de problème. Cependant, à partir des cas présentant des embranchements (exemple : assemblage, désassemblage), il devenait difficile de garder à la fois un code simple, facile à écrire et une bonne souplesse du programme (assurer une certaine généricité du code).

2 Révision de l'orientation de l'implémentation

Á ce point du projet, j'ai dû faire face à un choix peu évident sur le moment : soit je continuais sur cette lancée car finalement, le comportement du code correspondait à ce qui était recherché. La théorie de la combinaison des *briques de bases* se confirmait, mais la compréhension du code devenait assez ardue et la vision à long terme du projet était floue. Soit je repartais sur une autre idée avec donc une reprise de certains points de la phase de recherche.
Le résultat de ce choix fût déterminé lors d'une réunion avec mon encadrant. Nous avons isolé le problème majeur qui est la dépendance de certaines étapes d'un processus. En effet, certaines tâches doivent s'exécuter en parallèles à certains points puis doivent se rejoindre dans une future partie du processus. La difficulté cachée ici est que, lorsque des tâches s'exécutent en parallèles, elles sont indépendantes l'une de l'autre mais il faut quand même pouvoir déterminer le moment où cette parallélisation se termine afin de pouvoir poursuivre le processus (il faut que les deux ou plus branches parallèles soient finies). Quelqu'un doit alors *surveiller* l'avancement de cette parallélisation. L'idée de base consistait à créer une tâche lorsque toutes les conditions sont réunies pour son exécution, d'où la nécessité de ce *surveillant*. La complexification du code résultait du fait que ce surveillant devait être capable

de gérer un nombre de branches potentiellement différent pour chaque embranchement de chaque atelier.

2.1 Utilisation de graphes de précédences

Pendant la réunion, la solution émergente fût celle de l'utilisation d'un **graphe de précédence**. C'est un graphe orienté sans cycle qui va nous intéresser. Un graphe représente une gamme de l'atelier (depuis un atelier on peut donc créer plusieurs graphes). Ce graphe a pour nœuds les différents éléments de l'atelier nécessaires à la réalisation du produit de la gamme (machines, files d'attente, ...). Les arcs représentent le chemin entre ces machines.
Voici un exemple de représentation d'atelier à l'aide d'un graphe de précédence :

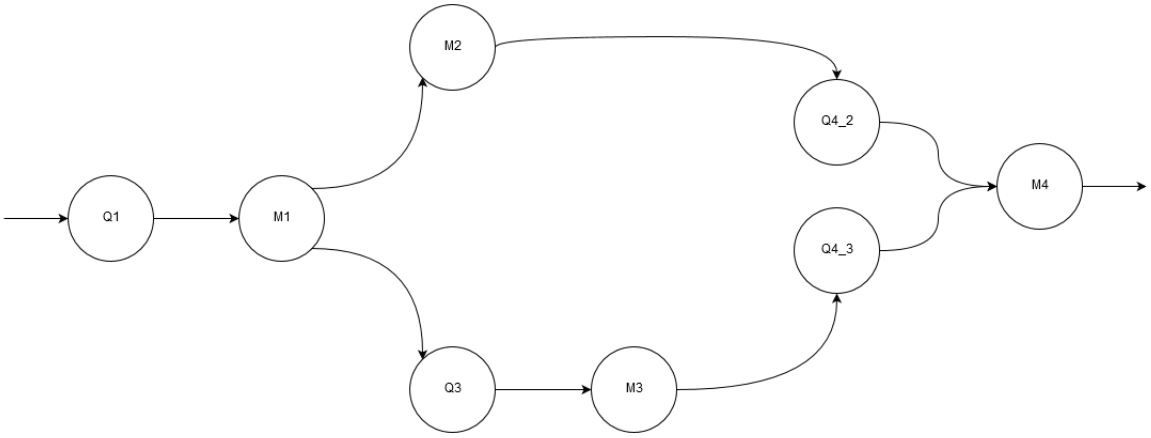


Figure 1 – *Exemple de graphe de précédence représentant un atelier*

Ce graphe représente une gamme d'un atelier possédant 4 machines "lamdba" et 4 files d'attentes. Le processus de cette gamme consiste donc à réaliser les tâches suivantes dans cette ordre :

Arriver dans l'atelier
Accéder à Q1 [1]
Accéder à M1
Se faire traiter par M1

Partie en parallèle

Accéder à M2	Accéder à Q3
Se faire traiter par M2	Accéder à M3
Accéder à Q4_2	Se faire traiter par M3
.	Accéder à Q4_3

Accéder à M4
Se faire traiter par M4
Libérer l'atelier

Dans le premier tableau, le processus s'exécute de façon séquentielle. Pas de difficulté à ce niveau là.

Dans le deuxième tableau, il faut penser que chacune des colonnes s'exécute indépendamment de l'autre mais toujours en parallèle. Á noter que la machine M1 ne sera libérée que lorsque le processus aura accédé à M2 **et** à Q3.

Pour accéder au dernier tableau, les deux branches parallèles doivent se rejoindre, c'est-à-dire que chacune d'elle doit réaliser toutes les tâches qu'elle doit faire. C'est seulement avec cette condition que le processus pourra reprendre son cours. Il repasse dans une période d'exécution *séquentielle*.

Je vais décrire en quoi cette représentation permet de répondre aux problèmes précédemment énoncés.

Tout d'abord, cette représentation est assez intuitive. Il suffit d'avoir un schéma de l'atelier pour pouvoir le reproduire aisément sous forme de graphe.

Un des problèmes majeurs était de gérer la parallélisation en créant les tâches dynamiquement dès qu'elles pouvaient être exécutées. Au détriment de plus d'utilisation de la mémoire, il s'avère que créer toutes les tâches à la création du processus permet de gérer ce problème. Chaque tâche à besoin de savoir quand elle peut s'exécuter. Cela se traduit par des conditions (des pré-conditions même) qui doivent être remplies pour que la tâche débute. Dans le cadre du projet, pour qu'une tâche puisse s'exécuter, il faut que tous ses prédécesseurs soient terminés. Ce qui fait que la première tâche ne possédant pas de prédécesseur peut immédiatement commencer à s'exécuter.
Pour reprendre l'exemple ci-dessus :
On va créer une tâche par noeud de notre graphe. La tâche Q1 est la première qui s'exécutera puisqu'elle ne possède aucun prédécesseur (sur la représentation graphique du graphe, cela signifie qu'il n'y a pas de flèche partant d'un noeud qui arrive sur Q1).
La tâche M1 elle ne peut s'exécuter qu'après la tâche Q1. Et en effet, Q1 est un prédécesseur (un parent) de M1.
Pour la tâche M4, il faut que ses deux parents (Q4_2 et Q4_3) aient accompli leur tâches (ce qui se traduit par la présence d'un élément dans chacune des deux files d'attente). Et récursivement, pour que Q4_2 et Q4_3 s'exécutent, il faut que leur parent à eux aussi soient terminés, ainsi de suite jusqu'à remonter à la racine du graphe. Il peut y avoir plusieurs noeuds *sans parents* dans l'atelier.

En ce qui concerne le lancement des tâches, le problème est corrigé. Maintenant, il faut déterminer comment déterminer le moment pour libérer les ressources. Nous avons vu que, toujours sur le graphe d'exemple, pour libérer la machine M1, il faut que M2 et Q3, ses **fils** aient commencé leur tâche. M2 sait lorsqu'il commence mais il ne connaît pas le statut de Q3 et inversement. Pour limiter les communications et les liens entre les tâches, c'est le processus parent qui va gérer la libération des ressources en se basant sur le statut de ses fils. M1 connaît son nombre de fils (ici 2). Il sait donc que pour se libérer il doit recevoir 2 demandes de libération de ressource. Lorsque ce nombre est atteint, il peut libérer la ressource. Pour l'expliquer autrement, le parent entretient un compteur sur le nombre de demande de libération de la ressource qui doit atteindre exactement son nombre de fils. Lorsque l'un des fils estime que la tâche peut commencer (donc qu'il prend en charge un partie des éléments bloquant son père), il effectue une demande de libération à son (ou ses) parent(s). Cette demande est prise en compte par le parent et incrémente le fameux compteur. Si celui-ci atteint son objectif alors la ressource est libérée, sinon, il reste au moins un fils qui n'a pas pu prendre en charge sa partie et la ressource est maintenue.

Avec ces concepts, il est désormais faisable d'envisager la reprise du développement du projet. Mais avant cela, il reste à savoir si ces concepts sont utilisables en langage Python et avec la bibliothèque Simpy.

2.2 Faisabilité de l'implémentation des nouveaux concepts

2.2.1 Les graphes de précédences

La première chose est de déterminer comment implémenter un graphe de précédence en Python. Sur ce sujet là, les graphes étant assez commun en informatique et le langage Python étant assez répandu, il fût simple de trouver une bibliothèque proposant des outils de création et de parcours de graphes.

La bibliothèque retenue est **networkx** [WWW8]. Les raisons de ce choix sont assez simples : la bibliothèque offre bien la gestion de graphe orienté, la documentation est fournie, les fonctions sont intuitives à l'utilisation et pertinentes, une couverture de code et des tests mis en avant sur le site (sérieux et justesse du développement).

Exemple de création d'un graphe orienté :

```
1  from networkx import DiGraph
2
3  graph = DiGraph() # Création du graphe orienté (DiGraph = Directed Graph)
4
5  # Ajout des noeuds
6  graph.add_node(1, attributNom = "PremierNode")
7  graph.add_node(2, attributNom = "DeuxièmeNode", attributB = 10)
8  graph.add_node(3, attributNom = "TroisièmeNode", attributC = MyObject())
9  graph.add_node(4, attributNom = "QuatrièmeNode", attributB = 42)
10 graph.add_node(5, attributNom = "CinquièmeNode", attributB = 0, attributC = "Une chaîne")
11
12 # Création des arcs
13 graph.add_edge(1,2)
14 graph.add_edge(2,3)
15 graph.add_edge(2,4)
16 graph.add_edge(3,5)
17 graph.add_edge(4,5)
```

Ce code représente le graphe suivant :

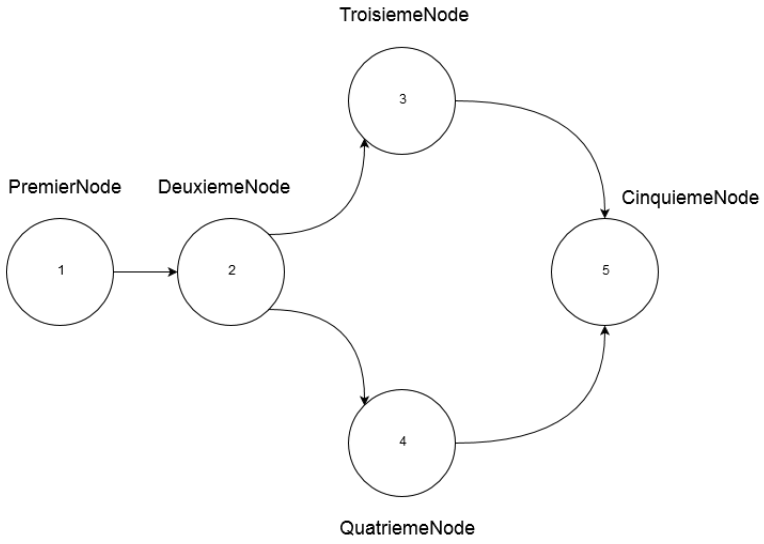


Figure 2 – *Graphe représenté par le code python*

L'outil proposé par cette bibliothèque est donc très simple d'utilisation et très souple. On peut ajouter des informations hétérogènes sur chacun des nœuds, ce qui permet de donner des caractéristiques spéciales pour certains noeuds (donc éléments de l'atelier). En effet, on peut imaginer qu'un four peut être réglé sur une certaine température alors qu'une scie circulaire nécessitera plutôt un nombre de tours par minute.
Si on pense en orienté objet, ces attributs pourraient en fait être directement des attributs de la machine en question plutôt que de les ajouter dans le graphe. Mais si cette idée est valable pour un atelier avec une seule gamme, elle ne fonctionne plus lorsqu'il y a plusieurs gammes utilisant les même machines. Ainsi, si un four peut chauffer du verre et du métal, sa température dépendra de la gamme (du matériau à fondre).
On pourrait alors placer cette information sur le processus. C'est une option intéressante. Ajouter des informations relatives au processus dans le processus est cohérent. Mais si la tâche doit passer soit plusieurs fois dans la même machine (je détaillerai la possibilité de ce cas plus tard) soit passer dans le

même type de machine plusieurs fois au cours du processus associé (des fours de plus en plus chaud par exemple) cette option n'est plus fiable. Avec ces nouvelles situations, le processus contiendrait beaucoup d'attributs compliquant la relecture et la portabilité de l'objet.
C'est pourquoi, les caractéristiques des machines sur une gamme sont placées directement sur les nœuds du graphe.

