# CHAPITRE 1

# MODÉLISATION ET OUTIL D'AIDE À LA DÉCISION

# 1 Modélisation et outil d'aide à la décision

## 1.1 Prise de décision et modèle

La *prise de décision* est « un processus qui consiste à sélectionner un ensemble d'actions, afin d'atteindre un ou plusieurs objectifs » NDLR : traduit de l'Anglais [Simon, 1977]. Ainsi une activité de gestion telle la planification implique un éventail de décisions : Que fait-on ?, Quand ?, Comment ? Où ? Avec qui ? D'autres activités telles l'organisation ou le contrôle sont aussi associées à la *prise de décision*.

Les gestionnaires de l'industrie sont constamment amenés à prendre des décisions à propos de différents types de processus ou d'installations. Ces processus ou installations sont appelés des systèmes. Un système est « une collection d'entités, comme des personnes ou des machines, qui interagissent ensembles, dans le but d'atteindre un objectif commun » NDLR : traduit de l'Anglais [Law et Kelton, 2000].

Bien souvent, les décisions des gestionnaires sont prises afin d'améliorer un système. Ainsi, on cherche à accroître ses performances en le rendant plus efficace et efficient [Turban et Aronson, 2000]. L'efficacité sert à mesurer à quel degré les objectifs sont atteints. On l'associe aux extrants du système, c'est-à-dire les produits finis ou les résultats, par exemple, le nombre d'anodes fabriquées par un vibrocompacteur. Quant à l'efficience, il s'agit de la mesure des intrants nécessaires à la génération des extrants, par exemple, la quantité de pâte nécessaire à la fabrication d'une anode.

Un *problème* survient lorsqu'un système ne rencontre pas ses objectifs, ne produit pas les résultats attendus, ou ne fonctionne pas selon les spécifications. Dans ce contexte, les gestionnaires doivent chercher à améliorer la performance du système. Or, bien souvent, les *problèmes* de l'industrie sont dits non-structurés, c'est-à-dire qu'ils ne possèdent pas de méthode de résolution standard [Turban et Aronson, 2000]. Toutefois, on s'entend généralement sur un *processus de décision* générique, formé de quatre phases distinctes, pour solutionner n'importe quel type de problèmes [Simon, 1977].

La première phase du *processus de décision* est l'*intelligence*. Elle consiste à effectuer un examen complet du système, à identifier et définir le problème. Au cours de la seconde phase, dite de *conception*, on met en œuvre un modèle, c'est-à-dire une représentation du système et on propose un ensemble de solutions. La phase suivante, dite du *choix* ou de la *décision*, implique de sélectionner une solution et de la tester à l'aide du modèle, pour s'assurer de sa viabilité. La phase finale, dite d'*implantation*, consiste à appliquer la solution choisie au système réel.

Comme on peut le constater, la *prise de décision* et la *solution du problème* sont des concepts fortement connexes. C'est pourquoi on les utilise généralement de façon interchangeable. Par ailleurs, au cœur même du *processus de décision*, on retrouve la notion de modèle. Un modèle est défini comme « une représentation d'un système concret qui permet d'investiguer les améliorations de ce système, ou de découvrir les effets de différents scénarios sur celui-ci. NDLR : traduit de l'Anglais [Pidd, 1998]. Le modèle repose sur un ensemble d'hypothèses qui prennent la forme de relations mathématiques ou logiques. Il s'agit d'un outil qui permet d'appuyer les gestionnaires au cours de la *prise de décision*. Le présent mémoire s'intéresse donc aux modèles, en tant qu'outils d'aide à la décision, appliqués à un système particulier, en l'occurrence le cycle de vie des anodes.

## 1.2 Étude de système et modélisation

Il existe principalement deux manières d'effectuer l'étude d'un système : l'*expérimentation réelle* et l'*expérimentation à partir d'un modèle*. « L'expérimentation réelle consiste à modifier certaines propriétés physiques du système et à le laisser opérer sous les nouvelles conditions. » NDLR : traduit de l'Anglais [Law et Kelton, 2000].

L'expérimentation à partir d'un modèle constitue la seconde manière d'effectuer une étude [Law et Kelton, 2000]. Elle implique de concevoir une représentation du système, c'est-à-dire le modèle, qui comme on l'a vu est à la base de la *prise de décision*. Le processus qui mène à la conception du modèle est la modélisation. Elle est divisée en deux branches : *iconique* et *symbolique*.

Les modèles *iconiques* ou *physiques* sont une représentation matérielle du système existant, comme la maquette d'un édifice en architecture [Pidd, 1998]. Les modèles *symboliques*, aussi appelés *mathématiques*, représentent les relations logiques et quantitatives du système réel [Hoover et Perry, 1990]. Un modèle symbolique peut notamment servir à concevoir une *heuristique*, effectuer une *simulation* ou obtenir une *solution analytique* [Law et Kelton, 2000] [Turban et Aronson, 2000]. Le diagramme ci-dessous illustre les différentes manières de mener à bien une étude et les principales utilisations des modèles symboliques.

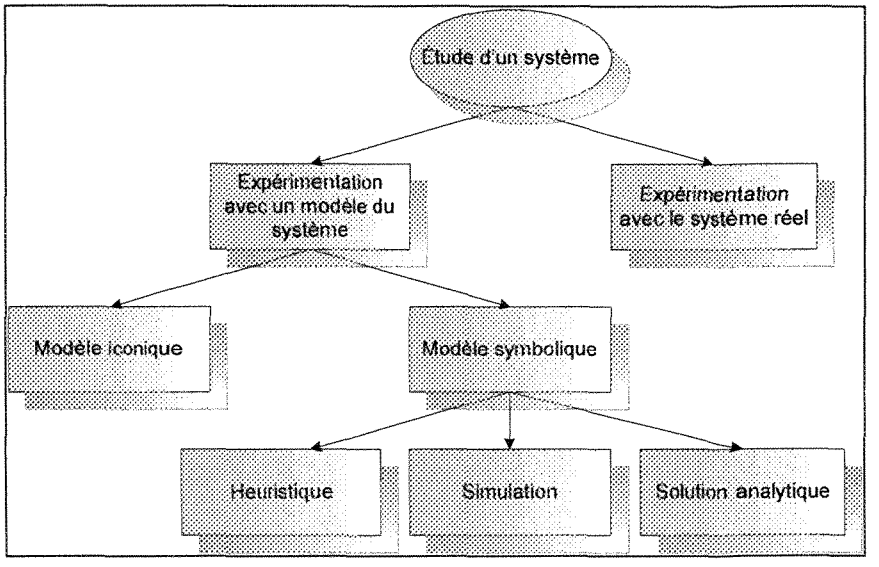


Figure 1 : étude d'un système

Lorsque le modèle est assez simple, il est possible d'utiliser ses relations et un ensemble de valeurs pour obtenir ce qu'on appelle une *solution analytique* [Law et Kelton, 2000]. On peut penser à des méthodes d'optimisation comme la programmation linéaire [Turban et Aronson, 2000]. Celle-ci permet d'estimer un espace de solution à partir d'un modèle, c'est-à-dire un ensemble d'équations et d'inéquations mathématiques [Teghem, 2003]. Cette approche n'a pas été considérée, puisque le type de décision qui nous intéresse découle de problématiques dont on ne connaît pas la procédure de résolution. Quant à l'*heuristique*, on la définie comme suit :

> Une règle empirique ou une méthode établie qui peut aider à trouver une solution, sans pour autant que celle-ci soit assurée, comme ce serait le cas avec un algorithme NDLR : traduit de l'Anglais [Giarratano et Riley, 1998].

Une méthode *heuristique* est donc une astuce qui permet de cibler une solution potentielle. Pour ce faire, on cherche et on évalue des pistes de solution, puis on raffine la recherche, à partir des connaissances acquises [Turban et Aronson, 2000]. Notons que notre modèle ne vise pas à découvrir une solution à partir de nouvelles *heuristiques* ou de méthodes reconnues, telle la recherche avec tabous ou les algorithmes génétiques [Dréo et Siarry, 2003]. Cependant, nous nous sommes intéressés à la *connaissance heuristique* [Friedman-Hill, 2003], découlant du système étudié, soit le cycle de vie des anodes, plus particulièrement, de la possibilité d'adjoindre cette connaissance à un *système expert* [Gonzalez et Dankel, 1993]. Cette question est approfondie au cours des chapitres suivants.

Dans le cadre du mémoire, nous nous intéressons à la modélisation afin d'effectuer des simulations logicielles. Ce type de simulation consiste à « implémenter le modèle sous la forme d'un programme informatique et d'imiter, donc de simuler, le comportement du système réel, en soumettant différents scénarios d'opération » NDLR : traduit de l'Anglais [Pidd, 1998]. Le modèle agit donc comme un véhicule d'expérimentation dans le cadre d'une approche essai et erreur. Ainsi, à la lumière des résultats obtenus, au cours d'une ou plusieurs simulations, les gestionnaires sont en mesures de choisir le scénario le plus profitable. Le diagramme ci-dessous illustre le *processus de décision*, lorsqu'on fait appel à des simulations.

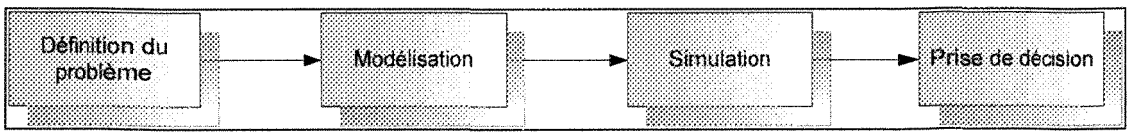


Figure 2 : prise de décision lors d'une modélisation et simulation

Le présent chapitre discute de modélisation et de simulation, en raison de l'intérêt que suscite cette approche au niveau de la prise de décision. Mentionnons qu'afin d'alléger le texte,

nous parlerons de *modélisation* mais qu'on sous-entend aussi la méthode de *simulation* qui lui est associée.

## 1.3 Avantages et désavantages de la modélisation

La modélisation d'un système n'est pas une panacée. Il s'agit d'un processus qui est habituellement coûteux et long. Dans bien des cas, on l'utilise une fois que d'autres approches, telle la modélisation iconique, les solutions analytiques, voire même l'expérimentation réelle, ont été tentées en vain [Pidd, 1998]. En contrepartie, on aboutit souvent à cette méthode, parce que les systèmes étudiés sont forts complexes [Law et Kelton, 2000]. Mentionnons toutefois que les avancements continuels des technologies de l'information, tant au niveau matériel que logiciel, font de la modélisation une avenue de plus en plus intéressante [Banks et al, 1998]. D'ailleurs, la liste ci-dessous énumère les principaux avantages et désavantages de cette approche.

**Les avantages :**

- *Compréhension des causes* : L'analyse des causes, à l'origine de certains phénomènes, constitue un des intérêts central des gestionnaires de l'industrie. La modélisation répond à ce besoin, en reconstruisant et en étudiant de façon minutieuse, les scénarios d'opération du système d'intérêt [Banks et al, 1998]. Par exemple, on peut se servir d'un modèle pour identifier des contraintes, telles des goulots d'étranglements, responsables d'un ralentissement des opérations.
- *Compréhension des interactions :* Les installations manufacturières modernes sont forts complexes. Souvent, il apparaît même impossible de considérer toutes les interactions qui se produisent à un moment précis. La modélisation permet de mieux appréhender un système, en analysant les interactions qui surviennent entre leurs variables clés. On peut donc prédire l'effet de ces variables sur sa performance [Banks et al, 1998].
- *Compression et extension du temps* : En comprimant ou dilatant le temps de simulation, le modèle permet d'accélérer ou de décélérer le fonctionnement du système et de l'étudier sur

différentes échelles [Law et Kelton, 2000]. Il est donc possible d'examiner un quart de travail en quelques instants, ou de simuler pendant plusieurs heures, les événements qui se produisent durant une minute.

