

Intégration de l'humain dans le pilotage des unités opérationnelles

**- Un Système d'Assistance Interactif générique -
(titre provisoire)**

MANUSCRIT DE THESE

présenté en vue d'obtenir

UN ACCORD POUR SOUTENANCE

Discipline : Informatique

par

GRANDGIRARD Emilie

SOMMAIRE

| | |
|---|----|
| Introduction | 1 |
| 1. L'homme au cœur des nouveaux systèmes de production..... | 5 |
| 1.1 Les principaux paradigmes de production | 6 |
| 1.1.1 La production de masse..... | 6 |
| 1.1.2 La production à flux tirés et maigre (le « lean »)..... | 8 |
| 1.1.3 La production agile..... | 10 |
| 1.2 L'apport des nouvelles technologies pour les systèmes de production modernes | 12 |
| 1.2.1 Le pilotage du système technique | 12 |
| 1.2.2 Le problème des opérateurs..... | 13 |
| 1.2.3 Vers un système de pilotage hybride..... | 16 |
| 2. Les systèmes de pilotage | 19 |
| 2.1 La structure d'un système pilotage..... | 19 |
| 2.2 Typologie du pilotage..... | 21 |
| 2.2.1 Le pilotage prédictif | 22 |
| 2.2.2 Le pilotage proactif | 22 |
| 2.2.3 Le pilotage réactif..... | 23 |
| 2.3 Les architectures de pilotage | 24 |
| 2.3.1 L'architecture de pilotage centralisée | 25 |
| 2.3.2 L'architecture de pilotage hiérarchique..... | 25 |
| 2.3.3 L'architecture de pilotage hétérarchique..... | 26 |
| 2.3.4 L'architecture de pilotage hybride | 27 |
| 2.4 Un système de pilotage adapté à l'interactivité..... | 28 |
| 3. Etude fonctionnelle du système..... | 30 |
| 3.1 Système de pilotage ou système d'aide à la décision ? | 30 |
| 3.2 Une architecture générique pour le SAI..... | 31 |
| 3.2.1 Typologie des décisions | 31 |
| 3.2.2 Cartographie des décisions d'ordre purement opérationnel..... | 32 |
| 3.2.3 Les composants fonctionnels du SAI..... | 35 |
| 3.3 Modélisation du SAI | 39 |
| 3.3.1 La modélisation des systèmes de production | 39 |
| 3.3.2 Une représentation UML du SAI | 40 |
| 3.3.3 De UML à la simulation..... | 46 |
| 4. Simulation du système | 50 |
| 4.1 De la modélisation à la simulation | 50 |
| 4.2 Les concepts de la simulation..... | 51 |
| 4.2.1 Les modèles à files d'attente | 52 |
| 4.2.2 Le processus de simulation | 53 |
| 4.2.3 La simulation de systèmes distribués..... | 55 |
| 4.3 Une méthodologie de simulation pour les systèmes de pilotage réactifs distribués | 56 |
| 4.3.1 Le logiciel de simulation Arena | 56 |
| 4.3.2 Le modèle générique de simulation | 57 |

| | | |
|-------|--|-----|
| 4.4 | Application : création du modèle de simulation du SAI correspondant à une unité industrielle existante | 64 |
| 4.4.1 | Présentation de l'unité | 64 |
| 4.4.2 | Le SAI associé à l'unité de réglages moteurs | 66 |
| 4.4.3 | Simulation du SAI couplé à l'unité de réglages moteurs | 68 |
| 4.5 | Apports du travail de simulation | 71 |
| 5. | Développement du système..... | 74 |
| 5.1 | Architecture de communication du SAI..... | 74 |
| 5.1.1 | Le modèle Client/Serveur..... | 75 |
| 5.1.2 | Les modes de communication Client/Serveur..... | 76 |
| 5.1.3 | Le middleware..... | 77 |
| 5.1.4 | Quelle architecture de communication pour le SAI ? | 80 |
| 5.1.5 | Application de l'architecture du Web au SAI | 82 |
| 5.2 | Analyse et programmation des composants | 85 |
| 5.2.1 | Les composants du Contrôleur Central | 85 |
| 5.2.2 | Les clients Contrôleurs de Ressources | 90 |
| 5.2.3 | Vers des Contrôleurs de Ressources clients et serveurs..... | 93 |
| 5.3 | Un exemple d'interactivité | 95 |
| 5.4 | Un environnement de test pour le SAI..... | 96 |
| 6. | Implantation du système..... | 100 |
| 6.1 | Champs d'application du SAI | 100 |
| 6.2 | Démarche d'implantation du SAI | 101 |
| 6.3 | Adaptation du SAI pour une unité hospitalière | 102 |
| 6.3.1 | Le SAI face à la problématique des hôpitaux | 102 |
| 6.3.2 | Présentation de l'unité d'endoscopie..... | 103 |
| 6.3.3 | Identification des Contrôleurs de Ressources | 105 |
| 6.3.4 | Identification des échanges entre l'unité et le SAI | 108 |
| 7. | La notion de confiance | 112 |
| 7.1 | Les résultats d'utilisation du SAI en conditions optimales | 112 |
| 7.2 | Les problèmes de confiance | 113 |
| 7.2.1 | Non réalisation d'opérations (cas A)..... | 114 |
| 7.2.2 | Non respect des opérations (cas B) | 115 |
| 7.3 | Un moteur de confiance pour le SAI..... | 116 |
| 7.3.1 | Les moteurs de confiance | 117 |
| 7.3.2 | Application des concepts des moteurs de confiance au SAI..... | 119 |
| 7.3.3 | Application à l'unité de réglages moteurs | 121 |
| | Conclusion..... | 124 |
| | Références bibliographiques | 128 |
| | Annexe 1. Le modèle de simulation..... | 136 |
| | Annexe 2. Scripts | 140 |
| | Annexe 3. Interactivité | 169 |
| | Liste des figures et des tableaux..... | 172 |

Introduction

Des études antérieures menées par Gertosio et Dussauchoy sur une unité industrielle (unité de réglages pour les moteurs diesel chez RVI) ont conduit à identifier un problème d'efficacité au niveau du pilotage des unités opérationnelles caractérisées par des processus fortement automatisés et un environnement entièrement dédié au bon fonctionnement des machines prenant en charge le processus [1]. En effet, une étude de simulation a montré qu'il existait une grande différence entre les résultats optimaux donnés par la simulation et les résultats réels. Ce décalage serait dû à un problème d'efficacité au niveau du pilotage des opérations annexes entourant le processus automatisé en lui-même, c'est-à-dire des opérations où l'humain est amené à intervenir telles que les tâches de manutention.

Le problème soulevé est intéressant dans la mesure où il est révélateur des problèmes d'organisation dans certaines unités à forte composante humaine. Or, avec les besoins croissants en flexibilité, l'homme devient aujourd'hui un acteur fort des systèmes de production. En fait, l'évolution constante du marché pousse les entreprises à offrir toujours plus de produits innovants, bons marchés et personnalisés, et donc à adopter des structures plus à même d'intégrer les changements. La production de masse, avec ses lignes standardisées, n'est plus adaptée au marché : les entreprises doivent augmenter la flexibilité de leur système de production en suivant, par exemple, les préceptes de la production agile. Ainsi, les systèmes de production modernes revalorisent les compétences et les connaissances de l'homme qui, de par ses capacités adaptatives, est vu comme le garant de la fiabilité, de la flexibilité et de la réactivité.

Paradoxalement, l'opérateur humain est également une des causes principales d'insécurité. Les incertitudes liées à son comportement et à sa méconnaissance du système de production peuvent réduire la performance globale d'une unité opérationnelle. Il est amené à cohabiter et à interagir avec un système complexe, souvent autogéré, et à synchroniser au mieux ses opérations avec l'ensemble des opérations effectuées, tout en répondant aux différents aléas de production. En réalité, un système de production peut être considéré, dans sa forme la plus simple, comme la rencontre de trois populations [2]: la population des produits, la population

des moyens de production et la population des opérateurs de production. La performance globale dépend de l'interaction entre ces trois populations, mais les systèmes de pilotage actuels font peu cas de la population des opérateurs. Longtemps considérée comme secondaire, la population des opérateurs prend cependant de plus en plus d'importance et s'avère, dans le même temps, la plus difficile à contrôler. C'est ainsi qu'à ce jour il paraît nécessaire d'étudier et de privilégier les interactions de l'opérateur avec le système afin d'exploiter au mieux ses capacités.

L'émergence de systèmes orientés humains répond en partie à cette nouvelle donne. Il s'agit de proposer, par le biais des nouvelles technologies, une palette de solutions pour intégrer complètement l'homme au sein du système de production en lui fournissant les informations nécessaires pour répondre au mieux aux exigences et aux aléas de la production. Au niveau opérationnel, cela se traduit plus concrètement par des outils d'aide à la réalisation de produits, au suivi de la production, au diagnostic et à la gestion des pannes...

Dans ce contexte, nous nous sommes intéressés à l'aide que les technologies de l'information pourraient apporter aux opérateurs des systèmes de production modernes. Le but est de les assister dans leurs décisions en leur procurant une information adaptée en temps réel. En effet, les objectifs de production engendrent une orchestration complexe d'opérations basée sur une quantité d'informations trop accablante pour le cerveau humain et, en définitive, peu de solutions réelles existent pour aider l'opérateur dans ses choix d'opération à réaliser. Nous souhaitons alors indiquer à tout moment aux opérateurs les opérations les plus prioritaires du point de vue du pilotage de la production et, considérant que toute action a un impact sur le déroulement de la production, prendre en compte leurs décisions finales pour revoir éventuellement ce pilotage. L'idée sous-jacente est de construire, en se basant sur l'interactivité, un système de pilotage qui réagit aux décisions des opérateurs. Un tel système permettrait de concilier un flux optimal de production et les contraintes humaines (indisponibilité, proximité de l'action, interventions de maintenance...). Cette étude fait suite aux travaux de Gertosio et Dussauchoy.

L'objet de cette étude est de définir un système générique tant au niveau de son architecture fonctionnelle qu'au niveau de son architecture de communication afin de permettre sa réutilisation sur toute unité, non nécessairement industrielle, rencontrant ce type de problème. Il convient alors d'identifier une architecture modulaire autorisant l'intégration des ressources hétérogènes d'une unité et facilitant en conséquence l'adaptation du système à une unité

quelconque. La modularité, l'hétérogénéité et la réutilisabilité sont en fait des concepts assez récents dans le pilotage de la production. Ces contraintes signifient qu'il faut se rapprocher, pour le développement du système, de la notion de système ouvert dans lequel les composants sont interopérables et aisément interchangeables [3].

Cette étude aboutit à la réalisation d'un prototype capable de communiquer avec une unité préexistante. Nous déplorons n'avoir pu l'expérimenter dans des conditions réels. Cependant, nous procédons, tout au long de nos travaux, à différents tests qui mettent en évidence les forces et les faiblesses d'un tel système.

Cette étude touche à des domaines aussi variés que le comportement humain (sciences cognitives), les automatismes, la gestion de production, les communications réseau... Dans ce mémoire, nous tacherons de ne pas nous égarer dans les multiples problématiques afférentes à ces vastes champs de recherche, mais à faire ressortir les concepts essentiels nécessaires à la compréhension de la problématique, ceci afin de se concentrer sur le cœur du sujet : le développement d'un Système d'Assistance Interactif (SAI) générique. Le mémoire est donc présenté de façon chronologique, de la présentation du problème à l'application du système obtenu à travers notre étude en passant par la démarche de conception.

Le **chapitre 1** constitue un préambule à notre étude. Il retrace brièvement l'historique de l'homme dans les systèmes de production et en ressort les besoins en information. Il pose la problématique en nous permettant de délimiter le périmètre d'action de notre système et donc d'en dégager le cadre décisionnel.

Le SAI appartenant à la classe des systèmes de pilotage, le **chapitre 2** présente plus en détail ce type de système et les différentes possibilités offertes pour le pilotage de la production. Il nous permet de choisir la solution adéquate, adaptée à la nature interactive du SAI.

Le **chapitre 3** offre une vue fonctionnelle du SAI à travers sa modélisation. Il présente un découpage des décisions prises un opérateur humain et nous permet, en fonction de ce découpage, d'établir une structure modulaire et distribuée pour le SAI. Le **chapitre 4** complète le travail de modélisation en proposant une simulation du SAI. Nous proposons alors une méthodologie pour la simulation de systèmes distribués. Ces deux chapitres ont pour objectif majeur d'identifier les échanges de données nécessaires au fonctionnement du SAI.

Le **chapitre 5** explique le développement du SAI. Il repose sur le choix d'une architecture de communication autorisant les échanges de données identifiés et sur des choix de programmation autorisant la répartition des traitements.

Enfin, le **chapitre 6** présente plus précisément les applications possibles pour le SAI ainsi que les résultats obtenus. Il nous permet d'évaluer les performances du SAI et d'en extraire sa principale faiblesse, qui s'avère être aussi sa force majeure, à savoir son interactivité. Le SAI étant basé sur une pleine coopération des humains, il est important de se prémunir des problèmes de fiabilité en lui ajoutant par exemple des procédures de calcul de confiance.

1. L'homme au cœur des nouveaux systèmes de production

Taylor considérait que le travail de l'ouvrier ne consistait pas à penser, mais seulement à exécuter des tâches savamment calculées pour ses aptitudes [4]. Il marquait alors les débuts de l'organisation scientifique du travail en cloisonnant les individus dans des rôles bien spécifiques, concentrés autour des machines de production. Depuis, les compétences des opérateurs de production ont été revalorisées. Leur satisfaction apparaissant comme source de productivité [5][6], ils se sont vus confiés de plus en plus de responsabilités. En parallèle, la dynamique de groupe a été privilégiée [7] pour aboutir aujourd'hui à des organisations holistiques¹ basées sur une rotation des tâches favorisant l'apprentissage et la polyvalence [8]. L'opérateur est ainsi devenu un élément clé des systèmes de production. Oborski retrace sa progression au sein des unités industrielles [9] (Figure 1).

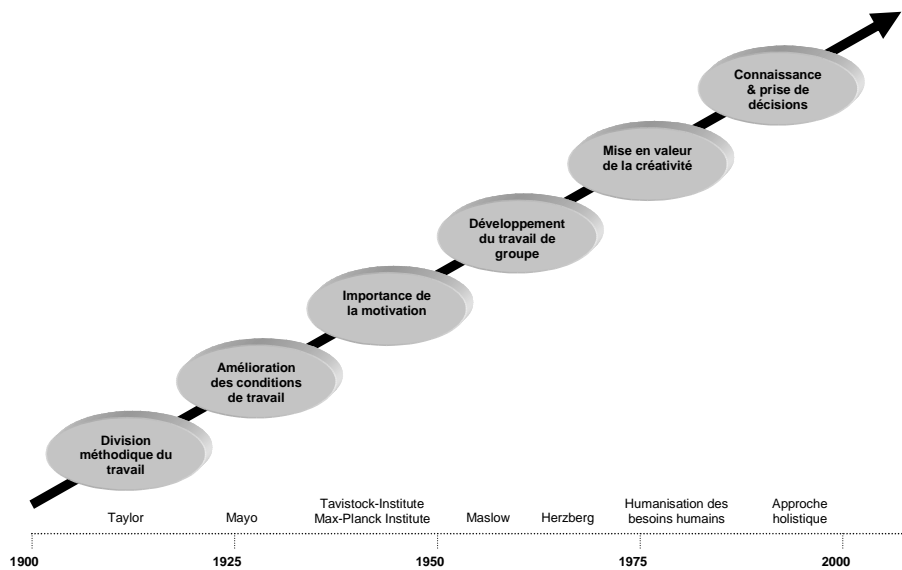


Figure 1. Développement du système social dans les entreprises. (Extraite de [9])

Cette transition de l'homme exécutant vers l'homme décideur n'est pas le seul fait d'études sociotechniques. Elle répond à un besoin croissant en terme de connaissances, de flexibilité et d'adaptabilité de la part des opérateurs de production. En effet, les systèmes de production ont dû subir de profondes mutations en raison de l'évolution des lois du marché. Nous nous proposons donc ici de mettre en évidence le rôle et l'importance de l'homme dans les différents paradigmes de production et le besoin croissant en information que cela implique.

¹ L'approche holistique consiste à traiter une organisation dans son ensemble plutôt qu'élément par élément.

1.1 Les principaux paradigmes de production

La notion générale de paradigme a été introduite par Kuhn comme l'ensemble des valeurs et des techniques que partage un groupe scientifique [10]. Dans le contexte industriel, un paradigme de production regroupe alors les pratiques que les entreprises adoptent en fonction de leur environnement interne et externe. L'histoire de la production a ainsi été marquée par différents paradigmes reflétant l'évolution de la demande : nous en recensons six ici mais il s'en développe de nombreux actuellement [11] (Figure 2).

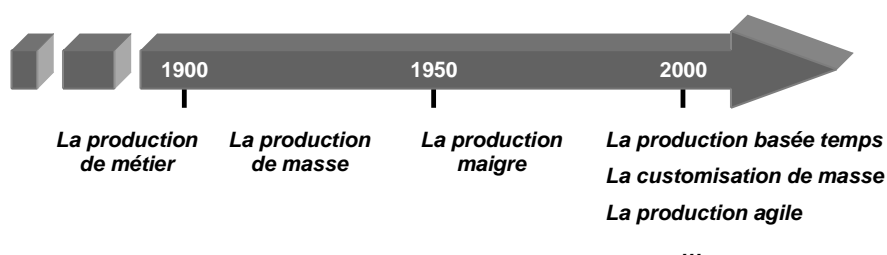


Figure 2. Evolution des paradigmes de production. (Extraite de [11])

La production de métier ne nous concerne pas dans la mesure où elle désigne les méthodes artisanales employées dans le monde pré-industrialisé avec la fabrication de produits de haute qualité à l'unité. Longtemps utilisée comme solution vis-à-vis d'une économie d'offre, c'est la production de masse qui s'est imposée comme le paradigme de production traditionnel. Cette logique a rapidement montré ses limites lors du passage à une économie de demande. La production dite maigre est alors apparue pour offrir une plus grande compétitivité aux entreprises. Actuellement, il n'existe pas de réel paradigme émergent s'opposant à ces paradigmes, mais plutôt différents modèles de production essayant de résoudre les problèmes posés par un marché de plus en plus global et volatile [12]. Le concept de production agile se distingue cependant des autres en prônant flexibilité et adaptabilité pour une réactivité maximale. Il est à noter que les nouveaux modèles de production ne sont pas exclusifs et qu'il est possible de s'orienter vers des stratégies hybrides en adéquation avec les caractéristiques de son secteur d'activité [13][14].

1.1.1 La production de masse

La production de masse consiste en la production d'une vaste quantité de produits similaires sur des lignes de production standardisées, le but étant de réduire les coûts grâce à des

économies d'échelle. Elle est donc liée à un marché stable demandant une faible variété de produits [15].

La production de masse est rendue populaire au début du XX^{ème} siècle notamment par le biais d'Henri Ford et de sa Ford T ² [16]. Le fordisme, adopté comme modèle de référence pour ce paradigme, s'inspire largement des théories tayloristes en terme d'organisation scientifique du travail. Les industries ayant adopté ce système de production sont par conséquent en recherche constante de productivité. Un tel système se caractérise par :

- un volume de production élevé
- une standardisation des produits
- un partage des ressources machines grâce aux lignes d'assemblage (1913)
- une spécialisation des ressources humaines grâce à la division du travail
- un flux de production continu

La production de masse a ainsi appelé à l'automatisation des machines qui, à son tour, a poussé à la rationalisation de l'environnement. Pour assurer le rendement, il est possible de créer un véritable espace automatisé (mécanisation des convoyeurs pour le transport, mise en place d'automates pour la manutention,...). Suivant ce principe, l'action de l'homme est réduite à son strict minimum ; il est marginalisé à des opérations de surveillance ou de maintenance d'un poste spécifique. Le taylorisme est implanté sur les lignes d'assemblage : l'ouvrier fait exécuter à sa machine des opérations identiques sur un flot de produits et il est payé à la pièce pour garantir sa productivité. Cependant, les employés les plus riches et les plus instruits refusent ce travail répétitif et débilitant de cette forme d'organisation du travail d'où une augmentation de l'absentéisme et de la malfaçon [17]. Certaines tentatives d'enrichissement du travail ont donc été menées, mais les ouvriers restent toujours au service des machines.

Bien que la production de masse ait rapidement créé de nombreuses richesses, les années 70 marquent le début de son déclin. Elle souffre en effet d'un manque de flexibilité dû à la rigidité de la fabrication en grande série [18]. Les lignes de fabrication sont configurées pour la réalisation de lots de produits identiques et il est difficile de changer une conception ou le processus de production une fois celles-ci mises en œuvre. Or le pouvoir d'achat a augmenté et les consommateurs commencent à être plus critiques envers le produit standard, bon marché mais de faible qualité. Ils recherchent plus de variété ce qui conduit à une

² La Ford T (1908-1927) est un modèle de voiture standardisé. L'application du fordisme a permis de faire passer son coût de 800 à 300\$. Elle en est devenue l'emblème au travers de la formule « any colour as long it is black ».

segmentation du marché incompatible avec la production par lots où la variété se limite à l'application de décorations en fin de ligne. Dans ce contexte, les économies d'échelle sont considérablement réduites. Les industries, perdant de leur compétitivité, doivent changer de stratégie et aller à la rencontre du consommateur. La concurrence asiatique a plus vite compris les enjeux de cette transition d'une économie d'offre à une économie de demande. Se démarquant des erreurs de la production de masse, les Japonais ont peu à peu développé leur propre paradigme de production : la lean production.

1.1.2 La production à flux tirés et maigre (le « lean »)

Le lean est une démarche visant à gérer les processus de production au plus juste afin de réduire le cycle de fabrication des produits, d'améliorer les flux et d'augmenter la qualité. L'objectif est de privilégier la valeur consommateur [19]. Un système lean repose ainsi sur deux fondamentaux : le déclenchement de la production à l'apparition du besoin (système à flux tirés) et la minimisation des gaspillages (système maigre).

Les concepts du lean trouvent leurs origines dans le TPS (Toyota Production System) instauré chez Toyota par Taiichi Ohno dans les années 50. Le terme lean apparaît bien plus tard dans « The Machine That Changed the World » pour décrire une philosophie de gestion [20]. Ce livre dénote l'influence positive qu'ont eue les méthodes utilisées dans le toyotisme sur le monde industriel automobile. L'idée de base de Ohno est la réduction progressive des activités ne générant pas de valeur ajoutée dans une optique d'amélioration continue du processus de production (kaizen). L'ensemble des actions d'optimisation concerne, selon lui, sept types de gaspillage (muda) [21] :

- la surproduction : il faut réguler la production en fonction de la demande pour éviter les excès de stock
- les délais d'attente : il faut améliorer les procédures de planification et d'ordonnancement pour un meilleur équilibrage de la charge de travail
- le transport : il faut agencer les unités de travail de façon à supprimer les opérations de transport inutiles et donc préférer une organisation en cellules
- le traitement : il faut réduire les temps relatifs aux opérations incontournables de la gamme de fabrication en automatisant au besoin certaines opérations

- le stock : il faut adopter la technique du juste-à-temps au travers d'outils comme le kanban³ afin de ne disposer que des matières premières nécessaires à la production
- le mouvement : il faut minimiser les déplacements humains car ils s'ajoutent au temps de cycle
- les rejets : il faut implémenter un système de procédures de secours et de qualité pour prévenir les défauts et les corriger, évitant ainsi les relances en fabrication

Dans la pensée lean, les ressources matérielles et humaines doivent être rentabilisées. La logique de l'automatisation a donc été conservée, mais pour augmenter la flexibilité les équipements doivent être multifonctions et facilement reconfigurables. Des techniques comme le SMED (Single Minute Exchange of Die) ont été développées pour permettre un changement rapide de la série fabriquée. D'autre part, avec le TPS, apparaît la notion d'autonotation (jidoka) qui reflète une automatisation à visage humain. L'autonotation consiste à faire des ouvriers des individus autonomes et responsables, capables de tirer le meilleur parti de l'automatisation du système productif [22]. Les machines sont par exemple équipées de dispositifs qui provoquent leurs arrêts dès qu'elles rencontrent un problème. Un opérateur n'a plus alors à surveiller constamment une machine et peut intervenir sur plusieurs postes au sein d'une cellule de production. Il devient polyvalent et s'intègre à un groupe. Parallèlement, dans le cadre de l'amélioration continue, chacun est invité à émettre ses propres suggestions sur le processus de fabrication. Le courant lean accorde donc une place plus importante à l'homme en redonnant un sens et une intelligence à son travail. L'ouvrier subit cependant une pression constante pour répondre à des objectifs de production fluctuants. Des conflits sociaux ont éclaté dans les années 90, notamment chez Toyota, les ouvriers estimant ne pas être payés à la hauteur de leur implication [23].

Le TPS a néanmoins permis à Toyota de s'imposer progressivement comme le leader mondial de l'automobile [24]. Les occidentaux, de leur côté, ont tenté de reproduire la démarche lean avec plus ou moins de succès. Considéré comme le « one best way » par de nombreuses entreprises, ce paradigme connaît encore aujourd'hui un réel engouement. Mais le lean est dans l'incapacité de répondre pleinement aux exigences du marché, étant donné que cette démarche est centrée sur la qualité et non sur la flexibilité. Les réalités économiques rendent nécessaires l'orientation vers des systèmes de production plus à même d'intégrer les changements. Se contenter d'appliquer les méthodes du lean ne peut pas suffire, et il est

³ Le kanban est basé sur la circulation d'étiquettes accompagnant les produits ou pièces détachées. Le retour d'une étiquette à son point de départ provoque la fabrication d'une nouvelle unité.

indispensable de se diriger vers une nouvelle politique dite agile qui seule donnera aux entreprises la réactivité nécessaire pour s'affirmer dans un contexte concurrentiel fort [25].

1.1.3 La production agile

Un système de production est qualifié d'agile lorsqu'il est capable de s'adapter rapidement aux évolutions du marché tout en restant compétitif. Un tel système doit pouvoir prendre en compte une variation de la demande, un changement au niveau de la conception des produits ou de leurs caractéristiques, ou encore une réduction du cycle de vie des produits [26]. La caractéristique principale des systèmes agiles est donc la flexibilité.

De ce fait, les FMS (Flexible Manufacturing Systems) sont reconnus comme les précurseurs des systèmes agiles. Un FMS est un système hautement automatisé composé de machines programmables (CNC = Computer Numerical Controlled) et de transporteurs robotisés (AGV = Automated Guided Vehicle), sous la direction d'un ordinateur central. Il est dit flexible car apte à passer d'un modèle de pièce à un autre par un simple changement de paramètres [27]. Un tel système sait s'autogérer. Bien qu'ils présentent des avantages significatifs, les FMS n'ont rencontré qu'un public assez limité en raison de leurs coûts exorbitants, de leur complexité de développement et de déploiement, et de leur fiabilité incertaine [28]. Ils sont de plus sujets à l'obsolescence et leur reconfiguration pour un nouveau produit s'avère assez fastidieuse. Les recherches actuelles s'orientent alors vers des ensembles plus petits donc plus facilement contrôlables et reconfigurables: les cellules flexibles.

Le concept d'agilité étend la notion de flexibilité en lui ajoutant les notions de réactivité et d'adaptabilité. Ainsi, l'idée directrice de la production agile, et des structures qui en ont découlé (fractales, bioniques, holoniques,...), est d'organiser l'ensemble des activités de l'entreprise de façon à ce qu'elle puisse déployer ou redéployer efficacement ses ressources en réponse aux modifications de la demande [29]. Goldman et al en donne les quatre principes fondamentaux [30]:

- l'apport de valeur au consommateur
- la constitution de partenaires virtuels permettant des collaborations temporaires sur la base d'un partage de ressources et de capitalisation d'informations (concept d'entreprise étendue)
- la capacité à réagir aux changements
- la valorisation des ressources humaines

La production agile revalorise pleinement les compétences et connaissances de l'homme : il devient un facteur clé du système de fabrication. Un système agile est en général moins automatisé car il est basé sur des choix de ressources qui privilégient l'adaptabilité [31]. Il rend ainsi sa place à l'homme au détriment de l'automatisation . Une participation active des employés est requise. Cela signifie qu'elle ne doit pas se limiter à l'exécution des tâches mais un suivi détaillé du fonctionnement des postes doit être prévu de façon à permettre à chaque employé d'ajuster ses activités en fonction de ses performances. Cette contrainte implique que les informations soient fournies en temps voulu et qu'elles soient adaptées à l'utilisation. Dans une organisation agile, les technologies à disposition sont exploitées au mieux pour augmenter les connaissances de chacun [32]. La connaissance est en effet considérée comme élément primordial au changement. Les opérateurs sont donc amenés à coopérer autour d'objectifs communs clairement identifiés, ceci afin d'élever leurs connaissances mutuelles et de renforcer leur autonomie décisionnelle. La communication entre acteurs doit être facilitée. Cette démarche a pour but final de transformer la connaissance et les idées en nouveaux produits et services afin de permettre une innovation constante en rapport avec l'évolution du marché.

En résumé, les systèmes de production modernes (production lean ou agile) ne traitent plus indépendamment les composantes organisation, ressources humaines et équipements. Ils cherchent à construire un ensemble cohérent réajustable en fonction de la demande. Dans ce contexte, les opérateurs, négligés dans la production de masse, jouent un rôle stratégique. Même si l'automatisation reste nécessaire (pour l'exécution de tâches de routine ou de précision par exemple), elle ne se fait plus à outrance. Les machines ne doivent en effet pas être vues comme une fin en soi ou comme un substitut, mais comme une aide technique pour l'opérateur qui est l'unique garant de l'adaptabilité [33]. Le tableau suivant résume l'évolution des opérateurs à travers les différents paradigmes (Tableau 1).

| | 1950 Reconstruction | 1970 Offre > Demande | 1990 Mondialisation |
|---|--|---|---|
| Paradigme prédominant | Production de masse <i>Idée : produire</i> | Production maigre <i>Idée : minimiser les coûts</i> | Production agile <i>Idée : s'adapter</i> |
| Implication des opérateurs | 1 opérateur pour 1 machine Opérateurs non qualifiés Ont peu d'influence sur la manière dont les opérations sont exécutées | 1 opérateur pour plusieurs machines Opérateurs polyvalents et qualifiés Proposent des améliorations du processus de production | 1 opérateur pour plusieurs machines Opérateurs polyvalents et hautement qualifiés Doivent faire preuve de responsabilité et d'initiatives |

Tableau 1. Place de l'homme au sein des paradigmes de production.

1.2 L'apport des nouvelles technologies pour les systèmes de production modernes

L'évolution des ordinateurs et des protocoles de communication a favorisé l'implantation des technologies informatiques dans les systèmes de production. L'information est rendue accessible et exploitable en tout point. Dans les systèmes de production modernes, l'objectif est d'intégrer les données et les processus en un ensemble applicatif cohérent : les différentes fonctions de l'entreprise sont alors fédérées autour d'une base de données commune pour satisfaire au plus juste aux besoins du consommateur [34]. Cette démarche d'intégration, où tout est supporté par l'informatique, est apparue dans les années 80 sous le nom de CIM (Computer Integrated Manufacturing). Même si l'architecture classique du CIM évolue vers une forme plus ouverte en conformité avec les enjeux des entreprises étendues, la logique de partage d'information reste aujourd'hui prédominante.

1.2.1 Le pilotage du système technique

L'architecture classique d'intégration de la production comporte 4 niveaux (processus de production, supervision, gestion de production et informatique de gestion) auxquels correspondent différents niveaux de décision : un niveau supérieur décide de ce qu'un niveau inférieur exécute (Figure 3) [35]. Les paramètres de production sont par exemple établis à plus ou moins court terme par le niveau supérieur à partir des besoins sur un horizon considéré et les données de production sont récupérées pour construire notamment des indicateurs de performance.

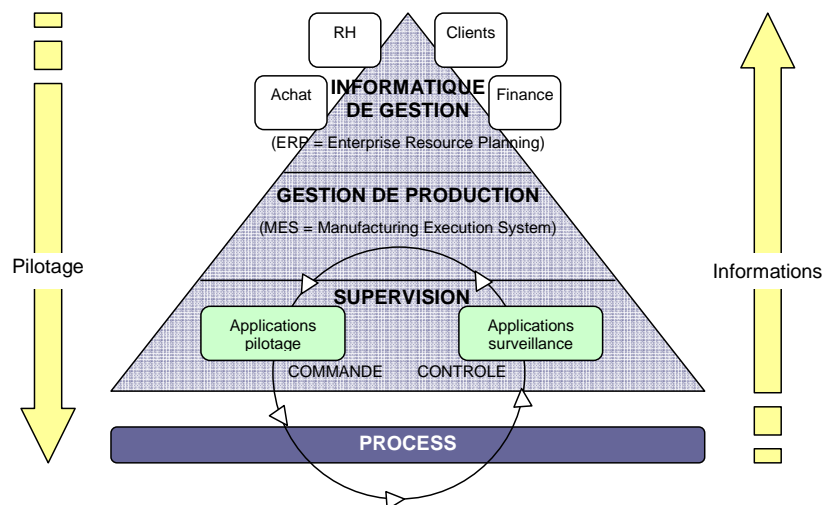


Figure 3. Pyramide d'intégration de la production. (Extraite de [36])

Le processus de production en lui-même a largement bénéficié des innovations informatiques en matière de communication. Au niveau strictement opérationnel, outre le développement de robots et d'équipements de plus en plus sophistiqués, les machines se sont vues équipées de puces programmables, capteurs, lecteurs de code-barres et autres capables de recevoir et d'échanger des informations, contribuant ainsi fortement à la supervision du processus de production [37]. Les architectures de pilotage, que nous détaillerons ultérieurement, ont suivi cette évolution : elles ont mis à profit ces nouvelles technologies pour commander et contrôler à distance le déroulement des opérations de façon autonome. Le problème de l'automatisation du pilotage a rapidement connu un réel engouement et les recherches à ce sujet sont donc abondantes, particulièrement en ce qui concerne le contrôle des cellules flexibles. Il fait encore aujourd'hui l'objet de nombreuses avancées pour la prise en compte des aléas de production.

1.2.2 Le problème des opérateurs

Cependant, nous avons vu précédemment que les systèmes modernes tendent à revaloriser l'homme en tant qu'acteur-superviseur-décideur tout en lui proposant des outils performants. Or celui-ci se retrouve face à un environnement rationalisé de plus en plus complexe, dont le fonctionnement sous-jacent lui est méconnu. Etant reconnu que la plupart des échecs des nouvelles technologies proviennent d'une mauvaise gestion des ressources humaines [38], il est primordial de prendre en compte les facteurs humains dans le pilotage de la production. Les systèmes automatisés de pilotage font en effet peu cas des humains dans leurs paramètres de fonctionnement. Toute la problématique des systèmes dits homme-machine réside dans les difficultés liées à la modélisation du système technique d'une part et à celle des décisions humaines d'autre part, le comportement de l'homme étant difficilement prévisible [39]. Millot, s'inspirant des travaux de Sheridan dans ce domaine, nous rappelle les principales tentatives de modélisation de l'opérateur et notamment les deux modèles antagonistes de Baron et de Rasmussen. Le modèle OCM (Optimal Control Model) de Baron (1968) est représentatif du courant « human engineering » : il s'attache à considérer l'homme comme un pilote optimal facilement formalisable en ne reproduisant que ses comportements réactifs vis-à-vis du système technique. Outre le fait qu'il ne prend pas en compte les paramètres d'état affectant les capacités de l'homme, un tel modèle n'est pas réaliste car il omet les dimensions cognitives de l'homme dans un contexte où celui-ci est amené à résoudre des problèmes. Le modèle multi-niveaux de Rasmussen (1986, révisé par Hoc en 1996) décrit lui deux niveaux

de comportements cognitivistes en plus du niveau basique des comportements réactifs appelés automatismes (Figure 4) : l'homme est capable d'identifier une situation et d'agir en conséquence en appliquant des procédures qu'il connaît (comportement basé sur des règles) ou qu'il invente (comportement basé sur des connaissances). Ces mécanismes cognitifs propres aux humains sont cependant difficiles à formaliser et à mettre en oeuvre. L'Intelligence Artificielle essaie d'introduire des réponses en proposant par exemple des systèmes experts censés, sur la base d'inférences, imiter le comportement humain, mais ce genre de système reste encore trop limitatif. Le modèle de Rasmussen marque un tournant dans la représentation du comportement humain face à son environnement et plus particulièrement dans la représentation du processus de décision. L'action de l'homme résulte d'un processus complexe dont l'issue est peu prévisible. Devant cette incertitude, il convient d'apporter à chaque étape une assistance à l'opérateur pour exploiter au mieux ses capacités cognitives et éviter les erreurs de jugement, plutôt que de tenter de prédire ce comportement.

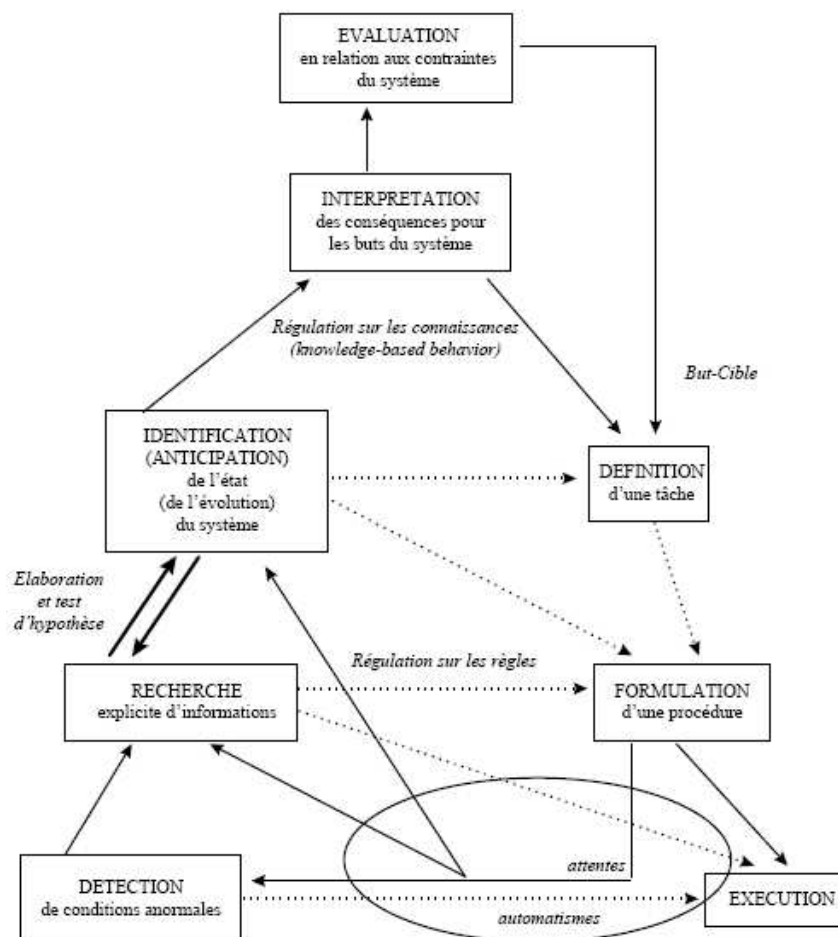


Figure 4. Modèle du comportement humain (modèle de Rasmussen révisé par Hoc).
(Extraite de [39])

Ainsi, les recherches actuelles ne remettent pas forcément en cause les modèles automatisés de pilotage du système technique mais visent à développer des outils pour l'intégration de l'homme, ce que Sun et Frick traduisent comme le passage logique d'une démarche CIM à une démarche CHIM (Computer and Human Integrated Manufacturing) [40]. Plus généralement, on parle du concept « interaction homme-machine ». Il s'agit en fait d'établir un lien entre le système de pilotage et les opérateurs. Au-delà de la commande des machines et de l'acquisition de données, l'objectif principal est de procurer aux opérateurs des informations pertinentes pour la conduite et la surveillance du processus de production assuré par les machines : il faut réduire le volume des informations dont ils disposent en une information compréhensible et accessible. L'informatique ubiquitaire (ou pervasive), avec la propagation massive de dispositifs mobiles et connectés tels que les TabletPCs et PDAs, encourage cette communication d'informations au sein de l'atelier en permettant un rapprochement vers les utilisateurs [41]. Dans ce contexte, les interfaces homme-machine jouent un rôle primordial et leur ergonomie doit être étudiée pour rendre intuitive et perceptivement évidente la complexité de l'environnement de travail (voir les interfaces dites écologiques [42]). Les outils d'assistance aux opérateurs concernent essentiellement les éléments suivants (éléments identifiables sur le modèle de Hoc donné en Figure 4) :

- l'exécution des tâches (témoins visuels simples renseignant sur l'état des machines, écrans déportés donnant les opérations exécutables, formalisation des procédés, commandes à distance...)
- le suivi du processus de production pour la détection d'anomalies (tableaux de bords, systèmes SCADA⁴ offrant une vue synthétisée de l'atelier, systèmes d'alarme...)
- l'aide au diagnostic et à la conduite à tenir lors de pannes (outils statistiques, systèmes à bases de connaissances, arbres de défaillances, ... [43])
- l'anticipation de l'état du système en cas d'action corrective (outils de simulation,...)

Aujourd'hui, un des défis est de faire évoluer cette relation homme-machine basique vers une véritable coopération où l'homme et la machine contribueraient mutuellement à la bonne marche du processus en instaurant un rapport de type humain-humain [44]. Cela consiste, entre autres, à introduire des capacités de coopération dans les machines afin que l'agent

⁴ SCADA (Supervisory Control And Data Acquisition) : Les systèmes SCADA sont typiquement employés pour la collecte de données et le contrôle. Cependant, certains systèmes permettent juste de surveiller la production.

humain et l'agent artificiel dialoguent et se répartissent les fonctions de décision pour une situation donnée le plus efficacement possibles. Néanmoins, les recherches dans ce domaine proposent surtout des cadres de conception pour de tels systèmes.

1.2.3 Vers un système de pilotage hybride...

Nos investigations relatives aux outils d'aide à décision pour l'opérateur soulèvent deux points intéressants.

1. Il apparaît que peu de solutions concrètes existent pour guider l'opérateur dans son choix d'opérations à exécuter. En effet, les travaux recensés se focalisent surtout sur la résolution de problèmes au sein de l'atelier, ce qui peut sembler logique dans la mesure où le rôle de l'homme se déplace vers des tâches de supervision. Pourtant, dans les unités semi-automatisées par exemple, les opérateurs et plus particulièrement les caristes sont amenés à choisir entre un grand nombre d'opérations ; il faut pouvoir leur proposer, dans un environnement changeant, les opérations les plus judicieuses à effectuer du point de vue du pilotage de l'unité et non pas se limiter à la liste des opérations potentielles. La problématique de l'allocation de ressources humaines est, en fait, souvent traitée en amont du pilotage. Elle fait d'ailleurs l'objet de nombreuses recherches en raison de la difficulté à satisfaire simultanément toutes les contraintes évoquées par Bennour et al. [45] : la contrainte de compétence des employés, la contrainte quantitative définissant le nombre d'employés devant être assignés à une opération, la contrainte de disponibilité des employés et la contrainte d'objectif correspondant à la réalisation des objectifs de performance de l'entreprise. Au niveau de l'unité, l'opérateur est alors considéré comme une ressource acquise, son affectation impliquant forcément sa présence. Cependant, cette affectation à priori est bien souvent éloignée des réalités de l'atelier puisque l'opérateur doit aussi gérer les aléas de production et donc être momentanément indisponible. L'allocation dynamique des ressources humaines pourrait dans ce cas constituer un premier élément de réponse, mais elle ne résout pas totalement les problèmes liés au comportement non déterministe de l'opérateur. Au final, l'ordonnancement de tâches est fréquemment réalisé par l'opérateur lui-même en fonction de la situation, d'où la nécessité de lui apporter une assistance pour se rapprocher au maximum des objectifs de performance [46]. Notre idée serait alors de s'affranchir de la problématique d'allocation en suggérant à tout moment à chaque catégorie d'opérateurs un jeu restreint de d'opérations prioritaires à exécuter compte tenu de l'état actuel du système.

2. Il apparaît que la décision finale de l'opérateur n'interfère pas avec le système de pilotage, dans le sens où son choix ne remet pas en cause le pilotage optimal initialement prévu ou du moins n'est-il pas pris en compte en temps réel. Pourtant, l'absence d'un opérateur sur une opération peut être considéré, au même titre qu'une panne de machine ou encore l'arrivée d'une commande urgente, comme une déviation par rapport au comportement attendu du système et peut donc conduire à un fonctionnement dégradé du système [47]. Il existe, pour les pannes, des procédures de réaffectation car celles-ci sont automatiquement signalées par le système technique, mais le cas de l'opérateur est plus compliqué dans la mesure où son choix n'est pas directement transmis au système de pilotage. Or toute action ou non-action d'un opérateur a une incidence sur le déroulement ultérieur des opérations. Par exemple, le choix d'une opération par un opérateur induit éventuellement une opération de transport que l'on doit affecter à un cariste (ou inversement). De même, le choix d'une opération moins prioritaire induit éventuellement un changement dans l'ordre des opérations suivantes afin de respecter les contraintes temporelles imposées par le plan de production. La prise en compte en temps réel des choix des opérateurs apparaît alors comme une évidence pour conserver une certaine cohérence et rester au plus proche de l'optimum. Il est donc important de récupérer ces choix. Dans son modèle formel d'interaction homme-machine, Shin et al. préconisent d'ailleurs l'envoi de messages par les opérateurs [48]. Notre idée serait alors de demander à l'opérateur d'indiquer, à chaque fois qu'on lui propose une liste d'opérations, l'action qu'il est prêt à exécuter, et d'adapter immédiatement l'ordre des opérations en fonction de ce choix.

En résumé, devant l'incertitude du comportement humain et l'impossibilité de le modéliser plus finement, l'interactivité nous semble un bon compromis. Nous nous proposons alors de réaliser un système de pilotage hybride centré sur les décisions humaines (mais gérant aussi les ressources machines). En ce sens, ce système se rapproche de la méthode ORABAID (ORdonnancement d'Atelier Basé sur l'Aide à la Décision) implantée dans le progiciel ORDO® [49]. Notre système a pour but principal de **suggérer** à tout moment aux opérateurs de l'unité les actions optimales à réaliser du point de vue du pilotage de la production et d'**adapter en temps réel** le pilotage compte tenu des **décisions** prises par ces mêmes opérateurs. L'interactivité est donc représentée ici comme une boucle composée de 3 phases : suggestion d'actions – décision de l'opérateur – adaptation des opérations (Figure 5).

Ce type de système permet de concilier les contraintes de production et les contraintes humaines afin de parvenir à une meilleure réalisation des objectifs et garantir ainsi une exécution robuste du plan de tâches, l'optimum étant recalculé à chaque action. Il assure de plus une relative flexibilité en laissant un certain degré de liberté aux opérateurs.

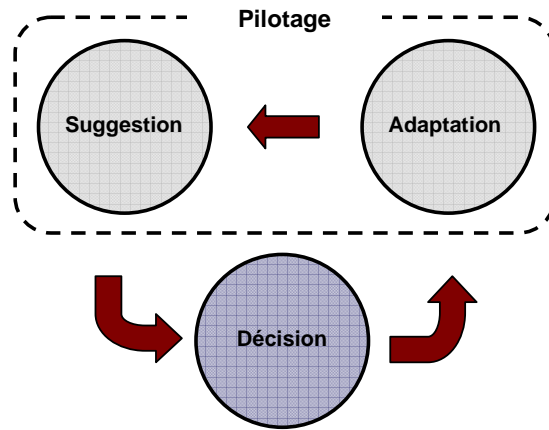


Figure 5. Modèle d'interactivité du système de pilotage.

Un tel système prend tout son sens dans les unités opérationnelles à forte composante humaine comme les organisations agiles ou même les structures hospitalières. En effet, vu l'augmentation croissante des coûts médicaux et la rareté des ressources, l'organisation des hôpitaux devient un réel point d'intérêt. Il faut trouver des solutions pour employer de manière efficace les différentes ressources dans le but de réduire les coûts sans diminuer la qualité de service. Certains hôpitaux ont déjà commencé à appliquer les techniques développées dans les industries, en particulier les techniques du lean, avec des résultats significatifs [50]. Il peut alors paraître judicieux d'appliquer notre système dans les unités hospitalières. Dans ce souci, nous souhaitons construire un système de pilotage le plus générique possible afin qu'il soit facilement adaptable à n'importe quelle unité opérationnelle. Il est important de préciser à ce stade que ce système n'a pas vocation à se substituer à la partie commande du pilotage. Notre étude définit en fait un cadre d'implémentation pour l'architecture décisionnelle d'un tel système.

Nous rappellerons donc, dans un premier temps, les caractéristiques des systèmes de pilotage conçus pour les systèmes de production automatisés.

2. Les systèmes de pilotage

Il existe, dans les unités industrielles, deux types de production : la production discrète dans laquelle les objets traités sont des unités discrètes composées de parties individuelles (ex. : meubles) et la production continue dans laquelle les objets sont non discrets (ex. : liquides). Bien que les contraintes soient différentes dans ces deux types d'unités de production, l'objectif est le même, à savoir l'optimisation de la production pour en réduire le coût et/ou maximiser le bénéfice. De plus, les industries doivent constamment répondre à de nouveaux besoins en terme de produits et de qualité et doivent donc être flexibles pour réagir face à une compétition croissante. Les échanges d'informations s'avèrent alors indispensables. Ce scénario a mené à des systèmes de production complexes intégrant des systèmes de pilotage de plus en plus élaborés.

Un système de pilotage supervise la production et gère en temps réel la réalisation du plan de fabrication issu de la planification. Il est utilisé pour augmenter la productivité dans des environnements dotés de machines intelligentes, capables de prendre en charge des processus complexes et variés, mais devant être rentabilisées étant donné leurs coûts. Une fois installé, le système de pilotage doit permettre de gérer l'unité entière. Un tel système englobe en général de nombreux sous-systèmes.

2.1 *La structure d'un système pilotage*

Un système de pilotage est responsable de la pile des tâches, de l'intégration des contraintes locales, du lancement, du suivi des produits et de chaque ressource d'une unité opérationnelle. Il comprend deux composantes : la conduite et la commande [51] (Figure 6).

- La conduite correspond au niveau décisionnel du pilotage. Elle assure la définition des décisions possibles, la prise de décision à proprement parler et l'application de celle-ci de manière cohérente avec les objectifs de production. Les décisions sont traduites en ordres et transmises au niveau commande.

Cette fonction peut être décomposée en sous-fonctions : le contrôle qui garantit la cohérence globale du système en gérant les interactions entre les différents centres de décision, le suivi qui collecte les données relatives au fonctionnement du système,...

- La commande traduit les ordres transmis par le niveau de conduite en une séquence d'instructions exécutables par les équipements de l'unité opérationnelle.

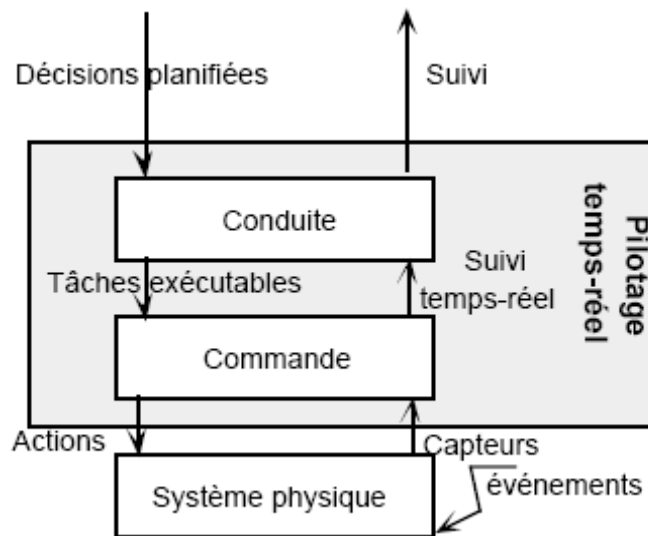


Figure 6. Les fonctions de pilotage. (Extraite de [51])

Le pilotage dépend donc directement des décisions d'ordonnancement planifiées par le niveau supérieur. Ordonnancer consiste à organiser dans le temps la réalisation de tâches, compte tenu de contraintes temporelles (délais, contraintes d'enchaînement,...) et de contraintes portant sur la disponibilité des ressources requises [52]. Il existe plusieurs types d'ordonnancement selon l'horizon temporel voulu : à long terme au niveau stratégique, à moyen terme au niveau tactique, à court terme au niveau opérationnel, et à très court terme ou en temps réel au niveau atelier. L'ordonnancement est affiné à chaque niveau. Nous distinguerons ici deux catégories d'ordonnancement :

- l'ordonnancement prévisionnel ou ordonnancement a priori, effectué en amont du pilotage et visant à optimiser l'utilisation des ressources dans le respect des délais pour répondre à la demande exprimée par les niveaux supérieurs
- l'ordonnancement en temps réel ou ordonnancement a posteriori, effectué par le système de pilotage et correspondant à la réalisation effective du plan de tâches conformément à l'état réel de l'atelier

Différents types de pilotage sont définis en fonction de la présence ou non d'un ordonnancement prévisionnel.

2.2 Typologie du pilotage

Trois classes de pilotage peuvent être spécifiées [53] : le pilotage prédictif, le pilotage proactif et le pilotage réactif. Cette classification se rapproche des travaux de Davenport et Beck dans le domaine de l'ordonnancement robuste [54][55]. Dans cette classification, le type de pilotage dépend de l'existence d'un ordonnancement prévisionnel et de la capacité à réviser cet ordonnancement.