J'ai évoqué la possibilité de repasser plusieurs fois dans sur la même machine au sein d'une gamme. Ce n'est pas un cas impossible (exemple : un atelier de lavage, un vêtement peut passer plusieurs fois dans la même machine à laver, essoreuse, ...). Pour représenter ce cas là, il y a une subtilité à respecter. Instinctivement, on tracerais un arc retournant vers la machine en question, ce qui aurait pour effet de créer un cycle dans le graphe. Un cycle rend impossible la finition d'un processus sans ajouter des contrôles et des paramètres supplémentaires sur la gestion des processus.
Pour éviter de complexifier la gestion des processus, qui est une des raisons pour lesquelles le projet s'est ré-orienté sur les graphes de précédences, il suffit de créer un autre nœud possédant comme attribut la machine en question. Cette configuration ne pose pas de problème sauf si on choisit de passer deux fois de suite dans la même machine. Dans ce cas là, à moins de spécifier un traitement spécifique, on risque de bloquer l'atelier (ce qui est en fait une bonne représentation de la réalité). Voici ce qui se passe dans ce cas là : Je vais prendre l'exemple d'une machine à laver

1. Le vêtement **entre** dans la file d'attente de la machine
2. Le vêtement **accède** à la machine et se fait laver
3. Il doit repasser une deuxième fois dans la même machine mais il **occupe toujours** la machine. Il attend que lui-même libère sa propre place pour continuer. Or, pour libérer sa place, il faut qu'il accède de nouveau à la machine.(on veut en fait attendre que la machine soit vide pour y replacer le vêtement qui occupe la machine)
4. Le système est alors en **interblocage**

Ici on part du principe que le vêtement ne repasse pas par la file d'attente mais le problème est le même si la file d'attente est remplie à un moment de la simulation.

Une solution pour éviter cela est de faire une vérification à la fin de l'utilisation de la machine pour savoir si le traitement doit recommencer. La clé est de ne pas libérer la ressource tant que l'on en a besoin.

2.2.2 Les éléments de l'atelier

Notre graphe représente la gamme d'un atelier. Cette atelier est composé de diverses machines, files d'attentes, zones de stockage, etc, ... La catégorie regroupant ces objets présents dans un atelier s'appelle les **élément de l'atelier**. Une machine est un élément, une file d'attente aussi, ...
Il est clair qu'il est impossible d'implémenter tous les éléments qui existent dans les ateliers (trop de cas spéciaux, de personnalisations possibles, ...). Le projet doit donc être très flexible pour accueillir de nouvelles implémentations d'éléments. La solution que je propose est l'utilisation de l'**héritage**. Je propose dans le code du projet une classe "abstraite" (qui ne peut pas être implémentée) de laquelle doit hériter toute implémentation de nouveaux éléments.
Cette classe sert à **normaliser** les éléments est s'assurer que les fonctionnalités développées à ce jour soient compatibles avec ces possibles ajouts. Cette classe s'appelle **WorkshopElement**. Elle hérite elle-même de la classe **Resource** de Simpy. Une Resource est un objet qui possède une capacité (nombre de tâches pouvant s'exécuter en simultané dessus, par exemple le nombre de place dans un four). Il est possible d'effectuer une **Requête** sur une Resource. Sipmy gère automatiquement

2.2.3 Les tâches

Un point négatif de cette nouvelle approche est que désormais un *Process* de SimPy ne correspond plus à un *Processus* de notre simulation. Dans la première tentative d'implémentation cela devait être la cas mais

dans l'approche orientée graphe, les **Process** représenterons les **tâches** d'un processus. Pour illustrer ce propos, prenons un atelier avec deux machines (A et B) en série :
Un **Processus** au sens **simulation** correspond à la gamme de cette atelier utilisant les **deux machines** dans un certain ordre (disons A puis B). Ce processus est composé de **deux tâches** : utiliser la machine A et utiliser la machine B. On soumettra donc **deux Process** à la simulation, *process_utiliser_A* et *process_utiliser_B*.

Une tâche peut être créée par un **élément de l'atelier**. Je suis parti du principe qu'un élément ne peut réaliser qu'une seule tâche dont les paramètres peuvent varier. Ici aussi la flexibilité était une contrainte non négligeable. On doit être capable de rajouter des nouvelles tâches à volonté pour pouvoir décrire tous les comportements des éléments d'une simulation. C'est pourquoi une fois de plus, j'ai eu recours à l'héritage et à la définition d'une classe abstraite nommée **BaseTask** qui définit les normes d'implémentation pour créer une tâche compatible avec l'application.

Par exemple, il faut obligatoirement implémenter la fonction **do_the_task** définissant la série d'instruction d'un élément de l'atelier.

Dans le but de simplifier la compréhension du code et d'améliorer la flexibilité, les files d'attentes possèdent également une *tâche* à exécuter. Cela peut sembler étrange aux premier abords mais il est possible que certaines files d'attente ait des comportements spéciaux (exemple : dans un fast food, les éléments en attente d'être servis sont jetés si ils ne le sont pas assez vite). De cette manière, il est possible de personnaliser le comportement de tous les éléments de l'atelier.

Enfin, une tâche possède en attribut un événement *task_is_finished_event* qui se réalise lorsque la tâche est jugée achevée.

2.2.4 Processus

Finalement, si les **Process** de SimPy ne correspondent plus aux **Process** de la simulation, qu'est-ce qu'un processus au niveau de l'implémentation ?

Un processus sera une classe lisant les données du graphe pour en créer une **tâche** par nœud. Avec cette nouvelle implémentation, toutes les tâches sont lancées à la demande de création du processus. Cependant, toutes celles qui n'auront pas rempli leurs prérequis seront interrompues par une instruction **yield**. Les prérequis consistent à attendre que les tâches parents réalisent leur événement *task_is_finished_event*.

Un des travaux de cette classe sera de créer cette liste de prérequis, qui est donc une liste d'événements. Ensuite, au début de chaque tâche, on peut attendre que tous ces événements se réalisent de cette façon :

```
1 yield AllOf(env, liste_prerequis)
```

AllOf est une fonctionnalité proposée par SimPy permettant de créer un événement qui se réalise suivant les états d'une liste d'événements. En particulier, l'événement AllOf se réalise lorsque **tout** les événements de la liste se réalisent. Il existe l'événement **AnyOf** qui se réalise dès qu'au moins un événement de sa liste se réalise.

Un processus a accès à un dictionnaire ayant pour clé un identifiant d'un nœud du graphe et en valeur la tâche créée lors de la création du processus. Un processus est terminé lorsque toutes ses tâches ont réalisé leur événement *task_is_finished_event*.

2.3 Conclusion de la révision

Pour répondre à cette nouvelle modélisation orientée graphe, le langage **Python**, **networkx** et **Simpy** semblent posséder tout les outils nécessaire pour répondre au problème.

3 Organisation des classes du projet

Pour répondre à la modélisation orientée graphe, j'ai implémenté des classes qui possèdent les relations suivantes :

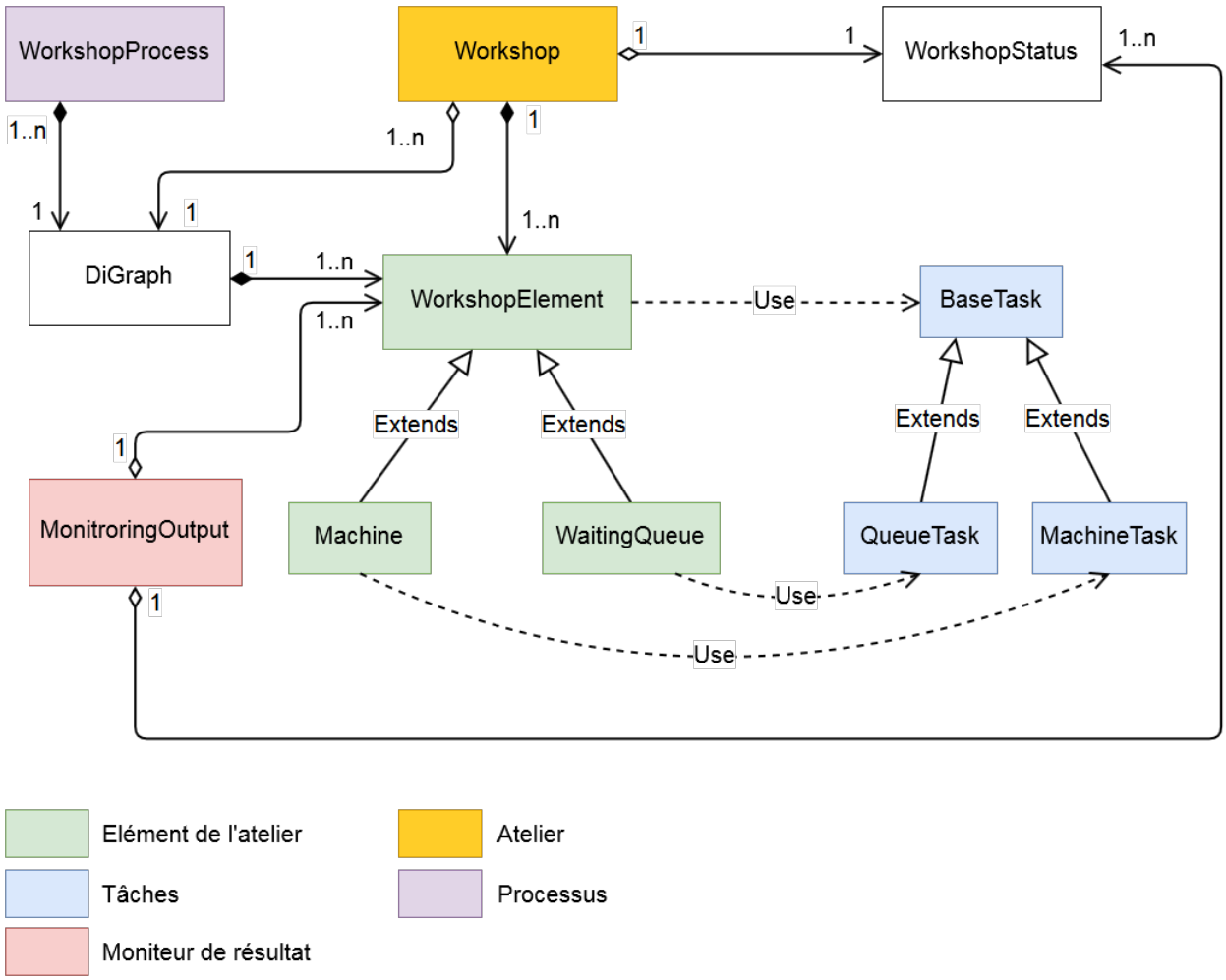


Figure 3 – *Relations entre les classes du projet*

Á ce point du rapport, j'ai déjà décrit l'intérêt des classes **WorkshopElement** (subsubsection 2.2.2), **BaseTask** (subsubsection 2.2.3), **DiGraph** (subsubsection 2.2.1) et WorkshopProcess (subsubsection 2.2.4).

Les classes **Machine** et **WaitingQueue** sont des implémentations de *WorkshopElement*. Elles ont été créées à titre d'exemple pour les futurs éléments d'ateliers à ajouter dans le projet. Elles servent également à vérifier la faisabilité du modèle et la justesse de l'implémentation.

Les classes **QueueTask** et **MachineTask** sont des implémentations de **BaseTask**. Elles aussi peuvent servir d'exemple pour l'ajout de nouvelles tâches exécutables par les éléments de l'atelier.

3.1 La classe Workshop

Cette classe représente l'atelier étudié. Elle contient un dictionnaire d'éléments (WorkshopElement) identifiés par une clé pouvant être n'importe quelle valeur pourvue qu'elle soit unique dans le dictionnaire.

3.2 La classe WorkshopStatus

L'objectif de la simulation est d'obtenir des résultats exploitables et de pouvoir retracer l'évolution de l'atelier au fil du temps. Il faut donc un responsable pour suivre l'évolution et l'enregistrer en mémoire ou dans un fichier.

La difficulté de cette tâche est qu'on ne peut pas prévoir tous les indicateurs qui seront utiles pour toutes les simulations. Il faut donc une fois de plus prévoir quelque chose d'assez flexible pour anticiper ces modifications ou ajouts. J'ai fais le choix de séparer l'atelier physique et le suivi de données sur cet atelier. La raison à cela est que de cette manière la classe *Workshop* pourra rester inchangée, évitant de désorganiser toute la logique de l'application. En cas d'un ajout d'indicateur, il suffit d'écrire le code correspondant dans la classe *WorkshopStatus*. Évidemment cette classe risque de devenir assez chargée, il serait alors plus judicieux de scinder cette classe en plusieurs plus spécialisées.

3.3 La classe MonitoringOutput

Voici la classe nécessitant certainement le plus de **refactoring** (ré-usinage de code).

Cette classe se charge de concentrer tous les éléments et statut de l'atelier qui doivent être surveillés et qui participeront dans l'affichage et la génération des données finales. Chaque élément possède ses propres indicateurs et certains indicateurs concerne plusieurs éléments à la fois. Il faut donc un responsable pour regrouper les éléments qui doivent l'être. La difficulté est que ce responsable doit savoir calculer les indicateurs de ces groupes.

Je vais prendre l'exemple des machines et des files d'attentes :

Les files d'attentes ont pour indicateur le taux d'occupation pendant la simulation. Ce taux est calculé en parcourant toutes les données stockées par la file pendant la simulation.

Pour les machines, le même taux nous intéresse mais en plus, on va rechercher le taux d'utilisation des machines (une machine est occupée lorsqu'un processus occupe un point de capacité de la machine et elle est utilisée lorsqu'elle réalise une opération sur un processus).

