INTRODUCTION
À LA
SIMULATION

Par Pascal Cantot

Ingénieur des Études et Techniques d'Armement

Promotion ENSIETA 1988-92

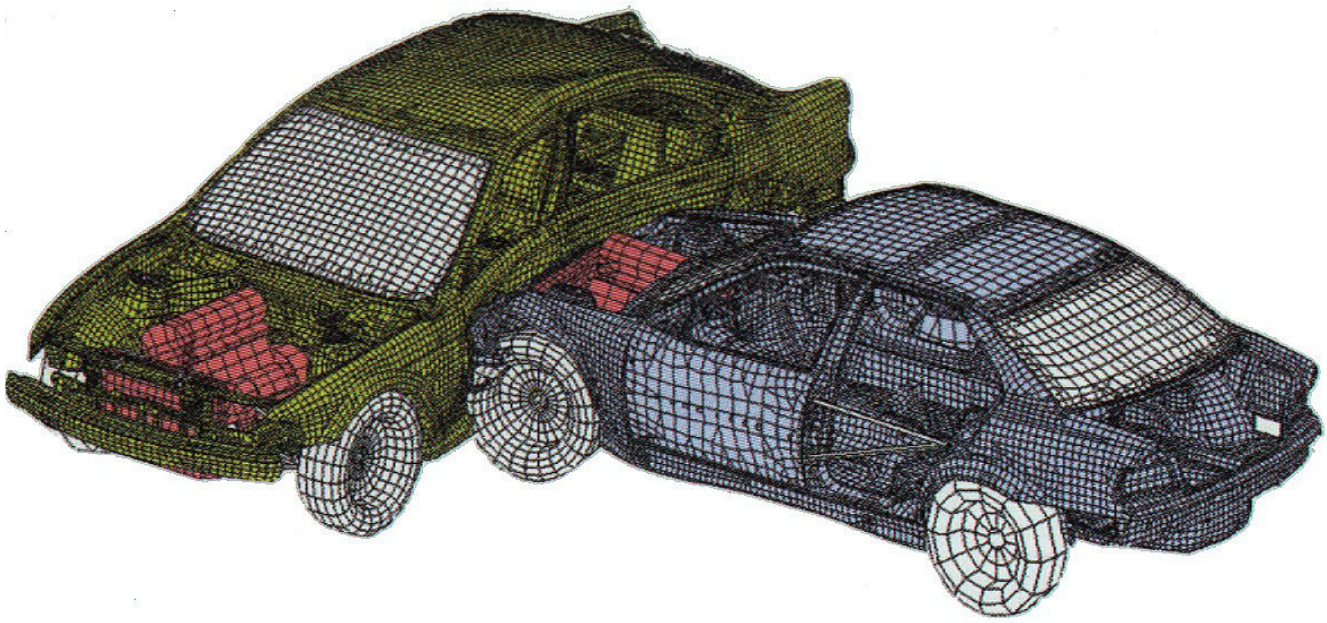


Image de couverture : Simulation de choc latéral entre deux véhicules BMW, réalisée avec le logiciel PAM-Crash (Programs in Applied Mechanics) de la société française ESI, sur calculateur SGI (cliché SGI).

RÉSUMÉ

Ce cours est destiné aux élèves de l'option « électronique » de l'École Nationale Supérieure des Ingénieurs des Études et Techniques d'Armement.

Il présente les enjeux et les techniques utilisés en modélisation et simulation. Il n'a pas l'intention d'être exhaustif, et de toute façon le domaine de la simulation est trop vaste et trop diversifié pour qu'un ouvrage, quel qu'il soit, puisse avoir une telle prétention.

En revanche, il permettra à son lecteur, qu'il soit élève de l'ENSIETA ou non, de découvrir ce domaine en pleine expansion et d'acquérir les bases nécessaires, qu'il pourra ensuite compléter par les spécificités de son métier.

Les premiers chapitres donneront quelques définitions et notions de base, et démontreront l'intérêt de la simulation (et ses limitations !). Puis, le décor posé, et avant de plonger dans la technique pure, des exemples concrets d'utilisation de différents types de simulations seront donnés.

Ensuite, les techniques de simulation proprement dites seront abordées : processus de modélisation, outils mathématiques (modélisation du hasard), génie logiciel (qualité, structures d'accueil).

Enfin, une partie du voile sera levée sur les technologies appelées à représenter l'avenir de la simulation, notamment les environnements synthétiques et la réalité virtuelle, dont nous pouvons voir les prémisses dès aujourd'hui.

En annexe, un **index**, un **glossaire** et une description des **acteurs** de la simulation pourront servir de référence au lecteur. Pour terminer, une **bibliographie** permettra au lecteur d'aller plus loin dans son apprentissage des techniques de modélisation.

REMERCIEMENTS

Je tiens à exprimer ma gratitude pour les personnes qui, directement ou indirectement, m'ont aidé à réaliser ce cours, et en particulier :

- M. Jean-Louis Igarza, directeur technique du Modeling and Simulation Coordination Office de l'OTAN, à qui je dois de travailler aujourd'hui dans ce domaine passionnant qu'est la simulation, et qui m'a tant appris au cours de 5 années de collaboration.
- M. Christian Sautereau, chef du département Méthodes de Simulation du Centre d'Analyse de Défense, « père » de la structure d'accueil ESCADRE.
- L'ICA Daniel Windheiser, chef du département d'ingénierie des Systèmes Complexes du Service Technique des Technologies Communes, qui a bien voulu m'autoriser à faire ce cours.
- M. Patrick Dropsy, Responsable du Domaine d'Expertise simulation à la Direction des moyens et Centres d'Essais (DCE).
- L'IETA Lui Kam, du Centre Technique d'Arcueil (DCE/CTA), relecteur attentif.
- M. Stéphane Di Vito, Scientifique du Contingent 99-00 (et un précieux collaborateur pendant neuf longs mois !).
- Mme Françoise Montigny, ingénieur à l'ONERA, qui a bien voulu assurer la partie du cours sur la simulation scientifique.
- Mes élèves de l'ENSIETA, dont les questions ont contribué à faire évoluer le cours.
- Toutes les autres personnes qui m'ont fourni des éléments (images, définitions, documents…) ou prodigué leurs conseils…

NOTE

Ce cours, plutôt conceptuel, a été réalisé presque entièrement sur mon temps libre, avec essentiellement des moyens personnels. Aussi, je n'ai pu être aussi exhaustif et précis que je l'aurais souhaité. Toutefois, je pense y avoir mis l'essentiel, tant pour les ingénieurs du secteur civils que les ingénieurs de la DGA.

Il s'agit de la première version de ce cours qui, je l'espère, s'actualisera, s'enrichira et… se corrigera au fil des années.

Merci de bien vouloir me faire parvenir vos remarques, critiques (constructives) et informations :

IETA Pascal CANTOT
26 Bd Victor
00457 ARMÉES
messagerie DGA : *Pascal Cantot*
Internet : *cantotp@wanadoo.fr*

AVANT-PROPOS

C'est avec grand plaisir que j'ai accepté de consacrer une longue série de soirées et de week-ends à la conception de ce cours, qui, j'en suis convaincu, vous servira dans votre vie professionnelle future, quelque soit le domaine dans lequel vous travaillerez, car le virtuel est maintenant omniprésent.

Je suis un IETA issu de la promotion 1992 (date de sortie) de l'ENSIETA, spécialité construction de matériels terrestres, option électronique. Passionné d'informatique et pratiquant les jeux de guerre (wargames) depuis le lycée j'ai sauté sur un poste au service informatique du Centre d'Analyse de Défense, un des hauts lieux de la simulation de défense française, et grand consommateur de jeux de guerre. Estimant devoir compléter mes connaissances, je suis parti un an à Toulouse afin d'obtenir un mastère en systèmes informatiques (SSI) à l'ENSICA.

Une fois en poste au CAD, j'ai commencé une vie d'ingénieur système, Unix d'abord puis NT. Incidemment, on fit appel à moi pour des problèmes réseaux, notamment les questions de connexion de simulations distribuées. L'établissement d'une liaison entre deux jeux de guerre (wargames) JANUS entre la France et les États-Unis fut mon premier contact concret avec le monde de la simulation. Et rapidement je décidai qu'il s'agissait là d'un créneau passionnant, et promu à un grand avenir.

C'est donc tout naturellement que trois ans plus tard je quittai le service informatique et les problèmes d'imprimantes qui n'impriment pas pour me retrouver « chargé de la simulation distribuée » au département « méthodes de simulation » du CAD. Le choix était fait, j'étais parti pour un moment dans le monde de la simulation.

Et trois ans plus tard j'ai récidivé : je suis maintenant chargé des affaires de simulation au département d'ingénierie des systèmes complexes du service technique des technologies communes (STTC) de la délégation générale pour l'armement, qui a la responsabilité au sein de la DGA de la simulation pour l'acquisition ainsi que des méthodes et outils communs pour la simulation.

Mon travail consiste, entre autres, à superviser des programmes de recherche dans le domaine de la simulation, à gérer des projets en coopération avec (beaucoup) d'autres pays et à mettre en place une politique commune de simulation au sein de la DGA, notamment dans le cadre de la simulation pour l'acquisition, terme recouvrant un concept d'intégration de la simulation au cycle d'acquisition des systèmes. Vaste programme…

J'ai accepté de faire ce cours pour deux raisons principales. D'abord, je pense qu'il est du devoir des « anciens » de contribuer à la formation de leur futurs collègues potentiels, et, plus généralement, de communiquer leur savoir et leur expérience aux générations suivantes. Ensuite, comme vous avez dû le comprendre, j'ai la conviction que le virtuel va être (ou est déjà ?) un des piliers techniques de notre métier, qu'on soit civil ou militaire, ingénieur ou commercial, électronicien ou mécanicien, et qu'il doit faire partie de la culture de tout ingénieur moderne.

TABLE DES MATIERES

TABLE DES FIGURES

1. INTRODUCTION

1.1 Les enjeux : la simulation, outil de la complexité

Le monde dans lequel nous vivons s'est engagé dans une course effrénée à la complexité.

Complexité des comportements d'abord, car aujourd'hui les qualifications techniques requises pour exercer un métier sont de plus en plus élevées, non seulement en raison de l'environnement technologique, mais aussi en raison des exigences de qualité actuelles. Ainsi, l'ouvrier tourneur-fraiseur peut-il se retrouver aux commandes d'une machine numérique dont la débauche de fonctionnalités n'a d'égal que son coût, et qu'il lui faudra apprendre à maîtriser, parfois pour usiner des pièces dans des conditions très particulières (atmosphère neutre, haute précision, etc.). Ainsi également le soldat, qui du statut de « porte-fusil » est devenu un « système d'arme » à part entière, participant à des conflits d'un genre nouveau, tels que ceux nécessités par les missions de maintien de la paix, et devant réagir à des vitesses inconnues jusqu'alors. Enfin, le prix des munitions explose (!), ce qui n'encourage pas à la multiplication des entraînements sur le terrain. Sachant qu'une bombe « intelligente » peut valoir dans les 100.000$, je vous laisse imaginer le coût de la formation d'un pilote de bombardier… Les méthodes de formation classiques ne sont donc plus satisfaisantes.

Complexité des systèmes, ensuite, car le temps où l'on pouvait traiter individuellement les composants d'un système est révolu : on ne considère plus isolément la tourelle d'un char, mais le système d'armes complet, avec son châssis, son équipage (qu'il faut former), ses munitions, son carburant, ses communications, sa maintenance, etc. On pourra même considérer des niveaux supérieurs, tels que les systèmes de forces, où l'on prendra en compte les interactions avec les autres unités pour la réalisation d'une mission. Ceci est également vrai pour une simple voiture particulière, mélange subtil de mécanique, d'électronique et d'informatique, et dont la conception a tenu compte des questions de fabrication, de marketing, de maintenance et même de recyclage en fin de vie. De plus, il faut gagner en compétitivité et donc réduire les coûts, tout en améliorant la qualité

Systèmes complexes

*Qu'est-ce qu'un système complexe ? En 1969, H.A. Simon définit un système complexe comme étant « un système fait d'un grand nombre d'éléments qui interagissent de façon complexe ». Une définition certainement plus claire est celle de Jean-Louis Le Moigne ([MOIG90]): « La complexité d'un système est caractérisée par deux facteurs : le nombre des éléments qui le constitue d'une part et le nombre des inter-*relations *d'autre part ».*

Globalement, on jugera donc la complexité d'un système non seulement en fonction du nombre de ses composants mais aussi en fonction des relations et dépendances entre ces composants. Un produit ayant une forte proportion de logiciel devient ainsi rapidement complexe.

Une propriété importante des systèmes complexes est que, lorsqu'on intègre ses sous-systèmes, on se retrouve souvent face à des propriétés émergentes non prévues initialement. Un système complexe est donc bien plus que le simple regroupement de ses composantes.

On trouvera dans les systèmes complexes des systèmes de systèmes tels que les navires (porte-avions Charles de Gaulle) ou les systèmes ayant une forte composante humaine (systèmes d'information et de commandement).

et les fonctionnalités offertes. C'est la quadrature du cercle pour l'ingénieur !

Pour gérer cette complexité, il est nécessaire de recourir à des méthodes et outils particuliers. La simulation est l'un des plus importants de ces outils.

Ce cours a pour objet de vous faire expliquer ce qu'est un modèle ou une simulation, comment on la construit et dans quels contextes on l'emploie. Nous verrons quels sont les apports du virtuel, mais aussi quelles sont ses… limitations.

1.2 Historique

- **VI^{ème} siècle av. J.-C.** : En Chine, le général et philosophe de la guerre Sun Tzu crée le premier jeu de stratégie, appelé ***Wei Hai***. Ce jeu consistait à capturer des territoires, tout comme le jeu de go, apparu vers 1200 av. J.-C. Vers 500 av. J.-C., apparaît en Inde le ***Chaturanga***, que les perses feront évoluer vers le jeu d'échecs, dont les pièces représentent les différentes catégories de militaires (infanterie, cavalerie, élephants…).

Figure 1: Blaise Pascal

- **30 après J.-C.** : Les premières tables à sable font leur apparition pour l'enseignement et le développement de tactiques.
- **1642** : Machine à calculer de Blaise Pascal. De nombreux savants (dont Leibniz) fabriqueront par la suite des machines similaires en améliorant régulièrement les performances. Toutefois, les performances de toutes ces machines étaient très limitées par des contraintes mécaniques.
- **1664** : Christopher Weikhmann, un allemand, invente un jeu dérivé des échecs, le *Koenigspiel*, avec une trentaine de pièces plus représentatives des unités de l'époque.
- **XV^{ème} siècle – XVIII^{ème} siècle** : Les savants prennent conscience que le monde est gouverné par des lois physiques : la philosophie fait peu à peu place à la physique dans l'explication du fonctionnement de l'univers. Parallèlement, les mathématiques se développent. Le calcul intégral, le calcul matriciel, l'algèbre, etc., donnent aux scientifiques les outils de base pour leurs études. Copernic, Galilée, Descartes, Kepler, Newton, Euler, et bien d'autres, ont apportés des éléments essentiels pour mettre en équations des phénomènes physiques tels que le mouvement des planètes, la trajectoire d'un projectile ou l'écoulement d'un fluide.
- **1780 :** Un allemand encore, Helvig invente un jeu doté de 120 pièces (fantassins, cavaliers et artillerie) par joueur, se déplaçant sur un plateau de 1666 cases, dont la couleur était représentative de la nature du terrain. Ce jeu eut un certain succès en Allemagne, Autriche, France et Italie, où il servait à inculquer des principes tactiques de base à la jeune noblesse.

- **1811** : le Baron Von Reisswitz, conseiller en matière de guerre auprès de la cour prussienne, invente un jeu de guerre appellé ***Kriegspiel***. Les princes Wilhelm et Friedrich furent tellement impressionnés qu'ils le firent adopter par l'armée prussienne. Le Kriegspiel se jouait sur une table couverte de sable et utilisait des figurines en bois pour représenter les différentes unités. Des règles régissaient les mouvements, les effets du terrain et les résultats des combats étaient calculés à l'aide de tables de résolution. Les *wargames* à figurines utilisés de nos jours par les amateurs du genre utilisent exactement les mêmes principes.

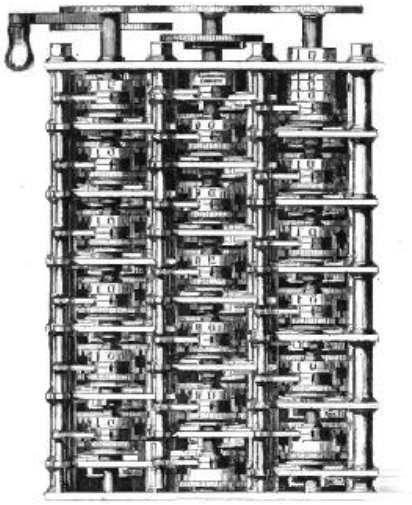


Figure 2: Machine à différences de Babbage (détail)

- **1833** : Babbage invente une « machine analytique », qui intègre la plupart des éléments d'un ordinateur moderne : unité de calcul, mémoire, registres, unité de contrôle, mémoire de « masse » (lecteur de cartes perforées). Bien que la construction fut un échec, il fut prouvé par la suite que l'invention fonctionnait.
- **1870** : le Prince Wilhelm estime que le Kriegspiel a joué un rôle significatif dans la victoire prussienne. À la même époque, le jeu était également utilisé en Italie, en Russie, aux USA et au Japon, notamment pour prédire le résultat des batailles. Néanmoins, les résultats n'était pas toujours exacts. Ainsi, avant la première guerre mondiale, les allemands utilisèrent une variante appelée *Free Kriegspiel* pour prédire l'issue de la guerre en Europe, mais ne mirent pas en évidence plusieurs actions décisives des alliés qui survinrent par la suite. Ce jeu fut également utilisé par le III^{ème} Reich et par les japonais durant la seconde guerre mondiale. Notons que les résultats des simulations ne furent pas tous pris en compte par les états-majors. Ainsi, le Kriegspiel montra que la bataille de Midway coûterait aux japonais deux porte-avions. Ceux-ci persistèrent et en perdirent quatre, ce qui entraîna par la suite leur défaite, devenu inéluctable.
- **1910** : « Tonneau » de la société Antoinette, premier appareil d'entraînement au pilotage. Une nacelle sur un plateau et quelques câbles permettaient d'enseigner à l'élève des techniques de base. Aux USA, le Sanders Teacher utilisait un avion au sol[1].

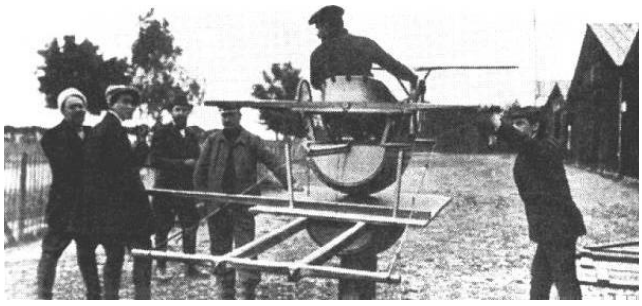


Figure 3: Tonneau Antoinette

- **1916** : Publication par F.W. Lanchester d'une théorie dynamique de combat. Les équations d'attrition de Lanchester sont encore largement utilisées dans les jeux de guerre.
- **1928** : Banc d'entraînement au sol de Lucien Rougerie pour l'initiation au pilotage.
- **1927** : Simulateur électro-pneumatique d'Edward Link (fabricant d'orgues et passionné d'aviation), reproduisant les réactions de l'avion. Les mouvements de la cabine étaient assurés par quatre soufflets pour le roulis et un moteur électrique pour

[1] Pour un historique plus détaillé des simulateurs de vol, voir [MOOR98].

le lacet. L'instructeur pouvait suivre les évolutions de l'élève grâce à une réplique du tableau de bord et à une table traçante. L'instructeur pouvait également paramétrer l'influence du vent. Les premiers exemplaires opérationnels furent livrés à l'US Air Force en 1934. L'entraîneur Link est considéré comme le premier vrai simulateur piloté.

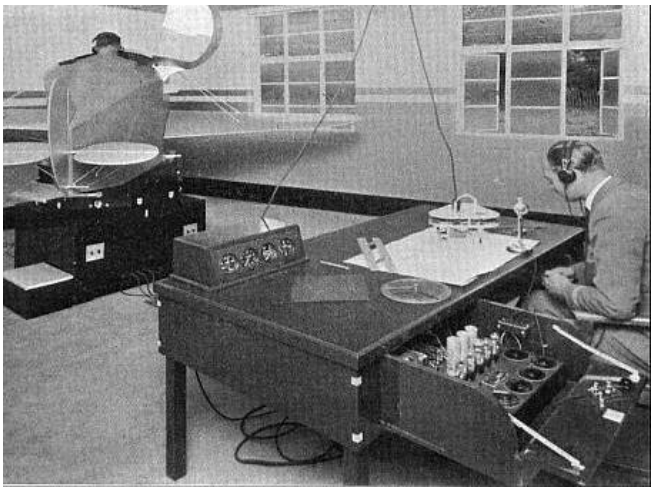


Figure 4: Entraîneur Link

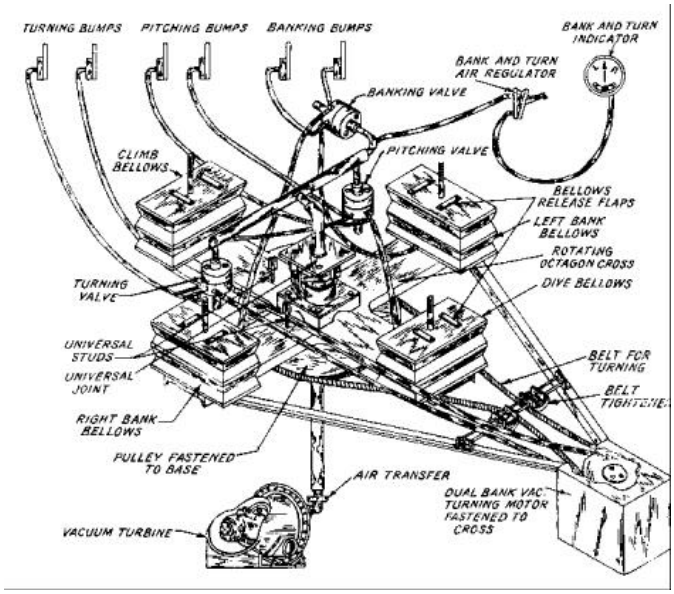


Figure 5: Le système pneumatique de l'entraîneur Link

- **1940** : Calculateur électronique ABC de John Atanasoff (université de l'Iowa). Bien qu'il comporte une mémoire et des circuits logiques, cet ancêtre innovant n'était pas programmable. Atanasoff fut néanmoins reconnu comme l'inventeur de l'ordinateur.
- **1940-45** : Les pilotes de l'US Navy s'entraînent grâce à des simulateurs de vol « Blue Box ».
- **1945-46** : Premiers ordinateurs modernes (ENIAC, Harvard MARK II). Utilisation pour le calcul de tables de tir et la mise au point des armes nucléaires. L'ENIAC avec ses 20 000 tubes pouvait effectuer une addition en 200 µs et une multiplication en 2 800 µs (soit de l'ordre de 1 000 opérations par secondes). Lors de sa mise en service en 1946, il tombait en panne toutes les 6 heures…

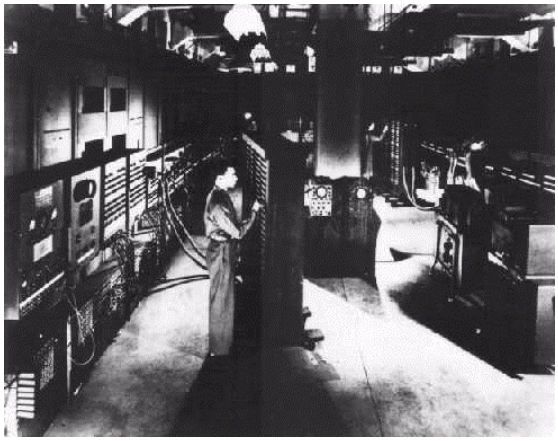


Figure 6: Ordinateur ENIAC (BRL)

- **1948** : Simulateur électronique de B377 pour la PanAm.

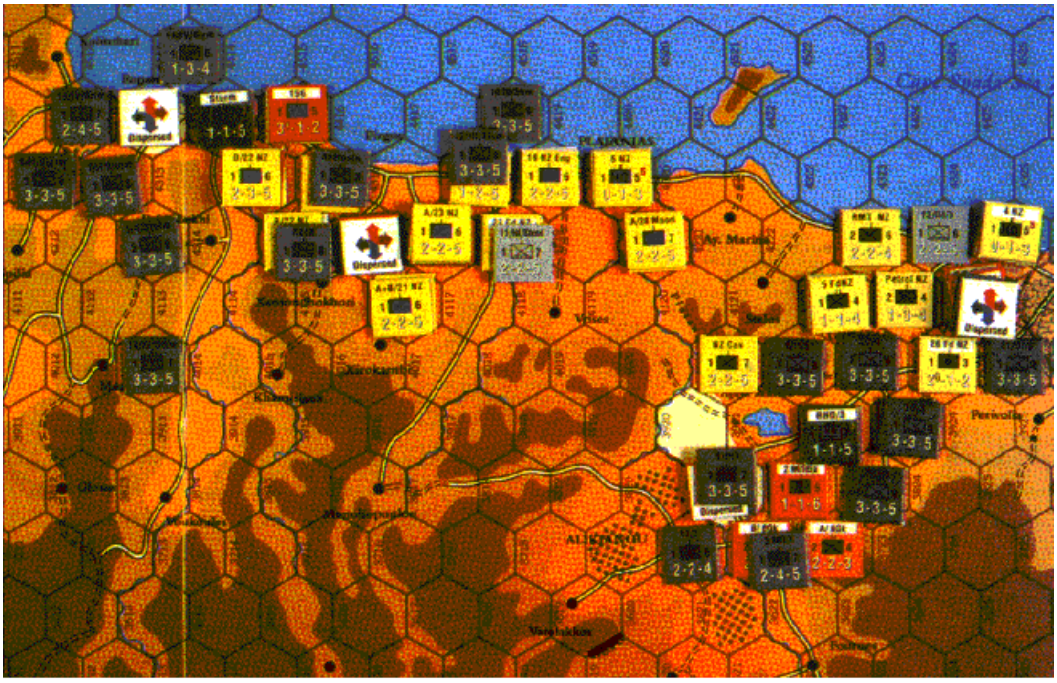


Figure 7: Wargame moderne "Operation Mercury" de GMT, simulation de l'assaut Allemand sur la Crête en 1941 (revue Vae Victis)

- **1952** : Charles S. Roberts invente un wargame utilisant un terrain partagé à l'aide d'une grille rectangulaire. Des morceaux de carton portaient les symboles des unités (infanterie, artillerie, blindés…), et des règles régissaient les déplacements et les combats. Ce jeu appelé Tactics connut un grand succès et Roberts fonda la compagnie des jeux Avalon Hill. Le wargame commercial sur plateau était né, et connu une grande expansion dans les années 70 : Panzerblitz, d'Avalon Hill, se vendit ainsi à plus de 200 000 exemplaires.
- **1951-56** : Au LRBA[2], le calculateur analogique SABA (Simulateur Aérodynamique et Ballistique de l'Armement) servira à mettre au point le premier missile sol-air français, le PARCA. Les calculateurs analogiques tomberont totalement en désuétude dans les années 70 au profit du numérique.
- **1957** : Premier langage de programmation symbolique universel, le FORTRAN. Orienté calcul scientifique, il sera extrêmement populaire dans le monde de la simulation, et le reste encore de nos jours pour les simulations scientifiques.
- **1958** : Système de défense aérienne SAGE (*Semi-Automatic Ground Environment*). SAGE, l'un des plus ambitieux projets militaires du siècle (comparable au projet Manhattan qui permit la réalisation de la première bombe atomique) fut le premier réseau informatique d'envergure. Destiné à la détection d'une éventuelle attaque soviétique, il reliait des centaines de stations radar à travers l'Amérique du Nord. Il utilisait des ordinateurs Whirlwind II, qui comprenaient 55 000 tubes, 175 000 diodes et 13 000 transistors. Entre autres innovations, le système utilisait le premier affichage graphique interactif, avec un « pistolet » électronique qui permettait à

[2] Laboratoire de Recherches Balistiques et Aérodynamiques de l'armement de Vernon, créé en 1946 par la DEFA (Direction des Études et Fabrications de l'Armement, ancêtre de la DGA française. Comprenant une soixantaine de savants allemands, il expertisera les V1 et V2 et sera à l'origine des premiers missiles français.

l'opérateur de pointer un mobile sur l'écran. SAGE fonctionnera jusqu'en 1984, et inspirera de nombreux développements ultérieurs, dont le système de réservation aérienne SABRE et par là même de son dérivé… le système de réservation SOCRATE de la SNCF.

Figure 8: Console SAGE avec pistolet électronique

- **1958** : langage algorithmique ALGOL. Ce langage qui connaîtra d'autres versions (Algol60, Algol68) est important car il inspira des langages très populaires tels que Pascal et Ada, ce dernier étant très employé par les militaires et l'industrie aérospatiale à partir du milieu des années 80.
- **1961** : Simulateur de Mirage de la société LMT, utilisant des transistors.
- **1963** : William Fetter crée un modèle « fil de fer » d'un avion pour étudier ses décollages et atterrissages. Il crée aussi « First Man », un pilote virtuel destiné à des études d'ergonomie.
- **1964-69** : Lancement d'ARPANET, qui deviendra l'embryon du réseau global Internet.
- **1965** : La société Digital Equipment commercialise le mini-ordinateur PDP-8. En raison d'un coût et d'une taille raisonnables, il sera diffusé à des milliers d'exemplaires et marquera une étape vers la démocratisation de l'informatique et des moyens de calcul scientifique.

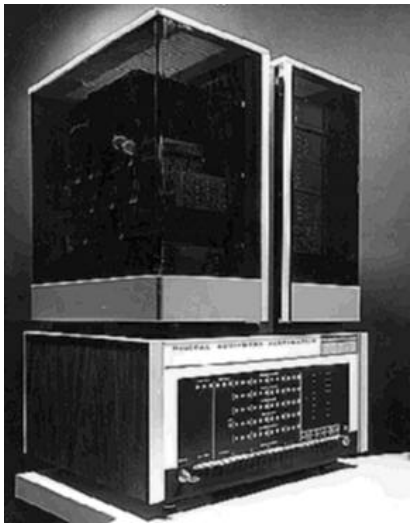


Figure 9: Mini-ordinateur PDP-8

- **1967** : General Electric réalise un simulateur d'avion en couleurs. Le graphisme et l'animation sont schématiques, et il requiert des moyens matériels énormes pour tourner.
- **1967** : Le premier langage orienté objet, SIMULA, est une extension d'Algol-60 (ancêtre des langages structurés tels que Pascal, Ada et C), dédiée à la simulation par événements discrets.
- **1971** : une jeune société américaine, Intel, intègre sur une puce de silicium les principaux constituants de l'unité de traitement centrale (CPU) d'un ordinateur. C'est la sortie de l'Intel 4004, le premier microprocesseur.
- **1972** : Novoview, générateur d'image d'Evans & Sutherland (graphisme fil de fer monochrome).
- **1972** : Langage C, par Dennis Ritchie. Bien qu'initialement conçu pour la programmation système, ce langage aura un succès universel. Sa version orientée objet, C++, est actuellement l'un des langages les plus populaires.
- **1974** : Ed Catmull invente le *Z-buffering*, technique améliorant la gestion de la visualisation 3D, notamment le calcul des parties cachées.

- **1976** : Jim Blinn développe le *texture mapping*, technique permettant d'améliorer considérablement le réalisme des images de synthèse en plaquant une image bitmap (i.e. une photo) sur la surface d'un objet en 3D, donnant ainsi l'illusion que l'objet est beaucoup plus détaillé qu'il n'est en réalité. Par exemple, en plaçant la photo d'un immeuble réel sur un simple parallélépipède, on pourra voir les fenêtres et tous les détails architecturaux de l'immeuble, en ne gérant qu'un volume 3D simple.
- **1979** : Le premier tableur, Visicalc, sera en grande partie à l'origine du succès des premiers micro-ordinateurs tels que l'Apple II, puis, quelques années plus tard, l'IBM PC. Cette feuille de calcul électronique permet d'effectuer des simulations simples (par itérations) et d'analyser des données, notamment dans le monde de la finance. Si Visicalc a disparu depuis longtemps du marché, des successeurs, tels que Microsoft Excel, sont devenus des outils bureautiques de base.
- **1980-81** : Premiers systèmes expérimentaux de réalité virtuelle à la NASA (VIEW, VIVED).
- **1987** : SIMNET, réseau expérimental de simulation distribuée.

Figure 10: L'arrivée des micro-ordinateurs met la simulation numérique à la portée de tous

- **1987** : Lancement du projet 2851 du DoD américain, pour la standardisation des bases de données d'environnement, qui conduira notamment au standard SIF.
- **1989** : Protocole DIS pour l'interopérabilité des simulations. Après l'expérience acquise avec SIMNET, le gouvernement américain lance des travaux sur un protocole dédié. La procédure, ouverte, vise à en faire un standard international, ce qui aboutira avec la norme IEEE 1278 en 1995. Orienté temps réel et de bas niveau, DIS n'était toutefois pas universel, ce qui conduira le département de la défense US à revoir sa stratégie et à imposer HLA.
- **1990** : ALSP, protocole de simulation distribuée pour les simulations constructives. En parallèle à DIS, une partie de la communauté américaine de la simulation constructive finança la conception par la Mitre Corporation d'un standard dédié aux simulations « non temps réel », qui sera lui aussi remplacé par HLA quelques années plus tard.
- **1991-92** : Guerre du Golfe. Pour la première fois, la simulation est utilisée massivement pour la planification des opérations, la répétition des missions et l'analyse après action. Les américains utilisent SIMNET pour entraîner les équipages de char et d'hélicoptère.
- **1992** : Environnement de réalité virtuelle CAVE de l'université de l'Illinois.
- **1994** : En France, deux simulateurs de Mirage 2000, l'un à Orange, l'autre à Cambrai, sont interconnectés grâce au protocole DIS et à une liaison RNIS[3]. L'année suivante, c'est au tour d'un wargame (JANUS) à être connecté entre la

[3] Réseau Numérique à Intégration de Services, version entièrement numérique du téléphone, commercialisé par France Telecom sous l'appellation *Numéris*. RNIS est une liaison non-permanente basée sur un ou plusieurs canaux à 64 kbit/s.

France et les USA via le réseau Internet. Enfin, en 1996, la France participe à une fédération expérimentale ALSP OTAN distribuée sur plusieurs pays d'Europe.

- **1995** : HLA, standard de simulation distribuée de haut niveau, comprenant une partie méthodologique fortement orientée vers la réutilisation des modèles, marquant une étape décisive vers la globalisation des environnements synthétiques.
- **1997** : SEDRIS, standard d'interopérabilité des bases de données d'environnement, applique des principes proches de HLA aux environnements des simulations.
- **1998** : NATO M&S Master Plan. Les 16 nations de l'OTAN se mettent d'accord sur un schéma directeur sur la modélisation et la simulation, consacrant notamment HLA comme le standard de prédilection.
- **1998** : HLA est adopté par l'EMA pour les simulations de niveau opérationnel et stratégique. La DGA, si elle ne le rend pas obligatoire (pour l'instant) le recommande fortement.
- **1999-2000** : Ces deux années verront la mise en place de plusieurs groupes de travail et schémas directeurs sur la simulation au sein des ministères de la défense français et étrangers. De larges projets communs sont lancés à l'OTAN et à l'UEO. Longtemps dispersée, la simulation s'organise et se renforce…
- **2000** : HLA est standardisé par l'IEEE (standard IEEE 1516).

2. GENERALITES

2.1 Définitions

Le lecteur pourra constater l'importance donnée aux définitions dans ce cours. En effet, la simulation s'appuie sur des concepts et un vocabulaire particuliers, qui sont importants à connaître. Comme il n'existe pas, en France, de définitions normalisées[4], j'ai tenté à chaque fois de trouver une définition correspondant au sens habituellement donné à chaque terme. Nombre de ces définitions sont dérivées des documents [DGA97], [IEEE94] et [DOD97]

2.1.1 Système

Ensemble de moyens et d'éléments matériels, logiciels, naturels ou artificiels, agencés de manière à satisfaire un objectif donné.

Exemples : réseau de communication, système d'armes, base de données…

Un système sera caractérisé :

- Par les **composants** qui le constituent
- Par les **relations** entre ces composants
- Par son environnement
- Par les contraintes qu'il subit
- Par ses évolutions au cours du temps

Prenons par exemple le système « boule de billard qui tombe ». Ce système est composé d'une bille, et son environnement est constitué par l'atmosphère et la Terre. Il subit de la part de cet environnement une force l'attirant vers le bas (gravité) et une force opposée due à la résistance de l'air.

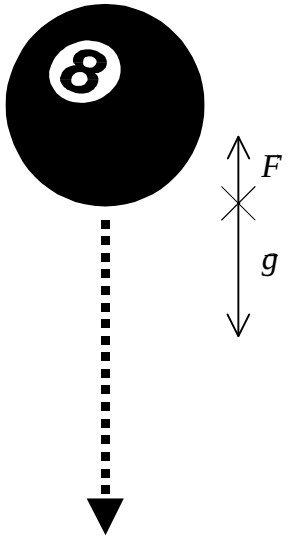


Figure 11: Boule de billard

2.1.2 Modèle

Le dictionnaire-guide sur la simulation [DGA97] donne la définition suivante :

Un modèle est une construction théorique qui représente la réalité…

[4] Un groupe de travail AFNOR vient d'être créé pour remédier à cette carence.

La définition « officielle » du département de la défense américain (DoD), donnée dans [DOD95], est :

> **On appelle modèle toute représentation (abstraction) physique, mathématique ou autrement logique d'un système, d'une entité, d'un processus ou d'un phénomène physique [construite pour un objectif donné].**

L'IEEE, organisme civil de normalisation, nous en donne deux autres :

> 1. **Une approximation, représentation ou idéalisation de certains aspects de la structure, du comportement, du fonctionnement ou des autres caractéristiques d'un processus, concept ou système du monde réel.**
> 2. **Une représentation mathématique ou logique d'un système ou du comportement d'un système au cours du temps.**

⇨ Dans ce cours, nous ne considèrerons essentiellement que des modèles relevant du 2° de la définition IEEE.

La **modélisation** est l'opération par laquelle le modèle est construit à partir de la réalité, par exemple par la mise en équations mathématiques du système. Cette modélisation peut nécessiter plusieurs étapes, telles que la décomposition d'un système trop complexe en sous-systèmes plus simples.

Reprenons l'exemple de la boule de billard qu'on lâche dans l'air, d'une altitude z_0. On s'intéresse au mouvement du centre de gravité de la boule, en ne considérant d'autres influences que la gravité terrestre (supposée constante ici).

Les variables d'état sont l'altitude z et la vitesse v.

Les conditions initiales à $t = 0$ sont l'altitude initiale z_0 et la vitesse initiale v_0.

Les équation cinématiques de la boule sont :

- Accélération : $a(t) = g$
- Vitesse : $v(t) = \int_0^t a(t)dt = gt + v_0$

D'où on en déduit : $z(t) = \int_0^t v(t)dt$

Soit : $\boxed{z(t) = \frac{1}{2}gt^2 + v_0 t + z_0}$

L'implémentation informatique de ce modèle donnerait par exemple (langage C) :

```
/* Modèle cinématique (version simple) */

float calcul_vitesse(float t)
{
      return g*t + Vo;
}

float calcul_position(float t)
{
      return g*t*t/2 + Vo*t + Zo;
}
```

Lors de l'établissement d'un modèle, il est essentiel d'avoir un sens critique du travail effectué. Ici, il est clair que ce modèle a un **domaine de validité** restreint : tant que la

vitesse est faible, il est correct, mais si la vitesse devient grande, il faudra faire intervenir la friction de l'air, sous la forme d'un terme *f(v)*, par exemple : $f(v) = -a.v$. Sans compter que nous avons supposé que le vent était nul, ce qui est rarement le cas dans la réalité (les artilleurs et les footballeurs en savent quelque chose !). Et si la hauteur de chute est importante, il faudra tenir compte de la variation de la gravité avec l'altitude (paramètre $g(x)$), faire intervenir les forces de Coriolis (d'où une déviation horizontale), etc.

En ne tenant compte que des frottements, le mouvement de notre boule devient quelque chose comme :

$$z(t) = \left(-\frac{mv_0}{a} + \frac{m^2}{a^2} g \right) \cdot \left(1 - e^{-\frac{a}{m}t} \right) - \frac{mg}{a} t$$

En fait, c'est *presque* la même chose qu'auparavant, l'importance donné au « presque » variant suivant les besoins de fidélité de la modélisation !

2.1.3 Simulation

La taxonomie des termes de simulation, nous l'avons vu, n'est pas vraiment uniforme suivant les domaines, comme nous avons pu le voir précédemment. Pour notre part, nous prendrons la définition la plus simple d'une simulation :

La simulation est l'implémentation d'un modèle dans le temps

« Simuler » consiste donc à faire vivre un ou plusieurs modèles en les faisant évoluer dans le temps.

L'action de simuler peut aussi faire référence au processus complet qui, à partir d'un besoin de modélisation, permet d'obtenir les résultats escomptés. Ce processus comporte trois grandes étapes :

1. La conception et le développement du modèle
2. L'exécution du modèle
3. L'analyse de l'exécution

La définition ci-dessus désigne *stricto sensu* la seconde étape, ce qui peut s'avérer trompeur, car le travail essentiel est la conception du modèle, de la qualité duquel dépendront les résultats, et l'interprétation de ces résultats, laquelle n'est pas toujours évidente : quand un calculateur « crache » plusieurs milliers de pages de chiffres en guise de sortie, cela peut prendre des semaines de travail pour en tirer la substantifique moelle…

Le terme « simulation » est utilisé par extension, et souvent à tort, pour qualifier l'ensemble des techniques destinées à l'évaluation ou l'analyse de systèmes, ou à l'entraînement, et basées sur des simulations correspondant à la définition ci-dessus.

Les anglo-saxons utilisent généralement l'expression « modeling and simulation » (M&S en abrégé) pour parler de ces techniques.

Pour éviter les confusions, on utilise souvent le terme plus précis d' « application de simulation » pour désigner le logiciel implémentant un modèle.

Nous verrons par la suite différentes typologies de modèles et de simulations.

Pour reprendre notre exemple de la boule qui tombe, le programme de simulation pourrait être le suivant :

```
/* boule_qui_tombe.c */

#include <stdio.h>

/* Constantes physiques */
const  float g = -9.81;       /* accélération de la pesanteur (m/s²) */

/* Conditions initiales */
float Vo = 0.0;               /* vitesse initiale (m/s) */
float Zo = 0.0;               /* altitude initiale (m) */

/* Modèle cinématique (version simple) */

float calcul_vitesse(float t)
{
  return g*t + Vo;
}

float calcul_position(float t)
{
  return g*t*t/2 + Vo*t + Zo;
}

/* "moteur de simulation" et "interface utilisateur" */

int main()
{
   /* Paramètres de fonctionnement du moteur de simulation */
   float Pas= 1.0;    /* pas de temps (s) */

   /* Variables extérieures */
   float t;           /* temps de la simulation */

   /* Variables d'état */
   float v;           /* vitesse */
   float z;           /* altitude */

   puts("Simulation de la chute d'une boule de billard:\n");

   printf("Altitude et vitesse initiales ? ");
   scanf("%f %f", &Zo, &Vo);
   printf("Pas d'intégration ? ");
   scanf("%f", &Pas);

   printf("Altitude initiale = %f m/s, Vitesse initiale = %f m\n", Zo, Vo);

   t = 0.0;
   do
   {
      v = calcul_vitesse(t);
      z = calcul_position(t);
      printf("t = %7.3f s --> z = %7.2f m, v = %7.2f m/s\n", t, z, v);
      t = t + Pas;
   }
   while (z > 0.0);

   puts("Paf!");

   return 0;
}
```

L'exécution de la simulation donne le résultat suivant, pour une boule lâchée depuis le haut de la Tour Eiffel :

```
Simulation de la chute d'une boule de billard:

Altitude et vitesse initiales ? 320.0 0.0
Pas d'intégration ? 1.0

Altitude initiale = 320.000000 m, Vitesse initiale = 0.000000 m/s
```

```
t =   0.000 s --> z =  320.00 m, v =    0.00 m/s
t =   1.000 s --> z =  315.10 m, v =   -9.81 m/s
t =   2.000 s --> z =  300.38 m, v =  -19.62 m/s
t =   3.000 s --> z =  275.86 m, v =  -29.43 m/s
t =   4.000 s --> z =  241.52 m, v =  -39.24 m/s
t =   5.000 s --> z =  197.38 m, v =  -49.05 m/s
t =   6.000 s --> z =  143.42 m, v =  -58.86 m/s
t =   7.000 s --> z =   79.65 m, v =  -68.67 m/s
t =   8.000 s --> z =    6.08 m, v =  -78.48 m/s
t =   9.000 s --> z =  -77.31 m, v =  -88.29 m/s
Paf!
```

Ceci représente le résultat brut de l'exécution de la simulation. Ici, nous n'avons que quelques données, mais imaginons une simulation du vol d'une fusée avec des dizaines de variables d'état et des milliers de valeurs !

C'est tout l'intérêt de la phase d'exploitation qui ne doit pas être sous-évaluée. Dans cette phase, nous pourrions ici étudier le « profil » de la chute de la boule et évaluer l'instant exact de l'impact, et la vitesse à ce moment précis. Le calcul analytique est simple, mais là encore il faut s'imaginer un système beaucoup plus complexe.

2.1.4 Composants d'une simulation

En pratique, une simulation requiert au moins les éléments suivants :

- Le modèle du système simulé, bien entendu (ex. : un avion de combat)
- Les données paramétrant le modèle (accélération maximale, armement, etc.)
- Le modèle décrivant l'environnement du système (terrain, météo, émissions électromagnétiques…)
- Les données d'environnement (ex. en météo : température, taux d'humidité, …)
- Le **moteur de simulation**, qui va gérer le temps et les évolutions du systèmes.
- Le **scénario**, qui décrit la situation initiale du système (par exemple la vitesse et la position initiale) et les événements prévisibles qui surviendront au cours du temps (par exemple l'arrivée d'une formation ennemie à tel endroit, à telle date).

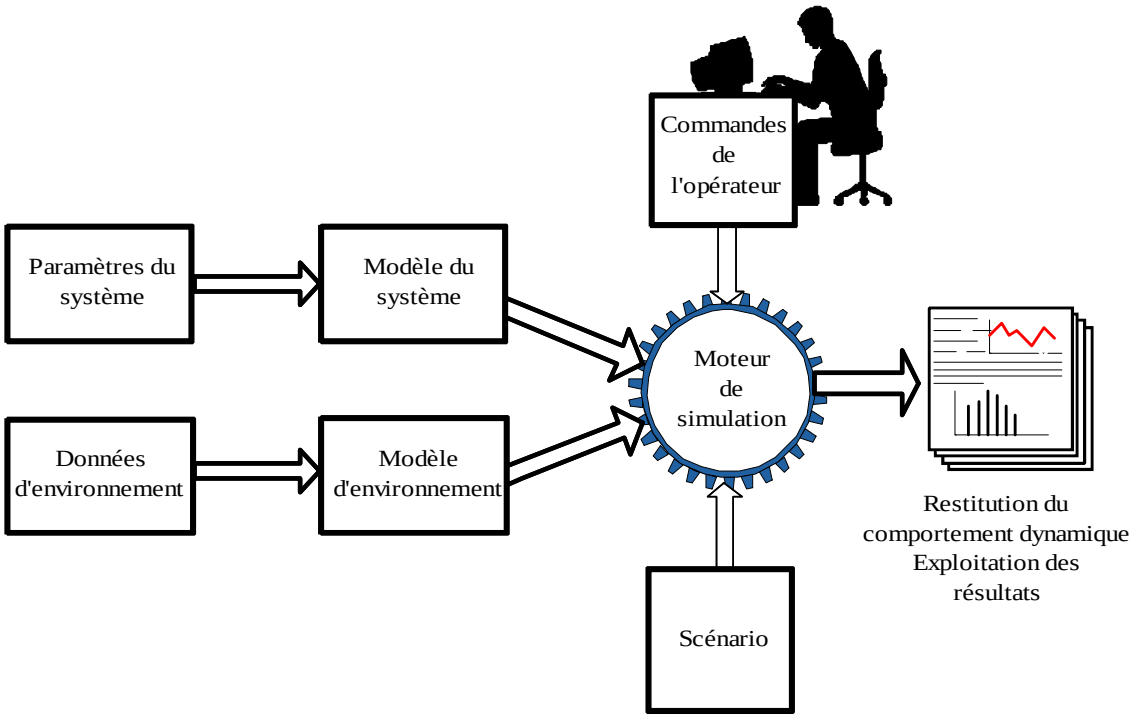


Figure 12: composants d'une simulation (schéma simplifié)

2.1.5 VV&A

Le VV&A (Vérification, Validation et Accréditation) est en quelque sorte l'aspect « qualité » de la simulation.

Le VV&A est un ensemble de procédures et de méthodes permettant de contrôler que le produit (le modèle ou la simulation) est conforme à ce que le client attendait.

2.1.5.1 Vérification

Processus visant à montrer que l'implémentation d'un modèle (i.e. son codage informatique) correspond bien à la description conceptuelle du système modélisé et aux spécifications du modèle ou de la simulation.

2.1.5.2 Validation

Processus visant à déterminer à quel degré un modèle est une représentation exacte du monde réel, pour un emploi donné.

2.1.5.3 Accréditation

Certification (par le client) de l'adéquation d'un modèle pour une utilisation donnée.

L'accréditation est un « label » qui n'est valable **que pour un emploi donné** du modèle ou de la simulation. Il a son importance dans le cadre d'une politique de réutilisation des modèles.

2.1.5.4 Importance du VV&A

Comme toute partie méthodologique, le VV&A est parfois lourd et consommateur de ressources, aussi certains, heureusement minoritaires, ont tendance à faire l'impasse sur cette contrainte. Ceci est une grave erreur, pour de multiples raisons…

En premier lieu, comme tout produit, une simulation (ou une étude à base de simulation) doit respecter un processus qualité, afin de s'assurer que le produit final sera conforme aux spécifications, au meilleur coût, et le prouver au client.

Ensuite, lorsqu'on s'inscrit dans une démarche de réutilisation des modèles (éventuellement par d'autres équipes de développement), et notamment des modèles, il vaut mieux ne réutiliser que des modèles et du code pour lesquels les conditions de validation sont connues (et donc dans lesquels on peut avoir relativement confiance).