- *Exploration de scénarios* : Même si la conception d'un modèle est un processus relativement long, une fois à terme, celui-ci permet de simuler le système à partir de paramètres multiples et d'une échelle de temps variable. On peut donc explorer et comparer un vaste éventail de scénarios [Banks et al, 1998].
- *Minimisation des coûts* : La modélisation permet de tester chaque aspect d'une modification du système et ce, sans engager de ressource [Banks et al, 1998]. Ainsi, on évite les problèmes propres à l'expérimentation réelle, puisqu'une fois qu'une décision est appliquée au système existant, un retour à l'arrière peut s'avérer fort coûteux, voire impossible [Pidd, 1998]. À ce titre, l'investissement pour la mise en œuvre d'un modèle, correspond habituellement à 1% du montant attendu pour la conception, ou la réingénierie du système [Banks et al, 1998].
- *Réplétion des simulations* : La modélisation permet de répéter plusieurs fois un même scénario de simulation. Il s'agit d'un avantage certain, car dans la réalité une expérience est rarement répétable [Pidd, 1998].
- *Simulation dynamique et événements transitoires* : Les solutions analytiques sont une avenue intéressante pour étudier un système. Cependant, elles comportent des désavantages lorsqu'on les compare à la modélisation. Notamment, parce qu'elles ne tiennent pas toujours compte de l'aspect dynamique ou des événements transitoires du système [Pidd, 1998]. Par exemple, les actions suivant l'interruption d'un procédé quelconque ne suscitent pas le même intérêt que celles impliquées durant une situation optimale.
- *Simulation des scénarios extrêmes* : L'un des objectifs de la modélisation est de pouvoir simuler les conséquences de scénarios extrêmes [Pidd, 1998]. Or, l'expérimentation de tels scénarios sur un système réel peut être excessivement dangereuse, voire totalement illégale.

**Les désavantages :**

- *Comparaison des scénarios* : Un modèle est basé sur des variables aléatoires [Baillargeon, 1990] [Banks et al, 1998]. Un scénario simulé produit un estimé des caractéristiques réelles du modèle, pour un ensemble de paramètres d'entrées donnés. Ainsi, pour chaque scénario, plusieurs essais indépendants doivent être effectués, afin d'obtenir un portrait global des caractéristiques du modèle. Pour cette raison, il est difficile de comparer plusieurs scénarios entre eux [Law et Kelton, 2000]. À l'inverse, les méthodes analytiques donnent les caractéristiques précises du modèle, pour un ensemble de paramètres d'entrées donnés.
- *Confiance abusive* : Le nombre important d'informations d'une simulation ou l'impact d'une animation dynamique peuvent générer une confiance excessive en une étude. Ainsi, un modèle invalide, peu importe la vraisemblance de ses résultats, risque de fournir peu d'informations sur les caractéristiques réelles du système [Law et Kelton, 2000].
- *Coût et temps* : La modélisation est une des méthodes les plus coûteuses pour étudier un système, que ce soit en en argent ou en temps. C'est particulièrement vrai lorsqu'on la compare aux méthodes analytiques [Law et Kelton, 2000].
- *Expertise de modélisation* : La conception d'un modèle requiert à la fois des spécialistes du système et du monde de la modélisation. La connaissance de la modélisation ne peut s'acquérir qu'avec le temps et l'expérience [Banks et al, 1998].
- *Interprétation des résultats* : Comme les résultats du modèle sont en fonction de variables aléatoires, il peut apparaître difficile de déterminer si les caractéristiques observées sont le résultat des interactions du modèle, ou de sa nature aléatoire [Banks et al, 1998].

## 1.4 Processus traditionnel de modélisation

Dans la littérature, des auteurs tels [Banks et al, 1998], [Hoover et Perry, 1990] et [Law et Kelton, 2000], s'entendent sur une démarche systématique commune, afin de mener à bien un projet de modélisation. Les jalons de cette démarche forment le processus de modélisation d'un système. La présente section a pour but d'introduire brièvement les étapes de ce processus.

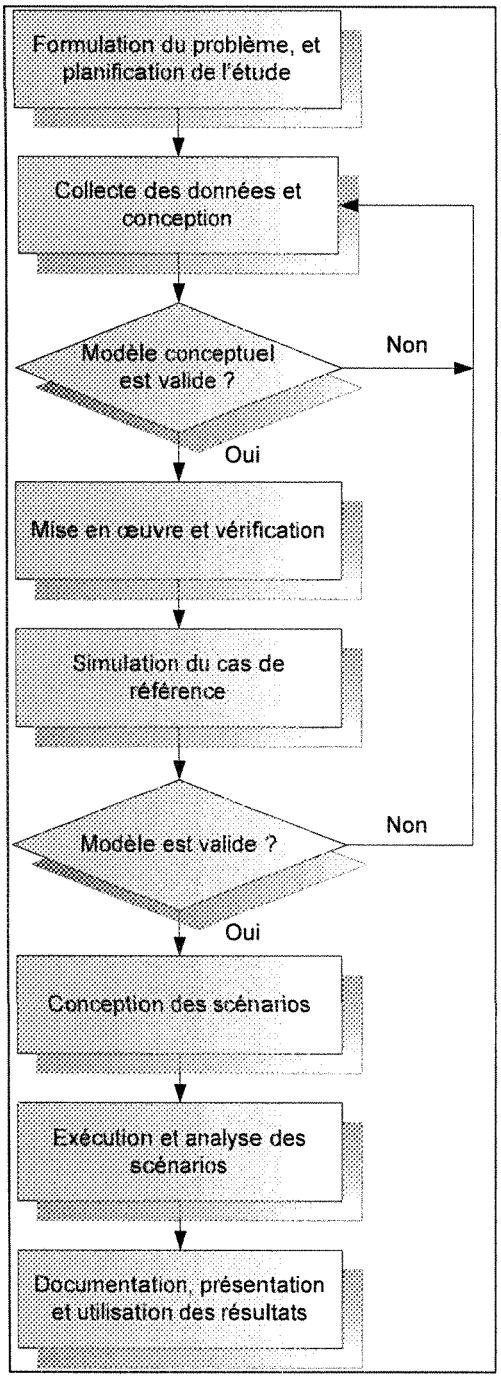


Figure 3 : processus de conception d'un modèle

**Formulation du problème et planification de l'étude :**

Un projet de modélisation débute par un énoncé du problème à résoudre. Les parties impliquées se réunissent ensuite pour discuter des objectifs généraux de l'étude, des interrogations spécifiques auxquelles elle doit répondre, des mesures de performances, de l'étendue du modèle, de la configuration du système modélisé, des logiciels ou technologies utilisés, de l'échéancier et finalement des ressources assignées.

**Collecte des données et conception du modèle :**

La seconde étape consiste à collecter des informations à propos du système, par exemple son fonctionnement, ses performances et ses procédures d'opération. À partir de ces indications, on définit des concepts comme les paramètres du modèle, les variables d'état, les entités impliquées, ou les distributions de probabilités utilisées. Ainsi, on peut poser des hypothèses sur le fonctionnement du système. Ces hypothèses constituent l'esquisse conceptuelle du modèle.

**Validation du modèle conceptuel :**

Les principaux acteurs impliqués dans le projet doivent ensuite valider le modèle conceptuel, c'est-à-dire chacune des hypothèses recensées à l'étape précédente. Ceci permet de s'assurer qu'elles sont conformes et complètes. En plus, cette étape évite d'éventuels retours en arrière, une fois que la mise en œuvre du modèle est entamée.

**Mise en œuvre du modèle et vérification :**

Au cours de la quatrième étape, le modèle est mis en œuvre à partir d'un langage de programmation quelconque ou d'un logiciel de simulation. Nous discutons de cette dernière possibilité à la section 1.8. Finalement, on doit vérifier l'exactitude du modèle, notamment en effectuant des tests unitaires, pour s'assurer qu'il fonctionne conformément aux hypothèses et pour corriger les défectuosités.

**Exécution et validation du modèle :**

Cette étape permet de s'assurer que le modèle est une représentation acceptable du système réel. Pour ce faire, on expérimente d'abord un scénario commun au modèle et au système existant. Ce scénario est ce qu'on appelle le cas de base ou de référence. Ensuite, on s'assure qu'il existe une certaine similitude entre les résultats des deux expériences, par exemple, en comparant leurs indicateurs de performances respectifs. Dans certains cas, le système de base est inexistant. Notamment, lorsqu'on réalise un projet de modélisation pour justifier la conception d'un nouveau système. C'est pourquoi il existe des méthodes de validations alternatives [Banks et al, 1998]. Celles-ci dépassent le cadre du présent mémoire. En effet, le type de modèles que nous considérons vise à soutenir la prise de décision, dans une optique d'amélioration d'un système existant. Néanmoins que le système de base soit réel ou non, il importe que les parties impliquées étudient les résultats, afin de valider le modèle. Une analyse de sensibilité [Law et Kelton, 2000] est aussi souhaitable, afin d'établir les paramètres qui ont le plus d'impact sur les mesures de performances. Ces paramètres doivent faire l'objet d'une attention particulière lors des étapes subséquentes.

**Conceptions de scénarios :**

Pour chaque scénario défini lors de la planification initiale, on doit déterminer des aspects tels la durée d'un essai de simulation (aussi appelée *réplication*) [Banks et al, 1998], le nombre d'essais, ainsi que la valeur des paramètres initiaux du modèle. Nous allons revenir à la notion d'essai un peu plus loin.

**Exécution et analyse des scénarios :**

Cette étape du processus permet d'expérimenter l'ensemble des scénarios simulés et d'estimer leurs performances respectives. Pour se faire, on s'appuie sur les indicateurs de performance définis lors de la planification. L'analyse des résultats doit également permettre de comparer les scénarios entre eux, afin d'établir celui qui est le plus performant.

**Documentation, présentation et utilisation des résultats :**

Cette étape consiste à documenter les hypothèses initiales, le fonctionnement du modèle et les résultats de l'étude. L'analyste en modélisation doit aussi présenter les résultats au requérant de l'étude. S'ils sont considérés suffisamment crédibles, les résultats peuvent être utilisés pour appuyer une décision stratégique.

## 1.5 Catégories de modèles

Comme on l'a mentionné, il existe plusieurs formes de modèles. Toutefois, dans un contexte de simulation, on a souvent recours à des modèles symboliques ou mathématiques. Ainsi, lorsque ce type de modèles est valide, il permet d'étudier le comportement du processus dans une optique d'amélioration [Hoover et Perry, 1990]. La discussion qui suit a pour but de présenter les principales catégories de modèles symboliques.

**Modèle statique versus dynamique :**

Les modèles statiques sont utilisés lorsqu'on doit considérer une représentation du système, à un moment précis, ou lorsque le temps n'est pas une considération essentielle. C'est pourquoi on a rarement recours à un tel type de modèle. La modélisation de Monte Carlo est la forme de modélisation statique la plus connue [Law et Kelton, 2000]. Elle permet d'estimer le fonctionnement d'un système, en s'appuyant sur les nombres aléatoires. Souvent, on a recours à cette méthode en mathématiques ou en physique, pour évaluer des intégrales définies fort complexes [Bratley et al, 1987]. À l'opposé de cette approche, la modélisation dynamique sert à représenter un processus dont les variables évoluent à travers le temps. On peut penser à un ensemble de clients qui se présentent à intervalle régulier, pour faire la file devant un guichet bancaire.