L'ordonnancement prévisionnel est en général déterministe dans la mesure où il ne tient pas compte des aléas survenant dans le système physique de production. Réalisé hors ligne, il affecte aux opérations des ressources et décide d'une date de début et une date de fin de façon à optimiser un critère donné (moyenne des dates d'achèvement des tâches, plus grande date d'achèvement des tâches ou makespan, moyenne des durées de présence des tâches, moyenne des retards de tâches,...). Les opérations sont alors considérées comme planifiées. Par la suite, nous définirons une tâche (ou ordre de fabrication) comme une séquence d'opérations à réaliser, caractérisée par une quantité à fabriquer, une date de mise en fabrication au plus tôt et une date de fabrication au plus tard. Chaque opération est elle-même caractérisée par un temps opératoire. Dans un problème d'ordonnancement donné, il existe N tâches et pour une tâche notée $i \in \{1,2,\dots,N\}$, l'ensemble des opérations à réaliser est désigné par $\{(i,1), (i,2), \dots, (i,n_i)\}$ où n_i est le nombre total d'opérations à réaliser pour exécuter i .

L'ordonnancement prévisionnel est dit total si le sous-ensemble des opérations planifiées est celui des opérations à réaliser. Il est dit partiel si le sous-ensemble des opérations planifiées est inclus dans l'ensemble des opérations à réaliser.

Les méthodes de résolution des problèmes d'ordonnancement prévisionnel sont très nombreuses et font appel à des techniques très variées. Il est nécessaire d'analyser et de modéliser le problème posé afin de déterminer s'il peut être résolu de manière optimale par un algorithme polynomial ou non. La littérature à ce sujet est très complète. Pour un panorama détaillé des techniques d'ordonnancement, le lecteur peut se rapporter à Blazewick et al [56]. Les méthodes de résolution d'ordonnancement puisent dans toutes les techniques de l'optimisation combinatoire (programmation linéaire, programmation dynamique, théorie des graphes,...). Ces méthodes garantissent en général l'optimalité de la solution fournie. Cependant, la plupart de ces algorithmes ne peuvent pas être utilisés pour des problèmes de grande taille, menant à une explosion combinatoire (problèmes dits NP-complets), d'où la nécessité de construire des méthodes de résolution approchée basées sur des métaheuristiques.

Widmer recense quatre approches dans l'utilisation de métaheuristiques [57] : l'approche constructive (méthodes complétant à chaque itération une solution partielle), l'approche de recherche locale (recuit simulé, méthode tabou,...), l'approche évolutive (algorithmes génétiques, algorithme de la fourmi,...) et l'approche hybride (combinaison de diverses métaheuristiques).

2.2.1 Le pilotage prédictif

Dans le cas du pilotage prédictif, un ordonnancement prévisionnel total est établi hors ligne (Figure 7). Cet ordonnancement est ensuite utilisé en ligne pour guider les décisions. Les paramètres de pilotage sont donc déterminés avant exécution sur le système réel et souvent implantés dans des automates programmables. Ce type de pilotage est cependant peu utilisable dans un environnement soumis aux perturbations.

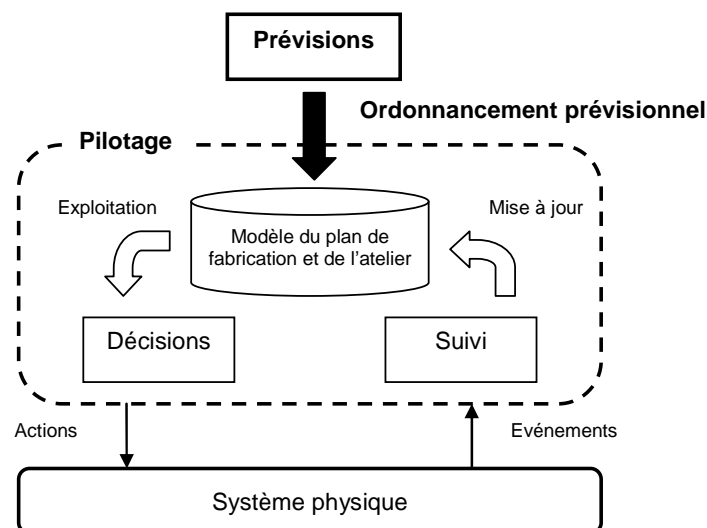


Figure 7. L'approche prédictive. (Extraite de [55])

2.2.2 Le pilotage proactif

Le pilotage proactif est basé sur les mêmes principes que le pilotage prédictif, mais cherche à anticiper les perturbations les plus certaines. Il utilise alors les plus souvent un ordonnancement prévisionnel partiel. Il s'agit d'anticiper les incertitudes en jouant sur la flexibilité de sorte à produire un ordonnancement, ou une famille d'ordonnancements, relativement insensible aux incertitudes (Figure 8). Ce type de pilotage permet d'absorber certains aléas mais montre ses limites dans un environnement fortement dégradé.

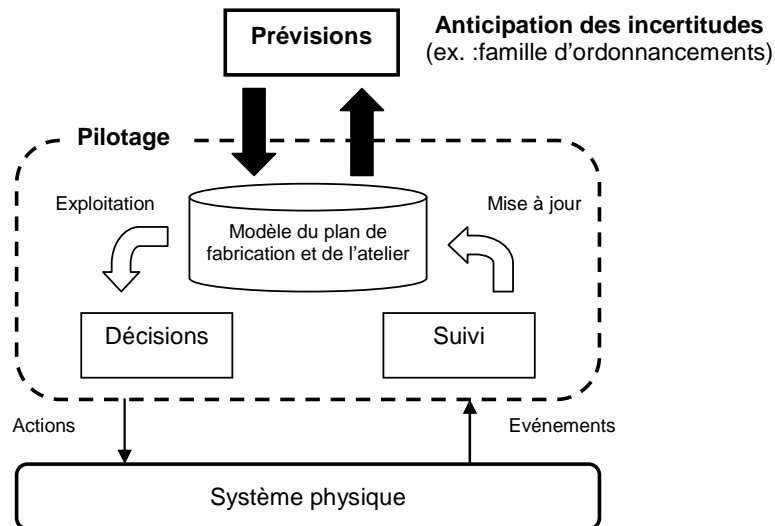


Figure 8. L'approche proactive. (Extraite de [55])

2.2.3 Le pilotage réactif

Dans le cas du pilotage réactif, aucun ordonnancement prévisionnel n'est élaboré hors ligne. L'ordonnancement est réalisé en temps réel : les allocations sont gérées dynamiquement au fur et à mesure de l'évolution événementielle du système de production (Figure 9). Le pilotage réactif se fait donc pendant l'exécution et sans anticipation. Il permet de réagir en temps réel aux événements imprévus et de minimiser leur impact sur le déroulement de la production. Un ordonnancement temps réel est de moindre qualité par rapport à un ordonnancement prédictif, mais il améliore la capacité à absorber les aléas de production.

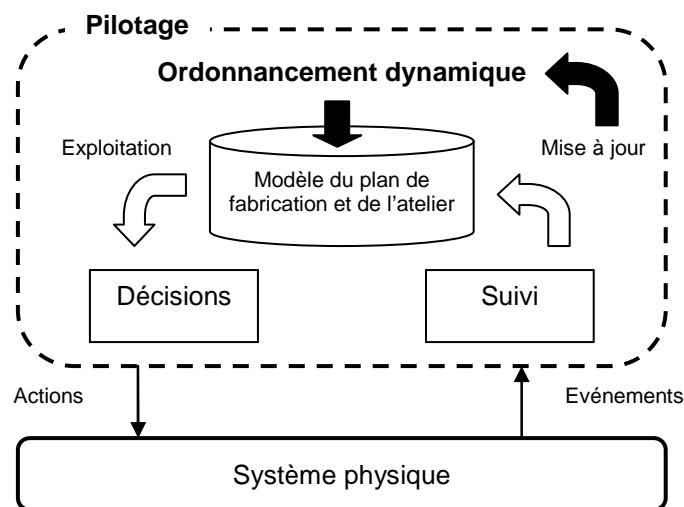


Figure 9. L'approche réactive. (Extraite de [55])

Une des méthodes les plus connues pour l'ordonnancement temps réel est la gestion des files d'attente par règles de priorité [58]. Cette méthode utilise des règles de priorité pour déterminer, chaque fois qu'une ressource se libère, la prochaine opération à traiter parmi toutes les opérations en attente de cette ressource. Le choix est réalisé en fonction des paramètres concernant les opérations (durée opératoire), les produits (date de fin au plus tard souhaitée) ou les ressources (taux d'utilisation). Il existe un très grand nombre de règles de priorité.

Le pilotage réactif, sans ordonnancement prévisionnel, est beaucoup plus récent que les autres types de pilotage et fait actuellement l'objet de nombreuses études. Il est important de préciser que les différentes approches peuvent se combiner pour satisfaire au mieux les objectifs : on obtient alors un pilotage de type prédictif-réactif dans lequel un ordonnancement prévisionnel est fourni mais éventuellement adapté en cours de production, ou encore un pilotage de type proactif-réactif.

2.3 *Les architectures de pilotage*

La variété et la complexité des fonctions de conduite imparties au système de pilotage imposent une décomposition en modules de pilotage. Définir une architecture de pilotage consiste alors à diviser le pilotage en différents groupes fonctionnels et à répartir, dans ses différents groupes, les fonctions de décision [59]. Elle doit être étudiée avec attention car elle conditionne la performance finale du système physique de production.

Les architectures de pilotage ont suivi l'évolution des technologies informatiques pour aboutir à des architectures modulaires complexes. En effet, le prix des équipements informatiques a rapidement diminué d'où leur multiplication dans les unités opérationnelles, tandis que la puissance de calcul n'a cessé d'augmenter. Dans le même temps, des protocoles de communication performants ont été développés et peu à peu améliorés.

Quatre architectures de base existent pour le pilotage des unités opérationnelles : l'architecture centralisée, l'architecture hiérarchique, l'architecture hétérarchique et l'architecture hybride [60]. Ces architectures ont été essentiellement expérimentées pour le contrôle des ateliers flexibles [61]. Actuellement, la définition d'une architecture de pilotage pour superviser la production aboutit le plus souvent à une architecture hybride, modulaire et distribuée, les autres architectures ayant montré leurs limites [62].

2.3.1 L'architecture de pilotage centralisée

Historiquement, l'architecture centralisée est la plus ancienne. Dans un contexte où les ordinateurs étaient très chers et les communications difficiles à établir, elle semblait être la solution la plus réalisable d'un point de vue technique.

Il s'agit de l'architecture la plus simple pour le pilotage des systèmes automatisés. Dans cette architecture, un ordinateur central est connecté à tous les équipements de l'atelier et gère seul tous les événements survenant au cours de la production (Figure 10).

Cette architecture a l'avantage d'offrir une optimisation globale, mais présente les inconvénients suivants :

- la fragilité du système (panne possible du superviseur) ;
- le temps de réponse parfois très long en raison du volume d'informations à traiter ;
- la complexité et l'efficacité relative du pilotage lorsque la dimension du système à piloter est trop importante, complexité du pilotage ;
- la saturation du système en cas de perturbations trop fréquentes ;
- la difficulté d'extension du système, tout ajout d'équipement requérant une profonde modification du système de pilotage.

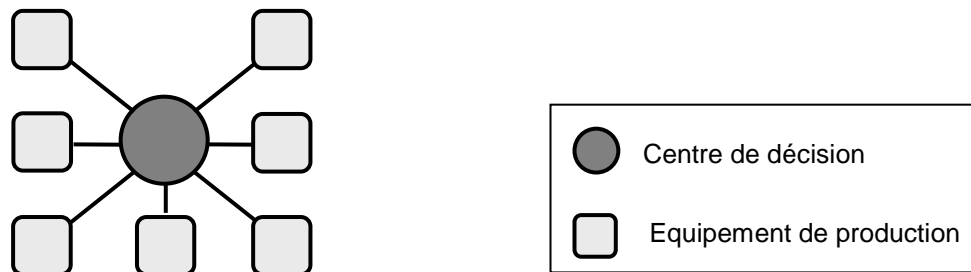


Figure 10. L'architecture centralisée.

2.3.2 L'architecture de pilotage hiérarchique

L'évolution de l'informatique permet aujourd'hui d'utiliser de nombreux ordinateurs puissants, de les spécialiser et de les relier facilement entre eux. De nouvelles formes d'architectures de pilotage, plus flexibles, sont donc apparues : les architectures hiérarchiques. L'architecture hiérarchique divise le système en plusieurs sous-systèmes sur 1 à n niveaux (Figure 11). Un problème complexe est ainsi décomposé en sous-problèmes plus simples, ce qui facilite sa résolution. Les interactions entre niveaux sont basées sur le concept maître-

esclave : chaque sous-système reçoit des instructions de son supérieur immédiat et fournit lui-même des instructions à ses subordonnés immédiats.

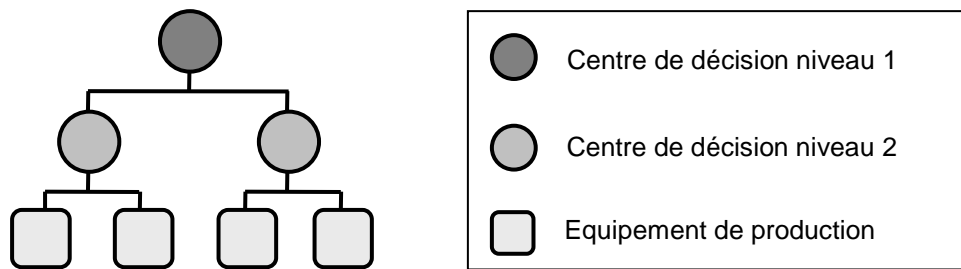


Figure 11. L'architecture hiérarchique.

Une telle architecture est en général plus efficace que l'architecture centralisée en terme de qualité de décision. Elle facilite de plus le déploiement du système et offre la possibilité de l'implémenter progressivement. Il est cependant nécessaire de préciser que la capacité décisionnelle des sous-systèmes reste ici limitée : elle consiste à affiner au fur et à mesure les décisions de haut niveau. Ainsi, l'apparition de perturbations par exemple réduit significativement la performance de ce type de système, étant donné que les informations concernant un incident doivent remonter l'ensemble de la hiérarchie pour une prise de décision globale.

Pour résoudre le problème de réactivité face aux aléas, un autre type d'architecture a été introduit : l'architecture hiérarchique modifiée [63], appelée aussi architecture coordonnée, où une coopération est possible au sein d'un même niveau (Figure 12). Les liaisons transversales entre sous-systèmes autorisent alors un échange d'information qui améliorent le traitement dynamique des perturbations.

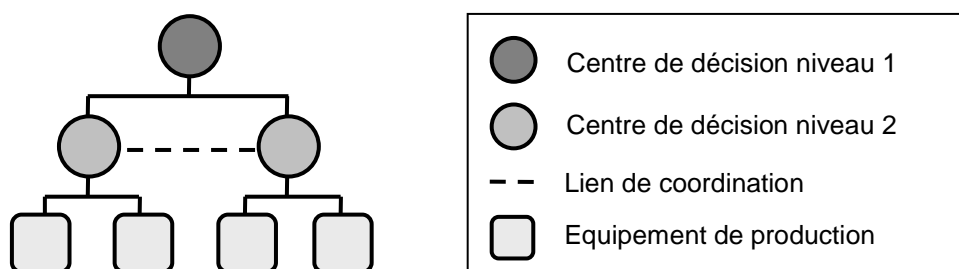


Figure 12. L'architecture coordonnée.

2.3.3 L'architecture de pilotage hétéralche

Finalement, grâce au développement de nouvelles techniques telles que l'intelligence artificielle et les systèmes multi-agents, un autre type d'architecture a émergé : l'architecture

hétérarchique. Elle a été proposée par Hatvany comme la fragmentation du système de pilotage en petits modules complètement autonomes (Figure 13), chacun d'eux poursuivant ses propres buts selon ses propres règles. Elle est fondée sur une distribution totale des capacités de décision.

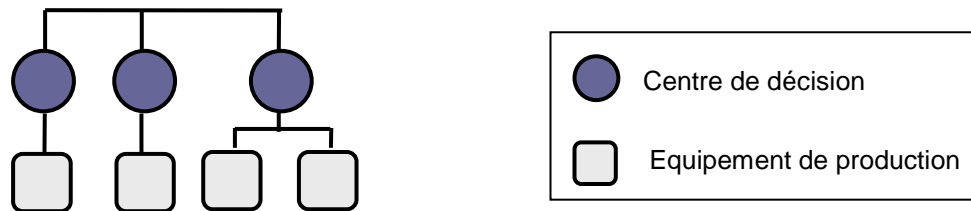


Figure 13. L'architecture hétérarchique.

Cette caractéristique confère une grande réactivité au système. Un autre avantage, dû à la non-dépendance, réside dans la capacité de modification du système (il suffit de changer le module concerné) et sa capacité d'extension (il suffit d'ajouter un module). L'optimisation des décisions est cependant réduite puisque les décisions sont prises localement et de façon autonome, sans vue globale du système, d'où un manque de synchronisation.

L'introduction des concepts holoniques répond typiquement à ce genre d'architecture, les holons étant définis comme des modules de production autonomes, flexibles et interchangeables [64]. Une telle infrastructure est le plus souvent supportée par une programmation multi-agents.

2.3.4 L'architecture de pilotage hybride

Selon les critères essentiels pour l'estimation des architectures de pilotage (extensibilité, adaptabilité et fiabilité), l'architecture centralisée se place au rang le plus bas, l'architecture hétérarchique au rang le plus haut, et l'architecture hiérarchique est considérée comme un intermédiaire entre les deux. Par conséquent, il peut être utile de combiner les avantages des deux extrêmes pour converger vers la meilleure architecture possible.

Les architectures hybrides exploitent les concepts hiérarchiques et hétérarchiques. Dans une architecture de pilotage hybride, les entités opèrent sous le contrôle d'un superviseur (concept hiérarchique) mais ont une latitude décisionnelle plus importante (concept hétérarchique). (Figure 14). Elle permet une meilleure prise en compte de l'imprécision qui peut accompagner les décisions des niveaux supérieurs, chaque entité de niveau inférieur organisant les tâches localement en fonction de ses contraintes. Le superviseur est utilisé pour

coordonner et assurer la cohérence des conduites locales gérées de manière autonome au niveau inférieur, dans la mesure où celles-ci n'ont pas accès à toutes les données relatives à l'environnement global de production. Ainsi, la décision est distribuée mais reste supervisée.

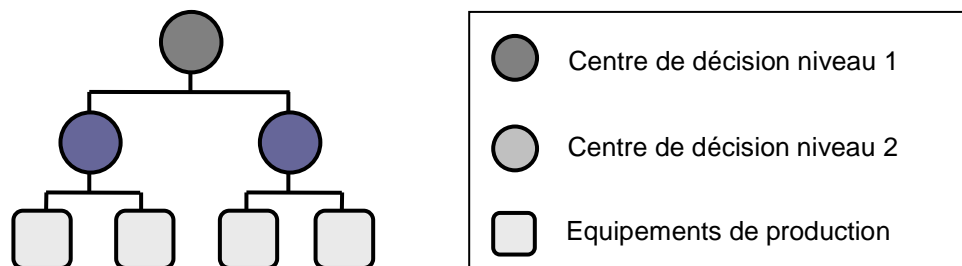


Figure 14. L'architecture hybride.

Il est également possible de faire collaborer les entités des niveaux inférieurs. Dans ce cas, on se retrouve dans une architecture de type CODECO (CONduite DEcentralisée COordonnée). Cette approche, proposée par le Laboratoire d'Automatique de Grenoble [65], répartit le processus de décisions sur deux niveaux hiérarchiques : un niveau supérieur supervise le niveau inférieur qui comprend des modules décisionnels autonomes coopérant directement les uns avec les autres pour mieux absorber les perturbations. Il s'agit en fait d'une combinaison entre l'architecture coordonnée (hiérarchique modifiée) et l'architecture hétérarchique.

2.4 Un système de pilotage adapté à l'interactivité

En résumé, le rôle d'un système de pilotage est de transférer les ordres de fabrication engendrés par la gestion de production vers le système physique de production. Il assume des décisions complexes inhérentes à l'application en temps réel d'un plan de tâches. Le pilotage est généralement précédé d'un ordonnancement préalable des tâches, dit ordonnancement prévisionnel. Cependant, les conditions incertaines de production font qu'il doit pouvoir être révisé et adapté à l'état réel du système. Le pilotage réactif, procédant à une allocation dynamique des ressources, semble, dans ce cas, être la meilleure solution. Il apparaît que les architectures de pilotage hybrides, à la fois hiérarchique et hétérarchique, sont les plus adaptées à ce type de pilotage en garantissant une réactivité certaine tout en assurant la cohérence des décisions. La décentralisation et la distribution des fonctions de décision réduit la complexité de la prise de décision et assure une meilleure flexibilité, une meilleure autonomie et une meilleure prise en compte des aléas. La solution obtenue n'est certes pas optimale, mais il est illusoire de vouloir conserver par exemple un ordonnancement global alors que les données nécessaires à son calcul ne seront plus valides lors de son utilisation en

raison des nombreuses déviations possibles. Une solution locale, d'une qualité inférieure mais plus réaliste, est alors acceptable.

En ce qui concerne notre étude, la nature interactive du système de pilotage voulu impose de manière évidente un pilotage de type réactif sans ordonnancement prévisionnel. L'approche ORABAID, que nous avons évoqué précédemment pour sa similarité dans sa volonté d'interactivité, est elle basée sur un pilotage réactif à ordonnancement prévisionnel partiel. Le pilotage exploite en temps réel une séquence de groupe de tâches permutables et s'inscrit dans une logique d'aide à la décision en proposant différents jeux d'ordonnancement aux opérateurs [66]. Dans notre cas, le pilotage est amené à évoluer en temps réel en fonction des événements de l'unité : les états machines et surtout les actions des opérateurs. Nous pouvons donc d'ores et déjà envisager une architecture hybride (de type CODECO ou non) pour fournir des décisions rapides. Un premier modèle global est donné et sera enrichi au fur et à mesure de l'avancement de nos travaux (Figure 15). Il convient alors d'étudier de manière détaillée le découpage fonctionnel de notre système.

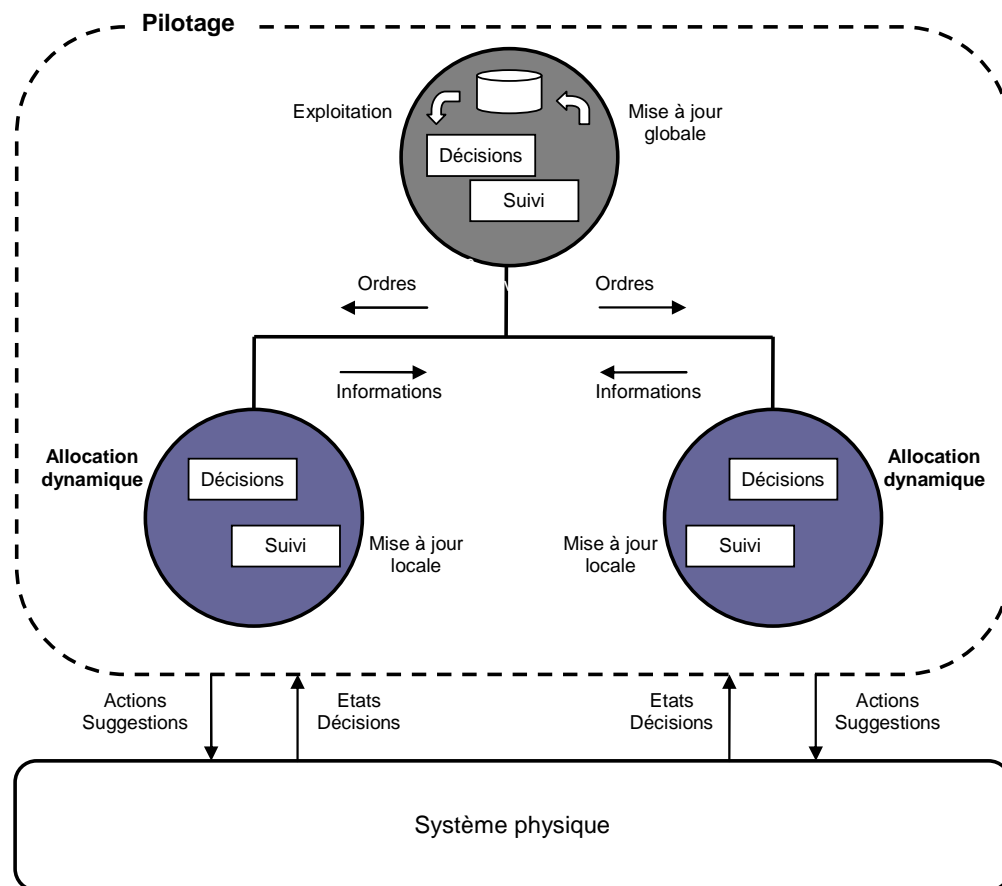


Figure 15. Modèle global de notre système de pilotage.

3. Etude fonctionnelle du système

Le système que nous souhaitons développer est essentiellement destiné à apporter aux opérateurs de l'aide au niveau des actions que chacun d'entre eux doit exécuter à très court terme. En effet, les objectifs de production induisent une orchestration complexe de tâches basée sur une quantité d'informations trop accablante pour le cerveau humain. Notre système se propose de suggérer en temps réel, à chaque catégorie d'opérateurs, les opérations prioritaires à réaliser du point de vue du pilotage de l'unité, et d'adapter ce pilotage en fonction des choix effectués. En étant centré sur les décisions des humaines, ce type de pilotage présente l'avantage de concilier le flux de production avec les contraintes des opérateurs (indisponibilité, proximité d'une action,...).

3.1 *Système de pilotage ou système d'aide à la décision ?*

Lorsque l'on parle d'aide à l'utilisateur, le terme généralement associé est système d'aide à la décision. Pourtant, le système dans son intégralité ne peut s'apparenter à un SAD/SIAD (Système d'Information et d'Aide à la Décision) puisque qu'il assume des fonctions de pilotage de production. Du strict point de vue de l'opérateur (sans considération du système sous-jacent), le système ne peut être non plus vu comme un SIAD à proprement parler. En effet, le terme SIAD est réservé aux niveaux supérieurs de gestion (tactique et stratégique) [67]. Un SIAD utilise en général des outils d'extraction de données pour fournir des informations structurées sous la forme de tableaux de bord par exemple. Il fait ensuite appel à la connaissance pour l'interprétation et l'exploitation du résultat. Dans notre cas, le résultat transmis à l'opérateur est totalement structuré. Pour cette raison, nous préférerons le terme d'assistance à celui d'aide à la décision.

En réalité, le système que nous étudions est globalement chargé d'assurer le bon déroulement de la production, c'est-à-dire ici de superviser les différentes ressources d'une unité opérationnelle pour répondre au mieux aux besoins de production. Il effectue par exemple l'allocation de ressources machines et l'allocation de ressources humaines, même si cette dernière est gérée de manière interactive, pour les différentes opérations nécessaires à la réalisation du plan de tâches. En ce sens, il appartient bien à la classe des systèmes de pilotage. Cependant, il est loin d'implémenter toutes les fonctions d'un système de pilotage (en particulier la partie commande) et sa vocation première est d'apporter de l'aide aux

opérateurs. Par la suite, nous parlerons donc d'un Système d'Assistance Interactif (SAI) pour le pilotage humain, mais il faut garder à l'esprit qu'il s'agit d'un système de pilotage simplifié.

3.2 Une architecture générique pour le SAI

Le SAI est destiné à être implanté dans différents types d'unités à composante humaine, telles que des unités industrielles semi-automatisées ou encore des unités hospitalières. Cette volonté d'adaptabilité implique une recherche de généricité, tant au niveau logiciel qu'au niveau fonctionnel. Il est donc primordial d'adopter une représentation générique des unités opérationnelles, utilisant une terminologie valable dans la plupart des cas.

Le SAI est fortement orienté sur les décisions humaines concernant le choix d'une opération à exécuter. Par conséquent, le découpage fonctionnel du système doit être envisagé sous cet angle.

3.2.1 Typologie des décisions

Le processus de décision, tel que défini par Simon [68], comporte trois phases :

- la collecte d'informations (Intelligence) qui correspond à l'identification du problème et à la compréhension de sa structure ;
- la conception (Design) qui correspond à la construction de scénarios ou à la recherche de solutions possibles pour la résolution du problème ;
- le choix (Choice) qui correspond au choix de la solution retenue, considérée par le décideur comme la meilleure pour résoudre le problème posé.

Simon recense trois types de décisions relatives à ce processus [69] :

- la décision structurée : les personnes impliquées dans le processus décisionnels sont capables de structurer le problème, de l'explicitier et de le formaliser, et la décision est alors dite programmable ;
- la décision semi-structurée : les personnes impliquées dans le processus décisionnels sont capables de ne structurer qu'en partie le problème car elles ne disposent que d'informations incomplètes et imparfaites;

- la décision non structurée : les personnes impliquées dans le processus décisionnel sont dans l'incapacité de structurer le problème car celui-ci est trop complexe en raison de la nature et de la quantité d'informations qu'il véhicule ;

Les outils d'aide à la décision permettent d'améliorer le degré de structuration des décisions concernées [70]. Dans ce contexte, il existe plusieurs systèmes d'information et d'aide adaptés à chacun des trois niveaux de gestion définis par Ansoff [71] (Tableau 2).

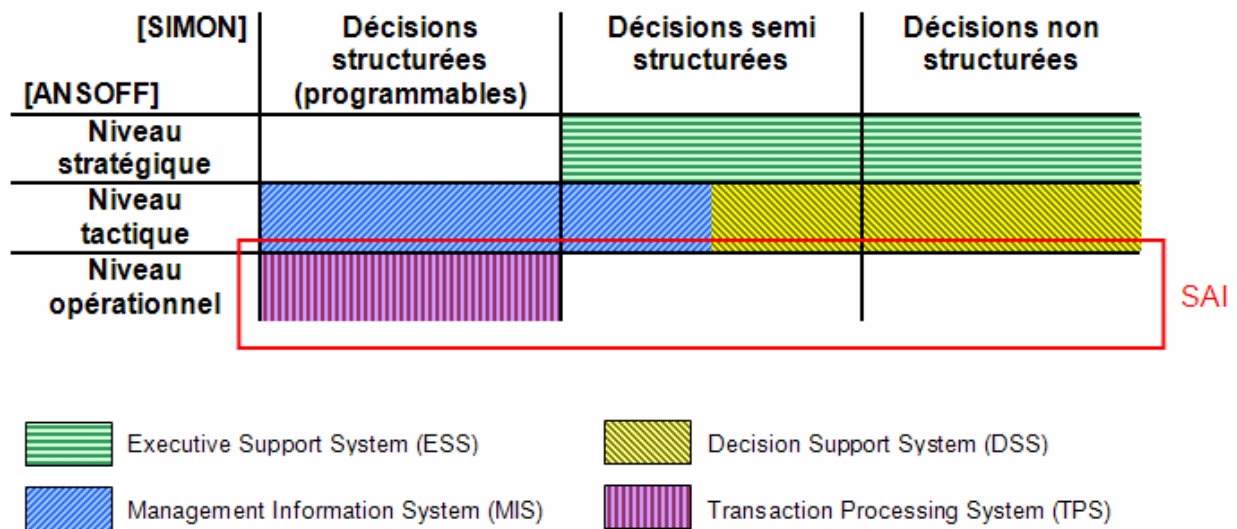


Tableau 2. Positionnement du SAI.

Par rapport aux principaux systèmes existants, le SAI se situe au niveau opérationnel. Il est censé couvrir, en ce qui concerne le choix d'une opération, tous les types de décisions humaines évoquées par Simon.

3.2.2 Cartographie des décisions d'ordre purement opérationnel

Traditionnellement, la classification des décisions peu ou pas structurées est associée à un comportement cognitif de haut niveau, stratégique ou tactique. Nous nous proposons de la ramener à un comportement cognitif beaucoup plus simple, d'ordre purement opérationnel : « choisir une opération à exécuter parmi un ensemble plus ou moins important d'opérations possibles ». Ainsi simplifiée, le degré de structuration de la décision d'un opérateur dépend de la quantité d'informations dont il dispose sur son environnement. Nous avons identifié trois possibilités, résultant du couplage homme/équipement mis en jeu.

- Les décisions opérationnelles structurées

Elles sont liées à des opérateurs qui n'interviennent que sur un et un seul équipement. L'opérateur est, pendant une période donnée, assigné de manière permanente à un équipement appartenant à un ensemble d'équipements consacrés à la réalisation d'une classe d'opérations (ex. : le réglage de moteurs). L'opérateur et l'équipement sont considérés comme un tout. Dans ce cas, l'opérateur doit juste attendre l'allocation d'une opération à son équipement selon la disponibilité de ce dernier, et il n'a aucune décision réelle à prendre dans la mesure où il n'a pas de choix à faire. En ce sens, sa décision peut être assimilée à une décision structurée. Cette catégorie d'opérateur est appelée 'opérateur statique'.

L'assistance pour un ensemble d'opérateurs statiques consiste ici dans l'allocation dynamique d'équipements inoccupés pour les opérations de plus haute priorité (Figure 16). Aucune action n'est suggérée aux opérateurs puisque les produits assignés sont directement transportés sur leurs propres équipements. Pour ces allocations, il est nécessaire de récupérer les états des équipements. Ces états sont supposés pouvoir être transmis par le personnel, mais le plus souvent ils sont transmis par l'équipement lui-même (degré d'automatisation normal d'une unité industrielle). A noter que le principe serait le même pour un ensemble d'équipements automatisés ne nécessitant aucun opérateur.

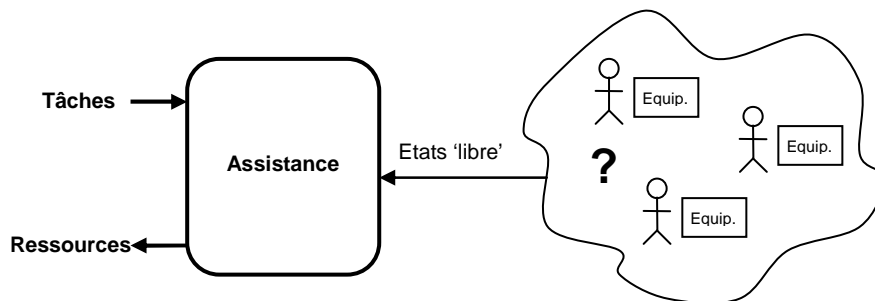


Figure 16. Un ensemble d'opérateurs statiques.

- Les décisions opérationnelles semi-structurées

Elles sont liées à des opérateurs qui interviennent sur un des N équipements consacrés à la réalisation d'une classe opération. L'opérateur est amené à opter, de manière ponctuelle, pour une opération à exécuter parmi N possibles. Dans ce cas, l'opérateur doit choisir entre un nombre relativement restreint de solutions en disposant, pour cette décision, de quelques informations directes sur son environnement de travail. En ce sens, sa décision peut être assimilée à une décision semi-structurée. Cette catégorie d'opérateur est appelée 'opérateur semi-mobile'.

L'assistance pour un ensemble d'opérateurs semi-mobiles se fait en deux étapes : une fois qu'un des équipements inoccupés est alloué, l'opération correspondante est suggérée aux opérateurs (Figure 17). Une allocation complète est composée de l'équipement et de l'opérateur. Pour ces allocations, il est nécessaire de récupérer à la fois les états des équipements et les choix faits par les opérateurs.

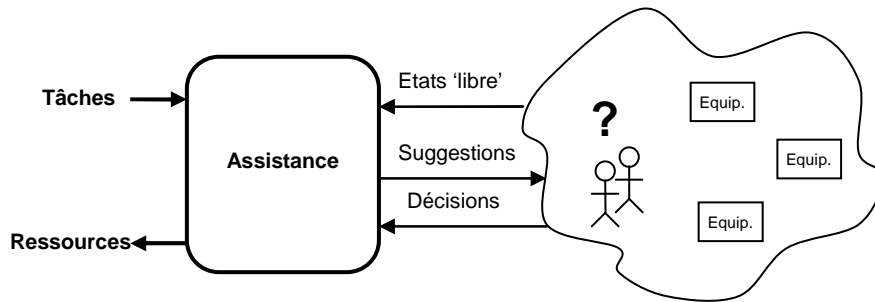


Figure 17. Un ensemble d'opérateurs semi-mobiles.

- Les décisions opérationnelles non structurées

Elles sont liées aux opérateurs qui effectuent les transferts de produits entre deux ensembles d'équipements (ou entre le stock d'entrée et un ensemble d'équipements, ou entre un ensemble d'équipements et le stock de sortie). L'opérateur doit alors sélectionner un point de départ parmi M équipements et un point d'arrivée parmi N équipements. Dans ce cas, l'opérateur doit choisir entre un nombre relativement élevé de solutions en ne disposant, pour cette décision, que de peu d'informations directes sur son environnement et sur l'état actuel de la production. En ce sens, sa décision peut être assimilée à une décision non structurée. Cette catégorie d'opérateur est appelée 'opérateur totalement mobile'.

L'assistance pour un ensemble d'opérateurs totalement mobiles consiste ici dans la proposition d'opérations de transport, induites par la volonté de réalisation d'une opération ou la fin d'une opération sur un équipement (Figure 18). Ces opérateurs disposent d'une marge de manœuvre et il est donc nécessaire, pour les allocations, de récupérer leurs choix.

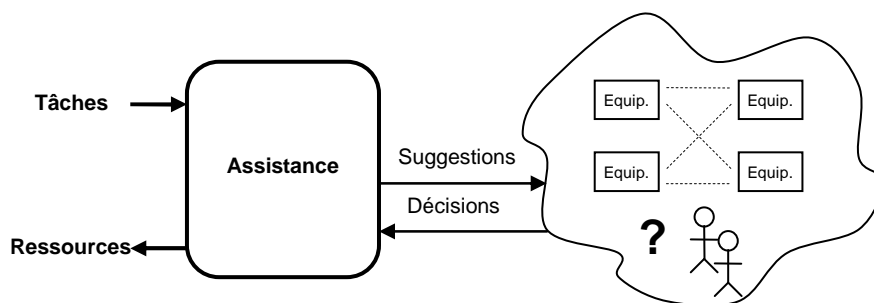


Figure 18. Les opérateurs totalement mobiles.

Ce découpage apparaît comme une solution logique pour catégoriser de manière générique les opérateurs d'une unité en fonction de leurs décisions. En effet, pour un opérateur, la complexité du processus de décision, et donc l'assistance fournie, diffère avec son périmètre d'action. Trois cas de figures sont alors envisageables pour le pilotage : l'opérateur est statique, l'opérateur est semi-mobile ou l'opérateur est totalement mobile.

3.2.3 Les composants fonctionnels du SAI

Constatant que l'assistance fournie dépend du type d'opérateur, nous avons naturellement été amenés à répartir les fonctions de pilotage du SAI en conséquence. Dans cette optique, nous avons défini différents modules de pilotage appelés Contrôleurs de Ressources.

- Les Contrôleurs de Ressources (CR)

L'idée est de créer des contrôleurs génériques chargés de gérer les décisions et le suivi propres à un ensemble de ressources ayant un degré d'implication humaine plus ou moins élevé, sachant qu'un ensemble de ressources est consacré à la réalisation d'une classe d'opérations :

- Un Contrôleur de Ressources Statiques (CRS) gère les opérations à réaliser par un ensemble de ressources comprenant des opérateurs de type statique. Chaque opération requiert un équipement, la présence d'un opérateur étant implicite (un opérateur « appartient » ici à un équipement). Le rôle essentiel du contrôleur est d'affecter les équipements aux opérations en fonction de leurs états (libre, occupé, début d'opération et fin d'opération,...).
- Un Contrôleur de Ressources Mobiles (CRM) gère les opérations à réaliser par un ensemble de ressources comprenant des opérateurs de type semi-mobile. Chaque opération nécessite un équipement et un opérateur. Le rôle essentiel du contrôleur est d'affecter les équipements aux opérations et de communiquer avec l'ensemble d'opérateurs pour trouver des opérateurs susceptibles de réaliser ces opérations.
- Un Contrôleur de Ressources de Transfert (CRT) gère les opérations à réaliser par un ensemble de ressources comprenant des opérateurs de type totalement mobile. En fait, chaque opération est entièrement assurée par un opérateur, même si celui-ci utilise un outil ou véhicule pour les réaliser (transport par chariot, manutention...). Le rôle essentiel du contrôleur est de communiquer de manière

interactive avec cet ensemble d'opérateurs de transfert : le contrôleur propose des opérations et obtient en retour les choix des opérateurs.

Chacun des contrôleurs génériques prend donc en charge un type de décision humaine (Tableau 3).

| [SIMON] [ANSOFF] | Décisions structurées | Décisions semi structurées | Décisions non structurées |
|---------------------|------------------------------------|----------------------------------|---------------------------------------|
| Niveau stratégique | | | |
| Niveau tactique | | | |
| Niveau opérationnel | Contrôleur de Ressources Statiques | Contrôleur de Ressources Mobiles | Contrôleur de Ressources de Transfert |

SAI

Tableau 3. Positionnement des contrôleurs du SAI.

Les Contrôleurs de Ressources disposent d'une autonomie dans leurs décisions. Mais pour que ces décisions restent cohérentes au niveau global, les contrôleurs doivent être coordonnés par un contrôleur central.

- Le Contrôleur Central (CC)

Chaque ensemble de ressources identiques s'occupant d'une opération particulière donne lieu à un Contrôleur de Ressources de type CRS, CRM ou CRT. Par exemple, à un ensemble de ressources semi-mobiles ayant la charge de la préparation de moteurs est associé un CRM appelé 'CRM de préparation'. Il est créé autant de Contrôleurs de Ressources que d'opérations possibles pour gérer au mieux localement les différents ensembles de ressources et s'adapter au mieux à leurs contraintes. Pour superviser tous les contrôleurs générés par une unité opérationnelle, un contrôleur spécifique de niveau supérieur, le Contrôleur Central, est nécessaire. Sa fonction est double :

- Il génère différentes listes d'opérations prioritaires qu'il répartit selon les contrôleurs de ressources. En fait, il exploite un plan de tâches donné pour le traduire en ordres compréhensibles chacun des contrôleurs de ressources.
- Il centralise les décisions d'allocation prises par les contrôleurs et effectue un suivi du déroulement de la production par le biais des contrôleurs de ressources, afin de mettre à jour en temps réel les listes d'opérations. Cette centralisation permet,

entre autres, de revoir la priorité de certaines opérations si besoin ou encore de synchroniser les opérations de transport.

Nous avons donc adopté, en nous inspirant des architectures de pilotage existantes, une architecture hybride à deux niveaux [72] : les décisions sont distribuées dans des modules autonomes (les Contrôleurs de Ressources de type CRS, CRM ou CRT), mais coordonnées par un système central (le Contrôleur Central) au niveau supérieur (Figure 19). Le SAI reste ainsi au plus proche des utilisateurs finaux et réagit rapidement aux différents événements survenant dans un ensemble de ressources. Cependant, nous n'avons pas opté pour la coopération des contrôleurs de ressources, comme dans une architecture de type CODECO, afin de marquer le caractère générique et agile de notre système : il peut facilement être implanté et reconfiguré dans la mesure où l'ajout/suppression d'un contrôleur n'impose pas la reconfiguration de liens trop nombreux.

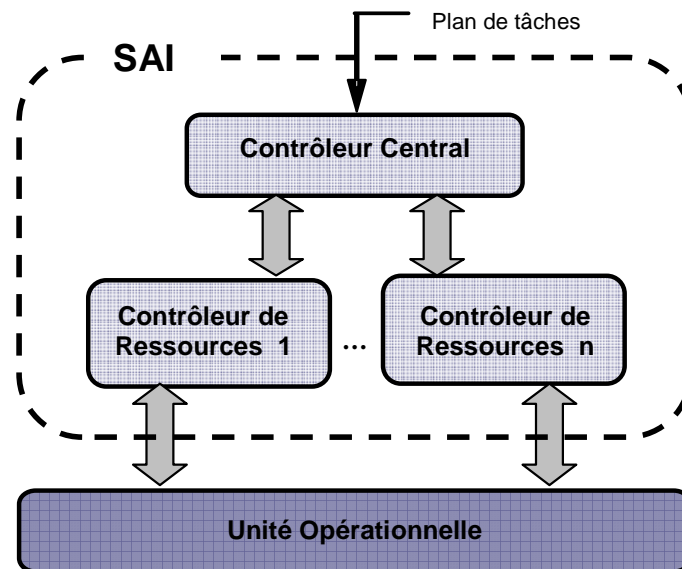


Figure 19. Architecture du SAI.

En résumé, une vue fonctionnelle du SAI consisterait à dire que le Contrôleur Central est chargé de gérer en temps réel les ordres de traitement en fonction de leurs contraintes temporelles. Pour cela, il effectue, à partir du plan de tâches, un découpage en opérations élémentaires et transmet les requêtes d'exécution d'opérations à chacun des contrôleurs concernés. Chaque Contrôleur de Ressources recevant une requête réalise les allocations de ressources en fonction de son type (CRS, CRM ou CRT) : elles sont basées sur la collecte d'événements en temps réel durant le fonctionnement de l'unité. Les données concernant le

compte-rendu d'exécution sont renvoyées au Contrôleur Central pour garantir une bonne coordination des opérations et procéder à d'éventuelles corrections sur l'ordre des opérations à venir.

Ce système de pilotage utilise donc trois principales sources d'information :

- Le plan de tâches fourni en amont : il indique une liste de produits à traiter dans l'unité pour une période donnée ainsi que les contraintes temporelles (reflétant les priorités des produits).
- Les états des équipements de production : ils signalent la disponibilité des équipements et l'état d'avancement d'une opération. Bien que le système ne commande pas directement les équipements de production, nous supposons que chaque équipement de production est une ressource capable de communiquer son état courant (libre, occupé, début d'opération, fin d'opération...).
- Les actions des opérateurs dans l'unité : un opérateur reçoit du système une liste mise à jour des opérations les plus prioritaires à exécuter et valide son choix quand il décide d'effectuer une opération de la liste. Les listes d'opérations proposées sont amenées à évoluer constamment selon les choix des opérateurs et l'avancement réel de la production. La création et la forme de ces listes dépendent de l'implication décisionnelle de l'opérateur.

En réalité, les échanges de données sont un peu plus complexes et constituent un élément primordial du SAI. Par conséquent, il est nécessaire d'étudier plus précisément la nature de ces échanges.

Les transmissions de données correspondent soit à des échanges à l'intérieur du SAI (entre Contrôleur Central et Contrôleurs de Ressources) soit à des échanges entre le SAI et l'unité (entre Contrôleurs de Ressources et équipements) :

- A l'intérieur du SAI, les transmissions de données se doivent de rester complètement indépendantes des ressources de l'unité. Les différents Contrôleurs de Ressources paraissent relativement homogènes du point de vue de leurs échanges ; ils sont du type « ordre/suivi » inhérent aux systèmes de pilotage distribué.
- Les échanges de données entre l'unité opérationnelle et le SAI conditionnent le fonctionnement de ce dernier, car le SAI doit transmettre des informations et

recupérer des événements pour une adaptation du pilotage en temps réel. Les communications sous-jacentes varient en fonction des équipements et les plateformes d'exécution peuvent prendre de multiples aspects (écrans tactiles, assistants personnels,...). Par souci de généricité, nous ne nous sommes donc pas intéressées à la partie commande et collecte du SAI. Cependant, il convient de bien cibler les événements à exploiter pour l'implémentation future des contrôleurs.

Un travail de modélisation doit nous permettre, dans un premier temps, de mieux analyser chaque problème.

3.3 *Modélisation du SAI*

Nous souhaitons modéliser le SAI dans son intégralité afin de mettre en évidence son comportement interne et ses interactions vis-à-vis du système physique de production (partie opérative).

3.3.1 La modélisation des systèmes de production

Compte tenu de la nature du système à modéliser, il nous faut d'abord choisir un formalisme adapté à la représentation des systèmes de production. En effet, le modèle d'un système de pilotage doit inclure un modèle de la partie opérative pour pouvoir réaliser son objectif.

Draghici et al dressent un bilan détaillé des méthodes de modélisation disponibles en vue de l'analyse, de la conception et de la spécification des systèmes de production [73] :

- Les approches orientées opérations (analyse structurée grâce à un diagramme de flux de données) : très limitées, elles visent essentiellement à déduire les données d'entrée nécessaires à une application et sont dans l'incapacité de produire un schéma de base de données.
- Les approches orientées données (Entité / Association) : elles définissent une structure générale de données, indépendante de programmes qui les manipulent. Ne s'intéressant qu'aux aspects structurels et informationnels, elles n'intègrent pas les aspects dynamiques (conditions de déclenchement dans le temps des contraintes opératoires difficilement représentables).
- Les approches orientées comportement (REMORA - J.S.D.) : elles sont capables de représenter les contraintes de dynamique et d'évolution dans le temps, mais ne

privilégient ensuite que soit l'aspect données (méthode REMORA) soit l'aspect traitements (méthode J.S.D.).

- Les approches combinées (MERISE - SADT) : elles prennent bien en compte les aspects traitement et données, mais de façon séparée.
- Les approches orientées objet : contrairement aux méthodes précédentes, elles tendent à définir une structure permettant de rassembler les aspects données et les aspects traitement. Avec l'avènement de l'approche objet dans le domaine de la programmation informatique, celle-ci est de plus en plus utilisée dans la problématique de l'étude des systèmes de production car elles possèdent de nombreuses propriétés intéressantes telles que la réutilisabilité des objets, l'évolutivité (faible couplage entre objets), la capacité d'intégration des informations (polymorphismes autorisant des exécutions différentes en fonction de l'état courant du système). L'utilisation de l'approche objet pour le pilotage des systèmes de production implique une décomposition en plusieurs niveaux.
- Les approches orientées agent : développées dans le cadre de l'intelligence artificielle distribuée, elles permettent d'aborder les problèmes complexes d'une manière distribuée et de proposer des solutions réactives et robustes. Un agent est en effet une entité physique ou virtuelle autonome capable d'agir dans un environnement et de communiquer directement avec d'autres agents pour aboutir à un objectif commun [74]. En conséquence, l'approche multi-agents fait partie des méthodes les plus utilisées pour l'ordonnancement dynamique et le pilotage des systèmes de production.

Actuellement, la modélisation des systèmes de production est principalement basée sur des techniques orientées objet ou agent. En ce qui concerne la conduite proprement dite, les propriétés de coopération des agents s'avèrent plus intéressantes. Ils sont particulièrement adaptés aux architectures de type CODECO. Cependant, compte tenu de l'absence de liens de coordination entre les contrôleurs dans l'architecture choisie, nous ne choisirons pas cette option. En effet, le formalisme objet nous sera plus utile, par la suite, pour la seconde étape de notre travail de modélisation qui consiste à simuler le système.

3.3.2 Une représentation UML du SAI

En construisant le modèle orienté objets du SAI, le système se voit doté d'une représentation plus fonctionnelle et surtout plus générique.

Le système peut en effet être considéré comme un ensemble d'objets dotés de différentes fonctionnalités (objet 'Produit', objet 'Ressource',...). Dans l'approche orientée objet, un objet est défini comme une collection de propriétés structurelles (les attributs) et procédurales (les méthodes) décrivant son état et les opérations qu'il est capable d'exécuter. La communication entre les objets s'effectue par l'envoi et la réception des messages qui déclenchent l'exécution d'une méthode et modifient éventuellement l'état de l'objet concerné.

UML (Unified Modeling Language) n'est pas à l'origine des concepts objet, mais il en donne une définition plus formelle et apporte la dimension méthodologique qui faisait défaut à l'approche objet. Il est ainsi rapidement devenu incontournable. En effet, il couvre toutes les phases d'un cycle de développement et il est indépendant du domaine d'application et du langage d'implémentation, ce qui en fait un langage universel [75][76]. UML constitue donc un mode de représentation idéal de notre SAI.

UML fournit un ensemble de formalismes de modélisation assez riche, adaptés à chaque besoin :

- spécification du système → diagramme de use-cases
- structure du système → diagramme de classes, diagramme d'objets
- interactions entre objets → diagramme de séquence, diagramme de collaboration
- dynamique des objets → diagramme d'états-transitions, diagramme d'activités
- réalisation → diagramme de composants, diagramme de déploiement

L'objectif annoncé est d'analyser le SAI d'un point de vue fonctionnel. Ceci passe par la définition des caractéristiques structurelles du système et l'étude des échanges associés. Nous nous limiterons donc dans un premier temps aux trois modèles de base : diagramme de use-cases, diagramme de classes et diagramme de séquence.

- Spécification du SAI (use-cases)

Les cas d'utilisation (use-cases) sont considérés comme une technique très efficace pour déterminer les besoins et les fonctionnalités du système [77]. Ils permettent de définir les limites du système ainsi que les relations entre le système et son environnement. La définition d'un use-case consiste dans l'identification des acteurs, c'est-à-dire tous les éléments extérieurs au système, qui interagissent avec lui. Les use-cases doivent décrire les interactions des acteurs avec le système pour l'accomplissement d'une tâche. Ils adoptent une vue boîte

noire dans le sens où les entrées et les réponses du système sont définies mais l'intérieur du système n'est pas encore réellement spécifié. Nous proposons un use-case pour représenter le comportement du SAI (Figure 20).