Le VV&A, s'il est correctement traité, comme doit l'être toute démarche qualité, peut s'avérer extrêmement rentable, à court terme (réduit les problèmes de mise au point, surtout sur des projets complexes) et à long terme (facilite la réutilisation du code), sans parler de la satisfaction du client.

2.1.5.5 Démarche VV&A

Un des problèmes actuels du VV&A est qu'il n'y a guère de processus standard, sauf, une fois de plus, aux États-Unis, où le DoD a définit une démarche dans une directive officielle (voir [DOD96]).

La Figure 13 est extraite de cette directive et résume la démarche.

Le processus VV&A étant souvent spécifique à un domaine ou à un type de simulation, nous ne décrirons pas en détail de méthodologie. Ce qui est important de retenir, c'est l'esprit exprimé par la Figure 13 : le VV&A est un processus **permanent**, et non à posteriori[5], auquel participent toutes les personnes impliquées dans le projet, y compris le client.

Nous reviendrons sur quelques méthodes de vérification et de validation au §5.1.1.

[5] En effet, il est plus facile d'assurer la qualité d'un produit avec une méthodologie d'ensemble et une maîtrise continue du projet que par l'application de seulement quelques contrôles ponctuels.

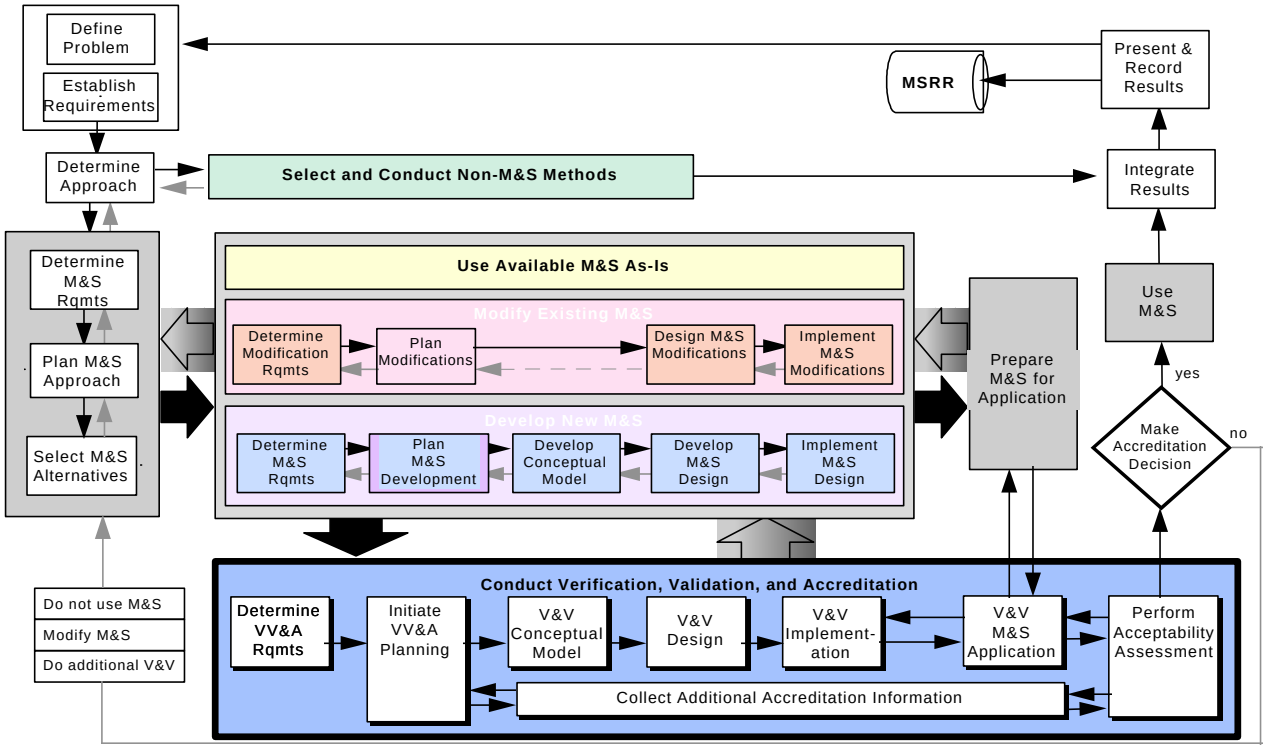


Figure 13: Processus VV&A pour la simulation suggéré par le DoD (version 1996)

2.2 Typologie

2.2.1 Introduction

Dans la mesure où il existe plusieurs « écoles », suivant les domaines et les pays, il est difficile de donner une typologie universelle de la simulation, même si le fond est souvent le même d'une culture à l'autre. Je vais néanmoins ici faire le tour des principales catégories de modèles et de simulation, puis nous verrons quelques classifications parmi les plus courantes.

2.2.2 Modèles

2.2.2.1 Modèle déterministe

Un modèle est dit **déterministe** lorsque, pour une entrée donnée, les résultats sont toujours identiques.

Par exemple, notre modèle de boule de billard qui tombe est déterministe, puisque deux exécutions avec les mêmes données d'entrée donnent les mêmes sorties.

Par exemple, le modèle utilisé pour la chute de notre boule de billard est tout à fait déterministe.

2.2.2.2 Modèle stochastique ou aléatoire

Un modèle est dit **stochastique** ou **aléatoire** lorsque ses résultats sont apparemment[6] déterminés par une ou plusieurs variables aléatoires qui représentent un processus incertain.

Les valeurs de ces variables aléatoires peuvent suivre une loi statistique. Ce type de modèle permet de rendre compte d'un processus aléatoire. Exemple : l'impact d'une balle sur une cible est entaché d'une erreur de trajectoire aléatoire (dispersion), suivant une loi connue (voir §4.3.2).

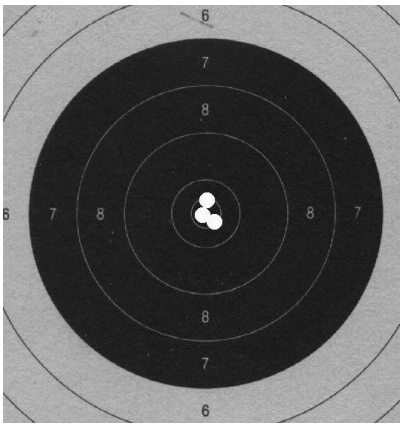


Figure 14: Cible après tir de 3 cartouches avec une arme prise dans un étau

Un tel processus est qualifié de **processus stochastique**. On parlera en outre de **processus de Markov** pour désigner un processus stochastique dans lequel la probabilité d'occurrence d'un événement ne dépend que des événements qui ont précédé. Si ce processus est discret, on dira qu'il constitue une chaîne de Markov.

Un modèle stochastique est par essence non-déterministe : pour une même entrée, le résultat ne sera pas forcément le même. De fait, lorsqu'on tire sur une cible, il est impossible de faire passer *volontairement* deux balles par le même trou, même si on place l'arme dans un étau pour l'immobiliser.

Autres exemples de processus aléatoire : panne d'un matériel, entrée d'un client dans un magasin, désintégration d'un noyau atomique instable…

On définira de même un **système déterministe** comme étant un système dont l'évolution est prévisible : à partir d'un état S_i, pour des conditions données, il évoluera nécessairement vers l'état S_j. En revanche, dans les mêmes conditions, un **système stochastique** pourra évoluer de plusieurs façons possibles.

Figure 15: Système déterministe

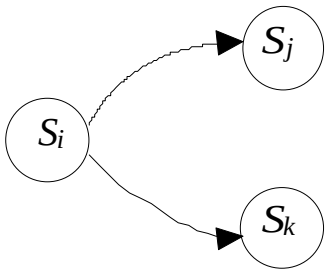


Figure 16: Système stochastique

Le fait qu'un modèle soit déterministe ou non peut résulter d'un choix de modélisation. Ainsi, la détermination de la ligne de vision (LOS, *line of sight*) d'une entité d'une simulation peut se faire suivant un mode géométrique (déterministe), dans lequel on tire une ligne entre le point visé et l'entité et on vérifie qu'il n'existe aucun obstacle, ou

[6] « Apparemment », car des systèmes très complexes, malgré le déterminisme de leurs composants, peuvent apparaître comme stochastiques, voire chaotiques. Ainsi, lorsqu'on provoque des vaguelettes à la surface d'un bassin, les réflexions sur les bords et interférences multiples rendent l'état de la surface de l'eau rapidement difficile à prévoir.

probabiliste, dans le cas où on souhaite modéliser l'inattention de l'entité, ou la perturbation introduite par la végétation ou le brouillard.

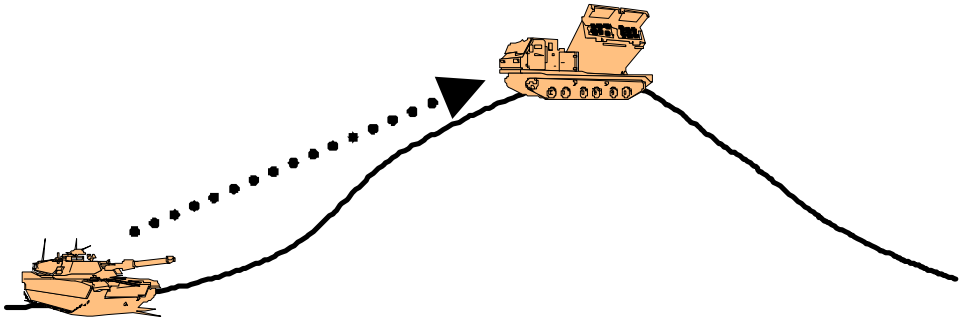


Figure 17: Calcul de LOS géométrique

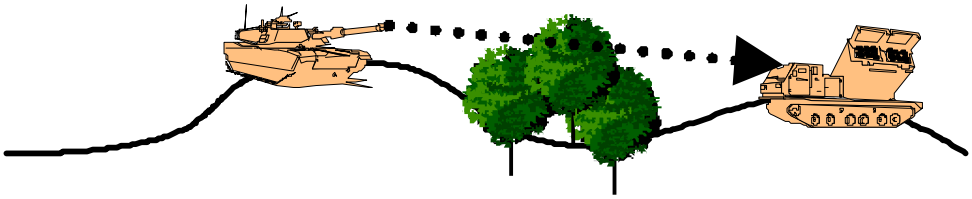


Figure 18: Calcul de LOS probabiliste

2.2.2.3 Modèle physique

Un modèle est dit **physique** lorsque certaines de ses caractéristiques *physiques* sont proches de celles du système que l'on modélise. Par exemple, une maquette d'avion destinée à un essai en soufflerie est un modèle physique.

Par opposition, on parlera de modèle **abstrait** lorsque le modèle représente le système de façon symbolique, par exemple par des équations ou un algorithme. Notre modèle de boule qui tombe est un modèle abstrait.

2.2.2.4 Modèle analytique

Un **modèle analytique** est un modèle abstrait décrit entièrement à l'aide d'équations mathématiques.

Notre modèle de boule qui tombe est ainsi un modèle analytique. On ne trouve pas seulement des modèles analytiques de phénomènes physiques. Ainsi, certains modèles économiques reposent sur des formules mathématiques modélisant la loi de l'offre et de la demande sur les marchés. Le modèle français CESAR de calcul de coût et de disponibilité des armements entre également dans cette catégorie.

2.2.2.5 Modèle conceptuel

Un modèle **conceptuel** décrit les composants d'un système et les relations entre eux, y compris sa logique de fonctionnement et les algorithmes des logiciels qu'il utilise. Le modèle conceptuel précise également les hypothèses prises et les limitations du système.

L'élaboration d'un modèle conceptuel est généralement l'une des premières étapes du processus de modélisation, car il permet de savoir ce que l'on modélise et dans quelles conditions. Il exprime la vision que l'ingénieur (réalisant la modélisation) a du système.

2.2.2.6 Modèle comportemental

Un modèle **comportemental** représente les réactions du système aux événements, i.e. par un processus décisionnel.

Par exemple, certaines sociétés de marketing utilisent des modèles de « clients virtuels » qu'ils font se déplacer dans un magasin afin d'optimiser la disposition des rayons et des accroches commerciales. Par exemple, à partir de l'état « recherche d'un produit pour la lessive », la règle « si une lessive Desh ou Bonix est en promotion, alors prendre cette lessive sinon choisir Arial » permettra d'effectuer la décision après examen de l'environnement.

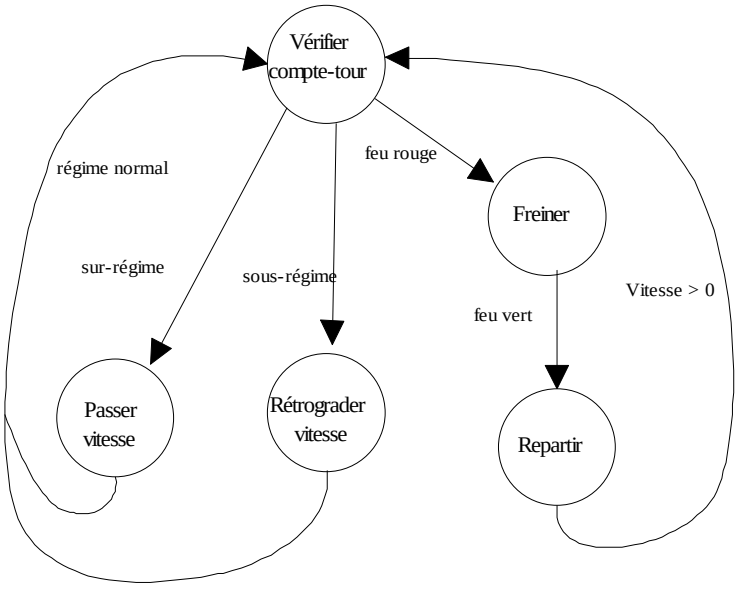


Figure 19: Modèle comportemental d'automobiliste (extrait)

Une implémentation typique des modèles comportementaux repose sur des automates à états, dont les transitions sont dictées par des conditions ou des règles.

2.2.2.7 Qualités d'un modèle

Un modèle doit être :

- Le plus simple et le plus clair possible
- Il doit être valide (et validable !)
- Le plus fidèle possible par rapport au besoin de la simulation
- Le plus efficace par rapport au but poursuivi

Le meilleur modèle est celui qui reflète le mieux les aspects du système réel que l'on veut étudier, compte tenu du besoin et des contraintes de moyens et de délais de développement.

2.2.3 Simulation

2.2.3.1 Simulation ouverte ou fermée

Une simulation est dite **ouverte** quand un opérateur a la possibilité d'intervenir sur le déroulement de son exécution, par exemple en modifiant l'état d'un des objets. Ex. : un simulateur de vol. On dit aussi qu'elle est **interactive**.

Par opposition, on dira qu'elle est **fermée** lorsqu'elle se déroule sans intervention humaine, à l'exception de l'entrée des conditions initiales. Ex. : notre simulation de boule de billard qui tombe est fermée, car il n'y a aucune influence de l'opérateur sur le résultat (mis à part les paramètres initiaux bien entendu).

2.2.3.2 Simulation numérique

On parle de **simulation numérique** (et de modèle numérique) lorsque la simulation est entièrement réalisée par logiciel, par opposition à une simulation comprenant l'homme (simulation constructive ou pilotée) ou du matériel dans la boucle (**simulation hybride**).

Le terme de « simulation numérique » est parfois utilisé pour désigner une simulation numérique (suivant la définition ci-dessus) fermée.

Dans ce cours, nous utiliserons le premier sens, plus général, pour l'opposer à la simulation basée sur des maquettes par exemple, qui n'est pas l'objet de ce cours.

Nous utiliserons le terme de **simulation scientifique** pour désigner une simulation numérique fermée du niveau phénomène physique (voir §2.2.4.2).

La simulation météorologique permettant de prévoir le temps est une simulation scientifique (numérique et fermée), faisant évoluer dans le temps un modèle du système « atmosphère terrestre » (comportant également un modèle statique du terrain et un modèle dynamique des océans, puisque tous deux influent sur le climat).

2.2.4 Quelques classifications

2.2.4.1 Généralités

Il est difficile de donner une typologie universelle de la simulation, car comme vous devez commencer à vous en douter, celle-ci comporte de nombreux domaines parfois très différents, et donc de multiples cultures.

Je vous propose donc ici quelques classifications de la simulation qui sont couramment utilisées, et qui devraient être acceptées par une large communauté internationale :

- Suivant le domaine simulé (niveau du modèle)
- Suivant la technique ou l'architecture employée (nature de la simulation)
- Suivant la finalité (emploi)

Selon les besoins, on trouvera également des classifications suivant la façon de gérer le temps (temps réel ou non), la nature discrète ou continue de la simulation, l'architecture (distribuée ou monolithique), le type d'utilisateur (grand public, industriel ou militaire), etc.

2.2.4.2 Selon le niveau

Une autre façon de classer les simulations consiste à considérer le niveau d'emploi, c'est-à-dire l'envergure physique du système étudié :

- **Phénomènes physiques**. Exemple : propagation d'un signal électro-magnétique.
- **Composants** (modèles techniques). Ex. : un autodirecteur de missile.
- **Sous-système** (fonctions). Ex. : le système de guidage/navigation d'un missile.
- **Système** (typiquement, simulation d'un duel). Ex. : missile, avion.
- **Tactique** (unité élémentaire). Ex. : escadrille d'avions.
- **Opérationnel** (division, brigade).Ex. : simulation d'une partie d'un conflit.
- **Stratégique** (théâtre). Ex. : simulation globale de la guerre du Golfe.

Cette classification se retrouve avec des variantes dans tous les pays. Ainsi, le schéma ci-dessous représente la classification utilisée à la DERA[7] britannique (extrait d'un papier de la DERA présenté à la conférence M&S OTAN d'octobre 2000 par R. Bird) :

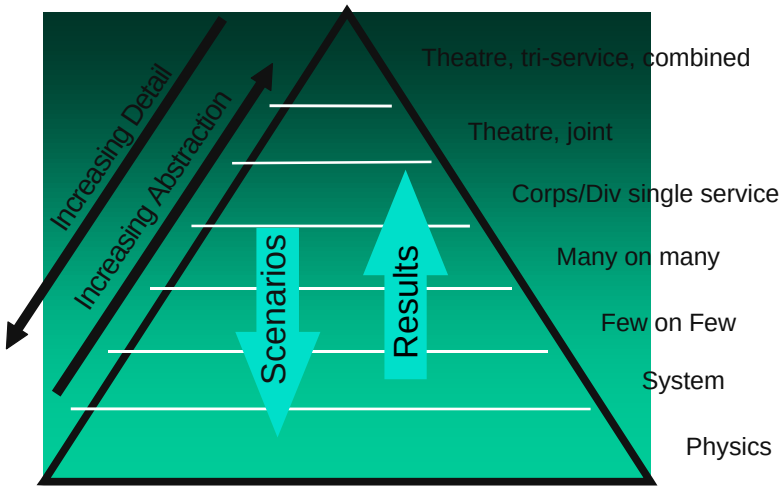


Figure 20 : Hiérarchie des modèles et simulations à la DERA

Les modèles les plus nombreux sont, bien entendu, ceux de plus bas niveau (physique, composant, sous-système et système).

Certaines simulation peuvent manipuler différents niveaux simultanément, en agrégeant par exemple plusieurs plates-formes pour représenter une unité.

[7] *Defence Evaluation and Research Agency*, équivalent de la DCE et du CAD au sein de la DGA française.

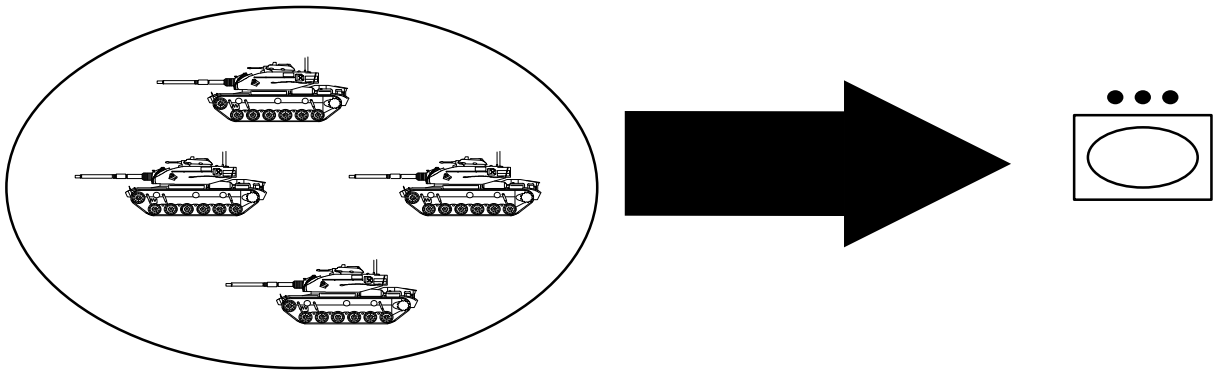


Figure 21: Agrégation d'un peloton de chars

L'**agrégation** d'entités en simulation est complexe, mais peut s'avérer nécessaire pour de multiples raisons :

- Représentation d'une organisation (hiérarchie d'une armée par exemple), qu'elle soit visuelle (visualisation agrégée à l'écran, mais traitement interne unitaire) ou conceptuelle (agrégation du comportement de 4 chars individuels pour avoir le comportement d'un peloton).
- Amélioration des performances dans les simulations manipulant un large nombre d'entités, en les regroupant, de façon à diminuer le nombre de traitements.
- Prise en compte de modèles comportementaux de haut niveau
- Fonctionnement conjoint de deux modèles de niveau différent.
- …

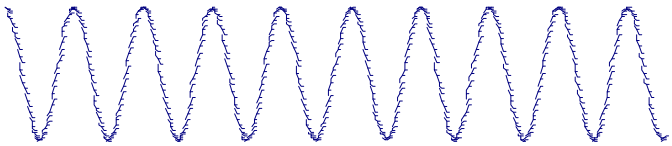


Figure 22: Phénomène physique (propagation d'onde)

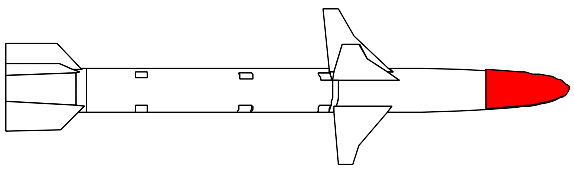


Figure 23: Composant (autodirecteur de missile)

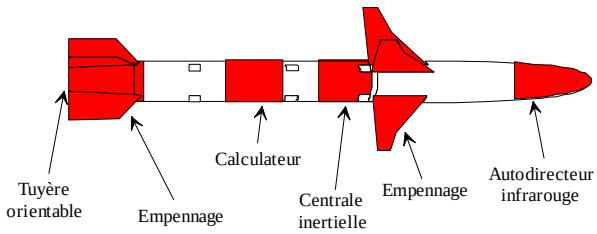


Figure 24: Sous-système (pilotage d'un missile)

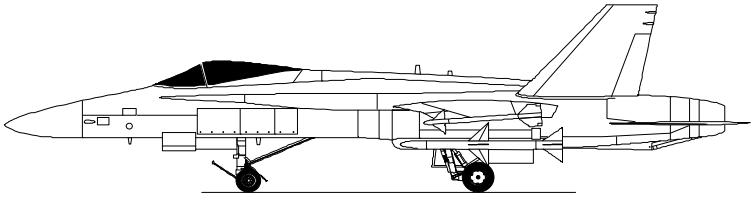


Figure 25: Système (système d'armes: avion avec missiles)

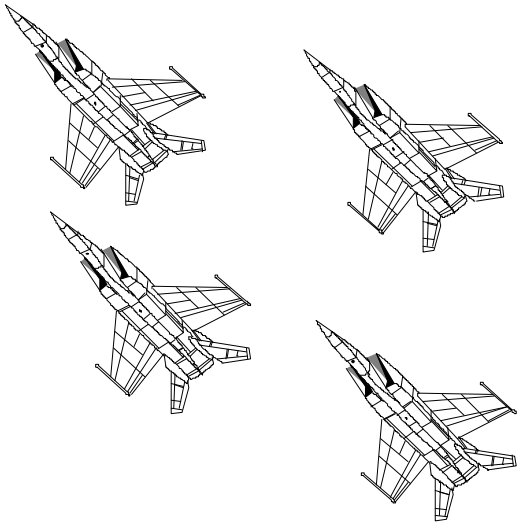


Figure 26: Niveau tactique (escadrille d'avions)

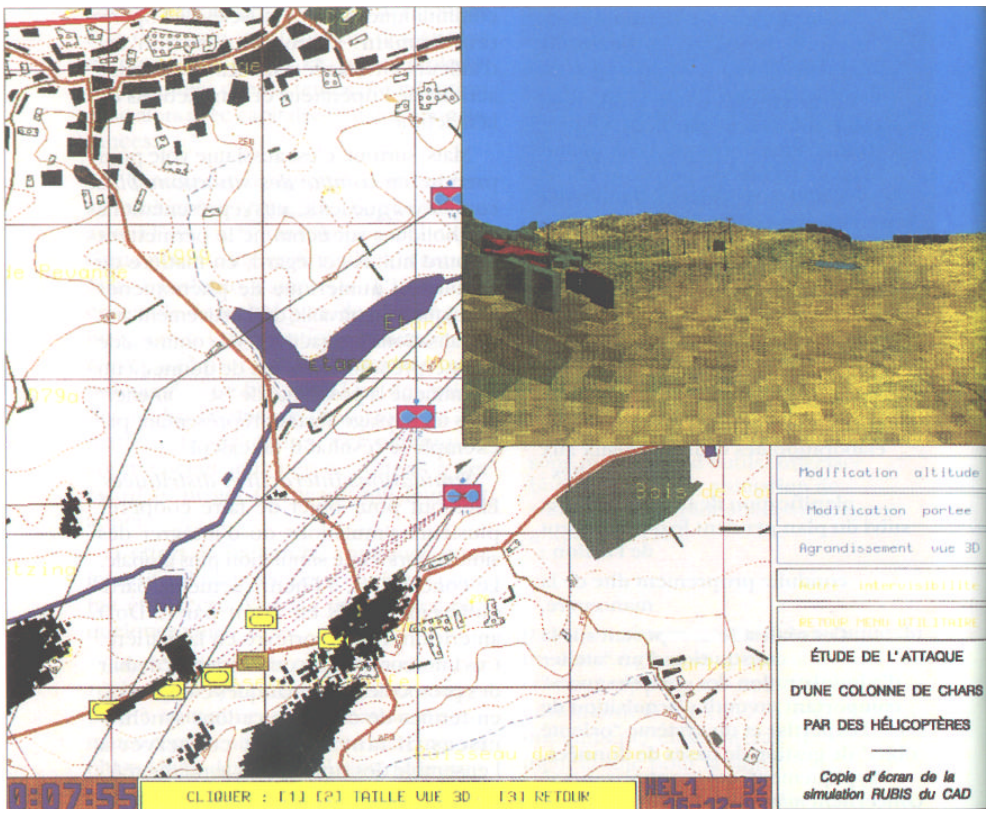


Figure 27: Niveau tactique (jeu de guerre RUBIS)

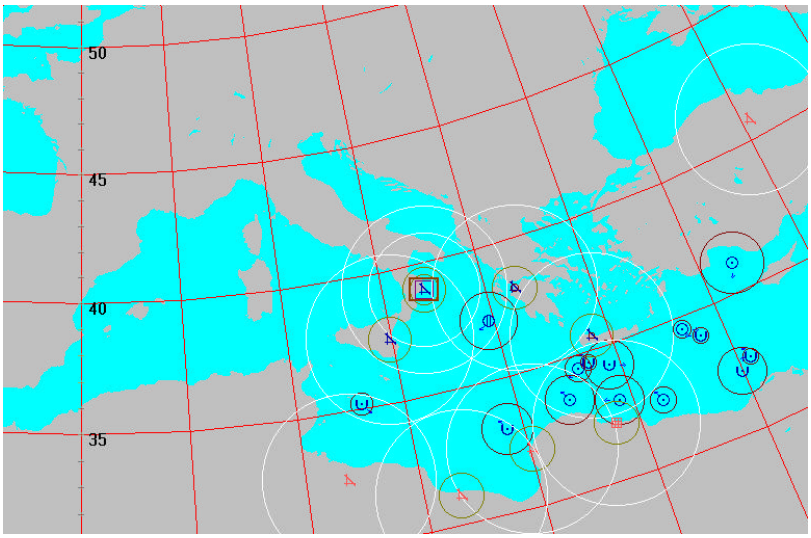


Figure 28: Niveaux opérationnel et stratégique
(Jeu de guerre aéronaval Harpoon, sur le théâtre méditerranéen)

2.2.4.3 Selon l'architecture

Les **simulations numériques fermées**, dites également simulations scientifiques[8], sont purement logicielles. Leur code modélise typiquement le comportement ou les évolutions dans le temps d'un système, souvent du niveau phénomène physique ou composant. Elles font souvent appel à des moyens de calculs intensifs. Ainsi, la grande majorité des supercalculateurs actuellement en service sert à la simulation numérique (météorologie, nucléaire, mécanique des structures, mécanique des fluides).

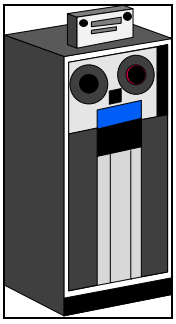


Figure 29: Simulation numérique fermée

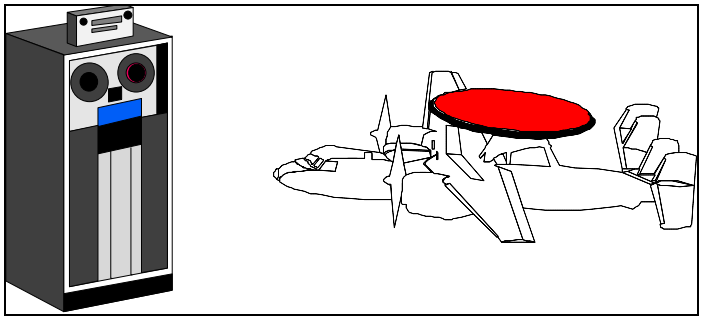


Figure 30: Simulation hybride

Les **simulations hybrides** intègrent dans la boucle du matériel réel. Le modèle numérique peut fournir à un composant d'un système les entrées correspondant à un environnement donné, de sorte que, vu du composant en question, tout se passe comme s'il était intégré dans le système en fonctionnement. Les simulations hybrides sont particulièrement utilisées à des fins de développement ou de qualification des matériels (composants et sous-systèmes).

Les **simulations constructives** sont également logicielles. Elles peuvent intégrer une modélisation de l'humain, par exemple d'un processus décisionnel d'engagement de combat contre une force ennemie. Elles sont souvent interactives, mais l'opérateur réel, s'il peut intervenir dans le déroulement de la simulation, n'est pas le seul élément déterminant pour le calcul des résultats. Les jeux de guerre (*wargames*) entrent typiquement dans cette catégorie.

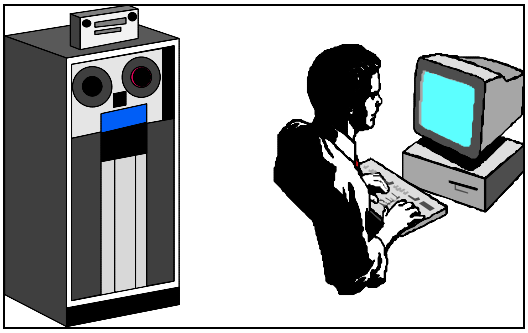


Figure 32: Simulation constructive

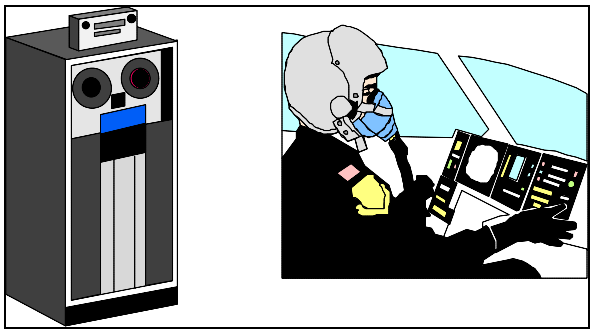


Figure 31: Simulation pilotée

[8] Bien que le terme « simulation scientifique » s'applique de façon plus restreinte, désignant plutôt des simulation numériques fermées de phénomènes physiques.

Les **simulations pilotées**, souvent désignées par le terme « **simulateurs** », en revanche, intègrent l'homme dans la boucle comme facteur prépondérant, à l'aide d'une reproduction des commandes réelles du système simulé. Les américains intègrent ce type de simulation dans ce qu'ils appellent la « simulation virtuelle » (*virtual simulation*), qui est un terme un peu plus vaste, puisqu'il recouvre également les simulations de SIC[9].

Les **simulations instrumentées**, enfin, intègrent des matériels réels opérés par des humains réels eux aussi. Ce matériel est alors spécialement instrumenté pour l'occasion. C'est le cas par exemple des systèmes laser qui se montent sur les armes d'infanterie et permettent, sur le terrain, de simuler les tirs. La terminologie anglo-saxonne est *live simulation*. Ce dernier terme recouvre souvent aussi, chez eux, la simulation hybride.

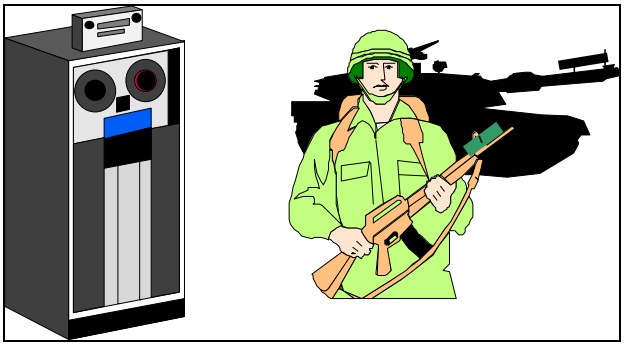


Figure 33: Simulation instrumentée

2.2.4.4 Selon la finalité

Une simulation peut être employée à bien des fins. Elle peut servir pour la **formation et l'entraînement**. Il s'agira alors de faire acquérir à l'opérateur une compétence et un comportement dans un certain type de situation. La simulation s'attachera donc à le plonger dans un environnement qui lui *paraisse* proche de la réalité.

La simulation peut également servir à aider les opérationnels à utiliser au mieux les forces dont ils disposent. Pour cela, elle leur permettra de réaliser des **études d'emploi** (mise au point de la doctrine), et les assistera dans la **planification** et la **conduite d'opérations** militaires. Dans le domaine civil, ce type d'étude pourra s'appliquer à l'optimisation d'un processus, d'une flotte de transport, etc.

Enfin, la simulation est un **outil de base de l'ingénieur**, tout au long du processus d'acquisition d'un système, permettant la **réalisation d'études pour spécifier[10], concevoir et qualifier** le système et ses évolutions ultérieures. La quintessence de l'utilisation de la simulation à des fins d'ingénierie système est la réalisation d'un **prototype virtuel**, modèle numérique du système complet, que l'on placera dans un **environnement synthétique** représentatif de sa mission. Il devient alors possible d'étudier, de tester, d'opérer un système et même de s'entraîner à son utilisation, avant qu'un prototype réel, souvent coûteux, ne soit fabriqué. Ainsi, une erreur de

[9] Systèmes d'information et de commandement, *C3I* en anglais.

[10] On estime que 70% du coût d'un programme d'armement est déterminé par les études amonts et les spécifications, d'où l'importance de la simulation pour limiter le risque d'erreur dans ces phases.

spécification ou de conception majeure, d'autant plus coûteuse qu'elle est traitée tardivement, aura plus de chances d'être découverte avant la mise en chantier d'un prototype, voire du système définitif dans le cas de productions à l'unité (porte-avions, sous-marin, …).

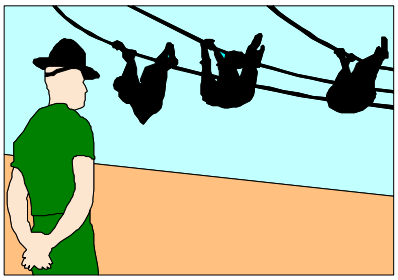


Figure 34: Entraînement et formation

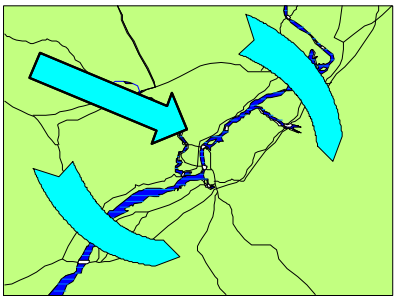


Figure 35: Aide à la décision (planification...)

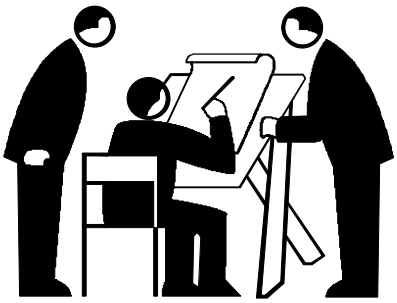


Figure 36: Ingénierie

2.3 Emplois

2.3.1 Ingénierie et acquisition

2.3.1.1 Intérêt de la simulation pour l'ingénieur

En amont de l'utilisation d'un système, on trouve une phase d'acquisition (côté client) et de développement/ingénierie du système (côté fournisseur)[11]. Ici, la simulation aide l'ingénieur à **spécifier**, **concevoir, développer** et **qualifier** le système.

La simulation a eu en effet un impact considérable sur le travail de l'ingénieur, en lui permettant d'effectuer des essais en virtuel à moindre coûts et en testant plus en amont différentes hypothèses de spécifications ou de conception.

Le produit fini s'en ressent bien entendu fortement. On le voit dans l'évolution des performances des avions par exemple :

- Du temps de Blériot, la mise au point des avions restait très empirique. On avait recours à des essais, souvent périlleux.
- Pour le Concorde, on fit un nombre très important d'essais en soufflerie, et on commença à utiliser des calculateurs informatiques.
- Dès les premiers Airbus, les essais en soufflerie, la CAO et la simulation furent généralisés. Le gain en coûts et en délais fut décisif pour conquérir un marché jusque là chasse gardée des américains (Boeing et McDonnell-Douglas).
- Enfin, le Rafale fut entièrement calculé avant les essais.

[11] Les outils de simulation utilisés chez le fournisseur et le client étant proches, nous les traiterons au sein d'un même paragraphe.

2.3.1.2 Aide à la spécification

Avant tout travail de conception, il faut que le client (l'acheteur) exprime son besoin. Veut-on un avion de ligne de 300 ou 400 places ? Combien de missiles le Rafale doit-il emporter pour mener au mieux une mission SEAD[12] ? Quelles sont les performances que rend possibles un nouveau matériau ?

On s'intéressera ici surtout aux fonctionnalités du système et de ses composants. Les simulations seront plutôt de haut niveau. Ce seront des simulations **technico-fonctionnelles** (simulation du *fonctionnement* d'un système ou d'un composant dans son contexte d'utilisation opérationnelle) ou **technico-opérationnelles** (simulation de la *mission* d'un système ou d'un ensemble de systèmes dans le cadre d'un scénario d'emploi). On parle également de simulation pour l'aide à la conception de l'outil de défense (SACOD).

L'objectif est d'obtenir des spécifications minimales (exprimant le besoin, sans plus, car le zèle coûte parfois cher en ingénierie), réalistes (le système doit être faisable avec les technologies actuelles ou à venir dans un futur proche) et suffisantes (pour réaliser la mission attendue du système par le client).

Notons que les **maquettes et prototypes virtuels**[13] sont des outils de plus en plus utilisées par les fournisseurs pour dialoguer avec leurs clients.

2.3.1.3 Aide à la conception

Il s'agit de trouver les solutions techniques en adéquation avec les spécifications. Pour cela, on fait largement appel à la simulation technique scientifique.

Les principales applications de la simulation scientifique en ingénierie sont les suivantes :

- Calcul de structures
- Crash
- Mécanique des fluides
- Hydraulique
- Thermique
- Vibrations, acoustique
- Électromagnétisme

L'objectif est le plus souvent d'optimiser les formes, les matériaux, les procédés, pour améliorer la réponse du système à une sollicitation physique. Par exemple, en haute-fidélité, on étudiera plusieurs types de caisses pour enceintes acoustiques, et on déterminera celles qui vibrent le moins dans les basses fréquences, qui sont l'un des problèmes majeurs qui se posent aux constructeurs d'enceintes et de haut-parleurs.

Thomson-Airsys, fabricant français de radars, a ainsi développé un atelier d'ingénierie de radar (AIR), qui est basé sur la simulation de bas niveau des émissions, propagation

[12] Suppression des défenses aériennes ennemies. Il s'agit de détruire les sites de missiles sol-air et les radars ennemis (par exemple à l'aide de missiles antiradar), afin de permettre au raid principal d'attaquer les installations sans subir de lourdes pertes.

[13] Une maquette est une simple représentation d'un système, alors que le prototype est capable d'exécuter (ou de simuler s'il est virtuel) les fonctionnalités du système, même s'il n'est pas totalement conforme au modèle de série.

et réflexions des ondes radars, limitant la réalisation de prototypes, avec les gains en temps et en coût induits, et permettant d'améliorer de façon sensible le rapport coût/performances du système final. Cet atelier logiciel utilise plusieurs produits du commerce, tels que Matlab[14].

2.3.1.4 Aide à la décision d'acquisition

Par la simulation, on peut comparer différentes propositions de fournisseurs et déterminer la meilleure. Pour reprendre un exemple dans le monde de l'armement[15], lorsqu'un pays acquiert un système d'armes (avions, hélicoptère, char…), de nombreuses simulations sont effectuées pour comparer les matériels. Ainsi, la Norvège, souhaitant moderniser ses forces aériennes, a utilisé des codes de simulation pour mettre en situation des F16 américains, des Rafale français et des Jas39 suédois dans différentes missions et comparer leurs performances. Difficile en effet de faire combattre autrement que virtuellement des Rafale (il n'en existait que quelques démonstrateurs à l'époque) contre des Mig-29 et des Su-27 !…

2.3.1.5 Qualification

Lorsque le fournisseur a livré le système (ou les premiers exemplaires), il faut le qualifier, c'est-à-dire vérifier sa conformité aux spécifications contractuelles.

La qualification se fait typiquement par des essais. Toutefois, certains essais peuvent être coûteux (par exemple des essais impliquant la destruction du système, ce qui est généralement le cas pour un missile !), ou difficile à effectuer (il est politiquement suicidaire de tester une tête nucléaire de nos jours…).

La simulation, si elle ne remplace pas totalement les essais, permet de limiter ceux-ci au strict minimum. D'autant que, réduction des coûts oblige, de moins en moins de moyens sont attribués à cette phase, pourtant essentielle.

Il est absolument nécessaire d'imbriquer étroitement simulation et essais. La simulation doit permettre de limiter et d'optimiser au mieux les essais, et les résultats des essais doivent servir à valider les simulations, à leur fournir les données qui leurs sont nécessaires et à améliorer leur fidélité.

2.3.1.6 Évolutions du système

Il ne faut pas oublier que le cycle de vie d'un système ne s'arrête pas à sa mise en service. En effet, lorsqu'un système a une durée de vie longue (plusieurs années, voire quelques décennies), il s'avère souvent nécessaire de le remettre à niveau régulièrement (*upgrade*, *retrofit*…).

Or, cette mise à niveau doit suivre un processus d'ingénierie similaire à celui qui a conduit à la réalisation du système. La simulation est donc encore appelée à intervenir (d'où l'intérêt de capitaliser précieusement les modèles développés précédemment, voir §2.3.1.8).

Enfin, lorsque le système arrive en fin de vie, il faut songer à son retrait du service, opération qui n'est pas toujours anodine, comme le montre le problème du désarmement

[14] Matlab est probablement le logiciel de simulation générique le plus employé en ingénierie. Il s'agit d'un produit commercial, pour lequel de nombreuses bibliothèques ou extensions ont été développées pour des applications variées. Notons qu'il existe un « clone » sous forme de logiciel libre, Scilab.

[15] La simulation est bien entendu largement utilisée dans l'industrie civile, mais il est vrai que, chez les militaires, le coût et la complexité des systèmes ont développé une importante « culture simulation ».

d'une bonne partie des forces stratégiques russes : quelles unités détruire ? quelle procédure de destruction suivre ? sur quel site ? avec quels moyens de transport ? comment assurer la sécurité ? que faire du plutonium ? comment traiter les composés chimiques utilisés pour la propulsion ? peut-on recycler des composants ? etc. Le problème est tel que les russes ont dû se faire aider financièrement et techniquement par… les américains !

2.3.1.7 Prospective

La simulation est également un outil précieux pour les **études amont**. Ce sont des études à but prospectif, réalisées principalement afin d'évaluer le potentiel de nouvelles technologies, de maintenir une base de compétences techniques et d'analyser les besoins à long terme[16].

On pourra ainsi comparer différents concepts pour un système appelé à être mis en service… dans quelques décennies. Par exemple, de nombreux pays étudient des avions militaires sans pilotes (UAV). On peut également citer les nombreuses études effectuées par la NASA pour préparer un éventuel vol habité vers Mars.

2.3.1.8 Simulation pour l'acquisition (SBA[17])

La simulation, nous l'avons vu, est utilisée aujourd'hui à toutes les étapes du cycle de vie d'un système : analyse du besoin, spécifications, qualification, formation et entraînement, étude des évolutions. La Figure 37 résume cette implication dans le **cycle d'acquisition** d'un système.

[16] Ainsi, à la DGA, il existe un plan prospectif à 30 ans (PP30), qui définit les besoins à long terme des armées, et par là même les priorités en matière d'études amonts.

[17] *Simulation Based Acquisition*. Il s'agit du terme anglo-saxon, souvent repris pour désigner en France la simulation pour l'acquisition, bien qu'il y ait quelques nuances entre les visions américaines, britanniques et françaises du SBA.

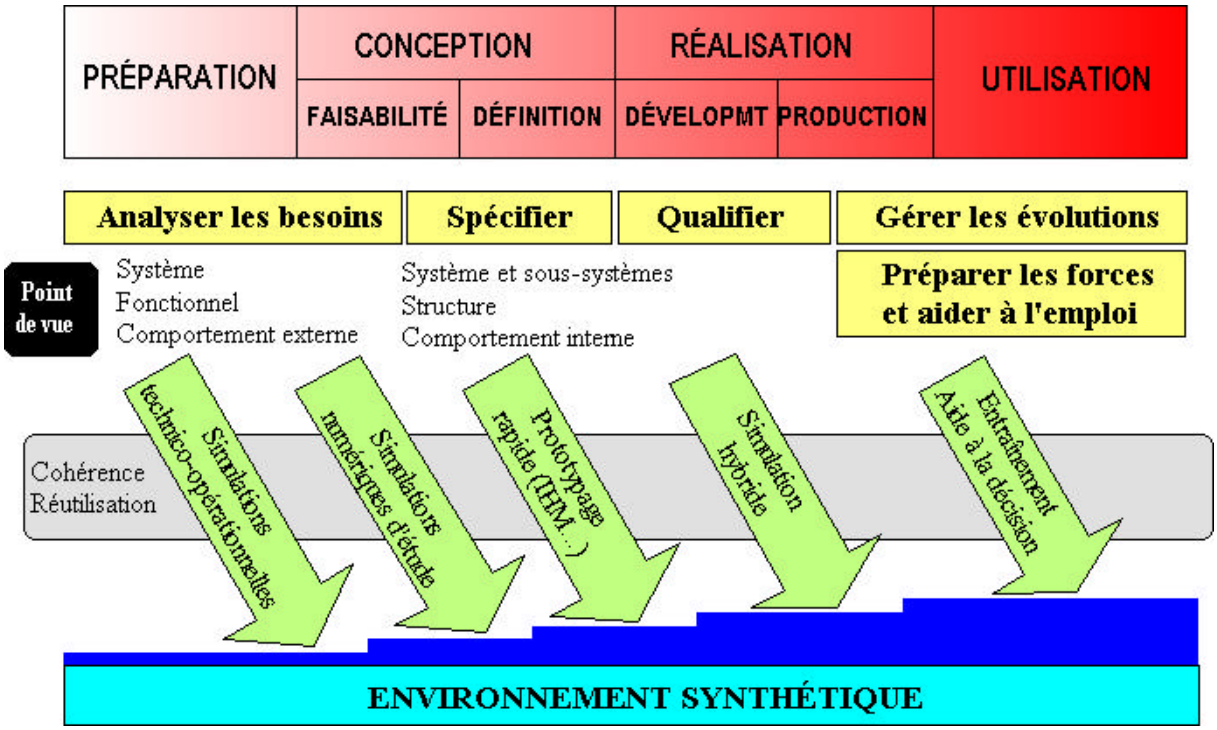


Figure 37: Simulation et processus d'acquisition

Actuellement, le système se retrouve modélisé à différentes étapes de sa vie. En effet, chacune de ces étapes est sous la responsabilité d'équipes distinctes. On voit ainsi souvent réaliser plusieurs modèles et outils similaires, sans cohérence aucune.

Le SBA est une « doctrine d'emploi » de la simulation visant à l'**intégrer dans le processus d'acquisition** de façon cohérente, en en faisant un outil essentiel et en permettant la réutilisation et l'interopérabilité des modèles, données et outils entre les différentes étapes de ce processus.

Le SBA doit permettre à terme, à travers la définition de standards, de méthodes et d'outils communs, de parvenir à créer des bibliothèques de composants de simulation réutilisables (documentés et validés bien entendu !), afin de pouvoir réaliser des **prototypes virtuels** de chaque système. Ces prototypes virtuels pourront alors utilisés dès les phases amonts.

D'origine américaine, ce concept est en cours de mise en place à la DGA, sous la responsabilité du département ingénierie des systèmes complexes du STTC[18], avec l'aide de la DCE et du CAD.

Aux Etats-Unis, l'introduction de ces nouvelles techniques de modélisation et simulation a ainsi permis de réduire le nombre d'éléments des sous-marins nucléaires d'attaque de nouvelle génération à 16 000 pièces contre 95 000 pour la génération précédente (Seawolf).

[18] Où travaille l'auteur de ces lignes, qui accorde une grande importance à ce projet.

2.3.2 Entraînement et formation

L'entraînement et la formation sont les activités de la simulation les plus médiatisées, à travers les simulateurs de vol, mais aussi spectaculaires qu'ils puissent être, ces derniers ne représentent qu'une partie des systèmes destinés à cette finalité.

La formation (*education* chez les anglo-saxons) vise à faire acquérir les compétences initiales à un élève sur une procédure ou un système. Les lacunes des connaissances et le manque d'expérience de l'élève rendent souvent dangereux son passage direct sur le système réel, d'où l'intérêt d'une étape intermédiaire en simulation.