**Modèle déterministe versus probabiliste :**

Lorsque les variables d'un modèle n'intègrent aucune notion de probabilité, il est dit déterministe [Hoover et Perry, 1990]. Par exemple, un ensemble d'équations différentielles, décrivant l'air qui s'écoule le long d'une aile d'avion, est déterministe. Une fois que les paramètres

initiaux et les relations, d'un tel modèle, ont été définis, les résultats qu'il génère demeurent fixes. Néanmoins, bien des systèmes doivent être modélisés en tenant compte de la notion de probabilité. Cette notion est intimement liée au concept de variable aléatoire [Baillargeon, 1990]. Une variable aléatoire est définie ainsi :

> Une variable qui ne prendra sa valeur (parmi différentes possibles) qu'après un tirage au hasard, c'est-à-dire après l'aboutissement d'un processus (expérience), dit aléatoire, dont le résultat demeure en dehors du contrôle de l'observateur. [Kakmier, 1982].

En attribuant une probabilité à toutes les valeurs d'une variable aléatoire, on obtient une distribution de probabilité. Or, lorsqu'on modélise un système, certaines valeurs affectées aux variables du modèle peuvent être plus ou moins probables que d'autres. Donc, pour atteindre un certain degré de réalisme, on considère les variables du modèle comme aléatoires et on cherche à estimer leur distribution de probabilité [Law et Kelton, 2000]. Par exemple, on peut penser à un modèle qui émule le comportement d'une file d'attente, où l'intervalle d'arrivée des clients suit une loi de probabilité normale.

Une méthode minimaliste, fréquemment utilisée lorsqu'on conçoit un modèle probabiliste, consiste à simuler un scénario, au cours d'un essai d'une durée arbitraire. Cet essai est ensuite considéré comme un estimé des caractéristiques du modèle [Law et Kelton, 2000]. Toutefois, comme les variables d'un modèle probabiliste sont basées sur des distributions de probabilités [Baillargeon, 1990], il est possible qu'elles aient une très grande variance. Ainsi, l'estimé obtenu au cours d'un essai particulier peut différer grandement des caractéristiques réelles du système. D'où l'importance d'utiliser des techniques statistiques, afin de concevoir et d'analyser une expérience de simulation basée sur de nombreux essais [Hoover et Perry, 1990].

Notons que les modèles de Monte Carlo [Bratley et al, 1987] sont un cas typique de modélisation probabiliste. Il existe également une sous-famille des modèles probabilistes, dite stochastique. En fait, un modèle est stochastique, lorsqu'il est à la fois probabiliste et dynamique. Les variables aléatoires d'un tel modèle varient donc à travers le temps [Processus stochastique].

**Modèle discret versus continu :**

L'aspect discret ou continu fait référence à l'ensemble des valeurs que peuvent prendre les variables du système. Ainsi, une variable continue peut prendre n'importe quelle valeur réelle comprise dans un intervalle, tandis qu'une variable discrète peut uniquement prendre un ensemble limité de valeurs [Hoover et Perry, 1990]. Dans le cas des modèles dynamiques, cette distinction, entre le discret et le continu, est souvent faite par rapport aux variables qui sont associées au temps. Ainsi, un modèle est discret lorsque ses variables d'état changent instantanément à intervalle régulier [Law et Kelton, 2000]. Par exemple si l'on cherche à modéliser une file d'attente, on devrait normalement opter pour une représentation discrète, car les valeurs des variables d'états, tel le nombre de clients dans la file, sont modifiés aussitôt qu'un nouveau se présente. À l'inverse, les variables d'état d'un système continu changent de façon ininterrompue à travers le temps [Pidd, 1998]. Il importe donc de modéliser tous les instants possibles. Par exemple, le déplacement d'un avion doit être modélisé de façon continue, lorsqu'on considère des variables d'état comme la vitesse ou le carburant consommé. Les modèles continus sont généralement des équations différentielles qui statuent le taux de changement des variables d'états. Toutefois un système continu n'implique pas obligatoirement un modèle continu et vice versa. En fait, cette décision dépend des objectifs de l'étude et des variables qu'on choisit de modéliser [Law et Kelton, 2000].

## 1.6 La modélisation à événements discrets

Comme nous venons de le mentionner, les variables d'un modèle discret changent au cours d'instants précis et dénombrables. C'est lors de ces instants qu'un événement survient. On définit un événement comme « une occurrence instantanée au cours de laquelle l'état du système peut évoluer » NDLR : traduit de l'Anglais [Law et Kelton, 2000]. Toutefois, les modèles discrets ne comportent pas toujours des événements qui modifient ses variables d'état. Par exemple, un événement peut servir à planifier la fin de la simulation, ou le commencement d'une nouvelle opération. C'est pourquoi on dit que l'état *peut* évoluer lorsqu'un événement est déclenché.

Compte tenu de cette notion événementielle, un constat s'impose, à savoir qu'un modèle avec des événements discrets est implicitement dynamique, car les variables du système changent à travers le temps. De plus, il est généralement probabiliste. Néanmoins, cette notion est facultative, puisque les modèles déterministes peuvent être considérés comme des cas spéciaux des modèles probabilistes [Law et Kelton, 2000]. Dans la littérature, un modèle qui est à la fois discret, dynamique et probabiliste est qualifié comme un *modèle à événements discrets*. Les sections qui suivent visent à approfondir ce concept, car les outils d'aide à la décision que nous avons conçus sont basés sur celui-ci.

### 1.6.1 Terminologie de la modélisation à événements discrets

Nous allons maintenant définir la terminologie découlant des principales composantes d'un modèle à événements discrets.

**Les objets du système :**

*Entité* : « Il s'agit des objets individuels du système qui sont modélisés et dont le comportement ou l'état doit être explicitement suivi. » NDLR : traduit de l'Anglais [Pidd, 1998] Par exemple, chaque anode ou chaque convoyeur qui les déplacent, forment des entités du système de manutention et de scellement d'anodes.

*Ressource* : « Il s'agit d'une entité statique qui fournit un service à d'autres entités dynamiques. » NDLR : traduit de l'Anglais [Banks et al, 1998] Par exemple, un bassin chargé de refroidir un ensemble d'anodes constitue une ressource.

**Organisation des entités :**

Lorsque les entités sont modélisées de façon individuelle, il faut généralement les regrouper et les caractériser séparément. Dans ce contexte, voici la terminologie qui est généralement utilisée.

*Attribut* : « Ce sont des valeurs qui caractérisent une entité. » NDLR : traduit de l'Anglais [Law et Kelton, 2000] L'ensemble des attributs d'entités permet de modéliser l'état du système.

Dans le cadre d'un modèle avec des événements discrets, les attributs changent au cours d'instants précis. Par exemple, chaque convoyeur du cycle de vie des anodes possède un attribut spécifiant s'il est actif ou inactif. Celui-ci est modifié au gré d'événements tels des arrêts planifiés ou non-planifiés.

*Classe ou enregistrement* : « Ce sont des groupes permanents qui représentent des entités identiques ou similaires » NDLR : traduit de l'Anglais [Pidd, 1998]. Par exemple, chacune des entités *anodes* forment une classe du même nom, puisqu'elles ont toutes les mêmes attributs.

*File ou liste* : « Ce sont des groupes temporaires d'entités qui possèdent des propriétés communes. ». NDLR : traduit de l'Anglais [Law et Kelton, 2000] On utilise fréquemment ce concept, afin de représenter une file d'entités en attente [Pidd, 1998]. En effet, à mesure que les entités traversent le système, elles requièrent l'utilisation de ressources. Lorsqu'une ressource n'est pas disponible, alors l'entité joint une file. Dans ce contexte, les files peuvent aussi être considérées comme des entités, puisqu'elles possèdent des attributs descriptifs. Par exemple, considérons un *convoyeur*, du cycle de vide des anodes, qui s'avère être une file d'entités *anodes*. Celui-ci possède des attributs tels son *état* (actif / inactif) ou le *nombre d'anodes* dans la file.

**Opérations des entités :**

Au fur et à mesure qu'une simulation avance, les entités interagissent entre elles et leurs états changent. Une terminologie précise est nécessaire pour décrire ces opérations.

*Activité* : « Il s'agit d'une opération ou d'une procédure qui est initiée lorsqu'un événement débute » NDLR : traduit de l'Anglais [Pidd, 1998]. C'est donc en raison des activités que l'état des entités est modifié. La durée de l'activité peut être une constante, une valeur aléatoire issue d'une distribution statistique, une valeur calculée à partir d'une équation, etc. [Banks et al, 1998] Par exemple, considérons l'activité *Refroidir anode* qui consiste à faire transiter une anode crue, au sein d'un bassin pour abaisser sa température. Cette activité contribue à faire évoluer l'état de l'entité *Anode* de *Chaude* à *Froide*.

La modélisation discrète implique l'identification des classes d'entités et des activités dans lesquelles elles s'engagent. Ainsi, il est possible de lier ces activités entre elles, afin de visualiser le cycle de vie de chaque classe d'entités. Les diagrammes d'activités [Fowler, 2004] sont une façon de modéliser les interactions entre les classes d'entités [Pidd, 1998]. Ils conviennent particulièrement lorsqu'on les applique à des systèmes dont la structure de base est la file. Ci-dessus se trouve une partie du diagramme des activités du cycle de vie des anodes. Il est formalisé à partir du formalisme UML qui est défini comme suit :

> Une famille de notations graphiques, supportées par un méta-modèle, qui aide à décrire et concevoir des systèmes logiciels, plus particulièrement des systèmes basés sur le formalisme orienté objet. NDLR : traduit de l'Anglais [Fowler, 2004].

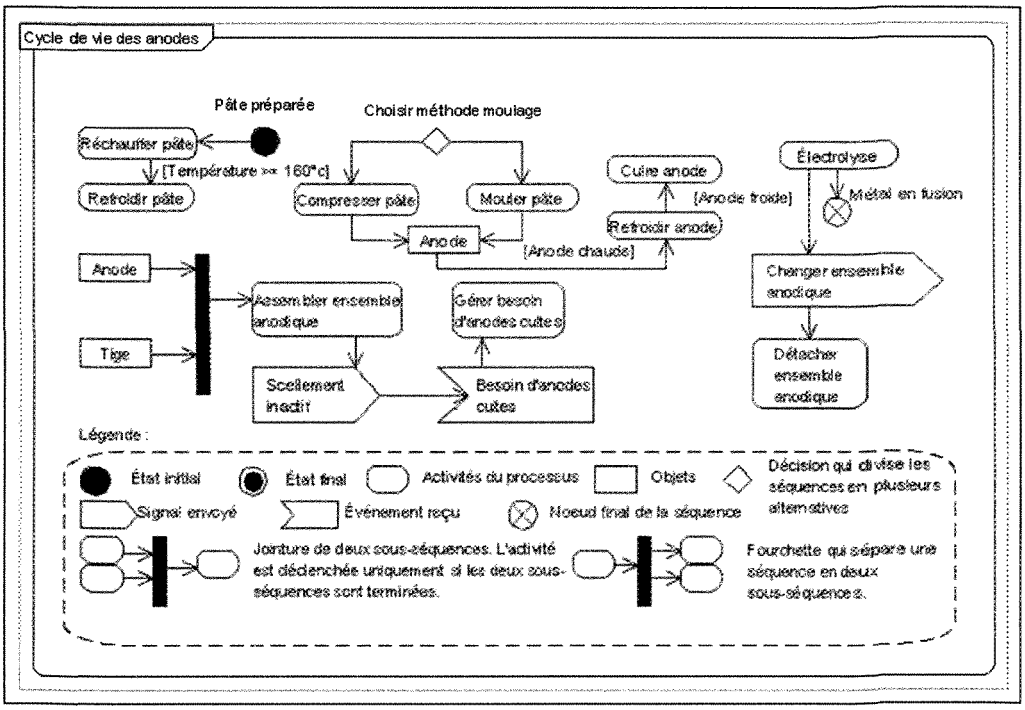


Figure 4 : diagramme d'activités

Il s'agit d'une notation utilisée tout au long du mémoire. Nous nous abstenons de la décrire en détails, comme elle est abondamment documentée, notons par [Fowler, 2004].