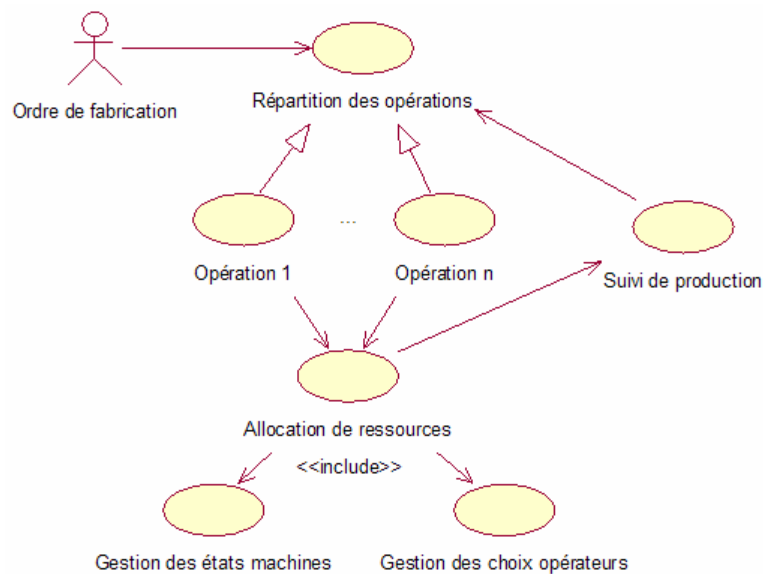


Figure 20. Use-case de traitement d'un ordre de fabrication.

Il décrit le comportement général du système par rapport à l'arrivée d'un ordre de fabrication généré en amont : découpage en opérations (rôle du Contrôleur Central) puis allocation et suivi des opérations (rôle des Contrôleurs de Ressources), sachant que les décisions des Contrôleurs de Ressources donnent lieu à un nouveau découpage. Bien que le SAI soit voué à interagir avec l'unité, nous supposons ici les ressources comme internes au système. Nous nous situons en fait au niveau correspondant à la structure décisionnelle du système et considérons les événements comme quasiment acquis puisque nous ne sommes pas amenés à gérer leur collecte. Nous étudions le fonctionnement du système dans sa globalité pour nous focaliser sur les composantes du SAI.

- Structure du SAI (diagramme de classes)

Le diagramme de classes simplifié associé à ce use-case modélise les différents contrôleurs du SAI, les principaux objets et méthodes nécessaires à ce traitement (Figure 21)⁵. Une opération est réalisée par une ressource appartenant à un ensemble de ressources identiques supervisé par un Contrôleur de Ressources générique associé à ce type de ressources. Les Contrôleurs de Ressources générés par l'unité sont coordonnés par le Contrôleur Central.

⁵ Le use-case et le diagramme de classes ont été réalisés à l'aide du logiciel Rational Rose.

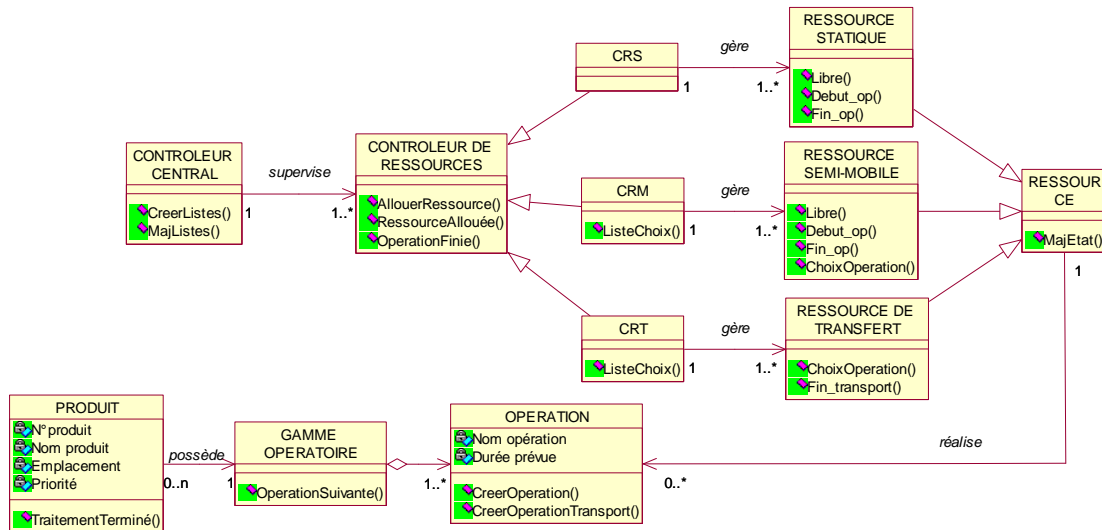


Figure 21. Diagramme de classes.

Notons qu'une modélisation plus fidèle, intégrant la partie opérative, reviendrait à utiliser la démarche adoptée par Fabian et al [78]. Cette méthode consiste à distinguer deux types d'objets : les objets externes représentatifs des équipements physiques et les objets internes qui sont des modèles informatiques des objets externes. Les objets externes exécutent les tâches qui leur sont imparties à travers les objets internes correspondants. Ceux-ci communiquent par des messages avec le contrôleur et les autres objets du système et transforment les messages reçus en messages compréhensibles par les objets externes. Chaque objet interne est divisé en deux parties : une partie générale qui participe à l'échange de messages et une partie spécifique qui communique avec les objets externes. Cette représentation permet alors d'implémenter les caractéristiques spécifiques aux équipements et d'assurer la traduction des ordres du système de conduite. La conduite n'est cependant réalisée ici que par un contrôleur qui assure la synchronisation et le contrôle des objets. De plus, elle ne prend pas en compte les caractéristiques humaines.

- Interactions entre objets (diagramme de séquence)

Le comportement du système, capturé par les cas d'utilisation, est basé sur des événements. Les cas d'utilisation permettent de définir les scénarii. Un scénario peut être vu comme un chemin particulier de la description générale donnée par un cas d'utilisation. En UML, un diagramme de séquence est représentatif d'un scénario.

Le diagramme de séquence, proposé ici en exemple pour notre SAI, représente le traitement d'un ordre de fabrication prioritaire ne contenant qu'une seule opération sur équipement mais induisant deux opérations de transport (Figure 22). Il met en évidence les différents messages entre instances. Il décrit notamment les événements déclencheurs, l'utilisation des différents contrôleurs (le contrôleur de l'opération de type CRS et le contrôleur des transports de type CRT) et la façon de coordonner leurs actions. Ce scénario donne les événements essentiels au bon déroulement du processus de production : prise en compte et réservation des ressources, affectation du transporteur et transport physique, exécution du process en lui-même, et enfin transport du produit hors de l'unité.

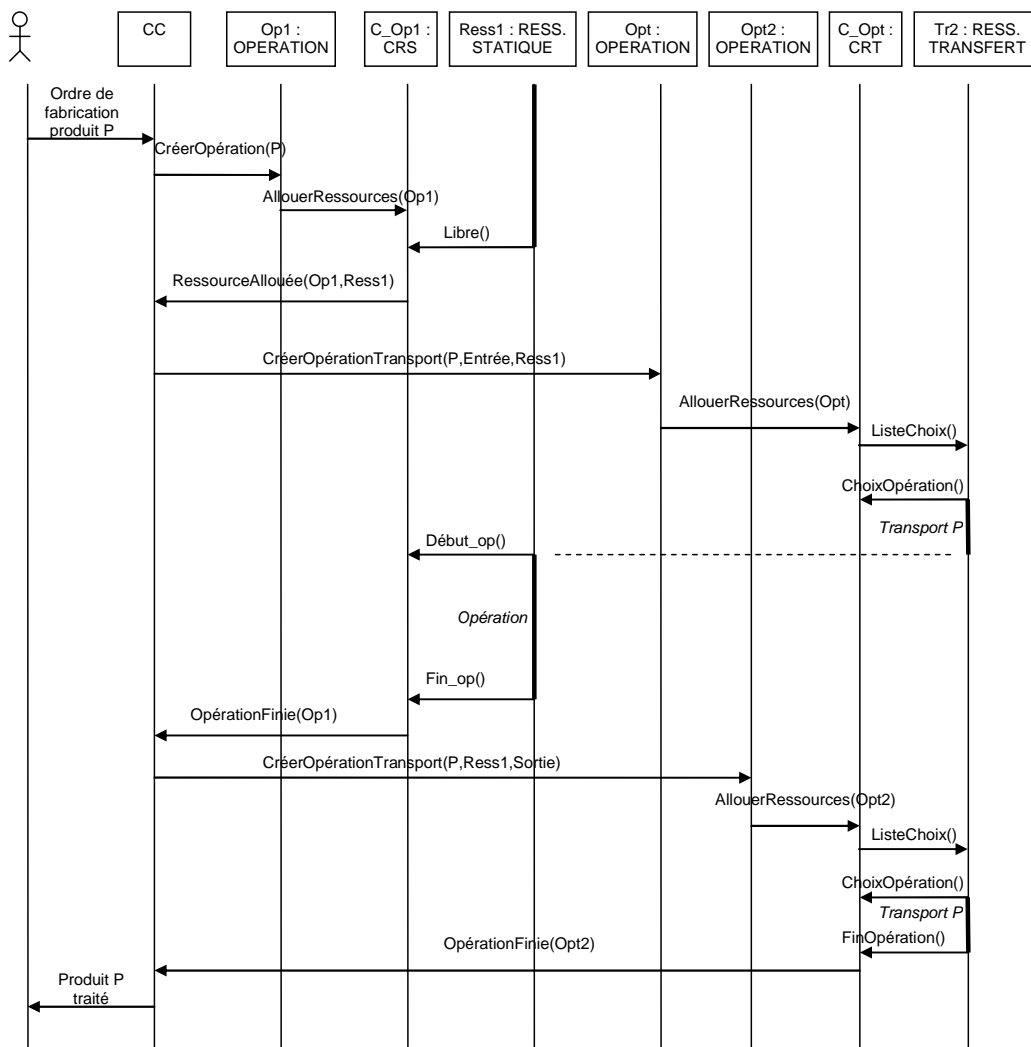


Figure 22. Diagramme de séquence.

Plus précisément :

1. L'ordre de fabrication du produit P est traité au niveau du Contrôleur Central CC qui crée les opérations possibles (ici l'opération sur équipement Op1) et les

transmet aux Contrôleurs de Ressources concernés (ici C_Op1 de type CRS) pour réalisation.

2. Dès qu'un équipement indique sa disponibilité (ici Ress1), C_Op1 l'affecte à l'opération Op1 et le signale au CC.
3. Le CC crée l'opération de transport correspondante (Opt) du stock vers Ress1 et la transmet au Contrôleur de Ressources concerné (ici C_Opt de type CRT) pour réalisation.
4. C_Opt insère l'opération Opt dans la liste suggérée aux opérateurs. Dès qu'un opérateur choisit d'effectuer Opt (ici Tr2), il le signale au contrôleur C_Opt. Tr2 effectue alors physiquement ce transport.
5. Une fois que le produit est amené sur Ress1, l'opération peut commencer. Lorsqu'elle finit, un événement est envoyé à C_Op1. Celui-ci signale la fin d'exécution au CC.
6. Le CC crée alors une nouvelle opération de transport (ici Opt2) de Ress1 vers la sortie. La procédure de pilotage est la même que pour Opt.
7. La fin du transport génère un événement transmis du contrôleur C_Opt vers CC. Le traitement du produit est, à ce moment, considéré comme terminé.

Ce scénario donne une première idée des échanges et des synchronisations nécessaires à la viabilité du SAI.

Généralement, le nombre de scénarii relatifs à un cas d'utilisation est limité, mais lorsqu'ils sont nombreux et qu'il est impossible de tous les décrire, un problème se pose dans la prévision du comportement de système. En UML, il est possible de modéliser le comportement dynamique sous la forme de diagramme d'états-transitions ou de diagramme d'activités. Cependant, ces modèles analytiques sont limités dans leur représentation de la réalité pour deux raisons :

- le temps n'y est pas exprimé de manière explicite ;
- ils sont basés sur des hypothèses simplificatrices et très restrictives.

La modélisation par simulation et les techniques d'évaluation de performance sont utilisées pour résoudre ces problèmes. Par conséquent, nous avons choisi de compléter ce travail préalable de modélisation par une simulation du SAI. Elle doit nous permettre d'évaluer le comportement du système au cours du temps.

3.3.3 De UML à la simulation

Gertosio et al proposent un cadre méthodologique pour le passage d'un modèle UML à un modèle de simulation qui justifie notre étude précédente [79]. Cette démarche fait le lien entre le formalisme UML et les techniques de simulation. Elle commence après la modélisation UML basique, incluant le diagramme de classes et les principaux cas d'utilisation, et comporte quatre étapes (Figure 23) :

Etape 1. Capturer le comportement du système par le biais de un ou plusieurs scénarii qui détaillent la boîte noire en spécifiant les instances d'objets avec leurs interactions.

Etape 2. Etablir une correspondance entre le(s) scénario(ii) et le modèle de simulation. Il s'agit de l'étape centrale de cette approche.

Etapes 3&4. Programmer et exécuter le modèle de simulation. Ces étapes permettent la validation, la vérification et le raffinement du modèle de simulation. Elles peuvent être reproduites plusieurs fois. Il est également possible de modifier les cas d'utilisation et de recommencer ce processus.

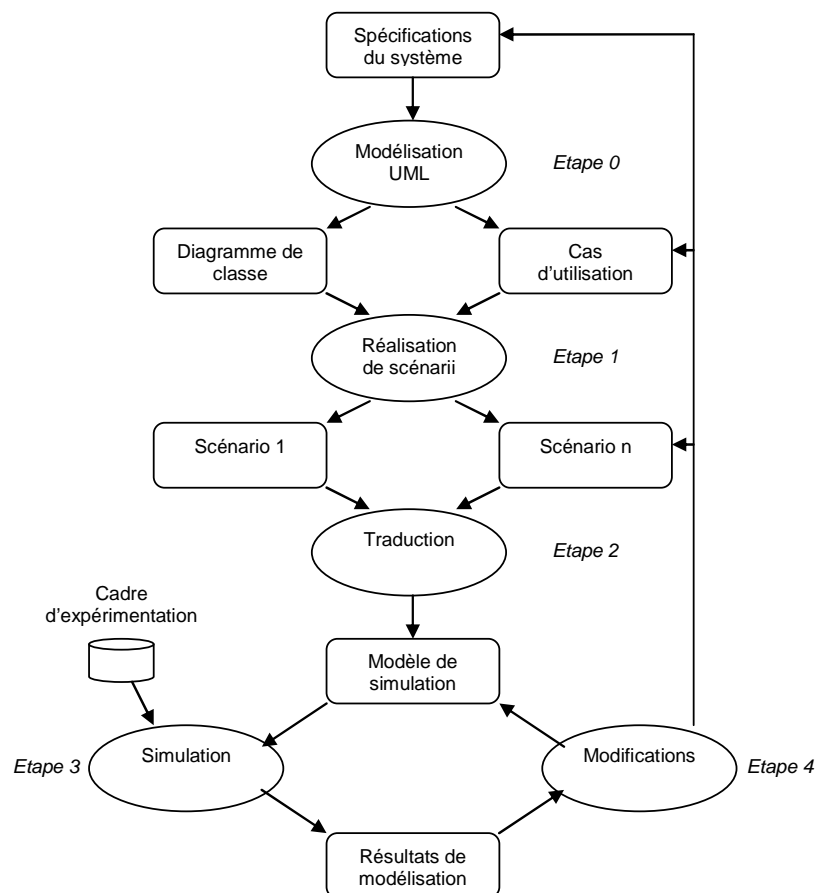


Figure 23. Méthode de passage de UML à la simulation. (Extraite de [79])

Sans détailler les concepts de la simulation (abordés dans le chapitre suivant), nous pouvons d'ores et déjà caractériser un modèle de simulation comme un ou plusieurs flux d'entités qui exécutent des processus permettant de modifier l'état du système. La démarche de passage d'un modèle UML à un modèle de simulation contribue à l'identification des processus et des entités associées.

La figure suivante explicite les principes de traduction d'un scénario UML en un modèle de simulation correspondant à l'étape 2 de la démarche (Figure 27).

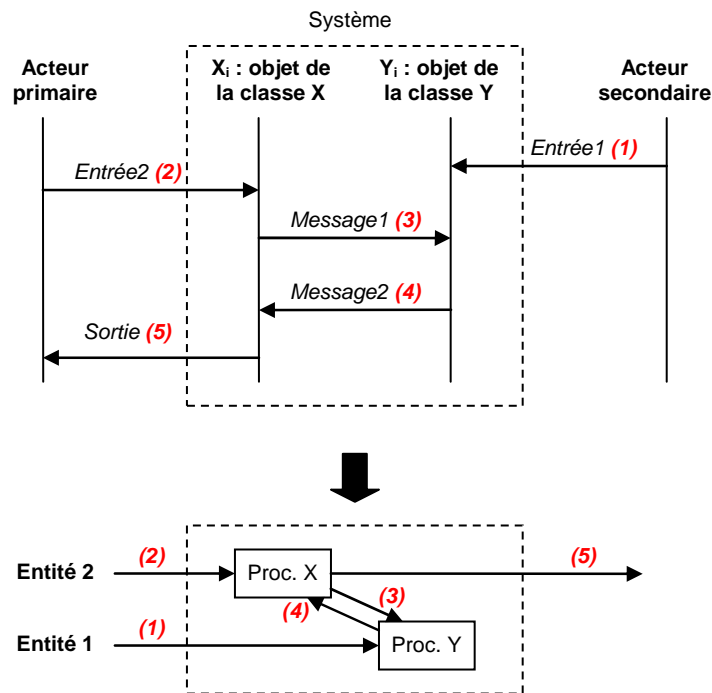


Figure 27. Le diagramme de séquence et le modèle de simulation associé.
(Extraite de [79])

L'acteur primaire est généralement trouvé lors de la définition d'un use-case : il s'agit de l'utilisateur du système. L'acteur secondaire, s'il existe, a un rôle administratif dans le système : il prépare le système afin qu'il soit utilisable par l'acteur primaire. Dans le diagramme de séquence, les instances d'objet peuvent être des logiciels, des machines ou des humains qui agissent au sein du système pour accomplir des tâches. La chronologie des événements présentés sur ce diagramme de séquence est :

1. L'acteur secondaire prépare le système avant l'arrivée de l'acteur primaire ('Entrée1').
2. L'acteur primaire interagit avec le système ('Entrée2').
3. X_i , instance d'objet de la classe X, envoie l'événement interne 'Message1'.

4. Y_i , instance d'objet de la classe Y, envoie l'événement interne 'Message2'.
5. Le système renvoie l'événement 'Sortie'.

Pour définir le modèle de simulation correspondant, Gertosio et al appliquent les règles suivantes :

- Chaque acteur est transformé en un type d'entité. Ainsi, l'acteur primaire du diagramme de séquence devient l'entité 2 du modèle de simulation et l'acteur secondaire devient l'entité 1.
- Chaque objet de classe est transformé en un processus. Ainsi, X_i devient une unité de ressource du processus X et Y_i devient une unité de ressource du processus Y.
- Les événements d'entrée correspondant au début de l'exécution de la simulation et deviennent ainsi les points d'entrée du modèle de simulation pour chaque entité.
- Chaque événement interne correspond à un événement généré par l'exécution du processus par une entité. L'exécution de l'action associée à un événement reflète l'exécution de la simulation sur l'ordinateur.
- L'événement de sortie correspond à la sortie de l'entité du modèle de simulation.

Dans notre scénario représentatif du traitement d'un ordre de fabrication par le SAI, l'acteur primaire est l'ordre de fabrication et les objets utilisés sont les contrôleurs. L'application de cette démarche sur notre scénario nous apporte principalement les éléments suivants :

- L'ordre de fabrication devient une entité du modèle de simulation. L'arrivée d'un ordre de fabrication constitue un point d'entrée du modèle de simulation. La fin de traitement d'un ordre de fabrication constitue un point de sortie du modèle de simulation.
- Les contrôleurs deviennent des processus à exécuter. L'entité ordre de fabrication doit s'emparer d'une unité de ressource d'un processus pour lancer le traitement associé. Les événements entre contrôleurs sont représentatifs de cette exécution.

Cependant, il ne faut pas oublier que nous n'avons modélisé, avec le formalisme UML, que le flux informationnel du SAI. Or les besoins de simulation nous obligent à modéliser également les flux physiques afin de mesurer l'impact du SAI sur l'unité opérationnelle et de mettre en évidence les interactions entre le SAI et l'unité opérationnelle. Dans ce cas, le produit est aussi considéré comme une entité traversant le modèle physique de l'unité et l'arrivée d'un produit constitue donc le point d'entrée véritable du modèle de simulation. Le cheminement

de cette entité à travers le modèle physique est contrôlé par le SAI composé des différents contrôleurs. Dans la modélisation objet, il nous a paru relativement difficile de représenter réellement le caractère distribué de notre SAI. Nous nous attachons, dans notre étude de simulation, à modéliser le couplage SAI/Unité et à formaliser au mieux les échanges internes inhérents à la répartition des fonctions de décision.

4. Simulation du système

Pour évaluer le comportement du SAI au cours du temps, il est nécessaire de disposer d'une représentation du SAI couplé au système à piloter. La simulation doit nous aider dans cette démarche. En première approche, nous avons identifié deux flux : le flux informationnel composé des ordres de fabrication et le flux physique composé des produits.

L'étude de simulation nous apparaît nécessaire pour valider l'architecture choisie et mesurer la pertinence et la performance du SAI. Trois objectifs supplémentaires s'imposent :

- identifier clairement les flux d'informations à l'intérieur et à l'extérieur du SAI, et recenser les incohérences et les difficultés de communications potentielles;
- faire ressortir les « noeuds de décisions », c'est-à-dire les endroits où l'on va pouvoir établir des règles de décision et les changer pour améliorer les performances du système ;
- mieux cibler nos cas d'application en jouant sur les paramètres afin de constater les impacts des temps de process, du nombre de transporteurs, de la spécialisation des transports...

4.1 De la modélisation à la simulation

La modélisation a pour but la création d'un modèle du système réel. Un modèle est une description du système qui représente une vue du système réel. Généralement, on distingue deux types de système :

- les systèmes physiques à états continus dont l'évolution se fait de manière continue dans le temps (ex. : processus de fusion).
- les systèmes à événements discrets dont l'évolution s'effectue à des instants discrets, les changements d'états étant déclenchés par des événements ponctuels (ex. : arrivée d'un client, achèvement d'un tâche).

L'évolution des systèmes physiques est usuellement décrite par des équations différentielles. L'évolution des systèmes à événements discrets est plus difficilement modélisable dans la mesure où peu de lois fondamentales sont disponibles, les lois d'entrée sont difficiles à évaluer et les événements aléatoires sont déterminants. La modélisation de leur comportement dynamique fait donc l'objet de nombreuses recherches.

Généralement, le comportement de tels systèmes est représenté à l'aide de formalismes mathématiques qui modélisent le processus sous la forme d'un réseau où sont décrites des règles de passage. Ces modèles doivent permettre la prise en compte des phénomènes de synchronisation et de concurrence (partage des ressources) engendrés par les événements. A cet effet, nous pouvons citer :

- les réseaux de Pétri introduits en 1962 et qui disposent depuis d'une multitude de variantes (réseaux de Pétri stochastiques, temporisés,...) [80];
- les réseaux à files d'attente ;
- les réseaux d'activités généralisés GAN introduits en 1964 par Elmaghraby [81] dont découle le modèle GERT (Graphical Evaluation and Review Techniques) de Pritsker (1966).

Ces modèles mathématiques, souvent dotés d'aspects stochastiques, sont à la base des techniques d'évaluation de performance et de simulation. Le modèle Q-GERT de Pritsker [82], développé en 1979, est d'ailleurs considéré comme un modèle assez général de simulation.

En fait, le processus de modélisation d'un système à l'aide de la simulation a pour but de définir un modèle discret utilisable sur un ordinateur dans le but d'évaluer le comportement du système.

La modélisation par simulation permet de modéliser de manière relativement simple des systèmes complexes. Elle est essentielle au niveau des unités de production et de services car elle constitue un puissant outil d'analyse capable d'apporter des éléments d'informations concernant leur fonctionnement, leur dimensionnement, leur capacité de réaction face à des aléas. Aujourd'hui, la simulation de systèmes est largement employée en production industrielle mais elle s'applique aussi à d'autres domaines très divers comme l'organisation de services (banques, hôpitaux), la circulation routière (simulation de carrefours) ou encore l'aéronautique (simulateur de vol ...).

4.2 Les concepts de la simulation

Développées par Zeigler à partir de 1976, les techniques d'évaluation de performances par la simulation peuvent prévoir le comportement de systèmes complexes. En fait, l'exécution du modèle de simulation permet d'évaluer et de raffiner la modélisation dépendant d'un

ensemble d'objectifs individuels ou combinés. Le processus de modélisation d'un système à l'aide de la simulation a pour but de définir un modèle discret utilisable sur un ordinateur. Le langage de simulation permet de traduire ce modèle dans une forme acceptable et exécutable par un ordinateur (ex. Arena, Witness,...). Zeigler définit les cinq éléments qui déterminent l'approche de base pour la modélisation par simulation [83] :

- Le système réel est en général trop complexe pour être modélisé tel quel. Il est caractérisé par un jeu de variables d'entrées (le segment d'entrée) et un jeu de variables de sortie (le segment de sortie). L'ensemble de toutes les combinaisons de paires d'entrée-sortie possibles donne le comportement d'entrée-sortie du système réel.
- Le cadre expérimental réunit un ensemble limité de circonstances sous lesquelles le système réel doit être observé. Il correspond à un sous-ensemble du comportement du système réel.
- Le modèle de base fournit une explication complète du comportement du système réel. Il s'agit d'un modèle capable de représenter tous les comportements du système réel. Il n'est jamais possible de connaître entièrement le modèle de base. Dans les cas réels, la grande complexité du modèle de base ne permet pas de développer le modèle de simulation correspondant.
- Le modèle simplifié (lumped model) est un modèle relativement simple valable dans le cadre expérimental. Il peut être codé en employant des langages de programmation différents pour être exécuté par un ordinateur.
- Le ordinateur est employé pour produire les paires d'entrée-sortie du modèle simplifié. Ce processus est appelé simulation du système.

Ainsi la création d'un modèle de simulation passe d'abord par la définition d'un cadre expérimental puis par la production du modèle simplifié relatif à ce cadre.

4.2.1 Les modèles à files d'attente

Le modèle de simulation est le plus souvent basé sur des modèles à files d'attente [84]. Un modèle à files d'attente comporte des serveurs et des entités. Le système est alors vu comme un ensemble de serveurs qui exécutent des traitements pour le compte des entités. Il est possible de définir un ou plusieurs types d'entités, chaque type d'entité constituant un flux qui traverse le modèle utilisant les ressources. Le serveur est caractérisé par plusieurs variables dont la plus importante est son état (libre ou occupé). Lorsque le serveur est occupé par une

entité et qu'une nouvelle entité a besoin d'utiliser ce serveur, cette dernière est stockée dans la file d'attente du serveur en attendant qu'il se libère (Figure 24).

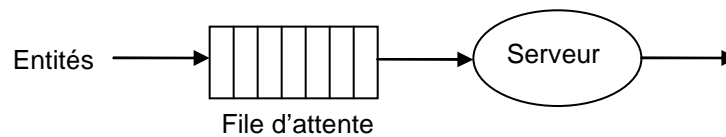


Figure 24. Principe d'un modèle à files d'attente.

Les langages de simulation permettent de créer aisément le modèle de simulation d'un système réel à partir d'une vue sous la forme d'un modèle à files d'attente.

4.2.2 Le processus de simulation

Le processus de simulation a pour but de créer un modèle de simulation le plus simple possible, qui représentera une vue du système et reproduira le problème posé. Il n'est pas toujours facile de définir les limites du système à modéliser. Aussi, l'approche vise à fixer un but au modèle que l'on construit, les limites de modèle et son niveau de détail, ainsi que les indicateurs de performance et les différentes alternatives à étudier. L'approche itérative de développement d'un modèle de simulation comprend neuf étapes (Figure 25) :

1. Formuler le problème : il s'agit de définir le problème à étudier avec les objectifs à atteindre.
2. Construire le modèle à files d'attente : il s'agit de créer une abstraction mathématico-logique du système. Un modèle est en effet une description du système mais aussi une abstraction de ce système.
3. Acquérir des données en vue de l'objectif fixé : il s'agit d'identifier puis de collecter les données nécessaires.
4. Traduire le modèle : il s'agit d'utiliser un langage qui permet de porter le modèle sur ordinateur.
5. Valider le modèle : il s'agit, à partir de tests et de mises au point du programme, de vérifier que le modèle se comporte comme le système réel.
6. Etablir un planning stratégique : il s'agit d'établir des conditions expérimentales de test.
7. Expérimenter le modèle : il s'agit d'exécuter le modèle pour obtenir des résultats.

8. Analyser les résultats obtenus précédemment.
9. Implanter le modèle et fournir une documentation : il s'agit de permettre à des utilisateurs d'utiliser le modèle pour résoudre leurs problèmes. Un modèle n'a pas atteint son objectif tant qu'il ne constitue pas une aide à la décision pour ses utilisateurs.

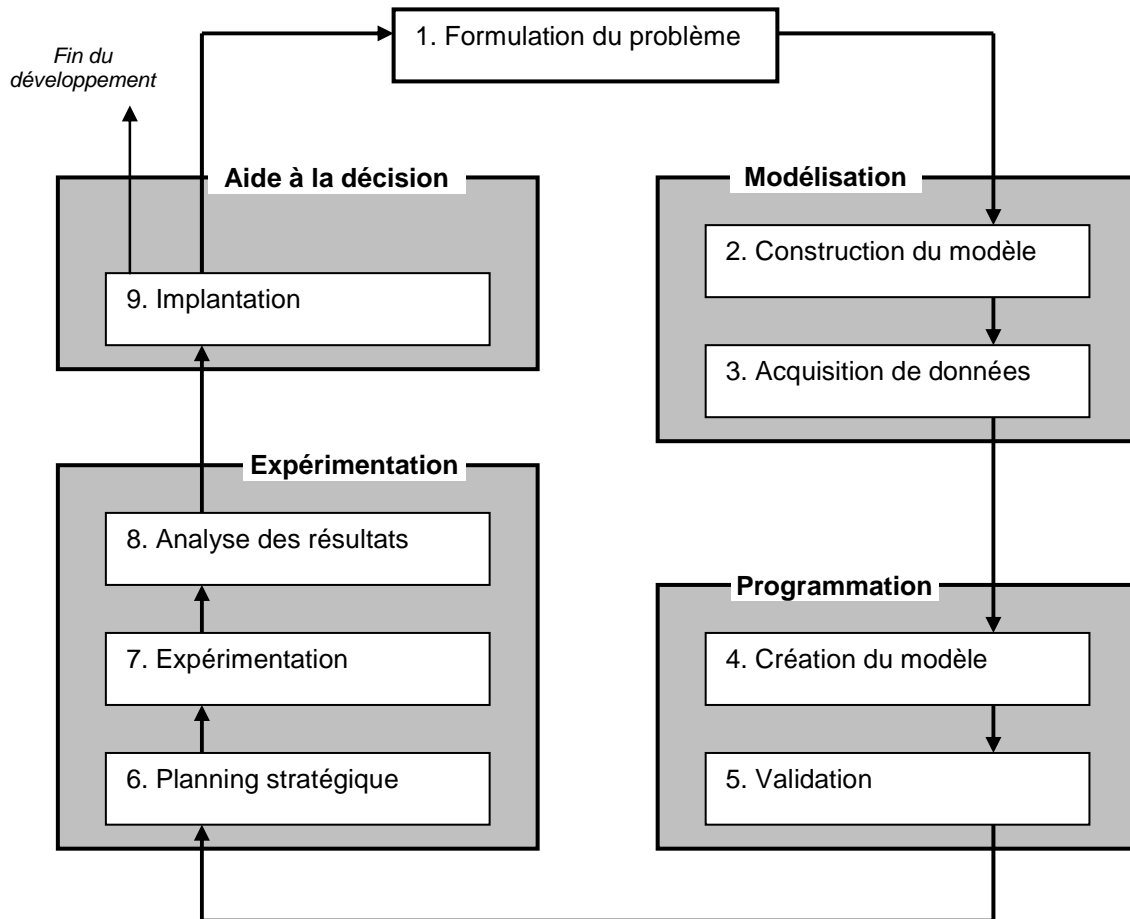


Figure 25. Processus de simulation. (Extraite de [84])

Pour la conception d'un modèle de simulation, trois éléments sont à prendre en compte: les états, les événements et le temps. Les états et les événements sont duaux dans le sens où un changement d'état dans un système apparaît comme le résultat d'une arrivée d'événement.

- Un état décrit le système durant un intervalle de temps.
- Un événement est un point dans le temps qui désigne un changement d'état. C'est une sorte d'état instantané qui n'a pas de durée.
- Deux méthodes de gestion du temps peuvent être utilisées : le découpage du temps (time slicing) consistant à gérer le temps comme une horloge virtuelle mise à jour

à intervalle régulier, ou, le plus souvent, l'organisation par événement (event scheduling) définissant une boucle sur événement où les événements sont placés puis testés selon un intervalle de temps minimum.

4.2.3 La simulation de systèmes distribués

Pour produire le cadre expérimental d'un modèle de simulation, il est nécessaire d'identifier les événements que le système échange avec son environnement. Quand le système est distribué, l'identification de ces événements est plus difficile. En effet, chaque partie du système global peut être considérée comme un système indépendant qui échange des événements, aussi bien avec son environnement qu'avec un autre système indépendant.

Il existe deux approches possibles pour la simulation de systèmes distribués :

- Le système est pris comme un ensemble et l'outil de simulation doit rendre transparentes les communications entre les diverses parties du système.
- Les communications entre les diverses parties du système sont importantes et l'outil de simulation doit permettre leurs modélisations.

La première approche correspond à la simulation distribuée, qui permet de distribuer la simulation sur plusieurs ordinateurs en réseau. Chaque partie du système distribué est modélisée indépendamment. L'environnement de simulation distribuée crée les conditions autorisant tous les modèles à communiquer entre eux [85]. Par conséquent, les communications réseau entre les différents modèles sont transparentes. HLA (High Level Architecture) proposée en 1996 par l'American DoD (Department of Defence) s'impose comme un standard dans la simulation distribuée. La norme HLA est devenue une norme non militaire à travers la norme IEEE1556. HLA repose sur un système nommé RTI (Real Time Infrastructure) qui régit la simulation dans sa totalité : nom d'objets, etc. RTI gère également les couches inférieures de l'architecture c'est-à-dire la communication d'objets entre les ordinateurs et fournit un service d'application à l'utilisateur final. Il existe plusieurs implémentations de RTI. DoD en a donné une il y a quelques années. Certaines d'entre elles, basées sur C++ ou ADA, sont libres et les autres, basées sur CORBA sont commerciales [86]. HLA offre des spécifications détaillées de mise en oeuvre de simulateurs dans des environnements distribués et hétérogènes de traitement de données et intègre la majorité des algorithmes connus et reconnus dans ce domaine. La simulation distribuée convient bien à la modélisation de systèmes physiquement distribués tels que, par exemple, les systèmes constitués de troupes mobiles (tanks, avions). Les domaines principaux d'application de la

simulation distribuée sont les calculs parallèles, le prototypage numérique, et l'intelligence artificielle pour l'industrie militaire et automobile.

La deuxième approche conduit à se servir de la simulation traditionnelle pour créer un modèle unique composé d'un ensemble de modèles indépendants. Cette approche est employée pour modéliser les systèmes distribués qui sont formés de sous-systèmes autonomes. Un sous-système est indépendant des autres sous-systèmes dans le sens où il peut exécuter des ordres mais reste responsable de ses décisions. La simulation vise alors à étudier à la fois les sous-systèmes eux-mêmes et les communications entre ces sous-systèmes. Il est supposé que les sous-systèmes communiquent ensemble par le biais d'événements discrets. Sarjoughian et Cellier [87], ainsi que Gonzalez [88], ont défini des environnements appropriés pour ce type de démarche.

Etudier la complexité des échanges du SAI couplé à une unité opérationnelle revient à utiliser la seconde approche, puisque la première rend transparentes les communications au sein des systèmes distribués. Nous développons ici un cadre méthodologique de simulation apparenté à cette approche [90][91] : il doit permettre la modélisation d'un système de pilotage réactif distribué associé à son unité opérationnelle⁶.

4.3 Une méthodologie de simulation pour les systèmes de pilotage réactifs distribués

Nous souhaitons établir un modèle générique de simulation pour les systèmes de pilotage composés de sous-systèmes autonomes. Le principe général consiste à établir un sous-modèle de simulation pour chaque sous-système et à faire communiquer chaque sous-modèle dans un modèle unique.

Il convient de préciser que nous avons utilisé le logiciel de simulation Arena. Nous en faisons une brève présentation avant de passer au modèle de simulation proprement dit.

4.3.1 Le logiciel de simulation Arena

Il existe sur le marché plusieurs logiciels de simulation permettant de créer et de mettre en œuvre des modèles de simulation. Ces logiciels sont plus ou moins performants en fonction

⁶ Ces travaux ont été présentés à la conférence internationale IEEE ETFA'05 et font l'objet d'un article dans la revue International Journal of Factory Automation, Robotics and Soft Computing (2006).

de la nature du cas à étudier. Nous pouvons citer Arena, Sirphyco, Cadence, QNAP, Automod, ARIS ToolSet.

Spécialement dédié à la modélisation et à la simulation des systèmes de production et très répandu dans le monde industriel, le logiciel Arena (version 7/8) nous semble le plus adapté à notre problème. Ce logiciel, s'appuyant sur la simulation à événements discrets, permet de créer des modèles de files d'attentes [89]. Selon la terminologie propre aux modèles à file d'attente, le concept de serveur avec file d'attente est matérialisé sous Arena comme une ressource munies de caractéristiques (règles d'attribution, états, taux d'occupation...) et les entités sont directement représentées sous forme d'entités munies d'attributs. Un modèle Arena est constitué par un ensemble de processus qui interagissent entre eux. Un processus est une séquence de transformations (représentée par une suite de blocs) que subit une entité pendant sa traversée du système. Une entité est alors un objet dont l'état est modifié par le processus qu'il traverse. Les changements d'états nécessitent principalement l'utilisation de ressources. L'utilisation d'une ressource par les entités inclut trois étapes :

1. Les entités sont stockées dans la file d'attente de la ressource (bloc QUEUE) jusqu'à ce qu'une unité de la ressource se libère.
2. Une unité de la ressource est réservée pour le traitement de l'entité (bloc SEIZE).
3. Une fois le traitement terminé, cette unité est libérée (bloc RELEASE).

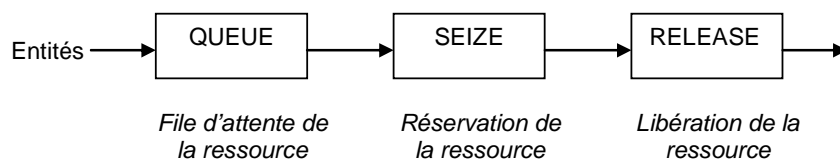


Figure 26. Réservation d'une ressource par une entité.

4.3.2 Le modèle générique de simulation

La méthodologie de simulation proposée a pour objectifs majeurs :

- la définition de squelettes génériques de modèle de simulation pour chaque type de sous-système présent dans un système distribué de pilotage réactif à architecture hybride (sans lien de coopération);
- la création d'un modèle de simulation unique reliant le modèle du système de conduite à celui de l'unité opérationnelle.

Elle envisage la construction du modèle du système global comme l'assemblage d'une collection de modèles de sous-systèmes indépendants. Ainsi, le modèle unique d'un système de pilotage distribué (à architecture hybride) réunit le modèle du contrôleur central, les modèles des différents contrôleurs de ressources et le modèle du système physique (Figure 28). Nous distinguerons ici deux types contrôleur de ressources : un contrôleur assimilable à un contrôleur de ressources statiques et un contrôleur assimilable à un contrôleur de ressources de transfert.

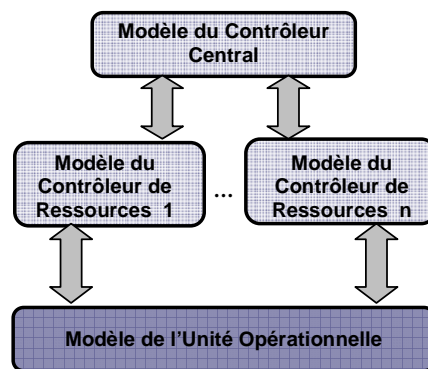


Figure 28. Modèles de simulation du système de pilotage.

La conception du modèle système entier implique que chaque produit traité par l'unité soit représenté par une entité appelée 'Entité Produit'. Cette entité est liée au flux physique du système. Une entité appelée 'Entité Ordre de Fabrication' doit également être créée. Nous avons en effet identifié auparavant la nécessité d'une telle entité reflétant le flux informationnel du système. En réalité, lorsqu'une Entité Produit (Entité_P) arrive dans le modèle, elle est dupliquée (grâce à un bloc DUPLICATE) : l'entité originale entre dans le modèle de l'unité opérationnelle, tandis que la copie, correspondant à l'Entité Ordre de Fabrication (Entité_OF) entre dans le modèle du système de pilotage et plus précisément dans le modèle du contrôleur central (Figure 29). Cette copie sert à superviser le traitement de son Entité Produit associée. Les entités de type Produit caractérisent le cheminement du produit à travers l'unité opérationnelle et les entités de type Ordre de Fabrication caractérisent les données de pilotage relatives à ce cheminement. Le traitement ces deux entités, distinctes mais dépendantes, permet de réaliser de manière effective le couplage système de pilotage / unité opérationnelle.

Quand une entité est dupliquée, elle hérite des attributs de l'entité originale (en particulier ici l'identification du produit 'no_produit'). Une fois le produit traité, l'entité Produit et l'entité

Ordre de Fabrication correspondante doivent être synchronisés (bloc MATCH sur l'attribut 'no_produit') avant destruction.

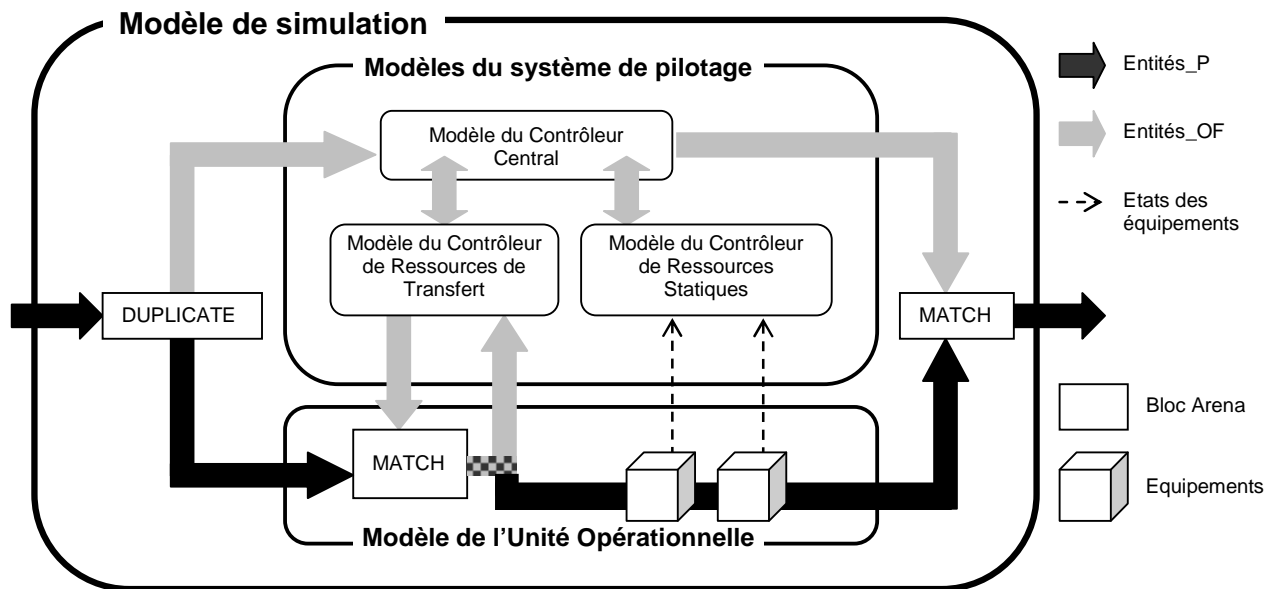


Figure 29. Duplication des entités Produit et Ordre de Fabrication.

- Le modèle de simulation du Contrôleur Central (CC)

Le Contrôleur Central reçoit un plan de tâches, généré en amont par une application externe, sous la forme d'une liste de produits à traiter. Ces produits possèdent des contraintes d'ordre temporelles relatives aux objectifs de production. La notion de priorité des produits est adoptée ici pour traduire ces contraintes. Dans le modèle de simulation, ce plan de tâches est modélisé comme l'arrivée d'entités de type Ordre de Fabrication en charge du traitement des produits réels associés. Les entités Ordre de Fabrication sont stockées dans une file d'attente (bloc QUEUE) qui dispose de règles spécifiques de gestion. Ainsi, la file peut par exemple ordonner préalablement les entités Ordre de Fabrication par niveau de priorité : l'entité Ordre de Fabrication ayant la plus haute priorité est traitée en premier.

Le Contrôleur Central est défini comme une ressource (au sens Arena) à capacité finie représentant sa capacité à traiter les produits simultanément. De ce fait, la définition d'une capacité de trois unités signifie que l'exécution du processus n'est possible que pour trois produits seulement. Pendant ce temps, les autres entités Ordre de Fabrication attendent dans la file qu'une unité de la ressource Contrôleur Central soit libérée par une entité dont le traitement est terminé. Chaque entité Ordre de Fabrication possède une gamme opératoire,

c'est-à-dire la séquence des opérations à réaliser par l'unité opérationnelle pour le produit associé.

Chacune des opérations de la gamme est supervisée par un des contrôleurs de ressources du niveau inférieur. Dans une architecture de pilotage hybride, ces contrôleurs sont indépendants, dans la mesure où ils sont autonomes dans leurs prises de décisions. Dès qu'une requête provenant du niveau supérieur (Contrôleur Central) est exécutée, le contrôleur de ressources renvoie l'indication 'requête exécutée'. Pour modéliser cet échange, il est nécessaire d'utiliser l'entité 'Ordre de Fabrication'. Concrètement, pour chaque opération à réaliser, l'entité Ordre de Fabrication navigue du modèle du Contrôleur Central vers le modèle du Contrôleur de Ressources (modèle CR) concerné (Figure 30). A partir de cette entité, le contrôleur de ressources peut gérer par exemple l'allocation de ressources pour l'opération voulue. Une fois la requête traitée, l'entité Ordre de Fabrication est retournée au Contrôleur Central avec la décision prise par le Contrôleur de Ressources.

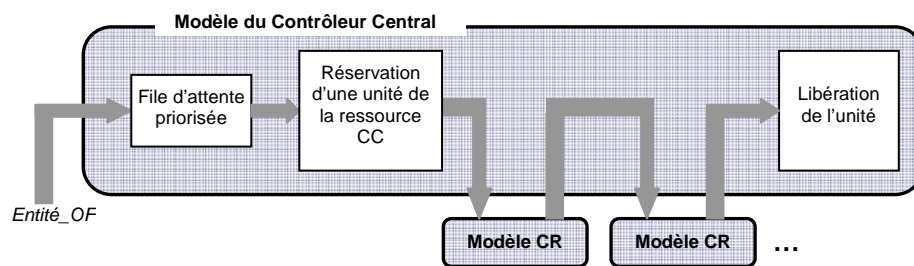


Figure 30. Modèle de simulation du Contrôleur Central.

- Les modèles de simulation des Contrôleur de Ressources

L'adoption d'une architecture de pilotage hybride (sans lien de coopération) implique que chaque Contrôleur de Ressources communique avec le Contrôleur Central, mais ne communique pas directement avec un autre Contrôleur de Ressources. Un Contrôleur de Ressources reçoit les requêtes du Contrôleur Central et lui retourne l'indication 'requête exécutée'. Chaque Contrôleur de Ressources est capable de prendre ses propres décisions en fonction des informations dont il dispose. Nous avons alors modélisé un Contrôleur de Ressources comme une ressource à capacité finie qui représente la possibilité de traiter plusieurs requêtes du Contrôleur Central simultanément. Tout comme pour le Contrôleur Central, une unité de la ressource Contrôleur de Ressources doit être réservée par l'entité Ordre de Fabrication transmise afin que ce dernier effectue une allocation appropriée en

rapport avec son environnement direct (Figure 31). Ceci est représentatif de la distribution de la décision. La modélisation relative à la décision d'allocation proprement dite dépend du type de ressources supervisées par le Contrôleur de Ressources.

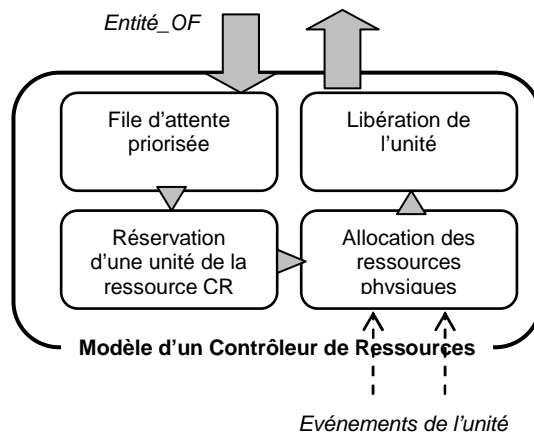


Figure 31. Modèle de simulation global d'un Contrôleur de Ressources.

Nous proposons ainsi deux modèles distincts pour les Contrôleurs de Ressources, étant donné que, sous Arena, le mode de gestion d'une opération de transport est différent de celui d'une opération sur équipement. Un premier contrôleur pourrait s'apparenter à un contrôleur de ressources de transfert et le second à un contrôleur de ressources de transfert.

- *Le modèle de simulation pour un Contrôleur de Ressources Statiques (CRS)*

Ce modèle est destiné à un Contrôleur de Ressources supervisant un groupe d'équipements nécessitant la présence d'opérateurs affectés de manière permanente ou d'équipements automatisés (équivalents aux ressources dites statiques). Un équipement de l'unité, ou l'opérateur affecté à cet équipement, est supposé capable d'envoyer son état actuel (libre, occupé, début_opération, fin_opération,...) au système de pilotage. Un Contrôleur de Ressources Statiques a en effet besoin de ces événements pour construire ses décisions. Ce processus de décision intervient dès lors qu'une entité Ordre de Fabrication a réservé une unité de la ressource Contrôleur de Ressources.

L'entité Ordre de Fabrication est alors employée pour simuler l'exploitation de ces événements 'état équipement' dans le système de pilotage. La réservation et la libération d'un équipement est réalisée par l'entité Ordre de Fabrication (suite de blocs QUEUE-SEIZE-...-RELEASE) au niveau du Contrôleur de Ressource (Figure 32). Cependant, au niveau de l'unité opérationnelle, l'équipement est en réalité utilisé par l'entité Produit. Le passage de l'entité Ordre de Fabrication à

l'entité Produit se fait par le biais d'une synchronisation effectuée lors du transport du produit sur l'équipement alloué. A la fin de l'opération, l'entité Produit et l'entité Ordre de Fabrication sont à nouveau désynchronisées ou plutôt dupliquées (bloc DUPLICATE) : l'entité Produit attend son prochain transport, tandis que l'entité Ordre de Fabrication retourne dans le système de pilotage pour simuler l'opération terminée.

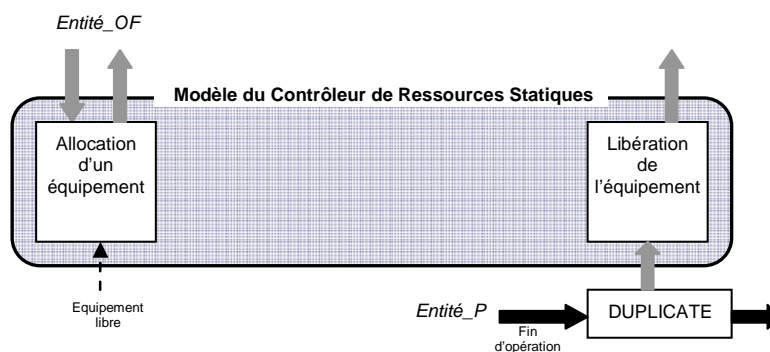


Figure 32. Modèle de simulation d'un CRS.

- *Le modèle de simulation pour un Contrôleur de Ressources de Transfert (CRT)*

Ce modèle est destiné à un Contrôleur de Ressources supervisant une flotte de transporteurs (équivalents aux ressources dites totalement mobiles). Durant le traitement d'un produit au sein de l'unité, le produit est amené à plusieurs reprises à attendre dans un emplacement donné (nommé 'Station de Départ') qu'il soit transporté vers un autre emplacement (nommé 'Station de Destination'). Les Stations de Départ ou les Stations de Destination peuvent être soit une zone de stockage, soit un équipement, soit la sortie de l'atelier. L'entité Produit reste dans la Station de Départ jusqu'à ce que le produit soit chargé et transporté vers la Station de Destination.