L'entraînement (*training*) a pour objet de maintenir et développer les compétences de l'utilisateur d'un système, en augmentant le temps passé sur le système, ou en expérimentant des situations particulières (pannes...). Quand il s'agit de systèmes dont le fonctionnement est onéreux ou le maniement délicat (avions...), on a largement recours à la simulation.

Le simulateur d'entraînement peut se focaliser sur la maîtrise globale du système (cas des simulateurs de vol) mais aussi sur un processus particulier (atterrissage d'urgence...).

Aujourd'hui, il existe une pléthore de simulateurs de formation/entraînement dans des domaines très variés, tels que par exemple :

- Pilotage d'un sous-marin
- Conduite d'une pelle mécanique
- Contrôle aérien
- Conduite d'une centrale nucléaire (voir l'exemple de la Figure 38)
- Maintenance de matériels aéronautiques
- Chirurgie laparoscopique
- ...

Les simulations destinées à la formation et à l'entraînement ont par conséquent pour objectif d'aider à la formation et à l'entraînement des personnels et d'en augmenter l'efficacité, avec un coût et un risque moindres.

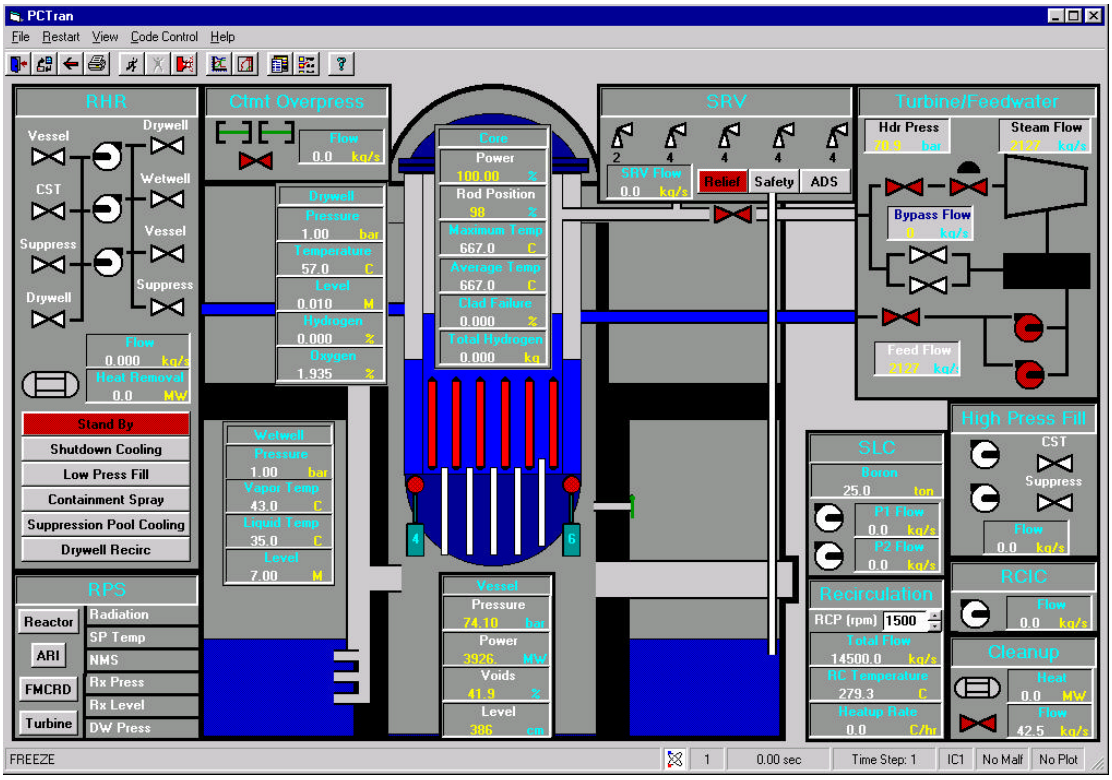


Figure 38: Simulation de centrale nucléaire (PCTran)

2.3.3 Aide à la décision

Dans un monde où les organisations sont de plus en plus complexes, la simulation devient un outil de tout premier ordre pour les décideurs, qu'ils soient ingénieurs, militaires, politiciens ou financiers.

En effet, en simulant les conséquences possibles d'une décision, il est possible de choisir la meilleure stratégie possible face à un problème donné.

Dans le domaine militaire, les principales activités où la simulation apporte actuellement une aide à la décision sont les suivantes :

- Planification à froid : études menées en temps de paix, visant à élaborer à l'avance des plans d'opérations ou des schémas directeurs pour des situations d'engagement (scénarios) hypothétiques, susceptibles de survenir à moyen ou long terme (quelques mois à plusieurs années).
- Préparation des opérations (ou planification à chaud) : en situation de crise, études menées en temps réel, permettant de déterminer la meilleure action (manœuvre, options de déploiement…). Les outils utilisés doivent permettre l'évaluation des différentes options en un temps très court, et avec une fiabilité élevée.
- Déploiement et logistique : étude, à partir de la liste des moyens disponibles et de diverses contraintes (délais, missions, infrastructures, etc.) de la meilleure façon d'acheminer une force donnée sur un théâtre d'opération et d'assurer son ravitaillement ultérieur (optimisation des flux

logistiques). Notons qu'une telle étude peut également servir à dimensionner les moyens de projection de l'armée (camions, wagons, navires, avions, …).

- Évaluation des modes d'actions : avant d'être mise à exécution, une mission peut être analysée et différents modes d'actions sont envisagés, ainsi que les réponses possibles de l'ennemi, puis évalués afin de déterminer la manœuvre qui donnera le meilleur résultat.
- Conduite des opérations : évaluation de la situation tactique, mesure de la dérive par rapport aux prévisions de la planification et replanification éventuelle à chaud.

Globalement, la méthode consiste la plupart du temps à simuler plusieurs fois l'évolution d'une situation donnée, en faisant varier certains paramètres (ceux sur lesquels doit porter le choix du décideur) à chaque itération.

L'utilisation de simulations pour l'aide à la décision impose un certain nombre de contraintes sur les logiciels utilisés :

- validité des modèles et des données (des vies humaines et des matériels coûteux sont en jeu) ;
- interface homme-machine particulièrement claire (l'idéal est de coupler une simulation au SIC[19] habituellement utilisé par le décideur) ;
- une traçabilité des résultats ;
- le déterminisme des résultats (par l'utilisation de modèles déterministes, ou par l'application de méthodes statistiques tels que Monte-Carlo aux résultats de modèles stochastiques) ;
- des temps de réponses compatibles avec le délai de prise de décision ;
- une haute disponibilité du système ;
- l'assurance de la confidentialité des données.

Dans tous les cas, quelque soit les résultats de la simulation, c'est toutefois au décideur que revient le choix. Celui-ci reste responsable, et, grâce à son expérience, doit conserver un esprit critique vis-à-vis de la machine.

2.3.4 Et les jeux vidéo alors ?

Nous avons beaucoup parlé de simulation pour l'ingénieur, pour les militaires, pour les pilotes… Mais, de très loin, le plus grand nombre de simulations sont vendues à des fins de… loisirs !

En se limitant aux simulations informatiques[20], nous pouvons distinguer deux principaux marchés, aux contraintes très différentes : les parcs à thème et les jeux vidéo (pour micro-ordinateur ou console de jeu).

[19] Système d'information et de communication. Un tel système analyse les informations en provenance d'un théâtre d'opérations et en présente une vue synthétique au décideur, et se charge de transmettre les ordres vers les exécutants. Bien que SIC (C3I en anglais) soit un terme surtout militaire, de tels systèmes sont de plus en plus courants dans le civil (supervision du fonctionnement d'une entreprise, d'un processus complexe, d'un réseau de télécommunication…).

2.3.4.1 Les simulations destinées aux parcs à thèmes

Ces simulations ont des budgets considérables (souvent plusieurs millions de dollars), avec un degré de réalisme souvent impressionnant (utilisation de cabines montées sur vérins, de techniques de réalité virtuelle, de générateurs d'images haut de gamme…). Néanmoins, l'interactivité est parfois limitée.

Ainsi, la simulation Star Wars des parcs Disney comprend une cabine dans laquelle prennent place les spectateurs. Elle est montée sur des vérins, permettant de leur faire ressentir, dans une certaine mesure, les accélération, les chocs et les mouvements du vaisseau spatial dont ils sont censés être les passagers. Toutefois, l'interactivité est nulle, puisque les spectateurs sont totalement passifs et ne peuvent agir sur le déroulement de la « simulation », qui est en fait une séquence enregistrée, toujours la même[21].

En revanche, il n'en va pas de même pour les Battle Tech Centers, qui permettent à leurs clients de prendre les commandes de robots de combat futuristes pour s'affronter à l'aide de simulateurs en réseau (voir Figure 39). De tels centres de simulation grand public coûtent des dizaines de millions de dollars, mais sont très populaires.

Dès lors, on peut se demander pourquoi ces installations ne partageraient pas certaines outils ou standards militaires (tels que DIS ou HLA, voir chapitre 9). En fait, ces standards ne sont guère appréciés des fabriquants de simulateurs pour parcs à thème en raison du très court cycle de développement des simulateurs grand public (typiquement 18 mois entre chaque génération) et la concurrence acharnée entre les différents acteurs qui, en l'absence d'un client « poids lourd » tel que l'armée, ne plaide pas pour l'ouverture des produits…

[20] Nous écartons ici les jeux de simulation sur plateau (type wargame par exemple) et les jeux de rôles (type advanced dungeons & dragons).

[21] On pourrait plutôt les qualifier de « cinéma immersif ». Toutefois, on voit apparaître certaines attractions de ce type dont le scénario peut varier en fonction des spectateurs, mais le plus souvent de façon limitée.

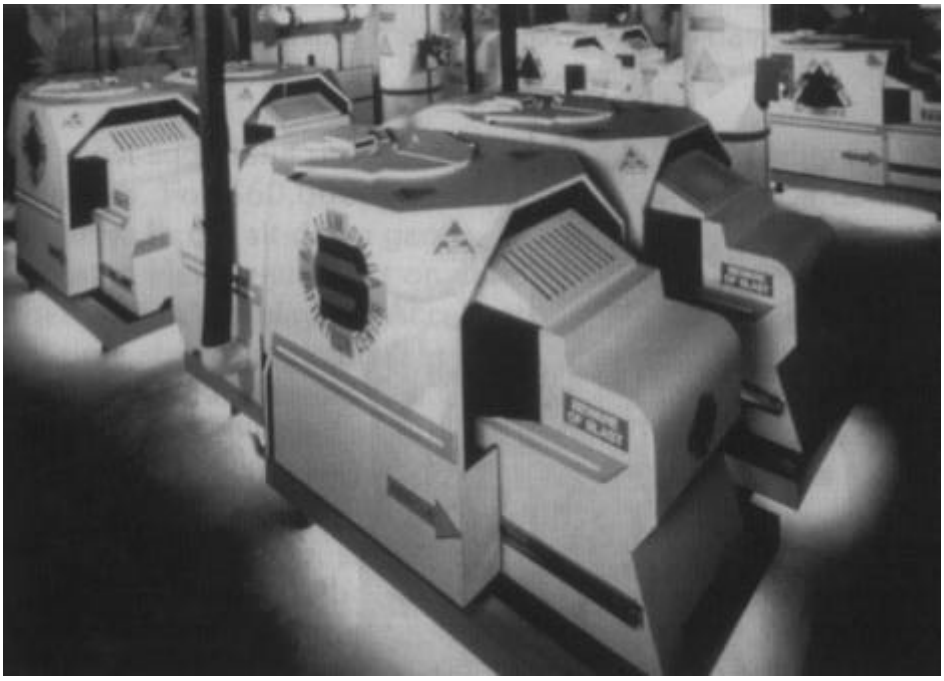


Figure 39: Cabines en réseau d'un Battle Tech Center

2.3.4.2 Les jeux de simulation pour micro-ordinateurs et consoles

Elles représentent des millions de clients[22] dans des thèmes aussi variés que la simulation de vol, la simulation d'entreprise, les wargames, etc. Dès la fin des années 70, on pouvait trouver des logiciels sur TRS 80, Commodore 64 ou, surtout, Apple 2, comme Cartels et Cutthroats, simulation économique simple dans laquelle le joueur crée un empire en lançant des OPA sur des entreprises, ou la série des Germany 1985, de la société SSI, permettant de jouer la troisième guerre mondiale sur divers théâtres (Centre Europe, Norvège, Arabie Saoudite…).

En 1979, Bruce Artwick crée un logiciel de simulation de pilotage d'un avion de tourisme de type Piper Cherokee. Ce logiciel était exceptionnel pour l'époque, puisqu'il comportait la génération de scènes 3D (avec un graphisme monochrome en « fils de fer » des plus dépouillés) sur un… Apple 2[23] ! Trois ans plus tard, la seconde version fut améliorée, notamment par l'introduction de la couleur. Un portage fut effectué sur IBM PC. Sa commercialisation fut confiée à une jeune société, Microsoft… Flight Simulator s'envola alors vers le succès que l'on sait, et se perfectionna au fil des versions au point d'entrer dans certaines écoles de pilotages américaines pour l'apprentissage des manœuvres de base des avions de tourisme. D'autres logiciels similaires, tels que Flight Unlimited, bénéficient également d'un haut degré de réalisme. Notons que de nombreux pilotes de l'US Air Force trouvent une seconde carrière dans les maisons d'édition de jeux vidéo, en tant que conseillers techniques, ce qui montre le souci actuel des éditeurs de proposer des produits les plus proches possible de la réalité.

[22] Le marché des consoles et jeux vidéo représente plusieurs milliards de dollars (4,3 G$ en 1996).

[23] Rappelons que cette machine, l'une des plus populaires à l'époque, était basée sur un microprocesseur 8 bits à 1 MHz et comportait 48 Ko de mémoire vive. Le graphisme « haute résolution » offrait une résolution de 280x192 pixels en 6 couleurs.

2.3.4.3 Le potentiel des simulations de loisirs

Dès lors, on peut se poser la question : pourquoi dépenser des millions à développer des simulations alors que, peut-être, certains jeux commerciaux, vendus 300 F dans les boutiques spécialisées ou les supermarchés, pourraient faire l'affaire ?

Figure 40: Les simulateurs de vol sur PC atteignent des performances intéressantes... (Flanker 2.0)

Les militaires se sont effectivement penchés sur le problème, menant plusieurs études (voir par exemple [NRC97]). Le résultat est en grande partie négatif : la plupart du temps, les jeux vidéos, obéissant à une logique commerciale, font un compromis entre réalisme et jouabilité. Car le grand public bouderait un logiciel trop complexe, ou qui demanderait des moyens de calcul trop importants : pas question dans un jeu commercial de caractériser un char avec 200 paramètres !

Parmi les produits qui sont toutefois sortis du lot, citons Advanced Squad Leader (jeu de guerre terrestre tactique), Tigers on the Prowl, Operation Crusader, Patriot et, bien sûr, Harpoon 2 (jeu de guerre aéronaval, niveaux tactique à théâtre). Ce dernier a fait l'objet d'une étude de la part de l'ANPROS[24], qui concluait que ce jeu pourrait être tout à fait utilisé dans les écoles pour l'apprentissage des tactiques de base de la guerre aéronavale, s'il n'était affublé d'un certain nombre de défauts rédhibitoires, le principal étant l'impossibilité d'opposer au joueur humain un adversaire à la hauteur, dans la mesure où l'intelligence artificielle de l'adversaire est insuffisante (l'opposant, géré par l'ordinateur, se contente de suivre le scénario et ne prend pas d'initiatives stratégiques), et ne peut être remplacé par un joueur humain, le logiciel ne supportant pas le jeu en

[24] Antenne de Planification, de Recherche Opérationnelle et de Simulation de la Marine Nationale.

réseau. Une prochaine version de Harpoon, prévue pour sortir courant 2001 (mais se fait attendre), devrait corriger ce problème et relancera probablement le débat.

Notons que les Marines US ont mené une expérience en finançant ID Software, éditeur des bests-sellers Doom et Quake, pour qu'il développe une version de Doom[25] spéciale, dans laquelle les monstres de la version originale sont remplacés par des troupes plus classiques. Ce « Marines Doom », qui peut regrouper plusieurs joueurs en réseau, est destiné à l'apprentissage des tactiques de base du combat d'infanterie : travail en binôme, inspection d'un bâtiment, etc. Des expériences similaires ont lieu au profit de l'US Army et de l'US Air Force[26]. En France, la DGA a évalué un moteur de simulation du comportement humain de la société de jeux vidéo MASA avec une doctrine et des scénarios opérationnels, et le produit s'est révélé supérieur sur plusieurs points à ce qui existait dans le domaine de la défense.

Quant à MäK Technologies, éditeur américain d'outils pour la simulation distribuée (DIS et HLA), il a fait le chemin inverse, en dérivant un jeu vidéo d'une simulation militaire.

Si, comme nous l'avons vu, il est difficile pour les militaires de se satisfaire de logiciels de jeu « sur étagère », ils peuvent exploiter des technologies civiles. Il n'est pas rare de voir des entraîneurs utilisant des commandes de vol (pédalier, joystick…) développés pour le monde du jeu vidéo. De même, l'engouement du public pour le 3D a permis aux PC de disposer d'un vaste choix de cartes graphiques très performantes (Voodoo3, Riva TNT2, Matrox G200, GeForce256, etc.) à des prix très bas.

Il est probable que dans les années à venir ces technologies duales entre le monde de la simulation militaire et de la simulation vont se développer, notamment dans le domaine de la formation et de l'entraînement de base (simulation pilotée, jeux de guerre…), même s'il est clair que le haut de gamme militaire restera avec ses technologies propres.

Encore faudra-t-il que chacun arrive à surmonter une barrière culturelle parfois pesante…

2.4 La simulation est-elle indispensable ?

2.4.1 Généralités

Comme vous pouvez déjà le deviner, la simulation peut s'avérer complexe et coûteuse. Le fait est que, par exemple, un simulateur de Mirage 2000 est pratiquement aussi cher qu'un avion ! Mais alors, la simulation est-elle vraiment rentable, par rapport à l'utilisation de matériels réels, de prototypes ou de maquettes ?

Pour bien comprendre l'intérêt de la simulation et son apport, nous allons voir des alternatives à la simulation, et dans quelle mesure elles ne permettent pas de remplacer la simulation, que ce soit pour des motifs techniques ou financiers.

[25] Dans Doom, le joueur est plongé dans un univers en pseudo-3D et vision subjective, et doit trouver son chemin dans un monde hostile. Pour cela, il devra collecter divers objets et surtout tuer un grand nombre de monstres.

[26] L'US Air Force a ainsi travaillé avec Spectrum Holobyte Inc. pour réaliser un entraîneur simplifié à partir du logiciel Falcon 4.0.

2.4.2 Entraînement sur matériels réels

Prenons un peu d'avance sur le chapitre 3, et considérons l'entraînement au pilotage d'un char AMX30/30B2, matériel pour lequel nous disposons de 30 années de retours d'expérience. Une étude de la société ABAK Systèmes, réalisée au profit de l'Armée de Terre, évaluait les économies réalisées par l'utilisation de simulateurs d'entraînement (voir [ARNO97]).

Il faut savoir qu'un char (tout comme un avion de combat) n'est pas seulement coûteux à l'achat, mais également à l'utilisation. Ainsi, le moteur V12 de 24 litres de cylindrée de l'AMX30 consomme environ 200 l au 100 km. Un obus d'exercice coûte 3 600F[27], et un obus flèche (perforant) près de 10 000 F. En outre, le canon doit être changé tous les 800 coups. Globalement, sans tenir compte des munitions, on peut estimer qu'un AMX30B2 coûte près de 20 000 F par heure de fonctionnement (imaginez un Mirage 2000 !).

Les simulateurs de tir peloton, déjà anciens puisque entrés en service en 1984, sont implantés à Canjuers et Saumur. Ils fonctionnent 16 heures par jour, 40 semaines par an. Ces simulateurs ont coûté à l'achat 200 MF, soit de l'ordre de trois fois le coût d'acquisition d'un nombre équivalents de chars, plus un montant similaire pour l'entretien, la maintenance, les évolutions et le fonctionnement.

Cependant, leur coût horaire est de 6 000 F, soit trois fois moins que le matériel réel, sans compter le coût bien plus important des munitions. Même en effectuant deux fois plus d'entraînement sur simulateur que sur le terrain, ce système permet d'économiser environ 500 MF par an.

Certes, on tombe à 30% de tirs « live », mais le simulateur permet également une instruction plus encadrée, plus méthodique, avec par exemple un contrôle sur la situation tactique, ou la possibilité de faire des mesures précises des temps de réaction et de la précision.

Une alternative possible est le remplacement des obus d'exercice par des rayons lasers. On supprime ainsi la source la plus importante de dépenses. Ce système est par exemple utilisé au centre d'entraînement tactique de Mailly (CENTAC). Une instrumentation adéquate des chars évoluant sur le terrain permet en outre aux instructeurs de visualiser le déroulement de l'exercice sur des consoles graphiques.

Pour l'instruction au pilotage, un calcul similaire donne une économie annuelle de l'ordre de 93 MF.

Les nouveaux simulateurs pour le char Leclerc augmentent encore l'efficacité de l'entraînement, par le réalisme de l'environnement synthétique et la possibilité de travailler en peloton (simulation distribuée).

Un exemple encore plus frappant est l'exercice américain « Return of Forces to Germany », qui impliquait en 1988 environ 97 000 individus, 7 000 véhicules et 1 080 chars, pour un coût de 54 millions de dollars (de l'époque !), dont presque la moitié pour indemniser l'Allemagne des dégâts causés à son environnement. En 1992, l'utilisation de simulations informatiques permit de limiter le déploiement à 16 500 troupes, 150 véhicules blindés et… aucun char, pour un coût de 21 millions de dollars. Les dégâts causés ne dépassèrent pas 250 000 dollars.

[27] Il s'agit de coûts aux conditions économiques de 1992 .

Bien entendu, il ne faut pas tomber dans le piège du « tout virtuel » : le simulateur ne permet pas d'obtenir un réalisme parfait, même si les techniques s'améliorent. Les entraînement sur le terrain restent indispensables, simplement il est possible de les limiter, et, partant de là, de diminuer les coûts à compétence acquise supérieure ou égale. Sans compter les avantages annexes mais non négligeables de la simulation : impact écologique réduit (ceux qui ont pu assister à un entraînement d'artillerie à Canjuers comprendront ce que je veux dire par là !), risque d'accident quasiment nul, etc.

2.4.3 Expérimentations, maquettes

Quand la simulation est utilisée à des fins d'ingénierie, il est possible, à défaut du matériel réel, d'utiliser des maquettes. L'exemple typique est celle de l'écoulement aérodynamique autour d'un missile (ou d'un avion, ou d'une voiture, etc.).

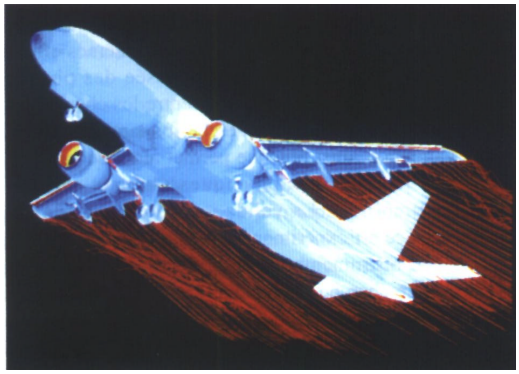


Figure 41: Simulation numérique de l'écoulement de l'air autour d'un avion de type Airbus au décollage (ONERA)

Prenons donc le document [ARNO97], dans lequel les auteurs, des anciens élèves de l'ENSIETA, se sont attachés à comprendre l'intérêt de la simulation numérique (méthodes d'Euler ou de Navier-Stokes) par rapport à des essais en soufflerie[28].

Rappelons d'abord que le prix d'un missile réel (non nucléaire) est de l'ordre de quelques millions de francs. Quant à un prototype, il peut coûter beaucoup plus cher que cela !

Le coût de la réalisation d'une maquette est de l'ordre de 150 kF à 1 MF. L'utilisation d'une soufflerie est facturée de l'ordre de 100 kF par jour, et il faut quelques jours typiquement pour effectuer les essais.

Figure 42: Maquette d'airbus a330-200 à la soufflerie S1 de Modane (ONERA)

Pour effectuer une simulation, il faudrait numériser le missile, en réalisant un maillage de ses contours (comptons 50 kF pour l'opération), et louer un calculateur tel qu'un CRAY (quelques kF de l'heure, la durée d'utilisation dépendant de la précision du maillage et du nombre de variables étudiées).

Le gain est important, sans compter que dans certains domaines l'empirisme règne. Ainsi, pour prendre un objet plus courant qu'un missile, considérons une enceinte acoustique. Vous avez dû remarquer une grande variété dans les formes et les matériaux des enceintes haute-fidélité. En fait, les constructeurs « testent » des concepts, avec plus ou moins de réussite. La mise au point est grandement empirique, car on maîtrise mal les phénomènes de propagation et de

[28] Certains considèrent que les essais en soufflerie sur maquette sont de la « simulation analogique ». Je n'emploiera pas ce terme dans ce contexte. Notons que dans les années 60, des machines analogiques ont été très utilisées à des fins de simulations. Le numérique les a remplacées, étant moins onéreux et plus souple.

réflexion du son à l'intérieur d'une enceinte, qui sont compliqués par les interactions avec le sol et les murs proches, les résonances de la caisse, etc.

Des sociétés comme Cabasse (basée à Brest) se lancent donc de plus en plus dans la mise au point de modèles numériques afin de diminuer le nombre de prototypes nécessaires pour mettre au point un produit commercialisable, et par là même non seulement diminuer les coûts de conception, mais aussi les délais, critère sensible dans un domaine soumis à une sévère concurrence.

Note importante

La simulation ne peut totalement remplacer les essais. En effet, la simulation a besoin de données d'entrée, de paramètres. Et ces données proviennent généralement… d'essais ! Les deux techniques sont donc complémentaires : les essais nourrissent les simulations, et celles-ci permettent de limiter et d'optimiser les essais.

Ainsi, lorsque la France a décidé d'arrêter définitivement les essais nucléaires, il a fallu se résoudre à effectuer quelques essais supplémentaires (avec le coût politique que l'on sait) afin d'affiner les modèles théoriques. Sans eux, il n'aurait probablement pas été possible de faire des simulations numériques fiables.

2.4.4 Méthodes formelles

La complexité et la part croissante du logiciel dans les systèmes modernes a conduit au développement d'une nouvelle branche de l'ingénierie : le **génie logiciel**. (voir page 171).

Des méthodes de spécifications et de développement (HOOD[29], SADT, OMT, Merise…) ont amélioré le processus de production de logiciels, mais leur rigueur reste insuffisante pour assurer la fiabilité de logiciels critiques, tels que ceux utilisés dans l'aérospatial, le nucléaire, les télécommunications, les transports, les transactions bancaires, etc.

Pour aller plus loin, de nouvelles techniques ont été développées, particulièrement dans les années 80-90, bien que les travaux sur le sujet aient commencé bien avant : les **méthodes formelles**.

Ces méthodes permettent de déroulement d'une ou plusieurs étapes du cycle de développement d'un logiciel avec une grande rigueur et une grande précision, notamment par l'utilisation d'un **langage formel**.

Il existe de nombreux langages formels, basés généralement sur un formalisme mathématique ou logique (prédicats, algèbres, graphes, réseaux de Pétri…) :

- Langages orientés objets : Z, VDM, B…

[29] Utilisé notamment à l'agence spatiale européenne (ESA), Arianespace, CNES, Eurocopter, Matra Marconi Space, COGEMA… Il existe maintenant des outils de génération automatique de code (Ada95…) à partir des spécifications HOOD.

- Langages de description de comportement : algèbres de processus (CCS, CSP), langages de description de protocoles (SDL[30], LOTOS, Estelle), langages synchrones pour décrire des systèmes réactifs (LUSTRE[31], ESTEREL[32]…).

Les domaines d'applications de ces langages étant souvent spécifiques, il en existe bien d'autres (RSL, SIGNAL, TRIO…), la liste ci-dessus est donc loin d'être exhaustive.

Ces outils peuvent être utilisés à différents moments du cycle de développement (voir [OFTA97]) :

- Validation d'un cahier des charges
- Aide à la rédaction de spécifications
- Vérification des spécifications[33] (cohérence, complétude...)
- Aide à la conception (voire génération de code)
- Tests (génération d'un arbre de tests pouvant être exhaustif)
- Génération d'une maquette ou d'un simulateur du système
- ...

L'utilisation des méthodes formelles est une tâche parfois ardue, mais qui permet la réalisation de logiciels « sûrs »[34]. Ainsi, le logiciel du métro automatique parisien METEOR (ligne 14) comporte-t-il des parties conçues en langage B.

Les méthodes formelles peuvent être utilisées en remplacement ou en complément des techniques de simulation numérique.

2.4.5 Méthodes quantitatives

Pour l'optimisation de certains problèmes (organisation, prise de décision, planification d'itinéraires...), des méthodes dites « quantitatives » ont été développées dès les années 50 et 60.

L'une de ces méthodes, qui connaît un certain succès actuellement, est la **programmation par contraintes** (PPC, voir [BART98]). Elle permet la description et la résolutions de problèmes combinatoires complexes, notamment dans les domaines de la planification. Ces problèmes sont exprimés sous la forme d'exigences[35] (par exemple, pour optimiser un déplacement en car : « Un car XYZ ne peut transporter plus de 20 personnes, « On ne doit pas faire circuler un car le dimanche », « Le chauffeur doit faire une pose de 10 mn toutes les 2 heures », etc.), et la solution est calculée par des solveurs de contraintes spécialisés.

---

[30] Utilisé pour des applications de télécommunication, et par l'ESA pour spécifier des composants logiciels critiques.

[31] Utilisé par Scheider Electric pour les logiciels de contrôle des centrales nucléaires, ainsi que par Aerospatiale Aéronautique pour des composants logiciels embarqués.

[32] Utilisé par Dassault Aviation pour le Rafale et par Thomson CSF pour les radars.

[33] L'auteur du présent document a d'ailleurs utilisé en 1993 le langage synchrone ESTEREL pour la vérification des spécifications d'un calculateur embarqué sur l'Airbus A340, pour le compte de la société Aerospatiale

[34] Un programme de plus de quelques centaines de lignes n'étant plus démontrable dans l'absolu, on entends par « logiciel sûr » un programme ayant un faible taux d'erreur, typiquement 10 fois moins élevé que la norme (par exemple 0,1 erreur pour mille lignes de code pour le logiciel embarqué de la navette spatiale).

[35] Ces exigences sont des contraintes sur les variables, c'est-à-dire des règles restreignant les valeurs que peuvent prendre ces variables. Ainsi, la variable « nombre de personnes transportées par rotation du car » est limitée à 20 dans notre exemple.

Issue des travaux en intelligence artificielle des années 70-80, elle a des applications aussi variées que :

- Les bases de données (contrôle de la cohérence des données)
- Imagerie numérique (cohérence géométrique lors de l'analyse d'une scène)
- Traitement du langage naturel
- Biologie moléculaire (séquencement de l'ADN)
- Courtage d'actions en bourse
- Conception de circuits électroniques
- Recherche opérationnelle
- …

C'est dans le domaine de la recherche opérationnelle (optimisation d'organisations, de processus, de parcours…) que la PPC peu concurrencer la simulation, bien qu'elle souffre de nombreuses limitations.

La PPC est ainsi assez utilisée dans les transports (SNCF, British Airways…), permettant l'optimisation des rotations de matériels ou des créneaux horaires. Chez les militaires, elle peut être parfois une alternative à la simulation pour les études technico-opérationnelles (par exemple pour des études en logistique, projection de forces…).

2.4.6 Complémentarité avec la simulation

Je l'ai déjà dit, mais j'y reviens une fois de plus, car c'est un point réellement fondamental, et il faut éliminer toute confusion à ce sujet.

La simulation et les méthodes que nous venons de voir sont **complémentaires**.

En effet, ces méthodes ont toutes leurs limites, et parfois la simulation est le seul moyen de mener à bien une étude.

D'un autre côté, la simulation a besoin des essais réels. En effet, un modèle nécessite des données (voir §5.1.1.4) en entrée, de la précision desquelles va dépendre celle du modèle. En outre, il faut bien avoir un référentiel de comparaison pour pouvoir valider les résultats d'une simulation.

Ces données peuvent parfois être calculées, mais sont souvent issues, directement ou indirectement, d'expérimentations réelles ou sur maquettes (essais en soufflerie…).

On ne simule bien que ce que l'on comprend bien.
Sans le réel, il ne peut donc y avoir de virtuel.

3. EXEMPLES D'EMPLOI

3.1 Généralités

Il est difficile de présenter tous les emplois possibles et imaginables de la simulation. Néanmoins, nous allons dans ce chapitre voir les cas d'utilisation les plus typiques, sans avoir la prétention d'être exhaustif : le virtuel n'a d'autres limites que celle de notre imagination.

La simulation militaire tient ici une place importante, mais il faut savoir que si la défense est particulièrement moteur dans certains domaines de la simulation, tels que les simulateurs pilotés ou les simulations distribuées, il ne faut pas pour autant sous-estimer le civil : c'est en effet chez celui-ci que l'on trouve les partisans les plus convaincus de la simulation, pression des coûts et de la compétition technique et commerciale oblige !

Il faut tout de même reconnaître aux militaires un rôle clef dans le domaine : leur volonté de mettre de l'ordre dans un domaine culturellement et techniquement très diversifié, et de créer des standards, y compris au niveau méthodologique, dont les civils devraient à terme tirer des bénéfices significatifs[36].

3.2 Comportement d'un parachute

Les études amont sont indispensables aux programmes d'armements, car elles permettent de faire les meilleurs choix techniques pour les systèmes concernés, et de maintenir à jour une capacité technique, de façon à pouvoir répondre aux besoins à long terme. Elles occupent en outre une place importante dans la politique de coopération.

L'exemple typique de simulation au profit des études amont est la simulation scientifique.

Ce type de simulation, très utilisé à la DCE[37], permet, entre autres, d'étudier et de valider des choix de conception, en réduisant ou évitant des essais réels souvent coûteux et parfois même dangereux, tout en offrant une grande souplesse.

[36] Ainsi, on commence déjà à voir des applications civiles du standard HLA, telles que le démonstrateur virtuel de l'ATV, le véhicule de transfert automatique européen qui sera chargé de ravitailler la station spatiale Alpha.
[37] Direction des Centres et moyens d'Essais.

En simulant le comportement d'un parachute placé dans un écoulement d'air, par un calcul couplé fluide/structure (voir Figure 43), le CAP[38] peut ainsi comprendre la dynamique du vol et calculer les performances aérodynamiques du système.

Cette représentation virtuelle, qui prend en compte les déformations dynamiques de la voile du parachute, permet d'acquérir tout un ensemble de données sur l'écoulement de l'air qu'il serait très difficile de mesurer à l'aide de capteurs lors d'un essai réel ou sur une maquette en soufflerie.

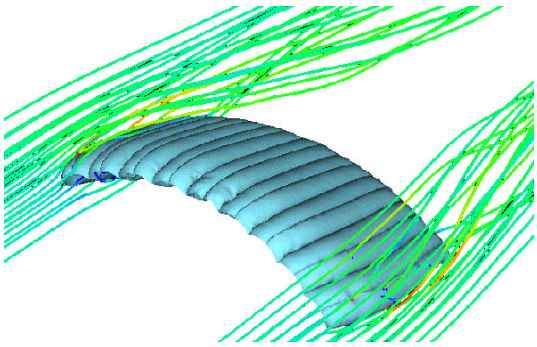


Figure 43: Écoulement des filets d'air autour d'un parachute (DGA/DCE/CAP)

Notamment, il est possible de s'intéresser à une particule d'air particulière pour étudier son cheminement. Ainsi, on peut voir qu'une particule qui entre dans un caisson extérieur traverse la paroi inter-caissons par les ouvertures (voir Figure 44), passe dans le deuxième caisson et s'échappe de l'extérieur du profil vers le bord de fuite (voir Figure 45). La couleur de la trajectoire est fonction de la vitesse de la particule.

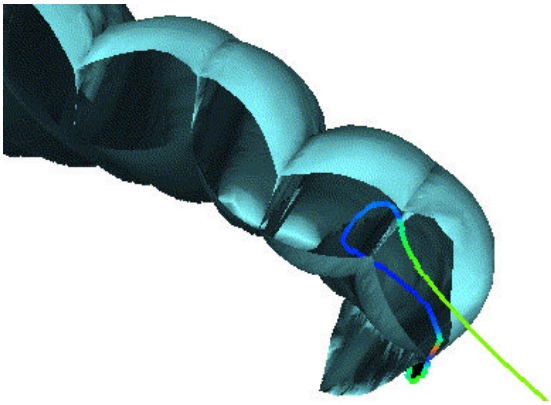


Figure 44: Cheminement d'une particule (face intérieure du parachute) (DGA/DCE/CAP)

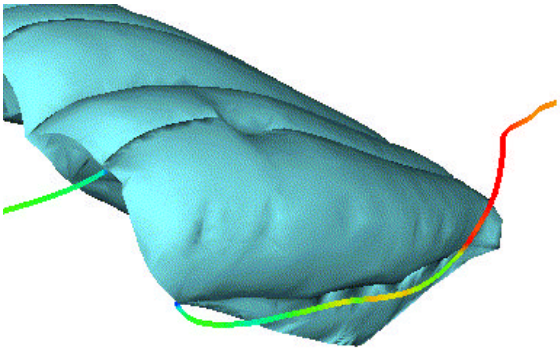


Figure 45: Cheminement d'une particule (face extérieure du parachute) (DGA/DCE/CAP)

Dès lors que l'on a un modèle numérique du système étudié (en l'occurrence le parachute), on peut le réutiliser à volonté. La Figure 46 montre ce même modèle de parachute, lui-même basé sur un modèle issu d'un logiciel de CAO[39], permettant, sans traitement supplémentaire, d'étudier les contraintes dans le tissu et les suspentes du parachute.

[38] Centre Aéroporté de Toulouse (DGA/DCE).
[39] Conception Assistée par Ordinateur.

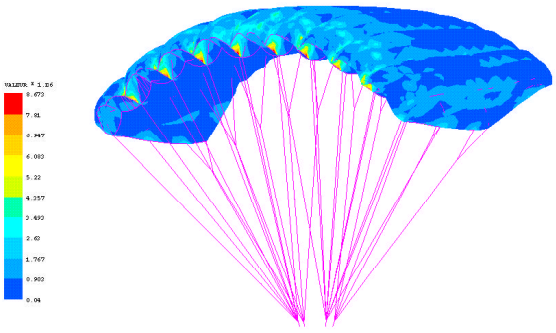


Figure 46: Contraintes de Von-Mises dans le tissu (DGA/DCE/CAP)

La simulation permet donc ici de voir ce qu'il serait difficile de mettre en évidence expérimentalement.

3.3 Simulation d'appontage d'un hélicoptère

Une étude d'un groupe de travail naval OTAN[40] vise à développer des prototypes virtuels afin d'effectuer des simulations d'appontages et de décollages de drônes (UAV) et d'hélicoptères. En effet, un navire est une plate-forme mobile, ce qui rend extrêmement délicats pour les pilotes d'hélicoptère les opérations d'atterrissage, surtout par forte mer. Le problème est encore plus aigu pour les appareils automatiques tels que les drônes.

La méthode classique consiste à passer par des maquettes ou des prototypes, que l'on instrumente et que l'on soumet aux actions d'une plate-forme mobile de test (voir Figure 47). On image aisément le coût d'un tel processus.

Figure 47: maquette d'UAV sur plate-forme mobile (Daimler-Benz Aerospace)

La vision de la ST-SBDVP est de pouvoir concevoir, construire, tester, entraîner et utiliser un système d'armes de façon virtuelle avant de commencer la fabrication.

Pour cela, il faut développer des prototypes virtuels validés, interopérables entre pays de l'OTAN, et être capables de les gérer tout au long du cycle de développement du produit. En bref, l'objectif de ce projet est de faire la démonstration des apports du SBA.

Figure 48: Prototype virtuel d'UAV

[40] Il s'agit du NG/6, groupe naval sur la conception des navires, et plus exactement de la « Specialist Team on Simulation Based Design and Virtual Prototyping ». 13 nations participent à cette ST-SBDVP.

L'Allemagne se chargea de développer un prototype virtuel d'UAV (voir Figure 48). Les français et les britanniques se chargèrent des modèles de calculs mécaniques des mouvements du navire et de l'interface entre le mobile aérien et le pont.

Ceci n'avait rien d'éviter, car de nombreux composants ainsi que des phénomènes physiques complexes (vagues, vent…) entrent en ligne de compte. Le modèle conceptuel de la Figure 49 montre bien les interactions en jeu.

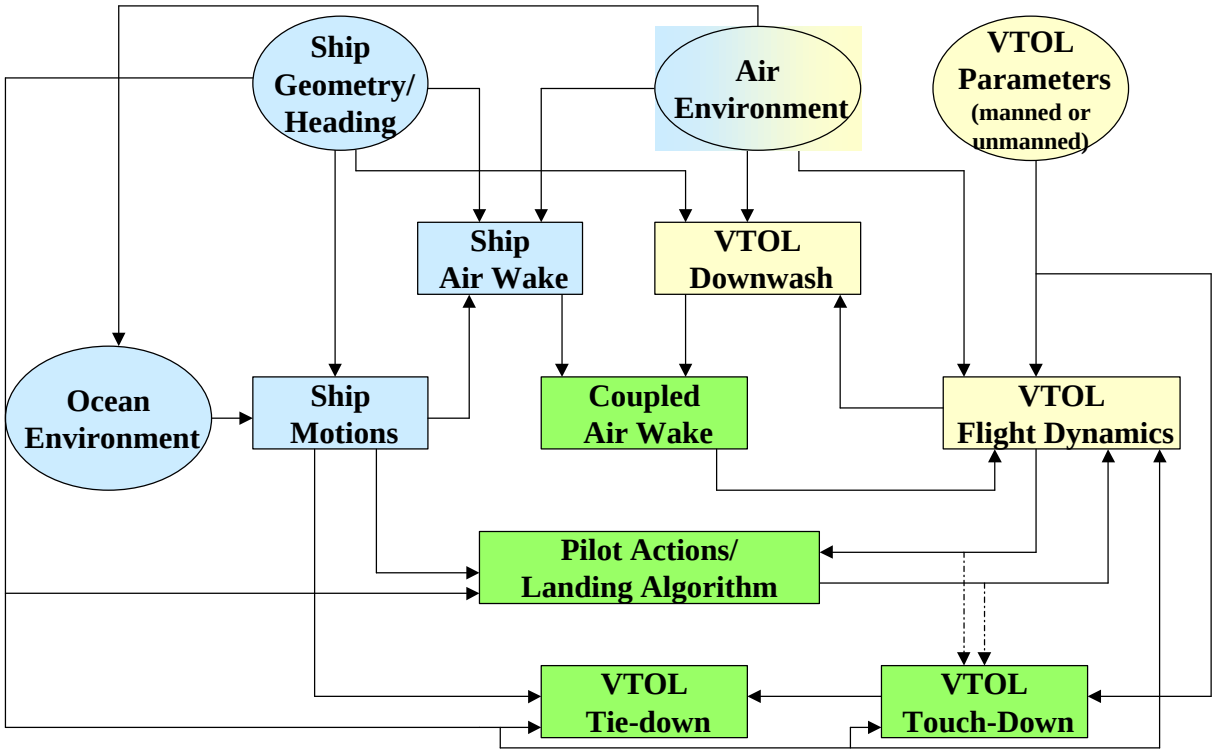


Figure 49: Modèle conceptuel de l'appontage et de l'arrimage d'un véhicule VTOL

Les différents éléments de ce modèle ont été implémentés comme autant de fédérés interopérant via le standard HLA, au sein d'une fédération baptisée NIREUS (*NATO/PfP Interoperability and Reuse Study*). Chaque modèle peut alors être développé indépendamment, ce qui est particulièrement intéressant dans le cadre de coopérations internationales, lesquelles sont devenus monnaie courante en raison des coûts de développement et de la complexité des systèmes modernes et, il faut bien le dire, une tendance à la mondialisation de l'économie favorisant les regroupements d'entreprises de nations différentes.

NIREUS est une bonne démonstration de l'apport de la simulation distribuée dans ce contexte multinational : le projet global comprend 4 équipes (intégration, navire, véhicule aérien, interfaces navire/véhicule), chacune étant elle-même multinationale. Avec une conception « classique » de la simulation (intégration au niveau code des différents modèles), il aurait été très difficile de mettre au point le logiciel final. Grâce à HLA, qui fournit d'emblée la méthodologie et les interfaces nécessaires, l'intégration se fait à un plus haut niveau, ce qui la facilite considérablement.

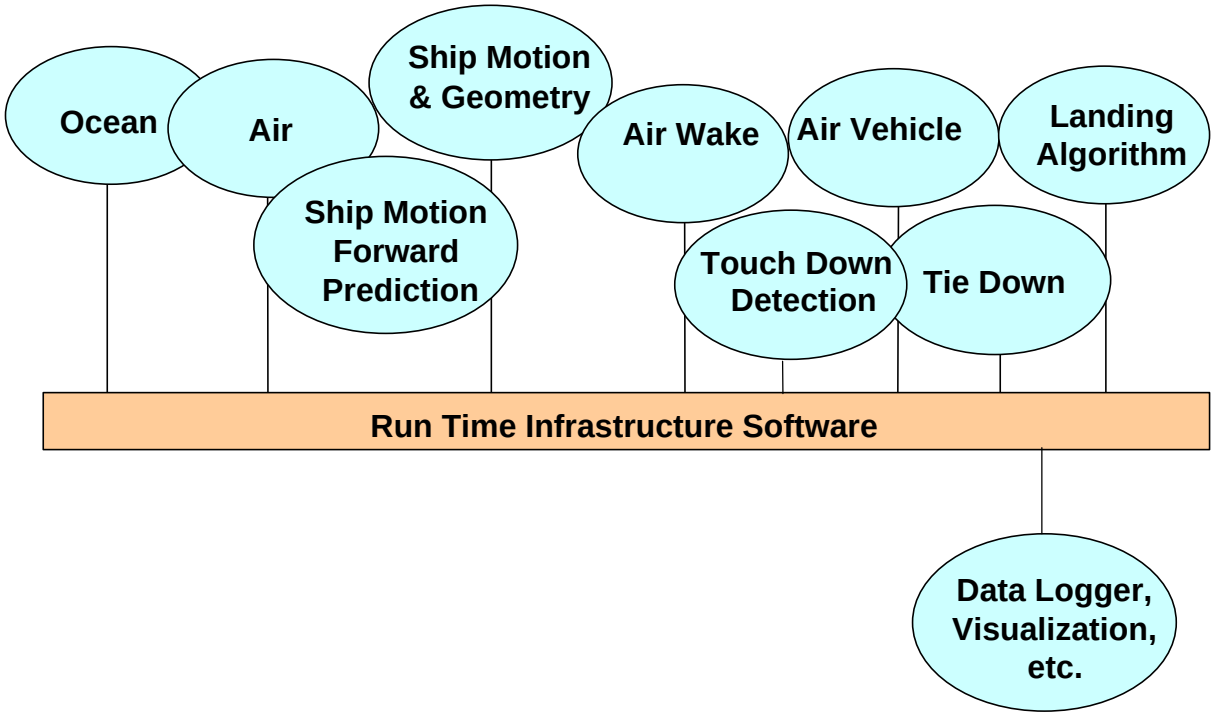


Figure 50: Fédération NIREUS

Cette démonstration des possibilités du prototypage virtuel et de la simulation distribuée, qui devrait s'achever fin 2001, est considérée par beaucoup de nations, dont la France (CTSN[41]), comme un projet pilote, préfigurant ce qui est probablement l'avenir de l'ingénierie de systèmes (navals ou autres).

Figure 51: Prototype virtuel de NH90 à l'atterrissage

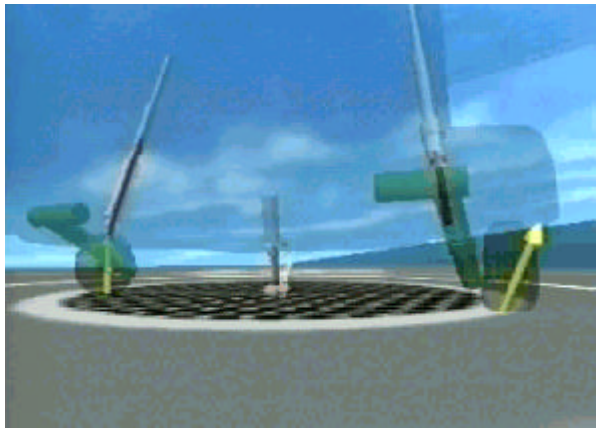


Figure 52: Visualisation des efforts sur le train d'atterrissage

[41] Centre Technique des Systèmes Navals (établissement de la DCE installé à Toulon).

3.4 Joint Theater-Level Simulation (JTLS)

Les jeux de guerre (*wargames*) sont particulièrement utilisés pour l'entraînement du commandement militaire (états-majors…), sans devoir mobiliser des effectifs importants sur le terrain (parfois des milliers de personnes, voir page 59). En France, l'état-major de l'armée de terre utilise ainsi des simulations américaines, Janus et BBS. Ces jeux de guerre peuvent aussi servir pour les études technico-opérationnelles (RUBIS set utilisé à cet effet au CAD), les études de doctrine, ou la planification.

Le développement de JTLS a commencé en 1983, pour le compte de l'US Army. Il s'agissait de développer une simulation interarmes et interarmées de niveau théâtre[42], pour l'entraînement du commandement, l'aide planification et l'analyse[43].