*Événement* : On considère généralement un événement comme « une occurrence de temps qui contribue à l'évolution de l'état du système » NDLR : traduit de l'Anglais [Banks et al, 1998].

Précisions encore une fois qu'un événement n'implique pas nécessairement un changement d'état, bien que ce soit souvent le cas.

Il peut y avoir différentes catégories d'événements. Certains d'entre eux sont *planifiables* et qualifiés de type *B pour « Bound to happen »* [Pidd, 1998]. Par exemple, considérons l'activité *Presser anode*, il est possible de planifier quand l'événement *Fin pressage* survient, en autant qu'on connaisse le moment où l'activité a débuté et sa durée. Notons qu'en plus d'une modification de l'état, les événements planifiables de type *B* vont généralement libérer des ressources et des entités, dans notre exemple : une presse hydraulique et une anode.

En contrepartie, il existe des événements qui ne sont pas dépendants du temps mais plutôt conditionnels à certaines activités. Donc on les qualifie de type *C* pour « *Conditional* » ou « *Cooperative* » [Pidd, 1998]. Par exemple, l'activité *Assembler composantes*, consistant à sceller une anode cuite avec une tige métallique droite, ne peut se produire que si l'événement *Début assemblage* statue que les deux composantes sont disponibles. Souvent, une activité va commencer par un événement de type *C* et se terminer par un de type *B*.

*Processus* : Il s'agit d'une séquence d'événements regroupés chronologiquement » NDLR : traduit de l'Anglais [Pidd, 1998]. Un processus est souvent utilisé pour décrire tout le cycle de vie d'entités temporaires, c'est-à-dire l'ensemble de leurs activités et de leurs événements. À ce titre, le cycle de vie des anodes d'une aluminerie, dont il est question au cours de ce mémoire, est un bel exemple de processus.

### 1.6.2 Gestion du temps d'un modèle à événements discrets

La modélisation à événements discrets implique que les changements d'états du système surviennent dans le temps [Pidd, 1998]. C'est pourquoi il importe d'une part, de conserver la valeur du temps présentement simulé, d'autre part, de mettre en œuvre un mécanisme pour avancer le temps de la simulation. La variable du modèle qui donne la valeur du temps actuellement simulé, s'appelle l'*horloge de la simulation* [Law et Kelton, 2000]. Cette valeur est exprimée en une unité de temps particulière, aussi utilisée pour caractériser certains paramètres d'entrée. En ce qui concerne

le mécanisme qui régit l'avancement de cette horloge, on s'entend généralement sur deux approches pour l'implanter : *incrément fixe* et *événement suivant.*

### 1.6.2.1 Incrément fixe

La méthode avec incrément fixe consiste à augmenter la valeur de l'horloge, par de petits intervalles de temps égaux, aussi appelés pas, d'une valeur de $\Delta t$ unités. Après chaque mise à jour de l'horloge, correspondant à un multiple de $\Delta t$, une vérification est faite, afin de statuer les événements qui sont survenus durant l'intervalle précédent. S'il y a un ou plusieurs événements, le modèle considère qu'ils se sont produits à la fin de l'intervalle et met à jour les variables d'état [Law et Kelton, 2000]. La figure ci-dessous illustre le principe.

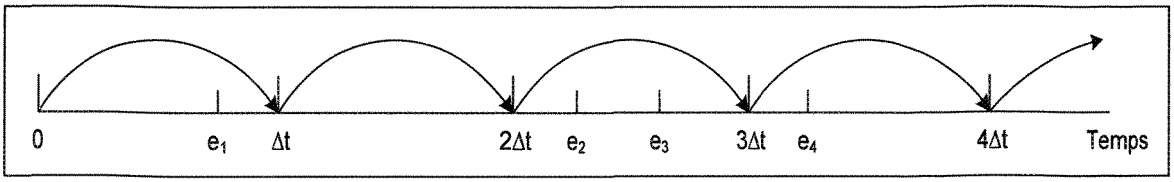


Figure 5 : avancement avec incrément fixe

Les flèches recourbées symbolisent l'avancement du temps de l'horloge. Les paramètres suivants décrivent le fonctionnement de la méthode *incrément fixe* :

*n* : le nombre d'événements qui surviennent durant la simulation.

$e_i$ : le temps où survient l'$i^{ème}$ événement, où i = [1, n]

$\Delta t$ : la durée d'un intervalle (pas) de temps.

*m* : le nombre d'intervalles (pas) de temps simulés.

$j \times \Delta t$ : le temps où l'on exécute les événements du $j^{ème}$ intervalle (pas), où j = [1, m].

En étudiant cet exemple, on constate que le premier événement survient en $e_1$, entre l'intervalle [0, $\Delta t$[. Toutefois, comme on l'a mentionné plus haut, le modèle considère uniquement cet événement après le passage de $\Delta t$ unités de temps.

Un désavantage de cette méthode réside dans le fait qu'il y a toujours des vérifications, même lorsqu'aucun événement ne s'est produit durant un intervalle. [Pidd, 1998] Par exemple, en

considérant [Δt, 2Δt[, on constate qu'aucun événement n'est survenu. Cependant, après 2Δt unités de temps, le modèle vérifie quand même les événements de l'intervalle [Δt, 2Δt[.

Pour contrer ces vérifications inutiles, on peut augmenter la valeur de Δt [Law et Kelton, 2000]. Ainsi, il y a plus de chance qu'un événement survienne à chaque intervalle. Cependant, cette augmentation risque d'occasionner une perte de réalisme. En effet, lorsque plusieurs événements surviennent dans un même intervalle, on doit choisir de façon arbitraire leur ordre d'exécution. Or, l'ordonnancement établi ne correspond pas nécessairement à la réalité.

À l'inverse, une diminution de la valeur de Δt améliore la précision mais accroît aussi la fréquence des vérifications. Le modèle devient donc beaucoup moins performant [Hoover et Perry, 1990]. Pour toutes ces raisons, on utilise surtout l'approche avec *incrément fixe*, lorsque le temps entre les événements est relativement constant et lorsqu'il y a de nombreux changements d'état [Law et Kelton, 2000].

### 1.6.2.2 Événement suivant

Avec cette méthode, l'horloge du modèle est d'abord initialisée à zéro. Puis, on estime le moment où des occurrences d'événements futurs surviennent. L'horloge est avancée jusqu'au premier événement. À ce point, l'état du modèle est mis à jour et au besoin, on modifie notre connaissance des événements futurs. Le processus se poursuit en avançant l'horloge au prochain événement et ainsi de suite. Normalement, le processus continue jusqu'à ce qu'une condition d'arrêt soit atteinte [Law et Kelton, 2000]. La figure ci-dessus illustre le principe.

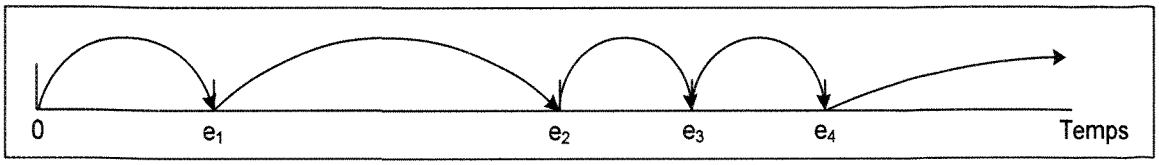


Figure 6 : avancement jusqu'à l'événement suivant

L'approche d'avancement jusqu'à l'*événement suivant* a deux avantages par rapport à l'approche avec *incrément fixe* [Pidd, 1998]. Le premier est que la durée des pas de temps s'ajuste

automatiquement, peu importe le nombre de changements d'état du système. Le fait est qu'en connaissant les événements futurs, on évite les vérifications inutiles pour les intervalles subséquents. De plus, cette méthode met en évidence les événements significatifs d'une simulation. En effet, comme la plupart des modèles discrets mettent à jour leur interface lorsque des événements se produisent, on peut clairement appréhender les principaux jalons qui contribuent à l'évolution de l'état du système.

En contrepartie, l'approche de type *événement suivant* nécessite de maintenir des informations pour contrôler le déroulement d'une simulation, par exemple une liste chaînée des événements futurs [Hoover et Perry, 1990]. De plus, l'écoulement du temps n'est pas sans à-coups, en raison des intervalles variables. Ainsi, les modèles qui implémentent cette méthode peuvent parfois apparaitre plus complexes et confus du point de vue de l'utilisateur.

Finalement, notons que [Law et Kelton, 2000] et [Pidd, 1998] considèrent la méthode *événement suivant* beaucoup plus générique. En effet, lorsque les événements d'un système surviennent à intervalle régulier, un modèle de type *événement suivant* devrait se comporter comme un modèle de type *incrément fixe*. C'est pourquoi on considère parfois cette dernière méthode comme un cas particulier de la méthode *événement suivant*.

### 1.6.3 Approche de conception d'un modèle à événements discrets

Les modèles à événements discrets sont généralement divisés en trois parties distinctes [Law et Kelton, 2000] [Pidd, 1998]: La première est le contrôleur qui régit le déroulement de la simulation. La seconde est la logique du modèle, à savoir l'expression des activités dans lesquelles les entités ou les ressources s'engagent. La dernière partie est un ensemble d'outils génériques qui permettent entre autre, l'affichage et la saisie d'informations, l'avancement du temps simulé, la génération de valeurs à partir de distributions de probabilités, ainsi que la production de rapports basés sur les mesures de performances.

Il apparaît utile que le contrôleur puisse être réutilisé pour différents types de problèmes [Pidd, 1998]. Cette notion de réutilisabilité dépend en fait de l'approche de conception employée.

Dans la littérature, on s'entend généralement sur quatre approches de conception qui nécessitent une implémentation différente du contrôleur. Ces approches sont : *la planification d'événements*, *la recherche d'activités*, *la méthode basée sur des processus* et *la méthode en trois phases* [Balci, 1988] [Banks et al, 1998]. La présente section a pour but de présenter brièvement chacune de ces approches.