En attendant, son entité Ordre de Fabrication correspondante essaie de réserver un transporteur libre pour exécuter le transfert du produit (bloc REQUEST) (Figure 33). Dès qu'elle obtient un transporteur, celui-ci doit tout d'abord se déplacer de son emplacement actuel vers la Station de Départ (bloc MOVE). L'entité Produit peut alors être transférée vers la Station de Destination. L'entité Ordre de Fabrication devient, pour le passage dans l'unité, son entité Produit associée et ceci nécessite donc une synchronisation entre les deux entités dans la Station de Départ. Les deux entités restent synchronisées pendant toute la durée du transport.

Le langage de simulation Arena [92] exige que l'entité qui a demandé le transporteur l'utilise et le libère ensuite. Une fois que le transporteur arrive dans la Station de Destination, les deux entités sont séparées : l'entité Produit poursuit son processus, tandis que son entité Ordre de Fabrication libère le transporteur puis retourne dans son Contrôleur de Ressources créant un événement 'transport effectué'.

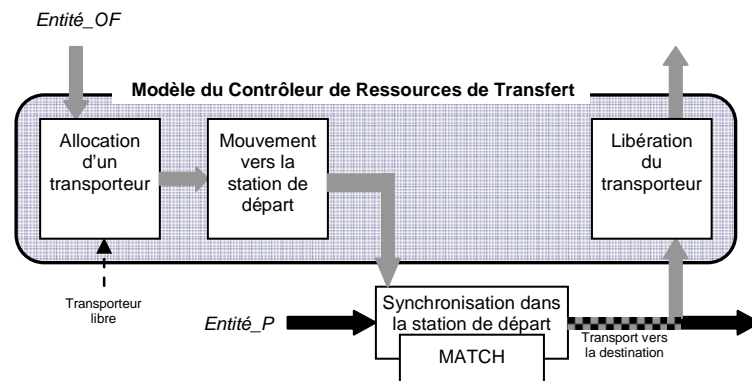


Figure 33. Modèle de simulation d'un CRT.

Pour couvrir tous les besoins du SAI, il est important de préciser que le modèle de simulation d'un Contrôleur de Ressources Mobiles se rapproche fortement de celui d'un Contrôleur de Ressources Statiques, à la différence près qu'il faut aussi réserver un opérateur dans la phase d'allocation de ressources physiques. En effet, un Contrôleur de Ressources Mobiles supervise en ensemble d'opérateurs appartenant à un ensemble d'équipements (nombre d'opérateurs inférieur au nombre d'équipements). En simulation Arena, la réservation et la libération d'un opérateur se fait de la même façon que la réservation d'un équipement (SEIZE puis RELEASE). L'entité Ordre de Fabrication doit alors, après avoir réservé un équipement, réserver un opérateur parmi un groupe d'opérateurs.

En résumé, simuler l'action d'un système de pilotage réactif distribué à architecture hybride consiste principalement à :

- distinguer l'entité Ordre de Fabrication de l'Entité Produit associée : le pilotage de l'unité est alors effectué par des mécanismes de synchronisation/désynchronisation des deux entités ;
- considérer les différents contrôleurs comme des ressources : la réservation d'une unité d'un Contrôleur de Ressources par un Ordre de Fabrication, afin que celui-ci gère les allocations, représente la décentralisation de la décision.

4.4 Application : création du modèle de simulation du SAI correspondant à une unité industrielle existante

Cette étude préliminaire nous sert de base pour simuler le SAI et étudier son application à une unité opérationnelle quelconque (unité à flux discret). Nous prenons, comme unité référence, l'unité de réglages moteurs où se sont révélés des problèmes d'organisation en terme de tâches (cf introduction). Il s'agit d'une unité industrielle traitant le réglage de moteurs diesels pour les camions. Cette unité est intéressante car elle illustre toutes les catégories de décision des opérateurs en ce qui concerne leurs choix pour l'exécution d'une opération. Elle regroupe donc tous les types de ressources définis pour une unité (ressources statiques, ressources semi-mobiles et ressources totalement mobiles) et est, en ce sens, représentative de notre problématique.

4.4.1 Présentation de l'unité

L'unité opérationnelle que nous étudions gère le réglage de moteurs diesels pour les camions, à hauteur d'environ 300 moteurs par jour. Le processus de réglage se déroule en trois étapes :

1. La préparation du moteur : celui-ci est rempli d'huile, d'eau et de diesel. Pour les branchements ultérieurs, le moteur est également doté de connecteurs. Cette opération s'effectue sur un des 6 Bancs Auxiliaires (BA) et nécessite l'intervention d'un opérateur. Elle a une durée totale de 15 minutes, dont 5 minutes avec opérateur.
2. Le réglage du moteur : celui-ci est vérifié et réglé. Cette opération complètement automatisée s'effectue sur un des 16 Banc de Régulation (BR). Elle a une durée totale de 25 minutes.
3. La « dépréparation » du moteur : celui-ci est vidé de l'huile, de l'eau et du diesel qu'il contient. Les connecteurs sont également enlevés. Cette opération s'effectue sur un des 6 Bancs Auxiliaires et nécessite l'intervention d'un humain. Elle a une durée totale de 15 minutes, dont 5 minutes avec opérateur.

Les moteurs réglés sont ensuite placés dans le stock de sortie de l'unité. Précisons que les moteurs proviennent de quatre lignes de production situées en amont, et sont placés dans le stock d'entrée avant d'être réglés.

Les opérations annexes (1 et 3) ont été mises en place pour optimiser l'utilisation des machines de réglage en elle-même coûteuses. Un travail de réorganisation a effectivement eu

lieu dans cette unité opérationnelle et a abouti à l'installation de 6 bancs de préparation et de 6 bancs de « dépréparation » autour des 16 bancs de réglage moteur [93].

Cependant, ce type d'organisation soulève un problème non négligeable : l'implication et la responsabilité de l'humain dans le bon déroulement du processus. Ceci se retrouve tout d'abord au niveau de la manutention, tâche centrale qui dirige en fait le départ d'une opération puisque qu'une opération de préparation / réglage / dépréparation ne peut bien évidemment pas débiter sans le moteur. Le nombre de caristes étant inférieur au nombre de trajets possibles, ceux-ci doivent au mieux organiser leurs transports en fonction des moteurs à régler. De même, il faut noter que seuls 3 ou 4 opérateurs sont présents pour le travail sur bancs auxiliaires, d'où la nécessité de bien gérer leurs tâches. Cette contrainte a été confirmée par une étude de simulation : les résultats obtenus dans les conditions réelles, bien que supérieurs à ceux de la précédente organisation, sont bien moins bons que ceux escomptés par la simulation. Il est important de préciser que l'automatisation complète des tâches annexes est non envisageable car, en plus d'être coûteuse et complexe, elle aurait plutôt desservi l'unité opérationnelle en terme de flexibilité de production.

La planification des tâches restant donc dédiée aux opérateurs, on comprend rapidement la nécessité de proposer une solution pour les guider dans leurs tâches quotidiennes. Nous notons qu'un système de feux a déjà été instauré afin de permettre aux opérateurs distants de connaître l'état d'une station (7 états différents dont « libre », « occupée »,...) (Figure 34). Pour compléter ce dispositif et répondre aux problèmes d'organisation évoqués ci-dessus, nous proposons d'implanter un SAI au-dessus de cette unité, ce qui permettrait aux opérateurs de savoir à tout moment ce qu'ils doivent faire.



Figure 34. Système de feux. (Extraite de [93])

Pour nos études ultérieures, nous choisissons de simplifier cette unité en ramenant le nombre de Bancs Auxiliaires à 3 pour chaque opération annexe et le nombre de Bancs de Réglage à 6 (Figure 35). 2 opérateurs assurent l'opération de préparation et 2 opérateurs assurent l'opération de dépréparation (que nous appelons par la suite 'vidange'). 4 opérateurs de transport disposent chacun de son propre chariot élévateur pour déplacer les moteurs à travers toute l'unité.

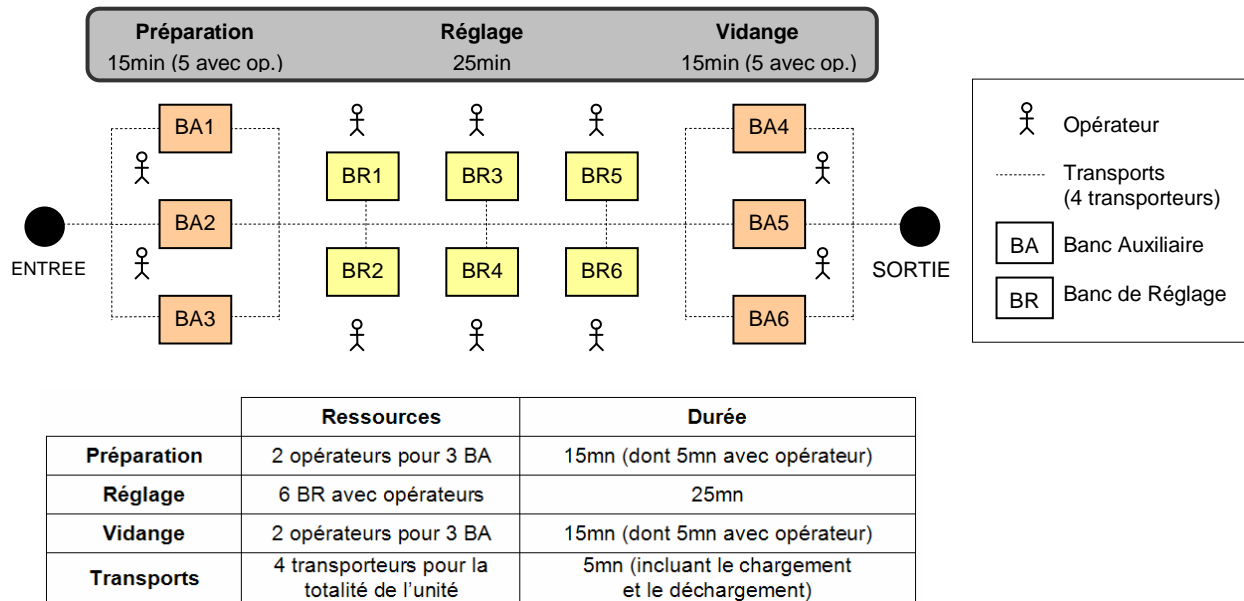


Figure 35. Schéma simplifié de l'unité de réglages moteurs.

4.4.2 Le SAI associé à l'unité de réglages moteurs

La réalisation d'un ordre de fabrication, correspondant ici au processus de réglage d'un moteur, conduit à sept opérations :

- une opération de transport du stock vers un des BA de préparation ;
- une opération de préparation exécutée sur le BA ;
- une opération de transport du BA vers un des BR ;
- une opération de réglage exécutée sur le BR ;
- une opération de transport du BR vers un des BA de vidange ;
- une opération de vidange exécutée sur le BA ;
- une opération de transport vers le stock de sortie.

L'opération de préparation, nécessitant un opérateur, est semi-automatique. Le nombre d'opérateurs est inférieur au nombre de BA et un opérateur de préparation a donc à choisir à

sur quel BA agir. L'ensemble des opérateurs de préparation constitue alors un ensemble de ressources semi-mobiles et doit, dans le cadre du SAI, être supervisé par un Contrôleur de Ressources Mobiles. Il est appelé 'CRM Préparation'. En fait, le contrôleur gère ici les équipements et les opérateurs.

L'opération de vidange suit le même principe. Dans le cadre du SAI, un Contrôleur de Ressources Mobiles est donc nécessaire. Il est appelé 'CRM Vidange'.

L'opération de réglage est totalement automatisée. Un opérateur est affecté à la surveillance d'un BR et n'a donc pas à choisir sur quel BR agir. L'ensemble des opérateurs de réglage constitue alors un ensemble de ressources statiques et doit, dans le cadre du SAI, être supervisé par un Contrôleur de Ressources Statiques. Il est appelé 'CRM Réglage'. En fait, le contrôleur ne gère ici que les équipements.

Ces trois opérations induisent quatre opérations de transport. Les transferts sont assurés par des conducteurs de chariots élévateurs, logiquement considérés comme des ressources totalement mobiles, couvrant un ou plusieurs secteurs. Il est possible de définir un ou plusieurs Contrôleurs de Ressources de Transferts (CRT) selon la couverture de la flotte de transporteurs : un CRT est créé pour superviser une flotte de transporteurs et un ensemble de chemins couverts par ces transporteurs. Par conséquent, un seul CRT est créé si les quatre secteurs sont pris en charge par une seule flotte de transporteurs. Dans le cas contraire, deux, trois voire quatre CRT sont créés. Ainsi, il est concevable d'obtenir 'CRT 1' pour le secteur Entrée-Préparation, 'CRT 2' pour le secteur Préparation-Réglage, 'CRT 3' pour le secteur Réglage-Vidange et 'CRT4' pour le secteur Vidange-Sortie.

Au total, nous obtenons pour le SAI de quatre à sept Contrôleurs de Ressources coordonnés par un Contrôleur Central (Figure 36).

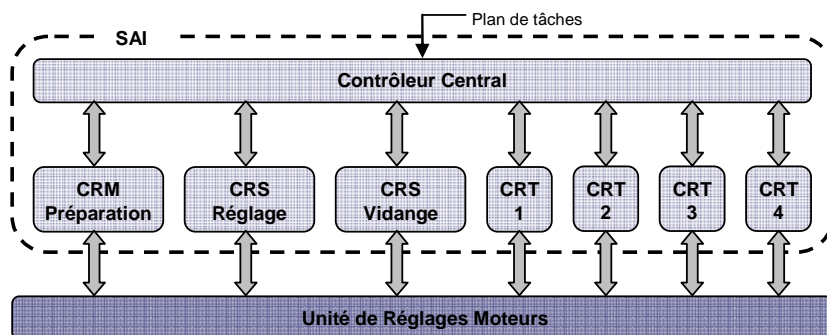


Figure 36. Le SAI de l'unité de réglages moteurs.

Un des intérêts majeurs de l'implantation du SAI sur une telle unité serait de suggérer, en temps réel, les trois tâches de transport les plus prioritaires aux conducteurs de chariot

élévateur, ainsi que les trois tâches de préparation (ou de vidange) les plus prioritaires aux opérateurs de préparation (ou de vidange). Il doit permettre de concilier une utilisation optimale des équipements de réglage avec les contraintes des différents opérateurs.

4.4.3 Simulation du SAI couplé à l'unité de réglages moteurs

Il s'agit de créer le modèle de simulation correspondant à l'unité opérationnelle définie précédemment. Dans un premier temps, nous partons d'une situation plus restreinte, le but étant de déterminer les points primordiaux du système et de se focaliser plus particulièrement sur les échanges internes au SAI. Le modèle pourra ensuite être enrichi au fur et à mesure selon les objectifs souhaités. Ainsi, la phase de vidange n'existe pas (car similaire à une phase de préparation), seulement 3 BR sont fonctionnels et un Contrôleur de Ressources de Transfert est associé à chaque secteur (d'où l'utilisation de 3 CRT).

Pour construire le modèle, nous nous inspirons des concepts de la méthodologie de simulation de systèmes de pilotage distribués évoquée précédemment. Appliquée au cas du SAI :

- l'entité 'Produit' devient l'entité 'Moteur' et l'entité 'Ordre de Fabrication' devient l'entité 'Ordre de Réglage' ;
- les ressources, au sens Arena, du modèle de simulation du SAI sont le Contrôleur Central, le CRM Préparation, le CRS Réglage, le CRT 1, le CRT 2 et le CRT3 ;
- les ressources du modèle de simulation du système physique sont les BA, les BR, les opérateurs de préparation et les transporteurs sont les conducteurs de fenwicks.

Par suite, la représentation des flux physiques et informationnels conduit à sept sous-modèles indépendants basés sur différents squelettes en fonction du type de contrôleur. Le modèle global contient donc : le modèle du Contrôleur Central (squelette d'un modèle CC), le modèle du CRM Préparation (squelette d'un modèle CRM), le modèle du CRS Réglage (squelette d'un modèle CRS), les trois modèles des CRT (squelette d'un modèle CRT) pour le SAI et le modèle du système physique.

L'unité opérationnelle « réelle » est décrite très simplement dans le sous-modèle du système physique. Nous modélisons les BA et les BR comme des ressources et les moteurs sont alors transportés depuis la station Stock vers une des équipements BA, puis vers un des équipements BR. Après le réglage, les moteurs sont transportés par un chariot vers la station Sortie. Les entités traversant ce modèle sont les Entités Moteur (Entité_M).

La partie décisionnelle de notre unité est représentée à travers le sous-modèle du SAI composé lui-même de six sous-modèles qui constituent le cœur de notre SAI. En effet, à chaque moteur (représenté par l'Entité M dans le modèle de l'unité physique), est associée une entité Ordre de Réglage (Entité_OR) traversant le modèle du SAI et gérant ainsi l'aspect décisionnel. Les six sous-modèles du SAI symbolisent les six contrôleurs qui traitent les ordres de réglages.

- Une Entité_OR doit s'emparer d'une unité de la ressource Contrôleur Central pour que commence le traitement du moteur correspondant.
- Le CRM Préparation reçoit, à travers l'Entité_OR, les demandes d'allocations en provenance du Contrôleur Central et doit effectuer l'affectation des BA et des opérateurs de BA en fonction de leur disponibilité.
- Le CRS Réglage reçoit, à travers l'Entité_OR, les demandes d'allocations en provenance du Contrôleur Central et doit effectuer l'affectation des BR en fonction de leur disponibilité.
- Les CRT reçoivent, à travers l'Entité_OR, les demandes de trajets à réaliser et doivent gérer les affectations de transporteurs de leurs secteurs.

Le modèle complet de simulation est donné en annexe (cf Annexe 1). Dans l'animation relative à ce modèle (Figure 37), les communications entre sous-modèles du SAI sont représentées par des lignes numérotées de 1 à 10 qui correspondent aux messages échangés entre le Contrôleur Central et les différents Contrôleurs de Ressources. Les événements transmis entre l'Unité Industrielle et les Contrôleurs de Ressources sont représentés par des flèches numérotées de 1 à 3 : ils correspondent aux commandes du SAI dans le sens SAI-Unité et aux états des ressources dans le sens Unité-SAI.

Ces communications peuvent être illustrés sous la forme d'un scénario de simulation. Les échanges de messages 1 à 4 et l'événement 1 sont ici représentés (Figure 38). Ce scénario partiel détaille le transport d'un moteur M1 du Stock Entrée vers le BA1. Le transport est effectué par le conducteur de chariot CH2. Du point de vue du SAI, ce transport inclut une décision au niveau du Contrôleur Central, du CRM Préparation (responsable de BA1, BA2, BA3) et du CRT 1 (responsable de deux chariots élévateurs CH1 et CH2 sur le secteur Entrée-Préparation).

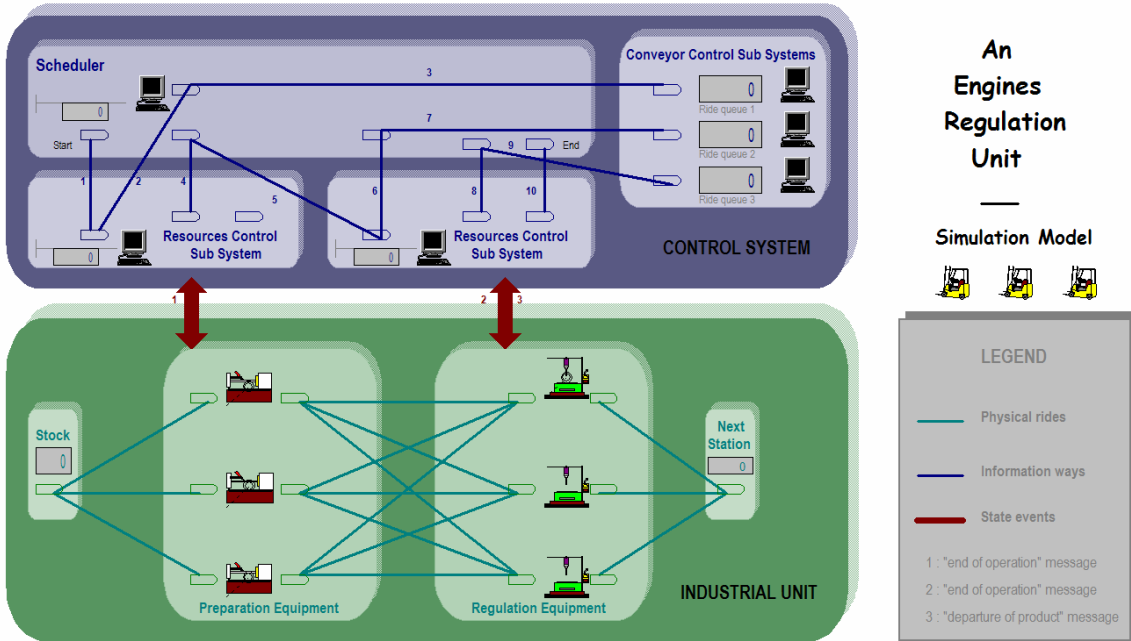


Figure 37. Animation du modèle de simulation du SAI couplé à l'unité de réglages.

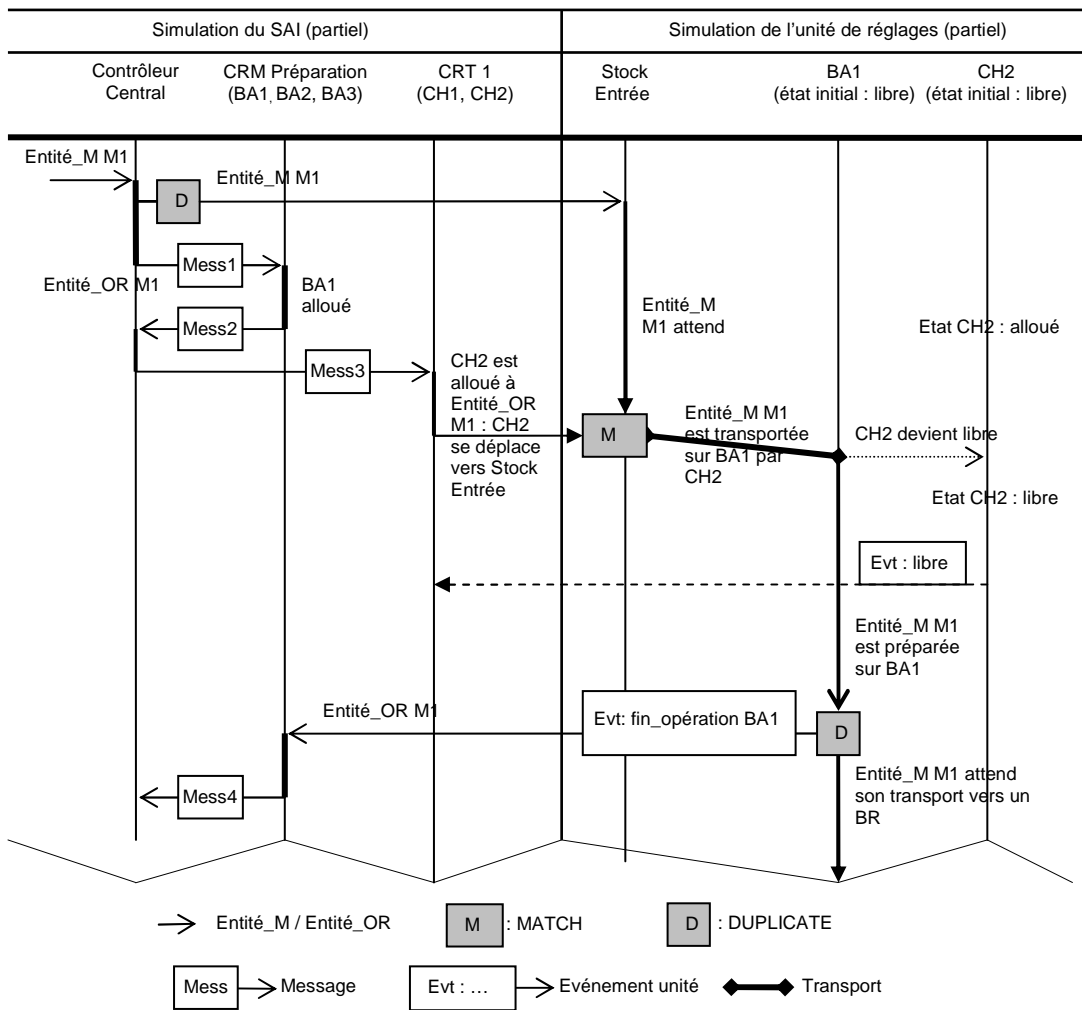


Figure 38. Scénario de simulation partiel du SAI couplé à l'unité de réglages.

4.5 Apports du travail de simulation

Cette étude de simulation affiche de manière évidente la distribution des décisions. Elle nous permet de vérifier dans un premier temps que l'architecture hybride choisie répond bien aux exigences de réactivité. En effet, après des tests concluants en conditions normales, des tests supplémentaires ont été effectués sur le modèle de simulation : arrivée inattendue d'un lot de produits urgents, introduction de pannes,... Nous avons alors constaté que les produits étaient constamment réordonnancés selon le critère de priorité au niveau local mais aussi au niveau global grâce à la notion de file d'attente priorisée et que les pannes étaient absorbées de manière immédiate efficace. Tout changement dans cet ordre a une répercussion sur la planification des opérations suivantes : la mise à jour est gérée par le Contrôleur Central. Ainsi, par exemple, un produit urgent entrant plus tard dans l'unité peut rapidement traverser l'unité en étant traité en priorité par le Contrôleur Central puis par les différents Contrôleurs de Ressources. La distribution des décisions permet de s'adapter, à tous les niveaux, aux contraintes et incertitudes de l'atelier.

Dans le modèle de simulation, le travail décisionnel se construit autour du flux des entités Ordre de Fabrication. Nous avons pu, à travers cette étude, mettre en évidence les points de décision :

- Il est possible de modifier les règles de gestion d'une file d'attente. Les Entités Ordre de Fabrication y sont stockées avant d'être prise en charge par un contrôleur. Par conséquent, les ordres de fabrication peuvent être ordonnés selon la règle voulue, adaptée aux objectifs, dans la file d'attente du Contrôleur Central. Il en est de même pour les opérations présentes dans les files d'attente des différents Contrôleur de Ressources. Il existe diverses modes de gestion des files d'attente : FIFO (First In First Out), LIFO (Last In First Out), LVF (Low Value First) classant les entités par ordre croissant de valeur d'un attribut donné, et HVF (High Value First) classant les entités par ordre décroissant de valeur d'un attribut donné. Des règles spécifiques peuvent être éventuellement implémentées. Dans le modèle, nous avons utilisé la règle HVF sur l'attribut 'priorité du produit'. Ce point de décision est représentatif de la coordination globale réalisée au niveau du Contrôleur Central en permettant de rattraper par exemple le retard d'un Ordre de Fabrication.

- Au niveau d'un Contrôleur de Ressources, il est possible de changer les règles d'affectation des ressources physiques. Des règles de décision sont en effet implantées dans le bloc de réservation (SEIZE pour les équipements ou opérateurs et REQUEST pour les transporteurs). Cette règle peut être soit choisie parmi celles proposées par le logiciel (« première ressource disponible », « ressource suivante »,..., « premier transporteur libre », « transporteur le plus près de la station »,...), soit décrite par nous-mêmes. Nous pouvons donc agir sur cette fonction pour développer l'algorithme d'ordonnancement le plus adéquat, jugé en fonction des résultats de simulation (temps de cycle le plus court possible, taux d'occupation des ressources,...). Cependant, pour les équipements, le problème se limite ici à un ordonnancement sur ressources parallèles identiques, donc il n'est pas nécessaire de définir une règle d'attribution compliquée. Ce point de décision est représentatif de l'allocation dynamique effectuée localement par chaque Contrôleur de Ressources.

En fait, la simulation répond à des questions de type « what-if », c'est-à-dire « Que se passerait-il si... ? ». Il peut ainsi être intéressant de jouer sur les différentes règles de décision et tester ainsi différentes stratégies pour trouver les meilleures options dans le cadre d'un pilotage automatique de l'unité. De la même manière, la variation de différents paramètres du modèle permet de mesurer l'influence de ceux-ci, ce qui s'avère utile pour dimensionner au mieux l'unité. De plus, la simulation peut révéler des lacunes dans l'organisation de l'unité. Dans notre cas, nous avons pu par exemple identifier la nécessité d'une solution pour marquer l'emplacement des produits dans le stock afin que le transporteur puisse le retrouver.

Cependant, il convient de préciser que la simulation ne permet pas de modéliser le choix d'un opérateur et ne souligne donc pas la nature interactive voulue pour le SAI. Le premier opérateur libre est affecté à l'opération la plus prioritaire, étant donnée que les opérations sont préalablement ordonnées selon un critère de priorité. Les opérateurs choisissent et exécutent de fait l'action la plus prioritaire. L'opérateur est assimilable à un automate qui reçoit des ordres et devient libre une fois qu'un ordre est exécuté, comme dans un pilotage optimal de la production, ce qui va à l'encontre de notre problématique.

Tout le travail de modélisation réalisé s'attache plus, en effet, à visualiser les échanges internes au sein d'un système de pilotage distribué réactif. Ces échanges sont :

- des requêtes du Contrôleur Central vers les Contrôleurs de Ressources pour l'exécution des opérations relatives aux ordres de fabrication ;
- des messages des Contrôleurs de Ressources pour signifier au Contrôleur Central que leurs décisions d'allocation ;
- des messages des Contrôleurs de Ressources pour indiquer au Contrôleur Central les opérations terminées afin que celui-ci puisse mettre à jour l'avancement de la production.

Il est évident que le caractère interactif ne peut apparaître dans ces communications internes. En effet, l'impact d'un événement de l'unité sur le SAI, que ce soit un état libre ou un choix opérateur, est à priori le même : allocation de la ressource par le Contrôleur de Ressources puis envoi de la ressource allouée au niveau supérieur. Ceci ne signifie que les Contrôleurs de Ressources soient identiques dans leur traitement d'allocation. C'est donc dans cette phase d'allocation que se trouvent les solutions d'interactivité telle que la construction d'une liste d'opérations possibles. Ainsi le développement de nos Contrôleur de Ressource justifiera le caractère interactif du système.

5. Développement du système

Le choix d'une architecture de pilotage distribuée amène à répartir la logique applicative entre différents contrôleurs. L'application correspondante est alors dite répartie. Une application répartie est en effet un ensemble de processus qui sont interconnectés via un réseau afin d'effectuer une tâche commune.

La programmation d'applications réparties est une réalité depuis de nombreuses années. Cependant les études se sont généralement focalisées autour de la mise en œuvre des composants logiciels de l'application en reléguant au second plan l'étude de l'assemblage de ces composants [94]. Le besoin d'intégrer plus fortement la répartition dans la conception des applications se fait ressentir de façon plus pressante au fur et à mesure de la généralisation des réseaux de communications. Dans ce contexte, l'architecture de communication est vue comme un élément primordial.

Nous étudions dans le paragraphe suivant les différentes possibilités pour l'architecture de communication du SAI, en gardant à l'esprit que l'objectif est de minimiser la part de logiciels spécifiques, de réduire au minimum les coûts et de pouvoir réutiliser l'architecture pour d'autres cas⁷.

5.1 Architecture de communication du SAI

Pour éviter les problèmes de transmission de données entre les différents niveaux du système, il est important d'adapter l'architecture de communication à l'architecture du SAI. Du point de vue de l'implantation, nous supposons que le Contrôleur Central est installé sur un ordinateur et que les Contrôleurs de Ressources sont installés sur un ou plusieurs ordinateurs placés dans l'unité opérationnelle. Tous les ordinateurs sont connectés par un réseau local de type Ethernet.

Ces suppositions nous poussent à étudier une architecture de communication pour système distribué. Précisons à cet effet que Tanenbaum et Van Steen définissent un système distribué comme une collection d'ordinateurs indépendants qui apparaît à ses utilisateurs comme un système unique et cohérent [95]. Cette définition comporte deux aspects : les machines sont autonomes (aspect matériel) et l'utilisateur pense qu'il n'a affaire qu'à un seul système (aspect logiciel).

⁷ Ces travaux font l'objet d'un chapitre dans le livre *Advanced Technologies : Research - Development – Application* (2006).

L'organisation d'une application répartie est principalement centrée sur la façon d'organiser les processus dans le système. Puisqu'un consensus sur l'organisation est difficile à trouver, beaucoup de chercheurs et de praticiens raisonnent en termes de clients qui exigent des services d'un serveur. Cette démarche aide à comprendre et à gérer la complexité des systèmes distribués.

5.1.1 Le modèle Client/Serveur

Dans un modèle de Client /Serveur de base, les processus sont divisés en deux groupes [96]:

- Un serveur est un processus qui implémente des services spécifiques. Il est considéré comme un fournisseur de services.
- Un client est un processus qui récupère un service sur le serveur en envoyant une demande. Il est considéré comme un consommateur de services.

La relation entre serveur et client est du type 1..N, c'est-à-dire que N clients sont rattachés à un serveur et le serveur doit pouvoir gérer plusieurs clients simultanément.

La communication est réalisée par un dialogue entre un processus client et un processus serveur (Figure 39). Le modèle Client/Serveur est un système à couplage faible, clients et serveur interagissant au moyen de messages. Le message transmis par un client à un serveur, décrivant l'opération à exécuter pour le compte du client, est appelé requête. Le message retour, transmis par un serveur à un client suite à l'exécution d'une opération et contenant le résultat de l'opération, est appelé réponse. Les rôles du client et du serveur sont dissymétriques :

- Le client émet des requêtes et attend les réponses du serveur.
- Le serveur reçoit les requêtes, les traite, et émet les réponses aux clients correspondants.

Le dialogue est, dans tous les cas, déclenché par le client demandeur de service, tandis que le serveur attend passivement les requêtes des clients.

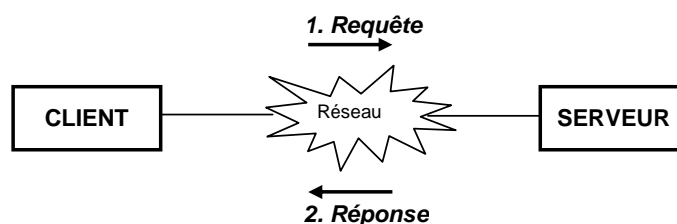


Figure 39. Le modèle Client/Serveur.

5.1.2 Les modes de communication Client/Serveur

Une des caractéristiques essentielles de la communication entre un client et un serveur est sa relation temporelle : synchrone ou asynchrone (Figure 40). Cette dichotomie doit toutefois être relativisée compte tenu du grand nombre de variations possibles pour les communications asynchrones (asynchrone avec rendez-vous, asynchrone préservant le FIFO, asynchrone sans garantie...) [97].

- Le mode de communication synchrone bloque le client jusqu'à ce que le message soit traité par le serveur. Le client attend donc la réponse pour continuer son exécution. Les modèles Client/Serveur conventionnels sont basés sur ce type de communication.
- Le mode de communication asynchrone reflète le cas contraire : le client n'est pas obligé d'attendre la réponse du serveur pour continuer son exécution. Il peut soumettre, pendant ce temps, soumettre d'autres requêtes.

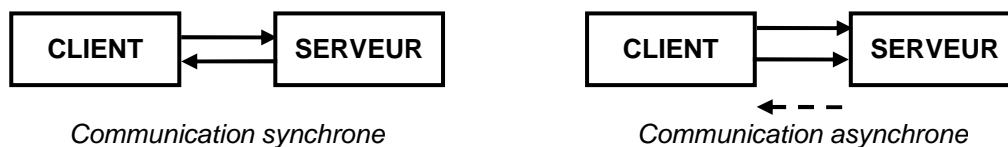


Figure 40. Les modes de communication synchrone et asynchrone.

En général, les requêtes synchrones sont utilisées pour des opérations élémentaires (ex. : demande d'un équipement de production) ou des opérations de courte durée (ex. : accès simple à une base de données) qui nécessitent absolument d'avoir été exécutées avant de poursuivre. Les requêtes asynchrones sont, elles, utilisées pour une opération qui peut prendre beaucoup de temps ou qui est exécutée sur un équipement intelligent (impression sur une imprimante, un équipement de production sophistiquée,...).

De la même manière, une communication peut se faire en mode connecté ou non connecté :

- Dans le mode non connecté, les messages sont envoyés librement.
- Dans le mode connecté, les messages sont précédés d'une ouverture de connexion et suivis d'une fermeture de connexion autorisant un meilleur contrôle des clients.

Le mode de connexion dépend généralement du protocole de transport sous-jacent (TCP fonctionne en mode connecté, UDP en mode non connecté...).

5.1.3 Le middleware

Un middleware (traduction littérale : « logiciel du milieu » ou encore « intergiciel ») permet d'assurer la communication inter-processus. Il est censé établir une liaison transparente entre le client et le serveur au travers d'un réseau. Le client/serveur idéal doit être indépendant de la plate-forme matérielle ou du système d'exploitation. Le middleware est donc au coeur du fonctionnement des applications réparties.

Certaines applications réparties utilisent directement l'interface de programmation offerte par les services réseau du système d'exploitation. Par exemple, certaines liaisons peuvent être prises en charge par des mécanismes de type sockets qui permettent aux processus localisés sur des machines différentes de s'échanger des messages. Cependant, pour s'affranchir des problèmes d'hétérogénéité de matériels et de réseaux dans les systèmes distribués tout en offrant une vue unique du système, les applications réparties modernes incluent le plus souvent un « middleware ». Il s'agit d'une couche additionnelle, située entre le niveau applicatif et le niveau réseau (Figure 41), fournissant des services permettant de rendre la distribution transparente [98]. Le middleware offre des services de haut niveau liés aux besoins de communication des applications (transport des requêtes, harmonisation des types de données, respect des protocoles, gestion de la performance...). Au sens du modèle OSI, le middleware vient se placer au-dessus de la couche de transport (couches 5, 6 et 7).

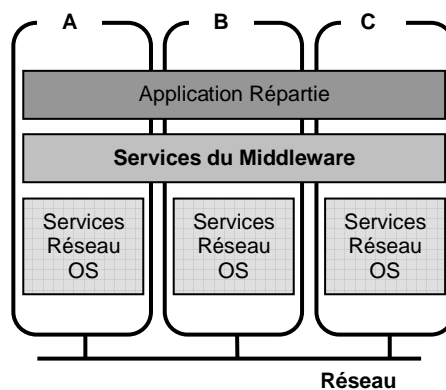


Figure 41. Structure générale d'un système distribué avec middleware. (Extraite de [98])

Pour rendre le développement des applications réparties aussi facile que possible, la plupart des middleware sont basés sur des paradigmes décrivant la distribution et les communications. Tanenbaum et Van Steen recensent cinq modèles de middleware de communication : le modèle des fichiers distribués, le modèle des opérations distribuées, le

modèle des objets distribués, le modèle des documents distribués et le modèle d'échanges de messages.

- Le modèle des fichiers distribués

Le modèle des fichiers distribués est un modèle relativement simple qui traite tout comme un fichier. Cette approche a été à l'origine employée par UNIX et les sockets. Les fichiers sont utilisés pour échanger des informations entre deux ou plusieurs processus. Dans ce cas, les processus souhaitant envoyer des informations écrivent dans un fichier à une certaine position et les processus souhaitant recevoir ces informations se positionnent aux bons emplacements dans un fichier, les lisent et les exploitent. Les systèmes de fichiers distribués (NFS) reposent sur une approche similaire. Dans de nombreuses situations, les services proposés par un tel middleware ne se situent qu'un étage au-dessus des services réseau du système d'exploitation.

- Le modèle des opérations distribuées : les systèmes RPC

Les systèmes RPC sont l'évolution logique des systèmes basés sur les sockets. RPC (Remote Procedure Call) se traduit par « appel de procédure distante ». Cela consiste en un appel permettant au composant client d'une application d'avoir accès à une fonction spécifique localisée sur un serveur distant, afin que le client ne supporte pas une trop lourde charge de calcul. En fait, le client transmet une requête au serveur et attend sa réponse. Bien que des systèmes RPC asynchrones existent [99], la plupart des systèmes RPC mis en oeuvre sont synchrones, de telle sorte que le client ne peut continuer son traitement avant que le serveur ne lui donne une réponse. Il convient de noter que l'environnement DCE (Distributed Computing Environment) facilite l'implantation des telles infrastructures.

- Le modèle des objets distribués : les ORB

Un ORB (Object Request Broker) assure l'interopérabilité entre des objets distribués en leur fournissant des services. En effet, grâce à un ORB, un client peut envoyer des requêtes visant des objets situés sur un serveur. Un ORB permet donc de gérer la communication et l'échange de données entre les différents objets définis dans un modèle. Nous pouvons identifier deux modèles bien connus adaptés à ce type de technologie : CORBA (Common Object Request Broker Architecture) développé par l'OMG et COM/DCOM (Distributed Component Object

Model) développé par Microsoft. Avec le développement de la programmation objet, ces technologies sont devenues très prisées. CORBA s'appuie sur un protocole de communication robuste (IIOP) et propose aussi un langage de description d'interfaces indépendant de tout langage de programmation (IDL) [100]. CORBA est considéré par beaucoup d'experts comme l'architecture la plus complète de cette catégorie de middleware et comme la plus fidèle aux principes objet [101]. Bien qu'elles soient prometteuses, les technologies CORBA et DCOM, et plus particulièrement CORBA, restent néanmoins délicates à mettre en oeuvre. De surcroît, le fait qu'elles soient basées sur des mécanismes propriétaires, impliquant une dépendance vis-à-vis d'un vendeur pour la maintenance, constitue parfois un frein à leur utilisation. A ces technologies peut être ajoutée une API Java développée par Sun : Java RMI (Remote Method Invocation). Java RMI autorise le développement de middleware ORB fonctionnant comme CORBA ou DCOM, mais est beaucoup moins riche sur le plan technique. Elle présente cependant l'avantage d'être indépendante. En réalité, Java RMI se rapproche plus des RPC objet dont le but est de mettre en commun les technologies RPC et la programmation objet. Dans un RPC objet, l'appel de procédure se fait sur un objet distant.

- Le modèle des documents distribués : le modèle Web

Le Web est probablement une des approches qui simplifie le plus l'utilisation de systèmes en réseau. Le succès du Web est principalement dû à l'usage du modèle de documents distribués. En effet, dans le modèle Internet, l'information est organisée en documents distribués : chaque document réside dans un ordinateur situé n'importe où dans le monde et est accessible depuis n'importe quel autre ordinateur. Le modèle Internet est basé sur le protocole de communication notoire HTTP. Il est important de préciser que la notion de document ne doit pas se restreindre à de l'information textuelle.

- Le modèle d'échange de messages : les MOM

Un MOM (Message Oriented Middleware) est un serveur qui joue un rôle d'intermédiaire dans la communication entre applications : il stocke les messages dans une file d'attente et attend que l'application correspondante soit prête à recevoir le message pour le lui envoyer [102]. Il offre ainsi un mécanisme de régulation des flux extrêmement intéressant. En fait, il s'agit en fait d'une généralisation de la boîte mail. Contrairement aux systèmes RPC et aux ORB, les MOM sont typiquement asynchrones et sont ainsi particulièrement bien adaptés pour des applications conduites par des événements. Les MOM peuvent cependant aussi

prendre en charge des appels synchrones. Malheureusement, les MOM, tel que MQ-Series proposé par IBM, sont essentiellement des produits propriétaires. Néanmoins, il existe une nouvelle API Java dénommée JMS (Java Messaging Service) [103] qui permet de développer un MOM capable de communiquer avec d'autres implémentations de MOM.

Le choix d'un des modèles précédents déterminera l'architecture de communication du système de pilotage distribué. Les deux premiers modèles sont plutôt utilisés dans le cas de communications synchrones et les trois autres dans le cas de communications asynchrones. Chacun présente ses propres avantages et inconvénients, et il convient donc de choisir le modèle le plus approprié à notre situation. Pour faire ce choix, il nous faut donc d'abord répondre à une question primordiale : les communications du Contrôleur Central sont-elles synchrones ou asynchrones ?

5.1.4 Quelle architecture de communication pour le SAI ?

Nous devons mettre en évidence le caractère spécifique des échanges entre les contrôleurs du SAI afin de choisir l'architecture de communication la plus adéquate. Notre étude de modélisation/simulation a révélé que les échanges du Contrôleur Central vers les Contrôleurs de Ressource sont des demandes d'exécution des opérations relatives aux ordres de fabrication (Figure 42). Il est évident que les communications correspondantes doivent être asynchrones dans le sens où le Contrôleur Central ne devrait jamais avoir à attendre la fin de l'exécution d'une opération d'une durée humaine pour continuer son traitement (durée opératoire largement supérieur aux pires performances d'un Client/Serveur). En conséquence, les ORB, les MOM et le Web constituent un premier sous-ensemble de choix pour l'architecture du SAI.

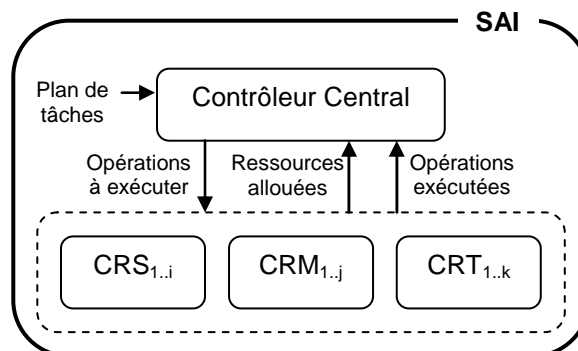


Figure 42. Communications internes au SAI.

Mais si nous analysons plus précisément les communications à l'intérieur du système, nous pouvons envisager une autre considération susceptible de simplifier l'implantation de l'architecture de communication :

1. Puisque nous avons opté pour une architecture hybride pour le système de pilotage, le SAI a naturellement une structure hiérarchique. Le Contrôleur Central tient alors le rôle de « chef » : il surveille le fonctionnement du système entier et voit les Contrôleurs de Ressources comme ses « subordonnés ». En fait, un Contrôleur de Ressources doit exécuter les opérations provenant du Contrôleur Central, en employant les ressources présentes dans l'unité industrielle, et renvoyer l'indication 'opération exécutée'.
2. En raison de la propriété hétéroarchitecturale conférée par une architecture hybride, un Contrôleur de Ressources est intelligent, c'est-à-dire qu'il a la latitude décisionnelle nécessaire pour exécuter seul les opérations. De plus, les Contrôleurs de Ressources ne communiquent jamais entre eux. Un Contrôleur de Ressources est donc autonome et indépendant.

Compte tenu des propriétés 1 et 2 du système, il est possible de redéfinir le SAI en termes de Client/Serveur. Dans ce cas, le Contrôleur Central est vu comme un serveur d'opérations à exécuter et les Contrôleurs de Ressources sont vus comme ses clients.

Etant donné le caractère asynchrone des communications, nous pouvons mettre en place sur le serveur Contrôleur Central, des documents distribués intitulés "Liste des Opérations à Exécuter" destinés aux différents Contrôleur de Ressources. Chaque client Contrôleur de Ressources devra alors périodiquement récupérer puis mettre à jour, sur le serveur, son document "Liste des Opérations pour Exécuter" (Figure 43).

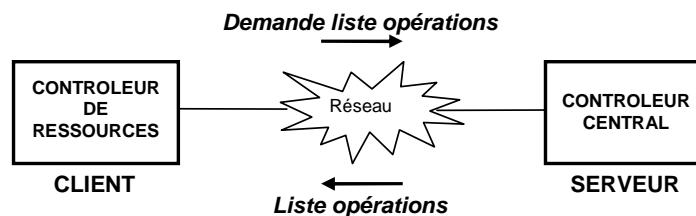


Figure 43. Le modèle Client/Serveur pour le SAI.

De ce point de vue, adopter l'architecture Client/Serveur du Web semble être la solution la plus judicieuse pour notre SAI [104]. Simple à mettre en œuvre et de plus en plus fiable, elle fait d'ailleurs l'objet de nombreuses applications dans le monde manufacturier [105]. Cependant, il faut garder à l'esprit que l'utilisation d'un modèle Client/serveur implique que

les clients se connectent eux-mêmes au serveur pour obtenir les opérations à exécuter. Nous devons donc définir quelques règles de communication pour adapter le SAI à l'architecture Client/Serveur du Web. Nous détaillons cette architecture dans le prochain paragraphe et nous intéressons plus particulièrement au rôle du protocole HTTP qui assure les communications.

5.1.5 Application de l'architecture du Web au SAI

L'architecture du Web est naturellement construite autour de l'architecture de Client/Serveur [106]. En effet, elle est caractérisée par trois éléments :

- un client communément appelé « Navigateur Web »
- un serveur, appelé « Serveur Web », avec qui le client communique
- un réseau TCP/IP sur lequel passent les communications

Un exemple typique d'utilisation est la navigation sur Internet : chaque fois qu'un utilisateur veut consulter un site Web, il en fait la demande au serveur en tapant l'adresse correspondante dans son navigateur et le serveur lui retourne l'information. Plus précisément, ce processus signifie que le client Web demande une ressource située sur le serveur. Une ressource est un morceau d'information dont la localisation sur le serveur peut être identifiée par une URL (Uniform Resource Locator). Une ressource est généralement un fichier mais elle peut prendre d'autres formes (résultat d'un script, résultat d'une demande générée dynamiquement,...).

HTTP (HyperText Transfer Protocol) est le protocole sous-jacent du navigateur [107] : il fournit le mécanisme de récupération de ressources. HTTP 1.1 est un protocole de niveau applicatif adéquat pour les systèmes d'information distribués, collaboratifs, et contenant divers médias. Il s'agit d'un protocole générique, sans état, c'est-à-dire que les informations ne sont pas sauvegardées et que les requêtes successives d'un même client sont traitées indépendamment. Au-delà de son usage classique pour l'hypertexte, HTTP sert dans de nombreuses applications telles que les serveurs de nom ou encore les systèmes de gestion d'objets distribués, grâce à l'extension possible de ses méthodes de requête, de ses codes d'erreur et de ses en-têtes.

- Le protocole HTTP

Le protocole HTTP est un protocole de requête/réponse fonctionnant au travers de sockets. Un client envoie une requête au serveur qui lui renvoie une réponse incluant généralement la

ressource demandée (Figure 44). Le format d'un message pour une requête ou une réponse est similaire et comporte trois parties [108] :

- une ligne initiale qui diffère selon le type de message (requête, réponse)
- des lignes d'en-tête qui fournissent des informations sur la requête ou la réponse, ou encore sur l'objet envoyé dans le corps du message (ex. : type MIME du contenu comme text/html ou image/gif, longueur du contenu,...)
- le corps du message (optionnel) contenant, dans le cas d'une requête, des données entrées par l'utilisateur ou des fichiers à envoyer au serveur, ou, dans le cas d'une réponse, la ressource demandée par le client.

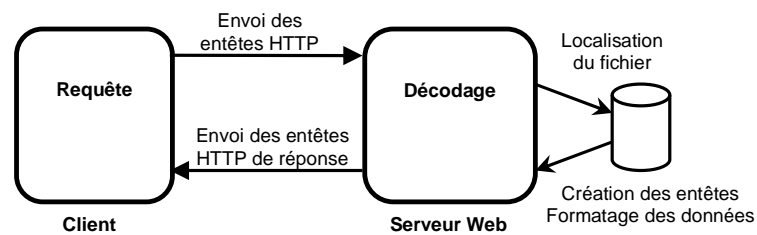


Figure 44. Le protocole HTTP.

Pour une requête, la ligne initiale, appelée ligne de requête, comprend un nom de méthode, le chemin local de la ressource demandée (URI) et la version de HTTP. Une ligne de requête typique se présente donc sous la forme :

```
GET /chemin/index.html HTTP/1.0
```

Pour une réponse, la ligne initiale, appelée ligne de statut, comprend la version de HTTP, un code statut représentatif du résultat de la requête et une expression expliquant le code statut. Une ligne de statut typique se présente donc sous la forme :

```
HTTP/1.0 200 OK (succès) ou HTTP/1.0 404 Not Found (échec : ressource non trouvée)
```

La méthode utilisée constitue un élément essentiel d'une requête http dans la mesure où elle conditionne l'action réalisée sur le serveur. Bien que la plupart des requêtes soient basées sur la méthode GET, quatre autres méthodes peuvent être appliquées. Chacune répond à un besoin précis :

- GET permet de récupérer une ressource ;
- HEAD équivaut à la méthode GET mais ne récupère que les en-têtes ;
- POST permet d'envoyer des données à un programme situé sur le serveur ;
- PUT permet de déposer des données sur le serveur ;
- DELETE permet de supprimer une ressource située sur le serveur ;

Ces cinq méthodes suffisent au développement d'applications Client/Serveur simples. Par exemple, la soumission d'un formulaire ne nécessite que l'emploi d'une méthode POST : les données du formulaire, envoyées grâce à cette méthode, sont traitées sur le serveur par un programme (script CGI). Si les interactions avec le serveur se veulent plus complexes, il existe des langages de script plus évolués (tels que JavaScript, VBScript, PHP,...) ou des mini programmes (tels que les servlets en langage Java) très utiles pour développer les applications du côté serveur. De la même manière, du côté client, le logiciel client n'est pas forcément un simple navigateur. Il peut être complété avec des fonctionnalités fournies par exemple par des applets (en langage Java), voire même être un programme applicatif réel effectuant ses propres traitements. Il est important de préciser cependant, qu'avec cette dernière option, les bénéfices du client léger sont perdus. Dans tous les cas, les communications sont aisément assurées par le protocole HTTP. L'architecture Web connaît actuellement un vif succès grâce à ce protocole très simple d'utilisation.