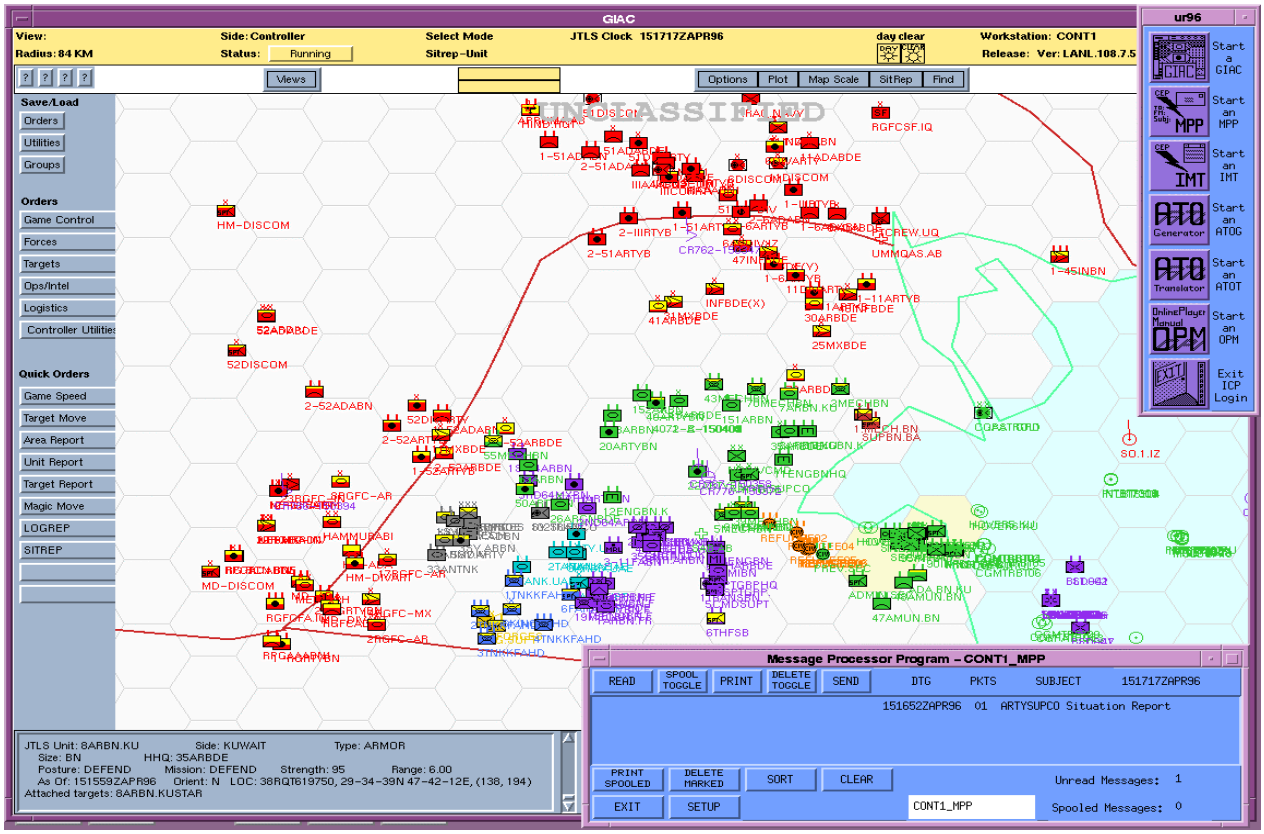


Figure 53: Jeu de guerre JTLS

JTLS prend en compte les forces aériennes, terrestres (y compris le génie) et maritimes, mais aussi les forces spéciales, le renseignement (*intelligence*), y compris le renseignement électronique, ainsi que la logistique et les communications.

JTLS est multi-joueur, c'est-à-dire qu'il peut avoir plus de deux camps. Ceci est une qualité très importante de nos jours. En effet, jusqu'à une période récente (fin des années 80), on ne considérait généralement dans les conflits que deux camps : les bleus (amis) et les rouges (ennemis), parfois appelés « azur » et « orange » pour ménager les susceptibilités. Or, les conflits modernes de niveau théâtre, tels que la guerre du Golfe

[42] Le public visé étant les CINC (Commander-In-Chief) et les états-majors de JTF (Joint Task Force).

[43] Notamment pour étudier le déroulement et les résultats d'une bataille passée.

ou l'intervention de l'OTAN en Yougoslavie, sont souvent des guerres de coalition, dans lesquelles plusieurs camps sont alliés, mais peuvent avoir des motivations ou des limitations propres. Ainsi, tel pays peut refuser toute opération contre un groupe ethnique donné. Ceci a bien entendu un impact important sur la conduite des opérations, et doit être pris en compte dans les simulations modernes. En outre, le développement important des « opérations autres que la guerre »[44] tels que le maintien de la paix au profit de l'ONU (Yougoslavie, Liban, Timor…) ou les interventions humanitaires (Rwanda…) requièrent également la prise en compte d'un ou plusieurs camps « neutre », représentant les populations civiles, qui jouent un rôle majeur dans ce genre de d'opération.

Dans JTLS, il peut y avoir jusqu'à 10 camps, chacun pouvant même être divisé en un nombre illimité de factions dont les allégeances peuvent être changées dynamiquement au cours du jeu, de même que les relations entre les camps. Le terrain est divisé en zones hexagonales (voir page 141). Plus de 300 utilisateurs (joueurs) peuvent être impliqués dans une simulation (voir Figure 54). JTLS peut être distribué sur un réseau local (LAN) ou distant (WAN). Chaque joueur dispose d'une station de travail avec une interface graphique et une messagerie avec laquelle il peut interagir avec les autres participants.

Il est également possible d'intégrer des participants réels sur le terrain (simulation dite « live » ou « instrumentée »).

Figure 54: Une salle pour les exercices JTLS

Le terrain peut faire jusqu'à 1800x1800 km. Il est décrit à l'aide… d'hexagones (voir Figure 53), comme les jeux de guerres du commerce ! Toutefois, les objets de la simulation peuvent être placées n'importe où, et pas seulement au centre des hexagones. Ce terrain influe bien entendu sur les mouvements et les interactions entre les unités.

JTLS est écrit essentiellement avec le langage SIMSCRIPT II.5 et tourne sur des stations de travail Sun SPARC relativement modestes.

[44] Opérations dites OOTW (*Operations Other Than War*). Les armées sont souvent démunies sur le plan doctrinal devant ce type d'intervention, très éloigné du schéma traditionnel (et clair) qui a prévalu durant la Guerre Froide, de sorte que les OOTW sont devenues l'une des priorités des développements en matière de simulation au sein de l'OTAN.

JTLS est utilisé dans de nombreux centres américains[45], OTAN, sud-coréens, britanniques, japonais, grecs, et au collège interarmées de défense (CID) en France.

Il fait actuellement l'objet d'une modernisation, notamment pour sa mise en conformité avec le standard HLA.

À côté de JTLS, le DoD américain a lancé d'ambitieux programmes de simulation interarmées, tels que JSIMS (fédération d'entraînement pour les commandants en chef de forces interarmées, pouvant servir pour la planification ou l'aide à la conduite d'opération, mise en service prévue en 2003), doté d'un budget dépassant le milliard de dollars. La composante terrestre de la fédération JSIMS, WARSIM 2000, servira également à l'entraînement interarmes des états majors de l'US Army (niveaux bataillon à corps d'armée), remplaçant plusieurs jeux de guerre existant, anciens et devenus difficiles à maintenir. WARSIM aura la possibilité d'être connecté à des C4I ou des matériels sur le terrain (simulation « live »). Une autre fédération comparable à JSIMS, JWARS, servira aux fins d'analyse de défense (études technico-opérationnelles, doctrine...).

3.5 Simulateur d'hélicoptère

L'exemple typique de simulation d'entraînement est celui du simulateur de vol. Certes, de tels dispositifs sont souvent aussi coûteux à l'achat qu'un appareil réel, mais l'heure de vol revient vingt à trente fois moins cher en virtuel.

Les simulateurs de vol reproduisent l'environnement du pilote (cockpit, réactions physiques de l'appareil, etc.), d'une façon plus ou moins fidèle suivant les besoins.

La montre un exemple de simulateur d'hélicoptère (Puma et Cougar) haut de gamme, destiné à l'école d'application de l'ALAT[46]. Il est constitué d'une cabine, réplique exacte du cockpit (voir Figure 55), montée sur des vérins hydrauliques qui lui donnent six degrés de liberté, permettant ainsi un grand réalisme dans la reproduction des mouvements de l'appareil. L'environnement visuel, issu de générateurs d'images, est projeté sur une sphère, donnant un champ visuel horizontal de 200°. Un processeur numérique est, lui, chargé de reproduire l'ambiance sonore de la cabine. Tout est donc fait pour immerger les pilotes dans un environnement virtuel le plus réaliste possible.

[45] Le maître d'ouvrage actuel est le Joint Warfighting Center à Fort Monroe, en Virginie.
[46] Aviation Légère de l'Armée de Terre.

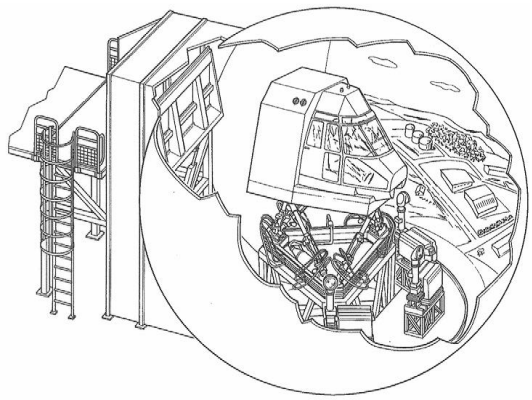


Figure 55: Écorché d'une sphère SHERPA (SOGITEC)

Figure 56: Vue du cockpit du simulateur d'hélicoptère SHERPA (SOGITEC)

Tout, ou presque, devient alors possible sans risques pour les entraînés. L'instructeur peut, depuis son pupitre, déclencher des pannes, modifier la météo et l'heure (pour passer en vol de nuit), changer le terrain ou la situation tactique, etc.

Si le simulateur ne remplace pas totalement le vol réel, il permet néanmoins de fortes économies sur la formation des pilotes, tout en l'améliorant, puisqu'ils pourront s'entraîner plus longuement, et expérimenter des situations difficilement reproductibles dans la réalité. La simulation est désormais devenue un élément incontournable de la formation des pilotes, tant civils que militaires.

3.6 Simulation de trafic automobile

(d'après [CAST97]).

Au début des années 90, un chercheur américain de Los Alamos, Chris Barrett, modélisa une ville virtuelle complète, copie virtuelle de celle d'Albuquerque (Nouveau-Mexique), avec ses 200 000 foyers et 400 000 voyageurs quotidiens se déplaçant sur 30 000 tronçons de routes. Cette simulation des mouvements quotidiens de la population, TRANSIMS, permettait non seulement d'étudier les embouteillages et de tester des projets d'urbanisme (aménagement des routes, construction d'un pont...) mais aussi de mener des études environnementales, car elle intégrait un module de simulation de pollution atmosphérique.

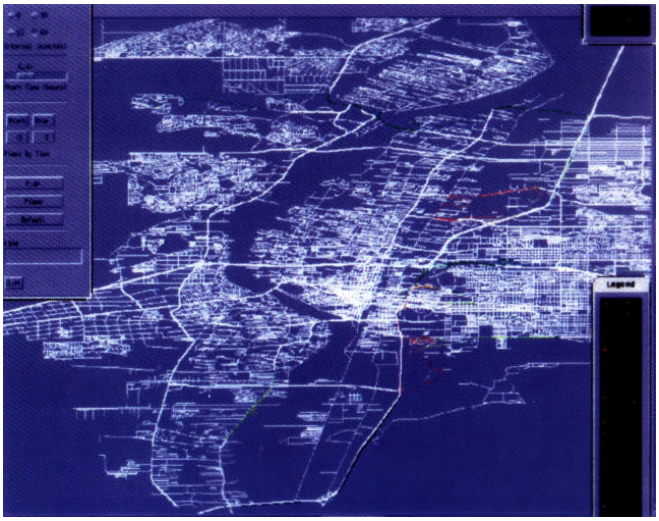


Figure 57: Alburquerquia in the morning...

Cette simulation était extrêmement ambitieuse, surtout pour l'époque, puisque chacun voyageur était modélisé individuellement, planifiant et exécutant son itinéraire propre. Le raffinement de la modélisation allait jusqu'à distinguer plusieurs type de véhicule et divers comportements (par exemple volonté de choisir l'itinéraire le plus rapide ou le plus confortable).

À chaque pas de simulation, représentant une seconde de temps réel, le moteur de simulation exécutait les déplacements calculés précédemment, et calcule pour chaque véhicule les taux d'émission de gaz polluants (monoxyde de carbone, oxyde d'azote, dioxyde de soufre).

La Figure 58 donne l'enchaînement des différents modules de TRANSIMS. Le rebouclage entre la simulation environnementale et la définition de la structure du réseau de transport s'effectue manuellement, à partir d'hypothèses faites sur les possibles aménagements d'urbanisme susceptibles d'améliorer les résultats.

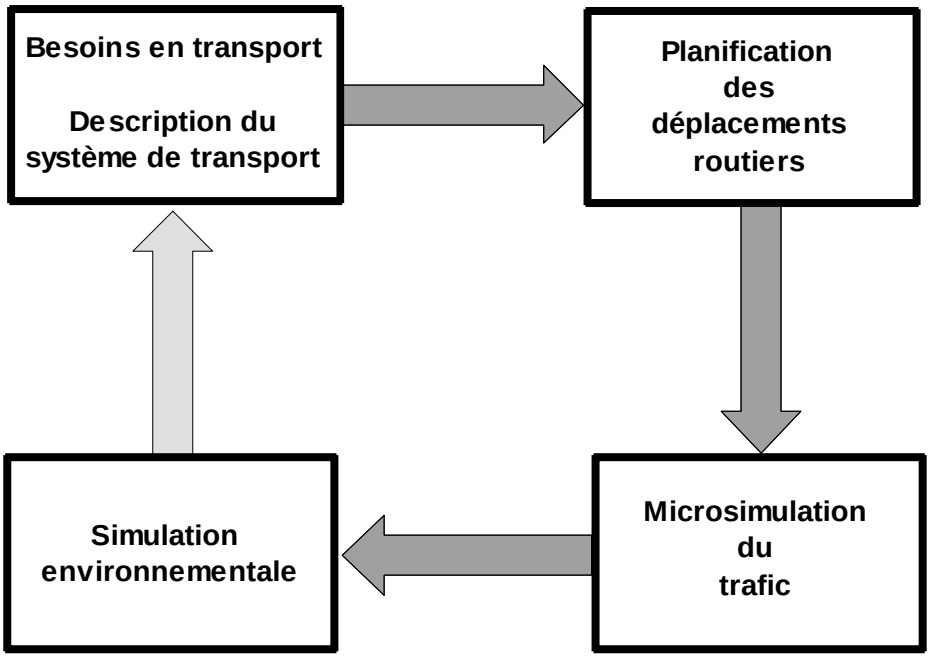


Figure 58: Structure de TRANSIMS

4. RAPPELS MATHEMATIQUES

4.1 Introduction

Albert Einstein disait : « Dieu ne joue pas aux dés ». En effet, il n'acceptait pas que la nature puisse avoir des comportements aléatoires, tel que l'émission d'une particule atomique lors de la désintégration de noyaux instables, phénomène notoirement imprévisible.

Dans notre vie, bien des événements semblent dépendre du hasard : les catastrophes naturelles, les accidents, les pannes, l'amour…

Toutefois, il y a parfois de l'ordre dans ce hasard : si l'on ne peut prédire à quel moment exact un noyau atomique va se désintégrer, il existe des lois physiques qui permettent de déterminer la période où cet événement a plus de chance de se produire.

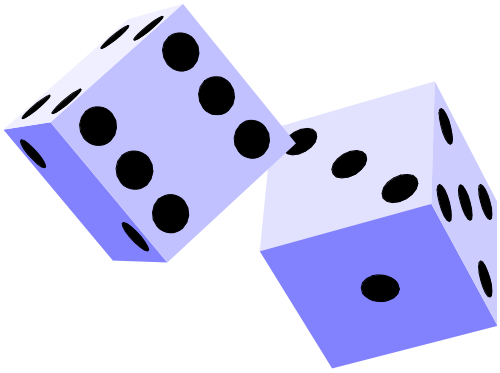


Figure 59: Si l'utilisation du hasard semble naturelle dans la vie de tous les jours, il n'en va pas de même dans un programme…

Nous allons voir dans ce chapitre quelques-unes des lois qui régissent le hasard, et comment simuler ce hasard. Pour cela nous nous baserons sur des études statistiques.

La **statistique** est en effet la branche des mathématiques appliquées, dont les principes découlent de la **théorie des probabilités**, et qui a pour objet le groupement méthodique ainsi que l'étude de séries de faits ou de données numériques (d'après le Petit Larousse).

4.2 Variables aléatoires

4.2.1 Définition

Une variable est dite **variable aléatoire** lorsque la grandeur qu'elle mesure dépend du hasard.

Exemples :

– Le résultat du lancer d'une pièce de monnaie (pile ou face). On peut associer à cet événement la valeur 1 pour pile et 0 pour face (**loi de Bernouilli**).

– Le résultat du jet de deux dés (somme des deux dés) :

x	2	3	4	5	6	7	8	9	10	11	12
n	1	2	3	4	5	6	5	4	3	2	1

x : valeur du tirage
n : nombre de combinaisons des deux dés permettant d'obtenir ce tirage

– Le bruit dans une communication radio.

– L'instant où se produit une panne dans un mécanisme.

Un phénomène peut être aléatoire parce qu'il dépend d'un processus chaotique, ou tout bonnement parce que les conditions initiales du processus ou son évolution sont trop complexes pour être calculables (du moins aujourd'hui), de sorte que la prochaine valeur de la variable aléatoire associée est parfaitement imprévisible. Ainsi, l'apparition de tempêtes pouvaient apparaître à une époque comme un phénomène aléatoire, mais grâce à de puissants moyens de calculs, nous savons prévoir le temps à plusieurs jours (avec une certaine marge d'erreur toutefois). En revanche, l'instant où survient la désintégration d'une particule reste parfaitement imprévisible.

4.2.2 Variables aléatoires discrètes

4.2.2.1 Définition

Une variable aléatoire est dite **discrète** si elle ne prend qu'un nombre fini ou dénombrable de valeurs. C'est le cas pour notre paire de dés (11 valeurs possibles).

On appelle **loi de probabilité** ou **distribution** de la variable aléatoire X la fonction associant à une valeur x du domaine de X la probabilité que cette v.a. prenne cette valeur :

$$\boxed{p : x \mapsto p(x) = \Pr\{X = x\}}$$

Propriété : Soit E l'ensemble des valeurs x_i que peut prendre X. On a :

$$\boxed{\forall x_i \in E, \quad 0 \le p(x_i) \le 1 \quad \text{et} \quad \sum_{i=1}^{card(E)} p(x_i) = 1}$$

4.2.2.2 Fonction de répartition

On appelle **fonction de répartition** (ou de « probabilités cumulées ») d'une variable aléatoire X la fonction associant à une valeur x du domaine de X la probabilité que cette v.a. prenne une valeur strictement inférieure :

$$\boxed{F : x \mapsto F(x) = \Pr\{X < x\}}$$

Reprenons l'exemple de nos deux dés et complétons le tableau :

x	2	3	4	5	6	7	8	9	10	11	12	13
n	1	2	3	4	5	6	5	4	3	2	1	0
$p(x)$	3%	6%	8%	11%	14%	17%	14%	11%	8%	6%	3%	0%
$F(x)$	0%	3%	9%	18%	29%	43%	59%	73%	84%	92%	97%	100%

X : variable aléatoire « valeur lue sur les dés »

E : ensemble des valeurs (réalisations) de X. E = { 2, 3, 4, …, 11, 12 }

x : une valeur lue sur les dés

n : nombre de combinaisons des dés réalisant x.

$p(x)$: probabilité que les dés sortent la valeur x.

$F(x)$: Probabilité que la valeur des dés soit strictement inférieure à x.

4.2.3 Variables aléatoires continues

4.2.3.1 Définition

Une variable aléatoire est dite **continue** si elle peut prendre toutes les valeurs possibles sur un intervalle continu (i.e. réel). Nous avions pris au début de ce cours l'exemple de la dispersion des balles arrivant sur une cible. La distance du point d'impact à la cible est une v.a. continue (c'est un exemple que j'aime bien, nous y reviendrons…).

On remarquera que la loi de probabilité telle que nous l'avons définie précédemment pour une v.a. discrète (§4.2.2.1) n'a pas de sens pour une variable continue. En effet, il existe une infinité (ou une quantité non dénombrable) de valeurs possibles pour la v.a., de sorte que la probabilité qu'elle prenne une valeur précise est nulle.

4.2.3.2 Fonction de répartition

La **fonction de répartition** d'une variable aléatoire continue est définie comme pour une variable discrète :

$$\boxed{F : x \mapsto F(x) = \Pr\{X < x\}}$$

4.2.3.3 Densité de probabilité

On définit la probabilité que X prennent une valeur dans l'intervalle [a,b[par :

$$\boxed{\Pr\{a \le X < b\} = F(b) - F(a)}$$

Nous définirons la **densité de probabilité** en un point x comme la dérivée de la fonction de répartition en ce point, si toutefois elle existe :

$$\boxed{f(x) = F'(x)} = \lim_{e \to 0}\left[\frac{F(x+e) - F(x)}{e}\right]$$

C'est le concept qui permet de définir la loi d'une variable aléatoire continue de la façon la plus commode.

On remarquera que $\int_a^b f(x)dx = \Pr\{a \le X < b\}$ et que $\int_{-\infty}^{+\infty} f(x)dx = 1$.

Pour bien comprendre ce que représente cette densité de probabilité, une analogie peut être faite avec la densité de la matière : un point n'a pas de masse, mais on définit la densité linéaire, surfacique ou volumique comme la dérivée de la masse en fonction de

x. En intégrant cette densité sur l'espace qu'occupe l'objet considéré, on obtient sa masse totale.

4.2.4 Caractérisation d'une variable aléatoire

4.2.4.1 Espérance mathématique

L'espérance mathématique *E* d'une variable aléatoire *X* est la moyenne des valeurs *x* possibles de cette variable, pondérées par les probabilités *p*(*x*) de ces valeurs.

$$\boxed{E(X) = \sum_{i=1}^{n} x_i p_i}$$ pour une variable discrète.

$$\boxed{E(X) = \int_{-\infty}^{+\infty} x f(x) dx}$$ pour une variable continue.

L'espérance est une mesure de la valeur moyenne attendue de la variable aléatoire.

Pour nos deux dés :

$$E(X) = 2\times0{,}03 + 3\times0{,}06 + \ldots + 11\times0{,}06 + 12\times0{,}03 = 7$$

La valeur moyenne des tirages est 7, résultat bien évidemment attendu pour des dés non pipés, puisque la distribution est symétrique (7 = (2+12)/2).

Propriétés : Soient X et Y sont deux v. a. quelconques, a et b deux constantes quelconques. On a :

$$E(aX + bY) = aE(X) + bE(Y).$$

Si X et Y sont *indépendantes*[47], on a : $E(XY) = E(X)E(Y)$.

4.2.4.2 Variance

La variance *V*(*X*) d'une variable aléatoire est l'espérance mathématique du carré de l'écart à l'espérance mathématique.

$$\boxed{V(X) = E\left(\left[X - E(X)\right]^2\right)}$$

La variance est une mesure de la puissance du « bruit » d'une variable aléatoire, c'est-à-dire de l'éloignement des valeurs qu'elle prend sur son domaine par rapport à l'espérance.

La variance du résultat du jet de dés est *V*(*X*) = 5,83.

4.2.4.3 Écart Type

L'écart type σ_x d'une v. a. est la racine carrée de la variance :

$$\boxed{s_x = \sqrt{V(X)}}$$

Pour nos dés, l'écart type est 2,42.

Dans le cas de l'analyse des impacts de balles sur une cible, l'écart type donnera une idée de la dispersion du tir.

[47] On dit que les variables aléatoires X et Y sont **indépendantes** si le fait que l'une d'elle prenne une valeur donnée ne modifie pas la probabilité que l'autre prenne n'importe quelle autre valeur. On notera que cette condition d'indépendance n'est pas nécessaire pour que E(*X*.*Y*) = E(*X*).E(*Y*), mais qu'elle est suffisante.

Cas particulier d'une variable de Bernoulli :

$$\begin{cases} \Pr\{X = 1\} = p \\ E(X) = (1-p).0 + p.1 = p \\ V(X) = (1-p).(0-p)^2 + p.(1-p)^2 = p(1-p) \end{cases}$$

4.2.4.4 Moments

On définit les **moments d'ordre *q*** d'une variable aléatoire par :

$$\boxed{m_q(X) = E(X^q)}$$

L'espérance mathématique est le moment d'ordre 1 : $m_1(X) = E(X)$

La variance peut alors s'écrire ainsi :

$$V(X) = E([X - E(X)]^2) = E\left(X^2 + E(X)^2 - 2XE(X)\right)$$
$$= E(X^2) - E(X)^2$$

$$\Rightarrow \boxed{V(X) = m_2(X) - m_1(X)^2}$$

Pour deux variables quelconques, on a :

$$V(X+Y) = V(X) + V(Y) + 2\,\mathrm{cov}(X,Y)$$

avec $\boxed{\mathrm{cov}(Y,Y) = E([X - E(X)][Y - E(Y)])}$ **covariance** de X et Y.

Si deux variables *X* et *Y* sont indépendantes, la covariance de *X* et *Y* est nulle. Donc :

$$V(X+Y) = V(X) + V(Y)$$

On définit également les **moments centrés d'ordre *q*** d'une variable aléatoire par :

$$m_q(X) = E\left([X - E(X)]^q\right)$$

On a alors : $V(X) = m_2(X)$

4.2.4.5 Intervalle de confiance

Un **intervalle de confiance** est un intervalle $[t_1, t_2]$ tel que, pour un niveau de confiance $(1-a)$ donné, on ait :

$$\boxed{\Pr\{\, X \in [t_1, t_2]\,\} = 1 - a}$$

4.3 Distributions statistiques

4.3.1 Introduction

« *Though this be madness, yet there is a method in it.* » *(W. Shakespeare, Hamlet, acte II)*

Polonius trouvait qu'il y avait une certaine logique dans la folie simulée d'Hamlet. Nous allons voir qu'il en est ainsi de nombreux phénomènes apparemment aléatoires.

Prenons une arme (je vous l'avais dit que j'aimais bien cet exemple !). Plaçons-la dans un étau et tirons 1 000 cartouches sur une cible. Nous obtenons une dispersion aléatoire des impacts tels que représenté sur la figure ci-contre.

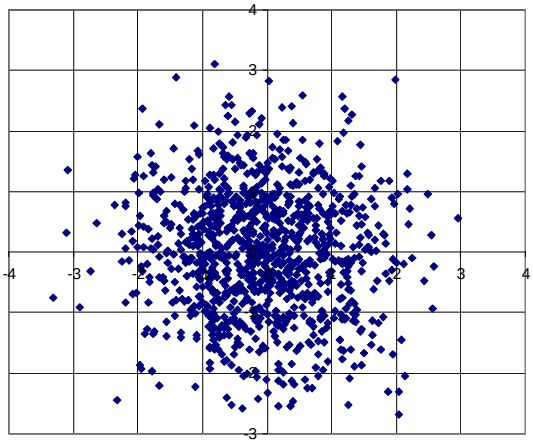


Figure 60: Impacts sur une cible

Apparemment, les impacts se font purement au hasard. Toutefois, on remarque que la densité des impacts est plus forte au centre de la cible, près du point visé (ici le centre de la cible aux coordonnées (0,0)), que sur les bords, heureusement d'ailleurs pour la fiabilité de l'arme !

Nous allons voir comment évolue justement cette densité des impacts. Avec beaucoup de patience, nous relevons les positions exactes des 1 000 impacts et nous les classons en fonction de leur abscisses, et nous traçons l'histogramme correspondant. Nous obtenons la figure suivante :

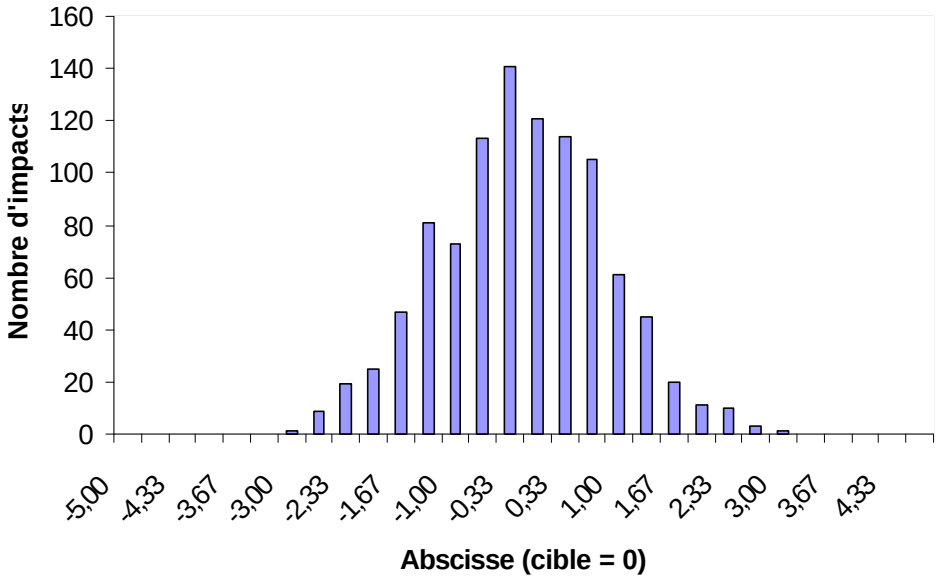


Figure 61: Répartition des 1000 impacts

Nous voyons apparaître une tendance, mais ce n'est pas très évident. Pour être certain, recommençons avec 10 000 impacts cette fois :

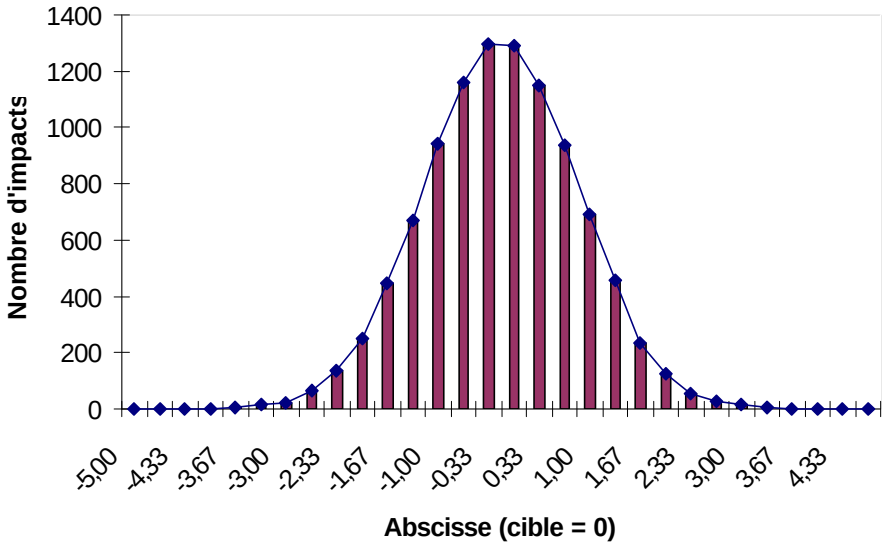


Figure 62: Répartition des 10 000 impacts

On constate ici que la répartition des impacts le long de l'axe X semble suivre une loi mathématique. **Hasard n'est donc pas synonyme de chaos**. Notons qu'il faut un nombre suffisamment important de valeurs (ou **échantillons**) pour que cette loi apparaisse clairement.

Nous verrons plus loin que les positions des impacts sur un axe donné passant par le point visé suivent une loi dite normale, dont la forme en cloche de la courbe de répartition est caractéristique.

4.3.2 Lois de répartition courantes d'une v.a.

4.3.2.1 Loi uniforme

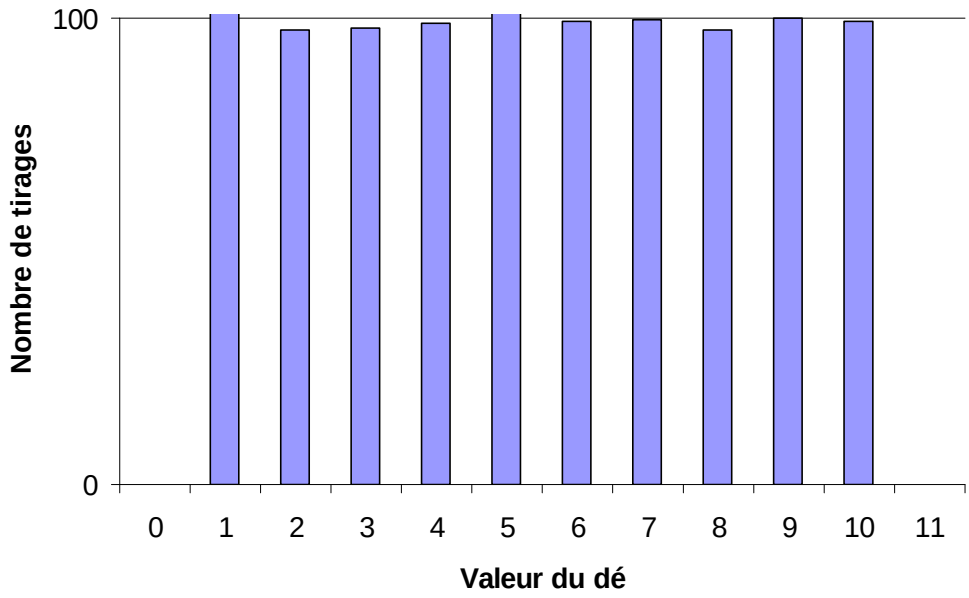


Figure 63: Répartition des 1 000 tirages d'un dé à 10 faces

Prenons un dé à dix face, tel que ceux utilisés dans les jeux de rôles, et lançons-les 1 000 fois de suite. Comme au §4.2, dénombrons les différents tirages et traçons l'histogramme correspondant :

Nous voyons que les tirages sont répartis uniformément suivant les 10 valeurs possibles.

Il s'agit d'une **loi uniforme.**

Dans le cas d'une variable aléatoire *continue*, il existe une loi analogue portant le même nom. Elle est caractérisée par sa fonction de densité de probabilité :

$$f(x) = \begin{cases} \dfrac{1}{b-a} & a < x < b \\ 0 & \text{sinon} \end{cases}$$

La fonction de répartition est alors :

$$F(x) = \begin{cases} 0 & x \le a \\ \dfrac{x-a}{b-a} & a < x < b \\ 1 & x \ge b \end{cases}$$

Exemple : Le dé à dix face, variable aléatoire uniforme discrète

x	1	2	3	4	5	6	7	8	9	10
n	105	98	98	99	104	99	100	97	100	100
$p(x)$	10%	10%	10%	10%	10%	10%	10%	10%	10%	10%
$F(x)$	0%	10%	20%	30%	40%	50%	60%	70%	80%	90%

4.3.2.2 Loi binomiale

Soit un événement dont le résultat peut prendre une valeur parmi deux (par exemple échec ou succès, pile ou face, etc.). Si les événements sont indépendants entre eux (i.e. si l'on effectue plusieurs tirages successif de manière indépendante), la variable aléatoire discrète représentant le nombre de succès ou d'échecs suit une **loi binomiale**.

L'exemple typique est celle de l' « urne de Bernouilli » : on prend une urne contenant N boules identiques, avec une proportion de p boules blanches et de $q = 1 - p$ boules noires. On effectue des tirages en remettant à chaque fois les boules dans l'urne, en mélangeant bien entre chaque tirage.

i la boule est blanche (ou si lévénement est un succès)

Soit la variable aléatoire x représentant la couleur de la boule tirée. On prend comme convention :

$$\begin{cases} x = 1 \text{ si la boule est blanche (ou si lévénement est un succès)} \\ x = 0 \text{ si la boule est noire (ou si lévénement est un échec)} \end{cases}$$

On a alors :

$$\begin{cases} \Pr\{X = 0\} = p \\ \Pr\{X = 1\} = q = 1 - p \end{cases}$$

D'où :

$$\begin{cases} E(X) = p \\ V(X) = p - p^2 = p(1-p) = pq \end{cases}$$

Et :

$$\boxed{\Pr\{X = x\} = \frac{N!}{x!(N-x)!} p^x q^{N-x}}$$

Ou bien :

$$\boxed{\Pr\{X = x\} = C_N^x \, p^x q^{N-x}} \text{ avec } C_n^k = \frac{n!}{k!(n-k)!} \quad (n \geq 0, k \in \{0,...,n\})$$

On notera cette loi binomiale ***B*(*n*, *p*)**.

Ce qui nous intéresse souvent, c'est la probabilité d'avoir moins de *x* échecs, autrement dit on recherche la fonction de répartition :

$$F(x) = \Pr\{X < x\} = \Pr\{X = 0\} + \Pr\{X = 1\} + ... + \Pr\{X = [x]\}$$

($[x]$ représentant l'entier immédiatement inférieur à *x*, c'est-à-dire *x*- si *x* est entier, la partie entière de *x* sinon)

$$\boxed{F(x) = \sum_{i=0}^{[x]} \mathrm{C}_N^i \; p^i q^{N-i}}$$

Calculons maintenant les caractéristiques de cette loi.

L'espérance mathématique de cette loi est :

$$E(X) = \sum_{x=0}^{N} x \Pr\{X = x\} = \sum_{x=0}^{N} x \mathrm{C}_N^x \; p^x q^{N-x}$$

C'est la somme de *N* variable de Bernoulli dont l'espérance est *p* et la variance *pq*. Donc :

$$\boxed{E(X) = Np}$$

$$\boxed{V(X) = Npq} \text{ et } s = \sqrt{Npq}$$

Prenons une chaîne de fabrication de bidules, dont le taux de défectuosité est de 1% (p=0,01). Les grossistes reçoivent les bidules dans des cartons de N=20 unités. Quelle est la probabilité d'avoir X bidules défectueux dans un carton ? $(X \in \{0,..., N\})$

N est la taille de notre échantillon.

On a : $\Pr\{X = i\} = \dfrac{N!}{i!(N-i)!} p^i (1-p)^{N-i}$

La probabilité d'avoir au plus un bidule défectueux dans le lot est $\mathrm{P}\{X \leq 1\}$:

$$\Pr\{X = 0\} = \frac{20!}{0!20!} 0{,}01^0 . 0{,}99^{20} = 81{,}79\%$$

$$\Pr\{X = 1\} = \frac{20!}{1!19!} 0{,}01^1 . 0{,}99^{19} = 16{,}52\%$$

$$\Pr\{X \leq 1\} = \Pr\{X = 0\} + \Pr\{X = 1\} = 98{,}3\%$$

Autre exemple : revenons à nos 2 dés à six faces. Quelle est la probabilité de sortir au moins deux fois un huit en N = 5 lancers ?

Nous avons vu qu'il y avait 36 combinaisons possibles des deux dés. Sur ces 36 combinaisons, toutes équiprobables, cinq permettent d'obtenir un huit : {2;6}, {3;5}, {4;4}, {5; 3}, {6;2}.

La probabilité d'avoir un huit en un jet est donc de $p = \dfrac{5}{36}$.

$$\Pr\{\text{2 huits en cinq jets}\} = C_5^2\left(\frac{5}{36}\right)^2\left(1-\frac{5}{36}\right)^{5-2} = 0{,}123$$

$$\Pr\{\text{au moins 2 huits en cinq jets}\} = \Pr\{\text{2 huits}\} + \Pr\{\text{3 huits}\} + \Pr\{\text{4 huits}\} + \Pr\{\text{5 huits}\}$$

Ce qui nous donne une probabilité de 14,5%

4.3.2.3 Loi normale

Une **loi normale** ou **loi de Gauss** ou loi de Gauss-Laplace régit une variable aléatoire continue dépendant d'un grande nombre de causes indépendantes, dont les effets sont additifs sans qu'aucune de ces causes ne soient prépondérantes.

Exemple de phénomènes aléatoires typiquement régis par une loi normale : erreurs de mesure, coordonnées des impacts de projectiles sur une cible, variations de la taille d'une pièce usinée, etc.

La densité de probabilité d'une v.a. normale continue est :

$$\boxed{f(x) = \frac{1}{s\sqrt{2\pi}} e^{\frac{-(x-m)^2}{2s^2}}} = N(m, s) \quad \text{définie pour tout } x \in \Re.$$

La loi de probabilité d'une variable normale dépend donc de deux paramètres s et m. On notera $\boldsymbol{N(m, s)}$.

En faisant dans la formule précédemment donnée pour $N(m, s)$ le changement de variable :

$$T = \frac{X - m}{s}$$

on obtient :

$$\boxed{f(t) = \frac{1}{\sqrt{2\pi}} e^{\frac{-t^2}{2}}} = N(0,1)$$

$N(0,1)$ est dite loi **normale**, **centrée**, **réduite**.

On peut montrer que l'espérance mathématique d'une v.a. distribuée selon $N(0,1)$ est 0 et que son écart type est 1. Sa médiane est 0.

Plus généralement, pour une variable aléatoire normale $N(m,s)$, m est l'espérance mathématique et la médiane (axe de symétrie de la courbe de Gauss), et s est l'écart type.

La figure ci-dessous donne l'aspect de $N(0, s)$ pour différentes valeurs du paramètre s :

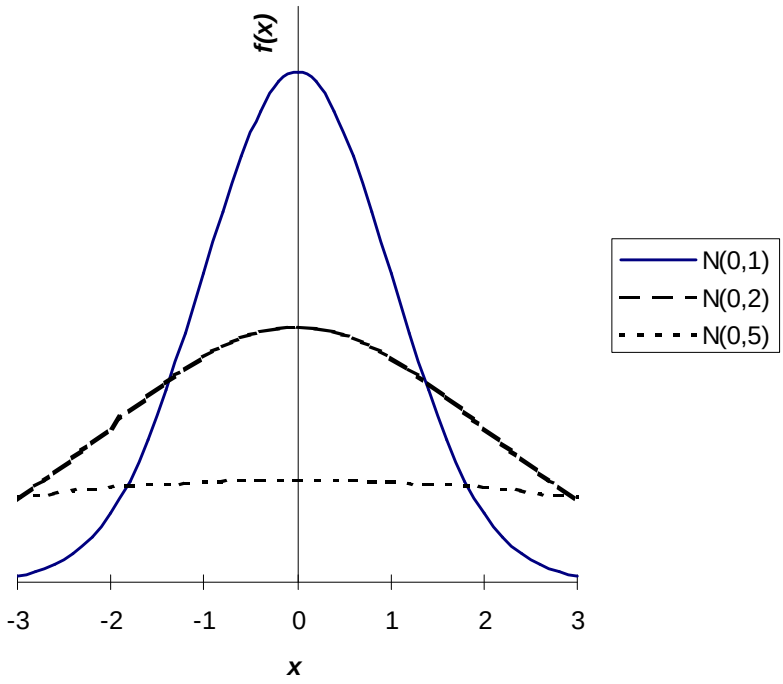


***Figure 64 : Loi* N*(0,* s*) pour différentes valeurs de* s**

On voit que plus l'écart type est grand, plus la courbe est étalée. Ce résultat était prévisible puisque cela correspond à une dispersion plus grande des valeurs. Dans l'exemple que nous avions pris, plus l'arme est précise, plus les impacts se resserrent autour du centre de la cible.

Propriété : la somme de plusieurs VA normales est une VA normale. Si ces lois sont d'espérance mathématique m_i et de variance s_i^2 et qu'elle sont indépendantes, on montre que la somme de ces variables soit une loi :

$$N\left(\sum m_i, \sqrt{\sum s_i^{\,2}}\right).$$

Par exemple, dans le cas de notre arme, la répartition abscisses des impacts suit une loi normale. Si on retire cette arme de l'étau pour tirer « à la main », on cumulera une dispersion due aux erreurs du tireur, qui suivent également une loi normale. La dispersion finale suivra toujours une courbe de Gauss suivant les axes X et Y[48], bien qu'un peu plus étalée.

Remarque : relation entre loi binomiale et loi normale. Si le nombre d'essais N est grand et que les probabilités p et q ne sont pas trop proches de 0 ou de 1 (typiquement si $Np > 5$ et $Nq > 5$), on peut approcher la distribution par une loi normale avec la variable :

$$z = \frac{X - Np}{\sqrt{Npq}}$$

[48] Si x et y suivent une loi de Gauss, ce n'est pas le cas de la distance $d = \sqrt{x^2 + y^2}$ du point d'impact au centre de la cible, puisque d n'est pas une combinaison linéaire de x et y. En fait, la distance suit une loi dite de Rayleigh.

4.3.2.4 Loi de Poisson

On appelle **processus de Poisson** la réalisation d'événements aléatoires dans le temps respectant les conditions suivantes :

- La probabilité de réalisation d'un événement dans un cours laps de temps Δt est proportionnelle à Δt (soit $l = p\Delta t$).
- Elle est indépendante des événements qui l'ont précédée.
- La probabilité d'occurrence de deux événements dans l'intervalle Δt est négligeable.

Exemples typiques de processus de Poisson : arrivée de clients à un guichet, atterrissages sur une piste d'aéroport…

La loi de Poisson peut aussi s'appliquer à la répartition d'événements dans l'espace (remplacer Δt par Δd). Exemples : présence de nœuds le long d'un arbre, recherche de trèfles à quatre feuilles dans un champ…

La loi de probabilité est alors :

$$\boxed{\Pr\{X = x\} = \frac{e^{-l} l^{x}}{x!}}$$

l est le paramètre de la loi de Poisson.

L'espérance mathématique est :

$$E(X) = \sum_{x=0}^{+\infty} x \frac{e^{-l} l^{x}}{x!}$$

Or, $e^{l} = \sum_{x=0}^{+\infty} \frac{m^{x}}{x!}$

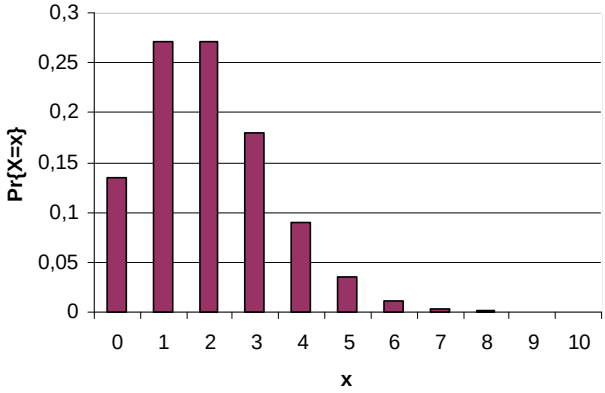


Figure 65: Distribution de Poisson (l = 2)

D'où l'on tire : $\boxed{E(X) = l}$.

De même : $\boxed{V(X) = l}$

Remarque : on peut approcher une loi binomiale par une loi de Poisson. En effet :

$$f(x) = C_N^x \, p^x (1-p)^{N-x} = \frac{N(N-1)...(N-x+1)}{N^x} \frac{(Np)^x}{x!} \left(1 - \frac{Np}{N}\right)^{N-x}$$

On voit que si p est petit et que N est très grand, le produit Np restant fini et égal à λ, on a :

$$f(x) \approx \left(\frac{e^{-l} l^{x}}{x!} \right)$$ On considère généralement que ces conditions sont réalisées si N > 50 et Np < 5.

On peut montrer également que lorsque λ devient grand, la loi de Poisson devient proche d'une loi normale, avec la variable :

$$z = \frac{(X - l)}{\sqrt{l}}$$

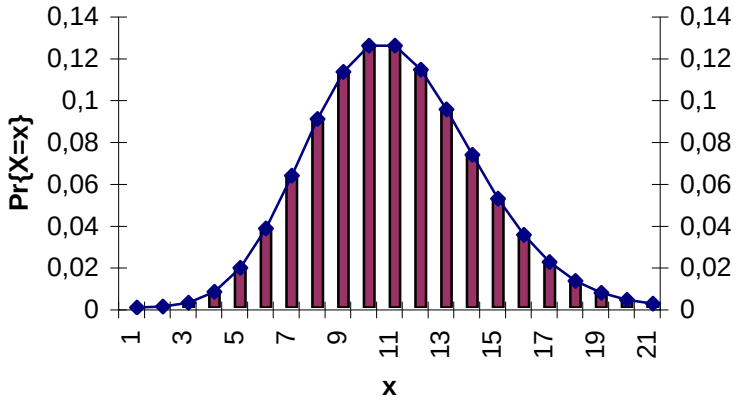


Figure 66: Distribution de Poisson (l = 10)

Exemple : le SAV de la Bidule Corporation reçoit chaque jour de 0 à 5 bidules défectueux. Le tableau suivant donne le nombre de jours où il n'y a eu aucun retour, un retour, etc., sur une période de 90 jours :

Nombre de retours *x*	Nombre de jours *f(x)*
0	15
1	27
2	24
3	14
4	7
5	3

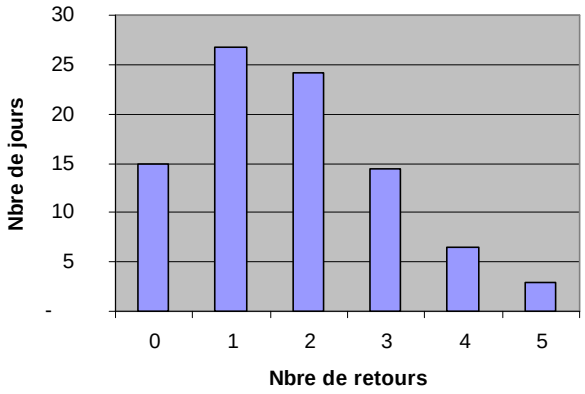


Figure 67: Histogramme de la répartition du nombre de retours de bidules par jour

On commence par calculer la moyenne du nombre de retours par jour :

$$\bar{x} = \frac{\sum_{i=0}^{5} x_i . f(x_i)}{\sum_{i=0}^{5} f(x_i)} = 1{,}73 .$$

La variance est :

$$s^2 = \frac{\sum_{i=0}^{5} x_i^2 \cdot f(x_i)}{\sum_{i=0}^{5} f(x_i)} - \left(\bar{x}\right)^2 = 1{,}66.$$

On souhaite trouver une distribution qui permette une approximation de la répartition des jours. On constate que :

$$E(X) = \bar{x} \approx s^2 = V(X)$$

On peut donc supposer que cette distribution peut être approchée par une loi de Poisson de paramètre $l = \bar{x}$.

$$\Pr\{X = x\} = e^{-1{,}73} \frac{(1{,}73)^x}{x!}$$

Nombre de retours x	$\Pr\{X = x\}$	Nombre de jours $f(x)$ (par calcul)
0	0,165299	14,877
1	0,297538	26,778
2	0,267784	24,101
3	0,160671	14,460
4	0,072302	6,507
5	0,026029	2,343

Il semble que la distribution de Poisson choisie « colle » aux valeurs expérimentales, mais comment déterminer le degré de corrélation entre l'approximation et la réalité ?

C'est l'objet de la section suivante (§4.4).

4.3.2.5 Loi exponentielle

Si le nombre d'arrivées de clients à un guichet pendant un intervalle donné peut être décrit par une loi de Poisson, l'intervalle de temps entre chaque client suit, lui, une loi exponentielle, de même que le temps de désintégration d'un noyau atomique instable, ou le temps au bout duquel un appareil tombe en panne.