Pour d'autres éléments, un autre taux pourrait apparaître. Comme il y a plusieurs taux par éléments de l'atelier, il est assez dur de trouver un moyen simple de créer un récupérateur d'indicateurs générique qui pourrait trouver dynamiquement les informations sur les éléments pour ensuite les extraire. Cependant, avec l'expérience que j'ai acquis en langage Python, je pense qu'il existe des manières de réaliser cela de façon assez "propre" et "pythonique".

Il faut à la fin de la simulation que tous les éléments qui nous intéressent puissent faire remonter leurs données. C'est pourquoi cette classe contient une liste par type d'élément et les éléments "s'inscrivent" à la liste correspondant à leur type pour se faire surveiller et apparaître dans les résultats finaux de la simulation.

Cette classe est assez conséquente en terme de lignes de code car elle contient les méthodes de calculs de chacun des éléments de l'atelier (et encore il n'y a que 3 types pour le moment) ainsi que les méthodes pour normaliser les données et celles pour préparer l'affichage et celles pour exporter vers un fichier CSV et celle pour afficher. Elle possède donc trop de responsabilité et devrait être scinder en plusieurs (au moins une pour gérer l'affichage et une pour préparer les données).

8 Analyse d'un atelier avec l'application

1 Rappel des objectifs

Le projet consiste à réaliser une application qui peut simuler le comportement d'un atelier de fabrication donné. Dans une simulation, ce qui intéresse après avoir construit le modèle et exécuté le programme, c'est de pouvoir observer les résultats et surtout pouvoir les analyser pour en déduire des conclusions sur les forces et les faiblesses de l'atelier simulé.

2 Sorties de l'application

2.1 Affichage des résultat

Le premier type de sortie de l'application est l'affichage. Á la fin d'une simulation, des courbes représentant l'évolution des indicateurs s'affichent à l'écran de l'utilisateur. Il y a une fenêtre par type d'élément de l'atelier plus une fenêtre par *WorkshopStatus* et une courbe par élément surveiller de l'atelier affichée dans la fenêtre correspondante.

Ces courbes sont précisées par des valeurs supplémentaires situées juste en dessous. Ces valeurs permettent de juger plus rapidement l'efficacité de l'atelier mais aussi d'éviter les "trompes l'œil" sur des simulations possédant beaucoup de données.

2.1.1 Présentation de l'affichage

L'affichage est réalisé à l'aide de la bibliothèque **pyplot** qui permet assez simplement d'afficher des graphiques (courbes, histogrammes, ...) à partir de vecteurs de données (ou de listes de données). J'ai essayé de trouver une solution rapide à mettre en place et simple dans un premier temps pour pouvoir valider le principe.
Pyplot a montré des limitations quand à la personnalisation de l'affichage de pleins de graphe sur la même fenêtre. Bien que très puissant car il place automatiquement les graphes sur les fenêtres en arrangeant leur taille pour que tout soit visible, on se rend compte très vite que l'affichage de données hors graphe (titre, légende, annotations, ..) réduit énormément la taille des graphiques et donc nuit à la lisibilité. C'est pourquoi la plupart des annotations des graphes sont abrégées et/ou partielle.

Néanmoins, pour ce début de projet, pyplot est tout de même très utile et rapide à mettre en place.

2.1.2 Exemple de résulat d'affichage

Dans cette section, je vais exposer l'affichage obtenu pour une simulation courte (10 processus) et une simulation plus longue (100 processus).

Voici le graphe de la gamme simulée :

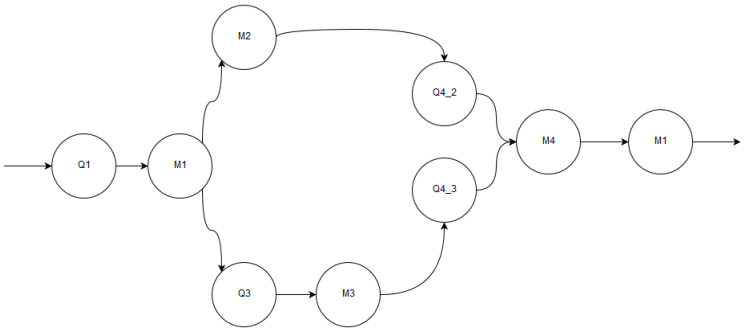


Figure 1 – *Graphe simulé*

Ce graphe présente une subtilité, un processus doit utiliser deux fois la machine M1, une fois lors vers le début et une fois tout à la fin.

Lorsque l'on lance la simulation avec 10 processus, on obtient les résultats suivants :

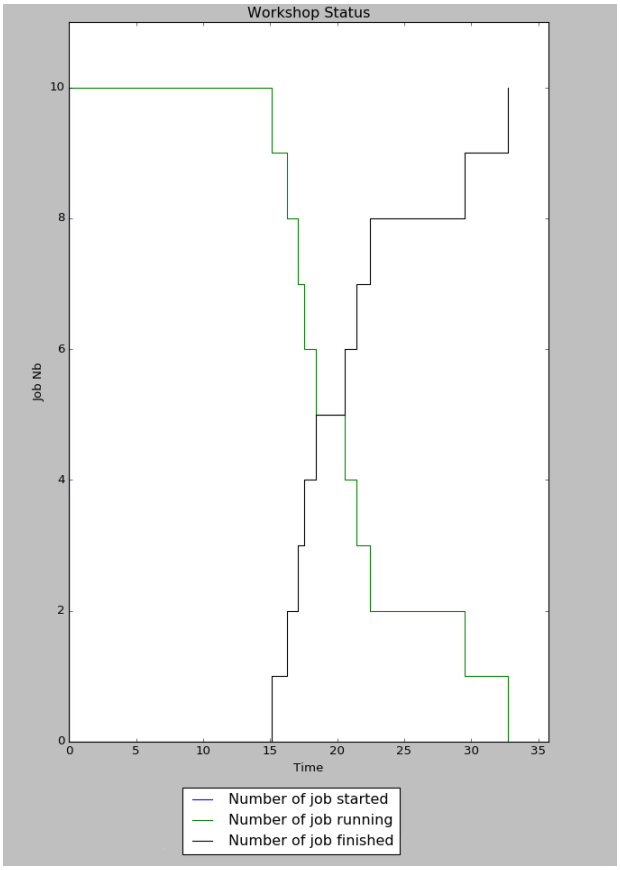


Figure 2 – *Résultats liés au WorkshopStatus*

Sur ce graphique, on peut observer l'évolution du nombre de processus démarré, en cours d'exécution et fini. On peut remarquer que globalement l'avancement est régulier jusqu'à environ t = 23. Ensuite, il y a

un ralentissement.

Passons aux graphiques suivants :

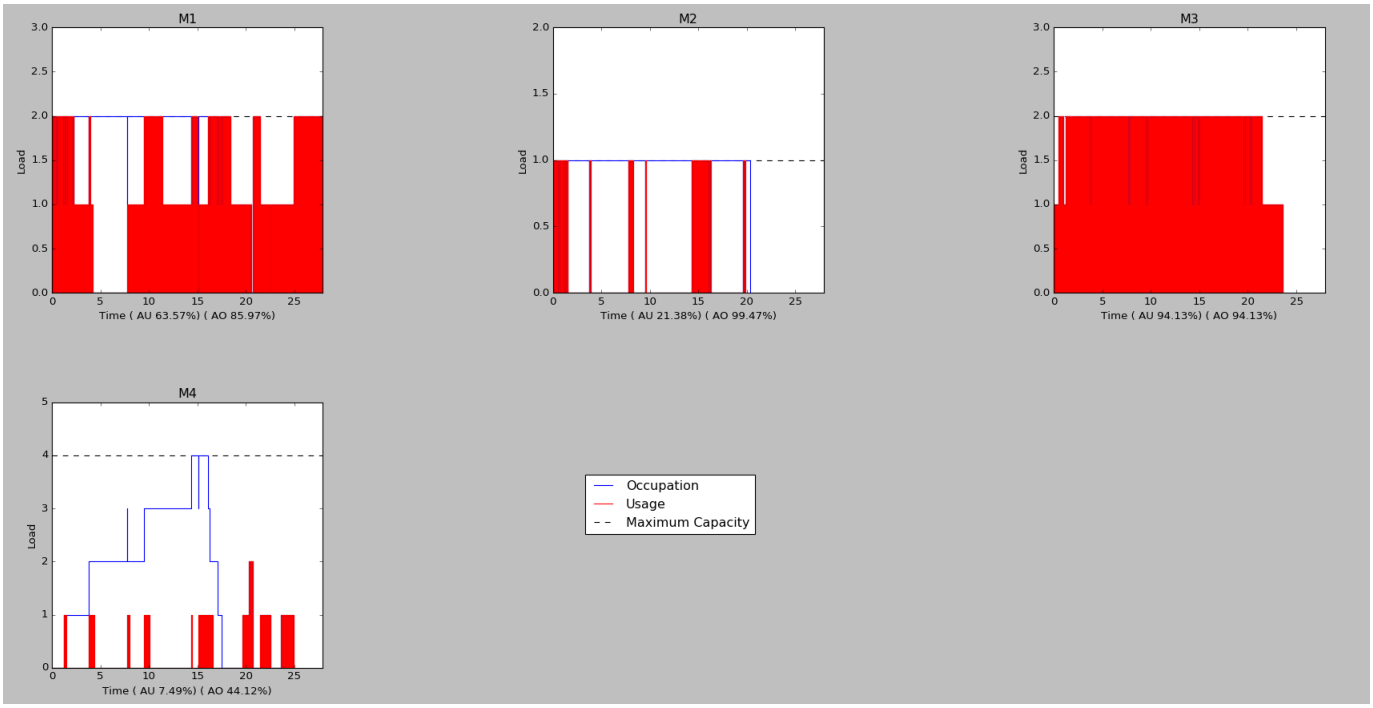


Figure 3 – *Résultats liés aux Machines*

Cette fenêtre présente 4 graphiques (1 par Machine de l'atelier). On peut suivre l'évolution de l'occupation des machines (en bleu) et celle de l'utilisation (en rouge).
Pour donner un début d'analyse, si sur un graphe on observe souvent la ligne bleue au dessus des zones rouges, c'est que la machine réalise ses tâches plus vite que la suite du processus n'avance. Ce n'est, à première vue, pas une ressource bridant l'atelier. Pour remédier à cela, on peut augmenter la vitesse de traitement des tâches suivant celle de cette machine ou bien augmenter la taille de la file d'attente (ou de la capacité de la machine) suivant la machine courante.

Á l'inverse, si on constate que la zone rouge est très souvent au niveau de la ligne bleue, cela peut indiquer que la machine est responsable d'un goulot d'étranglement. Pour en être sûr, il faut regarder si la tâche précédent la machine est souvent **bloquée** ou si la tâche suivante est souvent en **famine** (pas assez de travail).

En plus de regarder les courbes, on peut se baser sur les indicateurs situés juste en dessous. Ici, on a accès au **AU** (Active Utilisation rate) et au **AO** (Active Occupation rate).
L'Active Utilisation correspond au taux d'utilisation de la machine sur la période pendant laquelle la machine est nécessaire (de la date de sa première utilisation à la date de sa dernière utilisation). Les indicateurs nommés *Active* sont un précision des indicateurs nommés *Total* qui eux effectuent leurs calculs sur toute la durée de la simulation.
L'Active Occupation représente le taux d'occupation de la machine sur la période pendant laquelle la machine a été nécessaire.

Voici comment ces taux pourraient être interprétés [1] :

— AU : un haut taux signifie que la machine est très souvent en train de travailler. Un faible taux correspond à une faible durée de travail. Un taux de 100 % peut indiquer que la machine n'est pas assez rapide et qu'elle peut causer un ralentissement de la chaîne de production.

1. En fait l'interprétation dépendra du but recherché par la simulation.

— AO : un haut taux indique que la capacité de la machine est bien utilisée (suffisamment de tâche en simultanée). Un faible taux indique que la machine est peut être surdimensionnée ou que la tâche précédente est trop lente. Un taux de 100 % peut indiquer que la tâche suivante n'est pas assez rapidement réalisée ou que sa capacité (file d'attente ou machine) est trop faible.