En réalité, le Serveur Web agit comme un middleware : les requêtes du client passent par le Serveur Web qui les redirige vers l'application appropriée. Ce rôle de middleware est facilité par le développement de multiples standards adaptés à l'architecture Internet : JAVA pour la programmation, HTML pour les interfaces graphiques, HTTP pour les transports, XML pour les échanges de données, IIOP pour les communications inter-objets, LDAP pour les authentifications,... Ainsi, il autorise le déploiement d'applications génériques.

- Des règles de communications HTTP pour le SAI

Ajuster notre architecture de pilotage à l'architecture Web revient alors à définir le Contrôleur Central comme un Serveur Web dont les clients sont les Contrôleurs de Ressources. Dans ce cas, les contrôleurs peuvent utiliser le protocole HTTP pour s'échanger des messages. Trois règles de communications apparaissent (Figure 45):

- Un client Contrôleur de Ressources récupère sa liste d'opérations à exécuter sur le serveur Contrôleur Central. Il envoie pour cela une requête HTTP basée sur la méthode GET qui permet de rapatrier un document situé sur un serveur (document localisé grâce à une URI).
- Un client Contrôleur de Ressources transmet une décision d'allocation de ressources au serveur Contrôleur Central. Il envoie pour cela une requête HTTP basée sur la méthode POST qui permet d'envoyer des données à un programme

situé sur un serveur. Le Contrôleur Central met éventuellement à jour les listes d'opérations avec les données reçues.

- Lorsqu'une opération est exécutée, un client Contrôleur de Ressources transmet l'information au serveur Contrôleur Central. Il envoie pour cela une requête HTTP basée la méthode POST qui permet d'envoyer des données à un programme situé sur un serveur. Le Contrôleur Central peut alors mettre à jour les listes d'opérations à partir des données reçues.

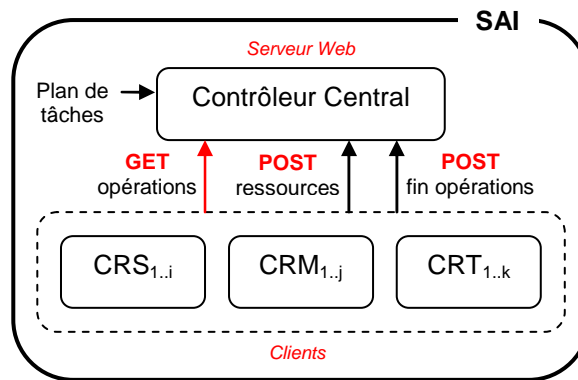


Figure 45. Communications HTTP du SAI.

5.2 Analyse et programmation des composants

Schématiquement une application répartie est composée:

- d'une infrastructure d'exécution générale (RPC, ORB, Internet, ...);
- de composants implémentant des services (servlets, composants COM, EJB,...);
- d'un protocole de communication permettant aux composants d'envoyer et de recevoir des messages (HTTP, IIOP, SMTP, ...).

Ces trois éléments sont généralement liés, dans le sens où le choix d'un élément oriente le choix d'un autre élément. Dans notre cas, nous nous sommes d'abord intéressés à l'architecture de communication et avons opté pour une infrastructure de type Web et son protocole de communication sous-jacent HTTP. Dans ce paragraphe, nous abordons le dernier point essentiel au fonctionnement d'une application répartie: l'implémentation des composants du système.

5.2.1 Les composants du Contrôleur Central

Le rôle du Contrôleur Central est de coordonner les actions des différents Contrôleurs de ressource. Il consiste essentiellement à créer et à mettre à jour les différentes listes

d'opérations destinées aux différents Contrôleurs de Ressources. Le traitement sur le serveur Contrôleur Central est déclenché par l'arrivée d'une requête HTTP provenant des clients Contrôleurs de Ressources. Les composants serveur généralement associés à la gestion de requêtes HTTP sont les servlets Java. En effet, Java, et plus particulièrement la norme J2EE, procure un arsenal assez complet d'API permettant de faciliter le développement d'applications réseau (ou autres) tout en restant indépendant de la plate-forme d'exécution [109]. Il est important de préciser que Java est un langage orienté objet [110].

- Le Contrôleur Central sous forme de servlet HTTP

Une servlet Java est un composant qui a pour but de générer une réponse à une sollicitation d'un service. Son principe de fonctionnement (architecture, fonctionnalités, configuration, déploiement) est décrit suivant une spécification officielle proposée par Sun [111].

Une servlet HTTP Java permet de traiter des données au sein d'un serveur HTTP (Figure 46). Ce programme Java s'exécute dynamiquement sur le serveur Web et permet l'extension des fonctions de ce dernier. Il peut exister une ou plusieurs servlets dans une application Web. Une servlet peut être chargée automatiquement lors du démarrage du serveur Web ou lors de la première requête du client. Une fois chargées, les servlets restent actives dans l'attente d'autres requêtes du client. Lorsqu'un client envoie une requête au serveur, ce dernier transmet à la servlet les informations relatives à la requête. Par la suite, le servlet crée une réponse que le serveur renvoie au client. Lors de la création de la réponse, le servlet peut utiliser toutes les fonctions du langage Java ou communiquer avec des ressources externes (fichiers, bases de données ou applications écrites en Java ou dans d'autres langages).

L'utilisation de servlets se fait par le biais d'un conteneur de servlets côté serveur. Celui-ci constitue l'environnement d'exécution de la servlet et lui permet de persister entre les requêtes des clients. La spécification de Sun définit les relations entre le conteneur et la servlet. Le conteneur reçoit la requête du client, et sélectionne la servlet qui aura à la traiter. Le conteneur fournit également tout un ensemble de services standard pour simplifier la gestion des requêtes et des sessions. Nous utiliserons ici Tomcat en tant que conteneur de servlets : Tomcat est un conteneur de servlets très connu développé par la société Apache [112] et qui présente l'avantage d'être libre de droits.

La classe de base d'une servlet HTTP est `HttpServlet`. Elle doit généralement implémenter au moins les méthodes suivantes :

- `init()` : traitement à faire lors du chargement de la servlet;

- destroy() : traitement à faire lors de la désactivation de la servlet;
- doGet() : traitement à faire lors de la réception d'une requête HTTP de type GET (le plus souvent, une page JSP est générée) ;

Il est possible de créer autant de fonctions que de types de requêtes HTTP. Nous pouvons ainsi implémenter une méthode doPost() traitant les requêtes HTTP de type POST.

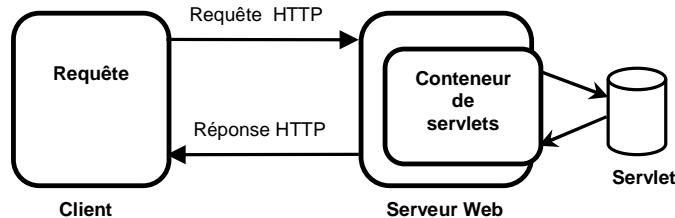


Figure 46. Schéma d'une servlet.

La définition d'une servlet est donc parfaitement adaptée à l'action du Contrôleur Central. Dans notre cas, la servlet permet de renvoyer une liste d'opérations à un Contrôleur de Ressources qui en fait la demande par une requête GET et elle contient également les traitements de mises à jour des listes d'opération effectués à la réception d'une requête POST. Nous avons identifié trois principales sources de révision des listes conséquentes aux besoins de coordination et de pilotage réactif de l'unité opérationnelle :

- L'arrivée d'une commande urgente donne des ordres de fabrication à faire passer en priorité. Les listes d'opérations doivent être réordonnées en fonction avant d'être renvoyées aux Contrôleurs de Ressources. Ce traitement fait suite à une requête GET (les listes sont vérifiées avant envoi).
- L'envoi d'une ressource allouée par un Contrôleur de Ressources Statiques ou Mobiles provoque logiquement la mise à jour des listes d'opérations destinées au Contrôleur de Ressources de Transfert concerné. En effet, une fois l'action choisie par une ressource, il est nécessaire de déplacer le produit vers cette ressource. De plus, l'opération correspondante est provisoirement supprimée de la liste concernée pour ne pas être réaffectée. Ce traitement fait suite à une requête POST.
- Sauf dans le cas d'une opération de transport, l'indication de fin d'opération conduit à l'ajout de l'opération suivante de la gamme, si elle existe, dans la liste destinée au Contrôleur de Ressources gérant cette même opération. Il peut éventuellement s'agir du transport final si le produit est arrivé en bout de gamme. Ce traitement fait suite à une requête POST.

Quelle que soit la requête du Contrôleur de Ressources (CR), une liste mise à jour est automatiquement renvoyée à celui-ci. La méthode doPost() fait donc appel à la méthode doGet(). Il convient de noter que, dans le cas d'un envoi de ressource ou d'une fin d'opération, les objets des classes Produit et autres classes relatives sont également modifiés sur le serveur afin de permettre une certaine traçabilité.

En conséquence, notre servlet présente sur le serveur Contrôleur Central a la structure globale suivante :

```
public class ContrôleurCentral extends HttpServlet() {

    public void init() {
        //création des listes d'opérations initiales
    }
    public void doGet() {
        //vérification de la liste destinée au CR et renvoi
    }

    public void doPost() {
        if (ressource allouée) {
            //suppression de l'opération de la liste du CR
            //création du transport correspondant et ajout dans
            la liste du CRT concerné
        }
        if (fin operation) {
            //ajout de l'opération suivante dans la liste du CR
            concerné
        }
    }
}
```

En réalité, la gestion temps réel amène à des traitements de synchronisation un peu plus complexes. Ceci est tout à fait concevable en raison de la pluralité des états ressource disponibles (début opération, libre, occupée,...). Tous les cas sont envisagés dans le code complet (cf Annexe 2).

Les fonctions du Contrôleur Central se résument en la création et l'entretien (alimentations, suppressions, remises en ordre) de listes d'opérations ordonnées pour les Contrôleurs de Ressources qui en font la demande, tout en respectant les contraintes de précedence. Il existe autant de listes que de Contrôleurs de Ressources.

- Choix techniques annexes

La gestion des listes conduit à deux choix techniques supplémentaires en ce qui concerne la façon de les concevoir à partir du plan de tâches et leur format. Les solutions peuvent

facilement se trouver parmi les multiples API procurées par Java. Les servlets ont en effet accès à la totalité des API Java.

1. Une base de données et un accès JDBC pour les ordres de fabrication

Nous supposons que les ordres de fabrication sont susceptibles d'être stockés dans une base de données quelconque et il faut pouvoir produire des requêtes pour en extraire les produits et les opérations. Nous utilisons le SGBD MySQL pour des raisons de gratuité.

L'API JDBC (Java Data Base Connectivity) permet à un programme de se connecter de façon transparente à n'importe quelle base de données en utilisant la même syntaxe, c'est-à-dire que l'API JDBC est indépendante du SGBD. En ce sens, elle peut être considérée comme un middleware d'accès aux données.

2. Le format XML et la librairie JAXP pour les listes d'opérations

Les listes d'opérations sont au format XML. XML (eXtensible Markup Language) est un langage d'échange et de structuration d'informations. La structuration est réalisée à l'aide de tags. Nous avons opté pour XML car il s'impose comme un standard pour l'échange de données dans la mesure où il est très facile d'analyser et d'extraire de l'information de ce type de fichier structuré [114]. Java s'est vue adjoindre deux API spécialisées dans le traitement de document XML. Il existe en effet deux manières de traiter des documents XML :

- Le modèle SAX (Simple API for XML) proposé par le groupe XML.org est en général utilisé pour « parser », c'est-à-dire décomposer et exploiter, un document XML : à chaque élément syntaxique lu, une méthode est déclenchée.
- Le modèle DOM (Document Object Model) proposé par le W3C est plus utilisé pour générer un nouveau document XML, mais il est aussi possible de manipuler un document XML existant.

L'API JAXP (Java API for XML Parsing) procure des implémentations des deux modèles. Généralement, la création/modification d'un document fait plutôt appel au premier modèle, tandis que l'analyse de document fait appel au second.

Dans notre cas, le document XML ainsi conçu a le format suivant (le format est évidemment différent pour une opération de transport, puisqu'il doit inclure l'origine et la destination):

```
<LISTEOPERATIONS NomOpération='OpérationContrôleur' Type='S/M/T'>
  <OPERATION Durée=15 IdOpération='Op1'>
```

```

        <IDPRODUIT Priorité=4>Prod5</IDPRODUIT>
    </OPERATION>
    <OPERATION Durée=15 IdOpération='Op2'>
        <IDPRODUIT Priorité=2>Prod2</IDPRODUIT>
    </OPERATION>
    ...
</LISTEOPERATIONS>

```

5.2.2 Les clients Contrôleurs de Ressources

Le rôle majeur d'un Contrôleur de Ressources est de procéder à l'allocation dynamique de ses ressources pour les opérations demandées. Il consiste alors à interroger régulièrement le serveur Contrôleur Central afin de récupérer sa liste mise à jour d'opérations à exécuter et à analyser cette liste afin d'affecter la ressource adéquate à chaque opération. L'interrogation du serveur se fait par une requête de type GET. Dès qu'une ressource est allouée, celle-ci est transmise au Contrôleur Central par une requête de type POST contenant le numéro de la ressource. Une fois l'opération terminée, le Contrôleur de Ressources indique son identifiant dans une requête de type POST (inutile dans le cas d'une opération de transport). A chaque POST, une liste mise à jour est également renvoyée par le Contrôleur Central.

Les Contrôleurs de Ressources ont donc tous la même procédure de récupération de liste et d'envoi de données (implémentée via une classe ClientHttp). Cependant, ils diffèrent dans leur phase d'allocation en fonction de la composante humaine qu'ils intègrent.

- Décision d'allocation pour un Contrôleur de Ressources Statiques (CRS)

Dans le cas d'un CRS, la composante humaine n'est pas prise en compte, étant donné qu'un opérateur appartient à un équipement. Seuls les états des équipements suffisent donc à l'allocation. Le CRS affecte ses propres ressources aux opérations selon la règle suivante, appropriée à la gestion par files d'attente : dès qu'une de ses ressources est libre, elle est allouée à l'opération la plus prioritaire de sa liste. Comme nous l'avons remarqué dans l'étude de simulation, cette règle peut être changée. En réalité, il affecte la première ressource libre trouvée à la première opération de la liste. Du fait que la liste est déjà ordonnée par priorité, cela traduit bien la règle énoncée. Pour chaque opération allouée, le CRS renvoie immédiatement sa décision au Contrôleur Central.

- Décision d'allocation pour un Contrôleur de Ressources Mobiles (CRM)

Dans le cas d'un CRM, l'opérateur prend part dans la décision d'allocation. En effet, celui-ci a à choisir entre plusieurs opérations sur équipement. Pour s'affranchir du caractère incertain d'une allocation automatique, nous proposons de lui suggérer en temps réel une liste composée des trois opérations les plus prioritaires. A la réception de la liste des opérations provenant du Contrôleur Central, le CRM génère donc tout d'abord une nouvelle liste destinée aux opérateurs appelée 'Liste Opérateurs'. Il est important de préciser que, pour une opération effectuée dans le cadre de ressources semi-mobiles, un équipement doit être affecté au préalable : seules les opérations pour lesquelles un équipement est disponible pourront être proposées aux opérateurs. Ainsi, lorsqu'il analyse la liste des opérations du Contrôleur Central, le CRM procède dans un premier temps de la même manière qu'un CRS pour rechercher un équipement libre pour une opération. S'il trouve un équipement, il insère l'opération dans la liste qui sera diffusée aux opérateurs en respectant la notion de priorité. Il répète cette action pour chaque opération, dans la limite de trois opérations pour la 'Liste Opérateurs'.

La notion d'interactivité entre alors en jeu. Dès qu'un opérateur fait un choix parmi l'ensemble d'opérations suggérées, la 'Liste Opérateurs' est censée être immédiatement mise à jour en fonction de ce choix. Une procédure de reconstruction est donc implémentée afin de revoir la Liste Opérateurs : suppression provisoire de l'opération relative à ce choix, réorganisation et intégration éventuelle de nouvelles opérations prioritaires. En effet, dans le même temps, la décision d'allocation complète (équipement+opérateur) est transmise par le CRT au Contrôleur Central qui effectue son traitement de synchronisation et lui renvoie une liste mise à jour d'opérations à allouer. La liste est donc vouée à être rediffusée en temps réel aux opérateurs à chacune de leurs décisions. De cette façon, les opérateurs sont toujours informés des opérations les plus prioritaires et ne peuvent pas à priori choisir la même action.

- Décision d'allocation pour un Contrôleur de Ressources Mobiles (CRT)

Dans le cas d'un CRT, l'opérateur est totalement responsable de son affectation. En effet, celui-ci est amené à choisir entre plusieurs transports. Comme pour les opérateurs d'un CRM, le CRT ne décide pas au préalable de son sort mais lui suggère en temps réel une liste de trois transports prioritaires à réaliser. La 'Liste Opérateurs' est construite à partir de la liste des opérations de transfert liées au CRT calculées par le Contrôleur Central. Les opérations de transfert ne dépendent pas en effet directement d'un équipement mais du déroulement des

autres opérations sur équipement, un transfert étant nécessaire dès lors qu'une opération sur équipement est programmée. La liste des opérations de transfert est différente dans la mesure où elle intègre une origine et une destination. Le CRT applique une partie de la procédure mise en place pour un CRM : il insère l'opération de transfert dans la 'Liste Opérateurs' préalablement créée et, aussitôt que l'opérateur a fait son choix, la liste est mise à jour.

L'interactivité est ici primordiale, en particulier si les opérateurs couvrent une grande partie de l'unité. Une opération prioritaire peut par exemple avoir lieu dans une partie complètement inaccessible, d'un point de vue informationnel, de l'emplacement actuel d'un opérateur. Une réalisation efficace de ces opérations de transfert conditionnera le bon déroulement des opérations d'où la nécessité de coordination au niveau du Contrôleur Central.

- Suivi de la production

L'avancement de la production est principalement marqué par les états des équipements de production. Tout échec ou succès dans la réalisation d'une opération peuvent en être déduits : un début d'opération sur équipement signifie effectivement que le transport correspondant a bien été effectué. Cela permet de mettre en place des procédures de recouvrement en cas de non réalisation ou éventuellement de panne. De même, la réception d'un état 'fin_opération' indique que l'opération est terminée. Nous supposons que ces états peuvent être transmis de manière automatique ou non par l'unité opérationnelle.

- Choix techniques

Ces traitements engendrent trois choix techniques concernant l'analyse d'une liste d'opérations, le format des listes destinées aux opérateurs, ainsi que la façon de gérer les états des équipements.

1. L'exploitation d'une liste d'opérations provenant du Contrôleur Central est réalisée grâce à l'API JAXP décrite précédemment
2. Pour les besoins futurs d'implantation, nous considérons que la liste 'Liste Opérateurs' est également au format XML et construite grâce à l'API JAXP.
3. Nous supposons ici que les états des équipements sont stockés et mis à jour dans une base de données Ressources (MySQL), partagée par tous les Contrôleurs de Ressources, à laquelle les contrôleurs accèdent via une connexion JDBC. Il est ainsi aisé d'extraire, à un moment donné, les ressources libres.

5.2.3 Vers des Contrôleurs de Ressources clients et serveurs

Les Contrôleurs de Ressource réagissent à des événements de l'unité. Les états des équipements permettent, entre autres, de suivre le déroulement de la production et les décisions des opérateurs constituent l'essence même du système. En réalité, les traitements de mises à jour évoqués ne peuvent être activés qu'à la réception de ces événements. Ainsi, pour obtenir une structure décisionnelle complète et réaliste, il est obligatoire de mettre en place les méthodes relatives à ces réceptions d'événements. En effet, sans les événements de l'unité opérationnelle, seules les communications dans le sens Contrôleur Central vers Contrôleur de Ressources sont véritablement implémentables. Il s'agit donc de préétablir une liaison standard entre l'unité opérationnelle et le SAI, du côté des Contrôleurs de Ressources.

Nous pouvons facilement présumer que les différents opérateurs d'une unité seront dotés d'ordinateurs ou de PDA incluant un navigateur susceptible de transmettre leurs choix (matérialisés par la soumission d'un formulaire menant à une requête HTTP). De même, il est tout à fait envisageable de transformer les états des équipements en requêtes HTTP. En conséquence, il est possible d'adopter, pour les communications entre Contrôleurs de Ressources et le SAI, une architecture Client/Serveur Web similaire à la précédente. Dans ce cas, un Contrôleur de Ressources devient serveur et ses opérateurs sont considérés comme des clients devant récupérer leur liste d'opérations à exécuter. Par suite, un Contrôleur de Ressources est bicéphale et tient deux rôles : il est client du Contrôleur Central du SAI et il est serveur d'un ensemble de ressources de l'unité opérationnelle (Figure 47).



Figure 47. Le rôle bipartite d'un Contrôleur de Ressources.

Ceci est tout à fait concevable puisque, dans le modèle Client/Serveur, un processus peut être client ou serveur ou client et serveur. Les réactions associées aux événements, évoquées dans le paragraphe précédent, doivent alors être insérées dans une servlet du serveur Contrôleur de Ressources. L'usage d'une servlet autorise les traitements principaux suivants :

- doGet() permet, pour un CRM / CRT, de renvoyer la 'Liste Opérateurs';
- doPost() permet, pour un CRS / CRM, de transmettre une fin d'opération au Contrôleur Central et, pour un CRM / CRT, d'affecter définitivement l'opérateur.

Notre SAI comprend au final un Contrôleur Central serveur et des Contrôleurs de Ressources clients et serveurs (Figure 48).

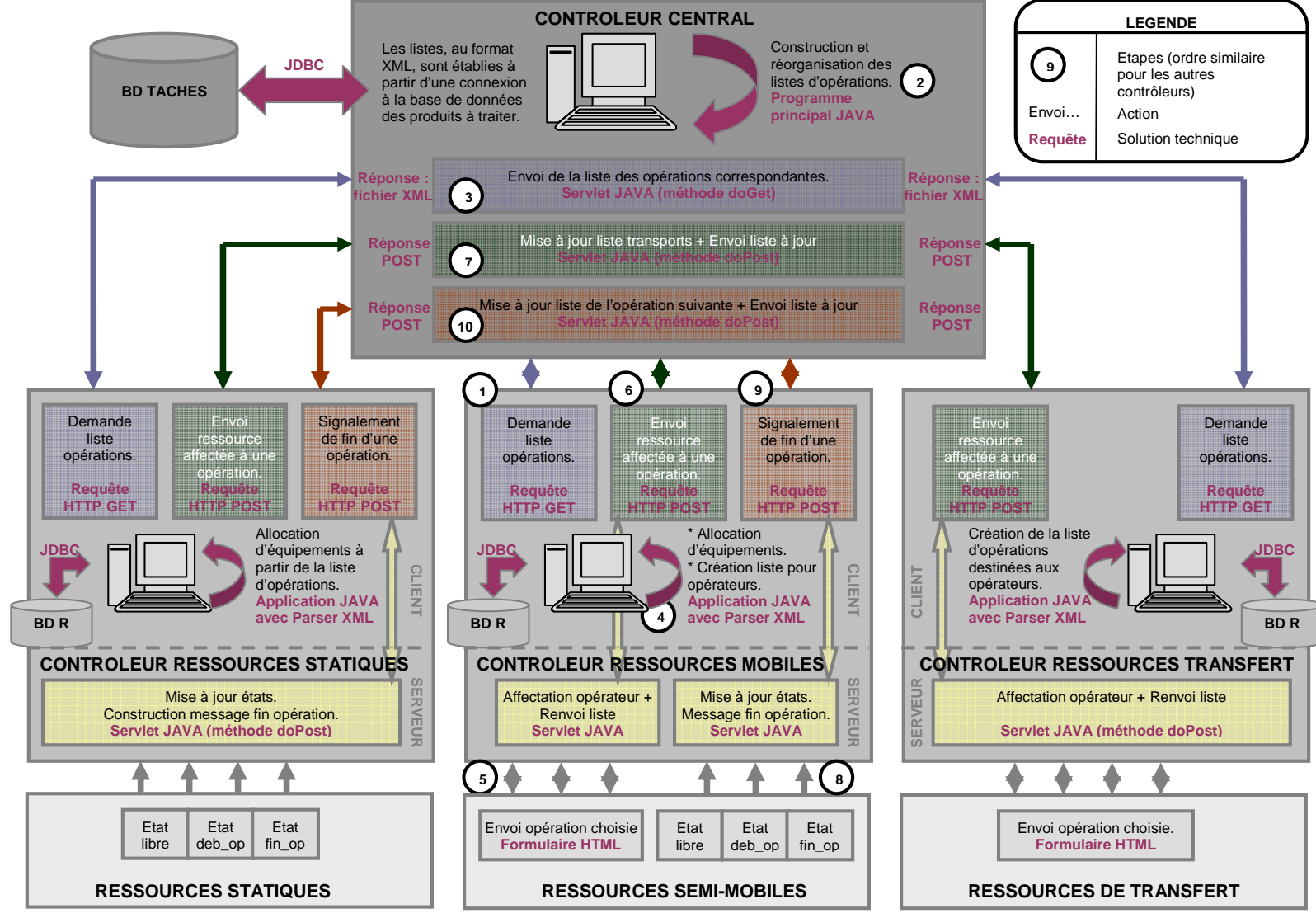


Figure 48. Schéma de développement du SAI.

5.3 Un exemple d'interactivité

Nous montrons ici un exemple du résultat produit pour un opérateur en se basant sur l'unité de réglages moteurs décrite précédemment (§4.4.1).

Pour notre unité référence, les quatre à sept Contrôleurs de Ressources (CRM Préparation, CRS Réglage, CRM Vidange et les CRT Transport) deviennent autant de clients HTTP du serveur Web Contrôleur Central. Ce dernier doit gérer autant de connexions que de Contrôleurs de Ressources définis. Les Contrôleurs de Ressources sont eux-mêmes serveurs de l'ensemble de ressources qu'ils prennent en charge. L'application du SAI ainsi conçu sur cette unité nous a permis de constater, dans un premier temps, son bon fonctionnement en termes de diffusion de listes aux opérateurs et d'interactivité. Les interactions sont établies à partir d'un formulaire HTML.

Dans cet exemple précis, nous considérons le réglage de cinq moteurs disposant chacun d'un attribut de priorité (la plus grande valeur correspond au moteur de plus haute priorité) : M1 (Priorité=1), M2 (Priorité=2), M3 (Priorité =2), M4 (Priorité =3), M5 (Priorité =1). Au départ, seule la liste des opérations de préparation est alimentée puisqu'il s'agit de la première opération de la gamme (préparation - réglage - vidange). Elle est récupérée par le CRM Préparation qui doit alors procéder aux allocations de ressources (1 BA + 1 opérateur pour chaque opération). Pour cela, il construit, après affectation des Bancs Auxiliaires (BA), une liste de trois opérations prioritaires destinée aux opérateurs de préparation.

L'opérateur qui en fait la demande peut obtenir, grâce à son formulaire HTML, cette liste composée de trois opérations : Préparation de M4 sur BA1, Préparation de M3 sur BA2, Préparation de M2 sur BA3. Dès qu'il valide son choix (ici le premier), la liste est mise à jour au niveau du SAI et lui est renvoyée immédiatement (Figure 49)⁸. Dans notre cas, il ne reste, après le choix de l'opérateur, que deux opérations possibles dans la mesure où il ne reste que 2 BA disponibles. Si l'unité disposait de 4 BA, la prochaine opération prioritaire pourrait être insérée (Préparation de M5 sur BA4).

En réalité, il vaudrait mieux bloquer l'opérateur, en le redirigeant éventuellement sur le premier écran, jusqu'à ce qu'il ait réalisé son opération et se considère comme à nouveau

⁸ Les actions et les listes associées au niveau du CRM Préparation et du Contrôleur Central sont présentés dans l'Annexe 3.

disponible. Si, par un hasard temporel, il arrive qu'une opération ait déjà été choisie, une indication est transmise à l'opérateur et il se voit remettre une liste remise à jour.

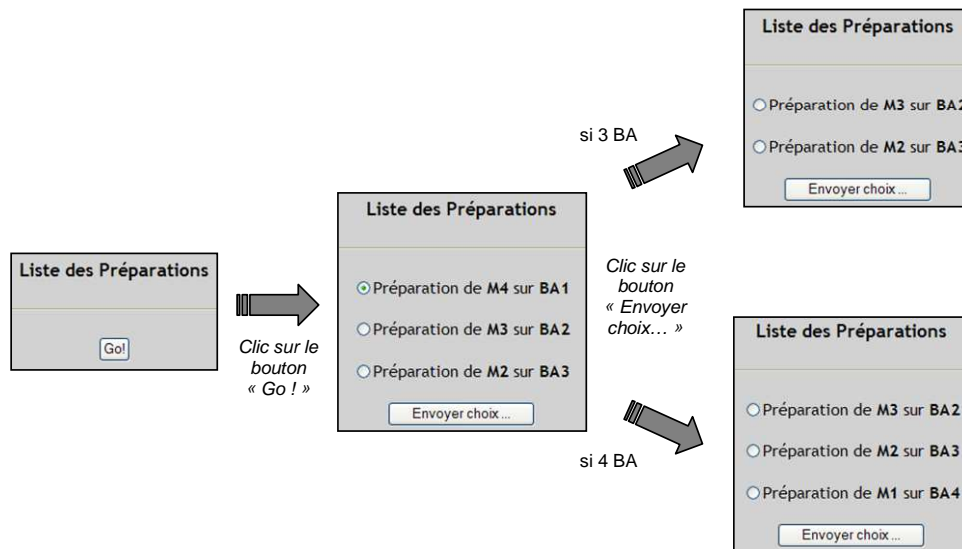


Figure 49. Enchaînement des écrans opérateurs.

Ces interfaces sont certes un peu « rustiques ». Elles sont obtenues à partir de feuilles de style XSL minimales. Il conviendra bien sûr d'étudier les interfaces pour emporter l'adhésion des utilisateurs pour une pleine implication, mais ceci ne fait pas l'objet de notre étude puisque nous ne gérons pas la partie opérative. Nous pourrions, pour les rendre plus conviviales, y intégrer le plan de l'atelier afin que les diverses opérations à effectuer se matérialisent par des points clignotants sur les endroits clés.

A travers ces interfaces, nous avons essayé de prouver l'interactivité et la réactivité de notre système. Il s'avère que la rapidité de traitement due à l'architecture distribuée choisie permet à priori de se rapprocher de la contrainte temps réel.

Cependant, nous déplorons de n'avoir pu expérimenter le SAI dans des conditions réelles. Les différents tests restent par conséquent assez limités. Les résultats ultérieurs d'application seront donc obtenus par le biais de la création d'un environnement de test qui reproduit l'envoi automatique de choix d'opérateurs et d'états provenant de cette unité opérationnelle.

5.4 Un environnement de test pour le SAI

Nous souhaitons réaliser un environnement permettant d'obtenir des résultats d'exécution par rapport à l'application du SAI sur une unité opérationnelle. Il est censé reproduire les comportements des différentes ressources de l'unité.

Il se veut réutilisable par d'autres unités à partir du moment où il est possible de décrire celle-ci en termes de listes de machines, listes d'opérateurs, et de gammes opératoires pour les produits passant à travers l'unité. Il s'agit d'une application JAVA capable de simuler le fonctionnement de l'unité, à savoir les actions des opérateurs et des machines. Cette application est basée sur un système de timers qui reproduisent les durées opératoires des différents éléments : les timers déclenchent les envois d'événements au SAI (états machines et choix des opérateurs) à l'aide de requêtes HTTP de type POST. Les timers permettent, en outre, d'établir des statistiques. L'unité est supposée ici fonctionner en mode normal, c'est-à-dire qu'aucune panne ne vient perturber le déroulement des opérations.

Une fois que l'unité est implémentée (classe Unité), l'environnement de test peut être lancé. Le squelette général de cet environnement se décompose comme suit :

1. Démarrage d'un timer général représentant le temps de production et servant à marquer les temps de cycle : ce timer ne s'arrête qu'à la fin du traitement du dernier produit.
2. Activation des Contrôleurs de Ressources SAI par le biais de requêtes HTTP de type GET afin que ceux-ci effectuent leurs décisions de pilotage.
3. Simulation des opérateurs.

Chaque opérateur demande une liste d'opérations au SAI (en réalité, à son Contrôleur de Ressources préalablement associé) grâce à une requête http de type GET. La réponse, donnée au format XML, est traitée et placée dans un tableau (de dimension 3). Le choix de l'opérateur est alors représenté par la désignation d'une des cases de ce tableau, la première case correspondant à l'action la plus prioritaire. Il existe plusieurs possibilités pour ce choix : choix prioritaire (première case du tableau), choix aléatoire (une des cases du tableau au hasard), ou encore choix le plus proche (un calcul de distances renvoie la case de l'action la plus proche). Dès qu'une case est désignée, le choix correspondant est renvoyé au SAI par une requête HTTP de type POST.

Dans le cas d'un opérateur sur équipement, celui-ci se déplace vers la machine concernée et attend le transport du produit. Dans le cas d'un transporteur, l'action est présumée commencer tout de suite. Un timer local, relatif au transporteur, est alors lancé et continue à tourner pendant toute la durée du transport incluant le chargement et le déchargement. A la fin du timer, le transporteur est débloqué, considéré comme apte à faire un nouveau choix. Le timer sera alors réinitialisé. La

machine d'origine est libérée (envoi de l'état libre par une requête HTTP de type POST) et l'opération sur la machine de destination peut débuter (envoi de l'état début_opération par une requête HTTP de type POST).

4. Simulation des équipements : elle consiste à envoyer des états à intervalles réguliers.

Si la machine nécessite l'intervention d'un opérateur, un timer relatif à l'opérateur est lancé dès que le produit arrive (puisque l'opérateur s'est déjà déplacé) et s'arrête à la fin de la durée prévue pour cette action (temps inférieur à la durée opératoire de la machine). Une fois le timer stoppé, l'opérateur est considéré comme débloqué et peut donc réaliser un nouveau choix.

Dans le même temps, un timer relatif à la machine est démarré. Au bout de la durée opératoire prévue, le timer est stoppé et un événement marquant la fin d'opération peut être envoyé (envoi de l'état fin_opération par une requête HTTP de type POST).

La gestion de timers (démarrage, arrêt, réinitialisation) nous permet donc d'obtenir un environnement à caractère événementiel (Figure 50).

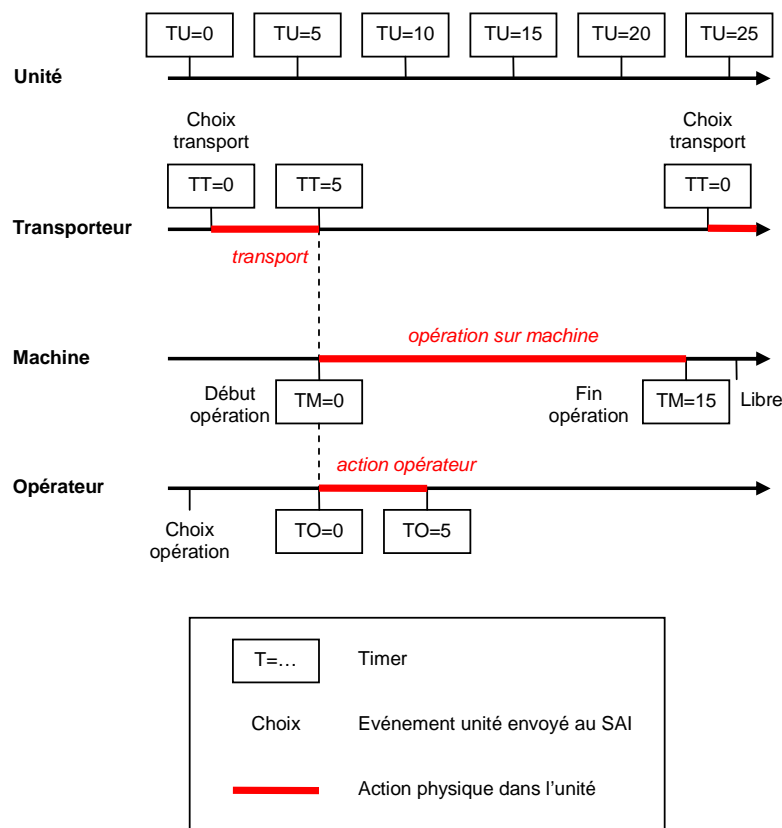


Figure 50. Gestion des timers pour l'environnement de test du SAI.

A partir de cet environnement, nous pouvons évaluer le fonctionnement du SAI selon deux critères :

- Le temps de cycle représente le temps de présence d'un produit dans l'unité depuis son placement dans le stock d'entrée jusqu'à son placement dans le stock de sortie. De même que le temps de production total, il doit être réduit au minimum.
- L'ordre de traitement des produits représente le respect de la priorité des produits. Les produits de plus haute priorité doivent être traités en premier.

6. Implantation du système

Le SAI a été conçu de telle manière qu'il soit le plus générique possible, tant dans son architecture fonctionnelle que dans son architecture de communication, et donc facilement implantable sur des unités opérationnelles préexistantes rencontrant des problèmes d'organisation en termes de tâches. Sa structure modulaire lui confère en effet des propriétés d'adaptation intéressantes.

6.1 Champs d'application du SAI

Il faut bien comprendre que le SAI convient aux unités opérationnelles dont l'ordonnancement global est complexe. Nous nous plaçons en effet dans le cas où, pour chaque phase d'une gamme opératoire, plusieurs ressources en parallèle peuvent postuler. On obtient alors une multitude de liaisons aussi bien entre les zones de stockage et les postes de charges qu'entre les postes de charge eux-mêmes : la combinatoire s'avère assez élevée. Le SAI n'aurait ainsi aucun sens à être appliqué à des simples lignes de transfert où il n'existe qu'un seul cheminement possible pour le produit. Dans ce dernier cas, il est bien évidemment inutile de fournir aux opérateurs la liste des opérations à exécuter.

De même, le SAI n'a pas vocation à supplanter systèmes de pilotage pour unités complètement automatisées. Ceux-ci ont largement fait leur preuve et sont capables de prendre en compte et de gérer de plus en plus d'aléas de production. En fait, étant donné sa nature interactive, le SAI prend toute sa valeur dans les unités opérationnelles possédant une forte composante humaine. Il est effectivement principalement centré sur les décisions des opérateurs et permet de les intégrer dans le pilotage de l'unité.

Plus particulièrement, le SAI s'adresse aux unités de grande dimension caractérisées par un process central long et coûteux, automatisé ou non, et nécessitant des opérations annexes autour de ce process supposant une intervention humaine. Un des objectifs majeurs du SAI est de permettre une meilleure organisation des transports au sein de l'unité en suggérant en temps réel aux transporteurs les opérations les plus prioritaires, dans la mesure où ceux-ci n'ont pas la visibilité adéquate pour prendre une décision optimale (d'autant plus si l'unité est cloisonnée). Il rend donc essentiellement service dans les unités où les transferts sont assurés par des moyens manuels et non par des AGV (Automated Guided Vehicle) ou autres

équipements de transferts automatiques. En ce sens, certaines unités hospitalières et leurs brancardiers peuvent aussi constituer un cas typique d'utilisation du SAI.

Ainsi, en dehors de son usage classique dans les unités de production, le SAI saurait être greffé à nombreuses autres unités opérationnelles, d'où la nécessité de décrire une procédure générale d'implantation.

6.2 Démarche d'implantation du SAI

Compte tenu de son caractère modulaire, le SAI peut être facilement ajusté à une unité donnée à partir du moment où il est possible d'en extraire les composantes génériques. Ainsi, nous définissons deux voire trois étapes importantes pour l'implantation du SAI sur une unité opérationnelle préexistante :

1. Identification des Contrôleurs de Ressources propres à l'unité.

Il s'agit, pour cela, de mettre en évidence les ensembles de ressources consacrées à la réalisation des opérations, ce qui revient à décomposer la séquence des opérations effectuées au sein de l'unité. En effet, chaque ensemble donne lieu un Contrôleur de Ressources. Pour chacun des ensembles, il est nécessaire de retrouver son type en fonction des ressources humaines qu'il met en jeu (statiques, semi-mobiles, totalement mobiles) afin de déduire le type du Contrôleur de Ressources associé (CRS, CRM, CRT). Une attention toute particulière doit être portée sur les opérations de transfert, étant donné que le nombre de CRT dépend des zones couvertes par les ensembles de ressources correspondants (de 1 à N secteurs, N représentant la totalité de l'unité). Cette première étape nous donne le nombre total de Contrôleurs de Ressources nécessaires en plus du Contrôleur Central et donc le modèle global du SAI. Chaque Contrôleur de Ressources identifiés deviendra un client HTTP du serveur Contrôleur Central, et sera serveur de son ensemble de ressources. A partir de ce moment, le SAI est « programmable » : il suffit de combiner les différents modules génériques implémentés pour chaque type de Contrôleur de Ressources.

2. Simulation du SAI couplé à l'unité (facultatif).

Cette étape permet de vérifier éventuellement l'interopérabilité du SAI avec l'unité concernée. En outre, la simulation peut contribuer à mieux appréhender l'unité et à faire ressortir les échanges unité/SAI.

3. Identification des échanges entre l'unité et le SAI.

Plus précisément, cette étape vise à analyser les différentes données transmises par les ensembles de ressources à leurs Contrôleurs de Ressources et la façon dont ces données peuvent être véhiculées. Ceci dépend entièrement de l'organisation spécifique de l'unité et des moyens informatiques mis à disposition.

Dans le paragraphe suivant, nous appliquons l'étape 1 et 3 de cette démarche à une unité hospitalière⁹.

6.3 *Adaptation du SAI pour une unité hospitalière*

Nous démontrons ici que le SAI est parfaitement adaptable à une unité autre qu'une unité industrielle, même si quelques changements mineurs s'imposent au niveau de la terminologie employée.

6.3.1 Le SAI face à la problématique des hôpitaux

Grâce au développement de l'informatique, les hôpitaux ont évolué dans trois grands domaines :

- la gestion de données pour les patients (base de données contenant les dossiers de patients, relations avec le système d'assurance) ;
- la gestion des biens et services commandés par l'hôpital;
- les nouvelles technologies à présent implantées dans les équipements médicaux.

En effet, les dépenses ont suivi les progrès de la médecine : les techniques sont plus efficaces et les docteurs sont plus qualifiés mais, en contrepartie, reviennent beaucoup plus cher. Les ressources matérielles et humaines sont rares, pas toujours disponibles et non multipliables en raison de leurs coûts, tandis qu'en parallèle la demande est en augmentation constante. D'autre part, les responsabilités consécutives aux opérations (comme la convalescence des patients) doivent aussi être prises en considération. Il est facile de comprendre que toutes ces contraintes sont difficiles à concilier pour gérer un service de manière optimale. L'aide à l'organisation des hôpitaux a été créée pour employer le plus efficacement possible les diverses ressources avec, comme objectif, la réduction des dépenses sans porter préjudice à la qualité de service.

⁹ Ces travaux ont été présentés à la conférence internationale IEEE SSSM'06.

Les unités industrielles connaissent et ont essayé de résoudre ce type de problème depuis longtemps, étant donné leur besoin constant de la rentabilité. Il n'est pas possible, cependant, de transposer directement le problème : l'unité industrielle traite des produits alors qu'une unité médicale traite des patients. Les soins médicaux constituent l'essence de la vie. Dans ce contexte, les concepts de productivité ou de rentabilité ne sont pas définissables pour une unité médicale. Néanmoins, les soins ont un coût qui ne cesse de croître et la complexité de l'organisation des hôpitaux apparaît alors aujourd'hui comme un réel point d'intérêt.

Par conséquent, il peut être intéressant d'appliquer les techniques développées dans l'industrie pour essayer d'organiser au mieux les blocs opératoires ou encore les services prenant en charge l'examen des patients. Dans notre cas précis, nous pensons que le SAI peut participer à une meilleure gestion d'une unité hospitalière pour deux raisons majeures :

- Contrairement aux autres systèmes de pilotage pour unités industrielles, le SAI se concentre vraiment sur l'optimisation des opérations qui sont dirigées par des humains. Cette caractéristique est d'une grande utilité dans un environnement hospitalier où les opérations manuelles sont fondamentales pour son bon fonctionnement ;
- Les objectifs sont plutôt semblables : une augmentation de l'efficacité et de la qualité du travail effectué dans unité. Dans les unités hospitalières, comme dans les unités industrielles, il est difficile de concilier une bonne utilisation des équipements et une bonne gestion du personnel.

L'intérêt d'un tel système dans un hôpital serait alors d'assurer une meilleure coordination entre les actions humaines environnantes et les activités de chirurgie ou d'examen des patients. Nous présentons ici un exemple d'application sur une unité hospitalière qui s'inspire du service d'endoscopie d'un grand hôpital de Lyon [115].

6.3.2 Présentation de l'unité d'endoscopie

L'endoscopie est un examen d'une cavité interne du corps humain, réalisé au moyen d'un endoscope (instrument muni d'un tube optique et d'un système d'éclairage que l'on introduit dans les cavités naturelles du corps afin de les examiner). L'activité d'endoscopie est une partie de l'activité de chirurgie digestive : elle contribue au diagnostic et au traitement de la pathologie digestive.

Il existe deux types d'endoscopie : l'endoscopie sous anesthésie générale (coloscopie, gastroscopie ou les deux) et l'endoscopie sans anesthésie (possible seulement dans le cas d'une gastroscopie). La différence essentielle entre ces deux types d'activité réside dans l'emploi des ressources humaines. En ce qui concerne les actes d'endoscopie sans anesthésie, les équipes sont constituées d'un Médecin Gastro-Entérologue (M_GE) et d'un Infirmier diplômé d'état (I). En ce qui concerne les actes effectués sous anesthésie, les équipes sont constituées, en plus du personnel précédent, d'un Médecin Anesthésiste-Réanimateur (M_AR) et d'un Infirmier Anesthésiste diplômé d'état (IA). Par la suite, nous désignerons par le terme d'endoscopie l'acte d'endoscopie sous anesthésie, et par le terme de gastroscopie l'acte d'endoscopie sans anesthésie.

Le service d'endoscopie (simplifié) se compose de 6 salles de gastroscopie, 4 salles d'endoscopie, d'une salle de réveil possédant 8 postes de réveil et d'une salle de lavage servant à la stérilisation des tubes (Figure 51). Les patients proviennent d'un lit d'hôpital dans le cas d'une endoscopie, tandis qu'ils arrivent directement de la salle d'attente dans le cas d'une gastroscopie.

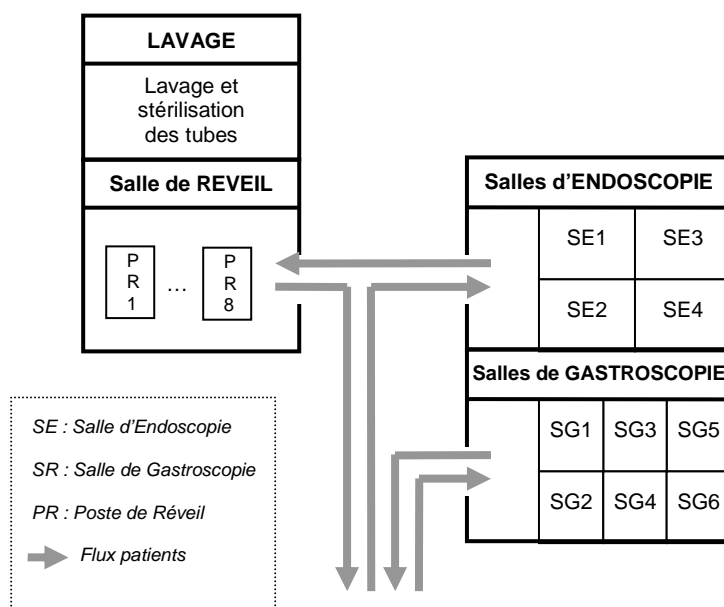


Figure 51. Représentation simplifiée du service d'endoscopie.

L'utilisation du SAI sur une telle unité permettrait d'assurer une meilleure coordination entre les opérations entourant les actes d'endoscopie/gastroscopie (à savoir le lavage des tubes et le réveil des patients) et surtout une meilleure organisation du transports des patients afin

d'éviter les pertes de temps. Les deux étapes de la procédure d'implantation du SAI sont détaillées dans les paragraphes suivants.

6.3.3 Identification des Contrôleurs de Ressources

La principale fonction du Contrôleur Central consiste ici à construire, pour une période donnée, la liste des opérations à exécuter en fonction du planning des patients à traiter. Pour identifier les divers Contrôleurs de Ressources et leur type, il faut d'abord répertorier les opérations nécessaires pour chaque type d'acte pratiqué. La différence entre les deux actes provient de l'anesthésie du patient.

Une endoscopie (acte avec anesthésie générale) se déroule comme suit :

1. transport du patient de son lit vers une des 4 salles d'endoscopie ;
2. anesthésie du patient, réalisée par un M_AR et un IA ;
3. endoscopie du patient, réalisée par un M_GE et un I au moyen d'un tube d'endoscopie stérilisé (durée : entre 15 et 20 min);
4. transport du patient de la salle d'endoscopie vers un des 8 postes de la salle réveil;
5. réveil progressif du patient sous la surveillance d'un I;
6. transport du patient de la salle de réveil vers son lit.

Une gastroscopie (acte sans anesthésie) se déroule comme suit :

1. gastroscopie du patient, réalisée par un M_GE et un I au moyen d'un tube de gastroscopie stérilisé, dans une des 6 salles de gastroscopie (entre 15 et 20min).

Afin de déterminer si la réalisation d'une opération implique ou non l'utilisation d'un Contrôleur de Ressources dans notre SAI (et si oui quel type de contrôleur), il est nécessaire de former les ensembles de ressources identiques à partir des séquences d'opérations précédentes.

- Les Salles de Gastroscopie

L'ensemble de ressources consacré à l'opération de gastroscopie est en partie composé de ressources matérielles : les Salles de Gastroscopie (SG) et les tubes stérilisés. Le personnel médical exigé (M_GE et I) est assigné de manière permanente à une SG, de sorte que l'allocation d'une équipe est inutile. Il s'agit donc de ressources dites statiques. Il convient de préciser que, même si la ressource SG ne peut communiquer lui-même son état actuel, un

employé est supposé capable de le communiquer au SAI. Conformément aux caractéristiques des Contrôleurs de Ressources, cet ensemble de ressources doit être pris en charge par un CRS nommé 'CRS Gastroskopie'.

- Les Salles d'Endoscopie

Les Salles d'Endoscopie (SE) peuvent être allouées de la même manière que les SG. La différence avec les SG vient de la gestion du personnel d'anesthésie exigé (M_AR et IA). Si une équipe d'anesthésie est assignée de manière permanente à chaque SE, l'ensemble est considéré comme un ensemble de ressources statiques. En réalité, les équipes d'anesthésie sont bien souvent en nombre inférieur aux SE. Il s'agit donc de ressources semi-mobiles où le personnel d'anesthésie est amené à choisir dans quelle SE intervenir. Une allocation complète nécessite donc l'allocation d'un SE (disposant de son M_GE, son I et son tube) et d'une équipe d'anesthésie. Conformément aux caractéristiques des Contrôleurs de Ressources, cet ensemble de ressources doit être pris en charge par un CRM nommé 'CRM Endoscopie'.

- Les postes de la salle de réveil

Un ou plusieurs Infirmiers (I) sont constamment présents dans la salle de réveil durant la période d'ouverture. En conséquence, l'allocation d'un I est inutile. Les Postes de Réveil (PR) constituent alors un ensemble de ressources statiques et l'allocation est faite en fonction de leurs états (un I est supposé capable de les communiquer au SAI). Conformément aux caractéristiques des Contrôleurs de Ressources, cet ensemble de ressources doit être pris en charge par un CRS nommé 'CRS Réveil'.

- Les transports de patients

Dans le cas d'une gastroscopie, les transports sont inutiles dans la mesure où les patients se rendent par eux-mêmes à la GR (après appel dans la salle d'attente). Dans le cas d'une gastroscopie, des Brancardiers (B) assurent les transports des patients de l'hôpital. Il s'agit bien évidemment des ressources de transfert contrôlées par un ou plusieurs CRT. Toutefois, nous supposons ici que le transport du patient d'une SE vers un PR est réalisé par un I de la salle de réveil, donc celui-ci n'est pas pris en compte. Il est possible de définir un CRT pour un ensemble de brancardiers associés à un ensemble de chemins couverts. Seul un CRT,

nommé 'CRT Transport', est créé ici car les deux trajets (Lit-SE, SE-Lit) sont couverts par le même ensemble de brancardiers. Nous supposons qu'un brancardier possède sa propre civière (ou est capable de déplacer le patient sur son lit).

- Le lavage (salle de lavage)

Le lavage et la stérilisation des tubes constituent une opération annexe effectuée par un infirmier (durée : entre 30 et 45 minutes). Un ou plusieurs Infirmiers (I) est assigné de manière permanente à la salle de lavage durant la période d'ouverture, de telle sorte que l'allocation d'un I est inutile. Les tubes de coloscopie et de gastroscopie peuvent être gérés de plusieurs façons. Si les tubes lavés sont considérés comme appartenant aux ressources matérielles de l'unité, un acte ne peut commencer sans la présence d'un tube lavé et la gestion du lavage des tubes ne nécessite pas la création d'un Contrôleur de Ressources spécifique. Dans ce cas, un des I précise simplement au SAI les états des tubes et les CRS Gastroscopie/Endoscopie en tiendront compte pour leurs allocations. Cependant, si les tubes sont en nombre restreints, il est possible d'assurer une meilleure gestion de l'opération de lavage-stérilisation en créant un Contrôleur de Ressources (de type CRS) qui pourraient indiquer au I le tube à laver à un moment précis. Pour simplifier, nous supposons ici que le nombre de tubes n'est pas limité et que ceux-ci sont lavés après chaque opération, donc nous n'avons pas besoin d'un nouveau Contrôleur de Ressources.