La densité de probabilité est :

$$\boxed{f(x) = l e^{-lx} \quad (x \geq 0,\ l > 0)}$$

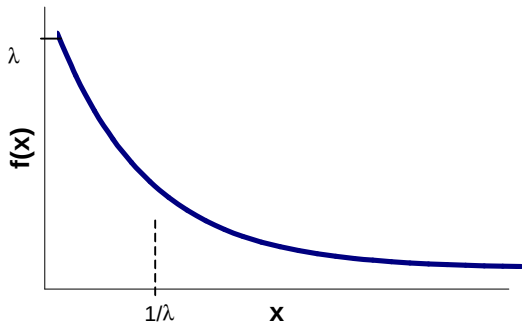


Figure 68: Loi exponentielle

On a : $\boxed{E(X)=\frac{1}{l}}$ et $\boxed{V(X)=\left(\frac{1}{l}\right)^2}$.

On notera que, bien que cette distribution soit fortement dissymétrique, la probabilité pour que x soit situé dans l'intervalle $\left[E(X)-\frac{s}{2},E(X)+\frac{s}{2}\right]$ (s étant l'écart-type de la distribution) est :

$$\Pr\left\{E(X)-\frac{s}{2}\le x\le E(X)+\frac{s}{2}\right\}=\int_{\frac{1}{2l}}^{\frac{3}{2l}} l e^{-lx}dx=0{,}83$$

4.3.2.6 Loi du c^2

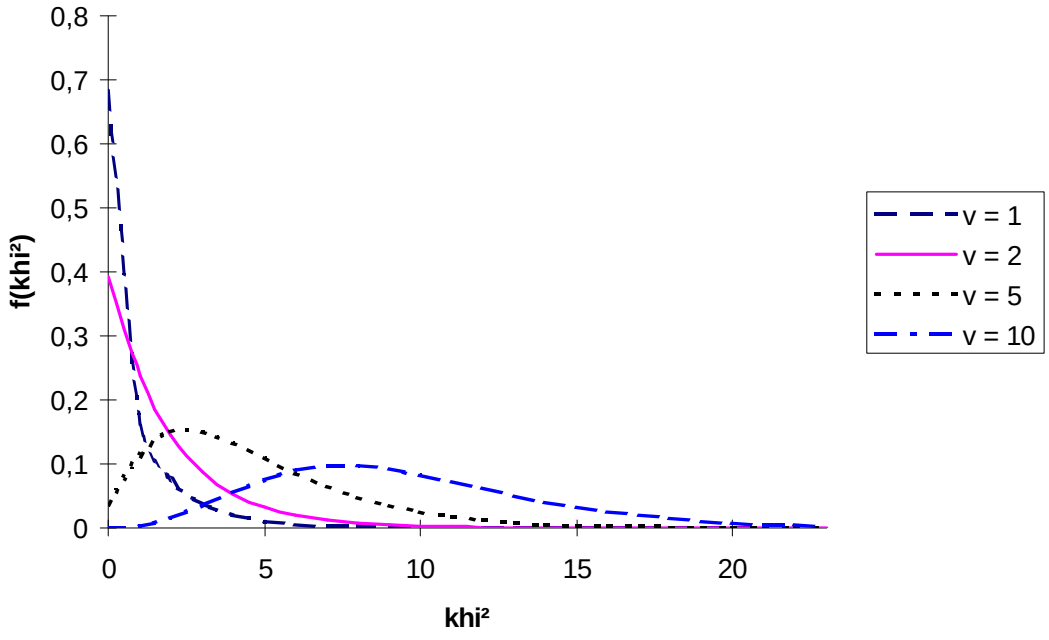
Soient n variables aléatoires X_i normales centrées données, réduits et indépendantes.

On définit c par : $\boxed{c^2=\sum_{i=1}^{n} X_i^{\,2}}$.

On dit que χ^2 soit une loi du khi-carré à n degrés de liberté.

Son espérance est $E(c^2)=n$ et $V(c^2)=2n$.

Cette loi a été découverte par K. Pearson.



Fgure 69: Test du c^2 pour différents dégrés de liberté

Remarque : lorsque n devient grand, la loi du χ^2 est assimilable à une loi normale.

Exemple de variable aléatoire pouvant suivre une loi du χ^2 : la distance des impacts sur une cible au point visé, quand la dispersion est homogène ($\sigma_x = \sigma_y$).

4.3.2.7 Loi de Student

Soient X et Y deux variables aléatoires indépendantes, telles que Y suit une loi du χ^2 à n degrés de liberté, et X une loi $N(0,1)$.

Soit la variable aléatoire : $\boxed{T = \frac{X}{\sqrt{\frac{Y}{n}}}}$

T suit une **loi de Student** à n degrés de liberté.

On a, pour $n > 2$:

$$E(T) = 0 \text{ et } V(T) = \frac{n}{n-2}.$$

Figure 70: W. Student (1876-1937)

4.4 Tests de corrélation

4.4.1 Comment reconnaître une distribution ?

Comme on a pu en avoir un aperçu précédemment, la reconnaissance d'une distribution est assez empirique : à partir d'échantillons, on trace l'histogramme, et on y met un peu d'intuition en fonction du type d'événement aléatoire.

Une fois que l'on a une hypothèse sur le type de distribution, on peut calculer une estimation des paramètres probables de la distribution, à partir notamment d'une estimation de l'espérance $\hat{E}(X)$ et de la variance $\hat{V}(X)$.

Dans le cas de la répartition du nombre de bidules en panne par jour, on a par exemple : $\hat{l} = \bar{x}$.

Mais comment estimer la probabilité que notre distribution théorique corresponde à la réalité ?

Les **tests statistiques** permettent de comparer deux séries d'observations (**échantillons**), ou une série d'observations et une distribution théorique.

Chaque échantillon est supposé extrait d'une population dont l'effectif est très grand. Si les deux populations suivent les mêmes lois statistiques, alors les écarts entre les deux sont purement aléatoires (i.e. suivent une loi normale centrée). Une étude des écarts permet alors d'estimer la probabilité pour que les populations soient similaires.

Bien entendu, dans la mesure où des phénomènes aléatoires entrent en jeu, le résultat n'est pas d'une exactitude garantie. C'est pourquoi on parlera de « risque à N% » pour indiquer qu'il y a N% de chances pour que le résultat du test soit faux. On travaille généralement avec des risques de 1% et 5% (certitude à 99% et 95% respectivement).

Il faut évidemment garder un certain esprit critique en utilisant les tests statistiques, car deux populations différentes peuvent très bien passer avec succès les tests si les échantillons sont mal choisis (par exemple dans un domaine où les deux populations sont similaires, alors qu'elles divergent ailleurs).

Il existe de nombreux tests. Le plus courant est le test du χ^2.

4.4.2 Méthode du c^2

Le test d'ajustement le plus connu est celui du χ^2 (prononcer « khi carré ») de K. Pearson.

Il est basé sur la comparaison d'effectifs de diverses classes de valeurs observées ($O_1, \ldots, O_n$) aux effectifs théoriques ($T_1, \ldots, T_n$). Pour que le test soit vraiment significatif, il faut au moins une dizaine de classes contenant chacune au moins 5 échantillons.

Hormis cette contrainte, les limites des classes peuvent être choisies arbitrairement. Toutefois, la loi normale étant une fonction définie de $-\infty$ à $+\infty$, il est usuel de considérer deux classes particulières $E_1 =]-\infty, O_1]$ et $E_n = [O_n, +\infty[$.

Classe	E_1	E_2	...	E_n
Population observée	O_1	O_2	...	O_n
Population théorique	T_1	T_2	...	T_n

La population totale est $P = \sum_{i=1}^{n} O_i = \sum_{i=1}^{n} T_i$.

On évalue la divergence entre les populations observées et théoriques en calculant la statistique χ^2 :

$$? ^2 = \frac{(O_1 - T_1)^2}{T_1} + \frac{(O_2 - T_2)^2}{T_2} + \ldots + \frac{(O_n - T_n)^2}{T_n} \text{ soit } \boxed{? ^2 = \sum_{i=1}^{n} \frac{(O_i - T_i)^2}{T_i}},$$

ou encore : $? ^2 = \sum_{i=1}^{n} \frac{O_i^{\,2}}{T_i} - P$.

Plus χ^2 est grand, plus la divergence est importante entre les deux séries. Si $\chi^2 = 0$, alors les deux séries coïncideraient.

On détermine ensuite le nombre de **degrés de liberté** *n*., avec $n = n - m$.

m est le nombre de relations liant les populations des classes entre elles, c'est-à-dire permettant de déduire la population d'une classe des autres effectifs. On remarquera que $m > 1$, puisque :

$$P = \sum_{i=1}^{n} O_i$$

donne une relation permettant de calculer la population d'une des classes à partir des autres.

On prend ensuite la valeur χ_a^2 du χ^2 pour le nombre de degrés de liberté et le seuil de signification *a* souhaité (on prend souvent $a = 0{,}01$ ou $a = 0{,}05$), valeur lue dans une table ou calculée par un logiciel. Pour que le test soit réussi, il faut que l'on ait $c^2 < c_a^2$.

Reprenons notre exemple avec des bidules (voir page 92).

Nombre de retours *x*	Nombre de jours *f(x)* (observés)	Nombre de jours *f(x)* (calculés)
0	15	14,877
1	27	26,778
2	24	24,101
3	14	14,460
4 ou 5	10	8,850

Nous souhaitons vérifier que la loi de Poisson que nous avons calculée est correctement ajustée aux observations. Nous allons faire cela avec le test du χ^2.

Nous reprenons les valeurs calculées précédemment en regroupant les deux dernières classes, afin que les effectifs de toutes les classes soient significatifs :

Nombre de retours *x*	Nombre de jours *f(x)* (observés)	Nombre de jours *f(x)* (calculés)
0	15	14,877
1	27	26,778
2	24	24,101
3	14	14,460
4 ou 5	10	8,850

Nous calculons la valeur du ? $^2 = \sum_{i=1}^{n} \frac{(O_i - T_i)^2}{T_i} = 0{,}24$

Les relations suivantes lient entre eux les effectifs des 5 classes :

$$E(X) = \frac{1}{l} \quad \text{et} \quad P = \sum_{i=1}^{n} O_i$$

On a donc 5 - 2 = 3 degrés de liberté.

Si on prend comme seuil de signification a = 0,05, la table du χ^2 nous donne : $\chi_{0,05}^2 = 7{,}81$, ce qui est largement supérieur à ce que nous avons calculé.

Au seuil de signification choisi, la différence entre la distribution observée et la distribution de Poisson calculée n'est donc pas significative.

4.4.3 Autres tests

Il existe d'autres tests (par exemple celui de Aspin Welch pour comparer deux moyennes), mais l'objet de ce cours n'est pas d'avoir un inventaire exhaustif des techniques statistiques.

La bibliographie à la fin de ce cours donnera quelques ouvrages pour approfondir ce sujet, par exemple [FOUR97] pour les statistiques ou [FISH96] pour Monte-Carlo. Des livres généraux sur la simulation ont souvent un chapitre sur les statistiques (par exemple [BANK98] et [KELT99]).

5. MODELISATION DE SYSTEMES

5.1 Généralités

5.1.1 Étapes de réalisation d'une simulation

Faire une simulation ne s'improvise pas.

En effet, pour être efficace et rentable, la simulation requiert une base méthodologique sérieuse et une rigueur que, parfois, comme trop souvent dans le monde du logiciel, on oublie. Virtuel n'est pas synonyme d'immédiat et de simplicité.

La modélisation d'un système n'est pas une tâche à prendre à la légère. En effet, les sources d'erreurs possibles sont nombreuses, comme nous le verrons par la suite, et les conséquences peuvent s'avérer désastreuses, par exemple une mauvaise décision de l'ingénieur ou du général, avec à la clé une coûteuse erreur de conception d'un système, voire des pertes humaines.

Il y a plusieurs approches méthodologiques possibles, respectant généralement le schéma suivant, avec parfois quelques variations suivant les différentes « cultures » :

1. ***Formulation du problème***
2. ***Préciser les objectifs et organiser le projet***
3. ***Modélisation et collecte des données***
4. ***Codage***
5. ***Vérification du code et des données***
6. ***Validation du code et des données***
7. ***Exécution de la simulation***
8. ***Dépouillement et analyse (critique) des résultats***
9. ***Rédaction du rapport final***
10. ***Mise en service de l'application et/ou capitalisation du modèle***

La Figure 71 présente, de façon simplifiée[49], l'organigramme correspondant à ce processus.

[49] Entre autres, ce schéma ne fait pas ressortir la nature continue des processus de vérification et validation.

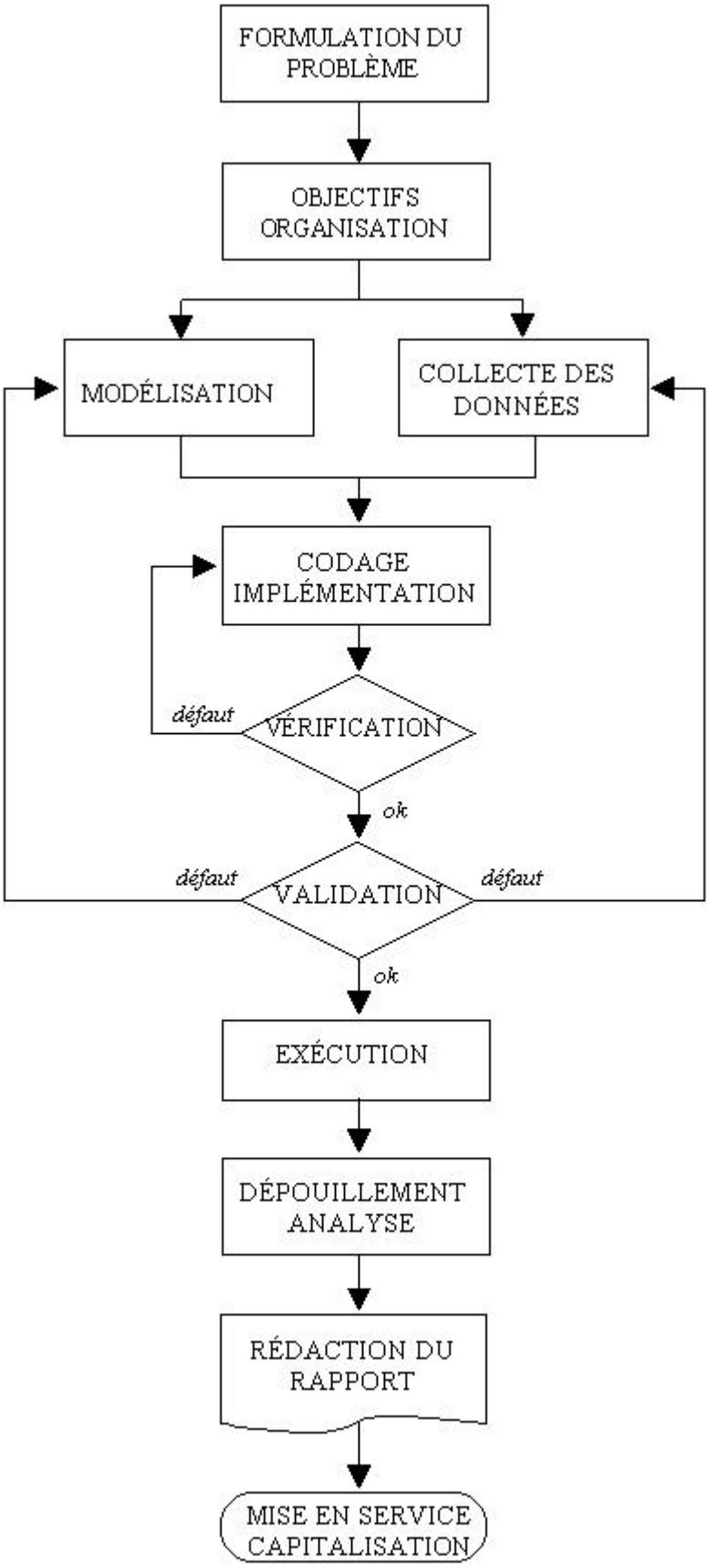


Figure 71: Processus global de modélisation & simulation

Nous allons maintenant étudier en détail chaque étape, en voyant quelles sont les questions à se poser, et quelles sont les méthodes qui peuvent être mises en œuvre.

5.1.1.1 Formulation du problème

Comme dans toute réalisation d'ingénierie, on commence par énoncer le **besoin**, en l'occurrence ici le problème que nous souhaitons adresser par le biais de la simulation.

Ce besoin peut être fourni par le demandeur (client), mais l'analyste qui va s'occuper de la modélisation doit s'assurer que tout est parfaitement clair, et que chacun a une bonne vision du problème posé (identification et délimitation du système étudié…).

Cette étape doit se faire sous forme de réunions interactives, où chacun doit pouvoir s'exprimer ouvertement. Le client n'a pas forcément le vocabulaire ou la culture scientifique nécessaire pour exprimer clairement son besoin, ou va le faire dans le vocabulaire de son métier, bien entendu incompréhensible pour tout individu extérieur. L'ingénieur chargé de réaliser l'étude devra, lui, relever les éventuelles ambiguïtés[50], contradictions, difficultés techniques, etc. Aucune solution ne doit être envisagée avant qu'un consensus n'ait été atteint sur le problème.

C'est une étape bien entendu critique, car toute erreur à ce niveau risque de n'être décelée qu'en fin de processus, ce qui serait évidemment catastrophique.

5.1.1.2 Objectifs et organisation

Le responsable du projet (i.e. le chef de projet) et le client préparent conjointement les **spécifications** de la simulation : objectifs à atteindre (i.e. questions à résoudre ou fonctionnalités attendues), environnement naturel et tactique, scénario, précision du modèle, critères de validation, exigence de qualité (réutilisabilité, évolutivité…), contraintes particulières (utilisation de certains formats de données ou de certains outils logiciels).

Les **objectifs** serviront à évaluer le résultat. Ils doivent être quantifiables (mesurables), et rester raisonnables en ambition et en quantité. En effet, des objectifs trop nombreux (plus de trois ou quatre) ou trop ambitieux risquent de provoquer une dispersion de l'équipe de projet. Mieux vaut se centrer sur un nombre limité de point précis et atteignables. Pour les mêmes raisons, il faut que l'agenda du projet soit réaliste : une équipe qui travaille dans la précipitation fait rarement du bon boulot, surtout en informatique ! Il faudra également tenir compte des besoins du client : si les résultats arrivent trop tard, ils ne serviront à rien.

Il est important de définir le **scénario** assez tôt, car il détermine les conditions dans lesquelles va évoluer le système simulé, et s'avère dimensionnant pour la simulation. En effet, on ne concevra peut-être pas de la même façon une simulation chargée de gérer une dizaine d'entité et une simulation en impliquant un millier : dans le second cas, il va falloir tenir compte des ressources machines disponibles et, par exemple, réduire la précision des modèles ou envisager de distribuer l'application[51].

[50] Ne pas hésiter à faire du zèle : un jour, j'ai confié une étude à un organisme de recherche. Après plusieurs réunions et courriers, nous sommes tombés d'accord sur un cahier des charges, et l'étude débuta. Au bout de trois mois, un premier rapport arriva. Consternation : ce n'était pas du tout ce que nous voulions. En fait, une phrase du cahier des charges avait été interprétée différemment par les deux parties…

[51] Aux débuts de la simulation distribuée, on a souvent vanté l'intérêt de la distribution pour, justement, répartir la charge de travail sur plusieurs ordinateurs. Aujourd'hui, on a tendance à relativiser cet emploi, car il est très

Parallèlement, le chef de projet **organise et planifie** le travail à effectuer. Il détermine les tâches à effectuer et évalue les ressources nécessaires pour chacune d'elle (exprimées généralement en homme-jour ou homme-mois). Il met en place un calendrier de déroulement du projet.

Là encore, tout se déroule comme n'importe quelle réalisation d'ingénierie.

5.1.1.3 Modélisation

L'étape de modélisation est particulièrement déterminante sur les performances de la simulation. Elle est le cœur du processus de modélisation et simulation. C'est pourquoi nous l'étudierons plus en détail dans la section 5.2), avec une prise en compte particulière des aspects statiques, dynamiques, temporels et aléatoires des simulations.

5.1.1.4 Collecte des données

La collecte des données est une opération difficile et souvent sous-estimée. Cette tâche, souvent longue et fastidieuse, doit être déroulée de préférence en parallèle avec la modélisation.

Le §5.2.6.2 donne plus de détails sur cette opération.

5.1.1.5 Codage

Lorsqu'un modèle est bien conçu, son codage de doit pas poser de problème particulier.

Avant de commencer le codage, il convient de bien réfléchir au choix de la plate-forme de développement, du langage[52], des conventions de programmation (noms des variables, commentaires…), et des outils (mise au point, vérification du code…).

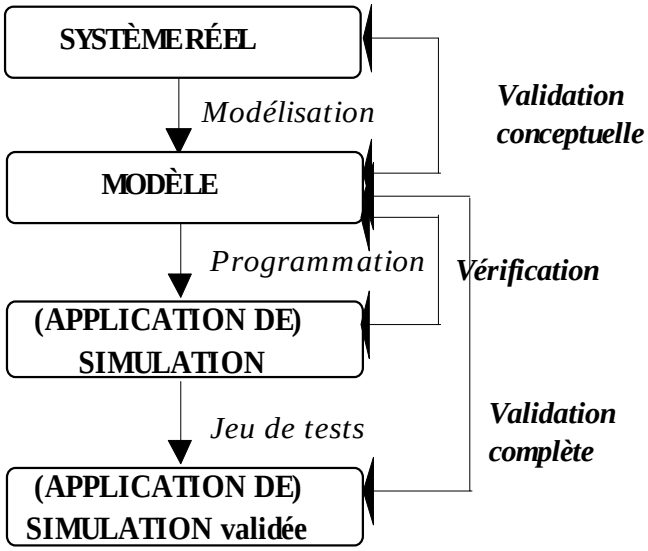


Figure 72: Position (simpliste) de la validation et de la vérification dans le développement d'une simulation

5.1.1.6 Vérification

La vérification est le processus par lequel on s'assure que l'**implémentation** d'un modèle est conforme à ses spécifications.

Cette vérification est tout à fait similaire à celle de n'importe quel code informatique. Elle utilise donc les techniques du génie logiciel (voir section 8.1).

---

délicat de bien distribuer une application, et le système résultant peut être très complexe… et parfois plus coûteux et moins performant que ne le serait une architecture avec une seule machine très puissante (j'ai déjà vu un tel cas).

[52] Il est triste de constater que, trop souvent, peu de réflexions sont menées sur le choix d'un langage, lequel relève alors de l'habitude ou d'une « conviction », et non de faits techniques. Même si un programmeur expérimenté peut (en théorie) faire à peut près n'importe quoi avec n'importe quel langage, certains langages peuvent faciliter le codage, ou au contraire concourir à un manque de fiabilité ou de portabilité.

Classiquement, la vérification se déroule en deux phases :

- En fin de conception détaillée, on effectue des tests unitaires sur les composants.
- En fin d'intégration de ces composants, on effectue des tests globaux.

Il existe de nombreuses techniques de vérification :

- Revues, inspections et audit (du code, de la documentation...) : la revue de code est une tâche pénible, mais pouvant s'avérer très fructueuse, pourvu que le code soit écrit lisiblement et correctement documenté[53].
- Tests unitaires et tests d'intégration : on exécute le programme avec des jeux de tests convenablement choisis. Ceci rejoint la validation, à ceci près qu'on choisira pour les données des valeurs plutôt destinées à vérifier le bon fonctionnement du programme (par exemple sa robustesse face à des entrées erronées). Comme pour la validation, l'exécution de ces tests peut être facilitée par la présence d'une visualisation graphique.
- Analyse syntaxique et sémantique du code : effectuée plus ou moins bien par le compilateur. Dans le cas d'un compilateur permissif (par exemple un compilateur C), il est possible de recourir à des outils de contrôle complémentaires tels que `lint`. Notons qu'un bon moyen de diminuer les erreurs de syntaxe et de sémantique consiste à utiliser un éditeur de texte dédié au développement, capable de détecter les erreurs de syntaxe à la volée et effectuant une « coloration syntaxique » mettant en valeur les différents éléments linguistiques du programme.
- Debugging : l'inclusion de code de mise au point à l'intérieur du code du modèle peut s'avérer lourde, mais facile grandement la vérification, par exemple en fournissant des valeurs intermédiaires non visibles habituellement. Dans l'idéal, un programme doit posséder un mode « debug » dans lequel ces bouts de codes de mise au point seront exécutés, alors qu'ils sont inactifs en mode normal. Une autre façon de faire la mise au point d'un code est d'utiliser un « debugger symbolique ». Les ateliers de développement évolués ont en principe un debugger symbolique capable de tracer le programme durant son exécution, ou même de l'exécuter pas-à-pas en affichant la ligne du code source en cours de traitement, et permettant de visualiser les variables et de les modifier.
- Assertions : il s'agit d'insérer en certains points critiques du programme des tests portant sur l'état du modèle. Si ces tests échouent, une erreur est générée. Par exemple, si la variable « altitude » devient négative pour un avion au-dessus de la mer, c'est que le modèle est dans un état anormal. Certains langages (Ada, EIFFEL...) sont capables de traiter automatiquement des assertions, de façon plus ou moins complète. De nombreuses bibliothèques fournissent des fonctions pour C, C++, etc.
- ...

5.1.1.7 Validation

La validation est un processus visant à s'assurer qu'un modèle ou une simulation **représente le monde réel** d'une façon **suffisamment précise** pour remplir les besoins d'une **utilisation donnée**.

[53] Malheureusement, beaucoup de programmeurs ont tendance à négliger cet aspect documentaire, de sorte qu'il n'est pas rare que seul l'auteur du code puisse en faire la relecture et la maintenance.

Il convient d'insister sur ce dernier point : on valide un modèle dans le cadre d'une certaine utilisation de ce modèle. L'ensemble des valeurs possibles pour les variables d'état du système, dans le cadre de cette utilisation pour laquelle il a été validé, est appelé **domaine de validité** du modèle.

Toute utilisation du modèle en-dehors de ce domaine de validité nécessite une nouvelle opération de validation. Toutefois, ceci entraînement vraisemblablement un surcoût nettement plus important que si l'on avait dès le départ validé un domaine plus large.

La validation est donc un processus essentiel, qu'il ne faut pas prendre à la légère.

La façon d'intégrer la validation dans le processus de développement d'un modèle varie suivant les auteurs. Mais la plupart des schémas que l'on peut trouver dans la littérature ne sont pas satisfaisant. En effet, la validation doit être une **action continue**, d'un bout à l'autre du développement du modèle. Y compris d'ailleurs après la mise en service du système : si un défaut est découvert et que le système doit subir des modifications (ou des évolutions), il faudra reprendre le modèle et le modifier en conséquence... puis le « revalider ».

La validation est donc un processus long et coûteux, dont on ne voit parfois jamais la fin, mais qui est indispensable, puisqu'elle dicte la confiance que l'on a dans les résultats d'une simulation.

Il va d'abord s'agit de valider le modèle conceptuel. Puis, le modèle informatique sera validé après la vérification de son implémentation, car ainsi on aura moins de chances que les écarts observés seront imputables à des erreurs de codage.

Outre la **validation conceptuelle** du modèle (en amont du codage) et sa **validation complète** (en aval), il est possible de procéder à la validation de modules internes du modèles, pourvu qu'ils soient correctement instrumentés. L'instrumentation d'un programme consiste à introduire des variables et des fonctionnalités qui n'ont d'autres buts que la mise au point du programme. Dans un modèle qui comporte de nombreux calculs intermédiaires (« sous-modèles »), on peut tirer de grands bénéfices d'une telle instrumentation : en amont, pour valider le modèle par parties, en aval pour rechercher la cause d'un dysfonctionnement. Une telle stratégie est très courante dans l'industrie. Ainsi, une voiture moderne ou un équipement vidéo grand public comportent toujours des connecteurs spéciaux pour les tests en sortie de chaîne de fabrication, mais aussi pour faciliter la détection des pannes lors de la maintenance.

Notons que la question de la validation concerne également les **données** utilisés en entrée du modèle et de la simulation : les données représentant les caractéristiques du système sont-elles correctes ?

Il existe de nombreuses méthodes de validation, avec des variations culturelles, comme c'est d'ailleurs le cas pour la vérification :

- Validation croisée par mesure des écarts entre les sorties du modèle et des essais réels. Ceci peut nécessiter des essais dédiés à la mise au point du modèle. L'exemple flagrant est celui de la reprise des essais nucléaires français en 1995, qui avait eu un coût financier et diplomatique des plus importants, mais qui étaient indispensables à la mise au point et la validation des modèles numériques d'explosions nucléaires. Comme il n'est en principe pas possible de couvrir par des essais tout le domaine de validité souhaité du modèle, il faut optimiser les essais et recourir à des techniques statistiques.

- Mesure des écarts des essais sur des maquettes ou en laboratoire. Si le phénomène modélisé ne peut être reproduit en essai réel, il peut être possible de le simuler à plus petite échelle (maquette d'avion dans une soufflerie) ou de reproduire en laboratoire une partie du phénomène (utilisation d'un laser Mégajoule de haute puissance pour provoquer la fusion de microsphères de deutérium/tritium et reproduire certains phénomènes survenant lors d'une explosion thermonucléaire).
- Mesure des écarts entre les sorties du modèles et celles d'un autre modèle plus fin et déjà validé : si on dispose de simulation validées de composants du systèmes, celles-ci peuvent servir à valider une simulation de plus haut niveau de granularité (i.e. une simulation du système).
- Remplacement de certaines parties du modèle par du matériel dans la boucle (simulation hybride). Cela permet de comparer les résultats obtenus avec le modèle et ceux obtenus avec le matériel réel (autodirecteur, centrale de navigation...). Notons que ceci nécessite de la part des matériels réels qu'ils soient puissent être interfacés avec une simulation informatique. C'est le cas par exemple des équipements de la fusée Ariane.
- Examen critique des résultats : il faut toujours garder un esprit critique sur les résultats d'une simulation, et prendre du recul par rapport aux mathématiques. Tracer quelques courbes de résultats après l'exécution d'une simulation en cours de validation permet souvent de détecter des problèmes. C'est aussi pourquoi il est intéressant, même pour une simulation numérique fermée, d'avoir une visualisation graphique interactive, qui sera utilisée au début pour la mise au point d'un modèle, puis abandonnée une fois la validation effectuée[54].
- Validation interne. Il s'agit d'exécuter un grand nombre de simulations avec les mêmes données, et à observer la stabilité des résultats. Toutefois, des simulations ayant des modèles fortement stochastiques pourront donner des résultats très divergents sans pour autant que les modèles soient faux. Une variante du procédé consiste à faire varier légèrement certaines données ou paramètres pour observer la stabilité du modèle. On parle alors de validation par « analyse de sensibilité ».
- ...

On trouvera dans [BANK98] une description plus complète de méthodes de vérification et validation.

Nous reviendrons sur ces questions de validation et vérification des simulations dans la section 5.4 sur le VV&A des simulations distribuées.

5.1.1.8 Exécution

On appelle *run* ou **réplique** l'exécution d'une simulation avec un scénario donné.

Dans le cas des modèles stochastiques, le résultat de plusieurs exécutions successives d'un modèle avec les mêmes données d'entrée peut produire des résultats différents. Aussi, il est impossible de conclure à l'issue d'une seule réplique. Par conséquent, on exécutera un grand nombre de fois la simulation en sauvegardant à chaque fois dans un fichier de sortie (« journal ») les valeurs des variables (d'état ou statistiques) que l'on

[54] C'est le cas par exemple de la structure d'accueil ESCADRE, dont l'utilisation typique consiste à lancer des exécutions multiples et non-interactives de simulations (pour ensuite faire des analyses par Monte-Carlo), mais qui a été dotée d'une interface graphique (probablement sa partie la plus lourde à maintenir !) afin de faciliter la mise au point et la validation des modèles.

souhaite étudier, sur lesquels on effectuera par la suite des calculs statistiques, pour en déduire les tendances générales.

Ces exécutions multiples sont très lourdes en mode **interactif** (avec l'utilisateur dans la boucle). C'est pourquoi de nombreuses simulations destinées à des analyses de Monte Carlo sont conçues pour pouvoir fonctionner dans un mode fermé, sans utilisateur dans la boucle, appelé **batch**. Toutes les données d'entrée doivent être déterminées dès le départ, de sorte que plusieurs dizaines de répliques peuvent être effectuées à la chaîne (ou en parallèle) sur un calculateur sans intervention de l'utilisateur.

5.1.1.9 Dépouillement des résultats

Les simulations peuvent générer des millions de données, difficiles à interpréter directement. Il faut donc effectuer une nouvelle étape, combinant analyse critique et calculs, pour parvenir à exploiter ces résultats. Cette étape est le plus souvent réalisée après l'exécution de la simulation (« off line »), à partir des fichiers de données générés.

Si l'exploitation des simulations déterministes n'appelle pas de commentaires particuliers, celle des simulations stochastiques présente en revanche une problématique propre.

En effet, comme nous l'avons vu précédemment, les simulations impliquant des processus aléatoires donnent des résultats différents d'une réplique à l'autre. Il faut donc effectuer plusieurs répliques, en nombre suffisant pour obtenir la précision requise.

Une technique très répandue pour exploiter les simulations aléatoires à **horizon fini**, c'est-à-dire limitées dans le temps (par exemple parce que les ressources utilisées par le système sont elles-mêmes limitées) : il s'agit de la technique statistique de **Monte-Carlo**.

Monte-Carlo permet en particulier de calculer une estimation du nombre de répliques à effectuer en fonction de l'intervalle de confiance souhaité pour les résultats (voir [FISH96].

5.1.1.10 Rédaction du rapport

Il doit répondre aux questions que se posait le client, et justifier de la conformité aux spécification. Son contenu doit être décrit dans le cahier des charges initial, puisqu'il est l'objectif final des travaux.

Si le logiciel ou le modèle est destiné à être réutilisé, il doit être documenté de façon appropriée.

5.1.1.11 Mise en service de la simulation

Si la simulation est destinée à un usage régulier chez un client (entraînement, études...), elle devra alors répondre aux mêmes critères de qualité que n'importe quel produit logiciel industriel, y compris sur les aspects formation, maintien en condition opérationnelle (MCO), etc.

Le client doit préciser les conditions et le contenu des livraisons : documentation, modèles conceptuels, dossier de validation, code source...

5.1.1.12 Capitalisation du modèle

Développer et valider un modèle est un processus qui peut s'avérer long et coûteux.

Il est donc naturel de rentabiliser cet investissement en mettant en place une architecture logicielle et une organisation qui permette de réutiliser au mieux ce modèle.

Ceci implique :

- Une validation rigoureuse du modèle dans un domaine bien identifié[55].
- Une documentation du modèle rigoureuse et claire.
- L'utilisation d'un environnement de simulation de type « structure d'accueil » (voir chapitre 8).
- Une structure recueillant les modèles et outils que l'on souhaite réutiliser, conçue de façon à faciliter leur accès par le plus grand nombre de développeurs.
- Une politique suivie, car dans l'anarchie il ne peut y avoir de réutilisation efficace.

La réutilisation systématique des modèles est le rêve de bien des architectes de systèmes de simulation et aussi... de financiers !

Mais en pratique, si on peut aisément définir une politique de capitalisation de modèle à l'échelle d'un établissement ou d'un domaine technologique bien délimité (par exemple les modèles de systèmes optroniques), de nombreuses tentatives de généralisation ont échoué, car, en fonction du domaine, de son type et du niveau de granularité, il y a de grandes différences entre les modèles, et il est très difficile d'intégrer dans un jeu de guerre un modèle de missile d'une grande finesse tel qu'on peut en rencontrer chez un concepteur de missiles.

Il faut donc réutiliser du mieux possible, en se posant toujours la question de la faisabilité technique et de la rentabilité financière à moyen et long terme.

5.1.2 Précision de la modélisation

Quand on modélise un système, il est nécessaire de le discrétiser et d'effectuer un certain nombre de calculs approchés. Bien entendu, plus la discrétisation est fine, plus les calculs sont précis, et plus la puissance de calcul requise et la capacité mémoire seront importantes (et les temps de calculs risquent de s'allonger, sans parler du temps de préparation des données).

En particulier, il n'est pas utile d'avoir une précision des calculs de 0,001% si les données en entrée sont connues avec une erreur de 1% !

C'est pourquoi il est important de définir dans le cahier des charges de la simulation la précision des résultats souhaités, ce qui influera sur les deux paramètres :

– La **granularité**, c'est-à-dire la taille des objets manipulés par la simulation (système complet, groupes de systèmes, composants…).

– La **finesse**, qui est la précision temporelle (pas de calcul) et spatiale (par ex. résolution du terrain numérisé).

– La **fidélité**, qui est le degré de similitude fonctionnelle et physique entre le modèle et le système qu'il représente.

[55] On rappelle que toute utilisation du modèle en dehors de ce domaine rend son accréditation caduque.

Les figures ci-dessous montrent comment la variation de finesse d'un maillage peut modifier la forme d'un modèle 3D de Boeing 747 : dans la version à 400 polygones, on ne distingue pas, par exemple, les supports des réacteurs.

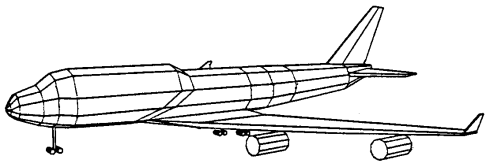


Figure 73 : Boeing 747-400 (400 polygones)

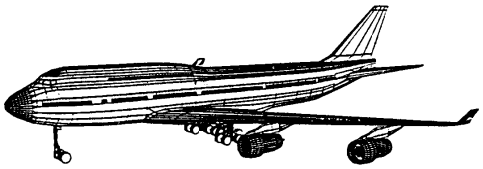


Figure 74 : Boeing 747-400 (11200 polygones)

Le modèle à 400 polygones peut suffire pour une visualisation grossière (simulateur de vol, de tour de contrôle…) mais probablement pas pour une simulation d'aérodynamisme, car trop de détails sont omis. En revanche, il nécessitera beaucoup moins de calculs.

5.2 Modélisation du système

5.2.1 Analyse statique du système

5.2.1.1 Délimitation du système

La première étape est de délimiter le système que l'on souhaite étudier. Étudie-t-on un avion entier ou un composant de l'avion ? À partir de là, on peut distinguer le système de son environnement, c'est-à-dire tous les autres objets et processus qui influent sur le comportement du système étudié.

Il faut aussi rapidement identifier le domaine dans lequel on va étudier le système. S'attachera-t-on au comportement opérationnel de l'avion durant ses missions (par exemple combat contre d'autres avions) ou à son comportement fonctionnel (aérodynamique des ailes, tenue en régime des moteurs...) ? Ceci permet en effet de déterminer quel niveau et quel type de modélisation est nécessaire.

5.2.1.2 Identification des constituants du système

Une fois le système bien identifié, il faut préciser sa composition (voir §2.1.2) :

- De quelles entités est-il constitué ?
- Quelles sont les relations entre ces composants ?
- Quelles sont les entrées/sorties du système ? (informations, perturbations, objets...)
- Quel est son environnement ?

5.2.1.3 Entités d'un système

Les entités d'un système sont les objets, les composants par lesquels le système est défini.

Par exemple, dans un jeu de guerre, ce sont les pions (un régiment, une compagnie, une section ou un fantassin individuel suivant le niveau de la simulation).

Ces entités peuvent être permanentes ou temporaires. Par exemple, dans le cas d'une file d'attente, le serveur est une entité permanente (elle existe dès le début de l'exécution et ne disparaît qu'à la fin). En revanche, les clients sont des entités temporaires (ils apparaissent au fur et à mesure des besoins et disparaissent lorsque le serveur a répondu à leur requête).

Ces entités sont décrites par des **variables** (dynamiques) ou **paramètres** (statiques) que l'on appellera **attributs**. Par exemple, pour une entité « avion », un paramètre pourra être son altitude plafond et une variable son altitude courante.

Notons qu'il existe des variables ou des paramètres s'appliquant à l'ensemble du système. On les qualifie alors de **globaux**. Ce serait le cas de l'altitude ou du plafond d'un avion si celui-ci constituait le système.

5.2.1.4 Environnement d'un système

L'environnement d'un système est l'ensemble des entités et des variables associées à ces entités, qui agissent sur le système mais ne sont pas affectées par son évolution.

On dira au contraire qu'une entité appartient au système étudié si elle agit sur lui et est influencée par lui.

La frontière entre le système et son environnement n'est pas toujours claire, et tient souvent de l'arbitraire, mais son identification précise est essentiel pour la modélisation.

L'environnement d'un système comprend notamment le domaine naturel (terrain, atmosphère, océan, espace).

5.2.2 Analyse dynamique du système

5.2.2.1 État d'un système

L'**état d'un système** est constitué de l'ensemble minimal des variables globales et des attributs de ses entités qui permet de prédire son évolution en l'absence d'aléas.

Les variables correspondantes sont appelées **variables d'état**.

Les paramètres ne sont pas des variables d'état : le fait qu'un avion ait un plafond de 30 000 pieds ne donne aucune indication sur l'état du système. De même, la variable « nombre de kilomètres parcourus depuis le décollage » ne décrit pas l'état du système. La vitesse et l'altitude de l'avion sont en revanche des variables d'état.

On qualifiera d'**événement** toute cause qui provoque un changement discontinu des variables d'états, qu'elle soit interne ou externe au système. Par exemple, la mise en route du moteur d'un avion, ou l'explosion de la charge militaire d'un missile près de l'avion.

5.2.2.2 Classification des variables

Toutes les variables n'ont donc pas le même rôle. Lorsqu'on analyse un système et ses entités, il est donc nécessaire d'identifier et de classer ces variables.

Prenons comme exemple un avion, et considérons les variables suivantes :

- Attitude (**vecteur** de 3 réels, roulis, tangage, lacet, exprimés en degrés)

- Position (vecteur de 3 réels, X, Y en mètres, Altitude en pieds)
- Vitesse (**réel**, en km/h)
- Plafond (réel, en pieds)
- Température de l'air (en °C)
- Distance parcourue depuis le décollage (réel, en kilomètres)
- État du moteur (**énumération** de valeurs discrètes : {arrêt, marche, démarrage})
- Train sorti (**booléen** : {non,oui}).
- Année de mise en service (nombre **entier**, de 1900 à 2099)
- Temps courant (réel, en secondes depuis la date du décollage)
- Date de décollage (vecteur de trois entiers, heures, minutes, secondes)
- Nom du pilote (**chaîne de caractère**)
- Cap à suivre (réel, en degrés)
- Nombre d'avions en attente au-dessus de l'aéroport (entier)
- ...

Remarquons que nous avons systématiquement précisé le **type de variable** (réel, entier, énuméré, booléen, …) et les **unités de mesure** utilisées. En effet, les utilisateurs ou les fichiers de données n'utilisent pas systématiquement les unités MKSA[56]. Il faut donc être prudent, car entre pieds et mètres il y a un facteur 3. Une sonde martienne de la NASA se serait écrasée sur Mars récemment pour cette raison !

On remarquera également dans notre exemple pourtant simple que les mélanges arrivent vite. Ainsi, l'altitude est en pieds alors que la position est en mètres et les distances mesurées en kilomètres. La date courante est un nombre réel, qui donne le nombre de secondes écoulées depuis le décollage, alors que la date de décollage est un ensemble de trois entiers donnant l'heure de l'événement. Ceci peut paraître compliqué et farfelu, et pourtant ce sont des choix courants dans une simulation d'avion.

Typiquement, on utilise pour les paramètres entrés ou lus par l'utilisateur les unités les plus usuelles, mais les calculs internes seront faits avec les unités MKSA. Par exemple, on utiliserait des mètres pour l'altitude, et des degrés Kelvin pour la température, mais lors d'une opération d'affichage, on reviendrait aux pieds et °C.

Une suggestion de classification fonctionnelle est présentée par la Figure 75.

- **Variable exogène** : extérieure au système (ex. : température de l'air)
- **Variable endogène** : interne au système (ex. : vitesse de l'avion)

Le temps est un cas particulier sur lequel nous reviendrons. Il peut être exogène (cas d'un simulateur temps réel lié à une horloge), ou endogène (si c'est le modèle qui en commande l'avance, ce qui est typiquement le cas d'une simulation à événements discrets).

Nous pouvons raffiner l'analyser d'une variable exogène :

- **Variable de décision** : sert à l'opérateur à influer sur l'évolution du système (ex. : le cap à suivre, s'il est imposé dynamiquement par l'opérateur).

[56] Unités de base du système international : mètre, kilogramme, seconde, ampère.

- **Variables non contrôlées** : il s'agit de variables qui sont déterminées par un modèle externe au système. Elles peuvent être déterministes (ex. : température de l'air, si elle est calculée à partir des conditions météo) ou aléatoires (ex. : le nombre d'avions en attente, qui est généralement un phénomène stochastique).

De la même façon, nous distinguerons dans les variables endogènes :

- **Variables d'état** : comme nous l'avons déjà vu, elles décrivent l'état du système (ex. : vitesse, position, attitude...).
- **Variables statistiques** : elles ne sont pas indispensables pour caractériser l'état du système, mais peuvent s'avérer utile pour le fonctionnement du modèle, l'information de l'opérateur ou le dépouillement ultérieur des résultats de la simulation (ex. : la distance parcourue).

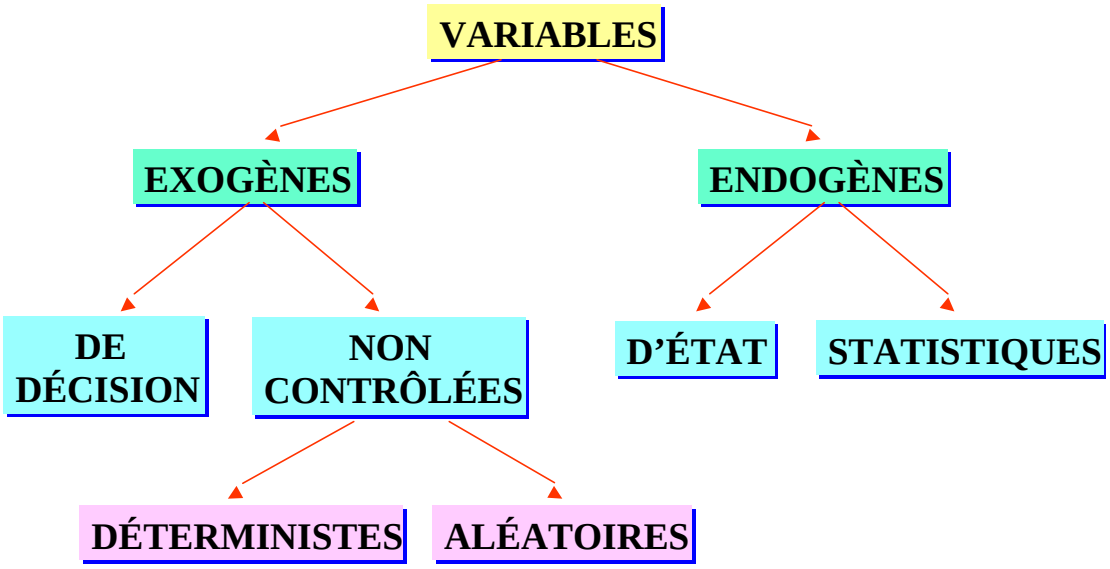


Figure 75: Classification des variables d'un système

Enfin, une distinction pourra être faite suivant la durée de vie des variables :

- **Variables persistantes** ou **permanentes** : elles existent et ont une valeur significative pendant toute la durée d'une exécution la simulation.
- **Variables temporaires** : elles n'existent ou n'ont de signification que pendant certains intervalles de temps.
- **Variables transitoires** : elles apparaissent que durant un instant déterminé (ou un pas de temps pour une simulation à temps discret).

5.2.2.3 Détermination des lois d'évolution

Lorsque les variables d'état sont identifiées, il faut déterminer les lois physiques et comportementales qui régissent leur évolution.

Si on note $X_i(t)$ l'ensemble des n variables d'état (avec $i \in \{1,\ldots, n\}$) et $S(t)$ l'état du système à un instant donné, cela revient à trouver les p fonctions F_j telles que :

$$S(t+d) = \{F_j(t, X_i(t))\}$$

avec $i \in \{1,\ldots, n\}$ et $j \in \{1,\ldots, p\}$. On rappelle que : $S(t) = \{X_i(t)\}$.

Les fonctions F_j sont appelées **lois d'évolution** du système. Elles peuvent être discrètes (en ce cas *d* représente le pas de la simulation) ou continues.

Les fonctions peuvent être représentées sous une forme explicite :

$$y = f(t, x)$$

Ou sous forme d'équations différentielles, qu'il faudra alors intégrer pour obtenir explicitement les nouvelles valeurs des variables d'état :

$$\frac{dy}{dt} = f(t, x)$$

Dans tous les cas, il ne faut jamais oublier de préciser, pour chaque fonction, son domaine de validité, et de vérifier que le système ne peut, durant une exécution de la simulation, se retrouver hors de ce domaine. L'intersection des domaines de validité des lois d'évolution (et des éventuelles hypothèses de modélisation) forme le **domaine de validité du modèle**.

La détermination de ces lois d'évolution relevant du domaine technique du système étudié (électromagnétisme, thermodynamique, mécanique des structures, etc.), nous ne détaillerons pas plus cette étape.

5.2.3 Approches de conceptualisation

5.2.3.1 Décomposition en sous-système

L'approche la plus courante pour conceptualiser un système consiste à le décomposer en sous-systèmes qui eux-mêmes pourront, si nécessaires, être décrits en termes de composants. Cette approche ne date pas d'hier, puisqu'elle nous est suggérée dans le *Discours de la Méthode* de Descartes, et est à la base des langages de programmation structurés tels que Pascal, C ou Ada.

La démarche la plus naturelle est descendante (***top-down***) : on part du système global, et on l'éclate en sous-systèmes.

Toutefois, la démarche inverse (***down-top***) est très utilisée dans le cas où l'on dispose déjà d'une bibliothèque de composants : on cherche à reconstituer le système en partant de composants déjà disponibles, que l'on adaptera ou spécialisera éventuellement.

Il existe plusieurs approches pour modéliser ensuite les sous-systèmes : approche par flots (d'informations ou d'entités), fonctionnelle (séquence logique de fonctions devant être activées), comportementales (règles régissant les réactions du système à son environnement), par graphes états-transitions (on s'attache aux conditions et stimuli qui permettent au système de passer d'un état à l'autre), etc. Bien évidemment, chaque approche a un impact sur la vision interne du système.

5.2.3.2 Approche orientée objets

L'approche de modélisation orientée objets est aujourd'hui la plus en vogue. Elle permet en effet une représentation du monde réel plus facile à conceptualiser pour le

développeur. Le succès des techniques à objets a permis la disponibilité sur le marché d'un grand nombre de produits, langages et méthodes orientés objets.

L'approche objets est la plus intuitive. C'est pour cela que le premier langage orienté objets, SIMULA, est né dans le domaine de la simulation.