Enfin, voici les résultats pour l'occupation des files d'attentes de l'atelier :

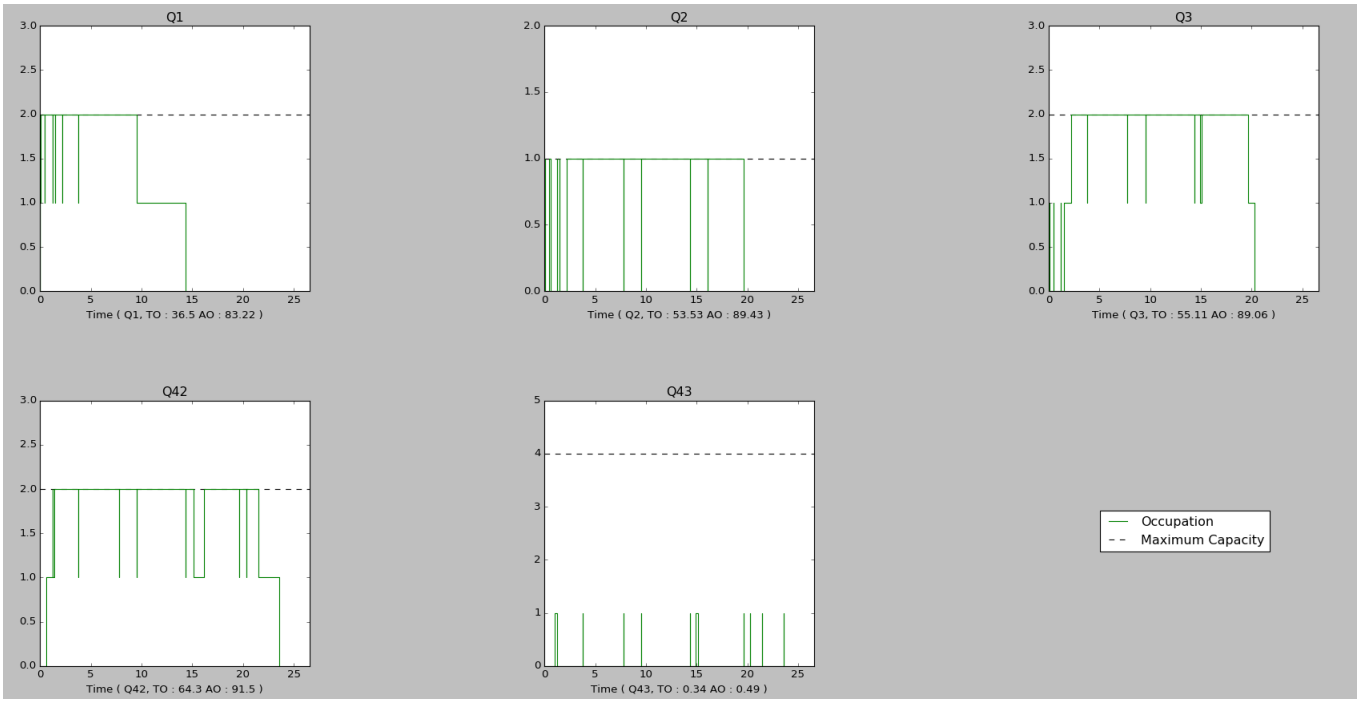


Figure 4 – *Résultats liés aux Files d'attente*

Pour les graphes, les remarques sont les mêmes que pour les machines. On peut dire par exemple que la file Q43 est surdimensionnée pour cette simulation puisqu'elle n'atteint jamais sa capacité maximum. Mais attention, si on accélère le traitement de la tâche alimentant cette file, il est possible que la capacité s'avère finalement bien choisie. Il faut essayer de se limiter à une modification à la fois en commençant par le début de la gamme. [2]

En ce qui concerne les indicateurs numériques, on a cette fois **TO** (Total Occupation rate) et **AO** (Active Occupation rate). Ces indicateurs se comportent comme ceux présents sur les machines.

Pour finir sur l'affichage, je vais présenter les résultats obtenus pour une simulation plus conséquente (100 processus démarré en même temps) :

Toujours en utilisant le même graphe, voici ce que l'on obtient pour le *WorkshopStatus* :

Quelque chose de très intéressant est visible ici : tous les processus n'ont pas pu être achevés. Cela est dû au deuxième passage dans la machine M1. Au bout d'un certain nombre de processus exécutés en simultanés, tous les éléments de l'atelier se retrouvent occupés à leur capacité maximum et donc tout l'atelier se bloque. Ici, grâce à la simulation et à l'analyse de ce graphique de résultat, on peut assurer que l'atelier est sous dimensionné par rapport à la charge de travail qu'il reçoit.

Voyons les autres graphiques :

Première remarque, les graphiques sont beaucoup moins visibles. Et plus on augmente la durée de la simulation (en rajoutant des processus par exemple) plus la lisibilité baisse. C'est alors que les indicateurs situés en dessous des graphes prennent toutes leur importance. Il permettent de rapidement déduire

2. Là encore, suivant les objectifs de la simulation les stratégies peuvent changer.

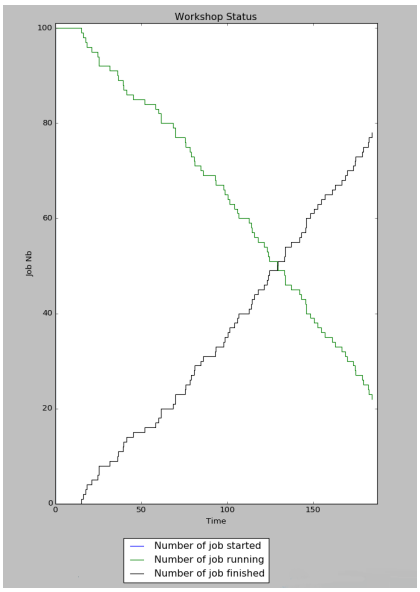


Figure 5 – *Résultats liés au WorkshopStatus*

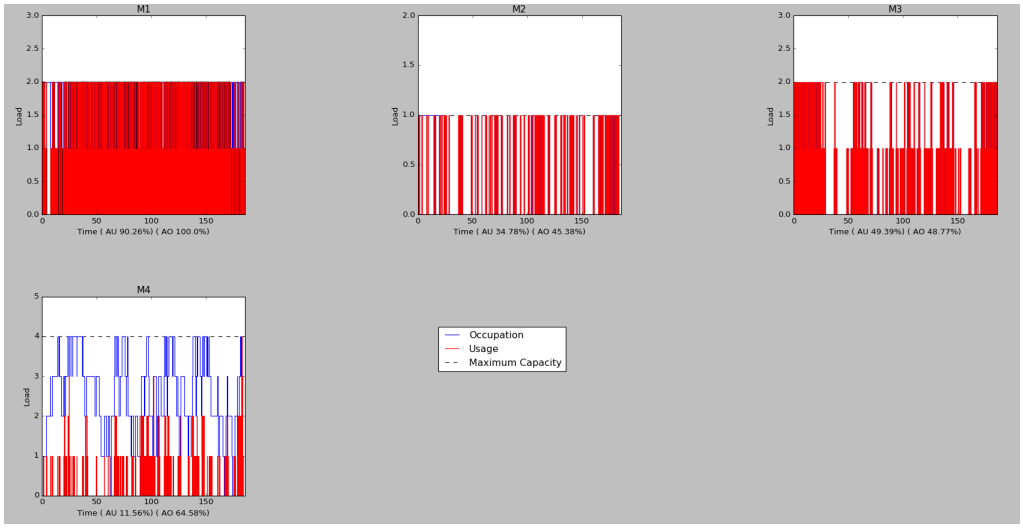


Figure 6 – *Résultats liés aux Machines*

les forces et les faiblesses de l'atelier. Une fois les ressources problématiques repérées, pyplot permet d'effectuer un zoom sur une partie d'un graphique, démêlant ainsi l'illisibilité initiale.

Seconde remarque, tout à l'heure, la machine M3 semblait sous-dimensionnée car elle possédait des taux d'occupation et d'utilisation proche de 100 %. Cependant, pour cette simulation on observe qu'elle n'est utilisée qu'aux alentours de 50 %. Cet exemple souligne bien l'importance de réaliser plusieurs simulations avec des situations différentes pour s'assurer de la robustesse d'un atelier.

Observons maintenant le résultats pour les files d'attente :

On retrouve le problème de la lisibilité sur ces graphiques. Une fois de plus les indicateurs numériques permettent de mieux cerner les capacités de l'atelier.

Notons que les files Q1 et Q2 semblent avoir des occupations similaires au niveau des graphiques. Cependant leurs indicateurs sont pourtant très éloignés avec quasiment 100% pour Q1 et environ 25 % seulement pour Q2.

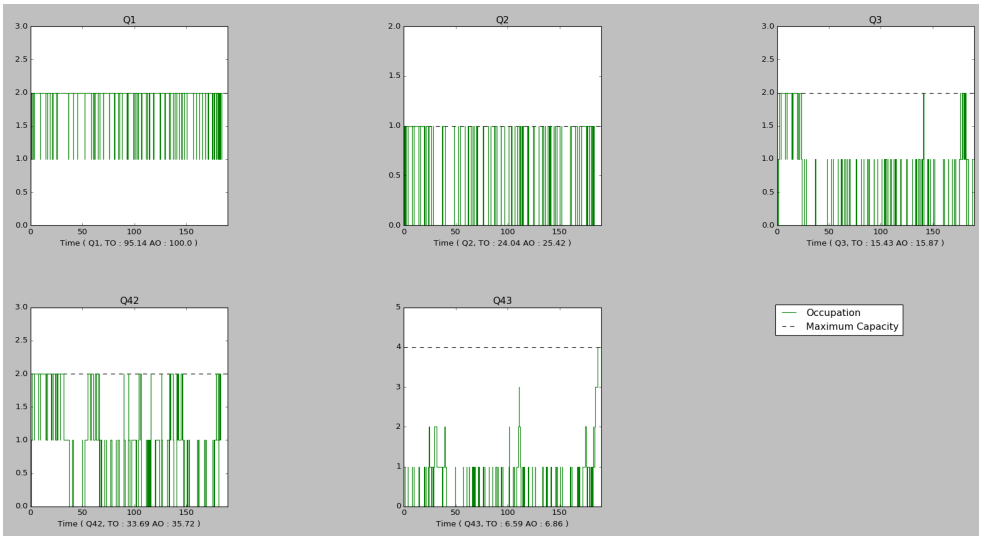


Figure 7 – *Résultats liés aux Files d'attente*

2.2 Export vers fichier CSV

Il est évident que les données doivent être persistées. Soit pour être relue humainement, soit pour être analyser informatiquement, soit tout simplement pour les comparer avec des données d'autres simulations.

J'ai donc choisi d'introduire une fonctionnalité pour exporter les données obtenues dans des fichier CSV (Comma-Separated Values). C'est simplement un fichier avec les données en brutes séparées par un point virgule.
L'export ressemble à ceci :

result_machine_usage.csv

```
1  M1;PROCESS_ID;1;2;1;3;2;4;4;5;3;6;6;7;7;8;8;5;2;9;9;10;10;2;1;4;4;11;11;1
2  M1;DATE;0;0;0.048072575663357785;0.048072575663357785;0.48134757353371693
3  M1;NB_USER;1;2;1;2;1;2;1;2;1;2;1;2;1;2;1;0;1;2;1;2;1;0;1;2;1;2;1;0;1;0;1;
4  M2;PROCESS_ID;1;1;2;2;4;4;3;3;6;6;7;7;8;8;5;5;9;9;10;10;11;11;12;12;13;13
5  M2;DATE;0.048072575663357785;0.5853494543202835;0.5853494543202835;1.2114
6  M2;NB_USER;1;0;1;0;1;0;1;0;1;0;1;0;1;0;1;0;1;0;1;0;1;0;1;0;1;0;1;0;1;0;1;
7  M3;PROCESS_ID;1;2;2;4;1;3;4;6;6;7;7;8;3;5;8;9;9;10;5;11;10;12;12;13;13;14
8  M3;DATE;0.048072575663357785;0.48134757353371693;0.9993589827802265;1.221
9  M3;NB_USER;1;2;1;2;1;2;1;2;1;2;1;2;1;2;1;2;1;2;1;2;1;2;1;2;1;2;1;2;1;2;1;
10 M4;PROCESS_ID;2;2;1;1;4;4;6;6;7;7;3;3;8;9;9;8;5;5;10;12;13;13;10;12;14;14
11 M4;DATE;1.2114574375342992;1.5137379910100277;3.790783393749266;4.3737964
12 M4;NB_USER;1;0;1;0;1;0;1;0;1;0;1;0;1;2;1;0;1;0;1;2;3;2;1;0;1;0;1;0;1;0;1;
13
```

Figure 8 – *Données exportées sur l'utilisation des machines*

Chaque ligne correspond à une liste de valeurs précédées par l'identifiant de la machine concernée et le nom de l'indicateur surveillé.

En brut et à l'œil humain, ces données ne sont pas très lisibles. Cependant elles peuvent très rapidement être importées dans un tableur ou être analysées informatiquement.

9 Création de tests

L'application présente une base de fonctionnalités sur lesquels reposent l'intégrité des données obtenues pour une simulation. Il faut donc s'assurer que ces fonctionnalités soient toujours justes. Pour cela, il est efficace de mettre en place un certain nombre de tests qui s'assureront qu'aucune modification du projet n'a entraînée de biais dans la création/gestion des données, en une seule expression, éviter la régression de code.

Ces tests ont deux objectifs principaux :

1. S'assurer que les fonctionnalités en place se comportent comme souhaité
2. Rassurer les futurs personnes qui ajouteront des fonctionnalités à ce projet. En effet, si les tests s'avèrent toujours juste après un ajout de fonctionnalité ou une modification du code existant (refactoring par exemple), cela signifie qu'il n'y a pas eu régression de code.

Un point sur la création des tests m'a déstabilisés par rapport aux différents cours que j'ai pu suivre : le projet est implémenté de manière à être très flexible (ajout de nouvelles tâches, ajout de nouveaux éléments d'atelier, etc ...) ce qui fait qu'il ne m'est pas possible de créer un jeu de test paré au différents ajouts possibles. De plus, le projet contenant des classes abstraites, il n'est pas possible de tester directement ces classes (car non instanciables). Pourtant, au vu des enjeux sur l'intégrité des données, il me fallait mettre quelque chose en place.