Au final, le service d'endoscopie génère quatre Contrôleurs de Ressources : le CRS Gastroscopie, le CRM Endoscopie, le CRS Réveil et le CRT Transport (Figure 52). Ils deviennent autant de clients HTTP du Contrôleur Central et de serveurs de l'unité.

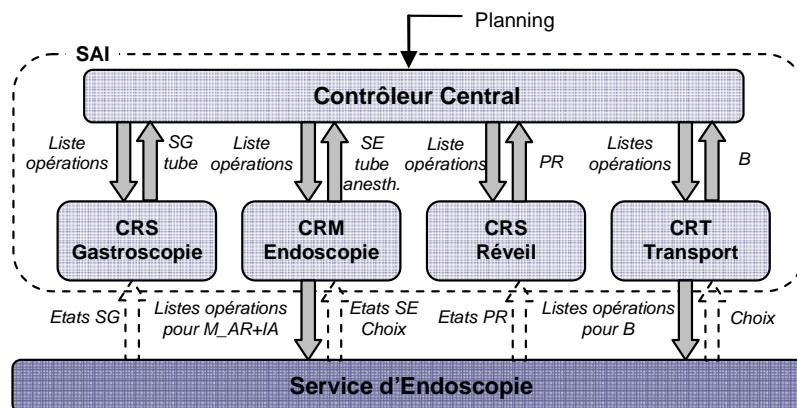


Figure 52. Le SAI de l'unité d'endoscopie.

6.3.4 Identification des échanges entre l'unité et le SAI

Il s'agit d'étudier plus exactement les échanges entre le service d'endoscopie et le SAI afin de préconiser des moyens de communications. En effet, pour effectuer sa décision d'allocation de ressources, le SAI a besoin d'événements provenant de l'unité : les états des ressources matérielles (SG, SE, tubes) et les opérations choisies par le personnel médical (équipe M_AR+IA, B). Les réactions du SAI à ces événements prédéterminés sont programmées dans la partie serveur des Contrôleurs de Ressources. Nous devons alors trouver, pour chaque Contrôleur de Ressource, quelles données de l'unité correspondent aux événements programmés et la façon dont elles peuvent être transmises au SAI.

Pour cela, il faut étudier le fonctionnement du SAI pour le traitement d'un patient. Les patients à traiter durant la période d'ouverture sont inscrits dans le planning du service. Le Contrôleur Central (CC) est capable d'établir, à partir de ce planning, des listes ordonnées d'opérations à exécuter destinées à chaque Contrôleur de Ressources. Nous supposons que les patients sont dotés d'une priorité en fonction de la place qu'ils occupent dans le planning.

- L'acte de gastroscopie

Pour un acte de gastroscopie, le CRS Gastroscopie recherche une SG inoccupée (contenant son équipe médicale), ainsi qu'un tube de gastroscopie lavé et stérilisé, afin de procéder à l'allocation de ressources (Figure 53). Le personnel médical assigné à la pièce (M_GE et I) signale quand l'acte commence (= état début_opération) et quand il finit (= état fin_opération). Une fois la gastroscopie réalisée, le traitement du patient est considéré comme fini. Dès que la SG est apte à recevoir un nouveau patient (= état libre), un membre du personnel est censé l'indiquer au SAI.

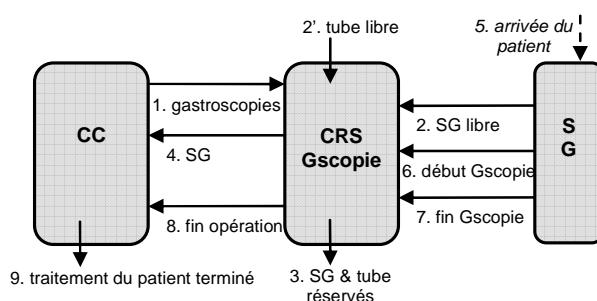


Figure 53. Le déroulement d'un acte de gastroscopie.

Ces communications peuvent être réalisées au travers d'un formulaire HTML programmé sur un ordinateur situé dans le hall attenant aux salles de gastroscopie. Le personnel d'une SG

(plus précisément un I) aurait alors simplement à cocher la SG concernée et à choisir son état dans une liste déroulante. La soumission du formulaire correspond à l'envoi d'état.

- L'acte d'endoscopie

Pour un acte endoscopie, le CRM Endoscopie recherche d'abord une SE inoccupée (contenant son équipe médicale), ainsi qu'un tube d'endoscopie lavé et stérilisé, afin de procéder à l'allocation de ressources (Figure 54a). Si une SE est libre, il propose par la suite l'acte d'endoscopie correspondant aux équipes d'anesthésistes (M_AR+IA). Aussitôt qu'une équipe choisit cet acte, cette information doit être retournée au CRM Endoscopie pour que celui-ci renvoie à l'allocation complète au Contrôleur Central. Etant donné que les équipes d'anesthésistes sont considérées comme mobiles, il serait judicieux de pourvoir les IA de dispositifs portables (type PDA) sur lesquels ils pourraient recevoir les listes d'anesthésies propriétaires, sous la forme d'un formulaire HTML, et cocher leur choix. La soumission du formulaire correspond à l'envoi du choix d'une équipe (requête HTTP de type POST). Dans le cas où ces équipes sont amenées à rester aux alentours des SE, le formulaire peut être présent sur le PC situé dans le hall attendant au SE (dans ce cas, l'équipe doit préciser son identifiant). En ce qui concerne les états des SE, ils sont envoyés, par un I de la SE, de la même manière que ceux des SG du fait que la procédure est assez similaire.

Lorsqu'une SE est allouée, Le patient doit y être transporté : une opération de transport entre le lit du patient et le SE est créée par le Contrôleur Central et proposée aux brancardiers par l'intermédiaire du CRT Transport (Figure 54b). Le brancardier qui décide d'accomplir ce transport transmet son choix au SAI. L'indication de fin de transport est ici inutile puisqu'elle n'induit aucune réaction du SAI. Il s'agit juste ici d'une action physique, le transfert au sein de l'unité, qui ne génère pas de nouvelle opération. Les brancardiers se déplaçant constamment dans l'hôpital, ils doivent être équipés de dispositifs portables pour pouvoir être informés des transports prioritaires et en choisir un (selon le même principe que les équipes d'anesthésistes).

Aussitôt que le patient arrive dans la SE, le personnel médical au complet est capable d'exécuter l'acte d'endoscopie traiter (anesthésie générale + endoscopie en elle-même). Le début et la fin de cet acte sont signalés au SAI par le I de la SE.

Une fois l'endoscopie finie, elle est suivie d'une phase dite de réveil afin le patient puisse se remettre de l'anesthésie : une opération de réveil est créée par le Contrôleur Central. Le CRS

Réveil recherche un PR disponible pour l'affecter au patient (Figure 54c). Chaque libération de PR fait l'objet d'un signalement au SAI (= état libre) effectué par un des I présents dans la salle de réveil. Il en est de même lorsqu'un PR devient occupé (= état début_opération). Dès que le patient est prêt à retourner dans son lit d'hôpital, le I doit le déclarer au SAI qui crée alors une opération de transport entre le PR et le Lit. Le PR est considéré comme libre quand le transport débute. Les communications entre le I et le SAI nécessitent l'installation d'un ordinateur dans la salle de réveil afin que le I puisse transmettre les états des PR.

De retour dans son lit, le traitement du patient est considéré comme fini.

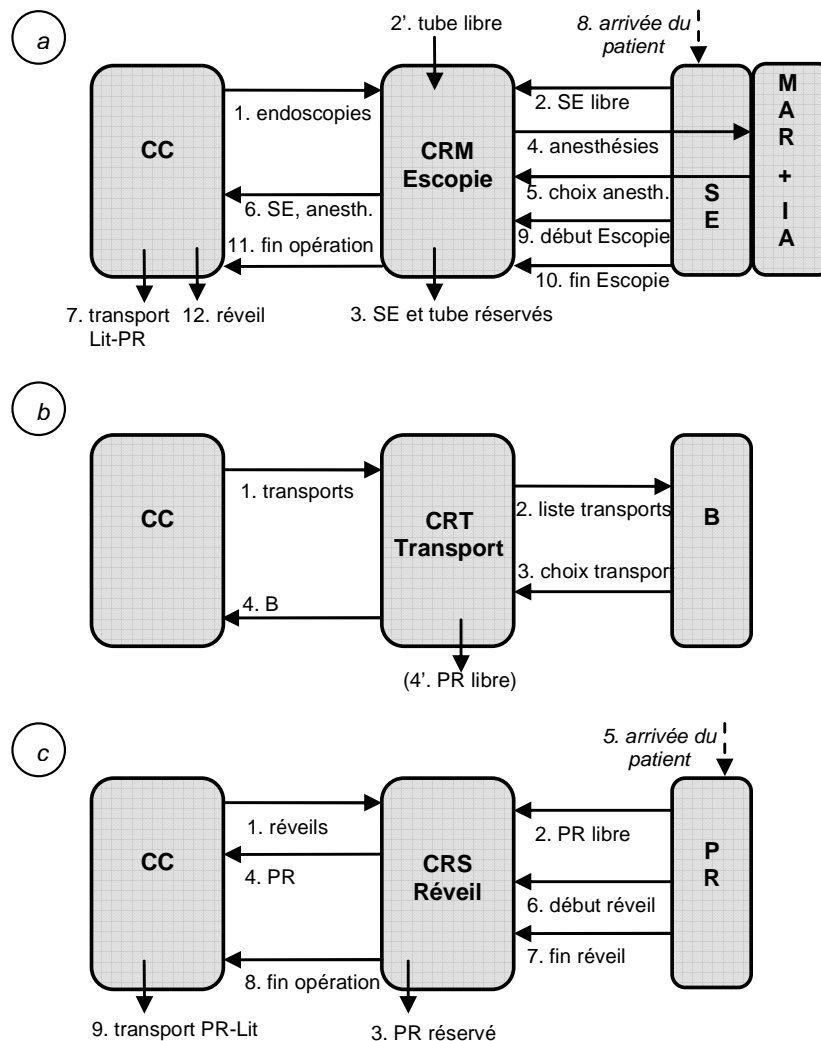


Figure 54. Le déroulement d'un acte d'endoscopie.

Il convient de préciser que, dans les deux cas, la gestion des états des tubes se fait directement au travers du SGBD associé à la BD Ressources : le personnel de la salle de lavage indique les tubes lavés et stérilisés ainsi que les tubes qui partent pour un examen.

Notre SAI est donc parfaitement adaptable à ce genre de service. Il permet d'assurer le bon déroulement du traitement des patients et permettrait, de même, d'intégrer les urgences en utilisant la notion de priorité : le patient arrivant en urgence est doté d'une priorité supérieure aux autres afin que le SAI privilégie son traitement.

Pour réaliser cette adaptation, nous avons admis que les patients sont assimilables, d'un point de vue terminologique, aux produits, le planning au plan de tâches, les équipements médicaux aux machines et le personnel médical aux opérateurs.

Cependant, les résultats d'application ne concerneront pas cette unité. En effet, disposant de données plus précises pour l'unité de réglages moteurs, celle-ci reste notre unité référence pour tester les performances du SAI.

7. La notion de confiance

Les résultats donnés ici portent sur l'unité industrielle de réglages de moteurs (§4.4.1). Les tests effectués permettent de visualiser le comportement du SAI dans des conditions optimales, mais permettent aussi de faire ressortir ses lacunes vis-à-vis du pilotage humain. La notion de confiance sera alors introduite pour pallier les problèmes relevés¹⁰.

Notre échantillon de test comporte 40 moteurs. Les priorités de ces moteurs sont réparties de façon équitables : 10 moteurs pour chaque priorité, les priorités allant de 1 à 4 (un moteur de Priorité4 étant le plus urgent). L'environnement de test reproduit alors le passage de ces 40 moteurs dans l'unité physique et les interactions avec le SAI en charge du bon déroulement de la production.

7.1 *Les résultats d'utilisation du SAI en conditions optimales*

Une utilisation optimale du SAI signifie que les opérateurs choisissent toujours la première opération dans leur liste de suggestions, à savoir l'action la plus prioritaire, et exécutent réellement l'opération choisie dans le temps imparti. Les résultats doivent être alors assez proches des optima donnés par la simulation.

Dans cet exemple précis, la totalité des moteurs de haute priorité (PR4) sont traités en moins de 162 minutes (Figure 55). Nous obtenons ici un temps référence pour une production qui respecte les contraintes temporelles des moteurs : 420 minutes (= 7 heures). Les moteurs de même priorité sont groupés et subissent le processus de réglage de manière ordonnée selon la valeur de priorité.

Les résultats sont assez semblables lorsque l'opérateur choisit une opération au hasard dans la liste d'opérations proposées ou encore lorsqu'il choisit l'opération la plus proche de son emplacement actuel puisqu'il s'agit quand même d'opérations qualifiées de prioritaires par le SAI et que les priorités sont de toute façon très rapidement réordonnées.

Ces premiers résultats voient en l'opérateur un agent disponible et fiable. Dans une unité gérée par le SAI, un opérateur est tout à fait libre : il peut décider d'exécuter ou non une opération suggérée en fonction de ses propres contraintes (pauses, interventions de

¹⁰ Ces travaux ont été présentés au symposium international IEEE ISIE'07 et font l'objet d'un article dans la revue International Journal of Factory Automation, Robotics and Soft Computing (2007).

maintenance, ...). Cependant, lorsqu'il choisit une action, il est important qu'il la réalise aussi rapidement que possible et de manière correcte, ce qui n'est pas forcément garanti.

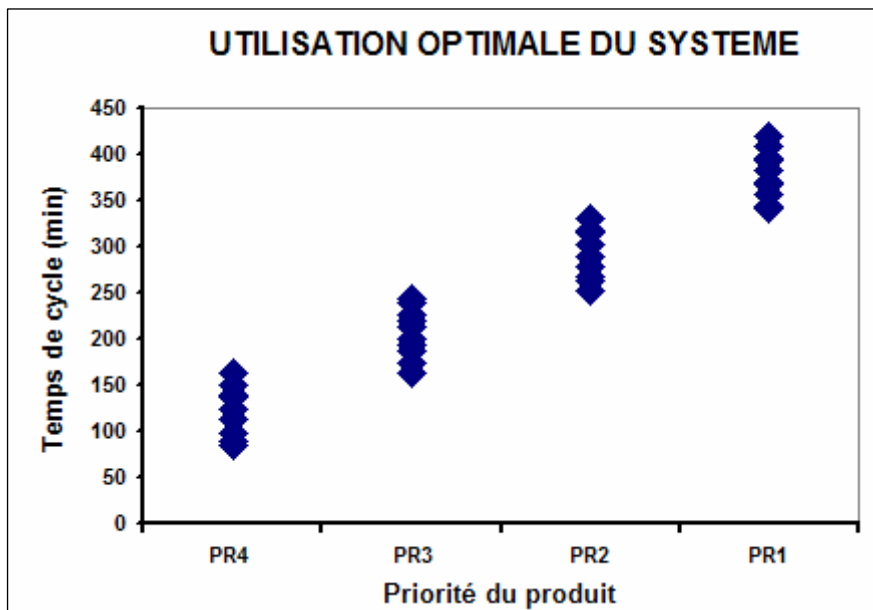


Figure 55. Temps de cycle des moteurs dans le cadre d'une utilisation optimale du SAI.

7.2 Les problèmes de confiance

La nature même de notre SAI suppose que le pilotage reste sous la responsabilité des opérateurs de l'unité. Chaque décision d'un opérateur a un effet sur les listes d'opérations proposées et a donc un impact sur le fonctionnement global de l'unité. Cette caractéristique pourrait devenir un défaut si les opérateurs ne coopèrent pas comme prévu. En réalité, le SAI repose sur l'implication et la bonne foi des opérateurs. Nous recensons trois problèmes majeurs liés à la confiance que l'on peut avoir ou non en un opérateur [116]:

- A. un opérateur n'exécute pas l'opération qu'il a choisie;
- B. un opérateur décide d'exécuter une opération qui n'est pas suggérée par le SAI ;
- C. un opérateur ne valide pas une opération qu'il a exécutée.

Il est essentiel pour le SAI de prendre en considération ces problèmes potentiels afin d'éviter un blocage du système. Des procédures de recouvrement ont donc été prévues dans le SAI. Pour discerner et corriger ces erreurs, chaque Contrôleur de Ressources procède à quelques vérifications basées sur des événements provenant de l'unité et transmet le résultat au Contrôleur Central pour que celui-ci rectifie les listes d'opérations en conséquence :

- A. Calcul de la durée de l'action d'un opérateur.

Un timer est lancé puis arrêté à la réception de différents événements (comme le début d'une opération sur équipement). Si le temps excède un seuil donné, le système présume que l'opération n'a pas été réalisée et ajoute de nouveau l'opération à la liste concernée.

B. Comparaison des produits.

Dès que le produit est placé sur l'équipement, son identifiant (`product_id`) est supposé être envoyé automatiquement avec l'état de l'équipement. Le système peut alors comparer celui-ci avec le `product_id` normalement alloué à cet équipement. Si les identifiants ne sont pas égaux, le système considère que la ressource de transfert a fait une erreur et a, de ce fait, effectué une action non suggérée. Le système enlève alors l'opération exécutée de la liste concernée et ajoute de nouveau l'opération prévue à la liste. Ce type d'erreur n'est pas possible pour les autres types de ressources dans la mesure où elles ne peuvent pas agir avant que le produit ne soit transporté.

C. Examen de la liste des opérations.

Au commencement d'une opération sur équipement, le système vérifie sa présence dans la liste concernée. Si elle est toujours dans la liste, le système suppose que l'opération a été exécutée sans avoir été choisie et l'enlève de la liste.

Bien que ces problèmes soient corrigés par notre SAI, ils sont susceptibles de provoquer des effets négatifs sur l'avancement de la production. Pour illustrer ce propos, nous avons mesuré les effets sur le flux de production de conditions d'utilisation du SAI différentes des conditions optimales : non réalisation des opérations choisies (cas A) et non respect des opérations proposées (cas B). Le cas C n'est pas pris en compte puisqu'il n'a aucun impact réel sur l'unité : une opération de la liste est réalisée dans un temps correct et, même si cette action n'a pas été validée, elle peut être rapidement enregistrée par le SAI à la réception de l'état de l'équipement correspondant. Notre environnement de test, après quelques modifications, est capable d'intégrer les problèmes d'utilisation et de tester donc ces changements.

7.2.1 Non réalisation d'opérations (cas A)

L'exemple relatif au cas A porte sur un opérateur de préparation qui n'effectue pas une opération sur deux (en raison d'autres activités inattendues).

Par rapport à une utilisation optimale du SAI, les priorités sont toujours respectées, mais le temps de production global a considérablement augmenté (Figure 56) : 519 minutes (= 8 heures 39 minutes) au lieu de 420 minutes. En fait, avant que le SAI ne détecte que l'opération n'est pas exécutée, les équipements restent réservés et ne peuvent être assignés à d'autres opérations, ce qui explique ce long retard. Le cas A soulève donc des problèmes significatifs au niveau du temps de production.

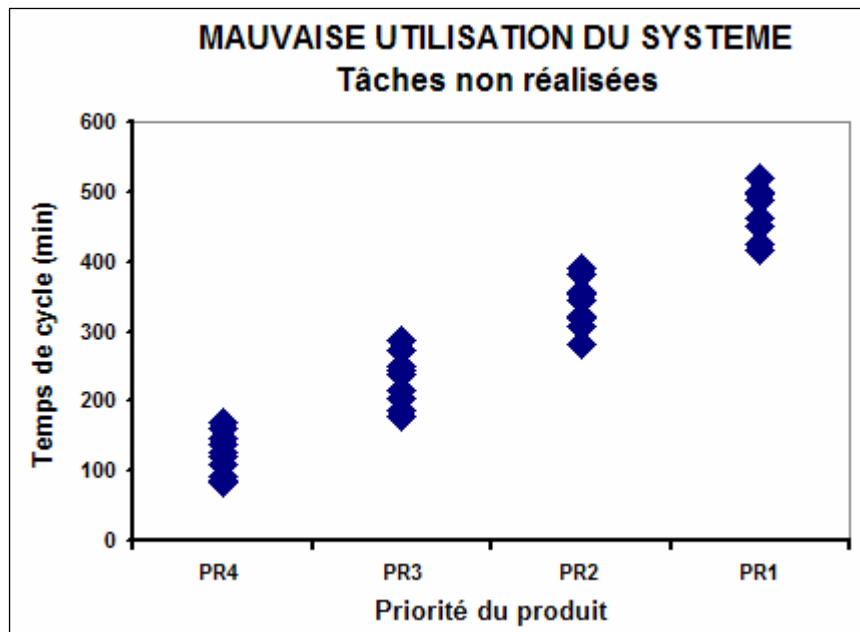


Figure 56. Temps de cycle des moteurs dans un cas de mauvaise utilisation du SAI (A).

7.2.2 Non respect des opérations (cas B)

L'exemple relatif au cas B porte sur les conducteurs de chariot élévateur qui n'exécutent pas correctement les opérations de transport dans le secteur StockEntrée-BA : les moteurs sont enlevés du stock au hasard (c'est-à-dire que les conducteurs de chariot élévateur ne suivent pas les numéros d'emplacement transmis avec les opérations de transport suggérées).

Par rapport à une utilisation optimale du SAI, le temps de production global est correct (7 heures 10 minutes), mais les priorités de produits ne sont pas respectées (Figure 57). Il est possible qu'un moteur de haute priorité (PR4) ne soit traité dans l'unité qu'après 422min. En réalité, les premières opérations de transport déterminent l'entrée de tel ou tel produit dans le processus de réglage, ce qui explique ici qu'un moteur de haute priorité peut être réglé beaucoup plus tard qu'un moteur de priorité plus faible. Le cas B soulève donc des problèmes significatifs au niveau du respect des priorités.

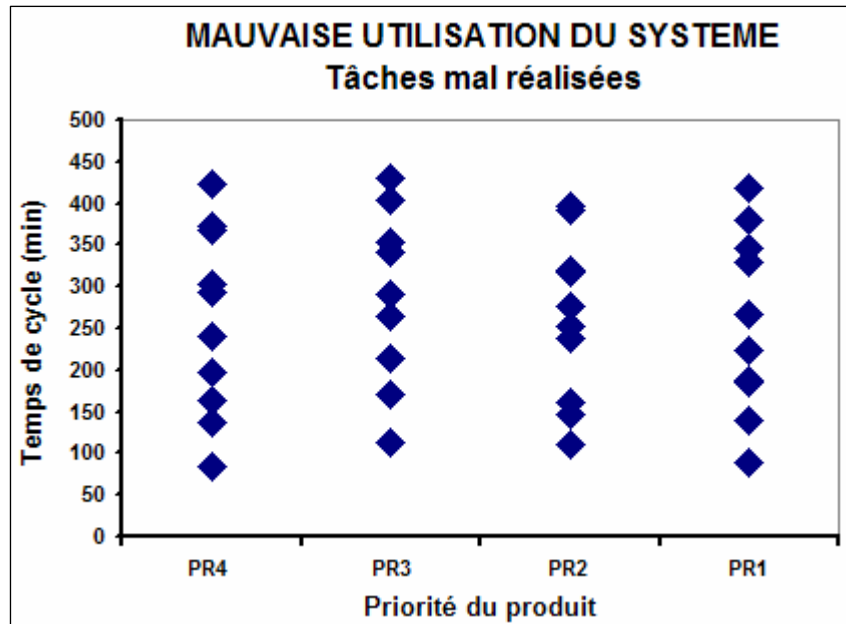


Figure 57. Temps de cycle des moteurs dans un cas de mauvaise utilisation du SAI (B).

En résumé, les conséquences d'une mauvaise utilisation du SAI démontrent que les interactions entre les opérateurs et le système sont cruciales pour la performance d'une unité. L'accumulation de problèmes dans les interventions humaines peut totalement désorganiser l'unité. Un bon fonctionnement implique que les propositions faites par le SAI soient vraiment suivies par les différents opérateurs.

Aussi, bien que l'interactivité constitue un des points fondamentaux du SAI, elle est aussi son point faible étant donné que les opérateurs peuvent être amenés à bloquer le système. Pour remédier à ce problème, nous nous proposons d'intégrer des mécanismes de calcul de confiance à l'intérieur du SAI. L'intérêt serait d'estimer, à un moment donné, la confiance que le système peut avoir en chaque opérateur et donc d'évaluer la fiabilité du système.

7.3 Un moteur de confiance pour le SAI

Le SAI ne peut fonctionner correctement et rendre les services attendus que si les opérateurs de l'unité coopèrent pleinement avec lui. La crédibilité des opérateurs peut alors être vue comme une faiblesse du système. Il est alors nécessaire d'introduire la notion de fiabilité du système : le SAI est d'autant plus fiable que les opérateurs sont de bonne foi (c'est-à-dire qu'ils respectent leurs choix).

Dans cette optique, nous envisageons de coupler au SAI un moteur de confiance afin de déterminer s'il peut réellement compter sur les opérateurs de l'unité et de préserver ainsi la qualité générale du système. Nous nous sommes, pour cela, inspirés des travaux précédents dans le domaine du calcul de confiance.

7.3.1 Les moteurs de confiance

Dans le monde réel, la confiance est utilisée entre deux personnes qui interagissent afin d'aller de l'avant, d'agir en dépit d'une incertitude quant au résultat de l'interaction, qui peut être négatif. Plus exactement, l'entité, appelée décideur, qui reçoit une requête de la part de la seconde entité, le demandeur, utilise le niveau de confiance dans ce demandeur pour décider si elle doit répondre favorablement à cette demande. Le but d'un moteur de confiance est de fournir une version informatique du concept humain de confiance. Il a la charge d'évaluer un niveau de confiance.

Marsh a introduit un des premiers modèles informatiques de calcul de confiance [117]. Une valeur de confiance calculée peut être vue comme la représentation numérique de la fiabilité ou du niveau de confiance de l'entité considérée. Une valeur de confiance est en fait définie comme une estimation du comportement futur de l'entité dans un contexte donné. L'estimation en elle-même est construite sur des preuves passées. Il est par exemple possible de surveiller en permanence les interactions entre entités et de comptabiliser à tout moment les résultats positifs et négatifs des interactions précédentes : la valeur de confiance est alors calculée comme le nombre de résultats positifs sur le nombre total de résultats.

Les composants de base d'un moteur de confiance incluent obligatoirement un composant de processus décisionnel (Figure 58). Celui-ci est invoqué quand une entité sollicitée, le décideur, doit statuer sur la suite à donner à une requête faite par une autre entité, le demandeur.

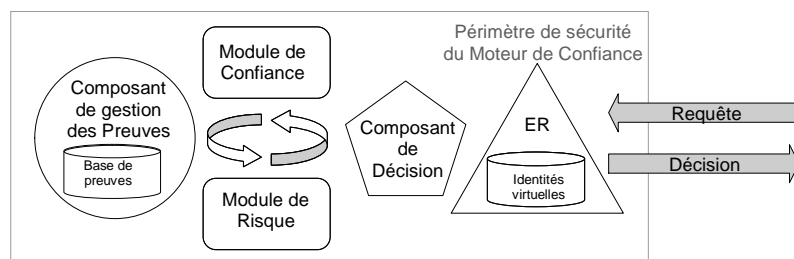


Figure 58. Architecture d'un moteur de confiance.

En fait, le composant de décision utilise deux sous-composants pour décider d'accorder ou non une demande :

- Le module de confiance calcule dynamiquement la valeur de confiance du demandeur en se basant sur des preuves. Une preuve est typiquement une observation directe du résultat d'une interaction précédente ou une recommandation donnée par une troisième entité.
- Le module de risque évalue dynamiquement le risque associé à l'interaction.

L'action choisie doit maintenir un ratio coût/bénéfice approprié. Selon le système de confiance, le poids de la valeur de confiance dans la décision finale peut être faible.

En arrière-plan, un autre composant prend en charge la collecte de preuves. Le composant de gestion des preuves collabore avec d'autres moteurs afin de maintenir une base de preuves à jour et adaptée au contexte. Cette base est utilisée pour mettre à jour les niveaux de confiance et de risque. Ainsi, risque et confiance évoluent en fonction du cycle de vie des preuves.

Le dernier composant d'un moteur de confiance est le module de reconnaissance des entités, appelé ER (Entité Reconnaissance) [118]. En effet, il est important de savoir quel demandeur fait une requête afin de retrouver le niveau de confiance. Il est responsable de la reconnaissance des entités qui interagissent, au moyen d'identités virtuelles ou de pseudonymes. La valeur de confiance retournée pour chaque entité peut être employée pour choisir l'entité la plus digne de confiance parmi un ensemble d'entités.

Dans le projet européen SECURE [119], des investigations ont été menées concernant les mécanismes de sécurité dynamiques et les propriétés d'auto-configuration liées à un calcul global de la notion humaine de confiance. Le résultat de ce projet est un moteur de confiance élaboré, basé sur un modèle formel de confiance. Le moteur de confiance se présente sous la forme d'une API Java et peut être appliqué à de nombreux domaines.

Les moteurs de confiance arrivent en force dans le domaine de l'intelligence artificielle. Pour l'heure, il est surtout utilisé pour assister l'être humain utilisateur dans les moments où celui-ci a besoin d'aide pour déterminer le niveau de confiance qu'il a en un dispositif (distributeur de billets située dans une zone malfamée) ou un message (susceptible d'être un SPAM). Bien que les premières applications existent déjà, il semble que les modalités d'interactions entre les utilisateurs et les moteurs de confiance demandent à être personnalisées selon le type

d'application et optimisées pour que l'utilisateur communique naturellement avec le moteur de confiance en vue de l'améliorer.

7.3.2 Application des concepts des moteurs de confiance au SAI

Les moteurs de confiance apparaissent donc comme une solution judicieuse pour résoudre en partie, ou en tout cas discerner, les problèmes de crédibilité éventuels quant aux décisions des opérateurs de production. Nous montrons ici comment de tels concepts peuvent être combinés avec notre SAI. Nous définissons ainsi clairement les éléments qui permettront d'établir le niveau de confiance [120].

Dans le SAI, les choix des opérateurs et l'accomplissement effectif des actions correspondantes doivent être constamment vérifiés. Les vérifications sont établies à partir de la réception ou non d'états provenant de l'unité. Chaque observation émanant de ces tests constitue ce que nous considérons comme une preuve utilisée pour peupler la base de preuves du moteur de confiance. Nous avons précédemment énuméré les circonstances principales dans lesquelles l'incertitude vis-à-vis des opérateurs est avérée et où le besoin de calcul de confiance se fait alors ressentir : non réalisation d'opérations (cas A), non respect d'opérations (cas B) et réalisation d'une opération non choisie (cas C). Nous pouvons attribuer un taux d'erreur à chacune de ces situations selon leur impact sur le fonctionnement du SAI et sur le déroulement de la production : les situations les plus risquées ont les taux d'erreur les plus élevés (Figure 59). Nous avons opté pour un taux de 100% dans le cas A, 90% dans le cas B et 50% dans le cas C.

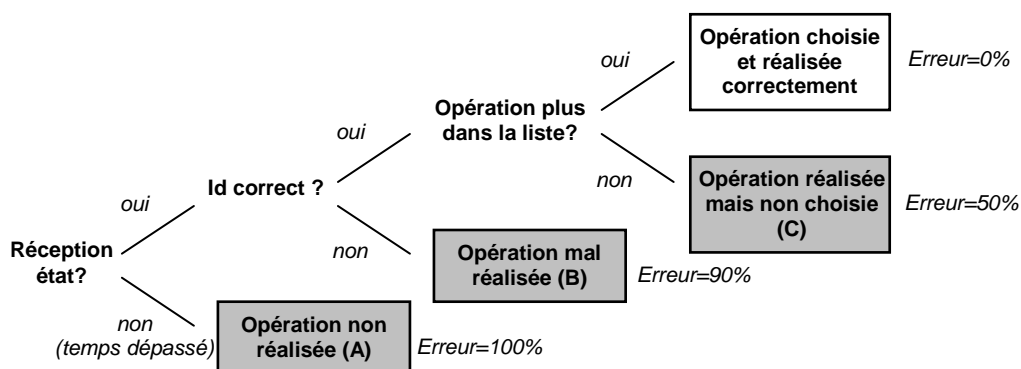


Figure 59. Arbre des observations.

Par conséquent, l'observation (Obs) liée à l'action d'un opérateur (Act) peut être formalisée comme un triplet :

$$\forall \text{Act}, \exists \text{Obs}=(\text{operator_id}, \text{CR_id}, \text{erreur}) \quad \text{erreur} \in [0, 100]$$

'operator_id' représente ici l'identifiant de l'opérateur concerné par l'action Act et 'CR_id' représente le Contrôleur de Ressources qui gère cette action. 'erreur' reflète l'erreur dans la réalisation de l'action et dépend du type de situation (A,B,C).

Une telle structure nous permet d'évaluer les performances passées des opérateurs selon quatre degrés de précision :

- L'ensemble des triplets contenant le même operator_id et le même CR_id est associé à la performance de l'opérateur pour l'exécution de l'opération gérée par le Contrôleur de Ressources.
- L'ensemble des triplets contenant le même operator_id est associé à la performance globale de l'opérateur.
- L'ensemble des triplets contenant le même CR_id est associé à la performance globale d'un groupe d'opérateurs exécutant une opération.
- L'intégralité des triplets est associée à la performance globale de la totalité des opérateurs de l'unité.

Nous soutenons qu'un Contrôleur de Ressources est capable de créer un triplet d'observation Obs chaque fois qu'un de ses opérateurs envisage l'exécution d'une action. Dès que le Contrôleur de Ressources détecte une erreur dans l'accomplissement de l'opération, il construit Obs avec le taux d'erreur correspondant (équivalent à un résultat négatif de l'interaction). Si aucune erreur n'a été détectée pendant le contrôle, il peut alors construire Obs avec un taux d'erreur égal à 0 (équivalent à un résultat positif de l'interaction). Cette observation est automatiquement transmise au moteur de confiance pour mettre à jour sa base de preuves.

A partir de ces preuves, le moteur de confiance peut estimer dynamiquement le risque relatif à l'action d'une entité opérateur en calculant la moyenne entre les taux d'erreur de la performance passée de l'opérateur dans le même contexte. Le SAI se réfère alors au moteur de confiance pour s'informer des possibilités d'échec d'une opération (Figure 60). De la même manière, le moteur de confiance peut estimer dynamiquement la valeur de confiance d'une entité d'opérateur en calculant le complément de son taux d'erreur moyen. De façon simplifiée, une valeur de confiance croît grâce aux résultats positifs des interactions et décroît à cause des résultats négatifs des interactions.

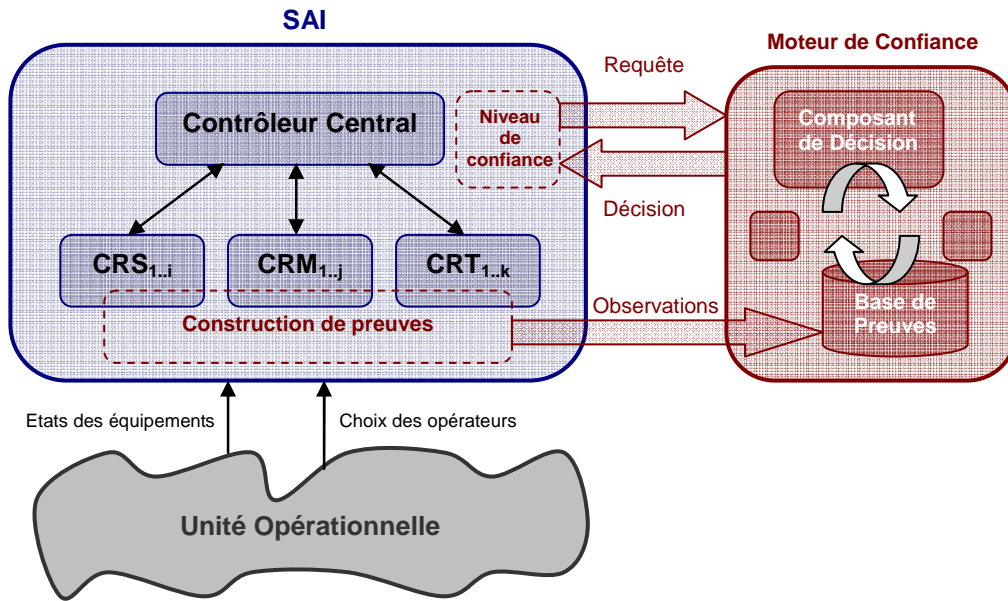


Figure 60. Un moteur de confiance couplé au SAI.

7.3.3 Application à l'unité de réglages moteurs

Dans notre cas, l'emploi d'un moteur de confiance présente deux intérêts majeurs :

- En fonction de la valeur de confiance calculée, le SAI pourrait utiliser le moteur de confiance pour décider d'accorder ou non à un opérateur l'exécution d'une action. La décision serait alors basée sur la politique de risque configurée dans le moteur de confiance par l'administrateur du système. Par exemple, l'opération ne serait pas acceptée si la valeur de confiance de l'opérateur est au-dessous d'un seuil fixé à 80 %.
- En agrégeant quelques valeurs de confiance, le SAI pourrait obtenir des indicateurs sur sa performance de fonctionnement et les transmettre au manager d'unité pour la mise en place de mesures correctives. Il convient en effet de définir des zones de fonctionnement indiquant la fiabilité actuelle du SAI : zone verte pour un bon fonctionnement, zone orange pour un fonctionnement moyen, zone rouge un mauvais fonctionnement,... Le moteur de confiance pourrait simplement calculer, par exemple, la moyenne de toutes les valeurs de confiance des opérateurs appartenant à un Contrôleur de Ressources spécifique afin de déterminer la zone de fonctionnement relative à chaque opération de l'unité.

Pour illustrer ces objectifs, nous proposons ici deux exemples. Les résultats de calcul de confiance ont été produits à partir de notre unité de réglages moteurs. Pour chaque problème

de confiance étudié précédemment pour les cas A et B (§6.3.3), nous donnons les valeurs de confiance calculées par le moteur de confiance intégré dans le SAI (Tableau 4).

| | Non réalisation d'opérations (A) | Non respect d'opérations (B) |
|----------------------|--|------------------------------------|
| OP_BA_1 | 50 | 100 |
| OP_BA_2 | 100 | 100 |
| OP_BA_3 | 100 | 100 |
| OP_BA_4 | 100 | 100 |
| CH1 | 100 | 69 |
| CH2 | 100 | 78 |
| CH3 | 100 | 92 |
| CH4 | 100 | 88 |
| Préparation | 77 | 100 |
| Vidange | 100 | 100 |
| Transport Stock-BA | 100 | 17 |
| Transport BA-BR | 100 | 100 |
| Transport BR-BA | 100 | 100 |
| Transport BA- Sortie | 100 | 100 |
| Unité | 96 | 87 |

Tableau 4. Valeurs de confiance (en %) des opérateurs de l'unité de réglages moteurs.

Les résultats négatifs inclus dans le calcul de confiance sont de la forme suivante :

A. Obs = (OP_BA_1, CRM Préparation, 80)

B. Obs = (CH1, CRT Transport, 90)

Si l'opérateur de préparation OP_AB_1 n'exécute pas une opération sur deux alors qu'il les a choisies (cas A), sa valeur de confiance est au final abaissée à 50 %. Par conséquent, le SAI pourrait le considérer comme non fiable et l'empêcher, par la suite, de s'occuper des opérations préparation urgentes.

Si les conducteurs de chariot élévateur (CH) n'exécutent pas correctement les opérations de transport du Stock Entrée vers les BA de préparation (cas B), leurs valeurs de confiance diminuent également de façon progressive. Néanmoins, ces valeurs restent correctes étant donné que le nombre total d'opérations de transport couvrant l'unité est assez élevé : les résultats positifs des autres transports compensent en grande partie les résultats négatifs des transports concernant le premier secteur. Au contraire, la valeur de confiance globale pour le premier secteur est très faible et reflète bien alors les problèmes de transport dans ce secteur. Ce type d'information est significatif pour le manager d'unité qui peut s'appuyer sur ces valeurs pour instaurer des actions correctives au niveau du secteur concerné.

Dans chaque cas, la valeur de confiance globale de l'unité n'est pas vraiment diminuée en raison du grand nombre d'opérations. Le but est d'atteindre 100 % pour une optimisation parfaite du plan de production.

En résumé, l'introduction de la notion de confiance dans le SAI permet de distinguer les points problématiques afférents aux décisions humaines. La non fiabilité d'un opérateur peut en effet nuire à la qualité générale du système. Actuellement, le SAI est capable de détecter les problèmes de confiance et de construire les observations correspondantes. Cependant, il est encore nécessaire d'implémenter le composant de décision qui donnerait suite aux résultats des interactions opérateurs/SAI afin de faire du SAI un outil vraiment robuste.

Dans l'avenir, les moteurs de confiance sont appelés à être de plus en plus employés pour résoudre les problèmes d'ordre humain. Ils peuvent être particulièrement utiles dans le monde de la production où le nombre d'intérimaires est élevé. En effet ceux-ci sont source d'erreurs étant donnée leur faible connaissance de l'entreprise.

Conclusion

L'objectif de cette étude était de proposer un outil pour le pilotage des unités opérationnelles à composante humaine, dans le sens où le système construit doit permettre une interactivité avec les opérateurs afin d'optimiser le flux de production en fonction de leurs contraintes. Le système repose sur un triptyque simple : suggestion – décision – adaptation. Il calcule, à partir d'un plan de tâches, les meilleures opérations à effectuer par les opérateurs, leur suggère et s'adapte en temps réel à leurs décisions finales. L'adaptation passe par une révision éventuelle de l'ordre des opérations futures et une allocation dynamique des ressources, d'où la nécessité d'un pilotage dit réactif. La fréquence de mise à jour doit être paramétrée de façon à garantir une certaine stabilité au système.

Cette étude s'est articulée autour du choix d'une architecture fonctionnelle générique pour le système et des contraintes organisationnelles que cela impose en termes d'échanges, de solutions de développement et d'implantation.

Ainsi, ayant admis que le découpage fonctionnel du système doit être effectué sous l'angle des décisions humaines, il apparaît essentiel d'adopter une architecture de pilotage modulaire. Nous avons en effet établi que, dans le cadre d'un choix d'opération, l'assistance fournie aux opérateurs varie en fonction du degré de structuration de leurs décisions et donc de leurs périmètres d'action. Trois sous-systèmes génériques de pilotage ont donc été identifiés, chacun prenant en compte un type de ressources : le Contrôleur de Ressources Statiques, le Contrôleur de Ressources Mobiles et le Contrôleur de Ressources de Transfert. Sachant qu'à chaque opération est associée un type de ressources, le système global peut être considéré comme une combinaison de sous-systèmes Contrôleurs de Ressources. Un Contrôleur de Ressources possède sa propre latitude décisionnelle mais, pour assurer la synchronisation des opérations, l'ensemble des Contrôleurs de Ressources est supervisé par un sous-système appelé Contrôleur Central. Nous avons donc opté pour une architecture de pilotage distribuée hybride.

La distribution des décisions de pilotage conduit à des échanges de données à l'intérieur même du système. Il est important de les définir pour aboutir à une architecture de communication adéquate. Le travail de modélisation, complété par la simulation du système distribué, est censé nous apporter des éléments de réponse. Dans cette optique, nous avons proposé une méthode, différente de la simulation distribuée, qui permet de représenter, à partir de la simulation classique, un système distribué de pilotage comme un ensemble de sous-systèmes autonomes et de mettre alors en évidence les échanges entre ces sous-systèmes. Les échanges au sein du système sont matérialisés par un flux d'entités informationnelles de type ordre de traitement. D'autre part, le lien avec le système piloté, auquel est logiquement associé un flux d'entités physiques de type produit, est réalisable grâce à un mécanisme de synchronisation/désynchronisation des deux catégories d'entités (ordre de traitement et produit correspondant). En distinguant flux décisionnel et flux opérationnel, il est possible d'évaluer l'impact du système de pilotage sur l'unité et de visualiser le cheminement des entités à travers le système global.

Ce travail de modélisation nous a montré que les échanges dans un système distribué de pilotage peuvent se résumer, de manière simplifiée, comme suit: pour chaque réalisation d'opération engendrée par un ordre de traitement, le Contrôleur Central en fait la demande au Contrôleur de Ressources concerné qui prend localement sa décision de pilotage en fonction de la disponibilité des ses ressources et envoie une indication quand l'opération est terminée. A condition de considérer que les Contrôleurs de Ressources sont à l'origine des demandes, il s'avère que ces échanges se rapprochent des communications requête-réponse pratiquées dans le mode Client-Serveur. Après étude des architectures de communication, nous avons privilégié l'architecture Client-Serveur du Web en raison de sa capacité d'ouverture et des multiples standards libres de droits qu'elle fournit, lui offrant une réelle indépendance vis-à-vis des plates-formes d'exécution. Ce choix réduit considérablement les problèmes d'interopérabilité et les difficultés de développement. Un Contrôleur de Ressources devient alors un client qui doit récupérer un document liste d'opérations à exécuter sur le serveur Web Contrôleur Central et lui envoyer ses décisions. Techniquement, les communications sont assurées par le protocole HTTP sous-jacent à l'architecture Internet : les requêtes HTTP sont envoyées par les Contrôleurs de Ressources et traitées, au niveau du Contrôleur Central, par une servlet Java renvoyant une réponse en fonction du type de la requête. Nous avons adopté le même principe pour les communications entre l'unité et les Contrôleurs de Ressources.

Le développement nous a permis de mettre en place, au niveau des Contrôleurs de Ressources, les procédures interactives caractéristiques de notre système. Ces procédures dépendent bien évidemment du type de contrôleur. A partir de la liste récupérée sur le serveur, un Contrôleur de Ressources est capable, si nécessaire, de construire lui-même une liste restreinte d'opérations pour les opérateurs qu'ils gèrent, après une éventuelle recherche d'équipements libres. Au même titre que l'état d'un équipement, le choix d'un opérateur est immédiatement pris en compte par son contrôleur pour une mise à jour en temps réel du pilotage. Ce côté interactif, fortement appuyé par l'architecture choisie, permet d'impliquer pleinement les opérateurs dans la gestion de l'unité. Dans ce contexte, les interfaces proposées devront être soignées pour emporter l'adhésion des utilisateurs.

Reste que la nature interactive du système pose des problèmes de fiabilité. Bien que l'interactivité soit un des points forts du système, il apparaît qu'elle constitue aussi sa principale faiblesse, dans la mesure où les opérateurs pourraient bloquer le système en ne respectant pas leurs choix initiaux. Les résultats d'application nous ont largement démontré les dysfonctionnements dus aux problèmes de bonne foi ou de compétence des opérateurs (retards de production conséquents, bouleversements des priorités). Nous avons donc pensé à coupler un moteur de confiance au système, afin que celui-ci puisse juger de la confiance qu'il peut avoir en chacun des opérateurs et donc estimer sa propre fiabilité. La confiance est appréciée selon la performance passée d'un opérateur fondée sur des preuves. Il est possible, à partir des événements de l'unité, d'alimenter la base de preuves du moteur de confiance et de calculer ainsi la valeur de confiance d'un opérateur, d'un ensemble d'opérateurs voire même de l'unité. Mais il reste nécessaire d'implanter un composant décisionnel qui autoriserait la réalisation d'une opération ou préconiserait la mise en place d'actions correctives en fonction des valeurs calculées.

Au final, nous obtenons une structure décisionnelle de base, modulaire et ouverte, pour un pilotage interactif d'unités opérationnelles préexistantes. Etant donné son caractère générique, le système est facilement adaptable à un grand nombre d'unités, non nécessairement industrielles, à partir du moment où le besoin s'en fait ressentir (opérations manuelles, processus longs et coûteux,...). Nous donnons, à ce propos, un exemple d'utilisation dans une unité hospitalière. Toute implantation requiert cependant une étude approfondie de

l'infrastructure de communication présente ou non dans l'unité afin d'identifier les possibilités de transmissions de données au niveau des opérateurs et des équipements.

Cette architecture de base pourrait par la suite être complétée et utilisée pour prendre en charge des décisions de pilotage plus évoluées.

Nous avons étudié, de manière théorique, l'apport d'un tel système sur unité, mais nous n'avons pu l'expérimenter dans des conditions réelles. Les entreprises (RVI, hôpitaux), que nous avons démarchées pour obtenir des données en vue de l'application du système, restent assez réfractaires pour le moment. En ce qui concerne les unités industrielles, celles-ci font souvent usage de modèles complètement automatisés ou totalement manuels et sont peu enclines aux changements, d'autant plus que leurs organisations actuelles ne sont pas forcément adaptées (ex. : problème d'identification des produits dans le stock). Dans le cas des unités hospitalières, le problème est quelque peu différent. Elles manquent de moyens informatiques et il est difficile de faire comprendre aux membres du personnel la nécessité de saisir des données alors qu'ils sont en pleine action (il faut qu'ils perçoivent que la saisie de données supplémentaires peut leur apporter un plus dans leur travail quotidien). En outre, leur principal souci se situe à un autre niveau (ex. : comment remédier au retard d'un chirurgien qui constitue la clé de voûte du système ?).

Pourtant, l'homme est sans doute la ressource flexible par excellence et il est donc amené à prendre de plus en plus d'importance au sein des unités opérationnelles. Il faut apprendre à l'intégrer au même titre que les équipements. Dans cette optique, nous croyons sincèrement que les fonctionnalités du système constituent un moyen efficace pour pallier les problèmes liés à son comportement peu prévisible. Il nous reste à vérifier cette hypothèse dans une unité réelle et non pas sur la base d'un simple environnement de test qui ne peut reproduire à lui seul les comportements humains.

Références bibliographiques

- [1] GERTOSIO Christine, DUSSAUCHOY Alain. *Modeling of test workshop controller*. 8th IEEE International Conference on Emerging Technologies and Factory Automation, 15-18 October 2001, Antibes (France), vol. 2, pp. 57-62.
- [2] RODDE G., 1989. Les systèmes de production – modélisation et performance. Paris : éditions HERMES, 330pp.
- [3] SOFTWARE ENGINEERING INSTITUTE. *Middleware* [online]. 1997. Available at <<http://www.sei.cmu.edu/str/descriptions/middleware.html>> (2004).
- [4] TAYLOR Frederick Winslow. *The Principles of Scientific Management*. New York: Harper and Brothers, 1911, 72p.
- [5] HERZBERG Frederick, MAUSNER Bernard, & SNYDERMAN Barbara B. *The Motivation to Work*. New York: John Wiley, 1959, 157p.
- [6] MASLOW Abraham H. *A Theory of Human Motivation*. Psychological Review, 1943, vol. 50, pp. 370-396.
- [7] ROETHLISBERGER Fritz J., DICKSON William J. *Management and the Worker*. 1st ed. Cambridge: Harvard University Press, 1939, 615p.
- [8] LINDBECK Assar, SNOWER Denis J. *Multi-Task Learning and the Reorganization of Work: from Tayloristic to Holistic Organization*. Journal of Labor Economics, 2000, vol. 18, n° 3, pp.353-376.
- [9] OBORSKI Przemyslaw. *Social-technical aspects in modern manufacturing*. International Journal of Advanced Manufacturing Technology, 2003, vol. 22, n° 11-12, pp. 848-854.
- [10] KUHN Thomas S. *The Structure of Scientific Revolutions*. 1st ed. Chicago: University of Chicago Press, 1962, 168p.
- [11] LEE W.G.K, BAINES T., TIAHJONO B., et al. *Towards a conceptual framework of manufacturing paradigms*. SIMTech technical reports, 2006, vol. 7, n° 3, pp.160-177
- [12] DE TONI Alberto, TONCHIA Stefano. *New production models: a strategic view*. International Journal of Production Research, 2002, vol. 40, n° 18, pp. 4721-4741.
- [13] MASON-JONES Rachel, NAYLOR Ben, TOWILL Denis R. *Lean, agile or leagile? Matching your supply chain to the marketplace*. International Journal of Production Research, 2000, vol.38, n°17, pp. 4061-4070.
- [14] O'BRIEN Vanessa. *Should Manufacturing Pursue a Lean, Agile or Leagile Strategy?* 2005. Research report, M. App. Sc. In Operations and Quality Management. Galway: National University of Ireland, 2005, 12p.
- [15] DUGUAY Claude R., LANDRY Sylvain, PASIN Federico. *From mass production to flexible/agile production*. International Journal of Operations & Production Management, 1997, vol. 17, n° 12, pp. 1183-1195.