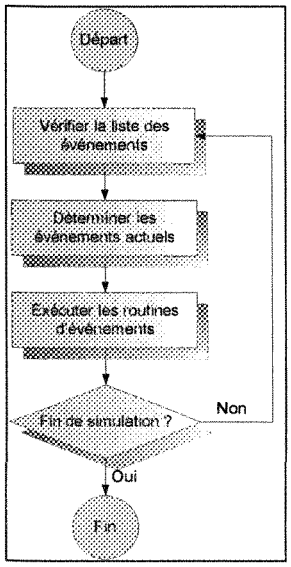


Figure 7 : planification d'événements

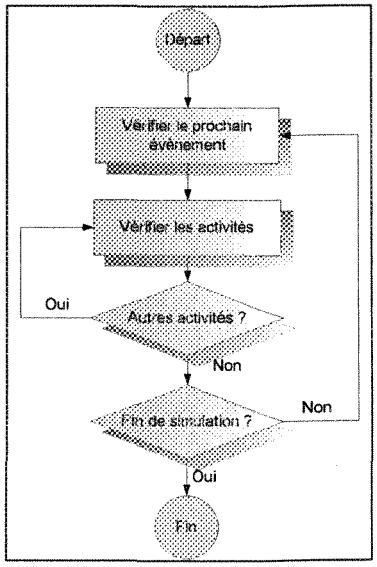


Figure 8 : recherche d'activités

### 1.6.3.1 Planification d'événements

C'est l'une des approches la plus souvent utilisée en modélisation de systèmes [Law et Kelton, 2000]. Elle fait en sorte que la logique du modèle est exprimée à partir de *routines d'événements*. Il s'agit en fait de programmes qui définissent les *conséquences* ou *actions* des événements, comme la modification des variables d'état ou la planification d'autres événements [Pidd, 1998].

Le contrôleur d'un modèle fondé sur cette approche doit avancer l'horloge de la simulation au prochain événement [Banks et al, 1998]. Ainsi, on assume qu'il maintienne une *liste d'événements*. Chaque élément de la liste contient une référence à la routine événementielle appropriée. Des éléments sont ajoutés lorsqu'un événement est planifié. Le fonctionnement d'un modèle de type

planification d'événements correspond à l'algorithme qui suit [Pidd, 1998] :

```
Répéter jusqu'à ce que la simulation soit terminée :
   1. Vérifier la liste des événements pour déterminer le prochain à se
      produire et déplacer l'horloge de la simulation au moment où il
      survient.
   2. Conserver le temps de l'horloge constant, déterminer tous les
      événements qui se produisent au temps de simulation actuel.
   3. Exécuter toutes les routines des événements actuels, afin de
      déterminer leurs conséquences.
```

Lorsqu'on modélise des systèmes complexes avec cette méthode, il est souvent difficile de considérer toutes les *conséquences* possibles au sein des routines d'événements. Par contre si l'on compare la planification d'événements à d'autres approches, comme la recherche d'activités, on constate que son exécution a tendance à être plus rapide.

### 1.6.3.2 Recherche d'activités

La recherche d'activités prend pour acquis que tous les événements sont conditionnels à des prémisses. Ainsi, lorsque les prémisses des événements sont rencontrées, des *activités* ou *actions* sont déclenchées [Banks et al, 1998]. Notons que cette approche considère même les événements *planifiables* de type *B* (« Bound to happen ») comme étant conditionnels. En fait, on leur associe une prémisse qui fait en sorte qu'ils se produisent uniquement lorsque l'horloge de la simulation a atteint un instant donné.

Le contrôleur d'un modèle qui implémente cette méthode n'a généralement pas besoin de maintenir une liste d'événements. Il suffit de connaître le moment de la prochaine activité. Toutefois, il est possible d'avoir recours à une liste. Cependant, contrairement à la *planification d'événements*, celle-ci contiendra seulement le temps de déclenchement de l'activité, comme nous n'avons pas besoin de référer à une routine d'événements [Pidd, 1998]. Il faut donc examiner tous les événements, pour déterminer ceux dont les conditions sont respectées. En résumé, le principe d'un contrôleur basé sur la recherche d'activités est le suivant :

```
Répéter jusqu'à ce que la simulation soit terminée :
   1. Vérifier quel est le prochain événement à se produire, en se basant
      sur la liste ou en parcourant tous les événements possibles, afin
```

d'obtenir celui qui a le temps minimal et déplacer l'horloge à l'instant prescrit.
2. Vérifier toutes les activités, pour déterminer celles qui peuvent être déclenchées et le cas échéant exécutez-les. Répétez cette étape, jusqu'à ce qu'il n'y ait plus d'activités à déclencher au temps simulé.

L'exécution d'un modèle conçu à l'aide de cette approche a tendance à être lente, comme on doit tester toutes les conditions des événements [Pidd, 1998]. Sa beauté réside d'avantage dans la simplicité de l'implémentation du contrôleur. Cette méthode est aussi dite à *deux phases*, comme elle est composée de deux étapes distinctes [Banks et al, 1998].

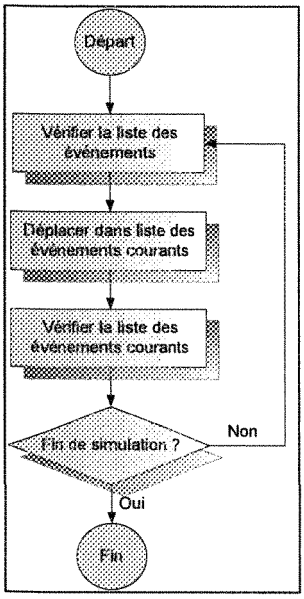


Figure 9 : méthode basée sur des processus

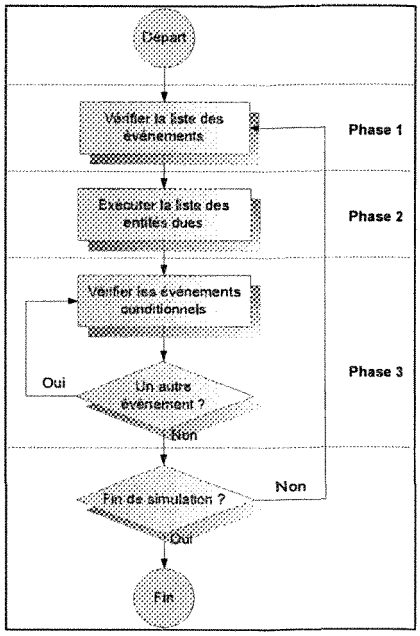


Figure 10 : méthode en trois phases

### 1.6.3.3 Méthode basée sur des processus

La méthode basés sur des processus fait en sorte que le programme émule le flot des objets à travers le système [Banks et al, 1998]. Contrairement aux autres méthodes qui atomisent le processus en événements ou en activités, celle-ci considère l'ensemble des opérations d'une classe d'entité. Ainsi, lorsqu'une entité est créée, le contrôleur lui applique le processus de sa classe. Par la suite, il se charge d'avancer l'entité à travers la séquence d'opérations du processus.

La progression d'une entité peut être interrompue par des délais *inconditionnels* ou *conditionnels* [Pidd, 1998]. Les premiers impliquent que l'entité s'arrête pour une période de temps quantifiable, par exemple à partir d'une distribution de probabilités. Un tel délai s'apparente donc à un événement de type *B* (« Bound to happen »). Quant aux délais *conditionnels*, ils font en sorte qu'il y a interruption, jusqu'à ce qu'une condition spécifique soit satisfaite. Par exemple, une anode peut demeurer en attente sur un convoyeur, jusqu'à ce qu'elle l'ait entièrement traversé et qu'un poste en aval soit disponible pour la traiter.

Cette notion d'interruption d'un processus oblige le contrôleur à maintenir deux informations : le *temps de réactivation* d'une entité, c'est-à-dire l'instant au cours duquel l'interruption prend fin, ainsi que le *point du processus* où elle se trouve. Ces informations sont enregistrées au sein de deux listes [Balci, 1988]. La première est la *liste des événements futurs*. Elle contient toutes les entités qui sont interrompues par des délais inconditionnels et dont le temps de réactivation est supérieur au temps actuellement simulé. La seconde liste est celle des *événements courants*. Elle est formée des entités interrompues par des délais inconditionnels et avec un temps de réactivation égal au temps actuel de l'horloge. De plus, elle incorpore les entités qui sont assujetties à des délais conditionnels. Ces deux listes permettent au contrôleur d'opérer comme suit :

1. Déterminer le prochain événement à partir de la *liste des événements futurs* et avancer l'horloge de la simulation à l'instant prescrit.
2. Déplacer les entités de la *liste des événements futurs*, dont le temps de réactivation est égal au nouveau temps de l'horloge, vers la *liste des événements courants*.
3. Tenter de faire progresser les entités de la liste des *événements courants* à travers leurs processus respectifs.
   3.1. Les entités qui progressent vont soit compléter leur processus ou s'interrompre, en raison d'un délai *conditionnel* ou *inconditionnel*.
   3.2. Les entités avec un délai *inconditionnel* sont déplacées vers la *liste des événements futurs*. Le contrôleur note leur prochain point de réactivation.

On ne doit pas nécessairement modéliser tous les processus d'un système. Par exemple, cela ne vaut généralement pas la peine de considérer le processus des ressources qui servent d'autres entités [Pidd, 1998]. Par contre, pour certains systèmes complexes, il importe de le faire. Or, lorsqu'on a un système avec plusieurs processus, certains d'entre eux peuvent intervenir sur

les délais *conditionnels* et *inconditionnels* d'autres processus et vice-versa. C'est ce qu'on appelle une approche de type *interaction de processus* [Banks et al, 1998]. Son implémentation est fort complexe et dépasse largement le cadre de celle qui a été décrite plus haut.

#### 1.6.3.4 Méthode en trois phases

Un contrôleur qui repose sur cette méthode maintient trois informations sur chaque entité [Pidd, 1998]. Il y a le *temps cellulaire* (de l'Anglais « Time Cell »), c'est-à-dire l'instant au cours duquel un événement planifiable de type *B* (de l'Anglais « *Bound to happen* ») se produit. Ensuite, on utilise une valeur booléenne pour établir la *disponibilité* de l'entité. Lorsqu'elle est libre, le *temps cellulaire* n'est pas représentatif. Par contre si l'entité n'est pas *disponible*, c'est qu'elle est engagée pour un événement de type *B* survenant au *temps cellulaire.* La dernière information est une référence vers le *prochain événement* planifiable de l'entité. Les trois phases de la méthode sont les suivantes [Balci, 1988] [Banks et al, 1998].

La *phase 1* débute par la recherche du prochain événement à travers la *liste des événements*. L'*horloge de la simulation* est maintenue constante au temps spécifié, jusqu'à la prochaine *phase 1*. En utilisant les trois informations décrites plus haut, le contrôleur recherche les entités *non-disponibles* avec le *temps cellulaire* minimum. Si certaines de ces entités ont un temps cellulaire égal au temps simulé, alors on les ajoute à une *liste d'entités échues* au temps actuel.

La phase 2 consiste à parcourir la *liste des entités échues*, afin d'exécuter leurs événements planifiables respectifs. Le contrôleur doit donc enlever chaque entité de la *liste des entités échues*, affecter leur état de *disponibilité* à vrai et exécuter l'événement de type *B* associé. Ceci peut faire en sorte qu'un nouvel événement soit planifié, pour l'entité traitée ou une autre.

Lors de la phase 3, le contrôleur vérifie les conditions des événements de type *C* (de l'Anglais « *Cooperative* » ou « *Conditional* ») qui sont remplies. Leurs actions sont ensuite exécutées. La vérification des événements conditionnels se poursuit, jusqu'à ce qu'il n'y ait plus aucun d'entre eux qui soient satisfaits.