La solution que j'ai retenu consiste à montrer l'exemple aux futurs codeurs du projet. J'ai donc implémenté des classes héritant de ces classes abstraites (Machine, MachineTask, WaitingQueue, QueueTask) et réalisé une base de tests sur ces classes là pour donner des idées pour les futurs ajouts.

Pour ce qui est des autres fonctionnalités, plus basiques, j'ai pu implémenter des tests unitaires plus classiques. Mais ces tests ne permettent pas d'assurer l'intégrité des données d'une simulation. Pour compléter ces tests, j'ai mis en place des tests d'intégrations qui consistent en la création d'un atelier précis et d'une gamme de cet atelier pour ensuite exécuter la simulation. Ces tests vérifient que les simulations préparées donnent les résultats attendus. Ces résultats sont ceux que la simulation doit obtenir.

Á cette étape du projet, tous les tests sont validés.

10 Améliorations et avenir du projet

Ce projet est parti de la bibliothèque SimPy pour en arriver à un ensemble de code proposant une solution pour simuler des ateliers de fabrication.

Le projet est très flexible puisqu'il permet d'ajouter de nouveaux éléments d'atelier, des nouvelles tâches, des nouveaux indicateurs, etc, ... Dans ce sens, il valide l'idée de départ. Cette première partie de développement résulte en un cœur d'application. Maintenant, il serait intéressant de la développer un peu plus.

1 Affichage des résultats

L'affichage des résultats est pour le moment très sommaire. Peu d'informations s'affichent à l'écran sur les indicateurs et l'apparence est assez austère.

L'amélioration de l'affichage n'est pas que "pour faire joli", c'est un point crucial dans l'aide à l'analyse des résultats de la simulation. Si l'affichage est trop limité, soit la détection de problèmes et d'avantages sera rendue plus difficile, soit elle sera induite en erreur.

La difficulté est de trouver un affichage qui rend lisible les données de simulations de longue durée (qui possèdent beaucoup de données). Ou alors, il faut trouver un système pour indiquer les zones "potentiellement critiques" sur le graphique (une famine passagère, une occupation maximale, ...). Pour simplifier la lecture des indicateurs, il pourrait être intéressant de proposer une coloration du texte (vert lorsque l'indicateur est bon, orange lorsqu'il est moyen, rouge lorsqu'il est mauvais). Pour parfaire cette idée, il faudrait demander à la personne qui exécute la simulation les valeurs qu'ils considèrent comme bonne, moyenne ou mauvaise.

Enfin, la représentation physique du graphe avec de la coloration sur les indicateurs des éléments directement sur les nœuds (pourquoi pas les arcs) permettrait d'observer la cause d'un goulot d'étranglement, d'une famine ou d'une surcharge de travail.

2 Exportation des données

L'exportation sous forme CSV est pratique pour informatiser l'analyse ou pour l'exporter dans un tableur mais il serait intéressant pour certaines personnes de générer des "documents" résumant les points

majeurs de la simulation (on peut prendre exemple sur les documents générés par Arena [WWW5]).

En option pour l'utilisateur, on pourrait demander quel format de données il souhaite en sortie (séparation par ' ;' ou par ',' ou autre chose).

3 Interface de création

La création de graphe pour représenter une gamme d'un atelier et la création même de l'atelier est faite "en dur" dans le code du projet. Pour rendre l'application plus agréable à utiliser et simplifier la création, il semble indispensable de proposer une interface graphique qui permettrait dans un premier temps de créer son atelier avec les éléments d'atelier implémentés disponibles. Ensuite, on pourrait demander d'indiquer les gammes de l'atelier qui seront traduites en graphes dans l'application. Enfin, l'interface proposerait de paramétrer la simulation (temps réel, as fast as possible, durée maximum, date de début, nombre de processus, conditions d'arrêt, etc...).

Le défi ultime étant de permettre la création de nouveaux éléments directement depuis cette interface (sorte de méta-langage de très haut niveau).

4 Test sur un cas réel

Pour s'assurer que le projet est porteur d'une solution à la simulation d'atelier de fabrication, il faudrait tenter de simuler un atelier réel. Si en plus cet atelier possède des enregistrements de données sur son comportement des périodes passées, on pourrait même comparer les résultats.

Le second intérêt de ce test serait de voir la part de réflexion, de recherche, de modélisation, d'ajout de fonctionnalités qu'il faut pour réaliser la simulation de cet atelier. Un bon résultat serait d'avoir un gain de temps sur la modélisation de l'atelier et l'implémentation de la simulation en elle-même.

5 Plus(+) de dynamisme

Pour le moment, les simulations données en exemple créent un paquet de processus et les soumettent à la simulation tous en même temps. Il y a alors un nombre fixe de processus, connu à l'avance et il n'est pas possible d'en détruire ni d'en créer. J'ai vérifié la possibilité de créer des processus pendant que la simulation se déroule. Il n'y a pas de problème, cela fonctionne très simplement. Par exemple, on peut placer des branches conditionnelles dans certaines tâches qui engendreront la création ou non d'un nouveau processus (par exemple, une panne de machine). On peut également créer des tâches qui serviront de générateurs de processus (la tâche s'exécute, elle soumet un processus, elle s'endort pendant une certaine durée, elle se réveille et recommence).

6 Vérifier les autres cas de base

Il reste des cas de base à vérifier comme par exemple le cas avec opérateur.

7 Généralisation du projet

Ce projet s'est limité à la simulation d'atelier de fabrication. Mais avec le niveau de flexibilité pour les ajouts d'éléments et de tâches, est-ce vraiment le cas ? Peut-on simuler autre chose que des ateliers de fabrication avec ce noyau de fonctionnalités ?

La réponse que je vais fournir se base sur l'implémentation qui a été réalisée dans ce projet. Il me semble compliqué d'assurer avec certitude que les propos qui vont suivre sont d'une totale justesse.

Si on prend les différentes étapes pour construire une simulation avec ce projet, il faut :

— Travailler sur un système possédant des ressources qui peuvent réaliser des actions (des tâches).
— Ce système doit tolérer la notion de processus. C'est-à-dire qu'il doit être cohérent de parler de suites d'actions à réaliser pour atteindre un but.
— Cette suite d'action (notre gamme) doit être représentable par un graphe orienté sans cycle.
— Il doit être possible de collecter des données sur l'évolution du système.

Ou alors, si on peut simplement faire une analogie avec la modélisation d'un atelier de simulation, on doit pouvoir utiliser ce projet pour simuler d'autres genres de système.

Exemple : Un réseau de transport en commun
Prenons les bus. Notre système est la compagnie qui gère les bus. Ce sont ses ressources. La compagnie possède également des arrêts de bus (encore des ressources) et elle a planifié des trajets.

Un processus tout simple est un bus qui fait sa tournée. Il doit ensuite passer par une succession de ressources (les arrêts) jusqu'à retourner à sa base.

Un arrêt peut exécuter la tâche suivante : faire monter les gens. Grossièrement, cela consiste à simuler une attente plus ou moins longue.

Le trajet entre deux arrêts peut être représenté par une tâche qui la aussi consiste à attendre une certaine durée pour simuler le déplacement du bus. On peut même simuler des accidents en rajoutant des branches conditionnelles dans la tâche.

11 Documentation

Afin d'aider la reprise du projet et de faciliter la compréhension du code, j'ai mis en place une génération de la documentation. Elle a été générée à partir du moteur Sphinx. Sphinx récupère les commentaires Python du projet et en construit une documentation sous forme de site web local (fichier html). Il est assez simple de retrouver un terme dans cette documentation et la lecture y est plus aisée que dans le projet en lui-même.

La génération est très simple, il suffit d'installer Sphinx sur son ordinateur puis de lancer la commande :

```
1 make html
```

dans la racine du dossier de documentation. Sphinx accepte beaucoup de paramétrage. On peut donc contrôler de manière précise la génération de la documentation. Je ne m'y suis pas beaucoup attardé pour le moment. Toute proposition pour améliorer l'aspect visuel de la documentation est bienvenue.

12 Rétrospection

Dans cette section, je vais faire un retour sur les éléments du projet que j'aurais peut être abordé différemment si je devais recommencer le projet avec les connaissances que j'ai acquis depuis.

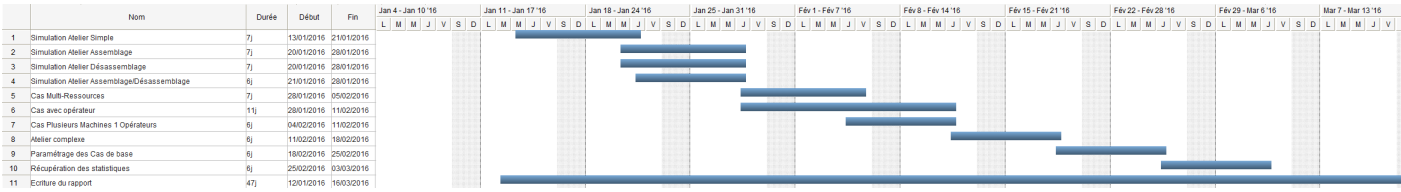
Pour ce qui est de la partie recherche, je pense avoir perdu du temps sur le fait d'avoir réalisé une étude plus poussée de l'aléatoire dans l'informatique et les simulations. Ce n'était pas un point très critique à ce point du projet. Á la place, je pense que j'aurais pu me concentrer davantage sur la maîtrise du vocabulaire technique afin d'être plus précis dans mes explications.

La phase de recherche a été très efficace cependant car si peu après le début de la phase de développement la modélisation choisie s'est avérée trop compliquée à implémenter, les recherches effectuées m'ont permis de rapidement m'adapter à la nouvelle modélisation orientée graphe.

Au niveau des connaissances sur les outils utilisés dans le projet, notamment Python et SimPy, il est clair que mon implémentation aurait changé. J'ai appris pendant les cours de Python qui se sont déroulés en parallèle de ce projet des mécanismes de programmation que je ne connaissais pas avant et qui tranchaient avec mes habitudes de programmation. Par exemple, j'ai appris que vérifier systématiquement le type d'un objet passé en paramètre à une fonction n'est pas une pratique très *pythonique*. Dans le cas de SimPy, la bibliothèque reposait sur des notions assez poussées de Python. La documentation étant succincte et le code fourni étant très optimisé et épuré, il m'a été difficile de bien saisir les possibilités et les limitations de SimPy tôt dans le projet.

13 Bilan sur la planification

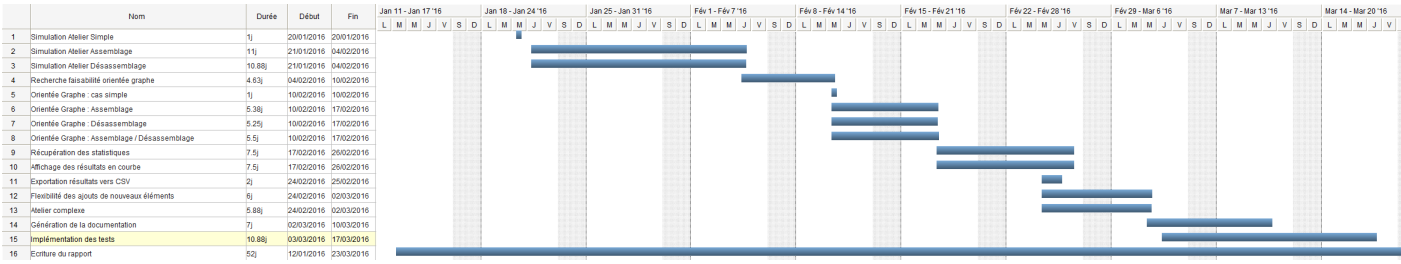
Á la fin de la période de recherche, j'ai construit un diagramme de Gantt à suivre pour cette phase de développement :



	Nom	Durée	Début	Fin
1	Simulation Atelier Simple	7j	13/01/2016	21/01/2016
2	Simulation Atelier Assemblage	7j	20/01/2016	28/01/2016
3	Simulation Atelier Désassemblage	7j	20/01/2016	28/01/2016
4	Simulation Atelier Assemblage/Désassemblage	6j	21/01/2016	28/01/2016
5	Cas Multi-Ressources	7j	28/01/2016	05/02/2016
6	Cas avec opérateur	11j	28/01/2016	11/02/2016
7	Cas Plusieurs Machines 1 Opérateurs	6j	04/02/2016	11/02/2016
8	Atelier complexe	6j	11/02/2016	18/02/2016
9	Paramétrage des Cas de base	6j	18/02/2016	25/02/2016
10	Récupération des statistiques	6j	25/02/2016	03/03/2016
11	Ecriture du rapport	47j	12/01/2016	16/03/2016

Figure 1 – *Planning prévu avant le développement*

Évidemment, à cause de la réorientation de l'implémentation au milieu de la phase de développement, la planification réellement suivie est très différente de celle prévue :



	Nom	Durée	Début	Fin
1	Simulation Atelier Simple	1j	20/01/2016	20/01/2016
2	Simulation Atelier Assemblage	11j	21/01/2016	04/02/2016
3	Simulation Atelier Désassemblage	10.88j	21/01/2016	04/02/2016
4	Recherche faisabilité orientée graphe	4.63j	04/02/2016	10/02/2016
5	Orientée Graphe : cas simple	1j	10/02/2016	10/02/2016
6	Orientée Graphe : Assemblage	5.38j	10/02/2016	17/02/2016
7	Orientée Graphe : Désassemblage	5.25j	10/02/2016	17/02/2016
8	Orientée Graphe : Assemblage / Désassemblage	5.5j	10/02/2016	17/02/2016
9	Récupération des statistiques	7.5j	17/02/2016	26/02/2016
10	Affichage des résultats en courbe	7.5j	17/02/2016	26/02/2016
11	Exportation résultats vers CSV	2j	24/02/2016	25/02/2016
12	Flexibilité des ajouts de nouveaux éléments	6j	24/02/2016	02/03/2016
13	Atelier complexe	5.88j	24/02/2016	02/03/2016
14	Génération de la documentation	7j	02/03/2016	10/03/2016
15	Implémentation des tests	10.88j	03/03/2016	17/03/2016
16	Ecriture du rapport	52j	12/01/2016	23/03/2016

Figure 2 – *Planning suivi lors du développement*

On peut voir sur ce diagramme la période où j'ai eu des difficultés avec la première proposition d'implémentation. Après cette phase cependant, on peut noter que les tâches ont à peu près la même durée. Le découpage suivi est donc assez équilibré finalement.