Le type de données structuré caractéristique des langages orientés objets est la **classe**. La classe comprend des **attributs** (variables décrivant la classe) et des **méthodes** (fonctions pouvant s'appliquer à cette classe). Les attributs d'une classe peuvent être masqués vis-à-vis de l'extérieur, leur manipulation ne pouvant alors se faire qu'à l'aide des méthodes. Ce procédé de regroupement de méthodes et données au sein d'une même structure s'appelle l'**encapsulation**.

En programmation orientée objet, les **objets** sont des **instances** d'une classe. Par exemple, on pourra définir la classe `Voiture`, dont une instance pourra être `Peugeot205`. La relation d'instanciation est une relation d'appartenance à une famille d'objets. Ainsi, on peut dire qu'une `Peugeot205` *est une* `Voiture`.

Dans les langages orientés objets, il est possible de dériver des nouvelles classes (classes dérivées) à partir d'une classe existante (classe de base ou classe parent). Chaque nouvelle classe héritera des méthodes et attributs de la classe de base, qu'elle pourra remplacer ou enrichir. Ce mécanisme appelé **héritage** est très puissant pour la **réutilisation du code**, qui est en fait l'objectif premier des techniques à objets. Il permet également d'abstraire des classes en regroupant leurs caractéristiques communes au sein d'une classe de plus haut niveau.

Ainsi, on peut définir une classe `Vehicule`, qui pourra ensuite être dérivée en classes `Avion`, `Voiture`, `Bateau`. On obtient alors une structure d'héritage arborescente :

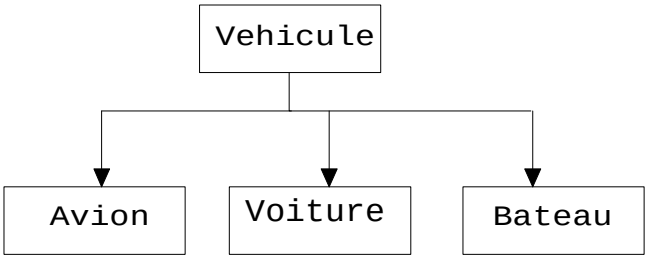


Figure 76 : Arbre d'héritage

Si la classe `Vehicule` a un attribut `Vitesse`, alors toutes les classes dérivées hériteront automatiquement de cet attribut. En revanche, la classe Avion disposera d'un attribut supplémentaire `Altitude`.

On remarque que si notre `Peugeot205` est une `Voiture`, elle est aussi un `Vehicule` et en possède les attributs, ainsi que les méthodes. Cette capacité à prendre plusieurs formes (en fonction du contexte) est le **polymorphisme**. Une méthode `Ralentir` de la classe `Vehicule`, par exemple, pourra s'appliquer tout aussi bien à un `Avion` qu'à un `Bateau`.

Il existe de nombreuses autres techniques en programmation orientée objets, mais il est généralement admis que pour être considéré comme orienté objet, un langage ou une méthode doit permettre ces trois mécanismes : encapsulation, héritage, polymorphisme.

Les techniques à objets sont très prisées de nos jours en programmation. Parmi les langages à objets les plus populaires, citons C++, Java, Ada95, Delphi (dérivé de Pascal), Eiffel, Smalltalk[57], Perl5, Python...

En simulation, la modélisation par objet est également très utilisée, car elle facilite le travail d'abstraction du système étudié : celui-ci est décomposé en famille d'objets (classes), ayant chacun des caractéristiques et des fonctionnalités (attributs et méthodes), et qui interagissent entre eux (appels de méthodes).

Il ne faut jamais perdre de vue les principaux objectifs des méthodes objets :

- **Créer une abstraction du monde réel de façon relativement naturelle.**
- **Faciliter la réutilisation du code développé, dans de bonnes conditions.**

Si un objet du monde réel peut être utilisé dans différents contextes, alors son abstraction informatique doit être réutilisable dans les mêmes conditions (sous forme de composant pour construire des objets plus complexes, ou un composant générique (classe de base) dont on dérivera par spécialisation de nouveaux objets (classes dérivées).

5.2.4 Déterminisme d'un modèle

Nous avons déjà vu ce qu'étaient un modèle déterministe et un modèle stochastique (voir §2.2.2).

Typiquement, le **modèle déterministe** sera utilisé pour la modélisation d'un phénomène physique dont les équations d'états sont bien connues et que l'on peut calculer sans ambiguïté les variables d'état. On rappelle qu'un tel modèle est caractérisé par la propriété suivante : pour une entrée donnée, les résultats en sortie sont toujours identiques.

Cela n'est pas le cas pour un **modèle stochastique** ou aléatoire, dont les résultats dépendent d'une ou plusieurs variables aléatoires. Les modèles stochastiques sont typiquement utilisés pour modéliser des phénomènes dans lesquels interviennent des facteurs aléatoires (une panne dans une machine ou la désintégration d'un noyau atomique lourd par exemple).

Mais parfois, l'utilisation de l'un ou l'autre type de modèle relève du **choix de modélisation**.

Par exemple, pour déterminer le résultat de l'explosion d'une charge militaire près d'une cible, différentes possibilités s'offrent à l'ingénieur :

- Simuler finement la gerbe d'éclats avec des modèles physiques détaillés de la forme et de la trajectoire de chaque éclat. Cela nécessitera des puissances de calcul importantes, mais on peut ainsi obtenir un modèle déterministe.
- Simuler les effets sous forme probabiliste, par exemple en déterminant la densité de probabilité d'impact d'un éclat sur une unité de surface en fonction de la distance à la source de l'explosion. Il s'agira d'un modèle aléatoire.
- Utiliser une fonction simple donnant directement les dégâts en fonction de la distance, par exemple : « à moins de 10 mètres, véhicule détruit, de 10 à 30 mètres,

[57] Smalltalk est souvent considéré comme la référence en matière de langages orientés objets, bien que certains réservent ce titre à Eiffel...

véhicule hors de combat, de 30 à 50 mètres, destruction antennes et optiques, au-delà pas de dégâts ».

Dans le cas des modèles aléatoires, le hasard est modélisé à l'aide d'un générateur aléatoire, suivant des algorithmes que nous détaillerons dans la section 5.3. En fait, il s'agit d'un générateur de **nombres pseudo aléatoires**, car ils suivent une série dont les éléments successifs sont liés par des relations mathématiques, de sorte que le comportement des simulations basées sur des modèles aléatoires peut souvent être reproduit. Cela a un intérêt considérable pour permettre le rejeu de situations, comme nous le verrons par la suite dans la section 5.3.

5.2.5 Le problème du temps

5.2.5.1 Généralités

Le temps est une variable essentielle dans une simulation. Il détermine en effet l'évolution de nombreuses variables d'état (par exemple la position d'un objet est fonction de sa vitesse et du temps) et est déclencheur d'événements (largage d'une bombe à l'instant *t*).

Le temps est, dans le monde réel, une variable continue. Pour être traité par informatique, il sera nécessaire de le discrétiser. Cette discrétisation, artificielle et arbitraire, peut être source d'imprécision et de problèmes de causalité.

5.2.5.2 Simulations temps réel

Il s'agit de simulation où le temps utilisé est généré par l'horloge de l'ordinateur. C'est le cas par exemple des simulateurs de vol pilotés.

Le problème est alors d'assurer les traitements nécessaires à la simulation *au pire* à la vitesse du temps réel, d'où l'utilisation de matériels performants et dédiés (générateurs d'images, unités de calculs puissantes...).

5.2.5.3 Simulation pas à pas

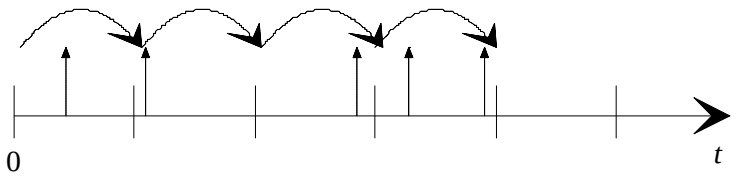


Figure 77: Avancement par pas de temps

5.2.5.4 Simulations à événements discrets

Les simulations à événements discrets voient leurs évolutions commandées par des événements, qui peuvent être aléatoires (voir Figure 78).

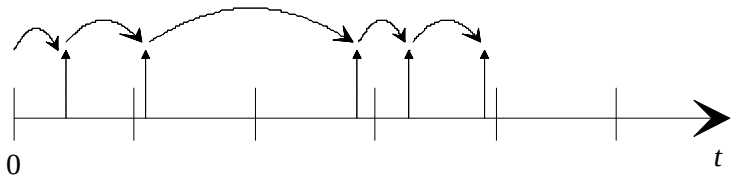


Figure 78: Avancement par événements

Ce type de simulation est très couramment utilisé, aussi nous lui consacrerons un chapitre entier.

5.2.5.5 Simulations mixtes

Les simulations dites « mixtes » (ou *combined* en anglais) sont régies à la fois par des lois d'évolution continues et discrètes.

Par exemple, la valeur d'une variable d'état continue peut, lorsqu'elle franchit un certain seuil, déclencher un événement (le largage d'un étage d'une fusée lorsqu'une certaine vélocité est atteinte…), ou un événement peut modifier les lois d'évolution physiques (un pilote agissant sur les commandes de son appareil…).

Il est alors nécessaire de définir précisément l'interface entre les variables continues et discrètes.

Les simulations mixtes sont de plus en plus courantes, les barrières entre le monde de la simulation continue et de la simulation à événements discrets s'estompant progressivement devant la nécessité du besoin[58].

5.2.5.6 La problématique de la simulation distribuée

En simulation distribuée, chaque fédéré (pour prendre la terminologie HLA) peut avoir son propre temps logique, ce qui pose généralement le problème de la synchronisation de chaque simulation, surtout pour les simulations non temps réel, pour lesquelles les différences de vitesse d'écoulement du temps peuvent être extrêmement variables.

En effet, si une simulation est en retard par rapport aux autres, elle est susceptible de générer un événement (ou une interaction) survenu dans le passé des autres fédérés, ce qui suppose de leur part, si l'on veut conserver la cohérence de la simulation, qu'ils puissent revenir dans leur passé (en annulant toutes les altérations de leur état postérieures à l'événement) pour prendre en compte. Bien évidemment, si quelques (rares) simulation font ainsi, il existe des méthodes plus sûres.

Une méthode couramment utilisée (dans ALSP et HLA par exemple) consiste à fournir aux fédérés un service de gestion du temps. Le fédéré, avant d'avancer du temps t au temps $t' > t$, demande l'autorisation à ce service (par `timeAdvanceRequest()`), lequel ne l'accorde (par `timeAdvanceGrant()`) qu'une fois certain qu'aucun événement n'arrivera avant t' . En pratique, on tient également compte du temps de traitement des événements par le fédéré, à travers le *lookahead*.

[58] Il ne faut pas se voiler la face : la communauté de la simulation souffrait bel et bien d'une telle division. Ainsi, jusqu'au début des années 90, les conférences d'été de la SCS étaient consacrées à la simulation continue et les conférences d'hiver à la simulation à événements discrets !

5.2.5.7 Variété de la gestion du temps

Les différents types de simulation n'ont pas la même façon de gérer le temps, ainsi :

- Une simulation pilotée ou « live » (matériel instrumenté sur le terrain) sera temps réel, avec des contraintes de temps particulièrement fortes pour la simulation live.
- Une simulation constructive fonctionnera soit par pas de temps, soit par événements (simulation à événements discrets, ou DES).

Il n'y a donc pas une méthode générale pour gérer le temps. Il faut choisir le mécanisme le mieux adapté à la situation.

Notons que les choses se compliquent très vite quand on souhaite créer une fédération de simulations distribuées (par exemple par HLA)... avec des simulations qui ne gèrent pas le temps de la même façon ! C'est là l'un des problèmes majeurs qui se posent lors de la mise au point d'un RTI (voir §9.5.4.3).

5.2.5.8 Dépendance au temps

Une simulation peut avoir deux types de relation vis-à-vis du temps :

- Elle peut être **contrainte par le temps** (*time-constrained*) : le temps est une variable exogène (par exemple une horloge matérielle), et est imposé à la simulation.
- Elle peut être **génératrice de temps** (*time-regulating*) : le temps est une variable endogène (par exemple en fonction d'événements).

Dans le cas particulier des simulations distribuées, on peut avoir des simulations dans les deux catégories à la fois. Par exemple, dans une fédération de simulations comprenant deux jeux de guerre jouant chacun un camp, chaque simulation, génératrice de temps, doit se synchroniser avec l'autre, et donc contrainte par le temps de l'autre simulation.

Dans le cas d'une fédération HLA, des fédérés peuvent être neutres par rapport au temps. C'est notamment le cas des outils de gestion de fédération, de journalisation de messages, etc. (voir la section sur HLA)

5.2.5.9 Temps réel

Dans une simulation **temps réel**, le temps simulé s'écoule à la même vitesse que le temps réel : une seconde simulée équivaut à une seconde écoulée dans le monde réel.

Il va sans dire que le respect du temps réel impose des contraintes particulièrement fortes à la simulation, comme tout système temps réel.

Ce type de simulation se rencontre notamment dans les cas où l'un des constituants de la simulation n'est pas contrôlé directement par le moteur de simulation (simulation pilotée, simulation hybride…).

5.2.5.10 Pas de temps

Dans une simulation à temps discret, on appellera **pas de temps** la durée de l'intervalle de temps entre deux calculs successifs de l'état du système.

5.2.5.11 Régulation par événements

Dans la simulation à événements discrets (voir chapitre 6), le temps de la simulation est la date du dernier événement lu dans la file d'attente des événements.

Les événements dans cette file doivent strictement être ordonnés chronologiquement, afin de respecter la **causalité** des événements traités.

5.2.5.12 Lookahead

Dans les fédérations de simulations distribuées (HLA, ALSP...), on associe à chaque fédéré régulateur de temps une durée de temps logique λ appelé *lookahead*, représentatif des temps de réaction du fédéré. Ainsi, dans HLA, pour un temps logique t, le RTI ne permettra pas au fédéré d'envoyer d'événement datés antérieurement à $t + l$. Le fédéré doit donc anticiper les événements qu'il génère.

L'intérêt du lookahead est surtout d'éviter les *deadlocks*[59]. En effet, prenons deux fédérés *A* et *B*, de temps logiques T_A et T_B ($T_A < T_B$), tous les deux à la fois régulateur de temps et contraints par le temps.

A peut générer des événements au temps T_A à destination de *B*, lequel en revanche ne peut envoyer à *A* d'événements à un temps supérieur à T_A (i.e. T_B) car alors il prendrait le risque de bouleverser l'ordre des événements. D'un autre côté, *B* ne peut avancer au delà de T_A, car il risquerait de recevoir un événement dans son passé. Aucun fédéré ne peut donc avancer son temps logique, et la fédération se trouve dans une situation de blocage.

Un moyen de remédier à cela est de permettre aux événements d'être transmis dans le désordre (i.e. par le RTI HLA). Mais ceci implique que si un fédéré reçoit un événement dans son passé, il soit capable de revenir dans un état antérieur pour le traiter correctement. Ceci est possible, mais loin d'être évident.

La méthode la plus courante est d'utiliser le lookahead. Il permet au RTI de déterminer la date minimale du prochain événement qu'il recevra d'un fédéré, ce qui lui donne le temps maximal auquel il autorisera les fédérés à avancer leur horloge.

En revanche, il y a des cas où un lookahead nul est nécessaire. Tel est le cas lorsque *A* et *B* doivent générer des événements à un même temps logique. Par exemple, si un objet géré par *A* entre en collision avec un objet de *B* : si les deux temps logiques ne sont pas strictement identiques, on pourra voir l'objet de *B* continuer sa route (le temps du lookahead) et être détruit par la collision... un peu plus loin, sans être localisé au même endroit que l'objet de *A* !

Dans une simulation avançant par pas de temps constants, le lookahead est bien entendu égal à la valeur de ce pas de temps.

Pour une simulation qui n'est pas régulatrice du temps, le lookahead n'a pas d'importance puisque qu'elle ne joue aucun rôle dans le calcul de la date courante.

[59] Situation de blocage de plusieurs processus qui s'attendent mutuellement. Par exemple, un processus *A* verrouille une ressource *R*. Un processus *B* ayant besoin de cette ressource se met en attente. À un moment donné, *A* a besoin de résultats issus de *B* et les lui demande. Cependant, *B* est bloqué en attente de *R*. Mais comme *A* ne libérera pas *R* avant d'avoir eu la réponse de *B*, il y *deadlock*.

5.2.6 Le problème des données

5.2.6.1 Destination des données

Ce serait une grande erreur, lors du développement d'une simulation, de se focaliser uniquement sur la modélisation conceptuelle et le codage du modèle, car ceux-ci ne valent rien sans les données qui les alimentent.

Les données jouent en effet un grand rôle dans une simulation :

- État initial du système : valeurs initiales des variables d'état.
- Paramètres du système : elles permettent de préciser les conditions d'utilisation du modèle et notamment l'adaptation d'un modèle générique à un cas particulier. Par exemple, un modèle générique d'avion de ligne pourra être paramétré pour être tantôt un Airbus A340, tantôt un Boeing 767.
- Données de validation : elles servent à calibrer le modèle et à le valider.
- Données de sortie : sauvegardes, stats, journaux, rejeu...

5.2.6.2 Collecte des données

La collecte des données nécessaires pour une simulation en quantité et qualité suffisantes est l'une des tâches les plus ardues du processus de modélisation et simulation. Dans certains cas, leur collecte est même impossible (phénomène physique ne pouvant être reproduit sur terre…) ou non envisageable (trop dangereux, trop coûteux…).

Il existe de nombreuses sources de données, mais chacune ont leurs limitations :

- Documentation technique du système
- Expertise des ingénieurs
- Expérimentations réelles (sur un exemplaire du système ou extrapolation à partir d'une maquette)
- Résultats d'autres simulations
- Jeux de données préexistantes
- …

Dans tous les cas, les données doivent être contrôlées et validées. De même qu'un processus de VV&A est requis pour un modèle, un processus de VV&C (vérification, validation et certification) est nécessaire pour les données.

5.2.6.3 Représentation des données

Les données sont une source fréquente d'erreurs, comme nous l'avons vu.

Un certain nombre de précautions élémentaires doivent être prises pour s'assurer de leur exactitude et de leur conformité à ce qui est attendu par la simulation.

En particulier, il faut choisir un format de fichier de données indépendant de la plate-forme et des unités utilisées par la simulation.

L'**indépendance de la plate-forme** passe par l'utilisation d'un fichier texte (code ASCII standard) qui a l'avantage d'être lisible et modifiable directement dans un éditeur. Toutefois, cette méthode ne permet pas toujours d'obtenir des performances suffisantes, notamment pour de grandes quantités de données. Dans ce cas, s'il y a utilisation d'un fichier binaire, il faut d'affranchir des problèmes de portabilité (*endianess*[60], représentation des réels, format des chaînes de caractères…) en définissant un format universel (ou un format standard existant) et en écrivant des routines d'accès à ces fichiers qui réaliseront alors la conversion vers et depuis les représentations internes propres à la plate-forme utilisée.

Si on utilise un logiciel externe pour gérer les données (par exemple un SGBDR), on choisira de préférence un produit portable (Oracle, MySQL, PostgreSQL, etc.), afin la garantir la pérennité des données.

La **cohérence des unités** utilisées par le fichier et par la simulation n'est jamais évidente. Si dans le fichier, l'altitude indiquée est « 3000 », est-ce 3 000 mètres ou 3 000 pieds ?

La meilleure solution consiste à définir un format de fichier dans lequel l'unité est systématiquement accouplée à la valeur. Quand le fichier est lu par la simulation, les routines de lecture pourront faire la conversion automatiquement vers le système d'unités utilisé par la simulation[61], évitant ainsi toute erreur d'interprétation.

Parmi les standards existants, **XML** (*eXtended Markup Language*) se démarque par sa lisibilité, son universalité et le large nombre d'outils qui le supportent. Il commence à être utilisé dans le monde de la simulation, et devrait connaître un large succès.

5.2.7 Les aspects informatiques

5.2.7.1 De la non-transparence de l'informatique

Bien des architectures logicielles modernes telles que HLA ou des langages de haut niveau comme Ada ou Java visent à masquer l'informatique sous-jacente (matériel, protocoles réseaux, etc.). Ceci est une bonne chose, car le développeur de modèles est généralement un expert technique d'un domaine particulier (optronique, météorologie, biologie...), et non un informaticien chevronné.

Toutefois, il serait illusoire de penser que l'informatique puisse être transparente à 100%. Ainsi, HLA, contrairement à DIS, rend invisible pour le développeur d'applications les couches réseau. Mais néanmoins, les performances du réseau auront une influence déterminante sur les performances d'une fédération distribuée sur ce réseau !

De même, certains choix de modélisation peuvent avoir une grande influence sur les performances de la simulation, parce qu'elles utiliseront moins de mémoire, ou seront plus facilement parallélisables sur une architecture multiprocesseurs.

Le choix du langage et de l'atelier de développement aura, lui, un impact direct sur les délais (donc les coûts) et la qualité du code.

L'informatique n'est donc jamais transparente, aussi faut-il l'intégrer dans les choix d'architecture, d'outils et de plate-forme.

[60] Ordre des octets d'un mot en mémoire : octet de poids fort avant l'octet de poids faible, ou le contraire.

[61] Une bonne pratique est d'utiliser, pour les calculs, les unités MKSA du système métrique international.

5.2.7.2 Rappels sur l'architecture des ordinateurs

Les principaux constituants d'un ordinateur (voir Figure 79) sont :

- Le processeur (*Central Processor Unit* ou CPU), qui exécute les instructions des programmes en langage machine, et comporte une unité de calcul arithmétique et logique (ALU). Il est secondé par une unité de calcul en virgule flottante (FPU) ainsi qu'une unité de gestion de la mémoire (MMU). La plupart des processeurs modernes intègrent tous ces composants.
- La mémoire, qui est soit figée et permanente (mémoire morte, ou ROM, contenant le logiciel de base de l'ordinateur, ou BIOS), soit dynamique et modifiable (mémoire vive, ou RAM, contenant le système d'exploitation, les programmes applicatifs et les données).
- Le sous-système graphique, qui permet d'afficher les résultats sur une console graphique. De nos jours, ce composant est devenu très complexe, car son rôle ne se limite plus à la simple production de signaux vidéo, mais au calcul du rendu de scènes 2D et 3D. Les cartes graphiques et générateurs d'images sont devenus des ordinateurs à part entière, embarquant un ou plusieurs processeurs et de grande quantités de mémoire vidéo dédiée
- Le sous-système d'entrées/sorties gère les communications avec les périphériques tels que le clavier, la souris, le réseau, les unités de stockage (disques durs, CDROM...), etc.

Bien entendu, ceci n'est qu'une version simplifiée, voire simpliste, de l'architecture d'un ordinateur, mais elle montre qu'un ordinateur est un tout, un regroupement de composants dont chacun va contribuer aux performances finales de la machine.

La simulation étant souvent amenée à effectuer de lourds calculs, à manipuler des quantités importantes de données et à restituer graphiquement des environnements synthétiques complexes, la question des performances est un souci majeur de l'ingénieur en simulation.

Bien qu'il n'entre pas dans notre propos de disserter sur l'architecture des ordinateurs, nous allons néanmoins voir par la suite quelques techniques utilisées pour optimiser les performances du matériel et des logiciels, compte tenu de l'impact qu'elles peuvent avoir sur la simulation.

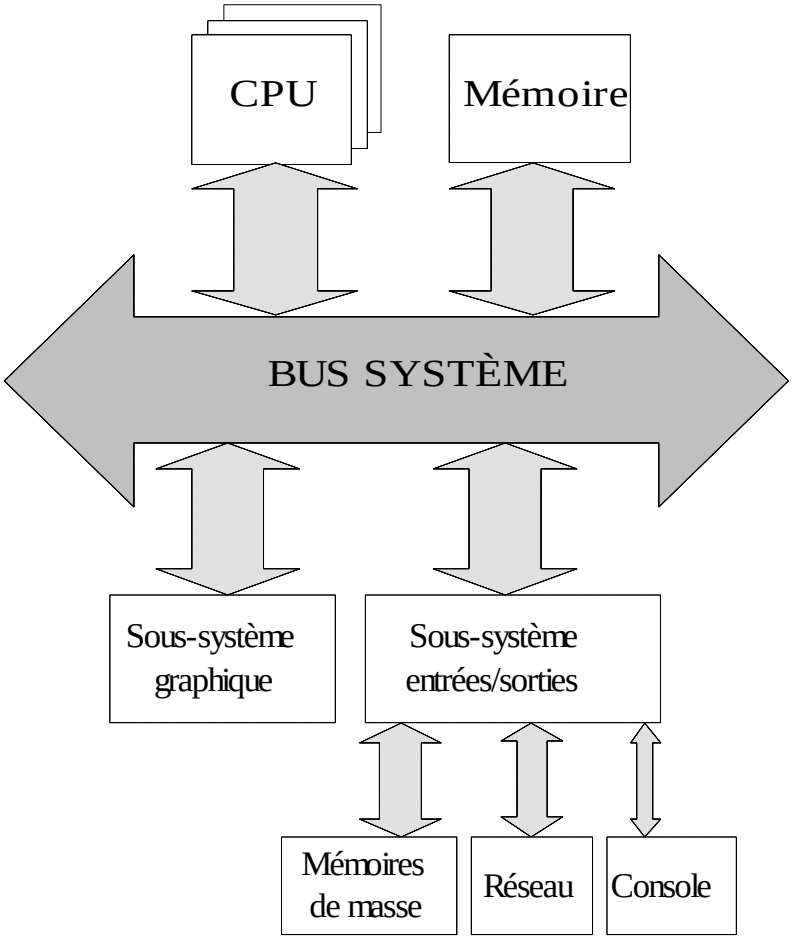


Figure 79: Architecture typique (simplifiée) d'un ordinateur

5.2.7.3 Les stratégies pour améliorer les performances des ordinateurs

Nous pourrions écrire des milliers de pages sur le sujet, mais ce n'est pas l'objet de ce cours, aussi nous allons nous borner, pour le matériel, à définir quelques termes et idées clés sur les performances des matériels.

- **Processeurs RISC et CISC** : les processeurs CISC (*complex instruction set*) utilisent un jeu d'instructions pléthorique (plusieurs centaines d'instructions). Nombre d'entre elles réalisent des opérations évoluées, mais la complexité de ces processeurs rend difficile l'écriture des compilateurs, qui ont du mal à exploiter pleinement les instructions de haut niveau. C'est pourquoi la plupart des fabricants se sont orientés vers une architecture RISC (*reduced instruction set*) qui comporte beaucoup moins d'instructions, mais celles-ci sont exécutés plus rapidement, et sont mieux exploitées par les compilateurs.
- **Mémoire cache** : le processeur fonctionne souvent beaucoup plus vite que la mémoire centrale de l'ordinateur. Quand il exécute une instruction nécessitant une lecture ou une écriture en mémoire vive (RAM), il doit attendre plusieurs cycles d'horloge que la donnée lui soit fournie. Une possibilité serait d'utiliser de la mémoire aussi rapide que le processeur, mais cela coûterait un prix exorbitant pour les grandes capacités. En pratique, on utilise un peu de mémoire rapide pour faire un tampon appelé « cache » entre le processeur et la mémoire principale. Quand le

processeur veut lire une donnée, il la demande au cache qui, la première fois, va aller la chercher en mémoire principale pour la donner au processeur, mais en même temps il va la stocker dans sa mémoire rapide, ainsi que les octets qui suivent, qui ont une forte probabilité d'être lus à leur tour par la suite. Lorsqu'une donnée est présente dans ce cache, elle est fournie immédiatement au processeur. En simulation, le cache mémoire a une grande importance en raison du nombre d'accès en mémoire. Il existe même des stratégies de stockages des données en mémoire permettant de favoriser les chances de succès d'une lecture dans le cache. Pour les écritures, le cache sert également d'intermédiaire pour accélérer l'opération. Notons qu'un tel principe est appliqué également entre les mémoires de masse (disque dur…) et la mémoire principale de l'ordinateur.

- **Mémoire virtuelle** : système permettant d'utiliser une partie d'un disque dur (fichier d'échange ou de « *swap* ») pour étendre la mémoire physique de l'ordinateur. Ainsi, un ordinateur avec 256 Mo de mémoire physique et un fichier de swap de 256 Mo apparaîtra à l'utilisateur comme ayant 512 Mo de mémoire. Le système d'exploitation va utiliser en priorité la mémoire physique pour répondre aux demandes d'allocation de mémoire des programmes. Quand il ne reste plus de place en mémoire physique, le système sauvegarde sur le disque une partie de la mémoire physique sur le disque dur afin de libérer de la place. Lorsque ces données « déchargées » de la mémoire physique seront requises, le système libérera une partie de la mémoire physique de la même façon que précédemment et rechargera les données en mémoire physique. La qualité des algorithmes de gestion et le paramétrage du fichier de swap sont déterminants.
- **Multiprocesseurs** : pour aller plus vite, la façon la plus simple est d'augmenter la vitesse du processeur (fréquence d'horloge), qui multiplie de façon proportionnelle le nombre d'instructions qu'il pourra exécuter par seconde. Toutefois, surviennent rapidement des problèmes électriques et thermiques. Une autre solution est d'utiliser plusieurs processeurs au sein d'une même machine : plutôt que de doubler la fréquence d'un processeur, on en fait fonctionner deux en parallèle. On trouve ainsi des calculateurs équipés de… plusieurs milliers de processeurs ! Cependant, leur exploitation efficace est ardue, et la puissance réelle est toujours très inférieure à l'impressionnante « puissance crête » indiquée dans les brochures.
- **Distribution** : une variante consiste à utiliser plusieurs machines distinctes dont on met en commun les ressources. L'exécution des applications est alors distribuée sur les différentes machines, par exemple à travers un réseau. Chaque machine reste néanmoins autonome. Dans le *clustering*, en revanche, les machines sont regroupées au sein d'un cluster qui est vu comme une machine unique.
- Processeurs secondaires dédiés : Ce que nous avons décrit était le *symmetric multiprocessing* (SMP), dans lequel tous les processeurs ont le même rôle. Un autre approche consiste à spécialiser les processeurs. C'est le cas maintenant dans tous les ordinateurs : il y a un processeur (voire plusieurs !) pour gérer l'affichage graphique (opération très gourmande en calculs), un autre pour le son (processeur de traitement de signal DSP), etc.
- **Benchmarks** : pour avoir une métrique des performances d'un ordinateur, on a recours à des logiciels de tests appelés benchmarks, qui exécutent un ensemble d'opérations supposées être représentatives d'une certaine utilisation de l'ordinateur, et en déduisent une mesure de vitesse : SpecInt, SpecFp, Whetstone, Winstone, etc. Les résultats de ces benchmarks est cependant à prendre avec prudence, car chaque

utilisation a tendance à être spécifique. De plus, quantité d'éléments peuvent influer sur les performances d'un ordinateur, qui est un ensemble complexe d'éléments matériels et logiciels…

5.2.7.4 Le choix des langages et outils

Les premiers programmes informatiques (dans les années 40-50) étaient écrits directement en langage machine, code binaire directement interprétable par le processeur. De ce fait et de l'absence totale de convivialité des machines de l'époque, ils étaient très ardus à développer, malgré leur taille modeste.

Au milieu des années 50, on améliora considérablement la tâche des développeurs en leur permettant d'écrire leurs programmes dans un langage de plus haut niveau, proche de préférence de la langue courante (COBOL, Pascal...) ou d'un formalisme particulier (APL, LISP...). Ces langages apportèrent de nombreux avantages, outre la simplification des développements, tels que la portabilité et une maintenabilité considérablement améliorée.

Comme le processeur de l'ordinateur ne comprend que le langage machine, on passe de ce langage de haut niveau, abstrait, à un code binaire exécutable par l'intermédiaire d'un interpréteur (si la traduction se fait « à la volée ») ou d'un compilateur (si elle se fait en une seule fois avant toute exécution). C'est le principe de la compilation qui est le plus utilisé aujourd'hui, notamment pour des questions de performances[62].

Il existe aujourd'hui des milliers de langages, parfois très spécialisés. Parmi les langages génériques, les plus couramment rencontrés en simulation sont :

- Langage séquentiels : les instructions se suivent linéairement et des instructions de branchement permettent de sauter d'un point du programme à un autre. Ces langages sont maintenant considérés comme obsolètes en raison de leur lisibilité médiocre notamment, mais FORTRAN, l'un des premiers langages informatiques, subsiste toujours, grâce à son efficacité pour le calcul numérique et l'existence de compilateurs parallélisants de très bon niveau).
- **Langages structurés** : le programme est décomposé en fonctions représentant typiquement les traitements à effectuer, et le code est structuré hiérarchiquement. Exemples : Pascal (aujourd'hui passé de mode, sauf dans son avatar très propriétaire, Delphi), C, Ada83.
- **Langages à objets** : les données et les programmes sont encapsulés dans des structures appelées « classes ». De nombreux mécanismes facilitent la réutilisation ultérieures de ces classes. Ce sont les langages les plus populaires actuellement : C++, Ada95, Java...

[62] Le cas du langage Java est un peu particulier, puisqu'il combine un compilateur vers un pseudo-code portable de bas niveau, le JavaCode, qui est interprété lors de l'exécution.

On trouve également des langages dédiés à la simulation :

- **Langages de simulation** : ils sont en nette régression, au profit de langages génériques plus facilement disponibles et moins onéreux (C++ notamment), bien que moins bien adaptés, ou de langages propriétaires. Citons parmi les plus connus des langages de simulation : GPSS, SIMSCRIPT, SIMULA, QNAP2... SIMSCRIPT, par exemple, fournit des mécanismes de files d'attente, une gestion événementielle du temps, des fonctions statistiques, etc., et permet de développer rapidement des simulations à événements discrets, mais il n'est commercialisé que par une seule société, ce qui ne lui apporte pas les garantie de pérennité d'un langage universellement répandu comme C++. Notons qu'il existe des bibliothèques dédiées à la simulation destinées à des langages génériques (C, C++ et FORTRAN notamment).
- Langages propriétaires de simulation : ce sont des langages permettant de contrôler des logiciels de simulation, tels que Matlab, Scilab, Simulink... Ils connaissent un certain succès en raison de la simplicité et de la rapidité de modélisation qu'ils offrent.

Le choix d'un langage doit être mûrement réfléchi. Il n'existe pas de langage parfait et universel, chacun a ses qualités et ses défauts.

Ainsi, le C est performant, et bien adapté à la programmation système, et bénéficie de bibliothèques étendues, mais il reste peu puissant et la fiabilité des programmes C laisse souvent à désirer en raison du laxisme des contrôles à la compilation (pas de typage fort par exemple) et de sa syntaxe parfois cryptique, et il n'est pas orienté objets.

Le C++ corrige certain de ces défauts, mais au prix d'une grande complexité, notamment pour la gestion de la mémoire.

Le langage Ada95 est également compliqué, mais favorise l'écriture de programmes fiables, d'où son utilisation pour des logiciels critiques en aéronautique (logiciels du Rafale par exemple), aérospatiale (logiciels de la fusée Ariane, de satellites...), etc. Malheureusement, Ada95 ne bénéficie pas de la notoriété du C++ et donc des bibliothèques de composants de ce dernier.

Java a tenté, semble-t-il, une synthèse entre le C++ (dont il reprend la syntaxe, hélas) et Ada95, et le pari est plutôt réussi. Sa principale qualité est aussi son principal défaut : il est compilé dans un pseudo-code qui est interprété au moment de l'exécution au sein d'une machine virtuelle. Ceci le rend portable au niveau du code objet, ce qui est remarquable, mais grève ses performances par rapport à un langage compilé en code machine natif comme C++ ou Ada95. Néanmoins, Java commence à être utilisé de plus en plus souvent en simulation, notamment pour la réalisation d'interfaces graphiques (portables), domaine dans lequel il a peu de concurrents.

Le langage FORTRAN est dépassé, mais subsiste toujours en raison du grand nombre de codes de simulations développés avec ce langage depuis la fin des années 50. Il reste très performant pour le calcul numérique, notamment sur des calculateurs parallèles.

Les langages spécialisés en simulation facilitent considérablement le codage des modèles, mais sont généralement destinés à un créneau particulier de la simulation, et

doivent souvent être employés conjointement à un autre langage (C ou C++ par exemple) pour le développement de l'IHM[63], des interfaces avec d'autres logiciels, etc.

Comment choisir ? Il n'est pas possible ici de conseiller tel ou tel langage, car, répétons-le, il n'existe pas de langage parfait et universel. Il faut poser les exigences et choisir en conséquence : vitesse de codage, facilité de mise au point, coût et qualité des compilateurs et outils, vitesse d'exécution, fiabilité, réutilisabilité, portabilité, disponibilité des outils et bibliothèques, disponibilité des compétences, etc.

5.2.7.5 Les stratégies pour améliorer les performances des logiciels

Là encore, il n'y a pas de solution miracle. La principale erreur à ne pas commettre est de se focaliser uniquement sur le code de simulation ou la puissance du processeur qui l'exécute. Il faut considérer le **système de simulation complet** : processeur, mémoire vive, mémoires de masse (performances de la mémoire virtuelle...), réseau (pour la simulation distribuée ou l'accès à des données externes), système d'exploitation, code de simulation...

En cas de problèmes de performances, voici quelques-unes des voies d'amélioration possibles (liste non exhaustive !) :

- Augmentation de la puissance du processeur : notamment sur un PC, il est possible, pour une somme modique (100 à 1000 € typiquement), de changer le processeur, par exemple pour passer à une fréquence d'horloge supérieure, sans pour autant modifier le reste de la machine. Et même si cela était le cas, mieux vaut parfois investir dans une nouvelle machine plutôt que de perdre plusieurs jours sur des campagnes de calcul.
- Multiplication du nombre de processeurs : on trouve maintenant des machines bi-processeurs bon marché. Des systèmes grand publics comme Windows NT ou Linux peuvent gérer respectivement 4 et 32 processeurs. Pour que l'utilisation de plusieurs processeur soit optimale, il est souhaitable que le logiciel soit *multithreads*, c'est-à-dire prévu d'emblée pour l'exécution parallèle sur un système multiprocesseur (voire un cluster, voir ci-dessous).
- Clustering : de plus en plus, les clusters (regroupement des ressources de plusieurs ordinateurs au sein d'un réseau local) apparaissent comme une alternative aux gros et coûteux calculateurs. En construisant un cluster à partir de plusieurs PC Linux bon marché, on peut obtenir un système d'une puissance importante. Une approche qui fait également son chemin est, grâce à Internet, l'utilisation de la puissance CPU non utilisée de stations de travail réparties sur un vaste domaine, par exemple en incluant un logiciel de calcul dans un économiseur d'écran. Les principales applications qui ont utilisé ce principe sont la cryptographie (décodage de messages par la force brute) et... l'analyse de signaux captés par des radiotélescopes dans l'espoir de capter des signaux extra-terrestres (programme Seti@Home).
- Utilisation de la distribution : exécuter un programme monolithique sur plusieurs processeurs est un exercice délicat, aussi est-il parfois plus simple de le scinder en plusieurs composants autonomes (par exemple un modèle d'avion et un modèle de missile) qui constitueront autant de fédérés au sein d'une application de simulation distribuée (voir section 9.5).

[63] Interface Homme-Machine, souvent employé pour désigner, de façon restrictive, l'interface utilisateur graphique (en anglais GUI pour *Graphic User Interface*).

- Augmentation de la mémoire physique (RAM) : les logiciels de simulation sont souvent très gourmands en mémoire (création dynamique d'objets, calculs sur de larges matrices...), aussi il s'agit d'un facteur très important de performances : si la mémoire physique d'un ordinateur vient à manquer, le système d'exploitation l'étend en utilisant une partie du disque dur (*swapping*), ce qui dégrade considérablement les temps de réponse (de l'ordre de quelques millisecondes au lieu de quelques nanosecondes !).
- Utilisation des options d'optimisation du compilateur : lorsque le programme est au point (et seulement alors), on peut gagner sensiblement en vitesse d'exécution en retirant certaines contrôles effectués durant l'exécution (*run-time*). Certains compilateurs (FORTRAN notamment) dits « parallélisants » peuvent générer du code optimisé pour l'exécution sur des machines multiprocesseurs.
- Optimisation « manuelle » du code : parfois, une simple réorganisation d'une boucle ou d'un tableau en mémoire peuvent engendrer des gains considérables. Mais il s'agit là d'un exercice délicat, qui requiert une connaissance profonde du fonctionnement de la machine, du système d'exploitation et du compilateur. On pourra s'aider d'un logiciel d'analyse de temps d'exécution du code (*profiler*). Gare à l'obsession de l'optimisation : un programme ne perd pas forcément son temps là où on pense ! Il faut également peser le coût de l'optimisation, qui peut revenir plus cher que l'achat d'un processeur supplémentaire.
- ...

5.2.7.6 Gestion de la mémoire

Une simulation manipule des quantités souvent importantes d'entités dynamiques.

Lors de la création d'une entité statique[64], le système alloue à l'application de simulation la mémoire correspondant à cette entité et à ses attributs une fois pour toute au moment de sa création.

En revanche, pour les entités temporaires[65], cette mémoire est requise, puis libérée à chaque création/destruction d'une entité. Ce qui n'est pas sans poser des problèmes de gestion de la mémoire.

En effet, l'allocation dynamique de mémoire requière une certaine stratégie sous peine de pertes catastrophiques de performances. La Figure 80 illustre l'un des problèmes rencontrés :

a) Nous avons ici schématisé une portion de la mémoire d'un ordinateur exécutant un programme.

b) Le programme en question a demandé l'allocation d'un bloc de mémoire de grande taille.

c) Deux autres blocs ont été alloués, à la suite du bloc précédent.

d) Les données pour lesquelles le bloc alloué en (b) ne sont plus utiles. Le programme les a donc détruites et a libéré la mémoire correspondante. Que va-t-on faire du « trou » ainsi fait dans la mémoire allouée ?

[64] Exemples de structures statiques : variable entière, tableau, chaîne de caractères de taille fixe...

[65] Exemples de structures dynamiques : listes, arbres, chaînes de caractères de taille variable, instances de classes C++ créés par `new`...

e) Le programme demande au système l'allocation d'un nouveau bloc de mémoire, de petite taille cette fois. Que va faire le système ? Allouer une partie du « trou » de mémoire libre (ce qui a été fait sur le schéma) ? Continuer à allouer séquentiellement la mémoire à partir du dernier bloc alloué ? Mais alors, que faire une fois arrivé à la fin de la mémoire ?

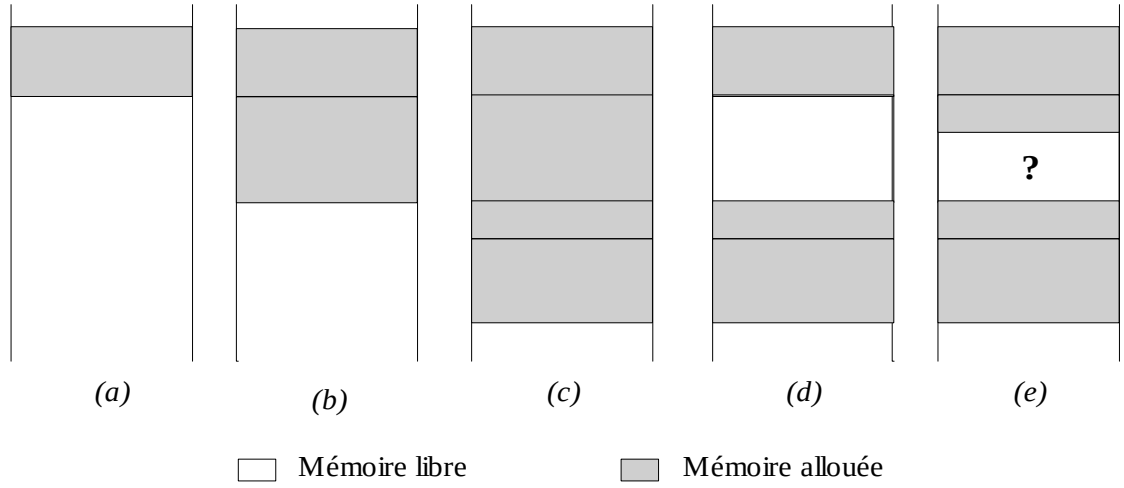


Figure 80: Allocation dynamique de mémoire

Le bon sens nous dit que, la mémoire étant limitée et chère, il faut essayer d'optimiser son occupation, et donc réallouer les « trous » provoqués par les libérations dynamiques. Oui, mais la gestion des zones disponibles finit par devenir coûteuse en temps de traitement au fur et à mesure que le taux de fragmentation de la mémoire augmente, puisqu'il faut maintenir des listes de ces zones et de leurs caractéristiques.

Les deux stratégies les plus courantes consistent à utiliser des algorithmes sophistiqués pour gérer la mémoire (cas de C ou d'Ada), ou alors à allouer les blocs de mémoire le plus simplement possible (l'un après l'autre), puis, lorsque la mémoire disponibles est trop faible ou que le processeur n'est pas utilisé, on déplace tous les blocs de façon à « recompacter » la mémoire. Cette dernière stratégie est appelée *garbage collecting* et est utilisée par exemple en LISP.

Lorsque la stratégie proposée par le système ne convient pas, l'utilisateur peut créer son propre programme de gestion de mémoire. Celui-ci demandera au démarrage de la simulation l'allocation d'un énorme bloc de mémoire qu'il se chargera d'allouer.

5.3 La simulation du hasard

La simulation du hasard est essentielle dans le monde de la simulation. Pour modéliser ce hasard, on utilise des générateurs de nombres aléatoires.

Il est difficile de trouver une définition parfaite d'un « nombre aléatoire ». On dira ici qu'une séquence est aléatoire si chaque terme est imprévisible à celui qui ne connaît pas l'algorithme, et que la répartition des termes soit uniforme et résiste à tous les tests statistiques traditionnels.

Avant la démocratisation de l'informatique, les statisticiens jetaient des dés, tiraient des cartes ou utilisaient des tables de nombres aléatoires. Ainsi, en 1927, L. Tippett publia une table de 40 000 nombres aléatoires tirés des données du recensement. En 1955, la firme RAND publia un énorme ouvrage contenant un million de nombres aléatoires. Ce monument fut utilisé par de nombreux statisticiens et experts de la simulation pendant quelques décennies.

Toutefois, ces tables avaient l'inconvénient d'être finies, et en même temps d'être trop grandes pour tenir dans l'étroite mémoire des ordinateurs des années 50-60.

Un moyen d'obtenir des nombres aléatoires à l'infini était d'utiliser des machines dédiées, telle celle de Kendall et Babington-Smith en 1939, qui produisit une table de 100 000 nombres. La table de RAND fut également calculée par une machine conçue à cet effet.

L'inconvénient de ces machines était leur non-déterminisme : il n'était pas possible de reproduire une séquence donnée, ce qui gênait la mise au point d'un programme ou l'analyse de l'exécution d'une simulation.

Le générateur aléatoire de Von Neumann

On prend un nombre de N chiffres X_0 appelé germe (par exemple $X_0 = 4321$). On élève ce nombre au carré ($X_0^2 = 8\ 671\ 041$), et on retient les N chiffres du milieu (6 710), qui donne la valeur suivante, et ainsi de suite.

On obtient alors :

4321	*6710*	*241*	*580*	*3364*	*3164*	*108*	*116*	*134*	*179*
320	*1024*	*485*	*2352*	*5319*	*2917*	*5088*	*8877*	*8011*	*1761*
1011	*221*	*488*	*2381*	*6691*	*7694*	*1976*	*9045*	*8120*	*9344*
3103	*6286*	*5137*	*3887*	*1087*	*1815*	*2942*	*6553*	*9418*	*6987*
8181	*9287*	*2483*	*1652*	*7291*	*1586*	*5153*	*5534*	*6251*	*750*
5625	*6406*	*368*	*1354*	*8333*	*4388*	*2545*	*4770*	*7529*	*6858*
321	*1030*	*609*	*3708*	*7492*	*1300*	*6900*	*6100*	*2100*	*4100*
8100	*6100*	*2100*	*4100*	*8100*	*6100*	*2100*	*4100*	*8100*	*6100*

On remarque qu'au bout d'un certain temps la suite dégénère, présentant un cycle répétitif de courte période.

L'algorithme de Von Neumann n'est donc guère satisfaisant.

5.3.1 Algorithmes déterministes

En 1946, John Von Neumann, mathématicien à l'origine de l'architecture d'ordinateur qui porte son nom (et qui est encore la plus répandue aujourd'hui, puisque c'est celle des PC et stations de travail), proposa un **algorithme déterministe** pour générer une suite aléatoire.

Ces algorithmes, qui n'utilisent initialement aucun élément aléatoire, produisent une suite prévisible de nombres. Ces nombres ne sont pas parfaitement aléatoires[66], ce qui

[66] On les qualifie de nombres « pseudo-aléatoires ».

peut être prouvé en appliquant sur un grand nombre d'entre eux des tests statistiques. Mais l'essentiel est que leur répartition apparaisse suffisamment aléatoire pour remplir le besoin. Avec l'algorithme de Von Neumann, N. Metropolis obtint une séquence de 750 000 nombres de 38 bits avant qu'une dégénérescence ne survienne (voir encadré).

5.3.2 Générateur à congruence linéaire

La plupart des générateurs aléatoires actuels utilisent un **générateur à congruence linéaire** (GCL), basé sur une formule de ce type :

$$X_{i+1} = (aX_i + c) \bmod m$$

a, *c* et *m* sont des constantes entières positives déterminées.

X_0 est appelée le **germe** (*seed* en anglais). Ce germe va déterminer toute la séquence, de sorte qu'il sera possible de rejouer plusieurs fois une même situation. Les bibliothèques de génération de nombres aléatoires ont généralement une fonction qui permet d'initialiser ce germe à une valeur donnée.

a et *c* sont respectivement le multiplicateur et l'incrément.

m est le modulo. Pour des raisons de performances de calcul, on aura tendance à choisir une puissance de 2 pour m, par exemple 2^{32}.

Si c = 0, on parle de GCL multiplicatif (GCLM), qui peut être intéressant pour améliorer les performances, bien que ce soit rarement nécessaire avec la puissance des ordinateurs modernes.

Prenons par exemple $a = 3$, $c = 7$, $m = 23$. On obtient les valeurs suivantes :

```
0       7       5       22      4       19      18      15      6       2       13
0       7       5       22      4       19      18      15      6       2       13
0       7       5       22      4       19      18      15
```

On remarque que $X_{11} = X_0$, ce qui est gênant ! On peut améliorer le score avec un choix plus pertinent, mais la suite sera toujours périodique, avec une période inférieure ou égale à *m*.

Les micro-ordinateurs actuels codant les entiers sur 32 bits, on pourra tout de même profiter d'une suite aléatoire de période 2^{32} soit 4 294 967 296, ce qui s'avère souvent suffisant. Toutefois, on remarque que nous n'obtenons qu'un ensemble discret fini de valeurs.