La *méthode en trois phases* est moins rapide que la *planification d'événements*, car elle oblige un examen de tous les événements conditionnels, de type *C*, à chaque fois qu'un événement planifiable, de type *B*, est déclenché. Par contre, elle est beaucoup plus performante que la recherche d'activités, car les événements planifiables et conditionnels sont différenciés.

La *méthode en trois phases* s'apparente à celle *basée sur des processus*, puisqu'ils considèrent des événements équivalents : *inconditionnel* (i.e. planifiables) et *conditionnels*. Cependant, la *méthode en trois phases* se distingue, car elle exécute les événements *planifiables* (phase 2) d'abord et les *conditionnels* ensuite (phase 3). Ceci peut éviter une situation d'interblocage, où une ressource est en attente d'entités qui doivent être traitées. En effet, comme on l'a mentionné plus tôt, les événements *planifiables* libèrent généralement les entités d'une activité et les *conditionnels* contribuent à les engager [Pidd, 1998].

### 1.6.4 Architecture classique d'un modèle à événements discrets

L'approche de type *planification d'événements* est généralement utilisée, lorsqu'on conçoit un modèle à événements discrets. Or bien que de tels modèles soient appliqués à différentes catégories de systèmes, il n'en demeure pas moins qu'ils ont souvent les mêmes composantes de bases. Cette section a pour but de présenter les principaux composants d'un modèle à événements discrets axé sur une approche de type *planification d'événements*.

- *État du système* : « Il s'agit d'une collection de variables qui est utilisée afin de décrire le système à un moment particulier de la simulation ». NDLR : traduit de l'Anglais [Law et Kelton, 2000].
- *Horloge de la simulation* : « C'est une variable qui permet de connaître la valeur courante du temps simulé. » NDLR : traduit de l'Anglais [Pidd, 1998].
- *Liste ou calendrier d'événements* : Sous sa forme la plus simple, il s'agit « d'une liste chronologique contenant le temps de la prochaine occurrence pour chaque type d'événements simulés ». NDLR : traduit de l'Anglais [Law et Kelton, 2000]. Dans le cadre d'un mécanisme *événement suivant*, c'est le balayage de cette liste, à la fin de l'exécution des événements, qui

permet d'avancer l'horloge. Par exemple, considérons un bassin de refroidissement qui traite un ensemble d'anodes acheminées par un convoyeur. D'un point de vue minimaliste, il existe deux types d'événements : l'arrivée et le départ d'une anode. La liste des événements doit donc comporter deux éléments, afin de représenter le temps de la prochaine arrivée et du prochain départ. Sous sa forme plus complexe, il peut s'agir d'une liste contenant toutes les occurrences d'événements futurs. Plusieurs éléments de la liste peuvent donc être associés à un même type d'événements.

- *Compteur statistique* : « Ce sont des variables utilisées pour contenir des informations statistiques à propos de la performance du système » NDLR : traduit de l'Anglais [Law et Kelton, 2000]. Généralement, le contenu des compteurs statistiques est affiché dans l'interface du modèle, afin que l'usager puisse voir l'évolution de la simulation.
- *Routine d'initialisation* : « C'est un sous-programme dont le rôle est d'initialiser l'état du modèle au temps 0 » NDLR : traduit de l'Anglais [Law et Kelton, 2000]. Plus précisément, la routine est chargée de mettre à zéro l'horloge de la simulation, d'initialiser les variables d'état, les compteurs statistiques et la liste des événements.
- *Routine de chronométrage* : « C'est un sous-programme qui détermine le prochain événement de la liste, il est aussi responsable de l'avancement de l'horloge, lorsque cet événement se produit » NDLR : traduit de l'Anglais [Law et Kelton, 2000].
- *Routine d'événements* : « Il s'agit d'un sous-programme qui met à jour l'état du système, lorsqu'un type d'événements particulier survient. Il y a généralement une routine par type d'événements » NDLR : traduit de l'Anglais [Law et Kelton, 2000].
- *Librairie mathématique* : « C'est un ensemble de sous-programmes qui servent à générer le comportement aléatoire d'une simulation, à partir de distributions de probabilités intégrées au modèle » NDLR : traduit de l'Anglais [Law et Kelton, 2000].

- *Générateur de rapport* : « C'est un sous-programme qui calcule les mesures de performances du système, en se basant sur les compteurs statistiques. Il sert aussi à produire un rapport au terme de la simulation » NDLR : traduit de l'Anglais [Law et Kelton, 2000].
- *Contrôleur ou programme principal* : « C'est un sous-programme qui invoque la routine de chronométrage, afin de pouvoir déterminer le prochain événement et transférer le contrôle à la routine d'événements appropriée. Il peut aussi vérifier la fin de la simulation et au besoin appeler le générateur de rapport » NDLR : traduit de l'Anglais [Law et Kelton, 2000].

## 1.7 L'approche orientée objet et la modélisation

L'utilisation du paradigme objet, en modélisation de systèmes, remonte à l'avènement de Simula, vers la fin des années soixante. Il s'agit du premier langage orienté objet. À l'époque, on utilisait surtout Simula pour concevoir des modèles d'événements discrets [Meyer, 2000]. Aujourd'hui, lorsqu'on fait allusion à cette approche, on parle généralement d'OOS (acronyme de l'Anglais « Object Oriented Simulation »). La OOS est une méthode qui consiste à modéliser un système à partir de ses objets (entités, ressources, etc.) qui interagissent entre eux, à mesure que la simulation avance [Banks et al, 1998].

### 1.7.1 Caractéristiques d'un modèle orienté objet

Pour qu'un modèle soit considéré comme orienté objet, il doit comporter certaines caractéristiques [Pidd, 1998] [Zobrist et Leonard, 1997], notons : l'*abstraction*, l'*encapsulation*, l'*héritage* et le *polymorphisme*.

L'*abstraction* est un processus qui permet de structurer un problème, sous la forme de types abstraits appelés *classes*. Celles-ci contiennent des *attributs* (données) et des *méthodes* (opérations) qui définissent un ensemble d'*objets*. Plus précisément, on dit qu'un *objet* est une instance de *classe* [Meyer, 2000]. Les *attributs* servent à décrire l'état d'un objet à un moment précis. Quant aux méthodes, elles représentent les opérations qu'un objet peut accomplir [Law et

Kelton, 2000]. Dans un contexte d'OOS, on associe les méthodes aux activités et événements d'un système. Quant aux attributs, ils symbolisent l'état interne du système [Pidd, 1998].

L'*encapsulation* est un principe qui fait en sorte que les *attributs* d'un *objet* peuvent uniquement être *modifiés* par ses propres *méthodes*. Les autres objets, qu'ils aient la même définition de classe ou non, peuvent seulement lire ses *attributs*. Les objets agissent donc comme des entités totalement indépendantes grâce à cette propriété [Pidd, 1998].

La capacité de définir une nouvelle *classe* à partir d'une autre est ce qu'on appelle l'*héritage*. Ce comportement évite de constamment réinventer la roue, puisqu'on peut concevoir une nouvelle classe qui descend d'une *classe* ancêtre prédéfinie. Cette nouvelle *classe* incorpore généralement des caractéristiques additionnelles [Banks et al, 1998] [Zobrist et Leonard, 1997].

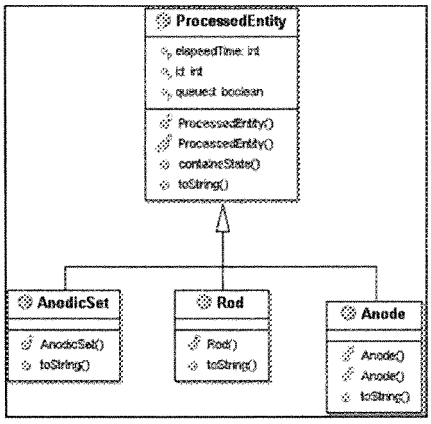


Figure 11 : exemple d'héritage et de polymorphisme

Finalement, la notion de *polymorphisme* fait en sorte que des objets, ayant une même classe ancêtre et des méthodes du même nom, adoptent un comportement différent, lorsqu'on invoque leurs méthodes homonymiques respectives [Law et Kelton, 2000]. Voici un exemple simple d'héritage et de polymorphisme dans le cadre du cycle de vie des anodes. Supposons une classe ancêtre qui représente les entités traitées par le système (*ProcessedEntity*). Les classes *Anode*, *AnodicSet* (Ensemble anodique) et *Rod* (Tige) héritent de la classe ancêtre *ProcessedEntity*. Ces classes ont une méthode *toString()*. Celle-ci retourne une chaîne de caractères qui donne l'état de l'objet. Cependant, cette méthode a un comportement polymorphe, car le format des informations

retournées va être particulier à chaque classe. La représentation UML [Fowler, 2004] d'une telle structure est illustrée ci-dessus.

### 1.7.2 Modèle orienté objet, cadre et cadriciel

L'idée de modélisation orientée objet est fort intuitive, puisqu'il est facile d'imaginer un système comme un amalgame d'objets [Banks et al, 1998]. Ainsi, on peut concevoir les différentes entités et ressources du cycle de vie des anodes, comme un ensemble d'objets : tour à pâte, bassin, vibrocompacteur, presse, table de transfert, fours, entrepôt, atelier de scellement, convoyeurs, anodes, tiges, ensembles anodiques, etc.

L'approche orientée objet permet d'accroître l'extensibilité et la modularité du modèle, en décentralisant chacune de ses portions, grâce aux notions d'abstraction et d'encapsulation. Ainsi, chaque classe agit comme une entité indépendante. Ceci implique que les changements sur l'une d'elles minimisent les répercussions sur les autres [Meyer, 2000]. Pour les même raisons, cette approche favorise également la réutilisabilité du modèle. En effet, comme les classes d'objets agissent comme des entités autonomes, on peut les réutiliser pour plusieurs domaines d'application connexes [Law et Kelton, 2000]. Par exemple si le contrôleur d'un modèle est encapsulé au sein d'un ensemble de classes, on peut l'utiliser plusieurs fois.

La notion d'héritage accroît aussi la réutilisabilité et la modularité du modèle [Zobrist et Leonard, 1997]. En effet, cette caractéristique permet d'élaborer une hiérarchie de classes. Celle-ci est formée à son sommet par les classes ancêtres qui représentent les objets les plus abstraits du système. Par la suite, on ajoute à cette hiérarchie des classes spécifiques, qui héritent des caractéristiques des classes abstraites et ainsi de suite. Cette notion d'héritage hiérarchique peut être représentée sous la forme d'une structure arborescente, comme celle du diagramme UML de la Figure 11.

Un regroupement de classes, d'un même niveau d'abstraction, est ce qu'on appelle un *cadre* (traduction de l'Anglais « *frame* ») [Banks et al, 1998]. Les *cadres* sont des librairies de classes utilisées lors de la mise en œuvre d'un modèle orienté objet. Contrairement à la classe, le *cadre*

n'est pas nécessairement une structure qui est implémentée dans un langage de programmation. Un *cadre* peut tout simplement être une représentation conceptuelle d'un niveau d'abstraction.

Lorsqu'on modélise à un plus haut niveau, il peut être avantageux de regrouper les *cadres* en *cadriciel* (traduction de l'Anglais « *framework* »). Un *cadriciel* est défini ainsi :

> Un ensemble de classes abstraites collaborant entre elles, pour faciliter la création de tout ou d'une partie d'un système logiciel. Un cadriciel fournit un guide architectural en partitionnant le domaine visé en classes abstraites et en définissant les responsabilités de chacune, ainsi que les collaborations entre classes. [Cadriciel]

Le cadriciel est un peu un coffre à outils générique qui permet d'adresser des problèmes similaires. La principale différence entre une hiérarchie de classes et un cadriciel est que la première est basée sur des relations d'héritage, alors que la seconde, s'appuie aussi bien sur des relations de composition ou de collaboration entre des classes [Banks et al, 1998] [Fowler, 2004].