Conclusion sur la partie développement

La phase de développement fût assez courte mais très enrichissante. La solution initiale provenant de la partie recherche était à la fois faisable et cohérente mais elle devenait de plus en plus compliquée à mettre en place et mes limitations en ce qui concernait les outils utilisés ne m'ont pas aidé. Le choix de revoir l'implémentation était un bon choix. C'est à ce moment là que des notions non utilisées, vues pendant la phase de recherche, sont devenues un atout majeur. Le changement de cap s'est très bien passé même s'il a entraîné une petite phase de recherche supplémentaire (surtout en terme d'outils : networkx par exemple).

Globalement, le cœur du projet est implémenté. Il faut maintenant le renforcer en testant sa compatibilité avec tous les cas de base présentés. Pour rester sur les tests, je pense que j'ai été trop léger sur le nombre de tests ou sur leur efficacité. Une partie de ce manque d'exhaustivité est dû à la subite réorientation du projet, qui a bousculé la planification initiale de la phase de développement. Pour finir sur les tests, je me suis intéressé à placer mon projet sur le serveur Jenkins de l'école mais en lisant la documentation de celui-ci, j'ai remarqué que le langage Python n'y était pas supporté. L'automatisation des tests est donc repoussée à plus tard.

Le point plus faible de cette phase est au niveau de la gestion des sources du projet. Je n'ai pas pris le temps de faire les démarches pour mettre en place un système de gestion de versions. Ce manque n'était cependant pas critique car j'étais seul à effectuer des modifications sur le code du projet. De plus, cela m'a obligé à assurer une bonne redondance des sources du projet pour pallier à une éventuelle panne matérielle.

Enfin, la création de documentation à l'aide d'un nouvel outil était une partie assez riche en apprentissage.

Troisième partie

Conclusion du projet

En conclusion de ce rapport, je vais faire un point sur l'état du projet par rapport aux objectifs qui étaient fixés.

Tout d'abord, à la question est-ce que SimPy est adaptée pour réaliser des simulations orientées processus, la réponse est oui. Les fonctionnalités proposées sont même très puissante et assurent une gestion du moteur de la simulation, ce qui permet de se concentrer sur les vrais besoins.

Ensuite, il semble possible de simplifier le processus de création d'une simulation d'un atelier de fabrication. Le principe retenu est légèrement différent de celui imaginé initialement mais il permet de représenter simplement un atelier dans l'application. De plus le code du projet est prévu pour supporter des ajouts et des personnalisations d'éléments, ce qui permettra de représenter un très grand échantillons d'ateliers différents.

La rédaction d'une documentation et la présence d'un bon nombre de commentaires au sein du code devrait aider à la reprise du projet. Á cela, on peut ajouter les classes "exemples" qui permettent de donner une idée sur le procédé pour ajouter des composants dans l'application.

Le projet est ouvert à de nombreuses améliorations pour être plus accessible et plus performant. Il possède également un but de généralisation consistant à vérifier sa portabilité vers d'autre domaine que les ateliers de simulation.

D'un point de vue formateur, ce projet m'a permis de mieux maîtriser le Python et de découvrir des bibliothèques de fonctions manipulant des concepts intéressants. L'approfondissement de notions sur la simulation était un bon exercice. Les sources d'informations étaient diverses (livres, articles, internet, cours, ...), j'ai ainsi pu comparer différentes approches. Les sources n'étaient pas toutes dans la même langue, certains termes techniques ont donc été assez dur à identifier dans les différentes langues.

J'ai beaucoup appris durant ce projet. Le fait de partir de zéro permet d'orienter le projet dans la direction qui semble la meilleure mais il faut faire attention à ne pas vouloir trop faire d'un coup, surtout lorsque le sujet est vaste et qu'il permet une grande flexibilité des possibilités à prendre en compte.

Je remercie mon encadrant Monsieur Christophe Lenté pour avoir proposé le sujet mais aussi pour m'avoir suivi et conseillé tout au long des phases de recherche et de développement.

Annexes

A Environnement de travail

1 Installation de l'environnement de travail

Dans cette section seront développés les différents éléments nécessaires pour pouvoir compiler et exécuter le code du projet. Je propose également une procédure d'installation étape par étape pour pouvoir reproduire l'environnement. Cette procédure est valable pour les systèmes d'exploitation Windows 7 64 bits [1].
Pour pouvoir travailler sur ce projet, je propose une distribution Python nommée *Anaconda3*(version 2.3.0) ainsi qu'un environnement de développement (IDE) Python, *PyCharm* en édition Community (version 4.5.4). En ce qui concerne l'aspect simulation, le *package* Python étudié s'appelle *SimPy* (Simulation for Python) (version 3.0.8).

Les instructions qui suivent sont pour l'installation des éléments ci-dessus. Il est tout à fait possible d'utiliser une autre distribution Python et un autre environnement de travail, ce qui est important est de *pouvoir utiliser le package SimPy*.

1.1 Distribution Python : Anaconda

Le but du projet est de créer des simulations et les simulations sont très liées à beaucoup de notions mathématiques telles que la génération de nombres aléatoires, les calculs statistiques, ... La distribution *Anaconda* en plus d'installer Python 3.4 propose un grand nombre de packages usuels pour représenter des notions issues des domaines *scientifiques*, *mathématiques*, de l'*ingénierie* et de l'*analyse de données*. Cependant, elle ne contient pas le package Simpy, il faudra l'installer après Anaconda [2].

L'installation de cette distribution permet de compiler du code rédigé en Python et d'avoir à sa disposition beaucoup de packages courant déjà installés.

Anaconda Nom : Anaconda3, Version : 2.3.0 pour Windows 64 bits, Version Python : 3.4, Site web : `https://www.continuum.io/`

1.2 Le package Simpy

Il existe deux manières pour installer ce package. Elles sont décrites sur le site de Simpy `https://simpy.readthedocs.org/en/latest/simpy_intro/installation.html`. Je vais "détailler" la plus pratique. Si

1. Ce système a été choisi en raison des ressources à ma disposition et de mon affinité avec cet OS (Operating System = Système d'exploitation)
2. WWW1.

vous posséder déjà une version de Python supérieure à la 2.7.9 (pour Python 2) ou à la 3.4 (pour Python 3) vous avez automatiquement un package *pip* installé avec Python. Il suffit alors d'ouvrir l'invite de commande et d'y recopier la commande suivante : *pip install simpy* (Attention, il faut disposer d'une connexion internet).

Ce package constitue l'objet d'étude de ce projet. Il sera détaillé dans ce rapport au Chapitre 5.

Simpy Nom : Simpy, Version : 3.0.8, Site web : https://simpy.readthedocs.org/en/latest/index.html

1.3 L'environnement de développement : PyCharm

Afin de simplifier l'écriture de code dans le projet, il peut être préférable d'utiliser un environnement de développement. PyCharm en est un spécialisé pour le code Python.

PyCharm Nom : PyCharm, Version : 4.5.4, Site web : https://www.jetbrains.com/pycharm/

1.4 Finalisation de l'installation

Après avoir effectué les trois étapes précédentes : installation d'Anaconda3, ajout du package Simpy et installation de l'environnement de travail, il reste une dernière étape. Lancez PyCharm pour lui indiquer l'emplacement de votre interpréteur Python. Une fois lancé, si PyCharm ne le propose pas directement, cliquez sur Fichier (ou File en anglais)
→ Paramètres (Settings)
→ Projet :<nom_ du_ projet> (Project :<project_ name>)
→ Interpréteur du projet (Project Interpreter)

et dans le champs Interpréteur du projet (Project Interpreter) indiquez l'emplacement de votre interpréteur Python. Il est possible d'utiliser le menu déroulant pour sélectionner "Ajouter Local" (Add Local) et le chemin proposé devrait être celui de Anaconda3. Il faut pointer sur l'exécutable *python.exe*.

Á la fin de cette sélection, la liste des packages disponibles s'affiche. Si l'ajout de Simpy a fonctionné, il doit apparaître dans la liste.
En cliquant sur "Ok", PyCharm prendra un peu de temps pour "relire" le code, si vous aviez ouvert un projet.

Comptes rendus hebdomadaires

Compte rendu n°1 du 17/09/2015

J'ai commencé par me documenter sur la notion de simulation pour en avoir une compréhension plus formelle. Pour cela, je me suis référé aux deux ouvrages que vous m'avez conseillé (Simulation Modeling and Analysis de Law et Computer Simulation Techniques de Naylor). J'ai pu comprendre le principe de la simulation en général et c'est avec vos premières explications que j'ai pu m'orienter sur la simulation orientée processus. De là, j'ai suivi votre conseil et j'ai cherché des cours sur ce type de simulations en ligne. Un cours de M.Richard M Fujimoto présente bien ce genre de simulation et fait une comparaison avec la simulation orientée événement. Ce cours m'a permis de mieux comprendre les notions que présentaient les livres étudiés. En cherchant d'autres cours, j'ai lu des choses sur le logiciel Arena qui permet de concevoir des simulations, et le logiciel SIMAN. Il y a des ouvrages à la bibliothèques sur ces logiciels et les premiers chapitres traitent de la simulation orientée processus en donnant d'autres définitions et exemples.

Avec toutes ces sources, je pense avoir une bonne compréhension de l'intérêt de la simulation ainsi que de ses avantages et de ses défauts.

En ce qui concerne la partie veille technologique, j'ai commencé à prendre en main le framework de python simpy3. Dans un premier temps j'ai reproduit les exemples du site puis j'ai modélisé un atelier très simple avec une arrivée de tâches (nombre et date connus à l'avance) et un poste de travail. La simulation est assez simple à implémenter et de manière générale, si il n'y a pas de nécessité pour une tâche de réquisitionner plusieurs ressources, l'implémentation d'une simulation reste simple. Suite à vos questions d'aujourd'hui, j'ai retravaillé mon implémentation pour incorporer la notion de priorité pour l'accès à une ressource et le suivi du nombre de retards dans la livraison des tâches. Ces deux nouveaux points fonctionnent sur un atelier simple.

Pour la prochaine séance, je vais continuer la documentation sur les simulations notamment sur la génération des entrées pour y inclure une part d'aléatoire suivant des lois de probabilités. Je vais aussi me perfectionner sur l'utilisation des "Containers" pour réfléchir au problème du foret qui s'use.

Compte rendu n°2 du 24/09/2015

J'ai continué la consultation des différents ouvrages de la bibliothèque pour avoir une vision plus complète des éléments qui interviennent dans une simulation. J'ai trouvé un livre (System Simulation de Geoffrey Gordon, côte 003 GOR) qui contient un chapitre intitulé "Industrial Dynamics". Ce chapitre propose une représentation d'une chaîne de production avec pour éléments des réservoirs reliés entre

eux par des "flux" plus ou moins rapide. L'explication est accompagnée d'un exemple assez pertinent. Cependant, je ne sais pas encore si cela représente bien la simulation orientée processus d'atelier de fabrication.

D'autre part, j'ai abordé la notion de loi pour les données d'entrées mais je ne sais pas comment les appliquer. J'ai réussi à introduire de l'aléatoire dans les arrivées mais cet aléatoire ne suit pas de loi explicite (j'utilise une fonction "random").

Finalement, j'ai essayé de mettre en place une "Ressource" un peu plus complexe que celles proposées par Simpy. Mon objectif était de représenter une machine pouvant tomber en panne qui possède un foret s'usant sur le temps. D'après la documentation de Simpy, la création de Ressources personnalisées est faisable mais la documentation est un peu légère sur ce sujet. Ce point de recherche est encore à travailler.

Compte rendu n°3 du 01/10/2015

J'ai implémenté les formules pour générer des suites de nombre aléatoires suivant des lois de probabilités. Je me suis basé sur les algorithmes proposés dans le livre de Law. Les lois "prêtes" sont : la loi de Poisson, la loi Uniforme et la loi Exponentielle (pour cette dernière, je ne sais pas quel paramètre prendre pour la moyenne). J'ai regardé si la répartition respectait celle des lois et cela semble le cas pour les trois lois(avec un petit échantillons de nombre). Pour comparer, j'ai aussi utilisé une bibliothèque Python nommée Numpy qui propose une fonction "Poisson" pour générer des nombres suivant la loi du même nom. Je suis en train de la comparer avec l'algorithme de Law sur de plus grands échantillons de nombre.