Il est important de retenir de ce CGL et du générateur de Von Neumann que la génération de nombres aléatoires est une question qu'il ne faut surtout pas sous-estimer, et qu'il faut se garder des apparences. Même un algorithme d'apparence compliquée peut dégénérer au bout d'un certain temps, avec un impact catastrophique sur le fonctionnement de l'application qui en dépend. Le grand informaticien Donald Knuth nous en donne un bon exemple dans le chapitre « Random numbers » de son célèbre ouvrage [KNUT71]. Par ailleurs, il nous donne des méthodes pour choisir avec plus de pertinence les valeurs de *a*, *b*, *m* pour un GCL.

Vous imaginez ce qui se passerait si le tirage du Loto reposait sur un système aléatoire déficient ? Il en est de même en simulation : les résultats des combats d'un jeu de guerre seraient biaisés, et pour peu qu'il s'agisse d'une simulation destinée à planifier une opération réelle, cela pourrait conduire à des décisions désastreuses et à des pertes humaines. Récemment, il a été ainsi découvert un défaut dans le générateur de nombres

aléatoire d'un programme de cryptographie populaire, qui risquait de compromettre les clés de chiffrement générées.

Il importe donc avant toute chose d'avoir un algorithme qui ne soit pas déterminé de façon aléatoire, et dont la validité soit démontrée mathématiquement. Ce n'est pas parce qu'un programme est capable de générer 1 000 nombres aléatoires apparemment « corrects » que le 1001ème ne va pas révéler une dégénérescence.

5.3.3 Autres méthodes

Congruence quadratique : $X_{n+1} = \left(dX_n^{\ 2} + aX_n + c\right) \bmod m$.

Générateur à récursivité multiple (GRM) : $Xn = \left(a_1 X_{n-1} + ... + a_k X_{n-k}\right) \bmod m.$

Générateur de Green, Smith et Klem : $X_{n+1} = \left(X_n + X_{n-k}\right) \bmod m$.

Générateur congruentiel inversif : $Z_n = \left(\tilde{X}_{n+1} \tilde{X}_n^{\ -1}\right) \bmod m$ ($\tilde{X}_i$ étant le $i^{\text{ème}}$ tirage non nul d'un GRM).

5.3.4 Utilisation d'un facteur aléatoire externe

Nous avons vu que le GCL ne donnait pas un résultat parfaitement aléatoire. On peut améliorer le résultat, au détriment de la reproductibilité, en injectant un bruit externe.

Ainsi, sur le micro-ordinateur Apple II, la procédure permettant la saisie d'un caractère au clavier fait tourner très rapidement un compteur en attendant la frappe de l'utilisateur. La valeur de ce compteur est ensuite incorporé dans le résultat de la fonction de calcul d'un nombre aléatoire. Peu coûteux en temps de calcul, ce système donnait de bon résultats, dans la mesure où le compteur de 16 bits faisait défiler toutes ses valeurs possibles (65536) en environ une seconde (l'Apple II est une très vieille machine, avec un processeur 8 bits à… 1MHz).

Certaines machines à sous des casinos utilisent un procédé similaire : un GCL tourne en permanence, des centaines de milliers ou des millions de fois par seconde, même lorsque la machine n'est pas utilisée. Lorsqu'un joueur appuie sur un bouton, il sélectionne les valeurs aléatoires requises.

5.3.5 Générateur idéal

En guise de conclusion, nous pouvons formuler les principales qualités d'un générateur aléatoire idéal :

- Démontrabilité
- Uniformité
- Indépendance
- Longue période
- Facilité d'implémentation
- Vitesse d'exécution
- Reproductibilité des tirages

5.3.6 Simulation des lois de répartition non uniformes

Notre GCL suit une loi de répartition uniforme. Pour lui faire suivre une autre loi, nous pouvons le transformer.

5.3.6.1 Méthode par rejet

On majore la densité par k*densité d'une loi facile à générer (par exemple, une loi uniforme).

5.3.6.2 Méthode par mélange ou composition

Décomposer la densité de la loi à générer en combinaison linéaire de lois faciles à générer :

$$f(x) = \sum_{i=1}^{n} p_i f_i(x)$$

Ceci peut être mis à profit pour diviser le domaine de la variable aléatoire en plusieurs sous-intervalles, sur lesquels la modélisation est plus simple, par exemple parce qu'il est possible d'avoir une approximation par une variable aléatoire uniforme, triangulaire...

5.3.6.3 Méthode de la transformée inverse

Soit $RU \in]0,1[$ la valeur renvoyée par notre générateur uniforme.

Soit $F(X)$ la fonction de répartition de la distribution que l'on étudie.

$$\Pr\{X \le x\} = \Pr\{F^{-1}(U) \le x\} = \Pr\{U \le F(x)\}$$

Le terme de droite est la fonction de répartition d'une distribution uniforme évaluée à $F(x)$. On va donc chercher à résoudre : $F(x) = RU$, ce qui nous permettra d'obtenir la fonction : $x = F^{-1}(RU)$ qui nous donnera le générateur aléatoire correspondant à notre distribution.

Ainsi, pour une **distribution uniforme** U(a, b) :

$$F(x) = \begin{cases} 0 & x \le a \\ \dfrac{x-a}{b-a} & a < x < b \\ 1 & x \ge b \end{cases}$$

D'où l'on tire le générateur aléatoire correspondant : $\boxed{X = a + (b-a)RU}$

Pour une **loi exponentielle**, on a :

$$F(x) = \begin{cases} 1 - e^{-lx} & (x > 0) \\ 0 \quad \text{sinon} \end{cases}$$

D'où on en déduit le générateur aléatoire par la transformée inverse :

$$X = -\frac{1}{l}\ln(1 - RU)$$

5.3.6.4 Autres méthodes

- Tables de recherche : on utilise des tables de valeurs pré-calculées. Mais ceci n'est surtout utile que pour les distributions discrètes. Attention à la vitesse de l'algorithme de recherche d'une valeur !
- Méthode des alias.
- ...

5.4 La validation des modèles et simulations

5.4.1 Introduction

Nous avons vu que des décisions, graves par l'engagement financier ou humain, peuvent être prises en se basant sur le résultats de simulations.

Mais dans quelle mesure peut-on avoir confiance dans les sorties du simulation ? Qu'est-ce qui nous dit qu'un modèle représente bien la réalité ?

Les techniques visant à s'assurer de la qualité d'une simulation sont regroupées sur le terme VV&A : Vérification, Validation et Accréditation[67].

5.4.2 Définitions

5.4.2.1 Validation

Processus visant à s'assurer qu'un modèle ou une simulation représente le monde réel d'une façon suffisamment précise pour remplir les besoins d'une utilisation donnée.

Il est très important de noter que la validation n'est valable que pour **un domaine d'emploi donné** et doit être remise en question pour toute nouvelle utilisation sortant de ce domaine.

5.4.2.2 Vérification

Processus visant à s'assurer que l'implémentation d'un modèle ou d'une simulation correspond bien à la spécification qui en a été faite.

La vérification se fait par examen du code et par des procédés basés sur les techniques de génie logiciel. De préférence, la vérification est un processus continu tout au long du développement, et non une opération ponctuelle finale.

Il n'entre pas dans les objectifs de ce cours de détailler les techniques de génie logiciel, néanmoins la section 8.1 rappelle quelques notions de bases.

5.4.2.3 Accréditation

Décision d'utilisation d'un modèle ou d'une simulation pour un usage donné.

[67] VV&A est le terme officiel du DoD. J'ai également vu V&V et VV&T (Testing).

5.4.2.4 Certification

Approbation délivrée par le producteur ou l'utilisateur des données après V&V (équivalent de l'accréditation pour les données).

5.4.2.5 Fidélité

Précision d'un modèle par rapport au monde réel

5.4.3 Enjeux du VV&A

On l'a compris, le VV&A est une démarche qualité appliqué à la simulation.

Pour le **client**, il s'agit tout bonnement de s'assurer qu'il disposera bien, à terme, d'un produit conforme à ses besoins.

Pour l'**utilisateur final**, il s'agit de s'assurer que les décisions prises à l'issue de simulations ne sont pas biaisées.

Enfin, pour le **développeur**, le VV&A facilite la réutilisation du code par autrui, en indiquant dans quelle mesure on peut avoir confiance dans un modèle réutilisé dans tel ou tel contexte.

Malheureusement, le VV&A est encore mal maîtrisé. Généralement, les développeurs de simulation intègrent bien un processus correspondant dans leur cycle de développement, mais il n'y a pas de cohérence à grande échelle : les critères de validation de l'un ne sont pas reconnus chez l'autre. En outre, le vide méthodologique n'arrange rien, et chacun a ses propres techniques. Il va sans dire qu'un tel manque d'organisation du VV&A est fortement préjudiciable à la fois à la confiance que l'on placera dans les simulations, mais aussi à la capacité des modèles à être réutilisés.

C'est pour ces raisons que le DoD américain a publié, en 1996, une directive établissant une organisation et une politique VV&A à l'échelle nationale (voir [DOD96]).

5.4.4 Techniques de validation de modèles

Nous avions vu dès les premiers chapitres de ce cours que la simulation était dépendante des essais réels ou sur maquette, afin d'obtenir les données nécessaire à la modélisation des systèmes étudiés. À nouveau, la simulation va faire appel à eux, afin de valider les résultats de la simulation.

On va ainsi comparer une situation réelle avec sa version simulée et mesurer les écarts pour déterminer la fidélité du modèle numérique.

Les sources de données de validation sont :

- Essais de matériels réels (tirs…)
- Essais de maquettes (soufflerie…)
- Simulations hybrides (matériel dans la boucle)
- Simulations de plus grande précision (validées)
- Simulations similaires (mais autre algorithme)
- Phénomènes ou systèmes similaires
- ..

De nos jours, les essais réels étant de plus en plus coûteux, il importe d'organiser le plus intelligemment possible les campagnes de validation. Au CTSN[68], où l'on étudie et qualifie les systèmes de missile mer-air des navires de la Marine Nationale, on établit un rebouclage des résultats des essais sur les simulations (pour les valider et les améliorer), mais aussi des sorties des simulations sur les essais (afin d'optimiser les configurations des essais, et ainsi obtenir le maximum de résultats avec le minimum d'essais).

Dans tous les cas, **le processus de validation (et de vérification) doit être continu**. C'est en effet une grave erreur que d'attendre que le code soit achevé pour commencer la validation : c'est en fait une opération qui doit se dérouler et être prise en compte[69] tout au long du cycle de développement.

Les principales causes d'erreurs dans les simulations sont :

- Génération de nombres aléatoires : le générateur choisi (voir section 5.3) comporte un biais.
- Erreurs de spécification des paramètres ou des modèles : le besoin a été mal recueilli ou exprimé, ou le modèle conceptuel est erroné ou a été mal interprété.
- Erreurs de programmation : erreurs d'implémentation. Par exemple, faute de frappe (un '+' au lieu d'un '*' dans une formule).
- Durée de la simulation : accumulation d'erreurs d'arrondis lors de calcul dans une simulation de longue durée. Ceci peut avoir des effets spectaculaires, et difficiles à diagnostiquer, puisque, a priori, les formules utilisées sont exactes. Dans les simulations distribuées temps réel, on assistait parfois à des problèmes de dérive des horloges des ordinateurs hôtes, désynchronisant ainsi les simulateurs couplés.
- Sensibilité à des paramètres clés : certains systèmes d'équations sont notoirement instables et ont une grande sensibilité aux variations, même infimes, d'un paramètre (« effet papillon »[70]). Il est de bon ton, lors de la validation d'un modèle, de faire varier légèrement les paramètres afin de s'assurer qu'on ne sera pas victime d'un tel effet.
- Erreurs dans la collecte des données de simulation : les données issues des essais réels sont erronées (erreur de mesure ou de transmission, ou conditions expérimentales inadaptées).
- Erreurs dans les processus d'optimisation : en améliorant un modèle afin d'en améliorer les performances ou la précision, on remet en question sa validité.
- Erreur de domaine : on a utilisé un modèle en-dehors de son domaine de validité (par exemple application à un domaine supersonique d'un modèle numérique d'écoulement aérodynamique prévu pour des vitesses subsoniques), sans effectuer de nouvelle validation. C'est une opération certes tentante, mais risquée !
- . . .

[68] Centre Technique des Systèmes Navals, à Toulon.

[69] Cette prise en compte peut se manifester par des validations intermédiaires de composants, mais aussi par l'instrumentation du code dans le but d'effectuer cette validation.

[70] Cette expression vient de la théorie suivant laquelle, en raison de la nature chaotique du comportement de l'atmosphère terrestre, le battement des ailes d'un papillon à Tokyo peut provoquer des ouragans dans les Antilles.

5.4.5 Le VV&A au DoD

Dans la mouvance HLA, le DoD, conscient de la menace que faisait poser le manque de cohérence des processus VV&A sur la réutilisabilité du code des simulations, en vint assez rapidement à mener des réflexions sur un processus général.

Ces réflexions sont concrétisées par la directive 5000.61 « DoD Modeling and Simulation (M&S) Verification, Validation and Accreditation » (voir [DoD96][71]). Ce document décrit la politique VV&A du DoD, les responsabilités de chacun et la marche à suivre pour mener à bien les tâches correspondantes.

Malheureusement, la méthodologie mise au point (voir Figure 81) n'est pas totalement satisfaisant, car il est orienté « processus » (et non produit), et il est possible d'avoir une VV&A valide uniquement par construction, en respectant simplement la directive. En outre, on peut voir sur le schéma que l'on a un ordre implicite vérification → validation → accréditation, alors qu'il faudrait étudier l'accréditation dès le départ, car c'est elle qui fixe l'objectif final à atteindre. Néanmoins, la directive du DoD n'est est pas moins remarquable car il s'agit de la seule démarche VV&A qui ait été faite à un niveau national pour le moment.

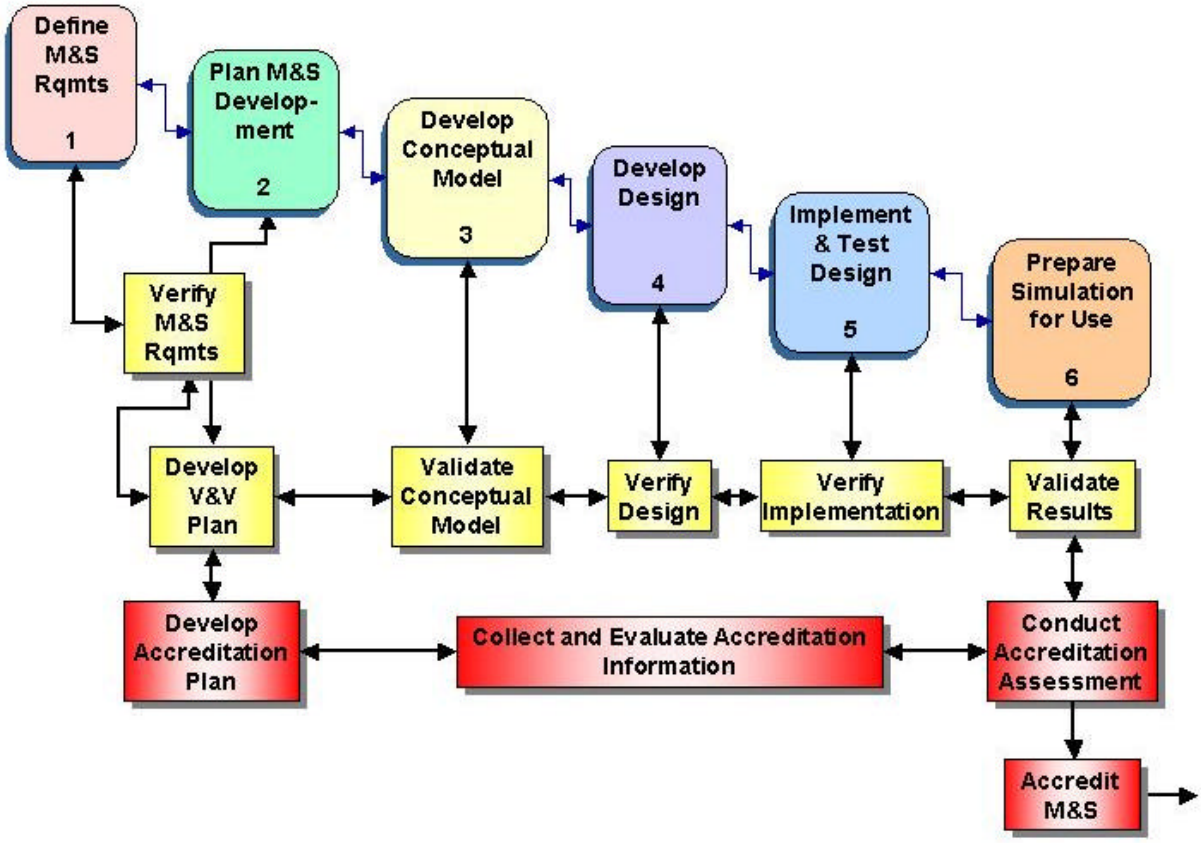


Figure 81: Processus VV&A du DoD (version 2000)

[71] On trouvera sur le site web du DMSO un guide méthodologique des plus intéressants.

5.5 Exemples de modélisation

5.5.1 Problèmes des trois corps

Pendant trois siècles, les plus grands mathématiciens (Euler, Laplace…) ont planché sur le problème des trois corps.

En mécanique céleste, depuis Newton, on savait déterminer la trajectoire d'un objet soumis à un champ de gravité, par exemple une planète orbitant autour d'une étoile, mais très vite on s'est aperçu que, très souvent, la solution trouvée s'écartait de la réalité, car les corps du système solaire subissent des influences multiples.

Ainsi, les calculs des mouvements de la Lune doivent prendre en compte non seulement la gravité terrestre mais aussi celle du Soleil, ce qui en fait un problème non plus à deux mais à trois corps, voire à N corps si on veut tenir compte des influences des autres planètes.

Soient m_1, m_2, …, m_N les masses des N corps, (x_i, y_i, z_i) les coordonnées de chaque corps dans un repère cartésien et r_{ij} la distance entre le $i^{\text{ème}}$ et le $j^{\text{ème}}$ corps.

Pour chaque coordonnée, les composantes sur l'axe x des forces s'exerçant sur m_1 sont :

$$km_1m_2\frac{x_1-x_2}{(r_{12})^3},\ km_1m_3\frac{x_1-x_3}{(r_{13})^3},\ \ldots,\ km_1m_N\frac{x_1-x_N}{(r_{1N})^3}$$

On procède de même pour les axes y et z.

D'après le théorème de la résultante dynamique appliqué au $i^{\text{ème}}$ corps suivant l'axe des x :

$$m_i\frac{d^2x}{dt^2}=-km_i\sum_{\substack{i,j=1\ldots N\\ i\neq j}}\left(m_j\frac{x_i-x_j}{(r_{ij})^3}\right)$$

On écrit cette équation différentielle de la même façon pour les composantes en y et z.

On obtient alors pour chaque corps un système de $3n$ équations différentielles du 2nd ordre, qui peut être transformé en un système de $6n$ équations différentielles du premier ordre. Pour 3 corps, on a donc 18 équations différentielles du premier ordre.

En utilisant les équations du mouvement du centre de masse (résultante cinétique), la conservation des moments angulaires et la conservation de l'énergie, on obtient respectivement 6, 3 et 1 intégrales indépendantes du système, soit 10 en tout, ce qui est insuffisant pour résoudre analytiquement les 18 équations différentielles (résultat démontré par Poincaré à la fin du XIX$^{\text{ème}}$ siècle).

Faute d'une solution analytique, il faut utiliser des calculs approchés, et la modélisation sur ordinateur est encore aujourd'hui la seule méthode permettant de déterminer l'évolution d'un ensemble de N corps célestes (planètes, étoiles, satellites, sondes…) qui interagissent entre eux.

En pratique, la résolution du problème à N corps sur un ordinateur utilise des approximations (par exemple des séries trigonométriques pour le mouvement perturbé de la Lune) ou des méthodes numériques de calcul d'intégrales (par exemple Range-Kutta…).

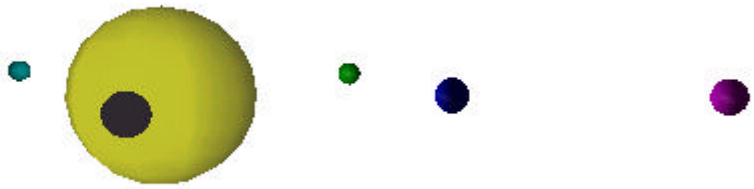


Figure 82 : Simulation informatique du problème à N-corps

Les simulations à *N* corps sont utilisées quotidiennement par les astronomes et les agences spatiales. Elles sont capitales pour la bonne compréhension de la mécanique céleste et pour les voyages dans l'espace. Ainsi, c'est en utilisant une telle modélisation que sont planifiées de façon optimales les trajectoires des sondes spatiales, afin de profiter au mieux des attractions gravitationnelles pour économiser du carburant.

5.5.2 Modélisation des combats dans un jeu de guerre

5.5.2.1 Description générale d'un jeu de guerre

Camps : bleu (les amis), rouge (les ennemis), plus des « neutres » et/ou des « suspects ».

Le découpage traditionnel du terrain dans un wargame utilise des cases hexagonales[72] (voir Figure 83). Les hexagones permettent en effet de partager un plan en secteurs identiques (ce qui n'est pas possible avec des octogones par exemple) et définissent 6 directions de déplacement (contre 4 avec des carrés). La taille des hexagones varie en fonction du niveau de la simulation : de quelques dizaines ou centaines de mètres pour une simulation tactique, à quelques dizaines de kilomètres pour une simulation de niveau opératif.

On considère qu'une unité occupe un hexagone, éventuellement conjointement avec d'autres unités. Chaque hexagone définit un compartiment de terrain homogène (nature, altitude, pente…). Des obstacles (coupures…) peuvent suivre les contours des hexagones. Leur franchissement, quand il est possible, peut pénaliser la vitesse de déplacement des unités.

Chaque unité a un potentiel de déplacement par pas de temps de la simulation. Chaque mouvement d'un hexagone à l'autre consomme une partie de ce potentiel, en fonction de la nature du terrain (route, forêt, montagne, marécage…), des besoins de franchissement (franchir une rivière peut prendre plusieurs minutes s'il n'y a pas de gué ou de pont), de la météo (le brouillard ou la nuit ralentit les déplacements), de la situation tactique (dans une zone sûre on ira plus vite que dans une zone de combat), etc.

[72] Les wargames informatiques peuvent parfois s'affranchir de cette contrainte en gérant les coordonnées géographiques d'une façon continue. Toutefois, les hexagones restent toujours populaires. Ainsi, le terrain du jeu de guerre JTLS (niveau opératif), utilisé dans de nombreux pays de l'OTAN, est divisé en cellules hexagonales.

Pour plus d'informations sur les wargames, voir [PERL90] et [DUNN92], deux ouvrages de référence, ainsi que [LIAR97], thèse française sur la question.

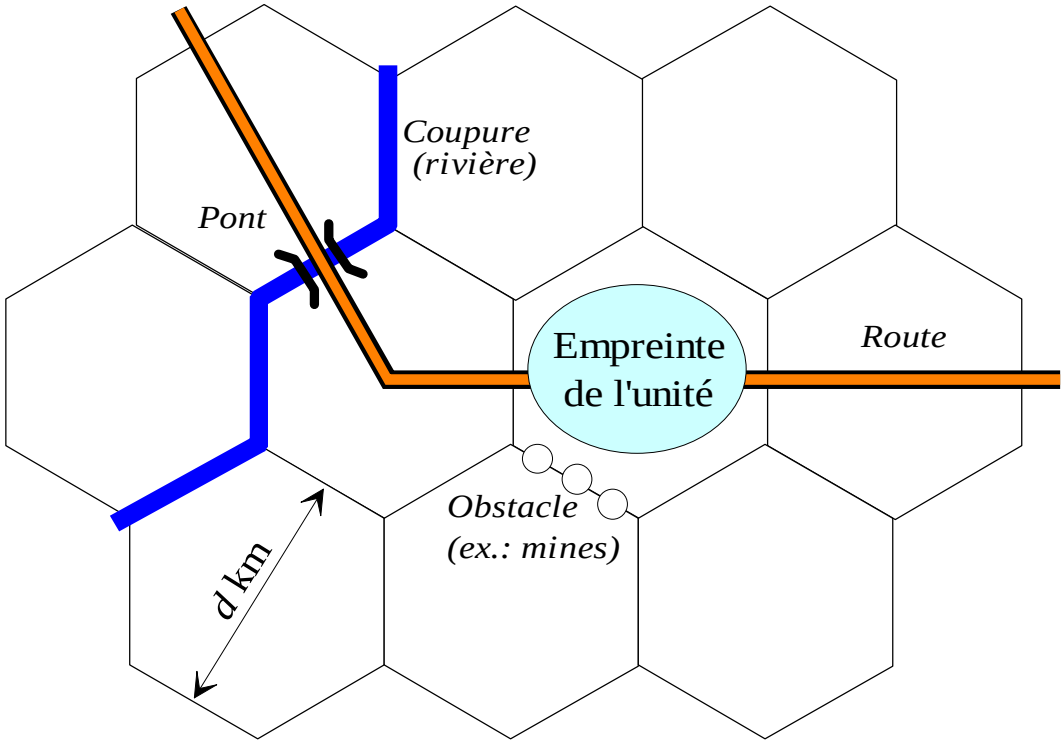


Figure 83 : découpage hexagonal classique du terrain

5.5.2.2 Méthode des jeux sur plateau

Dans la plupart des jeux de guerre sur plateau, on utilise un générateur aléatoire (modélisé par un jet de dé) pour déterminer le résultat des combats.

On commence par calculer le rapport des forces. Chaque unité a un potentiel de combat qui lui est propre, et pouvant être diminuée par l'absence de ravitaillement, la fatigue, les pertes des précédents combats (attritions), ou améliorée par la proximité d'un QG ou d'un grand général. Admettons par exemple qu'une unité d'infanterie (potentiel 4) est attaquée par deux unités de cavalerie (potentiel 6 chacune). On est alors à 12 contre 4, soit 3 contre 1 en faveur de l'attaquant.

On jette le dé. On obtient 4, ce qui est plutôt favorable à l'attaquant. Mais il faut aussi tenir compte des effets du terrain sur le résultats des combats. Le défenseur est sur une colline et l'attaquant en plaine. Un tableau des terrains nous indique qu'il faut alors soustraire 1 au jet de dé. On obtient 3 et on consulte le tableau de résolution des combats (voir Figure 84).

TABLEAU DES RÉSULTATS DES COMBATS								
Rapport des forces	DÉ							
	-1	0	1	2	3	4	5	6
1 contre 3	AE	AE	AE	AE	AD	AD	AD	R
1 contre 2	AE	AE	AE	AD	AD	AD,DD	R	DR
1 contre 1	AD	AD	AD,DR	AD,DD	AD,DRD	DR	DD	DRD
2 contre 1	R	R	AD,DR	AD,DD	DR	DD	DRD	DRD
3 contre 1	AD,DD	AD,DRD	DR	DD	DRD	DRD	E	DE
4 contre 1	DD	DRD	DRD	DRD	E	E	DE	DE
5 contre 1	DRD	DRD	DRD	E	E	DE	DE	DE
6 contre 1	DRD	E	E	DE	DE	DE	DE	DE
RAPPORTS HORS TABLEAU : 1 contre plus de 3 donne toujours AE, plus de 6 contre 1 donne toujours DE								

Figure 84: Tableau des résultats des combats

Le résultat définitif des combats est donc DRD, qui signifie que l'unité défenseur soit doit reculer (DR) et est désorganisée (D). L'une des unités attaquante peut alors occuper la place libérée et prendre la colline. Les autres résultats possibles sont R (rien), AE (attaquant éliminé), DE (défenseur éliminé), AR (attaquant recule), E (échange : le défenseur est éliminé, mais fait chez l'attaquant autant de dégâts que son potentiel de combat, en raison d'une défense acharnée), etc.

Évidemment, ceci est à répéter pour des centaines de combats durant une partie, en tenant à jour la liste des unités désorganisées, non ravitaillées, etc. Les jeux de guerre « manuel » sont donc très lourds, et parfois franchement injouables.

Très rapidement après l'avènement des ordinateurs, les militaires (puis à partir des années 1970 les joueurs amateurs) ont réfléchi sur la façon d'utiliser un calculateur pour gérer ces paramètres, ainsi que bien d'autres, pour avoir des simulations de combat réalistes et complexes, tout en restant humainement jouables.

Deux voies principales sont utilisées : l'une probabiliste (Monte-Carlo), l'autre déterministe (Lanchester).

5.5.2.3 Méthode de Lanchester

La modélisation analytique des combats a vraiment commencé avec les travaux du mathématicien anglais Frederick Williams Lanchester au début du XX^{ème} siècle, qu'il publia en 1916 dans un livre consacré aux forces aériennes.

Lanchester établit une théorie dynamique du combat. Par des systèmes d'équations différentielles il parvint à rendre compte de l'évolution des forces de deux parties combattant l'un contre l'autre.

Les hypothèses de départ sont :

- Les éléments d'un même parti disposent du même armement
- Chaque élément peut prendre à parti n'importe quel élément adverse
- Le coefficient d'attrition (efficacité d'un camp sur l'autre) est constant et connu

À partir de là, les équations de Lanchester se déclinent en deux lois, linéaire (plus simples) et quadratique.

Pour la **loi linéaire**, on considère que les pertes sont constantes par unité de temps, qu'un élément ne peut combattre que contre un seul adversaire à la fois, et que chaque combat est une succession d'engagements individuels dans l'espace et le temps.

On note X et Y les effectifs des deux camps (initialement X_0 et Y_0). On désigne par a et b les taux de pertes de chaque camps respectifs.

On a : $\frac{dX}{dt} = -a$ et : $\frac{dX}{dt} = -b$.

Les solutions de ce système linéaire sont : $X = X_0 - at$ et $Y = Y_0 - bt$.

La loi linéaire de Lanchester s'écrit ainsi : $\boxed{X_0 - X = \frac{a}{b}(Y - Y_0)}$.

Pour la **loi quadratique**, on considère que les pertes par unité de temps sont proportionnelles aux effectifs de l'adversaire.

On a : $\frac{dX}{dt} = -aY$ et : $\frac{dY}{dt} = -bX$.

Les solutions de ce système sont de la forme :

$X = X_0 \,\mathrm{ch}(ct) - d\, Y_0 \,\mathrm{sh}(ct)$

$Y = Y_0 \,\mathrm{ch}(ct) - d\, X_0 \,\mathrm{sh}(ct)$

Avec $c^2 = ab$ et $d^2 = a/b$.

La loi quadratique de Lanchester s'écrit alors : $\boxed{\left(X_0^{\,2} - X^2\right) = \frac{a}{b}(Y^2 - Y_0^{\,2})}$.

Ces modèles, au demeurant grossiers, ont été enrichis au fil du temps pour tenir compte de certains effets d'armes (artillerie...), de certaines tactiques (guérilla...), des pannes, des renforts, etc. Ainsi, pour les armes à effet de zone (artillerie), on utilisera l'équation :

$$\frac{dX}{dt} = -cXY \quad \text{et :} \quad \frac{dY}{dt} = -dXY$$

c et d sont les coefficients d'attrition par les armes à effet de zone. Ces coefficients seront déterminées par les caractéristiques des forces et les conditions du combat. Ainsi, l'infanterie à découvert aura un coefficient d'attrition élevé, alors qu'une unité de blindés à couvert dans une forêt subira une faible attrition).

De la même façon, on peut admettre que l'efficacité, en terme d'attrition, des armes « intelligentes » (bombes guides, missiles de croisière), est proportionnelle au carré de la force de l'adversaire :

$$\frac{dX}{dt} = -gY^2 \quad \text{et :} \quad \frac{dY}{dt} = -hX^2 .$$

(avec g et h coefficients d'attrition par les armes intelligentes).

En tenant compte des pertes dues au feu direct *aimed fire*) et indirect (armes à effet de zone ou *area fire*) renforts et des désertions, on obtient un modèle de calcul comme représenté ci-dessous :

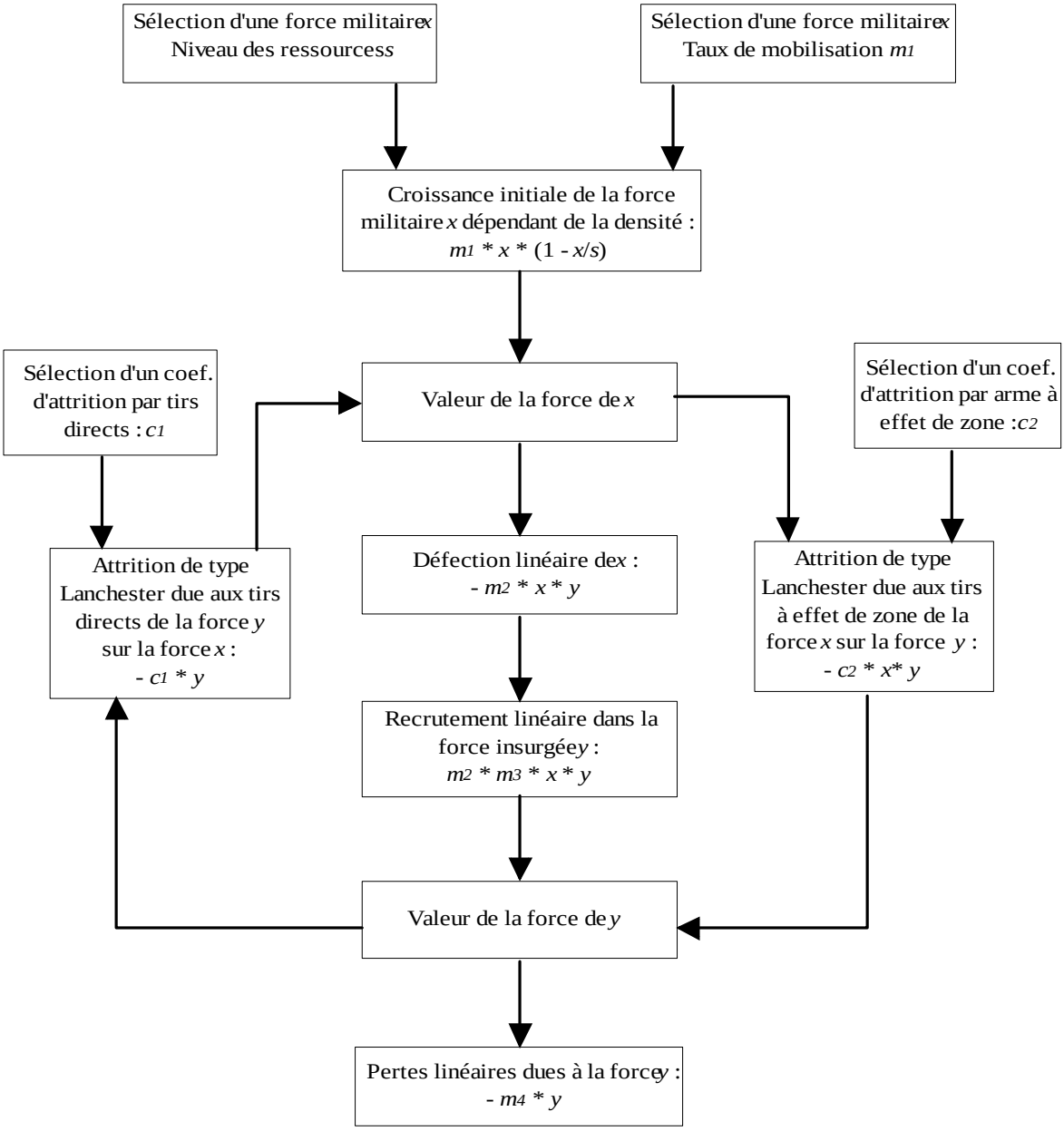


Figure 85: Exemple de modèle de combat de type Lanchester

(d'après [DOCK93])

Les équations de Lanchester sont valables pour un nombre important de combattants. Elles représentent une valeur plutôt moyenne des résultats des combats, et l'ont peut éventuellement faire intervenir un terme d'origine stochastique.

Elles ont été vérifiées lors d'affrontements réels (1ère et 2nde Guerre Mondiale, guerre de Corée...) avec d'assez bons résultats. En revanche, elles sont inefficaces pour bien des conflits modernes, tels que le maintien de la paix, ou pour le combat urbain, où les unités sont très petites (section, groupe, voire trinôme) et où l'individu prend une plus grande importance (Liban, Palestine, Kosovo...).

Un ouvrage très complet sur la théorie mathématique des combats est [DOCK93].

5.5.2.4 Méthode de Monte-Carlo

La méthode de Monte-Carlo est basée sur une approche probabiliste. Elle a été développée dans les années 40, pour le calcul de la diffusion (aléatoire par nature) des particules lors des réactions nucléaires.

Elle peut être utilisée notamment dans deux cas :

- Lorsqu'il n'existe pas de solution analytique pour la modélisation.
- Lorsque les phénomènes modélisés sont notoirement aléatoires et que l'on veut rendre compte de cette particularité.

Le résultat d'un tir, les dégâts faits sur un char par l'explosion proche d'un obus ou la détection d'une unité ennemie à travers une haie d'arbres sont des exemples de phénomènes aléatoires intervenant dans la résolution des combats.

Monte-Carlo permet donc de prendre en compte un certain nombre de variables aléatoires, suivant une loi de probabilité identifiée. En revanche, le résultat n'est plus déterministe : N exécutions du même combat dans les mêmes conditions initiales donneront N résultats différents. Il faut donc exécuter plusieurs fois la simulation pour obtenir une moyenne exploitable.

Un avantage de cette approche est aussi de restituer les cas extrêmes : une force largement supérieure en nombre peut très bien jouer de malchance et être battue. Ceci est très important dans les simulations destinées à la planification d'opérations : on teste par la simulation différents modes d'action amis face à différentes réactions possibles de l'ennemi, afin de déterminer quelle serait la meilleure tactique à adopter. Si une tactique offre 70% de chances de victoire totale, mais que dans une ou deux exécutions de la simulation, elle se solde par un terrible désastre, il pourrait être plus intéressant de choisir un autre mode d'action au résultat globalement moins bon, mais qui de risque pas d'entraîner de catastrophe... à moins que ces cas extrêmes ne soient un indicateur d'instabilité du modèle, ou même de son caractère erroné.

Parmi les utilisations les plus courantes de Monte-Carlo, citons :

- Les études technico-opérationnelles
- Les analyses boursières et économiques
- La prospection pétrolière
- La physique nucléaire
- L'astrophysique
- La simulation de trafic de véhicules
- ...

5.5.2.5 Autres méthodes

Figure 86: John H. Conway, univ. de Cambridge

On trouve parfois des **automates cellulaires** pour la modélisation des combats avec contacts entre les belligérants.

Les automates cellulaires servent à étudier l'évolution dans le temps d'organisations biologiques complexes (survie, reproduction...). Ils sont issus de théories mathématiques développées d'abord par Alan Turing (travaux sur l'automatisation des calculs par la programmation d'ordinateurs) dans les années 30, puis par des scientifiques tels que Von Neumann (1966), Codd (1968) ou Conway (1971).

L'exemple le plus connu d'automate cellulaire est le « jeu de la vie » de Conway. Ce jeu représente l'évolution d'une population sur un terrain divisé en cases (« cellules ») par une grille. Chaque case est soit vide, soit occupé par un individu de la population. À chaque pas de temps, on calcule l'évolution de chacun de ces individus en fonctions d'un ensemble de règle prédéterminées, comme par exemple :

- Si une cellule occupée est entourée de deux ou trois cellules occupées, il y a survie. S'il y a plus de trois cellules adjacentes occupées, la nourriture manque et l'individu meurt de faim.

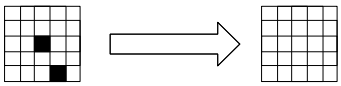


Figure 87: Jeu de la vie (mort d'une cellule)

- Si une cellule vide est entourée de trois cases occupées, alors il y a reproduction et un individu apparaît dans la cellule.

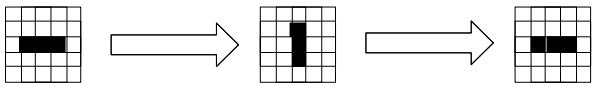


Figure 88: Jeu de la vie (oscillateur)

- S'il n'y a que zéro ou une cellule adjacente occupée, l'individu est isolé et meurt.

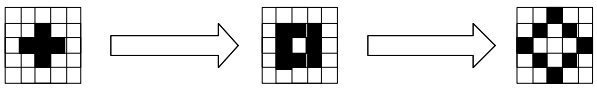


Figure 89: Jeu de la vie (décès par surpopulation)

Le jeu de la vie consiste à faire varier les règles et la répartition initiale de la population, afin de générer des motifs particuliers, stables, cycliques (tel que l'oscillateur de la Figure 88, ou celui de la Figure 89 qui après quelques cycles évolue vers quatre oscillateurs)...

Le jeu de la vie peut modéliser de façon assez réaliste des processus biologiques tels que le développement d'une colonie de bactéries sur un substrat nutritif, les déplacements urbains (la simulation TRANSIMS, que nous avons décrite page 75, est basée sur ce principe)... ou les combats.

On peut adapter ce principe à la résolution des combats, avec des règles telles que :

- Si une unité est entourée d'au moins trois unités amies, elle avance/
- Si une unité est adjacente à zéro ou une unité amie seulement, elle effectue une retraite.
- Si l'unité est adjacente à une seule unité ennemie, elle ouvre le feu sur elle.
- ...

6. SIMULATION PAR EVENEMENTS DISCRETS

6.1 Définitions

Un **système à événements discrets** est un système dont l'état change de façon discontinue en un certain nombre d'instants pouvant être ou apparaître aléatoires.

La simulation par événements discrets (*discrete event simulation*, habituellement abrégée DES ou DEVS) traite de la modélisation de tels systèmes.

L'exemple typique est celui des clients arrivant à un guichet de banque : le « guichet » est un système à événements discrets. L'entrée d'un objet « client » constitue un événement qui va modifier l'état du système.

Dans une simulation à événements discrets, on rencontre fréquemment les éléments suivants :

- Entités (voir §5.2.1.3).
- Ressources : éléments du système fournissant un service.
- Éléments de contrôles : permettent de modifier la réponse du système à un événement, par l'intermédiaire de switches, compteurs, règles logiques ou arithmétiques... Par exemple, une contrainte horaire (fermeture des guichets pendant la pause de midi).
- Opérations : ce sont les manipulations effectuées par ou sur les entités évoluant au sein du système. Par exemple le traitement d'un client au guichet ou le départ de ce client.

6.2 Principes de base

6.2.1 Répliques

Une **réplique** (*replication* ou *run* en anglais) est une exécution d'une simulation.

Typiquement, on effectue des séries de répliques pour des simulations comprenant des modèles stochastiques. Ces répliques vont produire des résultats qui seront analysées ensuite hors exécution (*off-line*) par des techniques statistiques telles que Monte-Carlo.

6.2.2 Initialisation

La phase d'initialisation peut parfois nécessiter une part important du code de la simulation.

On y effectuera, entre autres :

- La lecture des paramètres de la simulation et du scénario.
- L'initialisation des variables (compteur de temps, les variables statistiques...).
- La création des entités de la simulation.
- La détermination des événements pouvant être prédéterminés (par exemple les événements fixe prévus par le scénario ou les événements aléatoires indépendants). Par exemple, il est possible de déterminer à l'avance, par des tirages aléatoires, la date d'arrivée des clients d'un guichet, sachant que l'écart de temps séparant deux arrivées suit une loi exponentielle. On postera à l'avance tous ces événements dans la file d'événements (voir plus loin).

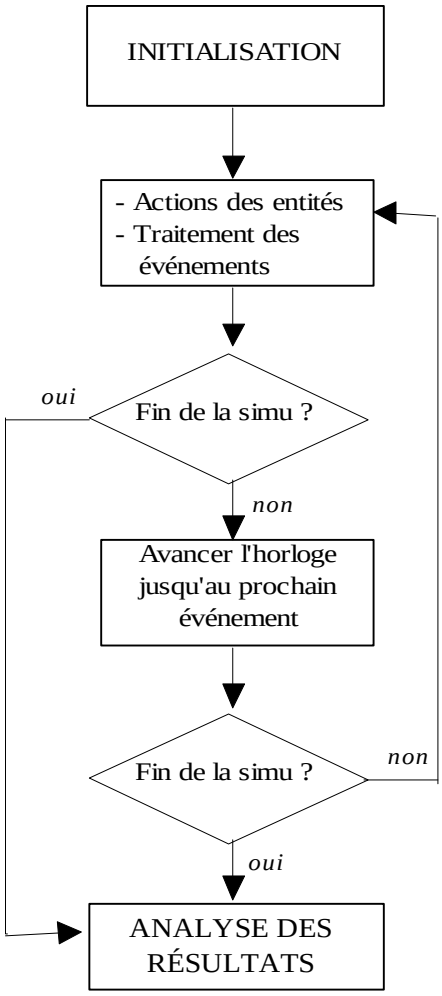


Figure 90 : Déroulement typique d'une réplique

6.2.3 Gestion des entités

On fait évoluer les entités en fonction de leur modèle comportemental, du scénario, de leur environnement, des événements qui surviennent, etc.

Chacune des entités est une instance d'un ensemble de variables (variables d'état, variables statistiques...), généralement regroupées au sein d'une structure, dont l'implémentation typique est une instance de classe C++.

Le moteur de simulation maintien une liste des entités et de leur état. Cette liste peut être implémentée sous diverses formes (listes chaînées, tableaux, arbres...). Voir, par exemple, [KNUT68] et [SEDG91] pour une présentation détaillée des principales structures de données utilisables.

Chaque entité suit un cycle de vie (création, attente, activité, destruction…), dont la forme est fonction du moteur de simulation utilisé. Voici un exemple typique, celui d'un acteur dans une simulation utilisant la version 5.0 de la structure d'accueil ESCADRE :

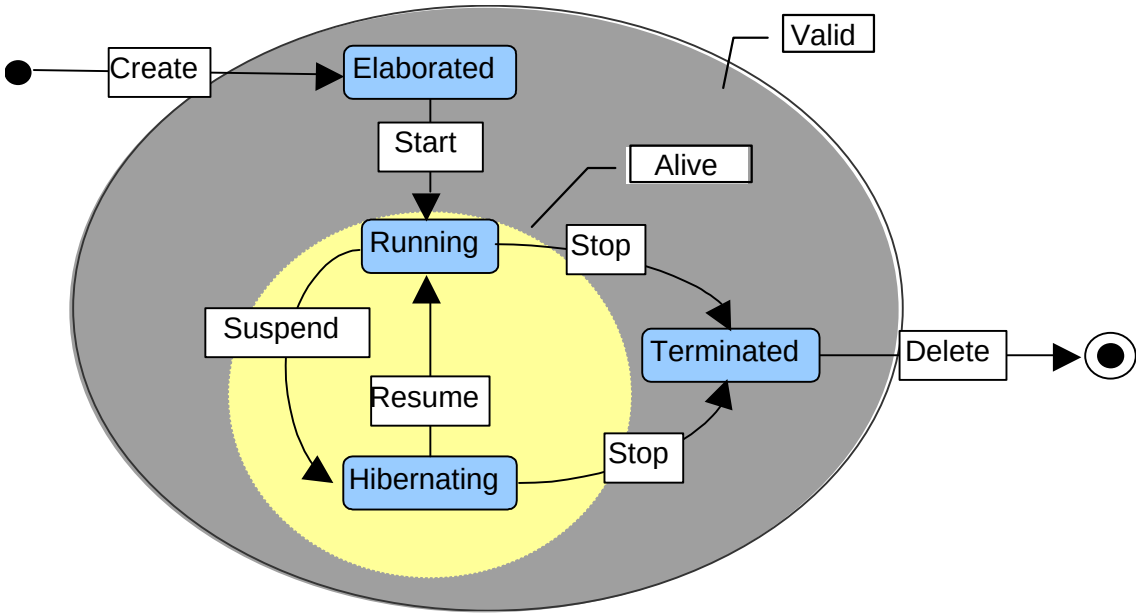


Figure 91: Cycle de vie d'un acteur (ESCADRE 5.0)

6.2.4 Gestion des événements

6.2.4.1 Liste d'événements

Comme nous l'avons déjà vu, dans une simulation à événements discrets, l'avance du temps est généralement conditionné par les événements qui se produisent.

Il est donc nécessaire, pour que le moteur de simulation puisse calculer le nouveau temps, que soit connu la date du prochain événement (qui à un instant t donné se trouve dans le futur à $t + \Delta t$). Ceci est possible en simulation à événements discrets par la connaissance préalable de l'enchaînement des événements futurs.

En effet, comme on l'a vu précédemment à propos de la phase d'initialisation, la date des événements futurs, même aléatoires, peut souvent être prédéterminé, soit lors de l'initialisation, soit à une date antérieure à l'événement.

C'est pourquoi la simulation n'a pas besoin d'évoluer suivant un temps continu ou un pas de temps régulier : elle peut se contenter de sauter d'un événement à l'autre, à condition toutefois de traiter chaque événements dans l'ordre chronologique (problème de la causalité sur lequel nous reviendrons).

Afin de pouvoir à tout moment déterminer la date et la nature du prochain événement, le moteur de simulation maintient une liste ordonnée chronologiquement de tous les événements générés au cours de la simulation qui n'ont pas encore été traités. Par exemple (temps simulé en heures décimales et nature de l'événement) :

```
0.000 : ouverture guichets
0.000 : arrivée client
0.000 : arrivée client
0.320 : arrivée client
0.563 : arrivée client
...
3.500 : début pause déjeuner
```

```
4.250 : fin pause déjeuner
...
6.155 : début panne ordinateur
6.670 : fin panne ordinateur
...
```

On remarquera dans notre exemple que deux clients attendaient déjà à l'ouverture des guichets. Le choix a été fait ici de les représenter comme deux événements « arrivée client » à l'instant 0.000.

Tous ces événements peuvent être prédéterminés lors de l'initialisation. Néanmoins, d'autres événements peuvent être produits en cours d'exécution. Ainsi, un guichet à l'état fermé peut décider de passer à l'état ouvert au bout d'un certain temps de latence quand tous les guichets ouverts ont au moins 5 clients qui attendent.

Il faut donc pouvoir stocker tous ces événements dans une structure de données ordonnée et dynamique (création/insertion et destruction fréquentes d'éléments), dont le premier élément (le prochain événement) doit pouvoir être aisément accessible et dans laquelle les opérations d'insertion doivent être simples.

La structure de donnée classiquement utilisée en simulation pour cela est la **file d'attente**, ou **FIFO**.

6.2.4.2 Files d'attente

Les files d'attentes (*queues*) sont des structures de données informatiques de type liste, dans lequel l'ajout d'une donnée se fait à une extrémité de la liste, et la lecture à partir de l'autre extrémité. Ainsi, le premier élément stocké dans la file d'attente est le premier lu, d'où l'appellation de FIFO (*First In, First Out*).