- [16] HOUNSHELL David A. *The Ford Motor Company and the Rise of Mass Production*. In: From the American System to Mass Production. London: Johns Hopkins University Press, 1984, pp. 217-262.
- [17] JULIEN Pierre-André, RAYMOND Louis, JACOB Réal et al. *L'entreprise-réseau*. Québec : Presses de l'Université du Québec, 2003, 534p.
- [18] ANDERSON David M. *The Shortcomings of Mass Production*. In: Build-to-order and Mass Customization. USA: CIM Press, 2004, pp. 75-87.
- [19] WOMACK Jones P., JONES Daniel T. *Lean Thinking: Banish Waste and Create Wealth in Your Corporation*. UK: Simon & Schuster, 1996, 352p.
- [20] WOMACK Jones P., JONES Daniel T., ROOS Daniel. *The Machine that Changed the World: The Story of Lean Production*. New York: Harper Perennial, 1991, 336p.
- [21] OHNO Taiichi. *The Toyota Production System: Beyond Large-Scale Production*. Portland: Productivity Press, 1988, 163p.
- [22] AYTAC Selim Erdem. Lean Manufacturing as a Human-Centred Approach for Manufacturing System Design. In: BRANDT Dietrich, ESTIEM (Eds). Reflections on Human-Centred Systems and Leadership - Summer Academy SAC, Eger (Hungary), 2003, pp. 1-20.
- [23] BOYER Robert, FREYSSINET Michel. *Rien n'est définitif. Tout modèle productif a des limites, 1986-1992*. Paris : GERPISA, 1999, 32p.
- [24] MESMER Philippe. *Toyota leader mondial : la victoire d'une méthode*. Le Monde, 2007.
- [25] KIDD Paul T. *Agile Manufacturing: Forging New Frontiers*. Boston: Addison-Wesley, 1994, 388p.
- [26] CHRISTOPHER Martin. *The Agile Supply Chain – Competing in Volatile Markets*. Industrial Marketing Management, 2000, vol. 29, pp. 37-44.
- [27] KALKUNTE M.V., SARIN S.C. WILHELM W.E. *Flexible Manufacturing Systems: A Review of Modeling Approaches for Design, Justification and Operation*. In: KUSIAK Andrew (Ed.). Flexible Manufacturing Systems: Methods and Studies. Amsterdam: Elsevier Science Publishers B.V. (North-Holland), 1986, pp.3-25.
- [28] MEHRABI M.G., ULSOY A.G., KOREN Y. *Reconfigurable manufacturing systems: Key to future manufacturing*. Journal of Intelligent Manufacturing, 2000, vol. 11, pp. 403-419.
- [29] WESTKAMPER E. *Adaptable Production Structures*. In: DASCHENKO Anatoli (Ed.). Manufacturing Technologies for Machines of the Future (21st Century Technologies). Berlin: Springer, 2003, pp. 87-120.
- [30] GOLDMAN S.L., NAGEL R.N., PREISS K., 1995. *Agile Competitors and Virtual Organisations: Strategies for Enriching the Customer*. New York: Van Nostrand Reinhold, 1995, 414p.
- [31] DOLGUI Alexandre, PROTH Jean-Marie. *Les systèmes de production modernes – Conception, gestion et optimisation*. Paris : Lavoisier, 2006, 415p.
- [32] KIDD Paul T. *Agile Manufacturing: Key Issues*. In: KIDD P.T & KARWOWSKI W. (Ed.). Advances in Agile Manufacturing: Integrating Technology, Organization and People. Amsterdam: IOS Press, 1994, pp. 30-31.

- [33] WARNECKE Hans-Jürgen. *Fractal Company – A Revolution in Corporate Culture*. In: DASCHENKO Anatoli (Ed.). *Manufacturing Technologies for Machines of the Future (21st Century Technologies)*. Berlin: Springer-Verlag Berlin Heidelberg, 2003, pp. 63-85.
- [34] KUSIAK Andrew. *Computational Intelligence in Design and Manufacturing*. New York: Wiley-Interscience, 2000, 535p.
- [35] MONFARED Mohammad Ali Saniee, YANG Jian-Bo. *Design of integrated manufacturing planning, scheduling and control systems: a new framework for automation*. *International Journal of Advanced Manufacturing Technologies*, 2007, vol. 33, pp. 545-559.
- [36] ACTORS SOLUTIONS. *Manufacturing Execution System* [online]. Available at <<http://www.actors-solutions.com/Manufacturing-Execution-System-MES>> (2005).
- [37] VINALS José. *L'utilisation des technologies de pointe dans le nouveau contexte de la production manufacturière*. Québec : Conseil de la science et de la technologie, 2006, 93p.
- [38] MITAL A., PENNATHUR A., HUSTON R.L., et al. *The need for worker training in advanced manufacturing technologies (AMT) environments: A white paper*. *International Journal of Industrial Ergonomics*, 1999, vol. 24, pp. 173-184.
- [39] MILLOT Patrick. *Systèmes Homme-Machine et Automatique*. Journées Doctorales d'Automatique, 21-23 septembre 1999, Nancy (France), pp. 1-24.
- [40] SUN Hongyi, FRICK Jan. *A shift from CIM to CHIM*. *International Journal of Computer Integrated Manufacturing*, 1999, vol. 12, n° 5, pp. 461-469.
- [41] WILSON John R. *Support of opportunities for shopfloor involvement through information and communication technologies*. London: Springer-Verlag, 2003, vol. 17, pp. 114-133.
- [42] VICENTE Kim J. *Ecological Interface Design: Progress and challenges*. *Human Factors: The Journal of the Human Factors and Ergonomics Society*, 2002, vol. 44, n°1, 62-78.
- [43] VENKATASUBRAMANIAN V., RENGASWAMY R., YIN K., et al. *A review of process fault detection and diagnosis*. *Computers and Chemical Engineering*, 2003, vol.27, n° 3, pp. 293-346.
- [44] HOC Jean-Michel. *From human-machine interaction to human-machine cooperation*. *Ergonomics*, 2000, vol. 43, n°7, pp. 833-843.
- [45] BENNOUR M., CRESTANI D., CRESPO O., et al. Evaluation d'une approche d'affectation des ressources humaines aux processus d'entreprise. *5^{ème} Conférence Francophone de Modélisation et SIMulation*, 1-3 septembre 2004, Nantes (France).
- [46] HOC Jean-Michel, MEBARKI Nasser, CEGARRA Julien. *L'assistance à l'opérateur humain pour l'ordonnancement dans les ateliers manufacturiers*. *Le travail humain*, 2004, vol. 67, pp. 181-208.
- [47] BLAZEWICZ J., ECKER K.H., PESCH E., et al. *Computer Integrated Production Scheduling*. In: *Scheduling Computer and Manufacturing Processes*. Berlin: Springer, 1996, pp. 423-471.
- [48] SHIN D., WYSK R.A., ROTHROCK L. *A formal control-theoretic model of a human-automation interactive manufacturing system control*. *International Journal of Production Research*, 2006, vol. 44, n° 20, pp.4273-4295.

- [49] BILLAUT Jean-Charles, ROUBELLAT François. *A new method for workshop real time scheduling*. International Journal of Production Research, 1996, Vol.34, N°6, pp.1555-1579.
- [50] MAC CARTHY M. Can car manufacturing techniques reform health care? *Lancet*, 2006, vol. 367, p.290.
- [51] TRENTÉSEAUX Damien. *Conception d'un système de pilotage distribué, supervisé et multicritère pour les systèmes automatisés de production*. Thèse en Automatique/Productique. Grenoble : INPG, 1996, 213p.
- [52] ESQUIROL P., LOPEZ P. *L'ordonnancement*. Paris : Economica, 1999, 141p.
- [53] MIRDAMADI Samieh. *Pilotage d'un atelier de production en temps réel à base de simulation de flux à événements discrets*. 8ème Congrès des Doctorants EDSys, 10 mai 2007, Albi (France), 6p.
- [54] DAVENPORT A.J., BECK J.C. *A survey of techniques for scheduling with uncertainty* [online]. 2000. Available at <<http://www.eil.utoronto.ca/profiles/chris/chris.papers.html>> (2005).
- [55] LA Hoang Trung. *Utilisation d'ordres partiels pour la caractérisation de solutions robustes en ordonnancement*. Thèse en Systèmes Industriels. Toulouse : INSA de Toulouse, 2005, 200p.
- [56] BLAZEWICZ J., ECKER K.H., PESCH E., et al. *Scheduling Computer and Manufacturing Processes*. Berlin: Springer, 1996, 491p.
- [57] WIDMER Marino. *Les métaheuristiques : des outils performants pour les problèmes industriels*. 3^{ème} Conférence Francophone de Modélisation et SIMulation, 25-27 avril 2001, Troyes (France).
- [58] MEBARKI N. *Une approche d'ordonnancement temps-réel basée sur la sélection dynamique de règles de priorité*. Thèse en Ingénierie Informatique. Lyon : Université Claude Bernard Lyon 1, 1995.
- [59] BESLON Guillaume. *Contrôle sensori-moteur par réseaux neuromimétiques modulaires – Approche pour le pilotage réactif en atelier flexible*. Thèse en Ingénierie Informatique. Lyon : INSA de Lyon, 1995, 333p.
- [60] SINGH N., RAJAMANI D. *Cellular manufacturing systems – Design, Planning and Control*. London: Chapman & Hall, 1996, pp. 212-273.
- [61] SMITH J.S., JOSHI S.B. *Reusable Software concepts applied to the developments of FMS Control software*. International Journal of Computer Integrated Manufacturing, 1992, vol. 5, pp. 182-196.
- [62] LEITAO Paulo, RESTIVO Francisco. *A Layered Approach to Distributed Manufacturing*. Proceedings of 1999 Advanced Summer Institute - LIFE Cycle Approaches to Production Systems, 21-23 September 1999, Leuven (Belgium).
- [63] DILTIS D., BOYD N., WHORMS H. *The evolution of control architectures for automated manufacturing systems*. Journal of Manufacturing Systems, 1991, vol. 10, n°1, pp.63-79.
- [64] INSTITUTE FOR MANUFACTURING – UNIVERSITY OF CAMBRIDGE. *Holonic Manufacturing Systems* [online]. Available at <http://www.ifm.eng.cam.ac.uk/automation/research/distributed_intelligent.html> (15/04/2005).

- [65] KALLEL, G., PELLET, X., BINDER, Z. Conduite Décentralisée Coordonnée d'Atelier. *Revue d'Automatique, de Productique et d'Informatique Industrielle*, 1985, vol. 19, n°4, pp. 371-387.
- [66] BRIAND Cyril, ROUBELLAT François. *Pilotage réactif et production à la commande*. 3^{ème} Conférence Francophone de Modélisation et SIMulation, Troyes (France), 25-27 avril 2001, vol. 1, pp.371-377.
- [67] SAGHED-TEHRANI M. *Information System's Roles and Responsibilities: Towards a Conceptual Model*. ACM Computer and Society, 2003, vol. 32, issue 3.
- [68] SIMON Herbert Alexander. *Le nouveau management - La décision par les ordinateurs*. 3^{ème} édition. Paris : Economica, 1980, 159p.
- [69] SIMON Herbert Alexander. *Models of Man : Social and Rational*. New York: John Wiley, 1957, 287p.
- [70] LEBRATY Jean-Fabrice, PASTORELLI Ivan. *Systèmes d'aide à la décision et perception de l'action*. 8^{ème} Colloque de l'AIM (Association Information et du Management), 21-23 mai 2003, Grenoble (France).
- [71] ANSOFF I. *Stratégie du développement de l'entreprise*. Edition française révisée. Paris : Les Editions d'Organisation, 1989, 287p.
- [72] GERTOSIO Christine, DUSSAUCHOY Alain. *A distributed Decision Support System for operational units. Application to a manufacturing unit*. Proceedings of the 9th IEEE International Conference on Emerging Technologies and Factory Automation, 16-19 September 2003, Lisbon (Portugal), vol. 2 pp. 575-581.
- [73] DRAGHICI George, BRINZEI Nicolae, FILIPAS Ioana. *La modélisation et la simulation en vue de la conduite des systèmes de production*. Les Cahiers des Enseignements Francophones en Roumanie, 1998, pp.110-130.
- [74] FERBER Jacques. *Les systèmes multi-agents. Vers une intelligence collective*, Paris : InterEditions, 1995, 522p.
- [75] LOPEZ N., MIGUEIS J., PICHON E. *Intégrer UML dans vos projets*. Paris : Eyrolles, 2000, 243 p.
- [76] JACOBSON I., BOOCH G., RUMBAUCH J. *The unified Software development Process*. Reading (Massachusetts): Addison-Wesley, 1999, 463p.
- [77] JACOBSON I., CHRISTERSON M., JONSSON P., et al. *Object Oriented Software Engineering: A Use Case Driven Approach*. Reading (Massachusetts): Addison-Wesley, 1992, 552p.
- [78] FABIAN M., LENNARTSON B. *Control of manufacturing systems; an object oriented approach*. Proceedings of IFAC - INCOM, 1992, Toronto (Canada).
- [79] GERTOSIO Christine, COMBES Catherine, DUSSAUCHOY Alain. *Towards a methodology to simulate an organization from a UML modelling - An operational efficiency for chronic renal dialysis unit*. International Conference on Industrial Engineering and Production Management, 26-28 May 2003, Porto (Portugal), vol. 2, p.218-226.
- [80] DAVID R., ALLA H. *Du Grafset aux réseaux de Pétri*. 2^{ème} Edition. Paris: Hermes, 1992.

- [81] ELMAGHRABY Salah E. *Activity Networks - Project Planning and Control by Network Models*. New York: John Wiley & Sons, 1977, 443p.
- [82] ALAN A., PRITSKER B. *Modeling and Analysis Using Q-GERT Networks*. New York: John Wiley & Sons, 1979, 456p.
- [83] ZEIGLER Bernard P. *Theory of Modelling and Simulation*. Malabar (Florida): Krieger Publishing Company, 1984, 435p.
- [84] FISHMAN George S. *Discrete-Event Simulation - Modeling, Programming and Analysis*. New York: Springer-Verlag, 2001, 537p.
- [85] ZEIGLER B.P., SARJOUGHIAN H.S. *Creating distributed simulation using DEVS M&S environments*. Proceedings of the Winter Simulation Conference, 10-13 December 2000, Orlando (Florida), vol. 1, pp. 158-160.
- [86] ALLIOT J.M. *Qu'est-ce que le Middleware?* [online]. 2003. Disponible sur <<http://www.recherche.enac.fr/~alliot/middle.pdf>> (2006).
- [87] SARJOUGHIAN H.S, CELIER F.E. *Discrete event modeling and simulation technologies: a tapestry of systems and AI-based theories and methodologies*. New-York: Springer-Verlag, 2001, 397p.
- [88] GONZALEZ F.G. *Simulation of Distributed system* [online]. 2003. Available at <<http://pegasus.cc.ucf.edu/~fgonzale/papers/DASD03.pdf>> (2006).
- [89] ROCKWELL SOFTWARE INC. *Arena 8* [online]. Available at <<http://www.arenasimulation.com>> (2004).
- [90] GERTOSIO Christine, GRANDGIRARD Emilie. *Modelling of distributed systems in one single simulation model: a way to study communications within distributed systems*. 10th IEEE International Conference on Emerging Technologies and Factory Automation 2005, 19-22 September 2005, Catania (Sicily), vol.2, pp.697-703.
- [91] GERTOSIO Christine, GRANDGIRARD Emilie. *A simulation framework for distributed systems - an approach to the study of data exchange in distributed systems*. In Recent advances in Control Systems, Robotics and Automation, International Society for Advanced Research, July 2006, pp.81-86.
- GERTOSIO Christine, GRANDGIRARD Emilie. *A simulation framework for distributed systems - an approach to the study of data exchange in distributed systems*. International Journal of Factory Automation, Robotics and Soft Computing, International Society for Advanced Research, July 2006, pp.36-41.
- [92] KELTON W.D., SADOWSKI R.P., SADOWSKI D.A. *Simulation with ARENA*. 2nd Edition. New York: McGraw-Hill, 2001, 631p.
- [93] ZOLEZZI P. *Etude et mise en place d'un nouvel atelier d'essais moteurs après modélisations utilisant des outils statistiques, de simulation et d'aide à l'innovation*. Thèse en Informatique. Lyon : Université Claude Bernard de Lyon 1, 2000.
- [94] BELLISSARD Luc. *Construction et Configuration d'Applications Réparties*. Thèse en Informatique, Systèmes et Communication. Grenoble : INPG, 1997, 75p.
- [95] TANENBAUM A.S., VAN STEEN M. *Distributed systems – Principles and Paradigms*. New Jersey: Prentice-Hall, 2002, 803p.
- [96] MIRANDA Serge, RUOLS Anne. *Client-Serveur*. Paris: Eyrolles, 1994, 234p.

- [97] CAROMEL Denis, HENRIO Ludovic. *A Theory of Distributed Objects: Asynchrony – Mobility – Groups – Components*. New York: Springer, 2005, 346p.
- [98] BERNSTEIN Philip A. *Middleware: A Model for Distributed Services*. Communications of the ACM, 1996, vol. 39, n°2, pp.86-97.
- [99] MICROSOFT. *Utilisation de la procédure RPC asynchrone avec les applications client/serveur* [en ligne]. Disponible sur : <<http://msdn.microsoft.com/library/fre/default.asp?url=/library/FRE/dntaloc/html/asynrpc.asp>> (2004).
- [100] ASI. *CorbaFAQ* [en ligne]. Disponible sur : <<http://asi.insa-rouen.fr/projets/FAQ/CorbaFAQ/11-14.html>> (2004).
- [101] FRANCE TELECOM. *Mémento technique n°14 : Bases de données et système d'information* [en ligne]. Disponible sur : <<http://www.rd.francetelecom.com/fr/conseil/mento14/chap4h.html>> (2004).
- [102] BAKKEN David E. *Middleware*. In: URBAN J., DASGUPTA P. (Eds). *Encyclopedia of Distributed Computing*. Boston: Kluwer Academic Publishers, 2003.
- [103] SUN MICROSYSTEMS. *Java Message Service API Tutorial - Messaging for the J2EE Platform*. Massachusetts: Addison-Wesley, 2002, 544p.
- [104] GERTOSIO Christine, GRANDGIRARD Emilie, 2006. *Communication architecture of an interactive assistance system for semi-automated manufacturing units*. In: LALIC Bojan (Ed.). *Advanced Technologies: Research - Development – Application*. Vienna (Austria): Advanced Robotic Systems International, 2006, pp.333-348.
- [105] HUANG G.Q., MAK K.L. *Internet Applications in Product Design and Manufacturing*. Berlin: Springer-Verlag, 2003, 272p.
- [106] CLOUX P.Y., DOUSSOT D., GERON A. *Technologies et architectures Internet - Corba, COM, XML, J2EE, .NET, Web services*. Paris : Dunod, 2002, 253p.
- [107] THE INTERNET SOCIETY. *Hypertext Transfer Protocol -- HTTP/1.1, RFC 2616* [online]. 1999. Available at <<http://www.w3.org/Protocols/rfc2616/rfc2616.html>> (2005).
- [108] MARSHALL J. *HTTP Made Really Easy* [online]. 2004. Available at <<http://www.jmarshall.com/easy/http/>> (2005).
- [109] HAROLD E.R. *Java Network Programming*. Sebastopol (California): O'Reilly & Associates, 2004, 880p.
- [110] CHARON Irène. *Le langage Java : concepts et pratique*. 2^{ème} édition. Paris : Lavoisier, 2003, 413 p.
- [111] SUN MICROSYSTEMS. *J2EE Java Servlet Technology* [online]. Available at <<http://java.sun.com/products/servlet/>> (2006).
- [112] THE APACHE SOFTWARE FOUNDATION. *Apache Tomcat* [online]. Available at <<http://tomcat.apache.org/>> (2006).
- [113] SUN MICROSYSTEMS. *Java SE – Java DB and Java Database Connectivity (JDBC)* [online]. Available at <<http://java.sun.com/javase/technologies/database/>> (2006).
- [114] BERNADAC Jean-Christophe, KNAB François. *Construire une application XML*. Paris : Eyrolles, 1999, 476p.

- [115] COMBES Catherine, DUSSAUCHOY Alain, CHAABANE Sondes, et al. *Démarche méthodologique d'analyse des données pour la planification des blocs opératoires: une application à un service d'endoscopie*. GISEH, 9-11 septembre 2004, Mons (Belgique).
- [116] GRANDGIRARD Emilie, GERTOSIO Christine, SEIGNEUR Jean-Marc. *Trust Engines to Optimize Semi-automated Industrial Production Planning*. 16th IEEE International Symposium on Industrial Electronics, 4-7 June 2007, Vigo (Spain).
- [117] MARSH S. *Formalising Trust as a Computational Concept*. PhD Thesis in Computing Science and Mathematics. Stirling (Scotland): University of Stirling, 1994, 184p.
- [118] SEIGNEUR Jean-Marc. *Trust, Security and Privacy in Global Computing*. PhD Thesis in Computer Science. Dublin: Trinity College, 2005, 187p.
- [119] SECURE. *Secure Environments for Collaboration among Ubiquitous Roaming Entities* [online]. Available at <<http://secure.dsg.cs.tcd.ie>> (2007).
- [120] GRANDGIRARD Emilie, GERTOSIO Christine, SEIGNEUR Jean-Marc. *Trust Engines to Preserve the Quality of an Operational Decision System*. In: PENNACHIO Salvatore (Ed). *Emerging Technologies, Robotics and Control Systems*. Palermo (Italy): International Society for Advanced Research, 2007, vol. 1, pp.98-104.
- GRANDGIRARD Emilie, GERTOSIO Christine, SEIGNEUR Jean-Marc. *Trust Engines to Preserve the Quality of an Operational Decision System*. *International Journal of Factory Automation, Robotics and Soft Computing*, 2007, Issue 3, pp.62-68.

Autres références (utilisées mais non citées dans le texte) :

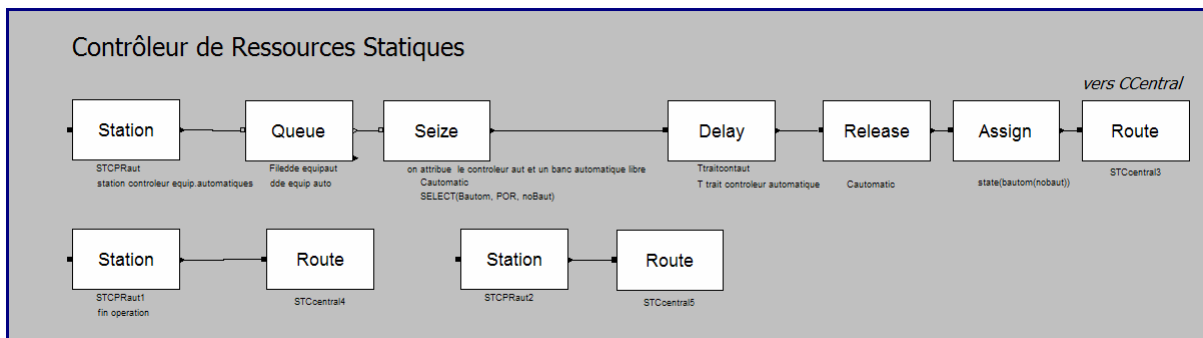
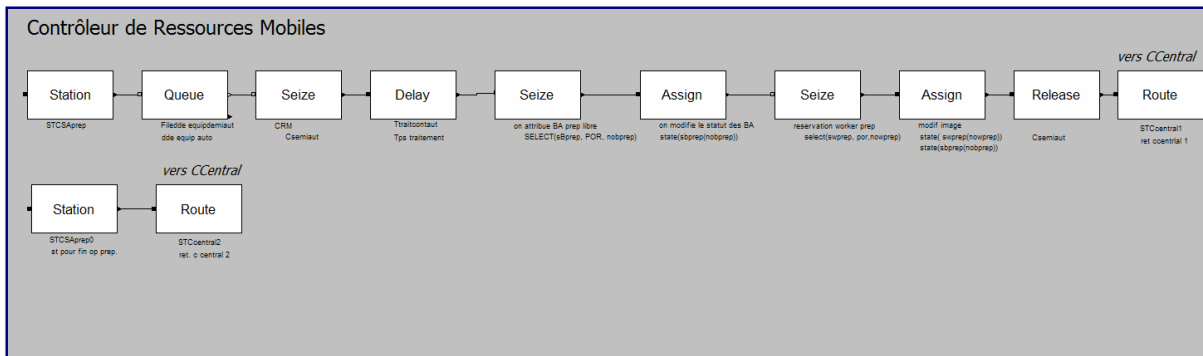
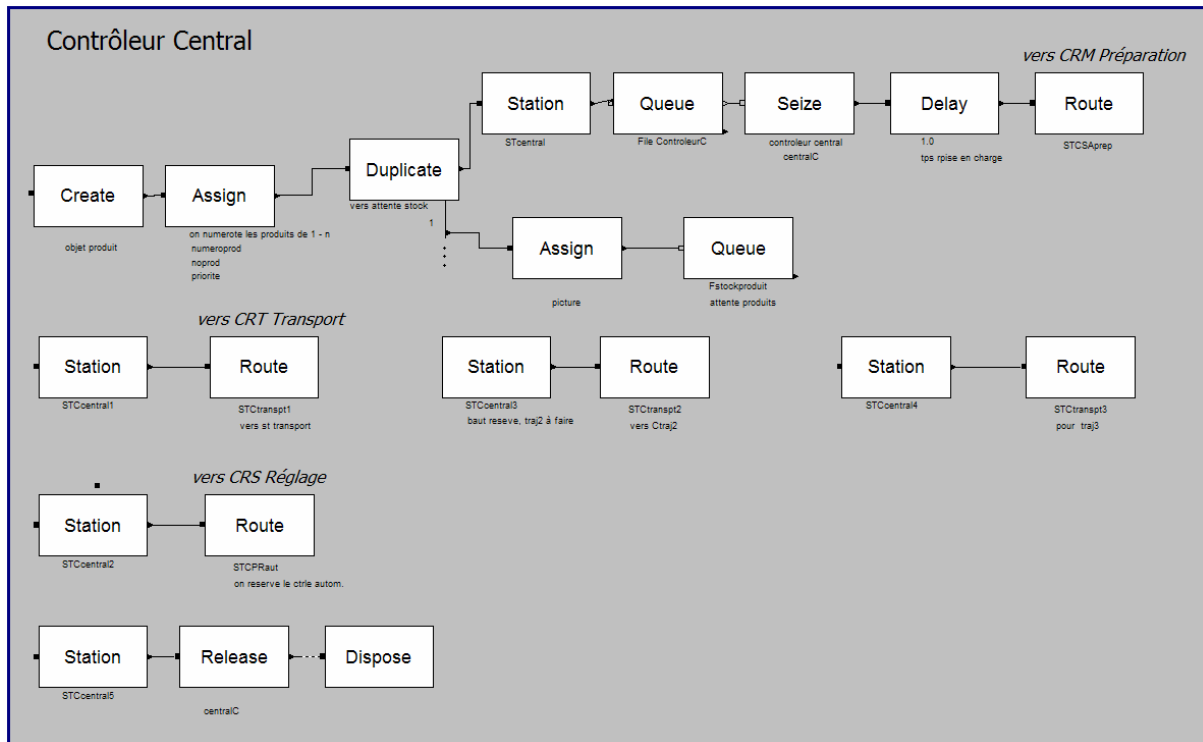
ECHALIER Florence. *Gestion de Production*. Support de cours Maîtrise MIAG. IUP MIAG de Lyon, 2002, 77p.

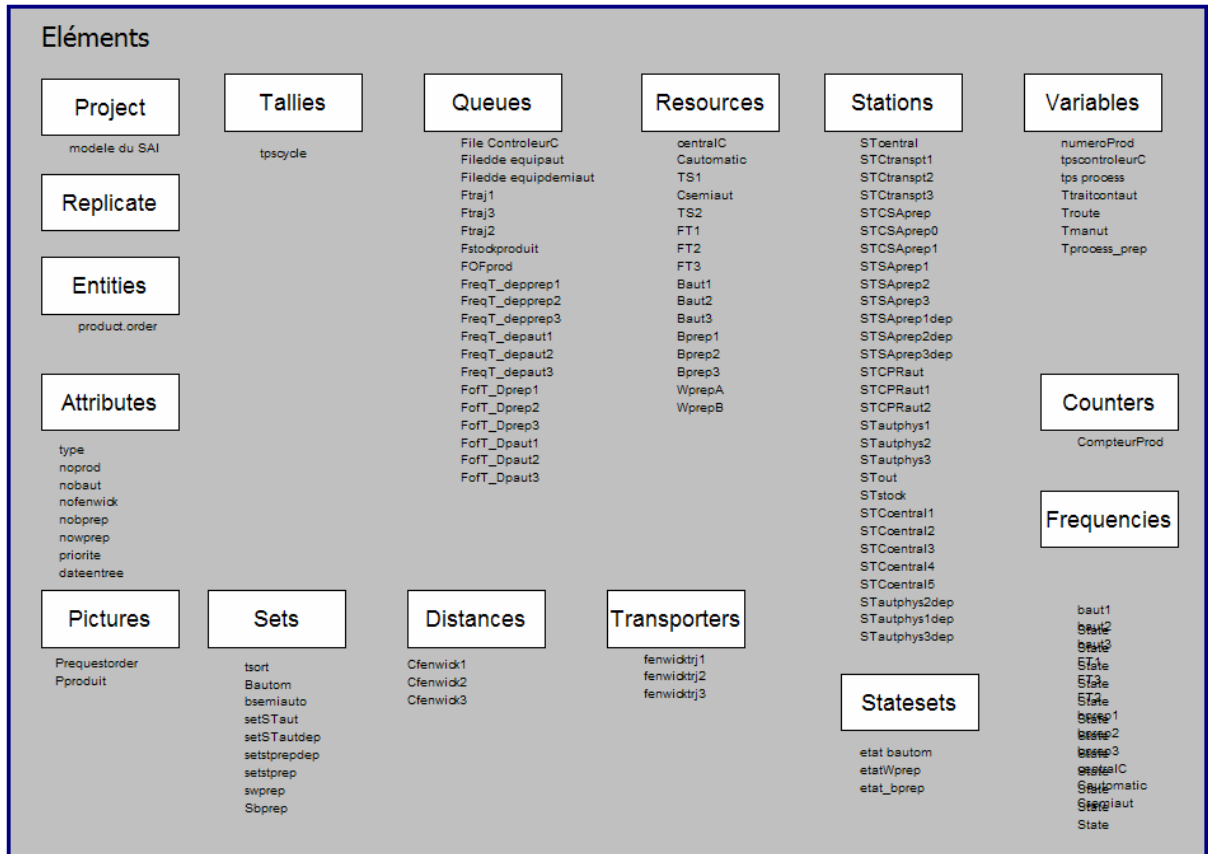
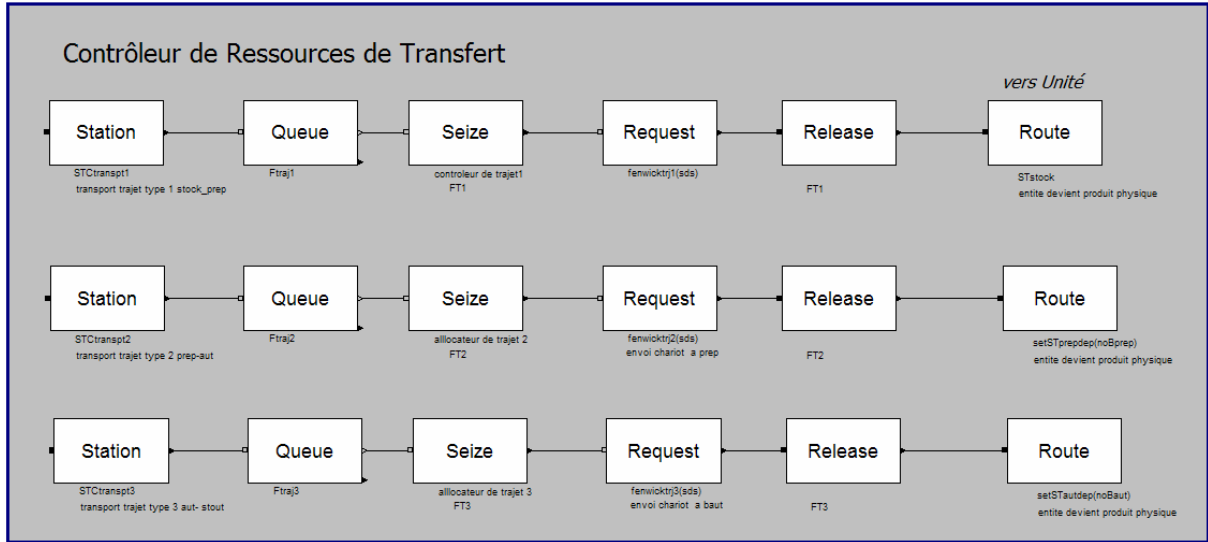
SOENEN René. *Systèmes à événements discrets*. Support de cours DEA ISCE. UFR Informatique de l'Université Claude Bernard de Lyon, 2003.

GERTOSIO Christine. *Modélisation des processus et simulation*. Support de cours M2 MIAG. UFR Informatique de l'Université Claude Bernard de Lyon, 2006, 38p.

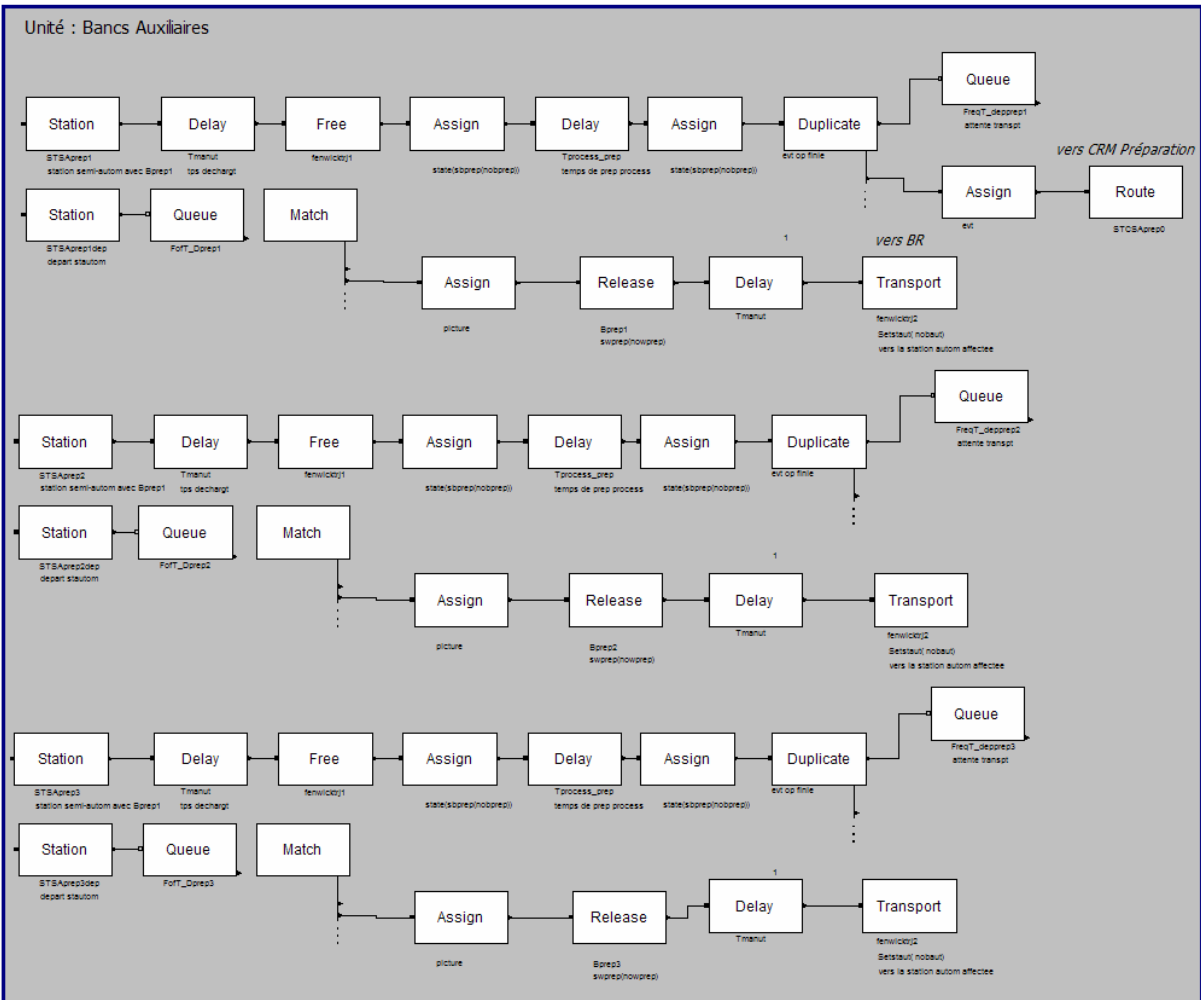
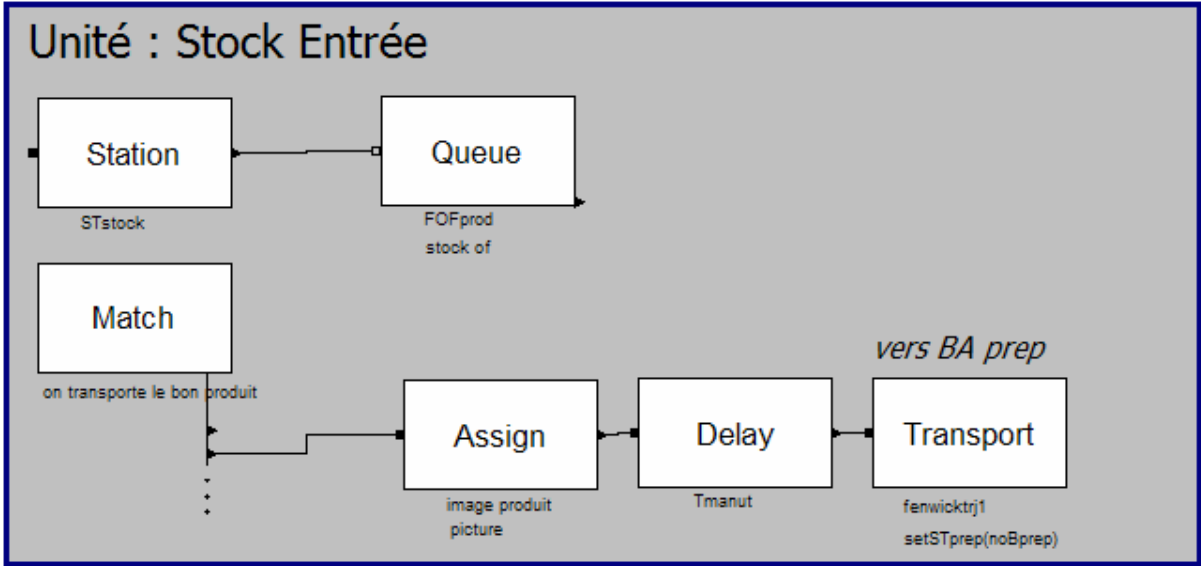
Annexe 1. Le modèle de simulation

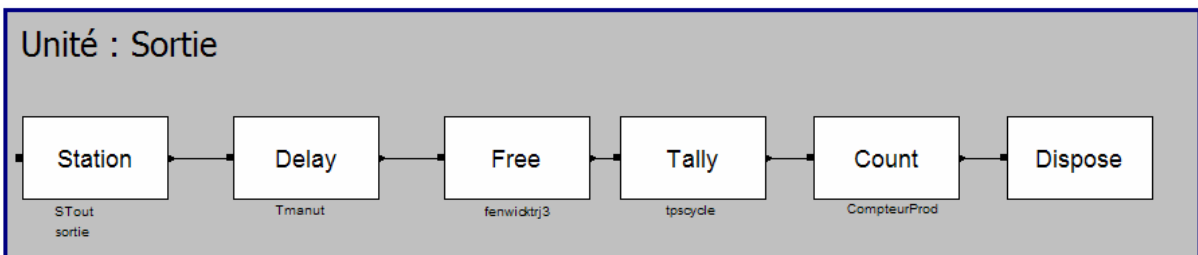
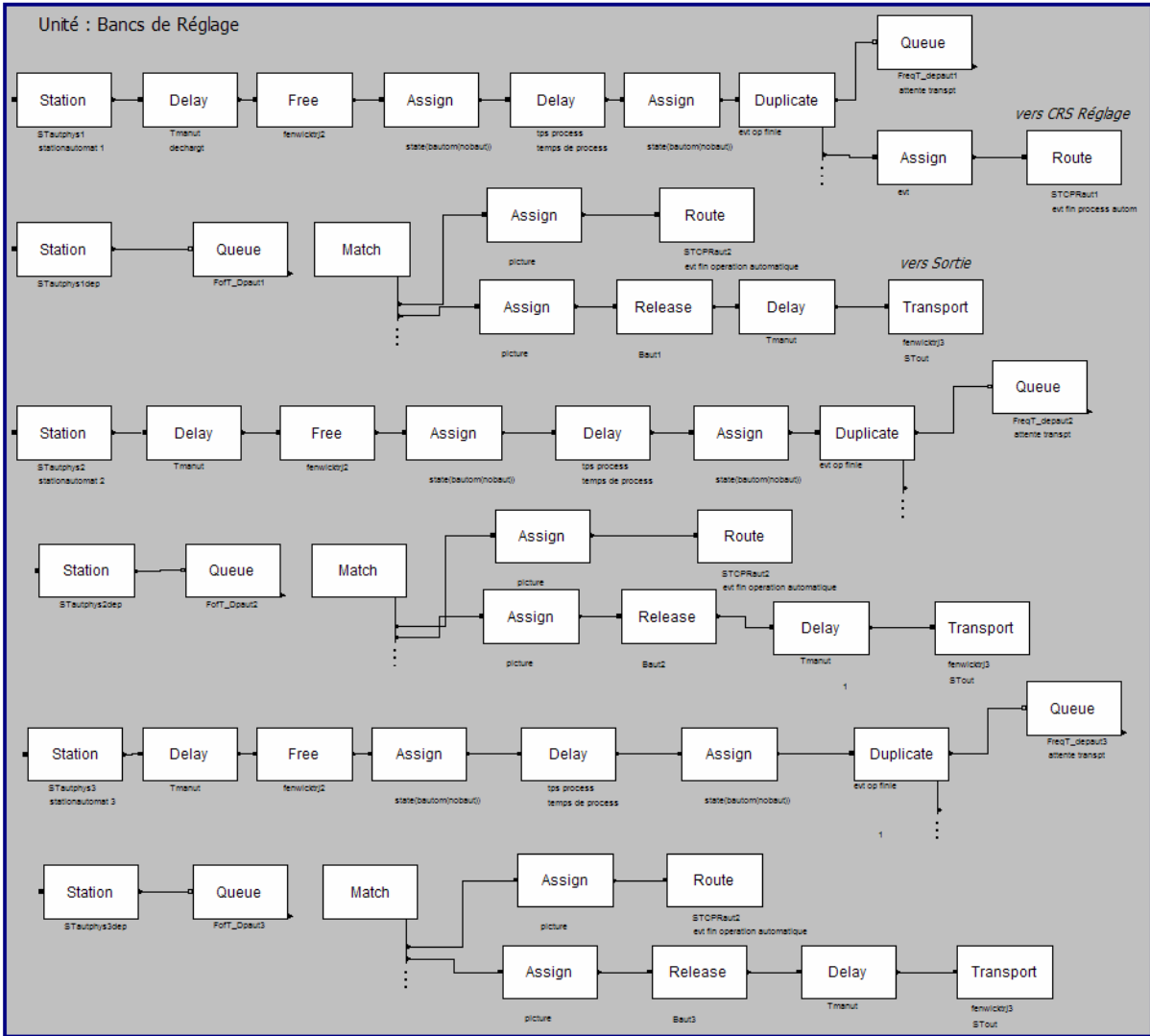
Sous-modèle du SAI





Sous-modèle de l'unité





Annexe 2. Scripts

Dans cette annexe est présenté le code de l'application relative au système. L'application est répartie entre le Contrôleur Central et les différents Contrôleurs de Ressources.

La partie résidant sur le serveur Contrôleur Central est développée sous la forme d'un projet Java Tomcat contenant les éléments suivants :

- Servlet* : récupère et traite les requêtes HTTP des clients Contrôleurs de Ressources.
- Classe ProduitJDBC* : récupère les ordres de traitement situés dans une BD, et en extrait les produits à traiter et les opérations associées à dispatcher selon les Contrôleurs de Ressources.
- Classe ListeTachesXML* : construit une liste d'opérations au format XML.
- Classe Produit : un objet de cette classe correspond à un produit à traiter.
- Classe Gamme : un objet de cette classe correspond à la liste des opérations nécessaires au traitement d'un produit.
- Classe Tache : un objet de cette classe correspond à une opération à réaliser. Un objet de la TacheM, héritant de la classe Tache, correspond à une opération de transport.
- Classe Confiance : alimente une base de preuves et calcule les valeurs de confiance.

Les parties résidant sur les Contrôleurs de Ressources sont aussi développées sous la forme de projets Java Tomcat. En effet, un Contrôleur de Ressources est certes client du Contrôleur Central mais aussi serveur de l'unité. Il contient les éléments suivants :

- Classe clientHTTP* : envoie des requêtes HTTP au serveur et traite les réponses.
- Classe AnalyseListeXML* : analyse la liste d'opérations reçue et procède à l'allocation de ressources. L'allocation diffère selon le type de ressource mise en jeu.
- Classe ListeTachesRessXML : crée la liste restreinte d'opérations pour les opérateurs.
- Classe RessourceJDBC : gère les équipements de l'unité à travers une BD.
- Servlet* : reçoit les événements de l'unité (choix opérateurs et états) et réagit en conséquence. Elle constate également les problèmes de confiance. Ces traitements diffèrent selon le type du Contrôleur de Ressources (CRS, CRM, CRT).

Seules les classes suivies d'un astérisque (*) sont présentées ici, les autres classes mettant en oeuvre des traitements classiques. Elles marquent les points centraux du système.

Les composants du Contrôleur Central

Servlet

```
import java.io.IOException;
import java.io.PrintWriter;
import java.sql.SQLException;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

public class CC extends HttpServlet {

    private static final long serialVersionUID = 1L;

    ProduitJDBC ListeProd;
    Confiance Conf;

    //RECEPTION COMMANDE GET
    protected void doGet(HttpServletRequest req, HttpServletResponse res) throws ServletException, IOException {
        //DEMANDE DE LISTE PAR CONTROLEUR
        String idC=req.getParameter("idControleur");
        if(idC!=null) { //cas demande liste taches
            res.setContentType("text/xml");
            PrintWriter out = res.getWriter();

            try {
                ListeProd.reconstruireListes();
            } catch (SQLException e) {
                e.printStackTrace();
            } catch (ClassNotFoundException e) {
                e.printStackTrace();
            }
            (ListeProd.recupereTaches(idC)).fluxSortie(out);
        }
    }

    //RECEPTION COMMANDE POST
    protected void doPost(HttpServletRequest req, HttpServletResponse res) throws ServletException, IOException {
        String idTache=req.getParameter("idTacheChoisie");
        if(idTache!=null) { //CHOIX FAIT PAR UN OPERATEUR
            //1. retrouver le produit correspondant à la tache
            Produit P=ListeProd.chercheProduitTache(idTache);
            //2. maj liste correspondant à la tache : suppression de la tache
            int i=P.getAvancement();
        }
    }
}
```

```

ListeTachesXML L;
String C;
if (((P.getGamme()).chercheTache(idTache)).getType().equals("M")) {          //CAS TRANSPORTEUR
    L=ListeProd.recupereListeTachesTransport(i);
    C=ListeProd.recupereControleurTransport(i);
}
else { //CAS OPERATEUR
    L=ListeProd.recupereListeTaches(i);
    C=ListeProd.recupereControleur(i);
}

if (L.supprimeNoeud(idTache)) { //TACHE TROUVEE
    //MAJ TACHE AVEC RESSOURCE CHOISIE
    String ressource=req.getParameter("ressource");
    if(ressource!=null) {
        if (((P.getGamme()).chercheTache(idTache)).getType().equals("SA")) {
            P.getGamme().getTache(i).setHumain(ressource);
        }
        if (((P.getGamme()).chercheTache(idTache)).getType().equals("M")) {
            P.getGamme().getTacheTransport(i).setHumain(ressource);
        }
    }

    //CAS DERNIERE TACHE : ARRETER L'UNITE
    if(ListeProd.verifFin()==true) {

        try {
            Conf.calculConfianceOperateur();
        } catch (SQLException e) {
            e.printStackTrace();
        } catch (ClassNotFoundException e) {
            e.printStackTrace();
        }

        L.finListe();
    }
    L.versFichierXML("C:"+ "\\SERVEUR\\"+C);
    //3. renvoi liste en reponse
    res.setContentType("text/xml");
    PrintWriter out = res.getWriter();
    L.fluxSortie(out);
    Tache T=P.getGamme().chercheTache(idTache);
    //CAS TACHE SA (CRM): ALLOCATION COMPLETE --> CREATION TRANSPORT CORRESPONDANT
    if (T.getType().equals("SA")) {
        if (T.getDebut()==false) { //ne pas recreeer le transport si le produit a déjà été déposé (pb operateur)
            if (i==0) { //CREER TACHE DE TRANSPORT DE DEPART
                // -> tache de transport du stock vers les machines Tache1
                Tache TacheActu=(Tache)((P.getGamme()).getListeGamme()).get(i);
                TacheM
                Tr=(P.getGamme()).creeTacheTransportDepart("Tr_Stock_"+TacheActu.getIdTache(),TacheActu);
            }
        }
    }
}

```

```

        ListeTachesXML LT=ListeProd.recupereListeTachesTransport(i);
LT.ajouteNoeudTransportTrie(Tr.getIdTache(),P.getNomProduit(),P.getPriorite(),Tr.getOrigine(),Tr.getDestination());
        LT.versFichierXML("C:"+"\SERVEUR\\"+ListeProd.recupereControleurTransport(i));
    }
    else {
        if (i<(P.getGamme()).getnombreTaches()) { //CREER TACHE DE TRANSPORT INTERMEDIAIRE
            Tache TacheActu=(Tache)((P.getGamme()).getListeGamme()).get(i-1);
            Tache TacheSuiv=(Tache)((P.getGamme()).getListeGamme()).get(i);
            TacheM
Tr=(P.getGamme()).creeTacheTransport("Tr_"+TacheActu.getIdTache()+"_"+TacheSuiv.getIdTache(),TacheActu,TacheSuiv,i);
            ListeTachesXML LT=ListeProd.recupereListeTachesTransport(i);

LT.ajouteNoeudTransportTrie(Tr.getIdTache(),P.getNomProduit(),P.getPriorite(),Tr.getOrigine(),Tr.getDestination());
            LT.versFichierXML("C:"+"\SERVEUR\\"+ListeProd.recupereControleurTransport(i));
        }
    }
}
}
}
else { //CAS TRANSPORT : SUPPRESSION DE LA TACHE DESTINATAIRE POUR EMPECHER UNE NOUVELLE REAFFECTATION
TacheM Tr=(TacheM)(P.getGamme()).chercheTache(idTache);
Tache TD=(Tr.getTacheDestinataire());
if (TD!=null) {
    String idT=TD.getIdTache();
    ListeTachesXML LTD=ListeProd.recupereListeTaches(i);
    LTD.supprimeNoeud(idT);
    LTD.versFichierXML("C:"+"\SERVEUR\\"+ListeProd.recupereControleur(i));
}
}
}
else { //TACHE DEJA CHOISIE -> RENVOI ERREUR
res.setContentType("text");
PrintWriter out = res.getWriter();
out.println("Erreur");
}
}
else {
String idT=req.getParameter("idtache");
String ress=req.getParameter("ressource");
String idP=req.getParameter("produit");
if(idT!=null && ress!=null) { //AFFECTATION DES RESSOURCES MACHINES-> MAJ PRODUIT ET CREATION TRANSPORTS
//changement objet produit avec la ressource affectée à la tache
Produit P=ListeProd.chercheProduit(idP);
P.affecteRessourceTache(idT,ress);
//mise a jour listes transports
Tache T=(P.getGamme()).chercheTache(idT);
if ((T.getType()).equals("A")) { //cas tache A (CRS)
int i=P.getAvancement();
if (i==0) { //CREER TACHE DE TRANSPORT DE DEPART
// -> tache de transport du stock vers les machines Tache1
Tache TacheActu=(Tache)((P.getGamme()).getListeGamme()).get(P.getAvancement());