Pour la semaine prochaine, je vais comparer les autres algorithmes avec les fonctions de la bibliothèque pour choisir la meilleur génération.

Compte rendu n°4 du 08/10/2015

Cette semaine, j'ai installé un environnement de travail sur le poste de la bibliothèque sur lequel je travaille. J'en ai profité pour écrire une procédure d'installation qui permet de compiler et d'exécuter du code Python avec utilisation de fonctions de Simpy. J'ai également ré-organisé le code qui devenait trop complexe pour rester lisible dans un seul fichier. J'ai poursuivi mon étude sur la génération de nombres aléatoires et plus particulièrement, j'ai comparé un algorithme de génération proposé dans le livre Simulation Modeling and analysis (de M.Law) avec une fonction proposée par le package de Python nommé numpy. De cette étude, j'ai extrait des résultats qui ne permettent pas, tel quel, de départager les deux générations. Pour tout de même essayer de les classer, je vais appliquer un test du Khi² (peut être complété par un autre test).

Compte rendu n°5 du 15/10/2015

J'ai poursuivi la comparaison des deux méthodes de génération de données aléatoires pour la loi de Poisson. J'ai utilisé le test du Khi² dans un premier temps et après quelques recherches, j'ai découvert le test de Kolmogolov qui permet également de savoir si un échantillon suit une loi donné.

Pour cette séance, j'ai également avancé dans la rédaction du rapport. J'ai écrit une bonne partie de l'introduction sur la simulation.

Pour la prochaine séance, je finalise la comparaison des deux méthodes de générations aléatoires et je complète le rapport pour y intégrer le résultat de celle-ci.

Compte rendu n°6 du 22/10/2015

J'ai essentiellement travaillé le rapport, notamment pour réorganiser la présentation des concepts de simulation et des systèmes. J'ai ajouté une section qui explique pourquoi on utilise la simulation dans le cadre des ateliers de simulations au lieu d'employer une autre méthode d'étude de système. J'ai récupéré la version d'essai d'Arena mais je n'ai pas pu l'essayer, la réunion ayant légèrement débordée sur le temps associé à un plantage de la machine virtuelle.

Pour la semaine de pause pédagogique, j'ai pour objectif de relire le rapport pour le mettre en forme et corriger les éventuelles fautes. Pour la reprise, j'envisage de finir l'étude de l'aléatoire et d'essayer Arena pour le comparer avec ce que propose SimPy.

Compte rendu n°7 du 05/11/2015

Pour cette séance, j'ai continué le rapport du projet. J'ai défini la notion d'Ateliers de fabrication mais il faut que je la complète avec le problème de la topologie. J'ai repris les caractéristiques des systèmes d'une simulation (statique/dynamique, orienté processus/événements, ...) pour en clarifier et harmoniser les explications puis j'ai déterminé les caractéristiques qui sont intéressantes dans le cadre du projet.

Pour la prochaine séance, je vais implémenter un exemple d'Atelier de fabrication simple en Simpy.

Compte rendu n°8 du 12/11/2015

Lors de cette séance, j'ai créé une simulation simple avec Simpy tout en essayant de faciliter la tâche pour les prochaines améliorations (surveillance des compteurs de la simulation, ajout de plusieurs machines, de plusieurs processus différents, ...) . Au final, la simulation fonctionne mais l'affichage de l'évolution de celle-ci est à retravailler car il va devenir très vite illisible lorsque l'on va augmenter le nombre de tâches.

J'ai eu un problème de machine virtuelle qui ne pouvait plus redémarrer, j'ai néanmoins pu le corriger assez rapidement.

Pour la prochaine séance, je vais compléter et complexifier le programme. Je vais aussi réfléchir à l'algorithme sur lequel se base la simulation orientée processus.

Compte rendu n°9 du 22/11/2015

Pour cette séance, je me suis concentré sur la rédaction de l'algorithme d'une simulation orientée processus. Je suis arrivé à un résultat qui semble fonctionner mais une question majeure m'empêche de finaliser l'algorithme. Voici le problème : Lorsque l'on choisit le prochain processus à exécuter dans la liste des processus, il est possible que plusieurs processus doivent s'exécuter dans le même pas de la simulation. Il faut les choisir un par un et chacun peut modifier les variables de la simulation. Ceci permet de gérer le cas où plusieurs processus souhaitent accéder à une même ressource en même temps (seul le premier de la liste des processus aura l'accès). Cependant, si on considère : - P1 processus souhaitant utiliser la ressource R - P2 processus possédant la ressource qui est sur le point de libérer R. Si la liste des processus est la suivante : (P2, P1) -> P2 va relâcher la ressource et P1 pourra la récupérer.

Mais si la liste est (P1, P2) -> P1 essaye de récupérer la ressource mais elle n'est pas encore disponible, P1 est reprogrammé pour le pas de simulation suivant (ce qui peut entraîner des problèmes dans le système).

Pour résumer le problème, l'ordre d'exécution de processus qui sont programmés sur le même pas peut entraîner des comportements non fidèles au système.

En réfléchissant à ce problème, j'ai trouvé deux solutions intéressantes : - On peut tenter d'affecter une priorité différente au processus suivant les instructions qu'ils auront à réaliser. Ainsi, un processus devant libérer une ressource aura une très grande priorité mais si plusieurs processus doivent libérer des ressources différentes pour en utiliser d'autres, il peut être compliqué d'évaluer les priorités. - Il peut être possible de boucler sur les processus à exécuter tant que l'un d'eux arrive à s'exécuter. Si aucun ne peut s'exécuter, ils sont alors tous reprogrammés au prochain pas.

Je n'ai pas encore déterminé si Simpy se préoccupe de gérer ce problème.

Dans cette séance, j'ai également commencé à proposer des algorithmes pour les différentes configurations de systèmes que vous avez proposé.

Pour la prochaine séance, je regarderai le comportement de Simpy plus en détail et je continuerai la recherche d'algorithmes.

Compte rendu n°10 du 10/12/2015

Je complète le rapport au fur et à mesure, je suis en train d'expliquer la démarche à suivre pour réaliser une simulation de A à Z. En parallèle, je réalise la modélisation d'ateliers de fabrication de plus en plus complexe en essayant de suivre les étapes que j'énonce dans le rapport pour veillez à la cohérence et la lisibilité du rapport.

Pour les ateliers "linéaires" (avec un seul chemin), la modélisation se réalise sans difficulté particulière, même avec la contrainte de stocks finis.

Pour l'exemple d'un 'désassemblage' (une machine effectue un traitement et le résultat est utilisé dans deux traitements parallèles), il y a une difficulté. Si on considère qu'après le premier traitement les deux autres traitements sont à exécuter dans le même processus et que les stocks entre le premier traitement et les autres sont finis : Lorsqu'un stock est plein, le premier traitement se bloque puisqu'un deux morceaux du résultat ne peux rejoindre son stock. Le problème est qu'il ne faut pas interrompre le processus puisque l'on doit tout de même traiter le morceau qui a su rejoindre son stock. Il faudrait donc que le processus génère un processus à part pour traiter ce morceau. En utilisant cette méthode, il n'y a pas trop de problème si on a deux machines en parallèles mais si on en met plus, avec la définition des processus que j'ai choisi cela semble se compliquer très vite.

Pour la prochaine séance, je continue de compléter le rapport et je vais réfléchir à ce dernier problème en essayant de changer la définition des processus ou peut être en divisant le processus en deux autres.

Compte rendu n°11 du 17/12/2015

Je continue la modélisation des différents cas d'ateliers de fabrication. Pour le moment, j'ai réussi à modéliser les cas linéaires tout en respectant l'approche orientée processus. Cependant, j'ai toujours des difficultés pour les cas avec des traitements en parallèle. J'essaye d'exploiter plusieurs pistes notamment parcourir la liste des traitements à effectuer pour réaliser le besoin voulu mais en sens inverse pour permettre d'utiliser les "matériaux" potentiels résultant d'anciens processus exécutés (par exemple, une machine M1 prend 1 élément en entrée et en génère 2 en sortie pour la machine M2 qui elle ne prend qu'un seul élément en entrée). L'approche est prometteuse, elle permet de garder une bonne cohérence sur la notion de processus mais le parcours des traitements se compliquent toujours lorsqu'ils sont parallèles. Si cette méthode n'aboutit pas, je pense simplifier la gestion des processus parallèles qui résultera en une baisse de la sémantique de "processus" dans la simulation.

Pendant la période de pause pédagogique, je finirai le rapport de la partie recherche ainsi que l'affiche type "poster".

Compte rendu n°12 du 14/01/2016

Pour cette séance, j'ai en parti répété ma soutenance pour ensuite la réaliser. Dans le temps qu'il me restait, j'ai fini la mise au point des algorithmes des cas de base et je les ai ajoutés au rapport. J'ai résolu le problème de traitements simultanés en permettant la création de "sous processus" dont l'évolution est surveillée par le processus principal. Ainsi, lorsque tous les processus "fils" sont terminés, le processus principal se réveille et enchaîne avec les traitements suivant.

Pour la prochaine séance, je vais commencer l'implémentation des cas de base. Normalement, les cas sans simultanéités sont censés être facile à implémenter. C'est donc sur la structure et l'organisation du code que je vais particulièrement m'attarder.

Compte rendu n°13 du 21/01/2016

J'ai implémenté le cas de base ainsi que le cas de deux machines en série en utilisant la bibliothèque Simpy. Ces deux premières simulations fonctionnent correctement. Simpy permet de facilement "attendre" qu'une condition se réalise et on peut écrire l'algorithme d'un processus pour ensuite le soumettre à l'ordonnanceur qui se charge de l'interrompre (pour attendre un temps de traitement d'une machine par exemple) et de reprendre l'exécution.

J'ai commencé à traiter le cas du désassemblage. Il me reste un problème à régler, celui de la taille finie des files d'attentes. Simpy ne propose pas de limite aux files d'attentes à une ressource. Je suis donc en train de chercher un moyen pour connaître le moment où un processus accède à une ressource (ou la libère) pour pouvoir autoriser d'autres processus à entrer dans la file d'attente.

Pour vérifier que les simulations fonctionnent, j'affiche sur un graphique l'utilisation des ressources sur le temps. Il y a 1 courbe par graphique, en abscisse il y a le temps de la simulation et en ordonnée l'id du processus qui utilise la ressource. Cela fonctionne assez bien sur un petit nombre de tâches mais cela devient vite illisible pour plus d'une quinzaine de tâches. Je reste attentif à de possible représentation graphique des résultats.

Compte rendu n°14 du 31/01/2016

Pendant la séance de cette semaine, j'ai continué l'implémentation du cas de désassemblage. La bibliothèque Simpy permet beaucoup de simplification du code mais j'ai encore quelques difficultés avec l'acquisition des ressources. Cependant, j'ai presque résolu ce problème. J'ai repris mon rapport pour corriger les fautes de français et ajuster certaines tournures de phrases.

Pour la prochaine fois, je pense pouvoir terminer l'algorithme du cas de désassemblage et enchaîner sur le cas d'assemblage.

Compte rendu n°15 du 11/02/2016

J'ai exploré l'approche orientée graphe du problème et j'ai réussi à en faire une implémentation fonctionnelle. Les possibilités sont encore assez limités mais on peut simuler des ateliers possédant une entrée et une sortie (je n'ai pas encore fait de test avec plusieurs entrée/sorties). Les machines ne possèdent pas de file d'attente pour le moment mais elles peuvent avoir des capacités différentes. On peut soumettre plusieurs processus identiques à la fois (j'ai essayé avec 50 processus programmés à l'instant t=0) et le résultat obtenu est cohérent.

La création d'atelier est assez simple. On crée nos ressources, on crée un graphe ayant pour nœud les machines(ressources) de notre atelier et pour arc le cheminement à effectuer entre les machines. Une fois le graphe créé, on le passe en paramètre à un "constructeur de processus complet" qui met en place toutes les relations entre les machines et qui définit la machine de sortie(la dernière étape). Ensuite, il suffit de soumettre ce processus à l'environnement de la simulation et il sera exécuté.

Le résultat du programme est un ensemble de graphiques (courbe) représentant l'utilisation des machines (nombre d'utilisateurs) à chaque instant de la simulation.

Pour la prochaine fois, je vais essayer de mettre en place les files d'attente et instaurer des tests pour vérifier le bon fonctionnement de la simulation.

Compte rendu n°16 du 25/02/2016

L'intégration des files d'attentes a fonctionné. Désormais, on peut simuler des ateliers de fabrications composés de plusieurs machines en séries ou en parallèles avec ou sans file d'attente. J'ai fait le choix de considérer les files d'attentes comme des nœuds à part du graphe représentant l'atelier. De cette manière, on peut représenter plus facilement son atelier et en plus, on réutilise des comportements déjà implémentés. L'autre choix consistait à passer un objet file d'attente à la construction d'une machine mais cela devenait compliqué lorsqu'il y avait plusieurs files d'attentes pour la même machine (cas d'assemblage).