### 1.7.3 Modèle basé sur des composants logiciels

En génie logiciel, on définit un composant logiciel comme : « un élément d'un système qui offre un service prédéfini et qui est capable de communiquer avec d'autres composants. » NDLR : traduit de l'Anglais [Software Component]. D'un point de vue minimaliste, un composant doit posséder les caractéristiques suivantes : usage multiple, non-spécifique à un contexte particulier, formé d'autres composants, encapsulé (voir la section 1.7.1), indépendant du déploiement et des différentes versions [Szyperski, 1999].

Les composants sont souvent des objets conçus à partir d'une spécification formelle. Les trois spécifications les plus connues [Young et Tag, 2001] sont : COM (« Component Object Model ») de Microsoft [COM], CORBA (« Common Object Request Broker Architecture ») de l'OMG [CORBA] et JavaBean de Sun Microsystems [JavaBeans].

| | COM | CORBA | JavaBeans |
|---|---|---|---|
| **Plate-forme** | Windows uniquement | Plusieurs dont AIX, Linux, Solaris et Windows | Plusieurs dont Linux Solaris et Windows |
| **Principaux langages de programmation** | Plusieurs dont C, C++, Delphi, Visual Basic | Plusieurs dont C, C++, Delphi, Java | Java uniquement |
| **Origine de la spécification** | Échange d'information entre applications | Environnement distribué | Machine virtuelle Java |

Tableau 1 : spécifications pour concevoir des composants

Les composants basés sur une spécification possèdent généralement une interface définie dans un langage particulier appelé IDL (de l'Anglais « Interface Description Language ») [Software Component]. Cette interface leur permet de se comporter de façon autonome et d'interagir entre eux, quelque soit la plate-forme ou le langage utilisé [Szyperski, 1999].

Pour bien des auteurs, la notion de composant est indissociable de l'approche orientée-objet. Selon les tenants de ce principe : « Un composant est ensemble de classes fortement liées qui agissent comme une unité » [Batory et O'Maley, 1992]. Alors que d'autres affirment que les composants sont des briques logicielles préfabriquées, qui peuvent être assemblées ensembles, un peu comme les composants d'un circuit électronique ou d'un bâtiment. On parle donc d'une notion qui va au-delà de l'approche orienté objet. Cette nouvelle approche est dite *orienté composante* [Software Component]. Dans le contexte de ce mémoire, nous ne comptons pas discourir sur ces aspects sémantiques. Toutefois, nous avons intégré la présente discussion, au sein d'une section sur les modèles orienté objets, car les composants sont souvent implémentés à partir de classes d'objets. Celles-ci sont généralement immutables et capturent un état initial par défaut [Szyperski, 1999].

Les composants sont aussi utilisés en modélisation. D'une part, parce qu'ils facilitent la division d'un système complexe en sous-systèmes élémentaires. Ensuite, parce qu'ils promeuvent la réutilisation logicielle et la simplification de la mise en œuvre du modèle [Pidd et al, 1999]. Ainsi, il devient possible de prendre des composants, répertoriés au sein d'une librairie ou d'un cadriciel et de les intégrer à l'environnement de travail du modélisateur. De tels composants peuvent représenter les objets d'un modèle (ressources, entités, files, etc.), afficher des informations à

l'écran, générer des rapports, effectuer des animations ou retourner une valeur issue d'une distribution de probabilités. Généralement, en modélisation de systèmes, on dit que les composants d'un cadriciel ont les particularités suivantes : capacité d'échanger des informations à l'aide d'un système de messagerie, simplicité de configuration, à partir d'un ensemble de paramètres et capacité d'interconnexion à des fins d'assemblage, grâce à une interface standardisée [Praehofer et al, 1999].

### 1.7.4 Modélisation et le formalisme DEVS

L'un des pionniers en matière de modélisation basée sur des composants est le docteur Bernard Ziegler de l'Université de l'Arizona. En 1976, il inventa une spécification formelle appelée DEVS (de l'Anglais « Discrete Event System Specification ») [Zeigler, 1995]. DEVS est en fait un formalisme théorique qui sert à spécifier un système discret et ce, au même titre que les équations différentielles peuvent le faire pour un système continu. Ainsi, on peut concevoir des modèles hiérarchiques et modulaires, qui implémentent le formalisme DEVS, à l'aide d'un langage de programmation et de standards logiciels [Zeigler et Sarjoughian, 2005]. DEVS permet de modéliser beaucoup plus facilement et efficacement un système, en le divisant en *composant* de *modèles,* en spécifiant le couplage entre chacun d'eux et en fournissant un cadre spécifique qui facilite la réutilisation des modèles [Young et Tag, 2001].

Il y a deux sortes de modèles du point de vue du formalisme DEVS [Zeigler et Sarjoughian, 2005]. Le premier est le modèle *atomique* ou de *base* qui permet de décrire le comportement dynamique d'un *composant de modèles.* Ainsi, on doit percevoir un *composant de modèles* comme une entité qui interagit avec son environnement, par le biais de *ports entrants* et *sortants.* Plus spécifiquement, lorsque des événements externes au *composant de modèles* sont reçus, à travers ses *ports entrants*, celui-ci détermine une réponse appropriée. De même, lorsque des événements internes contribuent à la modification de son état, ceux-ci doivent être transmis aux *ports sortants*, pour que les autres *composants de modèles* soient notifiés du changement. Le *modèle atomique* (M.A.) du formalisme DEVS possède la structure suivante [Zeigler, 1995] :

M.A. = $\langle X, S, Y, \delta_{int}, \delta_{ext}, \lambda, ta \rangle$, où

X est l'ensemble des ports pour chaque événement entrant

S est l'ensemble des états

Y est l'ensemble des ports pour chaque événement sortant

$\delta_{int} : S \rightarrow S$ est la fonction de transition interne

$\delta_{ext} : Q \times X \rightarrow S$ est la fonction de transition externe, où

$Q = \{(s,e) \mid s \in S, 0 \leq e \leq ta(s)\}$ est appelé l'ensemble de l'état total

e est le temps écoulé depuis la dernière transition

$\lambda : S \rightarrow Y$ est la fonction de sortie

$ta : S \rightarrow R^{+}_{0, \infty}$ est la fonction d'avancement du temps

Le tuple ci-dessus a la signification suivante : à un moment quelconque le système est dans un état $s \in S$. Lorsqu'aucun événement externe ne survient, le système demeure dans un état s pour un temps de $ta(s)$. Lorsque le temps d'inactivité est écoulé (c'est-à-dire $e = ta(s)$), le système retourne la valeur $\lambda(s)$ et change d'état pour $\delta_{int}(s)$. Si un événement externe $x \in X$ survient avant le temps d'expiration, c'est-à-dire le système est dans l'état total $(s,e)$ avec $e \leq ta(s)$, alors le système change son état pour $\delta_{ext}(s,e,x)$. Peu importe le type de transition, interne ou externe, une fois qu'il est survenu, le système tombe dans un nouvel état noté s' pendant $ta(s')$. Le processus se poursuit ainsi par la suite.

Le second type de modèles considéré par DEVS est dit *couplé* [Zeigler et Sarjoughian, 2005]. Il permet de définir comment connecter des *composants de modèles*, pour former un nouveau modèle. Ce modèle peut être utilisé à son tour, comme un *composant* d'un *modèle couplé*, donnant ainsi naissance à une hiérarchie de modèles [Young et Tag, 2001]. Le *modèle couplé* (C.M.) de DEVS est défini comme suit :

C.M. = $\langle X, Y, M, EIC, EOC, IC, SELECT \rangle$, où

X est l'ensemble des ports pour les événements entrants

Y est l'ensemble des ports pour les événements sortants

M est l'ensemble de tous les composants de modèles de DEVS

$EIC \subseteq X \times \cup_i X_i$ est la relation de couplage des entrées externes qui sert à connecter les ports entrants d'un composant, avec ceux du modèle couplé.

$EOC \subseteq \cup_i Y_i \times Y$ est la relation de couplage des sorties externes qui sert à connecter les ports sortants d'un composant, avec ceux du modèle couplé.

$IC \subseteq \cup_i X_i \times \cup_i Y_i$ est la relation de couplage interne qui sert à connecter les ports sortants d'un composant aux ports entrants d'un autre.

$SELECT\ 2^M - \phi \rightarrow M$ est une fonction qui choisit un modèle, lorsque 2 modèles ou plus sont couplés simultanément.

Depuis une dizaine d'années, la spécification DEVS a été utilisée en conjonction avec des standards de composants logiciels. Notons les travaux de [Young et Tag, 2001] avec COM, ceux de [Zeigler et al, 1999] avec CORBA ou encore ceux, de [Praehofer et al, 1999] avec JavaBeans. Ces recherches démontrent la possibilité et les avantages d'une intégration de DEVS avec des technologies logicielles qui comportent des caractéristiques modulaires, multiplateformes et multi-langages.

## 1.8 Systèmes visuels de modélisation interactive

Jusqu'ici, il a été question d'une approche de modélisation classique qui consiste à concevoir un modèle basé sur des événements discrets et à le mettre en œuvre, sous la forme d'un programme qui simule des scénarios d'opération. Cependant, il existe des logiciels de modélisation commerciaux qui sont utilisés pour effectuer de telles tâches. Ces systèmes, appelés VIMS (de l'Anglais « Visual Interactive Modeling System »), permettent d'élaborer un modèle dans un environnement visuel et interactif et d'exécuter des simulations dans ce même environnement [Pidd, 1998].

Notre recherche n'a pas pour but de faire l'apologie ni le procès de ces systèmes. Cependant, nous comptons montrer qu'il est possible de modéliser un système dont le fonctionnement repose sur des connaissances, en utilisant une approche alternative, dont il est question au cours du chapitre 3. En attendant, la présente section vise à présenter brièvement, le fonctionnement et les caractéristiques des VIMS.

### 1.8.1 Fonctionnement d'un VIMS

Habituellement, un modèle conçu à partir d'un VIMS débute avec un écran vierge, auquel le modélisateur ajoute des icônes qui représentent les principaux objets du système. Les icônes sont ensuite liés ensembles, pour former un réseau qui capture les interactions logiques des objets du système [Law et Kelton, 2000]. Globalement, il existe deux approches pour modéliser un tel réseau.

La première approche est dite *basée sur les machines* [Pidd, 1998], puisque chaque icône représente une machine, c'est-à-dire une ressource ou un groupe de ressources. Quant aux liens, ils montrent l'itinéraire d'entités passives qui parcourent le système. Par exemple, dans le cadre de la modélisation du cycle de vie des anodes, une ressource équivaut à un poste du système (tour à pâte, bassin de refroidissement, fours, entrepôt, etc.) qui traite des anodes, c'est-à-dire les entités passives. Ce type de réseau convient quand les entités cheminent à travers un processus complexe ou lorsque les ressources effectuent plusieurs activités.