```

```

        TacheM Tr=(P.getGamme()).creeTacheTransportDepart("Tr_Stock_"+TacheActu.getIdTache(),TacheActu);
        ListeTachesXML L=ListeProd.recupereListeTachesTransport(i);
        L.ajouteNoeudTransportTrie(Tr.getIdTache(),P.getNomProduit(),P.getPriorite(),Tr.getOrigine(),Tr.getDestination());
        L.versFichierXML("C:"+ "\\SERVEUR\\"+ListeProd.recupereControleurTransport(i));
    }
    else {
        if (i<(P.getGamme()).getnombreTaches()) { //CREER TACHE DE TRANSPORT INTERMEDIAIRE
            Tache TacheActu=(Tache)((P.getGamme()).getListeGamme()).get(P.getAvancement()-1);
            Tache TacheSuiv=(Tache)((P.getGamme()).getListeGamme()).get(P.getAvancement());
            TacheM
            Tr=(P.getGamme()).creeTacheTransport("Tr_"+TacheActu.getIdTache()+"_"+TacheSuiv.getIdTache(),TacheActu,TacheSuiv,i);
            ListeTachesXML L=ListeProd.recupereListeTachesTransport(i);

            L.ajouteNoeudTransportTrie(Tr.getIdTache(),P.getNomProduit(),P.getPriorite(),Tr.getOrigine(),Tr.getDestination());
            L.versFichierXML("C:"+ "\\SERVEUR\\"+ListeProd.recupereControleurTransport(i));
        }
    }
}
else {
    //ENVOI D'ETAT PAR UNE MACHINE
    String etat=req.getParameter("etat");
    if(etat!=null) {
        //1. retrouver le produit
        idP=req.getParameter("idProduit");
        Produit P=ListeProd.chercheProduit(idP);
        if(etat.equals("fin_operation")) {
            //2. maj gamme
            P.avancerGamme();
            //3. maj listes : creation de la tâche suivante
            int i=P.getAvancement();
            if (i<(P.getGamme()).getnombreTaches()) { //TACHE INTERMEDIAIRE:AJOUT DANS LISTE DE TACHE SUIVANTE
                // -> ajout de la tache suivante de la gamme dans la liste XML suivante
                ListeTachesXML L=ListeProd.recupereListeTaches(i);
                Tache T=P.tacheSuivante();
                L.ajouteNoeudTrie(T.getNomTache(),idP,T.getDuree(),P.getPriorite());
                L.versFichierXML("C:"+ "\\SERVEUR\\"+ListeProd.recupereControleur(i));
            }
            if (i==(P.getGamme()).getnombreTaches()) { //DERNIERE TACHE -> CREER TACHE DE TRANSPORT DE FIN
                //mise à jour produit : produit complètement traité
                // -> ajout de la tache suivante de la gamme dans liste XML suivante (Transport vers sortie)
                Tache TacheActu=(P.getGamme()).getTache(i-1);
                TacheM
                Tr=(P.getGamme()).creeTacheTransportFin("Tr_"+TacheActu.getIdTache()+"_SORTIE",TacheActu);
                ListeTachesXML LT=ListeProd.recupereListeTachesTransport(i);
                LT.ajouteNoeudTransportTrie(Tr.getIdTache(),idP,P.getPriorite(),Tr.getOrigine(),Tr.getDestination());
                LT.versFichierXML("C:"+ "\\SERVEUR\\"+ListeProd.recupereControleurTransport(i));
            }
        }
        if(etat.equals("debut_operation")) {

```

```

//2. supprimer la tâche correspondante(normalement déjà supprimée par choix opérateur ou cariste)
int i=P.getAvancement();
Tache T=(P.getGamme()).getTache(i);
ListeTachesXML L=ListeProd.recupereListeTaches(i);
L.supprimeNoeud(T.getIdTache());
L.versFichierXML("C:"+"\SERVEUR\\"+ListeProd.recupereControleur(i));
String idR=T.getHumain();
if(idR!=null) {
    String idC=ListeProd.recupereControleur(i);
    pbConfiance(idR,idC,0);
}
}
if(etat.equals("occupee")) {
//3. suppression de la tache de transport correspondante (normalement déjà supprimée)
int i=P.getAvancement();
Tache T=(P.getGamme()).getTache(i);
T.setDebut(true);
ListeTachesXML LT=ListeProd.recupereListeTachesTransport(i);
LT.supprimeNoeudTransport(idP);
LT.versFichierXML("C:"+"\SERVEUR\\"+ListeProd.recupereControleurTransport(i));
Tache TR=(P.getGamme()).getTacheTransport(i);
if(TR.getHumain()==null) { //pb confiance, tache correctement executee mais jamais choisie
    String idR=null;
    String idC=ListeProd.recupereControleurTransport(i);
    pbConfiance(idR,idC,70);
}
else {
    String idR=P.getGamme().getTacheTransport(i).getHumain();
    String idC=ListeProd.recupereControleurTransport(i);
    pbConfiance(idR,idC,0);
}
}
if(etat.equals("libre")) {
int i=P.getAvancement();
if(i==P.getGamme().getnombreTaches()) {
    Tache TR=P.getGamme().getTacheTransport(i);
    if(TR.getHumain()==null) { //pb confiance : tache correctement effectuée mais non choisie
        String idR=null;
        String idC=ListeProd.recupereControleurTransport(i);
        pbConfiance(idR,idC,70);
    }
    else { //confiance OK
        String idR=P.getGamme().getTacheTransport(i).getHumain();
        String idC=ListeProd.recupereControleurTransport(i);
        pbConfiance(idR,idC,0);
    }
}
//mise à jour produit : produit complètement traité
P.setFin(true);
if(ListeProd.verifFin()==true) {
    try {

```



```

        risque=90;
        (P.getGamme()).supprimeTransport(i);
        LT.supprimeNoeudTransport(prod);
        LT.versFichierXML("C:"+"\SERVEUR\\"+ListeProd.recupereControleurTransport(i));
        res.setContentType("text"); //renvoyer la priorité du produit réellement traité
        PrintWriter out = res.getWriter();
        out.println(Preel.getPriorite());
    }
    else { //TACHE TRANSPORT NON EFFECTUEE
        Produit P=ListeProd.chercheProduit(prod);
        int i=P.getAvancement();
        Tache T=(P.getGamme()).getTache(i);
        if(T.getDebut()==true) { //pb au niveau de l'opérateur (transport effectué)
            idR=T.getHumain();
            idC=ListeProd.recupereControleur(i);
            risque=100;
            ListeTachesXML L=ListeProd.recupereListeTaches(i);
            L.ajouteNoeudTrie(T.getIdTache(),prod,T.getDuree(),P.getPriorite());
            L.versFichierXML("C:"+"\SERVEUR\\"+ListeProd.recupereControleur(i));
        }
        else { //pb au niveau du transport
            TacheM Tr=P.getGamme().getTacheTransport(i);
            idR=Tr.getHumain();
            idC=ListeProd.recupereControleurTransport(i);
            risque=100;
            if(i==(P.getGamme()).getnombreTaches()) {
                ListeTachesXML LT=ListeProd.recupereListeTachesTransport(i);
                LT.ajouteNoeudTransportTrie(Tr.getIdTache(),prod,P.getPriorite(),Tr.getOrigine(),Tr.getDestination());
                LT.versFichierXML("C:"+"\SERVEUR\\"+idC);
            }
            else {
                ListeTachesXML L=ListeProd.recupereListeTaches(i);
                if(T.getDebut()==false) {
                    L.ajouteNoeudTrie(T.getIdTache(),prod,T.getDuree(),P.getPriorite());
                }
                L.versFichierXML("C:"+"\SERVEUR\\"+ListeProd.recupereControleur(i));
            }
        }
    }
    pbConfiance(idR,idC,risque);
}
}
}
}
}

public void init() { //construction listes
    try {
        ListeProd = new ProduitJDBC("dbproduit","root","upzmmu14");
    }
}

```

```

        Conf = new Confiance("dbconfiance","root","upzmmul4");
    } catch (Exception ex) {
        ex.getMessage();
    }
}

public void pbConfiance(String idR,String idC, int risq) { //construction de preuves
    try {
        Conf.ajoutePreuve(idR,idC,risq);
    } catch (SQLException e) {
        e.printStackTrace();
    } catch (ClassNotFoundException e) {
        e.printStackTrace();
    }
}
}

```

Classe Produit JDBC

```

import java.sql.*;
import java.util.*;

public class ProduitJDBC extends Produit {

    ArrayList ListeProduits;
    ArrayList ListesTaches;
    ArrayList ListeControleurs;
    ArrayList ListesTachesTransport;
    ArrayList ListeControleursTransport;
    String dsn;
    String user;
    String mdp;

    public ProduitJDBC(String dsnPROD, String userPROD, String mdpPROD) throws SQLException, ClassNotFoundException {
        dsn=dsnPROD;
        user=userPROD;
        mdp=mdpPROD;
        ListeProduits=new ArrayList();
        ListesTaches=new ArrayList();
        ListeControleurs=new ArrayList();
        ListesTachesTransport=new ArrayList();
        ListeControleursTransport=new ArrayList();
        //CONNEXION A LA BD PRODUIT
        Connection connect=null;
        try {

```

```

        Class.forName("com.mysql.jdbc.Driver").newInstance();
    } catch (Exception ex) { System.out.println("SQLException: "+ex.getMessage()); }
    try {
        connect = DriverManager.getConnection("jdbc:mysql://localhost/"+dsn+"?user="+user+"&password="+mdp);
    } catch (SQLException ex) {
        System.out.println("SQLException: " + ex.getMessage());
        System.out.println("SQLState: " + ex.getSQLState());
        System.out.println("VendorError: " + ex.getErrorCode());
    }
    //CHERCHER NB DE TACHES DANS LA GAMME POUR CREER LISTES : autant de controleurs que de taches
    Statement S_1 = connect.createStatement();
    ResultSet RSet=S_1.executeQuery("select idProduit from Produit");
    RSet.first();
    String idProd=RSet.getString("idProduit");
    String ancienneTache="IN";
    Statement S_2 = connect.createStatement();
    ResultSet RGSet=S_2.executeQuery("select idT from Gamme where idP='"+idProd+"'");
    while(RGSet.next()) {
        String idTache=RGSet.getString("idT");
        Statement S_3 = connect.createStatement();
        ResultSet RTSet = S_3.executeQuery("select nomtache,dureetache,type from tache where idtache='"+idTache+"'");
        RTSet.next();
        String typeT=RTSet.getString("type");
        ListesTaches.add(new ListeTachesXML(typeT,RTSet.getString("nomtache")));
        ListeControleurs.add("C"+idTache);
        ListesTachesTransport.add(new ListeTachesXML("M","Tache transport"));
        ListeControleursTransport.add("C"+ancienneTache+idTache);
        ancienneTache=idTache;
        RTSet.close();
        S_3.close();
    }
    RGSet.close();
    S_2.close();
    RSet.close();
    S_1.close();
    ListesTachesTransport.add(new ListeTachesXML("M","Tache transport"));
    ListeControleursTransport.add("C"+ancienneTache+"OU");
    //CREER PRODUITS ET METTRE TACHES DANS LA LISTE CORRESPONDANTE
    Statement S1 = connect.createStatement();
    ResultSet RS=S1.executeQuery("select idProduit,priorite from Produit");
    while(RS.next()) {
        //exploitation de la ligne courante
        Gamme G = new Gamme();
        String idP=RS.getString("idProduit");
        System.out.println("CREATION DU PRODUIT "+idP);
        int pP=RS.getInt("priorite");
        Statement S2 = connect.createStatement();
        ResultSet RG=S2.executeQuery("select idT from Gamme where idP='"+idP+"'");
        while(RG.next()) {
            String idT=RG.getString("idT");

```

```

Statement S3 = connect.createStatement();
ResultSet RT = S3.executeQuery("select nomtache,dureetache,type from tache where idtache='"+idT+"'");
while(RT.next()) {
    String nT=RT.getString("nomtache");
    int dT=RT.getInt("dureetache");
    String tT=RT.getString("type");
    Tache T=new Tache(idT+idP,nT,dT,tT);
    if(G.getnombreTaches()==0) {
        ((ListeTachesXML>ListesTaches.get(0)).ajouteNoeudTrie(idT+idP,idP,dT,pP);
    }
    G.ajouteTache(0,T);
}
RT.close();
S3.close();
}
RG.close();
S2.close();
ListeProduits.add(new Produit(idP,G,pP));
} //ligne suivante
RS.close();
S1.close();
connect.close();
//TRANSFORMATION EN FICHIERS XML DES LISTES DE TACHES
ListeProduits.trimToSize();
ListesTaches.trimToSize();
ListeControleurs.trimToSize();
ListesTachesTransport.trimToSize();
ListeControleursTransport.trimToSize();
for(int i=0;i<ListesTaches.size();i++) {
    ((ListeTachesXML>ListesTaches.get(i)).versFichierXML("C:"+"\SERVEUR\\"+ListeControleurs.get(i));
}
for(int i=0;i<ListesTachesTransport.size();i++) {
    ((ListeTachesXML>ListesTachesTransport.get(i)).versFichierXML("C:"+"\SERVEUR\\"+ListeControleursTransport.get(i));
}
}

public void reconstruireListes() throws SQLException, ClassNotFoundException {
//CONNEXION A LA BD PRODUIT
Connection connect=null;
try {
    Class.forName("com.mysql.jdbc.Driver").newInstance();
} catch (Exception ex) { System.out.println("SQLException: "+ex.getMessage()); }
try {
    connect = DriverManager.getConnection("jdbc:mysql://localhost/"+dsn+"?user="+user+"&password="+mdp);
} catch (SQLException ex) {
    System.out.println("SQLException: " + ex.getMessage());
    System.out.println("SQLState: " + ex.getSQLState());
    System.out.println("VendorError: " + ex.getErrorCode());
}
//CREER PRODUITS ET METTRE TACHES DANS LA LISTE CORRESPONDANTE SI PRODUIT PAS ENCORE EXISTANT (=changements planning)

```

```

Statement S1 = connect.createStatement();
ResultSet RS=S1.executeQuery("select idProduit,priorite from Produit");
while(RS.next()) {
    //exploitation de la ligne courante
    String idP=RS.getString("idProduit");
    if(chercheProduit(idP)==null) {
        int pP=RS.getInt("priorite");
        Gamme G = new Gamme();
        Statement S2 = connect.createStatement();
        ResultSet RG=S2.executeQuery("select idT from Gamme where idP='"+idP+"'");
        while(RG.next()) {
            String idT=RG.getString("idT");
            Statement S3 = connect.createStatement();
            ResultSet RT = S3.executeQuery("select nomtache,dureetache,type from tache where idtache='"+idT+"'");
            while(RT.next()) {
                String nT=RT.getString("nomtache");
                int dT=RT.getInt("dureetache");
                String tT=RT.getString("type");
                Tache T=new Tache(idT+idP,nT,dT,tT);
                if(G.getnombreTaches()==0) {
                    ((ListeTachesXML>ListesTaches.get(0)).ajouteNoeudTrie(idT+idP,idP,dT,pP);
                }
                G.ajouteTache(0,T);
            }
            RT.close();
            S3.close();
        }
        RG.close();
        S2.close();
        ListeProduits.add(new Produit(idP,G,pP));
    }
} //ligne suivante
RS.close();
S1.close();
connect.close();
//TRANSFORMATION EN FICHIERS XML DES LISTES DE TACHES
ListeProduits.trimToSize();
for(int i=0;i<ListesTaches.size();i++) {
    ((ListeTachesXML>ListesTaches.get(i)).versFichierXML("C:"+"\SERVEUR\\"+ListeControleurs.get(i));
}
for(int i=0;i<ListesTachesTransport.size();i++) {
    ((ListeTachesXML>ListesTachesTransport.get(i)).versFichierXML("C:"+"\SERVEUR\\"+ListeControleursTransport.get(i));
}
}

public Produit chercheProduit(String nomP) { //RETROUVER UN PRODUIT DANS LA LISTE DES PRODUITS
    Produit P=null;
    for(int i=0;i<ListeProduits.size();i++) {
        P=(Produit)(ListeProduits.get(i));
        if (P.compareProduit(nomP)) {

```

```

        }
        return P;
    }
}
return null;
}

public Produit chercheProduitTache(String idT) { //RETRouver LE PRODUIT CORRESPONDANT A UNE TACHE
    Produit P=null;
    for(int i=0;i<ListeProduits.size();i++) {
        P=(Produit)ListeProduits.get(i);
        if(P.chercherTache(idT)) {
            return P;
        }
    }
    return null;
}

public ListeTachesXML recupereListeTaches(int i) {
    return (ListeTachesXML>ListesTaches.get(i);
}

public ListeTachesXML recupereListeTachesTransport(int i) {
    return (ListeTachesXML>ListesTachesTransport.get(i);
}

public String recupereControleur(int i) {
    return (String>ListeControleurs.get(i);
}

public String recupereControleurTransport(int i) {
    return (String>ListeControleursTransport.get(i);
}

public ListeTachesXML recupereTaches(String idControleur) {
    ListeTachesXML L=null;
    for(int i=0;i<ListeControleurs.size();i++) {
        if((ListeControleurs.get(i)).equals(idControleur)) {
            L=(ListeTachesXML)(ListesTaches.get(i));
            return L;
        }
    }
    for(int i=0;i<ListeControleursTransport.size();i++) {
        if((ListeControleursTransport.get(i)).equals(idControleur)) {
            L=(ListeTachesXML)(ListesTachesTransport.get(i));
            return L;
        }
    }
    return null;
}
}

```

```

    public boolean verifFin() {
        for(int i=0;i<ListeProduits.size();i++) {
            if (((Produit)ListeProduits.get(i)).getFin()==false) {
                System.out.println("PRODUIT "+((Produit)ListeProduits.get(i)).getNomProduit()+" PAS FINI ");
                return false;
            }
        }
        return true;
    }
}

```

Classe ListeTachesXML

```

import org.w3c.dom.*;
import javax.xml.parsers.*;
import org.apache.xml.serialize.*;
import java.io.*;

```

```

public class ListeTachesXML {

    protected Document ListeTaches;
    protected Element Racine;
    protected File fic;
    protected String typeTaches;

    public ListeTachesXML() {};

    public ListeTachesXML(String type,String nom) {
        ListeTaches=null;
        typeTaches=type;
        DocumentBuilderFactory fabrique = null;
        try {
            fabrique = DocumentBuilderFactory.newInstance();
            DocumentBuilder builder = fabrique.newDocumentBuilder();
            ListeTaches = builder.newDocument();
            Racine=ListeTaches.createElement("LISTETACHES");
            Racine.setAttribute("TypeTaches",type);
            Racine.setAttribute("NomTache",nom);
            ListeTaches.appendChild(Racine);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}

```

```

public void ajouteNoeud(String IdT,String IdP,int dT, int pP) {
    if(!chercheNoeud(IdT)) {
        Element Premier=ListeTaches.createElement("TACHE");
        Premier.setAttribute("IdTache",IdT);
        Integer duree=new Integer(dT);
        Premier.setAttribute("Duree",duree.toString());
        Element Second=ListeTaches.createElement("IDPRODUIT");
        Integer priorite=new Integer(pP);
        Second.setAttribute("Priorite",priorite.toString());
        Text SecondTxt=ListeTaches.createTextNode(IdP);
        Racine.appendChild(Premier);
        Premier.appendChild(Second);
        Second.appendChild(SecondTxt);
    }
}

public void afficheNoeud(Element n) {
    System.out.println("Noeud:"+n.getTextContent());
}

public boolean supprimeNoeud(String IdT) {
    NodeList lTaches = ListeTaches.getElementsByTagName("TACHE");
    Node n=null;
    for(int i=0;i<lTaches.getLength();i++) {
        n=lTaches.item(i);
        if (((Element)n).getAttribute("IdTache").equals(IdT)) {
            Racine.removeChild(n);
            return true;
        }
    }
    return false;
}

public boolean supprimeNoeudTransport(String nomProd) {
    NodeList lTaches = ListeTaches.getElementsByTagName("TACHE");
    Node n=null;
    for(int i=0;i<lTaches.getLength();i++) {
        n=lTaches.item(i);
        if ((n.getFirstChild().getTextContent()).equals(nomProd)) {
            Racine.removeChild(n);
            return true;
        }
    }
    return false;
}

public boolean chercheNoeud(String IdT) {
    NodeList lTaches=ListeTaches.getElementsByTagName("TACHE");
    Node n=null;

```

```

        for(int i=0;i<lTaches.getLength();i++) {
            n=lTaches.item(i);
            if (((Element)n).getAttribute("IdTache").equals(IdT)) {
                return true;
            }
        }
        return false;
    }

    public void afficheList() {

        NodeList lTaches=ListeTaches.getElementsByTagName("TACHE");
        Node n=null;
        for(int i=0;i<lTaches.getLength();i++) {
            n=lTaches.item(i);
            System.out.println("TACHE "+i+1+" : "+((Element)n).getAttribute("IdTache"));
        }
    }

    public void ajouteNoeudTrie(String IdT,String IdP,int dT, int pP) {
        if(!chercheNoeud(IdT)) {
            NodeList lProduits = ListeTaches.getElementsByTagName("IDPRODUIT");
            int prior=new Integer(pP).intValue();
            Node noeudAInserer=null;
            int i;
            for(i=0;i<lProduits.getLength();i++) {
                Node n=lProduits.item(i);
                int priorite=new Integer(((Element)n).getAttribute("Priorite")).intValue();
                if (priorite<=prior) {
                    noeudAInserer=n;
                    break;
                }
            }
            if(i==lProduits.getLength()) {
                noeudAInserer=null;
            }
            Element Premier=ListeTaches.createElement("TACHE");
            Premier.setAttribute("IdTache",IdT);
            Integer duree=new Integer(dT);
            Premier.setAttribute("Duree",duree.toString());
            Element Second=ListeTaches.createElement("IDPRODUIT");
            Integer prioriteProduit=new Integer(pP);
            Second.setAttribute("Priorite",prioriteProduit.toString());
            Text SecondTxt=ListeTaches.createTextNode(IdP);

            if(noeudAInserer==null) {
                Racine.appendChild(Premier);
            }
            else {
                Racine.insertBefore(Premier,noeudAInserer.getParentNode());
            }
        }
    }
}

```

```

    }
    Premier.appendChild(Second);
    Second.appendChild(SecondTxt);
}
}

public void ajouteNoeudTransportTrie(String IdT,String IdP, int pP, String o, String d) {
    if(!chercheNoeud(IdT)) {
        NodeList lProduits = ListeTaches.getElementsByTagName("IDPRODUIT");
        int prior=new Integer(pP).intValue();
        Node noeudAInserer=null;
        int i;
        for(i=0;i<lProduits.getLength();i++) {
            Node n=lProduits.item(i);
            int priorite=new Integer(((Element)n).getAttribute("Priorite")).intValue();
            if (priorite<=prior) {
                noeudAInserer=n;
                break;
            }
        }
        if(i==lProduits.getLength()) {
            noeudAInserer=null;
        }
        Element Premier=ListeTaches.createElement("TACHE");
        Premier.setAttribute("IdTache",IdT);
        Element Second=ListeTaches.createElement("IDPRODUIT");
        Integer priorite=new Integer(pP);
        Second.setAttribute("Priorite",priorite.toString());
        Text SecondTxt=ListeTaches.createTextNode(IdP);
        Element Third=ListeTaches.createElement("ORIGINE");
        Text ThirdTxt=ListeTaches.createTextNode(o);
        Element ThirdBis=ListeTaches.createElement("DESTINATION");
        Text ThirdBisTxt=ListeTaches.createTextNode(d);

        if(noeudAInserer==null) {
            Racine.appendChild(Premier);
        }
        else {
            Racine.insertBefore(Premier,noeudAInserer.getParentNode());
        }
        Premier.appendChild(Second);
        Second.appendChild(SecondTxt);
        Premier.appendChild(Third);
        Third.appendChild(ThirdTxt);
        Premier.appendChild(ThirdBis);
        ThirdBis.appendChild(ThirdBisTxt);
    }
}

public void versFichierXML(String nomFichier) {

```

```

    try {
        XMLSerializer ser;
        if (nomFichier.equals("")) {
            ser = new XMLSerializer(System.out, new OutputFormat("xml", "UTF-8", true));
        }
        else {
            fic = new File(nomFichier+".xml");
            ser = new XMLSerializer(new FileWriter(fic), new OutputFormat("xml", "UTF-8", true));
        }
        ser.serialize(ListeTaches);
    } catch (Exception e) {e.printStackTrace();}
}

public void fluxSortie(PrintWriter out) throws FileNotFoundException, IOException {
    try {
        String ligne ;
        BufferedReader fichier = new BufferedReader(new FileReader(fic));
        while ((ligne = fichier.readLine()) != null) {
            out.print(ligne);
        }
        fichier.close();
    } catch (Exception e) {
        e.printStackTrace();
    }
}

public String getTypeListe() {
    return typeTaches;
}

public void finListe() {
    Element Premier=ListeTaches.createElement("FIN");
    Racine.appendChild(Premier);
}
}

```

Les composants d'un Contrôleur de Ressources

Classe clientHTTP

```
import java.io.*;
import java.net.*;
import org.xml.sax.*;

public class clientHTTP {

    private Socket client;
    protected static String serveur="localhost";
    protected static int port=8443;
    protected SocketAddress adr;
    protected String reponse="";

    public clientHTTP(String serv,int p) {
        serveur=serv;
        port=p;
        client=null;
        try{
            client=new Socket(serv,port);
        }catch(Exception ex){
            erreur("Impossible de se connecter au service ("+serveur+","+port+"), erreur : "+ex.getMessage(),3);
            return;
        }
    }

    public clientHTTP() {
        client=null;
        try{
            client=new Socket(serveur,port);
            client.setReuseAddress(true);
            adr=client.getRemoteSocketAddress();
            client.close();
        }catch(Exception ex){
            erreur("Impossible de se connecter au service ("+serveur+","+port+"), erreur : "+ex.getMessage(),3);
            return;
        }
    }

    public void connectClient() {
        try {
            client=new Socket();
        }
    }
}
```

```

        client.setReuseAddress(true);
        client.connect(adr);
    } catch (IOException e) {
        e.printStackTrace();
    }
}

public void requeteHTTP(String param, String IdControleur, RessourceJDBC R){
    connectClient();
    PrintWriter OUT=null;
    BufferedReader IN=null;
    String commande=null;
    String réponse=null;
    try {
        //envoi commande pour demande liste taches XML
        OUT=new PrintWriter(client.getOutputStream(),true);
        IN=new BufferedReader(new InputStreamReader(client.getInputStream()));
        commande="GET /ControleurCentral/CC"+param+" HTTP/1.0";
        OUT.println(commande);
        OUT.println("");
        while((réponse=IN.readLine())!=null) {
            if (réponse.equals("")) break;
        }
        //decomposition de la liste XML reçue
        corpsReponse(IN,IdControleur,R);
    }catch(Exception ex){
        System.err.println("Envoi : l'erreur suivante s'est produite : " +ex.getMessage());
    }
    try{
        IN.close();
        OUT.close();
        client.close();
    }catch(Exception ex){
        System.out.println("[Envoi : fin du thread d'envoi des commandes au serveur]");
    }
}

public boolean requetePOST(String param){
    connectClient();
    PrintWriter OUT=null;
    BufferedReader IN=null;
    String commande=null;
    String réponse=null;
    String S="";
    try {
        //envoi commande pour envoi ressources, choix et etats
        OUT=new PrintWriter(client.getOutputStream(),true);
        IN=new BufferedReader(new InputStreamReader(client.getInputStream()));
        commande="POST /ControleurCentral/CC"+param+" HTTP/1.0";
        OUT.println(commande);
    }
}

```

```

        OUT.println("");
        System.out.println("Envoi de la commande "+commande+" au serveur.");
        //réception réponse
        réponse=IN.readLine();
        while((réponse=IN.readLine())!=null) {
            if (réponse.equals("")) break;
        }
        //cas ou une tache n'existe plus dans la liste (suppression noeud impossible) -> tache deja choisie
        if((S=IN.readLine())!=null) {
            if(S.equals("Erreur")) {
                System.out.println("CETTE TACHE A DEJA ETE CHOISIE.");
                return false;
            }
            else {
                reponse=S;
            }
        }
    }catch(Exception ex){
        System.err.println("Envoi : l'erreur suivante s'est produite : " +ex.getMessage());
    }
    try{
        IN.close();
        OUT.close();
        client.close();
    }catch(Exception ex){
        System.out.println("[Envoi : fin du thread d'envoi des commandes au serveur]");
    }
    return true;
}

public String getReponse() {
    return reponse;
}

public void corpsReponse(BufferedReader IN, String IdControleur, RessourceJDBC R) {
    //decomposition liste XML
    try
    {
        InputSource S=new InputSource(IN);
        Class c = Class.forName("org.apache.xerces.parsers.SAXParser");
        XMLReader reader = (XMLReader)c.newInstance();
        analyseListeXML handler = new analyseListeXML(IdControleur,R);
        reader.setContentHandler(handler);
        reader.parse(S);
    }
    catch(Exception e){System.out.println("Erreur pour le fichier XML du controleur "+IdControleur+" : "+e);}
}

public void erreur(String msg, int exitCode) {
    System.err.println(msg);
}

```

```

        System.exit(exitCode);
    }
}

```

Classe AnalyseListeXML

```

import java.sql.SQLException;
import org.xml.sax.*;
import org.xml.sax.helpers.*;

/**
 * Classe utilisee pour gérer les evenement emis par SAX lors du traitement du fichier XML
 */
class analyseListeXML extends DefaultHandler {

    private String tagCourant = "";
    protected String donnees;
    protected String parametre;
    protected String typeListe;
    protected RessourceJDBC RJ; //ressources machines auxquelles on affecte automatiquement des taches
    protected String ressource;
    protected String tache;
    protected String o;
    protected String d;
    protected String IdP;
    protected String pP;
    protected String IdControleur;
    protected String nomTache;
    protected int DureeTache;
    protected ListeTachesRessXML ListeTachesRessources; //liste taches opérateurs (attention : n'y mettre que 3 taches, les plus prioritaires)
    protected int nbTaches=0; //nb actuels de taches dans la liste des opérateurs
    protected int nbAffect=0; //nb de ressources réservées
    protected int nbRessources;

    public analyseListeXML(String IdC, RessourceJDBC R) {
        super();
        parametre="";
        ressource=null;
        tache="";
        typeListe="";
        nomTache="";
        IdP="";
        IdControleur=IdC;
        RJ=R;
    }
}

```

```

        pP="";
    }

    /**
     * Actions a réaliser lors de la detection d'un nouvel element.
     */
    public void startElement(String nameSpace, String localName, String qName, Attributes attr) throws SAXException {
        tagCourant = localName;
        if(tagCourant.equals("LISTETACHES")) {
            typeListe=attr.getValue("TypeTaches");
            nomTache=attr.getValue("NomTache");
            if(!typeListe.equals("A")) { //construction listes XML ressource
                ListeTachesRessources=new ListeTachesRessXML(IdControleur);
            }

            if(!typeListe.equals("M")) {
                try {
                    RJ.connectionRessource();
                    nbRessources=RJ.nbRessources(IdControleur);
                } catch (SQLException e) {
                    e.printStackTrace();
                } catch (ClassNotFoundException e) {
                    e.printStackTrace();
                }
            }
        }
        if (tagCourant.equals("TACHE")) {
            tache=attr.getValue("IdTache");
            if(!typeListe.equals("M")) { //CAS SEMI-AUTO OU AUTO : RECHERCHE RESSOURCE MACHINE
                try {
                    DureeTache=new Integer(attr.getValue("Duree")).intValue();
                } catch(Exception ex) {}
                nbAffec++;
            }
        }
        if (tagCourant.equals("IDPRODUIT")) {
            String prior=attr.getValue("Priorite");
            Integer prioriteProduit=new Integer(prior);
            pP=prioriteProduit.toString();
        }
        if (tagCourant.equals("FIN")) {
            ListeTachesRessources.finListe();
        }
    }

    /**
     * Actions à réaliser lors de la détection de la fin d'un element.
     */
    public void endElement(String nameSpace, String localName, String qName) throws SAXException {
        tagCourant=localName;
    }

```

```

if (tagCourant.equals("IDPRODUIT")) {
    if(!typeListe.equals("M")) { //CAS SEMI-AUTO OU AUTO : RECHERCHE RESSOURCE MACHINE
        try {
            if(nbAffec<nbRessources) {
                RJ.connectionRessource();
                ressource=RJ.ressourceLibre(DureeTache,IdControleur,IdP);
                if(ressource!=null) {
                    parametre="?idtache="+tache+"&ressource="+ressource+"&produit="+IdP;
                    (new clientHTTP()).requetePOST(parametre);
                }
            }
            else {
                ressource=null;
            }
        } catch(Exception ex) {}
    }

    if(ressource!=null) { //TACHE PAS ENCORE AFFECTEE A UNE RESSOURCE ET RESSOURCE LIBRE TROUVEE
        if(typeListe.equals("SA")) { //SI PAS TACHE AUTOMATIQUE, RECHERCHER UNE RESSOURCE HUMAINE
            //AJOUTER TACHE DANS LA LISTE DES OPERATEURS
            nbTaches++;
            if (nbTaches<3) { //on ne met que les 3 taches les plus prioritaires
                ListeTachesRessources.ajouteNoeudRessourceTrie(tache,nomTache,IdP,"",pP,ressource);
            }
        }
    }
}

if (tagCourant.equals("DESTINATION")) {
    nbTaches++;
    if (nbTaches<3) { //SI PAS TACHE AUTOMATIQUE, RECHERCHER UNE RESSOURCE HUMAINE
        ListeTachesRessources.ajouteNoeudTransportTrie(tache,nomTache,IdP,pP,o,d);
    }
}

tagCourant = "";
parametre="";
ressource=null;
}

/**
 * Actions à réaliser au début du document.
 */
public void startDocument() {
}

/**
 * Actions à réaliser lors de la fin du document XML.
 */
public void endDocument() {
    //MAJ FICHER LISTE DES OPERATEURS
    if(!typeListe.equals("A")) {

```

```

        ListeTachesRessources.versFichierXML("C:\\CONTROLEUR\\"+IdControleur);
    }
}

/**
 * Actions à réaliser sur les données
 */
public void characters(char[] caracteres, int debut, int longueur) throws SAXException {
    donnees = new String(caracteres, debut, longueur);
    if (tagCourant.equals("IDPRODUIT")) {
        IdP=donnees;
    }
    if (tagCourant.equals("ORIGINE")) {
        o=donnees;
    }
    if (tagCourant.equals("DESTINATION")) {
        d=donnees;
    }
}
}
}

```

Servlet (valable ici pour un CRM)

```

import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;
import java.io.PrintWriter;
import java.sql.SQLException;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

public class CT1 extends HttpServlet {

    private static final long serialVersionUID = 1L;
    RessourceJDBC RJ;
    String IdControleur;

    //RECEPTION D'UNE COMMANDE GET
    protected void doGet(HttpServletRequest req, HttpServletResponse res) throws ServletException, IOException {
        //DEMANDE LISTE PAR UNE RESSOURCE HUMAINE
        clientHTTP cli = new clientHTTP();
        String parametre="?idControleur="+IdControleur;
        cli.requeteHTTP(parametre, IdControleur, RJ);
    }
}

```

```

        //renvoi liste
        String xslTest="xsl"+IdControleur+".xsl";
        //envoi de liste
        PrintWriter out=res.getWriter();
        res.setContentType("text/xml");
        String réponse="<?xml version=\"1.0\" encoding=\"windows-1252\"?>"+<?xml-stylesheet type=\"text/xsl\"
href=\""+xslTest+"\"?>\n";
        String ligne ;
        BufferedReader fichier = new BufferedReader(new FileReader("C:\\\\CONTROLEUR\\\\"+IdControleur+".xml"));
        ligne=fichier.readLine(); //on enlève la première ligne <xml...>
        while ((ligne = fichier.readLine()) != null) {
            réponse+=ligne+"\n";
        }
        fichier.close();
        out.println(réponse);
    }

//RECEPTION D'UNE COMMANDE POST
protected void doPost(HttpServletRequest req, HttpServletResponse res) throws ServletException, IOException {
    String choixTache=req.getParameter("choix");//recup de la ressource aussi importante si moteur de confiance
    if(choixTache!=null) { //CHOIX DE TACHE PAR UN OPERATEUR
        String ressource=req.getParameter("ressource");
        String produit=req.getParameter("produit");
        //POST vers serveur pour tache choisie -> enlever tache de liste des taches
        clientHTTP cli = new clientHTTP();
        String parametre="?ressource="+ressource+"&idTacheChoisie="+choixTache;
        if (cli.requetePOST(parametre)==false) { //suppression de la tache impossible -> tache déjà choisie
            //renvoi erreur pour non lancement de la tache
            res.setContentType("text");
            PrintWriter out = res.getWriter();
            out.println("Erreur");
        }
        else { //tache possible -> indication traitement tache
            try {
                Ressource R=RJ.chercheRessource(ressource);
                R.ajouteChoix(produit);
                RJ.changeEtat(ressource,"choix");
                RJ.tacheActuelle(produit,ressource);
            } catch (SQLException e) {
                e.printStackTrace();
            } catch (ClassNotFoundException e) {
                e.printStackTrace();
            }
        }
        PrintWriter out=res.getWriter();
        res.setContentType("text/html");
        out.println("<html>");
        out.println("<head>");
        out.println("<title>Taches</title>");
        out.println("</head>");
        out.println("<body>");
    }
}

```

```

        out.println("<center>");
        out.println("<H3>Traitement de la tâche "+choixTache+".</H3><hr>");
        out.println("<form name=\"frmTaches\" action=\"http://localhost:8080/Control/\"+IdControleur+\" method=\"POST\">");
        out.println("<table>");
        out.println("<tr>Identifiant contrôleur : <input type=\"text\" name=\"idControleur\" value=\"+IdControleur+\" </tr>");
        out.println("<tr><input type= \"submit\" value=\"Recommencer\"></tr>");
        out.println("</table>");
        out.println("</form>");
        out.println("</center>");
        out.println("</body>");
        out.println("</html>");
    }
}
else {
    //ENVOI D'ETAT PAR UNE RESSOURCE MACHINE
    String ressource=req.getParameter("ressource");
    String etat=req.getParameter("etat");
    if (ressource!=null && etat!=null) {
        try {
            System.out.println("Mettre la ressource "+ressource+" a l'etat "+etat+".");
            //SI ETAT=FIN_OP METTRE LISTES A JOUR
            if(etat.equals("fin_operation")) {
                String p=RJ.getProduit(ressource);
                RJ.changeEtat(ressource,etat);
                clientHTTP cli = new clientHTTP();
                String parametre="?idProduit="+p+"&etat="+etat;
                cli.requetePOST(parametre);
            }
            //ETAT=OCUPEE : VERIFICATION DU PRODUIT DEPOSE
            if(etat.equals("occupee")) {
                String p=RJ.getProduit(ressource);
                RJ.changeEtat(ressource,etat);
                String produit=req.getParameter("produit"); //produit réellement traité sur la machine
                if(!produit.equals(p) && p!=null) { //produit réel différent de produit prévu
                    /*calculer confiance*/
                    /*1. maj BD*/
                    /*maj de la ressource normalement prévue pour le produit*/
                    String ressourcePrevue=RJ.chercheAffectation(produit,IdControleur);
                    if(ressourcePrevue!=null) {
                        if(RJ.getEtat(ressourcePrevue).equals("reservee")) {
                            RJ.changeEtat(ressourcePrevue,"libre");
                        }
                    }
                    /*maj la tâche réellement exécutée sur la ressource*/
                    RJ.tacheActuelle(produit,ressource);
                    /*maj de la tâche pour l'opérateur concerné par le produit prévu*/
                    String operateurPrevu=RJ.chercheOperateur(p,IdControleur);
                    if(operateurPrevu!=null) {
                        RJ.tacheActuelle(produit,operateurPrevu);
                        Ressource R=RJ.chercheRessource(operateurPrevu);
                        R.ajouteChoix(produit);
                    }
                }
            }
        }
    }
}

```

```

    }
    /*2.réhabiliter tâche prévue (=annuler la suppression) et la tâche transport correspondante*/
    /*3.supprimer la tâche réelle (et créer la tâche suivante...) et maj produit correspondant*/
    clientHTTP cli = new clientHTTP();
    String parametre="?pbProduit="+p+"&produit="+produit+"&ressource="+ressource;
    cli.requetePOST(parametre);
    /*4.retourner priorité du produit réellement traité et la machine prévue pour ce produit*/
    res.setContentType("text"); //renvoyer la priorité du produit réellement traité
    PrintWriter out = res.getWriter();
    out.println(cli.getReponse());
    out.println(ressourcePrevue);
}
else {
    clientHTTP cli = new clientHTTP();
    String parametre="?idProduit="+p+"&etat="+etat;
    cli.requetePOST(parametre);
}
//lancement timer : verifier au bout de 15 min si debut_op sur machine sinon pbconfiance
try {
    String operateur=RJ.chercheOperateur(produit,IdControleur);
    if(operateur!=null) {
        Ressource R=RJ.chercheRessource(operateur);
        //lancement timer : verifier au bout de x min si deb_op sur machine sinon pbconfiance
        R.lanceTimer();
    }
} catch (SQLException e) {
    e.printStackTrace();
}
}
//ETAT=DEBUT_OP : METTRE LISTES A JOUR (CAS OU LES OPERATEURS N'ONT PAS ENVOYE UNE TACHE QUI A ETE FAITE)
if(etat.equals("debut_operation")) {
    RJ.changeEtat(ressource,etat);
    //maj action operateur pour confiance
    String p=RJ.getProduit(ressource);
    String operateur=RJ.chercheOperateur(p,IdControleur);
    if(operateur!=null) {
        try {
            Ressource R=RJ.chercheRessource(operateur);
            RJ.changeEtat(operateur,"realisation");
            R.setAction("realisation");
            R.stopTimer();
            //RJ.changeEtat(ress,"fin_operation");
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
}
clientHTTP cli = new clientHTTP();
String parametre="?idProduit="+p+"&etat="+etat;
cli.requetePOST(parametre);
}

```

```

        //ETAT LIBRE : UTILE POUR CONFIANCE DANS LES TRANSPORTEURS
        if(etat.equals("libre")) {
            String p=RJ.getProduit(ressource);
            RJ.changeEtat(ressource,etat);
            clientHTTP cli = new clientHTTP();
            String parametre="?idProduit="+p+"&etat="+etat;
            cli.requetePOST(parametre);
        }
        }catch(Exception ex) {}
    }
    else { //PAGE HTML FORMULAIRE D'UNE RESSOURCE : DEMANDE LISTE
        doGet(req,res);
    }
}

}

public void init() {
    IdControleur="CT1";
    RJ=new RessourceJDBC();
    destroy();
}

public void destroy() {
    //REMISE A 0 BD RESSOURCE CONTROLEUR
    try {
        RJ.connectionRessource();
        RJ.razBD(IdControleur);
        RJ.creeRessources(IdControleur);
    } catch(Exception ex) {}
}

}

```

Annexe 3. Interactivité

Cette annexe est relative au paragraphe 5.3. Elle présente les listes et les traitements réalisés au niveau du Contrôleur Central et du CRM Préparation lorsque un opérateur souhaite exécuter une opération de préparation. Il s'agit en fait d'un enchaînement de copies d'écran obtenues lors de l'exécution du programme.

CONTROLEUR CENTRAL

CONTROLEUR PREPA

OPERATEUR PREPA

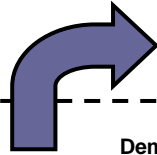
Construction liste tâches préparation (XML)

```
<?xml version="1.0" encoding="UTF-8" ?>
- <LISTETACHES NomTache="Tache
  preparation" TypeTaches="SA">
- <TACHE Duree="15" IdTache="T1M4">
  <IDPRODUIT
    Priorite="3">M4</IDPRODUIT>
  </TACHE>
- <TACHE Duree="15" IdTache="T1M3">
  <IDPRODUIT
    Priorite="2">M3</IDPRODUIT>
  </TACHE>
- <TACHE Duree="15" IdTache="T1M2">
  <IDPRODUIT
    Priorite="2">M2</IDPRODUIT>
  </TACHE>
- <TACHE Duree="15" IdTache="T1M5">
  <IDPRODUIT
    Priorite="1">M5</IDPRODUIT>
  </TACHE>
- <TACHE Duree="15" IdTache="T1M1">
  <IDPRODUIT
    Priorite="1">M1</IDPRODUIT>
  </TACHE>
</LISTETACHES>
```

Votre controleur : CT1
 La tâche T1M4 est affectée à la machine AB1.
 Création de la tâche de transport de Stock entree vers AB1.
 La tâche T1M5 est affectée à la machine AB2.
 Création de la tâche de transport de Stock entree vers AB2.
 La tâche T1M2 est affectée à la machine AB3.
 Création de la tâche de transport de Stock entree vers AB3.

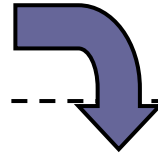
Création transports (XML)

```
<?xml version="1.0" encoding="UTF-8" ?>
- <LISTETACHES NomTache="Tache transport"
  TypeTaches="M">
- <TACHE IdTache="Tr_Stock_T1M4">
  <IDPRODUIT Priorite="3">M4</IDPRODUIT>
  <ORIGINE>Stock entree</ORIGINE>
  <DESTINATION>AB1</DESTINATION>
  </TACHE>
- <TACHE IdTache="Tr_Stock_T1M3">
  <IDPRODUIT Priorite="2">M3</IDPRODUIT>
  <ORIGINE>Stock entree</ORIGINE>
  <DESTINATION>AB2</DESTINATION>
  </TACHE>
- <TACHE IdTache="Tr_Stock_T1M2">
  <IDPRODUIT Priorite="2">M2</IDPRODUIT>
  <ORIGINE>Stock entree</ORIGINE>
  <DESTINATION>AB3</DESTINATION>
  </TACHE>
</LISTETACHES>
```



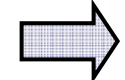
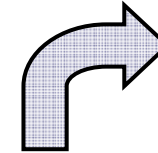
Demande liste préparation

Envoi de la commande GET /ControleurCentral/CC?idControleur=CT1 HTTP/1.0 au serveur.
 Reponse commande : HTTP/1.1 200 OK
 <---Server: Apache-Coyote/1.1
 <---Content-Type: text/xml;charset=ISO-8859-1
 <---Content-Length: 588
 <---Date: Mon, 30 Jan 2006 14:24:35 GMT
 <---Connection: close



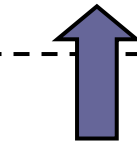
Affectation de ressources

Envoi de la commande POST /ControleurCentral/CC?idtache=T1M4&ressource=AB1&produit=M4 HTTP/1.0 au serveur.
 Réponse commande : HTTP/1.1 200 OK
 <---Server: Apache-Coyote/1.1
 <---Content-Length: 0
 <---Date: Mon, 30 Jan 2006 14:24:36 GMT
 <---Connection: close
 Envoi de la commande POST /ControleurCentral/CC?idtache=T1M5&ressource=AB2&produit=M5 HTTP/1.0 au serveur.
 Réponse commande : HTTP/1.1 200 OK
 <---Server: Apache-Coyote/1.1
 <---Content-Length: 0
 <---Date: Mon, 30 Jan 2006 14:24:36 GMT
 <---Connection: close
 Envoi de la commande POST /ControleurCentral/CC?idtache=T1M2&ressource=AB3&produit=M2 HTTP/1.0 au serveur.
 Réponse commande : HTTP/1.1 200 OK
 <---Server: Apache-Coyote/1.1
 <---Content-Length: 0
 <---Date: Mon, 30 Jan 2006 14:24:36 GMT
 <---Connection: close



Création tâches opérateurs (XML)

```
<?xml version="1.0" encoding="UTF-8" ?>
- <LISTETACHES idControleur="CT1">
- <TACHE IdTache="T1M4"
  NomTache="Tache preparation">
  <IDPRODUIT>M4</IDPRODUIT>
  <RESSOURCE>AB1</RESSOURCE>
  </TACHE>
- <TACHE IdTache="T1M3"
  NomTache="Tache preparation">
  <IDPRODUIT>M3</IDPRODUIT>
  <RESSOURCE>AB2</RESSOURCE>
  </TACHE>
- <TACHE IdTache="T1M2"
  NomTache="Tache preparation">
  <IDPRODUIT>M2</IDPRODUIT>
  <RESSOURCE>AB3</RESSOURCE>
  </TACHE>
</LISTETACHES>
```



Demande liste de tâches

CHOIX DE TACHES



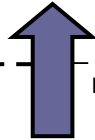
Choix de tâches

Id controleur :

Tache preparation de M4 sur AB1
 Tache preparation de M3 sur AB2
 Tache preparation de M2 sur AB3

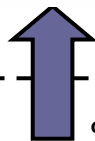
Suppression tâche préparation

```
<?xml version="1.0" encoding="UTF-8" ?>
- <LISTETACHES NomTache="Tache
  preparation" TypeTaches="SA">
- <TACHE Duree="15" IdTache="T1M3">
  <IDPRODUIT
    Priorite="2">M3</IDPRODUIT>
  </TACHE>
- <TACHE Duree="15" IdTache="T1M2">
  <IDPRODUIT
    Priorite="2">M2</IDPRODUIT>
  </TACHE>
- <TACHE Duree="15" IdTache="T1M5">
  <IDPRODUIT
    Priorite="1">M5</IDPRODUIT>
  </TACHE>
- <TACHE Duree="15" IdTache="T1M1">
  <IDPRODUIT
    Priorite="1">M1</IDPRODUIT>
  </TACHE>
</LISTETACHES>
```



Demande mise à jour listes

Envoi de la commande POST /ControleurCentral/CC?idTacheChoisie=T1M4 HTTP/1.0 au serveur.
 Réponse commande : HTTP/1.1 200 OK
 <---Server: Apache-Coyote/1.1
 <---Content-Type: text/xml;charset=ISO-8859-1
 <---Content-Length: 490
 <---Date: Tue, 31 Jan 2006 09:52:20 GMT
 <---Connection: close



Choix opérateur

Choix de tâches

Id controleur :

Tache preparation de M4 sur AB1

Tache preparation de M3 sur AB2

Tache preparation de M2 sur AB3

Choix de tâches

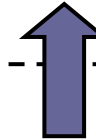
Id controleur :

Tache preparation de M3 sur AB2

Tache preparation de M2 sur AB3

Création tâche régulation

```
<?xml version="1.0" encoding="UTF-8" ?>
- <LISTETACHES NomTache="Tache
  equipement" TypeTaches="A">
- <TACHE Duree="25"
  IdTache="T2M4">
  <IDPRODUIT
    Priorite="3">M4</IDPRODUIT>
  </TACHE>
</LISTETACHES>
```



Demande mise à jour listes

Mettre la ressource AB1 a l'etat fin_operation.
 Envoi de la commande POST /ControleurCentral/CC?idTache=T1M4 HTTP/1.0 au serveur.
 Réponse commande : HTTP/1.1 200 OK
 <---Server: Apache-Coyote/1.1
 <---Content-Length: 0
 <---Date: Tue, 31 Jan 2006 10:13:45 GMT
 <---Connection: close



Indication fin de tâche

TIME :20
 Envoi de la commande POST /Control/UI?ressource=AB1&etat=fin_operation HTTP/1.0 au serveur.

Liste des figures et des tableaux

- Figure 1. Développement du système social dans les entreprises.
- Figure 2. Evolution des paradigmes de production.
- Figure 3. Pyramide d'intégration de la production.
- Figure 4. Modèle du comportement humain (modèle de Rasmussen révisé par Hoc).
- Figure 5. Modèle d'interactivité du système de pilotage.
- Figure 6. Les fonctions de pilotage.
- Figure 7. L'approche prédictive.
- Figure 8. L'approche proactive.
- Figure 9. L'approche réactive.
- Figure 10. L'architecture centralisée.
- Figure 11. L'architecture hiérarchique.
- Figure 12. L'architecture coordonnée.
- Figure 13. L'architecture hétérarchique.
- Figure 14. L'architecture hybride.
- Figure 15. Modèle global de notre système de pilotage.
- Figure 16. Un ensemble d'opérateurs statiques.
- Figure 17. Un ensemble d'opérateurs semi-mobiles.
- Figure 18. Un ensemble d'opérateurs totalement mobiles.
- Figure 19. Architecture du SAI.
- Figure 20. Use-case de traitement d'un ordre de fabrication.
- Figure 21. Diagramme de classes.
- Figure 22. Diagramme de séquence.
- Figure 23. Méthode de passage de UML à la simulation.
- Figure 24. Principe d'un modèle à files d'attente.
- Figure 25. Processus de simulation.
- Figure 26. Réservation d'une ressource par une entité.
- Figure 27. Le diagramme de séquence et le modèle de simulation associé.
- Figure 28. Modèles de simulation du système de pilotage.
- Figure 29. Duplication des entités Produit et Ordre de Fabrication.
- Figure 30. Modèle de simulation du Contrôleur Central.
- Figure 31. Modèle de simulation global d'un Contrôleur de Ressources.
- Figure 32. Modèle de simulation d'un CRS.
- Figure 33. Modèle de simulation d'un CRT.
- Figure 34. Système de feux.
- Figure 35. Schéma simplifié de l'unité de réglages moteurs.
- Figure 36. Le SAI de l'unité de réglages moteurs.
- Figure 37. Animation du modèle de simulation du SAI couplé à l'unité de réglages.
- Figure 38. Scénario de simulation partiel du SAI couplé à l'unité de réglages.
- Figure 39. Le modèle Client/Serveur.
- Figure 40. Les modes de communication synchrone et asynchrone.
- Figure 41. Structure générale d'un système distribué avec middleware.
- Figure 42. Communications internes au SAI.
- Figure 43. Le modèle Client/Serveur pour le SAI.

Figure 44. Le protocole HTTP.
Figure 45. Communications HTTP du SAI.
Figure 46. Schéma d'une servlet.
Figure 47. Le rôle bipartite d'un Contrôleur de Ressources.
Figure 48. Schéma de développement du SAI.
Figure 49. Enchaînement des écrans opérateurs.
Figure 50. Gestion des timers pour l'environnement de test du SAI.
Figure 51. Représentation simplifiée du service d'endoscopie.
Figure 52. Le SAI de l'unité d'endoscopie.
Figure 53. Le déroulement d'un acte de gastroscopie.
Figure 54. Le déroulement d'un acte d'endoscopie.
Figure 55. Temps de cycle des moteurs dans le cadre d'une utilisation optimale du SAI.
Figure 56. Temps de cycle des moteurs dans un cas de mauvaise utilisation du SAI (A).
Figure 57. Temps de cycle des moteurs dans un cas de mauvaise utilisation du SAI (B).
Figure 58. Architecture d'un moteur de confiance.
Figure 59. Arbre des observations.
Figure 60. Un moteur de confiance couplé au SAI.

Tableau 1. Place de l'homme au sein des paradigmes de production.

Tableau 2. Positionnement du SAI.

Tableau 3. Positionnement des contrôleurs du SAI.

Tableau 4. Valeurs de confiance (en %) des opérateurs de l'unité de réglages moteurs.