L'ajout d'autre élément (espace de stockage par exemple) sera possible. Il suffit de se baser sur les éléments déjà existant. Pour ajouter un nouvel élément, il faut deux choses : - hériter de la classe WorkshopElement, qui va définir les attributs et partager les fonctions de base. On peut ensuite ajouter de nouveaux attributs si l'on souhaite - définir en quoi consiste le job (tâche) de cet élément et implémenter la fonction pour créer ce job (certains on besoin de plus d'informations que d'autres). Pour cela, on hérite d'abord de la classe BaseJob et on implémente la fonction do_the_job.

Pour mieux guider d'éventuels ajouts, j'ai utilisé les mécanismes propices d'implémentation(héritage, méthodes abstraites, ...).

En ce qui concerne la sortie de la simulation, pour le moment on a accès à l'évolution de l'occupation des files d'attentes dans une première fenêtre et dans une seconde, il y a pour chaque machine la courbe de l'évolution de l'occupation et celle de l'utilisation. L'occupation détermine si quelque chose est physiquement sur la machine et prend donc 1 place de capacité. L'utilisation concerne les moments où la machine effectue réellement un traitement. Toutes ces données sont ensuite écrites dans des fichiers CSV (1 fichier pour les files d'attente, 1 pour l'occupation des machines et 1 pour l'utilisation des machines).

Avec un petit peu d'entraînement et peu de processus, il est possible de déterminer quelle(s) machine(s) sont responsables d'un goulot d'étranglement.

Pour la prochaine fois, je vais me concentrer sur l'écriture de test pour assurer du bon fonctionnement de l'application et aider à sécuriser les prochains développement. Je vais essayer de mettre en place le suivi du nombre de processus en cours de traitement et ceux en retard. Enfin, je vais continuer la rédaction du rapport et de la documentation du projet.

Compte rendu n°17 du 03/03/2016

Pendant cette séance, j'ai amélioré la récupération des résultats de la simulation. Désormais, on a accès pour les machines : - au taux d'occupation par machine sur la durée où la machine est utilisée (entre le début de la première tâche qui arrive sur la machine et la fin de la dernière) - au taux d'usage(durée pendant laquelle la machine effectue réellement un traitement) par machine sur la durée où la machine est utilisée

pour les files d'attentes, on a accès au : - taux d'occupation par file sur toute la durée de la simulation - taux d'occupation par file sur la durée où la file est utilisée (entre le début de la première tâche qui entre dans la file et la fin de la dernière)

Bien sûr, on peut choisir les valeurs à afficher (par exemple on peut afficher les taux sur toutes la durée de la simulation pour les machines).

J'ai également commencé à compléter la documentation. Pour réaliser une documentation plus agréable à parcourir, j'ai mis en place Sphinx qui permet de générer de la documentation rapidement sous forme de plusieurs page web (qui sont en local, pas besoin de connexion internet ni de serveur sur la machine pour les consulter).

Pour la prochaine séance, je vais finir la documentation et créer des tests pour vérifier l'implémentation du projet et faciliter la reprise de celui-ci.

Compte rendu n°18 du 10/03/2016

J'ai fini la documentation du code du projet. Je peux me concentrer sur l'implémentation de tests. J'ai déjà crée plusieurs tests unitaires, je dois maintenant écrire des tests pour tester un ensemble de fonctions. Ceci sera fait en utilisant la simulation d'un petit atelier dont on connaît les résultats. Si à un moment du projet ce test ne passe pas, c'est qu'il y a un problème au niveau de l'implémentation. Cela permettra de pouvoir faire évoluer le projet en étant assuré que l'implémentation respecte les contraintes de base.

Une fois les tests finis, je vais réfléchir sur les possibles évolutions du projet et notamment sur les points suivants : - est-ce que le projet peut intégrer la notion de panne de ressources - est-ce que le projet peut prendre en compte le fait que certaines ressources nécessitent un opérateur pour fonctionner - est-ce que l'on peut simuler un temps de transport entre les différents éléments de l'atelier

et également réfléchir sur la possibilité de réutiliser le projet pour un autre problème que la simulation d'atelier de fabrication (exemple : simuler un réseaux de bus dans une ville).

Compte rendu n°19 du 17/03/2016

J'ai trouvé d'où venait le problème de l'aléatoire. Je n'utilisais pas la bonne loi pour générer les nombres. J'utilisais une loi de poisson avec des paramètres très faible (entre 0 et 2), les tirages de nombre d'une loi de poisson étant entier, l'aléatoire ne semblait pas fonctionner correctement. La raison pour laquelle cela a fonctionné un moment était dû au fait que j'avais multiplié les paramètres des lois par 100. Donc l'aléatoire donnait un résultat intéressant (visible). J'ai remplacé ces lois de poisson par des lois exponentielles(arbitrairement) qui représentent mieux l'aléatoire.

Pendant cette séance, j'ai fini l'écriture des tests pour le projet et recompilé la documentation pour la mettre à jour. Le reste de la séance fût consacré à la rédaction du rapport.

Pour la prochaine séance, l'objectif est de constituer le matériel pour la soutenance.

Webographie

[WWW1] Inc. Continuum Analytics. *Continuum Analytics*. URL : https://www.continuum.io/ (visité le 21/10/2015).

[WWW2] Inc.Richard M Fujimoto Continuum Analytics. *Discrete Event Simulation - Process Oriented Simulation*. URL : http://www.acm-sigsim-mskr.org/Courseware/Fujimoto/Slides/FujimotoSlides-02-EventOrientedSimulation.pdf (visité le 05/11/2015).

[WWW3] Inc.Richard M Fujimoto Continuum Analytics. *Discrete Event Simulation - Process Oriented Simulation*. URL : http://www.acm-sigsim-mskr.org/Courseware/Fujimoto/Slides/FujimotoSlides-04-ProcessOrientedSimulation.pdf (visité le 05/11/2015).

[WWW4] *Math is fun*. URL : https://www.mathsisfun.com/algebra/mathematical-models.html (visité le 22/10/2015).

[WWW5] TradeMark Media. *Site officiel d'Arena*. URL : https://www.arenasimulation.com/ (visité le 05/01/2016).

[WWW6] *Nasa : National Aeronautics and Space Administration*. URL : http://www.nasa.gov/centers/marshall/history/gallery/msfc_iow_7.html#.Vii2LX7hCM8 (visité le 22/10/2015).

[WWW7] Team SimPy. *Site de documentation de SimPy*. URL : https://simpy.readthedocs.org/en/latest/ (visité le 05/01/2016).

[WWW8] NetworkX developer team. *Site de documentation de networkx*. URL : https://networkx.github.io/ (visité le 17/03/2016).

[WWW9] *The Python Wiki*. URL : https://wiki.python.org/ (visité le 22/03/2016).

[WWW10] *Wikipédia Mathematical Model*. version française disponible. URL : https://en.wikipedia.org/wiki/Mathematical_model (visité le 22/10/2015).

Bibliographie

[1] Geoffrey Gordon. *System Simulation*. 1969, p. 1. isbn : 0138817979.

[2] W David Kelton, Randall P Sadowski et David T Sturrock. *Simulation With Arena*. Third Edition. 2004. isbn : 0072856947.

[3] Averill M. Law. *Simulation Modeling And Analysis*. Fifth Edition. 2013, p. 2–3.

[4] Thomas H Naylor, Joseph L Balintfy, Donal S Burdick et Kong Chu. *Computer Simulation Techniques*. Third Edition. 1968. isbn : 471630608.

[5] C Dennis Pegden, Robert E Shannon et Randall P Sadowski. *Introduction to Simulation Using SIMAN*. Second Edition. 1990, p. 12–13. isbn : 0073401323.

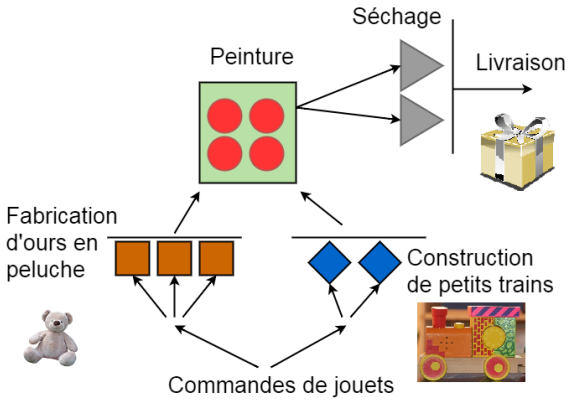
Simulation d'ateliers de fabrication : Introduction au principe de la simulation orientée processus

Florian Montalbano

Encadrement : Christophe Lenté

Présentation du problème

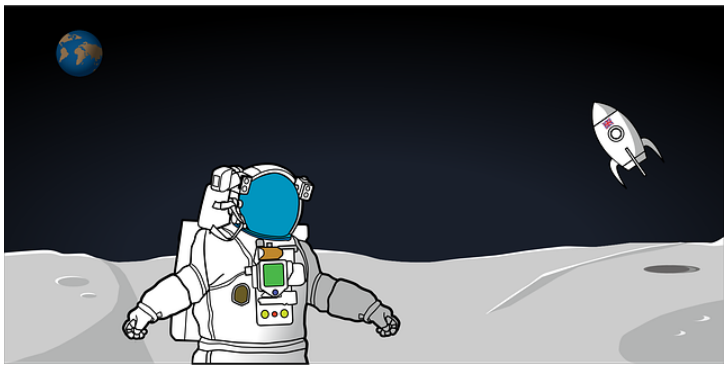
Vous voilà chef de votre propre atelier de fabrication !
Mais il y a trop de commandes, la production n'arrive pas à suivre.
Vous devez choisir quelle forme ajouter pour augmenter au maximum votre production afin de combler le plus de commandes possibles !
Quelle forme choisiriez-vous ?



Votre usine de fabrication

Une Solution : Simulation

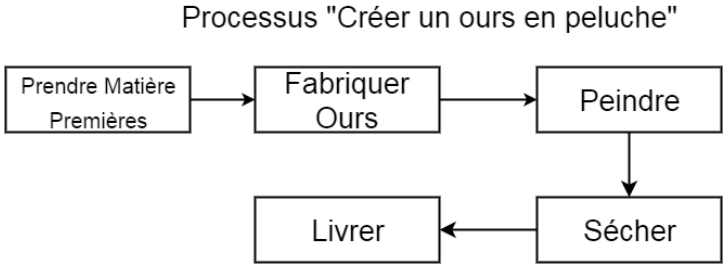
Ce genre de problème peut être résolu en réalisant des simulations.
Les simulations permettent de faire plein de tests pour trouver la meilleure solution et en plus, sans déranger le système actuel.
Cependant, concevoir une simulation n'est pas si simple ...



L'Homme a pu marcher sur la Lune après la réalisation d'un grand nombre de simulations très complexes !

Simplifier la construction

La représentation de l'usine ci-dessus semble intuitive.
Le but de ce projet est de "préparer" un certain nombre de blocs élémentaires de simulation pour rendre plus intuitif la construction de simulations d'ateliers de fabrication.



Si on remplace le mot "ours" par "petit train", on obtient le processus de création des petits trains

Ecole Polytechnique de l'Université François Rabelais de Tours
Département Informatique
64 avenue Jean Portalis, 37200 Tours, France
www.polytech.univ-tours.fr

POLYTECH TOURS Informatique

Simulation d'ateliers de fabrication

Introduction au principe de la simulation orientée processus

Résumé

Comment améliorer le rendement d'un atelier de fabrication ? Faut-il investir dans une machine supplémentaire ? Si oui, laquelle ? Qu'est-ce que je vais gagner en effectuant ce changement ?
Pour répondre à ces questions, on peut essayer d'acheter un exemplaire de chaque machine et d'essayer pendant une certaine période de temps de constater les changements. Si on a du temps et des moyens, cette solution fonctionnera. Cependant, si aucun des changements ne se révèle efficace ... c'est du gâchis de temps et de moyen. C'est pour éviter ce type de déception qu'on peut être amené à utiliser un outil très puissant mais souvent difficile à mettre en place : *la simulation*. Avec cet outil, on effectuer autant de tests que l'on souhaite sans avoir ni à déranger le système en place (si il existe) ni à investir dans du matériel qui s'avèrera peut être inefficace et ni à attendre. Ce rapport propose une introduction à la simulation orientée processus et une recherche sur la possibilité de faciliter la création de simulation d'ateliers de fabrications à partir d'une bibliothèque python nommée SimPy.

Mots-clés

Simulation orientée processus, SimPy, Ateliers de fabrications

Abstract

How can one proceed to improve my manufacturing workshop's efficiency ? Should one invest in an additional machine ? If yes, which one ? What will one gain from this modification ?
In order to answer these questions, one can buy one exemplar of each machine type he owns, try each of these new machine and wait for results. With time and resources, one can have his answer, but should it that none of these did even a little improvement... it's clearly a waste of means and time. A best way to answer these questions without neither having to disturb the actual system (if it already exists) nor investing for potentially inefficient machine nor having to wait is to use *simulation*. This work introduce the world of process oriented simulation and a study on the feasibility of easing the creation of simulation for manufacturing workshop from a python librairy named SimPy.

Keywords

word, key, two words, fourth word

Tuteurs académiques
Christophe Lenté

Étudiants
Florian Montalbano (DI5)