La seconde approche consiste à élaborer un réseau *basé sur des tâches* ou *activités* [Pidd, 1998]. Chaque icône du modèle représente une activité qui implique une ou plusieurs entités ou encore, une ou plusieurs ressources. Ainsi, les tâches sont liées logiquement pour montrer leur degré de dépendance. Cette approche convient lorsque l'aiguillage des entités est relativement simple ou lorsqu'il y a coopération entre plusieurs ressources

### 1.8.2 Caractéristiques d'un VIMS

Il existe plusieurs caractéristiques souhaitables, lorsqu'il vient le temps de sélectionner un VIMS. En voici quelques-unes selon [Law et Kelton, 2000] :

- Flexibilité de modélisation : Ce critère doit permettre de modéliser un système, peu importe son degré de complexité. Ainsi, il doit être entre autre possible : de définir et changer les attributs d'entités ainsi que les variables globales du modèle, d'avoir recours à des fonctions mathématiques et de créer de nouvelles structures de modélisation.
- Facilité d'utilisation et d'apprentissage : Une interface graphique conviviale et fonctionnelle est de rigueur avec un VIMS. Par exemple, il importe de pouvoir utiliser des constructions visuelles, comme des icônes ou des blocs, qui sont ni trop primitifs ni trop complexes.
- Modélisation hiérarchique : Cette caractéristique est importante afin de pouvoir réutiliser des parties d'un modèle et les combiner en structures hiérarchiques.
- Facilité de débogage : Cet aspect permet de suivre une entité à travers le modèle, de voir l'état du modèle, lorsqu'un événement survient et d'affecter automatiquement une valeur à un attribut ou une variable.
- Vitesse d'exécution : Cette caractéristique intervient principalement durant la phase de simulation.
- Possibilité d'exportation ou d'importation : Il est avantageux d'importer (ou d'exporter) des données d'un (ou vers un) tableur ou une base de données.
- Simulation de scénarios basés sur un paramètre : En changeant un paramètre particulier (comme par exemple, le nombre maximal d'entités d'une file), on peut voir immédiatement son impact, notamment sur certaines mesures de performances (par exemple, le temps moyen d'une entité dans le système).
- Combinaison discret et continu : Pour certains modèles basés sur des événements discrets, il peut être avantageux de simuler certaines parties en tant qu'un phénomène continu.
- Routines externes : Parfois, la logique d'un modèle est si complexe qu'il importe de la mettre en œuvre dans un langage de programmation, d'où l'intérêt de pouvoir référencer des routines externes.

- État initial : Lorsqu'on conçoit un modèle d'un système manufacturier, il est souvent utile de pouvoir l'initialiser dans un état non-vide, par exemple où toutes les ressources traitent des entités et où les files sont à demies pleines.
- Sauvegarde de l'état terminal : Au terme de la simulation, il est souvent utile de préserver l'état final, afin de pouvoir redémarrer à nouveau, à partir de celui-ci.
- Coûts : En plus des coûts associés au VIMS, il importe de considérer ceux attribuables au support technique, à l'installation, à la maintenance, aux mises à jour, au matériel informatique et aux logiciels additionnels.
- Configuration recommandée : Cette considération découle d'aspects tels que : la compatibilité avec un ou plusieurs système(s) d'exploitation(s), la quantité de mémoire vive ou la puissance du processeur qui sont recommandées.
- Animation et graphiques dynamiques : Les objets clés du système doivent idéalement être représentés par des icônes qui changent de position, de couleur et/ou de forme à mesure que la simulation avance. Il existe deux types d'animation : *concurrente* et *post-traitée*. L'animation *concurrente* fait en sorte que l'affichage est rendu en même temps que se déroule la simulation. Ceci peut ralentir l'exécution. Le second type implique que les changements d'état sont préservés sur disque et utilisés pour un affichage subséquent. Lorsque l'animation est *concurrente*, il est utile de pouvoir interrompre la simulation et la redémarrer plus tard.
- Capacités statistiques : Le VIMS doit normalement posséder un générateur de nombres aléatoires. Lorsqu'on l'adjoint à une distribution de probabilité, on peut obtenir des valeurs pour certaines activités ou phénomènes associées au modèle. Si une distribution de probabilité ne peut être utilisée, il est souhaitable de recourir à une distribution empirique, basée sur les données du système. La plupart des VIMS devraient aussi permettre d'établir un intervalle de confiance, pour une mesure de performance, comme le temps passé dans le système [Baillargeon, 1990].

- Support aux usagers et la documentation : Cette caractéristique comprend le support téléphonique ou par courriels, le guide de l'usager, l'aide en ligne avec des exemples, la formation, les démos, les mises à jours, les conférences, etc.
- Génération de rapports et de graphiques : Les rapports doivent être fournis pour les principales mesures de performance du système. Il est souhaitable de pouvoir les configurer. Des graphiques statiques doivent être aussi fournis sous plusieurs formes : histogramme, avec barres, pointe de tarte, courbes, etc.

### 1.8.3 Arena un exemple de VIMS

Au moment de l'écriture de ce mémoire, Arena de Rockwell Automation [Arena1] est un des VIMS les plus populaires dans le monde de la modélisation de systèmes. Arena est abondamment utilisé pour modéliser les processus de l'industrie, comme les installations manufacturières. Il permet d'analyser la performance actuelle et les améliorations qui peuvent être apportées aux processus [Arena2]. Arena est disponible en plusieurs éditions : *Basic Edition*, *Professional Edition*, *Factory Analyzer*, *Contact Center* et *3D Player* [Arena1]. Cependant, la présente discussion est principalement orientée sur la première édition.

Avec Arena, l'utilisateur construit un modèle expérimental, en plaçant des modules (c'est-à-dire des boîtes de différentes formes) qui représentent les processus ou la logique d'un système. Les connecteurs qui les relient spécifient le flot des entités [Arena2]. Les modules sont configurés à l'aide de boîtes de dialogue. Un modèle peut avoir un nombre illimité de niveaux de hiérarchie. On peut afficher des animations en deux dimensions et des graphiques dynamiques ou statiques (histogrammes, avec barres ou courbes). Arena dispose d'importantes capacités statistiques. Ainsi, il est possible de modéliser des files d'attentes complexes, par exemple à partir de processus non-stationnaires de Poisson (pour les systèmes dont l'arrivée des entités se fait à un taux variable) et de construire des intervalles de confiance pour les mesures de performances [Law et Kelton, 2000].

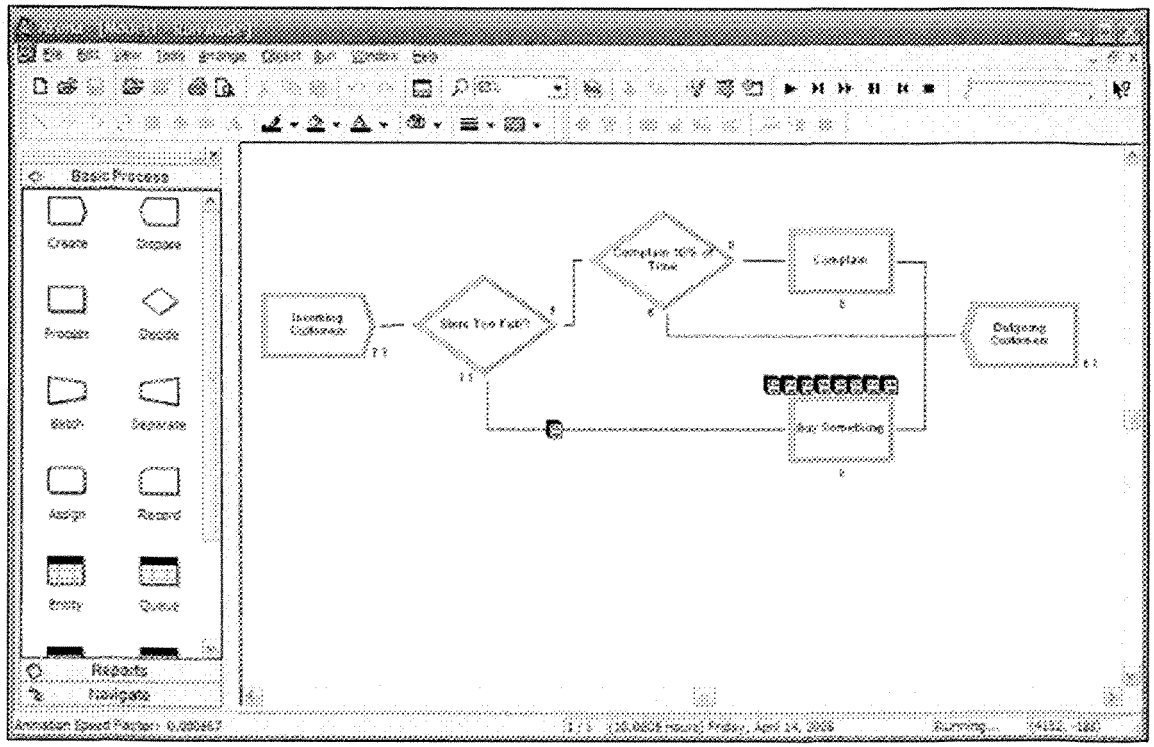

Figure 12 : un modèle conçu avec Arena 9.0

Arena est uniquement compatible avec la famille de systèmes d'exploitation Windows [Arena3]. Conséquemment, il s'intègre aisément aux technologies de Microsoft. Si on a besoin d'algorithmes plus spécifiques, il est possible de recourir à des routines VBA (Visual Basic for Applications) [VBA]. Arena supporte également l'importation de diagrammes Visio, l'importation/exportation de feuilles de calcul Excel ou de bases de données Access et la génération de rapports, à l'aide de Crystal Report. Il est aussi possible d'ajouter des composants COM (ActiveX) [COM] à l'environnement de modélisation [Arena2].

### 1.8.4 La modélisation classique et les VIMS

Un VIMS, comme Arena, possède plusieurs fonctionnalités qui simplifient le processus de modélisation d'un système, comparativement à une approche classique qui consiste à mettre en œuvre le modèle, sous la forme d'un programme informatique. D'ailleurs selon [Law et Kelton, 2000], un VIMS diminue le temps d'implémentation du modèle, il fournit les éléments à la base de

sa conception (voir la section 1.6.4), en plus de faciliter sa maintenance, sa modification et la détection des erreurs.

En contrepartie, le fonctionnement d'un VIMS n'est pas nécessairement connu par les spécialistes en modélisation. Dans ce contexte, l'apprentissage et la formation d'un ou plusieurs spécialistes sur différents VIMS complexifie le processus de modélisation. Néanmoins, la plupart des spécialistes connaissent au moins un langage de programmation [Law et Kelton, 2000]. Or, il existe des cadriciels de simulation, mis en œuvres dans différentes langages de programmation, qui possèdent des fonctionnalités similaires aux VIMS [L'Écuyer et al., 2002]. Certains de ces cadriciels sont mêmes gratuits. Ainsi, on peut se questionner sur la pertinence des VIMS, lorsqu'on sait qu'un cadriciel ou une librairie de simulation peut convenir dans bien des cas.

Finalement, mentionnons que les VIMS sont beaucoup moins flexibles [Law et Kelton, 2000]. D'ailleurs, avec Arena, on utilise souvent des modules de code VBA [Arena2], pour répondre à des besoins qui ne peuvent être mis en œuvres avec ses fonctionnalités de base. C'est d'ailleurs le cas de notre problématique, puisque une partie du système à modéliser s'appuie sur la notion de *connaissance heuristique* [Friedman-Hill, 2003] qui nécessite de recourir à des notions d'intelligence artificielle. À ce titre, l'utilisation de méthodes d'intelligence artificielle, dans le cadre de la modélisation d'un système, fait l'objet de la discussion du chapitre suivant.