La Figure 92 illustre le fonctionnement d'une telle structure :

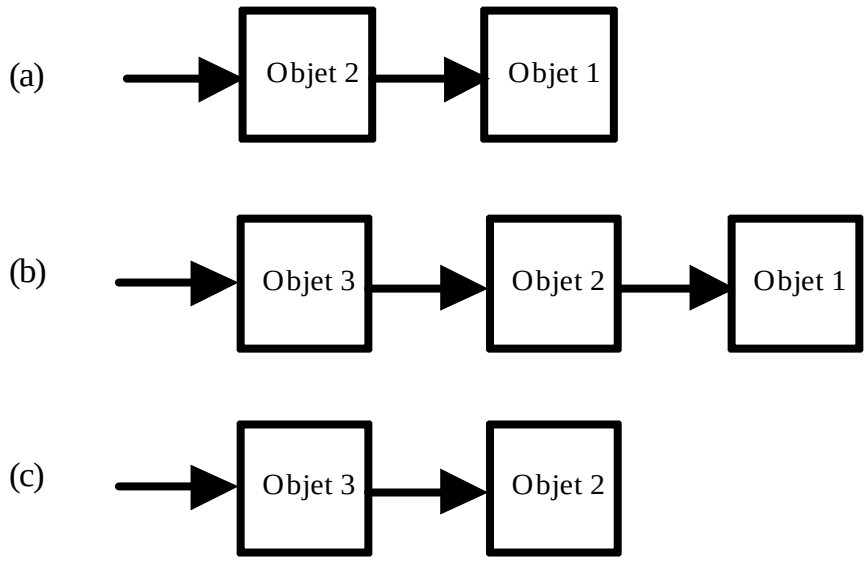


Figure 92 : Fonctionnement d'une file d'attente

(a) On considère une file d'attente comprenant deux objets. L'objet 1 a été le premier à être stocké dans la structure, suivi de l'objet 2. Un lien (i.e. un pointeur) relie l'objet 1 à l'objet 2. Un pointeur spécial indique le dernier élément de la file (un second pourrait aussi indiquer l'objet en tête de la liste, mais cela n'est pas obligatoire).

(b) On veut rajouter un objet 3 dans la FIFO. Il devient alors le dernier élément et on lui associe un pointeur sur l'avant-dernier élément, en l'occurrence l'objet 2.

(c) On souhaite traiter le premier élément de la file. Pour cela, on va simplement retirer l'élément en tête de liste, c'est-à-dire objet 1. Le pointeur de l'objet 2 vers l'objet 1 est supprimé.

Ce type de structure est très utilisé pour gérer des requêtes multiples vers une ressource unique, par exemple des travaux d'impression vers une imprimante. Si celle-ci est occupée, chaque travail est mémorisé dans une file d'attente et le traitement se fera dans l'ordre d'arrivée de ces requêtes.

6.3 Exemples d'outils et de simulations par événements discrets

6.3.1 Produits sur étagère

6.3.1.1 MATLAB

MATLAB est un environnement généraliste de calcul numérique extrêmement populaire actuellement. Commercialisé sur plateforme PC/Windows par la société The Mathworks Inc., il comporte de nombreux outils pour l'acquisition et l'analyse de données (y compris traitement de signal, statistiques…), le calcul numérique (traitement de matrices, polynômes, équations différentielles…), la visualisation et le traitement d'image, le prototypage et le développement d'algorithmes, et, bien sûr, la modélisation et simulation.

Bien que MATLAB ne soit pas un outil entièrement dédié à la simulation, c'est toutefois l'un des plus répandu actuellement pour cet usage[73], notamment grâce à sa convivialité, son ouverture (il est aisé d'ajouter ses propres modules, en langage propriétaire ou en C++, FORTRAN, Ada ou Java) et son extension Simulink, boîte à outils pour la simulation.

Simulink est un outil interactif de modélisation et d'analyse de système dynamique qui, classiquement, repose sur des schémas-blocs (la bibliothèque standard est particulièrement riche dans le domaine du traitement de signal). Il peut gérer le temps de façon discrète ou continue.

Voir : http://www.mathworks.com.

Notons qu'il existe un « clone » de MATLAB sous forme de logiciel libre, SCILAB, développé par l'INRIA.

Voir : http://www-rocq.inria.fr/scilab/.

[73] Début 2001, The Mathworks revendiquait 500 000 utilisateurs dans le monde.

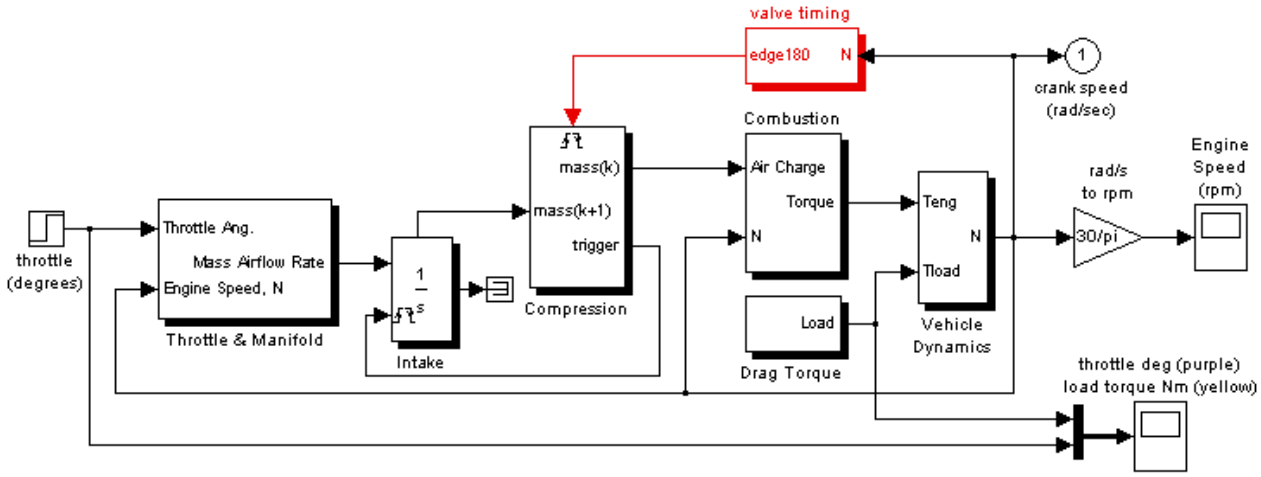


Figure 93: Schéma-blocs Simulink pour la simulation d'un moteur

6.3.1.2 GPSS/H

GPSS (General Purpose Simulation Software), de la société Wolverine Software Corp., permet la modélisation de systèmes par décomposition en blocs représentant chacun une « activité » du système. Ces blocs, qui peuvent être prédéfinis ou conçus par l'utilisateur, sont connectés entre eux par des chemins de transition.

GPSS manipule trois types d'entités : les transactions (entités temporaires), les unités de traitement *(facilities)* et les unités de stockage (*storages*). Les transactions se déplacent d'un bloc à l'autre. Les unités de stockages peuvent contenir plusieurs transactions. Ces entités peuvent être des opérateurs humains, des machines, des camions, des messages, etc.

Le passage d'une transaction d'un bloc est l'autre est effectué en fonction des résultats de conditions logiques et mathématiques.

GPSS est donc très orienté vers la représentation de systèmes basés sur des flux et des files d'attente. Malgré cette limitation, il reste très utilisé, notamment aux USA, fort de ses origines remontant aux années 60.

Voir : http://www.wolverinesoftware.com.

6.3.1.3 ESCADRE

ESCADRE est une structure d'accueil créée par le Centre d'Analyse de Défense de la DGA pour réaliser des simulations d'études technico-opérationnelles.

On trouvera une description détaillée d'ESCADRE dans la section 8.3.

Voir : http://escadre.cad.etca.fr:1815/.

6.3.1.4 DEVS

DEVS est un outil et une méthodologie de développement de simulations à événements discrets. Le §8.2.3.6 en donne une description détaillée.

Voir : http://www.acims.arizona.edu/.

6.3.1.5 OPNET

OPNET est un produit orienté vers la modélisation de réseaux et de protocoles de communication.

Orienté objets, il permet de définir la structure du réseau à l'aide d'un éditeur graphique, avec la possibilité de créer plusieurs niveaux hiérarchiques afin de modéliser des topologies complexes.

Les protocoles et autres processus sont modélisés à l'aide de machines à états finis.

OPNET est populaire dans l'ingénierie des réseaux, et une large bibliothèque de fonctions et de réseaux sont disponibles (HTTP, TCP, IP, ATM, Ethernet, réseaux sans fil, etc.).

OPNET est un produit commercial de la société OPNET Technologies Inc.

Voir : http://www.mil3.com/opnet/home.html.

6.3.1.6 ARENA

ARENA est un produit commercial orienté vers la simulation de flux : on définit à l'aide d'un éditeur graphique le diagramme (« flowchart ») des processus, les ressources et les flux.

Ainsi, au Brésil, la société Ford a utilisé Arena pour modéliser entièrement une unité de production d'automobiles, afin de tester différents agencements des postes de l'usine et détecter les goulets d'étranglement.

Voir : http://www.arenasimulation.com.

6.3.1.7 SIMSCRIPT

SIMSCRIPT est un produit commercialisé par la société CACI Products Co. Issus de travaux de la RAND Corporation dans les années 60, il s'agit d'un langage procédural dédié à la simulation événementielle (notamment de processus).

SIMSCRIPT est disponible sous de nombreux environnements (Windows, Unix…).

Voir : http://www.caciasl.com.

6.3.2 Exemple complet d'une DES simple

6.3.2.1 Énoncé du problème

M. Charles Attand possède un magasin de matériel vidéo. L'un des produits phares de son commerce est le lecteur de DVD Shitsu KRX-2000 (prix : 500 €), dont il vend un exemplaire tous les deux jours environ.

Le stock initial est de 15 lecteurs. On effectue une commande de 12 unités dès que le stock tombe en-dessous de 4. Le délai de livraison est en principe de 7 jours, mais une variation d'environ ± 2 jours est à attendre en pratique.

Cette stratégie est-elle payante ? Quel est le chiffre d'affaire perdu en raison des éventuels invendus ?

6.3.2.2 Analyse du problème

Le **système** simulé est le stock de lecteurs. On s'attachera à étudier ses évolutions.

Les **entités** gérées par le système sont les clients et les lecteurs.

Les **variables d'état** sont : le temps, le nombre de lecteurs en stock.

Les **variables statistiques** sont déterminées par les résultats que l'on souhaite obtenir : nombre de clients insatisfaits, nombre total de clients.

Les **événements** sont : arrivée client, commande, livraison (en fait, dans notre implémentation, nous n'utiliserons pas d'événement « commande »)

Les **paramètres de la simulation** sont : stock initial, prix de chaque lecteur, volume des commandes, seuil de commande, intervalle moyen entre deux clients, délai moyen de livraison et intervalle de variation de ce délai.

On note deux **processus stochastiques** :

- L'arrivée des clients. L'intervalle entre deux clients suit une loi exponentielle, de paramètre $\lambda = 1/E(x) = 0{,}5$.
- Le délai de livraison, qui suit une loi normale que nous approcherons pour simplifier les calculs par une loi triangulaire de paramètres (5, 7, 9).

Il semble naturel d'opter pour une simulation à événements discrets pour résoudre ce problème.

6.3.2.3 Classes pour gérer les objets du système

Nous avons besoin de classes pour gérer des événements datés et une file d'attente ordonnée chronologiquement. Nous utiliserons pour cela des classes C++ définies dans un module `fifo.cpp` et spécifiées comme suit :

```
/*****************************
 *                           *
 *         F I F O . H       *
 *                           *
 *****************************
 *
 */

#ifndef _FIFO_H
#define _FIFO_H
```

```
// Codes d'erreur
#define  OK                    0
#define  EMPTY_QUEUE_ERROR    -1
#define  INVALID_TIME_ERROR   -2
#define  OTHER_ERROR          -9


typedef   double Time;                 // Type temps

#define   TIME_UNKNOWN   -999999.999   // Valeur speciale


/***********************************/

// Événément simple
//
class Event
{
   protected:
      int   id;
      Event   *nextEvent;      // Evénement suivant, ou NULL

   public:
      int   type;             // Type d'évt (défini par l'utilisateur)
      int   parameter;        // Paramètre ( "     "      " )

   Event(int type, int parameter=0);
   Event(Event& evt);              // Constructeur de copie
   void setNextEvent(Event *event);
   Event *getNextEvent(void);
   int getID(void);
};


// Timbre à date
//
class TimeStamp
{
   protected:
      Time   ts;

   public:
      TimeStamp(Time ts=TIME_UNKNOWN);
      Time getTimeStamp(void);
      void setTimeStamp(Time ts);

   // On aurait pu faire plus simple, mais c'est aussi pour l'exemple!
};


// Événement daté
// (TSE *est* un événement *avec* un TS)
//
class TimeStampedEvent : public Event
{
   protected:
      TimeStamp   ts;

   public:
      TimeStampedEvent(int type, Time ts, int parameter=0);
      TimeStampedEvent(TimeStampedEvent &evt);
      Time getTimeStamp(void);
      void setTimeStamp(Time ts);
      TimeStampedEvent *getNextEvent(void);
};
```

```
// File d'événéments TSE
//
class TSEventQueue
{
   private:
      TimeStampedEvent   *first;
      TimeStampedEvent   *last;
      int   length;

   protected:
      bool   TSO;

   public:
      TSEventQueue(bool ordered);
      TimeStampedEvent *postEvent(TimeStampedEvent *evt);
      TimeStampedEvent *getNextEvent(void);
      TimeStampedEvent *readNextEvent(void);
      void flush(void);
      int getLength();
};

#endif      // _FIFO_H
```

FIFO est une librairie de classes pour gérer des files d'attente d'événements. Elle regroupe les classes suivantes :

- `Event` : événement, comportant un type et un paramètre entiers dont la valeur est fixée par l'utilisateur. Par exemple, le type peut être un code indiquant « Arrivée client » et le paramètre peut indiquer s'il s'agit d'un homme ou d'une femme.
- `TimeStampedEvent` : événement avec un timbre à date. Cette date est du type `Time`, définit comme un `double`.
- `TSEventQueue` : file d'attente d'événements datés. Le constructeur prend un paramètre booléen indiquant si la file est ordonnée par ordre chronologique (TSO) ou par ordre d'arrivée des événements (FIFO pure). Une méthode `post` permet de poster un événement. Cette méthode fait une copie de l'événement qui lui est passé en paramètre. Pour la lecture, on notera la différence entre `read()` et `get()` : le premier laisse l'élément dans la file, alors que le second l'en retire. Attention, il est du ressort de l'utilisateur de la méthode `get()` de libérer la mémoire allouée pour cet élément.

On ajoute aussi les routines suivante de génération de nombres aléatoires selon les lois uniforme sur [0,1], triangulaire et exponentielle :

```
/*****************************
 *                           *
 *     R A N D O M . H       *
 *                           *
 *****************************
 *
 */

// Fonctions aleatoires

float alea(void);
float alea_triangle(float a, float b, float c);
float alea_exp(float lambda);
```

6.3.2.4 Listing de la simulation

- Préambule : lit les fichiers de spécification des bibliothèque, définit les types, constantes et variables globales.

```
/*****************************
 *                           *
 *   M A G A S I N . C P P   *
 *                           *
 *****************************
 *
 */

#include <stdio.h>
#include <assert.h>

#include "fifo.h"
#include "random.h"
#include "magasin_utils.h"

enum Event_Types { RIEN=0, CLIENT, LIVRAISON, FIN };
```

- Paramètres de la simulation : dans l'idéal, ces paramètres devraient être lus au clavier ou dans un fichier, afin de permettre leur modification sans recompilation. L'unité de temps choisie est le jour.

```
int main()
{
   /*** PARAMÈTRES DE LA SIMULATION ***/

   // Parametres arrivee clients (loi exponentielle)
   const float lambda = 1 / 2.0;

     // Parametres livraison (loi triangulaire)
   const float a = 5.0;
   const float b = 7.0;
   const float c = 9.0;

   // Parametres simulation
   const Time tmax = 360.0;
   const int stock_initial = 15;
   const int seuil_commande = 3;
   const int livraison = 12;
```

- Variables de la simulation (d'état et statistiques) :

```
/*** VARIABLES DE LA SIMULATION ***/

   // Variables  statistiques
   int nbre_clients = 0;
   int nbre_insatisfaits = 0;
   int nbre_commandes = 0;
     int stock_maxi = 0;
   int stock_moyen = 0;

   // Variable d'état et entités
   TSEventQueue file_evt(true);
   int stock = stock_initial;
   Time Temps_Simu;
```

- Initialisation de la simulation : crée la file d'attente, pré-calcule les dates d'arrivée des clients, et poste un événement FIN pour s'assurer de l'arrêt de la simulation au bout d'un an.

```
/*** INITIALISATION DE LA SIMULATION ***/

   // Tampon pour construire les événements à poster
   TimeStampedEvent *temp_evt = new TimeStampedEvent(0.0, RIEN);
   Time t;

   file_evt.postEvent(temp_evt);      // Pour démo

   // Poster l'evenement FIN à tmax
   temp_evt->setTimeStamp(tmax);
   temp_evt->type = FIN;
   file_evt.postEvent(temp_evt);

   // Poster les evenements clients jusqu'à 360.0
   t = (Time) alea_exp(lambda);

   while (t < tmax)
   {
      temp_evt->setTimeStamp(t);
      temp_evt->type = CLIENT;
      file_evt.postEvent(temp_evt);
      t = t + alea_exp(lambda);
   }
```

- Boucle de simulation : prend dans la file d'attente l'événement suivant et le traite, jusqu'à ce qu'on arrive sur l'événement FIN. Le temps de la simulation est le temps de l'événement courant (d'où l'importance de respecter l'ordre chronologique !).

```
/*** BOUCLE DE SIMULATION ***/

   // Sortir les evts (CLIENT, LIVRAISON ou FIN) de la file et les traiter
   //
   TimeStampedEvent *evt;
   int evt_type;
   do
   {
      evt = file_evt.getNextEvent();
      assert(evt != NULL);
      Temps_Simu = evt->getTimeStamp();
      printf("%9.3f: ", Temps_Simu);
      evt_type = evt->type;

      switch(evt_type)
      {
         case CLIENT: ...

         case LIVRAISON: ...

         case FIN: ...
      }
       delete evt;             // Détruit l'événement après usage
   }
   while (evt_type != FIN);
```

- Détail du traitement de l'événement CLIENT : met à jour les variables statistiques, gère l'état du stock, et effectue une commande (en fait, génère un événement LIVRAISON) si nécessaire.

```
case CLIENT:
          nbre_clients++;
          printf("CLIENT numero %d - ", nbre_clients);
          if (stock > 0)
          {
            stock--;
            printf("servi");
          }
          else  // stock insuffisant
          {
            nbre_insatisfaits++;
            printf("INSATISFAIT");
          }

          // Faut-il effectuer une commande?
          if (stock == seuil_commande)
          {
             // Calcule la date de livraison
             printf(" (Commande de %d unites)", livraison);
             temp_evt->setTimeStamp(evt->getTimeStamp()+alea_triangle(a,b,c));
             temp_evt->type = LIVRAISON;
             file_evt.postEvent(temp_evt);
             nbre_commandes++;
          }
          printf("\n");
          break;
```

- Arrêt de la simulation : détruit les objets dynamiques afin de libérer proprement la mémoire allouée et affiche les résultats (variables statistiques...).

```
/*** ARRET DE LA SIMULATION ***/

   // Destruction des objets dynamiques
   delete temp_evt;
   // ...

   /*** RESULTATS DE LA SIMULATION ***/

   // Affiche les statistiques collectees
   puts("--------------------------");
   printf("Nombre de clients   = %d\n", nbre_clients);
   printf("Nombre de clients insatisfaits = %d (%2.2f%%)\n",
          nbre_insatisfaits, (float)nbre_insatisfaits / nbre_clients * 100);
   printf("Nombre de commandes = %d\n", nbre_commandes);

   printf("Appuyez sur <Entree>... ");
   getchar();

}   // main()
```

6.3.2.5 Exécution

```
    0.000: RIEN
    0.021: CLIENT numero 1 - servi
    0.029: CLIENT numero 2 - servi
    0.846: CLIENT numero 3 - servi
    0.913: CLIENT numero 4 - servi
    1.792: CLIENT numero 5 - servi
    2.282: CLIENT numero 6 - servi
    3.822: CLIENT numero 7 - servi
    4.258: CLIENT numero 8 - servi
    6.668: CLIENT numero 9 - servi
   12.656: CLIENT numero 10 - servi
   13.299: CLIENT numero 11 - servi
   14.474: CLIENT numero 12 - servi (Commande de 12 unites)
   14.705: CLIENT numero 13 - servi
   17.101: CLIENT numero 14 - servi
   18.763: CLIENT numero 15 - servi
   18.848: CLIENT numero 16 - INSATISFAIT
   19.209: CLIENT numero 17 - INSATISFAIT
   21.714: Livraison de 12 unites
   22.589: CLIENT numero 18 - servi
   24.902: CLIENT numero 19 - servi
   ...
  348.621: CLIENT numero 175 - servi
  348.657: Livraison de 12 unites
  350.284: CLIENT numero 176 - servi
  352.682: CLIENT numero 177 - servi
  353.188: CLIENT numero 178 - servi
  353.930: CLIENT numero 179 - servi
  358.502: CLIENT numero 180 - servi
  358.652: CLIENT numero 181 - servi
  360.000: FIN
--------------------------
Nombre de clients   = 181
Nombre de clients insatisfaits = 16 (8.84%)
Nombre de commandes = 13
```

Nous nous limitons dans notre exemple à une seule exécution, mais dans la mesure où des processus stochastiques interviennent, il faudra en toute rigueur effectuer plusieurs exécutions (par exemple une dizaine, avec un germe aléatoire différent), et faire une étude statistiques des différents résultats.

6.3.2.6 Dépouillement des résultats

1 client sur 12 environ repart insatisfait, ce qui représente une perte de chiffre d'affaire de 4 000 € pour ce seul modèle de lecteur, ce qui est loin d'être négligeable.

La stratégie de gestion du stock est donc à revoir.

En faisant varier les paramètres de la simulation, on pourrait optimiser par exemple les quantités de lecteurs commandés et le seuil de commande.

7. SIMULATION SCIENTIFIQUE

« I cannot conceive that anybody will require multiplications at the rate of 40,000 or even 4,000 per hour ; such a revolutionary change as the octonary scale should not be imposed upon mankind in general for the sake of a few individuals. »

F. H. Wales (1936)

(Voir cours de Mme Montigny)

8. MOTEURS DE SIMULATION ET STRUCTURES D'ACCUEIL

8.1 Rappels de génie logiciel

« As long as there were no machines, programming was no problem at all; when we had a few weak computers, programming became a mild problem and now that we have gigantic computers[74], programming has become an equally gigantic problem. In this sense the electronic industry has not solved a single problem, it has only created them — it has created the problem of using its product. »

Edsger. W. Dijkstra. "The Humble Programmer"
(Conférence aux Turing Awards, 1972)

8.1.1 Définitions

Le *génie civil* est l'art de faire des constructions telles que routes, ponts, etc., suivant leur cahier des charges (coût, délai, fonction, résistance, longévité...). Il ne faut pas faire moins que le cahier des charges ou l'ouvrage d'art ne remplira pas ses fonctions, voire s'écroulera. Il ne faut toutefois pas faire plus, car alors il coûtera inutilement plus cher (plus de matériaux, de main d'œuvre, de temps…).

Le génie logiciel consiste tout simplement à s'assurer que le logiciel produit remplit un certain nombre de critère de qualité.

On appelle **génie logiciel** l'application de méthodes scientifiques au développement des logiciels, afin de favoriser le développement d'un logiciel de qualité.

Un **logiciel de qualité** est un logiciel qui fait ce qu'on attend de lui, pas plus, pas moins.

Assurer la **qualité d'un produit**, c'est tout mettre en œuvre pour que le produit rencontre le besoin du client au meilleur coût.

8.1.2 Pitié !

Comme tout "génie" le génie logiciel[75] a une base scientifique rigoureuse.

Il est inadmissible qu'un programmeur se « fasse plaisir » au détriment des besoins du client ou des impératifs de qualité.

[74] Certes, c'était il y a 30 ans, et les « gigantesques ordinateurs » de l'époque pourrait aujourd'hui prêter à sourire, cependant le discours de Dijkstra n'a jamais été aussi vrai...

[75] *Software Engineering* en anglais. Le terme fut adopté à la suite d'une conférence OTAN à Garmisch, en Allemagne, sur le problème du logiciel, et sonnait déjà comme un cri d'alarme des experts face aux dépassements de budgets, retards et dysfonctionnements des logiciels…

C'est hélas fort courant : les programmeurs, ne sont pas toujours facile à encadrer, notamment en raison d'une tendance à l'indépendance et de la forte technicité de leur travail. Cependant, le logiciel a pris une importance telle de nos jours que les défauts des logiciels engendre des coûts pour la société se chiffrant en milliards d'euros. Un exemple ? Regardez le « bug de l'an 2000 » (Y2K), qui, à l'échelle mondiale, aura coûté des dizaines de milliards de dollars, tout cela pour économiser quelques octets… Si cela était compréhensible dans les années soixante, où la mémoire était très onéreuse, que penser des nombreux logiciels incompatibles avec l'an 2000 sortis dans les années 90 ?

Les informaticiens et les utilisateurs semblent admettre, fort curieusement, qu'un logiciel qui plante deux fois par jour et détruit des fichiers de temps à autre est quelque chose de tout à fait « normal ». C'est une croyance répandue : « *it's not a bug, it's a feature* ». Et c'est un désastre pour l'industrie du logiciel, car les dysfonctionnement d'un programme n'ont rien d'une fatalité.

Accepteriez-vous que votre voiture tombe en panne tous les 100 km ? Que le système de freinage ne réponde que 90% du temps ? Qu'il faille la changer tous les 2 ans parce les pompes à essence ne sont plus compatibles ? Qu'il y ait des autoroutes réservées aux Renault et d'autres aux Peugeot ?

Non, bien sûr. Et pourtant c'est ce qui arrive souvent dans le monde du logiciel, la qualité ne semblant pas toujours être la principale préoccupation des informaticiens, comme nous le rappelle Jacques Printz à propos des tests, en des termes plutôt crus (voir [PRIN98]) :

> « **Le scandale des tests**. Les tests sont la honte de la communauté informatique, tant au niveau industriel, qu'au niveau universitaire. Il est véritablement stupéfiant de constater le faible intérêt suscité par l'étude de la problématique des tests, alors que l'activité de tests représente entre 50 et 75% du coût réel des projets informatiques. L'occultation de l'erreur humaine y est certainement pour quelque chose, mais le vrai problème du test tient à la nature exclusivement sémantique de la définition des tests. Le test est ce qui valide, *in fine*, la bonne correspondance entre le modèle informatique que l'ordinateur exécute et la réalité. »

Le logiciel doit être soumis aux mêmes règles que tout autre produit industriel

Fort heureusement, lorsque des vies humaines sont en jeu, les sociétés appliquent ces règles. Ainsi, les logiciels embarqués des Airbus sont vérifiés formellement, et la probabilité d'apparition d'un bug, si elle n'est pas nulle, est très faible, et tout a été mis en œuvre pour limiter les conséquences d'un tel incident.

Certes, mais quel rapport avec la simulation ?

Certains logiciels de simulation sont très onéreux. Par exemple, la structure ESCADRE ; utilisée pour les simulations technico-opérationnelles au centre d'analyse de défense, comporte plus de 300 000 lignes de codes, et a été à la base d'une quarantaine d'applications de simulation. Un bug dans cette structure d'accueil affectera ainsi l'ensemble des applications dérivées, avec un coût que l'on imagine aisément élevé. D'où le souci permanent des concepteurs de cette application d'assurer la qualité de leur produit.

Cette section a pour objectif de vous donner quelques notions de base de qualité logicielle, tout en ayant à l'esprit qu'il ne faut pas faire de sur-qualité non plus : inutile de développer une simulation d'étude avec les méthodes de génie logiciel utilisées pour la fusée Ariane !

D'ailleurs, puisqu'on parle d'Ariane, rappelez-vous l'explosion du vol 501 en 1997, à la suite d'une défaillance logicielle. Coût : 5 milliards de francs ! En fait, il ne s'agissait pas vraiment d'un « bug », mais d'une erreur de spécification et d'une négligence lors des tests d'intégration.

Terminons cette introduction par quelques chiffres :

- 50% des logiciels ne servent jamais, 80% une seule fois
- 70% du coût global du logiciel est dû à la maintenance
- 42% du coût de maintenance vient d'une modification des spécifications.
- 60 à 80% des erreurs d'un logiciel livré sont des erreurs de spécification ou de conception !
- Au ministère de la défense, on estime qu'il y a 4 bugs pour 1000 lignes de code. Au CNES, cette proportion est de 1 pour 1000.
- Un ingénieur développeur est facturé environ 1 MF / an par les sociétés de service.

8.1.3 Critères de qualité d'un logiciel

On utilise généralement les dix critères suivants, issus des travaux de James McCall[76] et de Bertrand Meyer[77] :

- Validité
- Fiabilité
- Extensibilité
- Réutilisabilité
- Compatibilité
- Efficacité
- Portabilité
- Vérifiabilité
- Intégrité
- Facilité d'emploi

8.1.3.1 Validité

Validité : Aptitude d'un produit logiciel à réaliser exactement les tâches définies par sa spécification.

C'est le facteur de qualité de base. Il faut que les besoins du client soient remplis, mais il ne faut pas non plus passer plus de temps que nécessaire au développement du logiciel car cela engendrerait des coûts inutiles : la sur-qualité, c'est de la non-qualité.

8.1.3.2 Robustesse

Robustesse (ou fiabilité) : Aptitude d'un logiciel à fonctionner même dans des conditions anormales.

[76] McCall réalisa en 1977 une étude sur le problème de l'ingénierie du logiciel pour le compte de General Electric.

[77] Expert français du génie logiciel et de la conception orientée objets, auteurs de nombreux ouvrages.

Le logiciel doit pouvoir diagnostiquer une situation anormale et, s'il le peut, rétablir la situation ou continuer dans un mode dégradé, tout en signalant le problème à l'utilisateur à l'aide d'un message d'erreur clair, qui permette de remonter aisément à la source du problème, qu'elle soit dans les données d'entrée, l'environnement d'exécution ou le programme lui-même. Si le programme ne peut continuer, il doit terminer proprement son exécution en indiquant son diagnostic de la situation. « Proprement » cela veut dire sans perte de données, sans perturber le fonctionnement du reste du système.

Le système d'exploitation Microsoft Windows me fournit régulièrement des exemples de ce qu'il ne faut pas faire dans ce domaine. Lors du « plantage » d'un programme, il se contente le plus souvent d'afficher un message d'erreur générique, accompagné du contenu des registres du processeur, ce qui n'est pas vraiment pratique pour le diagnostic et le debugging du logiciel (voir Figure 94 : si avec ça vous arrivez à comprendre ce qui s'est, vous êtes très fort !). Sans parler des diagnostics faux : ainsi, Word 6.0 pour NT indiquait qu'il n'y avait plus d'espace libre sur le disque dur s'il n'arrivait pas à accéder à un fichier (i.e. si l'utilisateur n'a pas les droits d'accès suffisants). Combien d'utilisateurs ont été dérouté par un tel message, alors que le disque affichait plusieurs Go d'espace disponible[78] ?

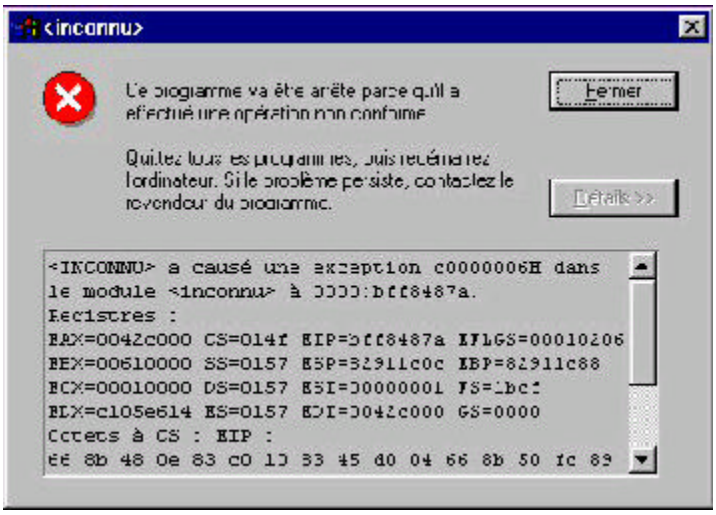


Figure 94: Un message d'erreur bien obscur!

La Figure 95 nous donne en revanche un exemple de diagnostic clair. Il s'agit d'une erreur survenue dans un module Ada de la structure d'accueil ESCADRE (voir §8.3). Ici, on voit qu'une exception (condition anormale de fonctionnement) est survenue lors de l'ouverture du fichier `ar_missile_tno.don`, dont le chemin d'accès complet apparaît. Il s'agit d'un problème portant sur le nom du fichier (`Name_Error`), ligne 118 du module `low_level_io_.verify`, appelé à la ligne 83 du module `io-open_text.adb`.

On a donc immédiatement le type de l'erreur, la donnée et la fonction mises en cause et même le numéro de la ligne fautive, ainsi que toute la pile des appels de sous-programmes.

Évidemment, ce luxe de précision a un prix : ESCADRE est parsemé de code servant à traiter les erreurs. Cela peut s'avérer extrêmement lourd à la longue, mais ce coût est

[78] Ceci est d'autant moins admissible qu'il s'agit vraisemblablement d'un acte de paresse du programmeur, qui a interprété le code d'erreur du système (renvoi d'un pointeur NULL) comme une erreur particulière, sans chercher à établir un diagnostic plus précis.

largement compensé par un meilleur diagnostic des bugs et donc une mise au point facilitée.

Il faut dans tous les cas doser la rigueur de la programmation en fonction de la criticité du logiciel. Ainsi, un système d'exploitation ou un logiciel bureautique destinés à être diffusés à des dizaines de millions d'exemplaires devraient être irréprochables !

```
Io.Open_Text.Open: File managment error
  File_Name  => INPUT_DIR:ar_missile_tno.don
  Translated => D:\Users\sautereau\ada95\apps50c\src\air\data\ar_missile_tno.don
  Exception  => Ada.IO_Exceptions.Name_Error
  ---- Exception_Information ----
  Ada.IO_Exceptions.Name_Error ()
  Exception traceback:
    118   system.rts.low_level_io_.verify            low_le$1.bdy
    190   system.rts.low_level_io_.open_create       low_le$1.bdy
    278   system.rts.low_level_io.open               low_le$1.bdy
     79   system.rts.tgt.char_io.open                tgt_ch$1.bdy
    283   ada.text_io.open                           text_io.bdy
     83   io.open_text.open                          io-open_text.adb
  Exception handled at:
     85   io.open_text.open                          io-open_text.adb
```

Figure 95: Exemple de diagnostic clair...

8.1.3.3 Extensibilité

Extensibilité : Facilité d'adaptation d'un logiciel à une modification ou une extension de ses spécifications[79].

Un logiciel qui n'est pas conçu pour évoluer facilement peut coûter très cher par la suite, puisque, comme nous l'avons vu précédemment, 30% du coût global d'un logiciel est dû à des changements de spécifications. Et encore, ce n'est qu'une moyenne…

8.1.3.4 Réutilisabilité

Réutilisabilité : Aptitude d'un logiciel à être réutilisé en tout ou en partie pour de nouvelles applications.

C'est du bon sens : il ne faut pas réinventer ce qui existe déjà. Comme nous le rappelle Eric Raymond (voir [RAYM97]) :

> « Good programmers know what to write. Great ones know what to rewrite (and reuse). »

En fait, un bon programmeur est un programmeur… paresseux !

En fait cette « paresse » permet de minimiser les coûts et délais, tout en améliorant la qualité du produit, puisqu'on peut alors se concentrer sur la qualité du nouveau code, l'ancien ayant déjà été testé et validé[80].

[79] Le terme anglais est « scalability », et revient souvent dans le domaine de la simulation, celle-ci devant en effet, au cours de leur vie, s'accommoder des changements dans les caractéristiques des systèmes qu'elles simulent.

La réutilisabilité est une qualité très importante, à la base de la programmation orientée objet. C'est un enjeu d'importance dans le monde de la simulation, comme nous le verrons par la suite, et est la principale raison d'être des structures d'accueil.

8.1.3.5 Compatibilité

Compatibilité : Aptitude d'un logiciel à pouvoir être combiné avec d'autres programmes.

La compatibilité est une qualité qui permet à différents logiciels de collaborer entre eux, par exemple par le biais d'une interface de programmation (API) commune. C'est l'objectif de standards tels que CORBA, DIS, HLA…

8.1.3.6 Efficacité

Efficacité : Aptitude d'un programme à bien utiliser les ressources du matériel.

Un logiciel doit être suffisamment efficace pour remplir son cahier des charges. Mais il ne faut toutefois pas surestimer ce critère : on voit souvent des programmeurs perdre du temps à optimiser leur logiciel, alors que celui-ci est pleinement fonctionnel tel quel, et que le gain ne sera à peine sensible.

Pour certains logiciels toutefois, l'efficacité peut être crucial. Ainsi, les systèmes d'exploitation, les logiciels de calcul numérique (le temps d'utilisation d'un calculateur peut être facturé très cher) ou les systèmes temps réel (générateurs d'images, simulateurs de vol…).

Il existe des logiciels spécialisés, appelés *profilers*, qui calculent le temps d'exécution de chaque instruction d'un programme et permettent de savoir exactement où un programme perd du temps. Et comme le dit un dicton d'informaticien, « ce n'est jamais là ou l'on pensait »…

8.1.3.7 Portabilité

Portabilité : aptitude du logiciel à être adapté à différents environnements.

L'importance de ce critère dépend beaucoup du cahier des charges et de la longévité attendue du logiciel. En effet, en informatique, les environnements (plate-forme matérielle, systèmes d'exploitation, interfaces, etc.) évoluent très rapidement. Dès qu'un logiciel est censé avoir une pérennité supérieure à quelques années, il faut envisager de futurs portages sur de nouveaux environnements.

La portabilité peut être contraignante pour le développeur (utilisation de langages normalisés et d'interfaces standards, organisation du logiciel en couches indépendantes, discipline de codage, etc.), mais, prise en compte dès le départ, son coût peut être très raisonnable. Mais dans le cas contraire, gare à la facture !

J'ai un parfait exemple de problème que peut poser une négligence dans ce domaine. Un logiciel très complexe a été développé pour le compte d'un ministère par un grand groupe industriel. Résultat : plus de 100 millions de francs, dix années d'études et de développement, près d'un million de lignes de code. Et ce monstre tournait uniquement

[80] Il faut toutefois se méfier, car un composant peut être validé dans un domaine particulier et ne pas convenir dans un autre contexte. Pour revenir à l'exemple du vol 501 d'Ariane, c'est exactement ce qui s'est produit : la centrale de navigation d'Ariane 4 a été réutilisée pour Ariane 5 sans passer de tests de validation correspondant à cette nouvelle plate-forme. Or, les paramètres dynamiques d'Ariane 5 sortaient du domaine de validité du logiciel de cette centrale, d'où une défaillance en cours de vol qui amena la destruction de la fusée.

sur des stations de travail Sun, avec un système d'exploitation Unix, SunOS. Or, peu de temps avant sa sortie, la société Sun décide de remplacer SunOS par un nouveau système, Solaris, plus moderne et plus performant. Et cessa la maintenance de SunOS et n'assura plus la compatibilité de ses nouveaux ordinateurs avec ce système. Pas de chance, le logiciel en question n'était pas entière portable, et le coût du passage en Solaris nécessitait la réécriture d'une partie du code, pour un coût de plusieurs millions de francs. Ajouté à une quantité pléthorique de bugs, et à une conformité médiocre du produit à ses spécifications initiales, cela conduisit à l'abandon pure et simple de ce logiciel, qui n'entra donc jamais en service...

Il y a dix ans, SunOS était le haut du panier en matière du système d'exploitation, et, apparemment, il avait été estimé que sa disparition brutale était peu probable. Malheureusement, on ne peut être sûr de rien pour ce qui est des évolutions de l'informatique...

8.1.3.8 Vérifiabilité

Vérifiabilité : Facilité de préparation des procédures de recette et de certification.

Il s'agit de tout mettre en œuvre pour faciliter les tests et la mise au point (*debugging*) du logiciel, par intégration de modes de mise au point, la création de jeux de tests, doublés éventuellement d'un calcul du taux de couverture des tests.

L'**instrumentation du code** est souvent négligé, mais est précieux pour les tests et la mise au point du logiciel. Il s'agit d'introduire dans le code du logiciel des instructions imprimant sur un terminal ou dans un fichier les actions effectuées par le code (ces sorties sont appelées **traces**). Ces instructions ne sont activées que dans un mode dit de « debugging » ou de « trace ». Voici des exemples de traces :

```
TRACE: [MemAllocator] free memory = 28576 KB
TRACE: Test.c 99 : Lecture du fichier
TRACE: Test.c .main() : Read configuration file
TRACE: [Portable_OS] Package elaboration successful
```

On conçoit bien qu'un mode de debugging allié à un diagnostic d'erreur (voir §8.1.3.2) est lourd à mettre en œuvre, mais améliore considérablement la fiabilité du logiciel, tout en réduisant de façon drastique les coûts de mise au point.

8.1.3.9 Intégrité

Intégrité : Aptitude des logiciels à protéger leurs différentes composantes (programmes, données, documents) contre des accès ou des modifications externes.

Il s'agit de se protéger contre les virus et les actes de malveillance, mais aussi contre les erreurs de programmation.

Ce critère est assuré en partie par le système d'exploitation (contrôle des droits accès à des données, gestion des accès concurrents à une ressource…) et/ou un logiciel intermédiaire (*middleware*), comme le RTI de HLA.

8.1.3.10 Facilité d'emploi

Facilité d'utilisation : facilité avec laquelle les utilisateurs d'un logiciel peuvent apprendre comment l'utiliser, le faire fonctionner, préparer les données, interpréter les résultats et aussi diagnostiquer et réparer les effets des erreurs éventuelles.

Il faut en effet songer à l'utilisateur. Les interfaces graphiques ont globalement amélioré la situation, mais elles ne font pas tout : la confusion des menus, l'absence de documentation claire, l'envoi de message d'erreurs sous forme de codes numériques, tout cela concours à gêner la tâche de l'utilisateur.

8.1.4 Cycle de vie d'un logiciel

8.1.4.1 Cycle en cascade

Le cycle de vie d'un produit représente la succession des étapes menant du besoin du produit à son exploitation, voire son retrait du service.

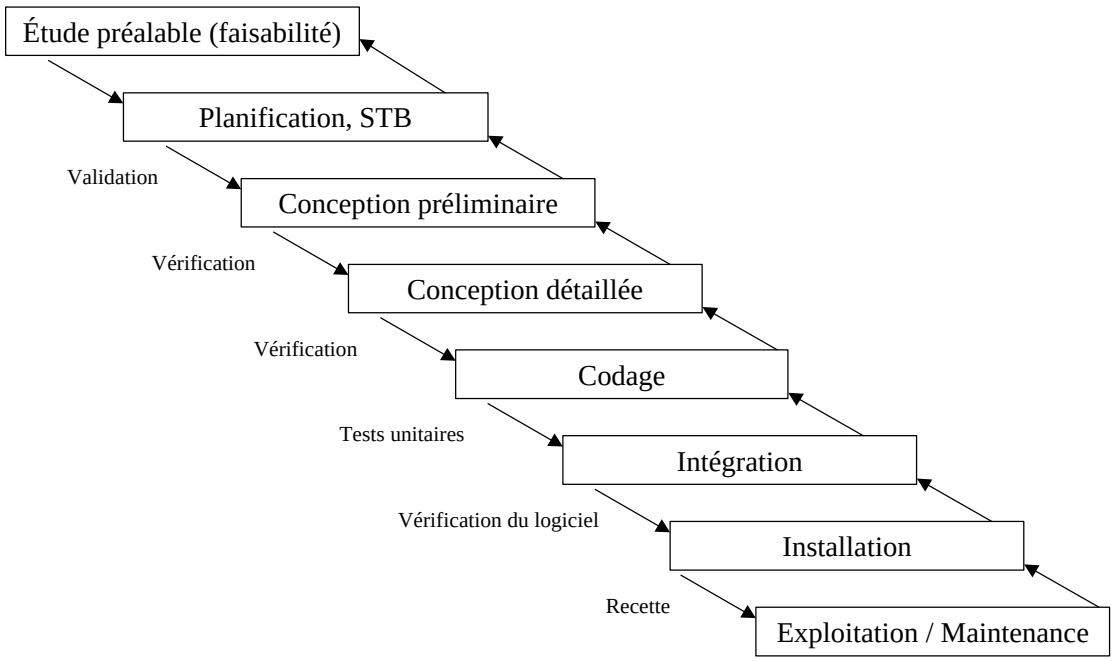


Figure 96: Cycle de vie logiciel en cascade (avec retour)

Pour un logiciel, le cycle de vie « classique » est montré par la Figure 96.

8.1.4.2 Étude préalable

L'**étude préalable** correspond à des études amonts. Elle comprend une phase exploratoire, durant laquelle est mis en évidence le besoin et estimé le budget et la faisabilité générale du produit. Puis vient une phase conceptuelle, durant laquelle est rédigé un cahier des charges, calculé un budget précis et planifié le projet.

8.1.4.3 Spécification

Il consiste à transformer l'expression générale du besoin et souvent informelle que contient le cahier des charges à une description préciser du système à réaliser : fonctionnalités, interfaces, flux d'informations, performances, qualité, etc.

Si cela n'a pas déjà été fait durant l'étude préalable, il faut en outre dès la spécification se poser la question de la validation finale du produit : à partir de quels jeux de tests et procédures va-t-on pouvoir le qualifier ?

La spécification engage le reste du développement. Une erreur à cette étape peut coûter extrêmement cher. Les défauts classiques d'une spécification sont :

- Incomplétude : il manque certains détails dans les fonctionnalités décrites.
- Incohérence : il y a des contradictions au sein de la spécification.
- Ambiguïté : une même description peut avoir plusieurs interprétation (notamment suivant la culture du lecteur).
- Manque de clarté : le texte est difficile à comprendre. Même s'il n'est pas nécessairement erroné, sa complexité peut constituer une source d'erreurs.
- Sur-spécification : certaines descriptions suggèrent une solution, ce qui relève du domaine de la conception. La spécification doit se cantonner au besoin.
- Bruit : la spécification comporte des détails inutiles, voire sans rapport avec son objet.
- Redondance : une même description se retrouve à plusieurs endroits, ce qui non seulement rallonge inutilement le document, mais en plus nuit à sa maintenance (risque d'apparition d'incohérence en cas de modifications).
- Non validable : tout ou partie des exigences exprimées ne peuvent être validées, ce qui rendra délicate, voire impossible, le contrôle de la fourniture finale.
- Non conformes au besoin : décrivent un besoin autre, plus large ou plus restreint que le besoin réel.

8.1.4.4 Conception générale

Une fois le système clairement et formellement spécifié, on va pouvoir commencer sa réalisation. Dans un premier temps, on conçoit son architecture globale, c'est-à-dire établir la liste des composants logiciels qui le composeront et leurs relations, ainsi que les principales structures de données utilisées.

Dans une approche orientée objets, cela consisterait notamment à déterminer l'arborescence des classes du système.

Une **revue** de conception générale est effectuée pour valider les choix.

8.1.4.5 Conception détaillée

On raffine la conception générale pour chaque composant logiciel jusqu'au niveau de détails le plus bas.

Chaque conception de sous-ensemble fait l'objet d'une ou plusieurs revues de conception.

8.1.4.6 Codage

À partir de la conception détaillée, on écrit les programmes correspondants.

Dès cette étape des opérations de vérification du code doivent être effectuée, afin de déceler les erreurs le plus en amont possible.

8.1.4.7 Tests unitaires

Chaque composant logiciel est testé et validé individuellement par rapport à ses spécifications.

8.1.4.8 Intégration

Les composants sont rassemblées au sein du système afin de pouvoir collaborer suivant les spécifications globales du système.

On observe souvent des propriétés émergeantes (comportement collectif non prévu) lors de cette phase, en nombre d'autant plus grand que le système est complexe.

8.1.4.9 Validation globale

On valide l'ensemble du système pour vérifier sa conformité aux spécifications. On suit pour cela le plan de validation établi lors de la phase de spécification.

Cette opération est également appelée **recette** du logiciel.

Notons qu'un échec lors de la validation unitaire ou globale peut nécessiter une modification du codage, mais aussi parfois de sa conception, voire des spécifications (incomplétude ou contradiction dans les spécifications par exemple). Le coût d'une telle modification peut être très élevé, d'où l'importance d'une détection la plus en amont possible de ces erreurs (par exemple, pour des systèmes critiques tels qu'un système de transport, on utilise parfois des méthodes formelles pour valider les spécifications ou la conception[81]).

8.1.4.10 Mise en service

Cette phase consiste à livrer et installer le produit avec sa **documentation** chez le client. Cela peut inclure des activités de formation.

Cette phase est loin d'être anodine, car lors des premiers mois d'utilisation en situation réelle, il est fréquent de voir surgir des problèmes non décelés auparavant lors de la validation (les tests ayant tendance à se faire dans des « conditions de laboratoire »). Il faut alors collecter les rapports d'incidents et procéder aux corrections nécessaires.

[81] Il est intéressant de noter à ce sujet que le métro METEOR (ligne 14 du métro parisien) a été spécifié en langage B et a subi une validation formelle durant sa conception. De fait, malgré la complexité de ce système, il n'a souffert depuis sa mise en service d'aucun problème (bug ou panne) majeur, ce qui est assez remarquable.

8.1.4.11 Exploitation et maintien en condition opérationnelle

Le logiciel peut être utilisé de nombreuses années, et devoir évoluer. De plus, certaines erreurs peuvent surgir plusieurs années après la mise en service.

Ainsi, à l'occasion du remplacement du système de contrôle aérien de West Drayton, qui fonctionnait correctement depuis des années, un bug fut découvert dans un programme en Cobol écrit... 20 ans auparavant !

Un système, et tout particulièrement s'il est à base de logiciel, est donc « vivant » tout au long de sa vie, et évolue constamment. Ceci doit être bien évidemment pris en compte très en amont pour éviter des surprises désagréables...

8.1.4.12 Cycle de vie en V

Cette représentation du cycle de vie d'un logiciel met en valeur les correspondances entre les tests de validation et ce qui est validé. Ainsi, les tests unitaires valident la conception détaillée, etc.

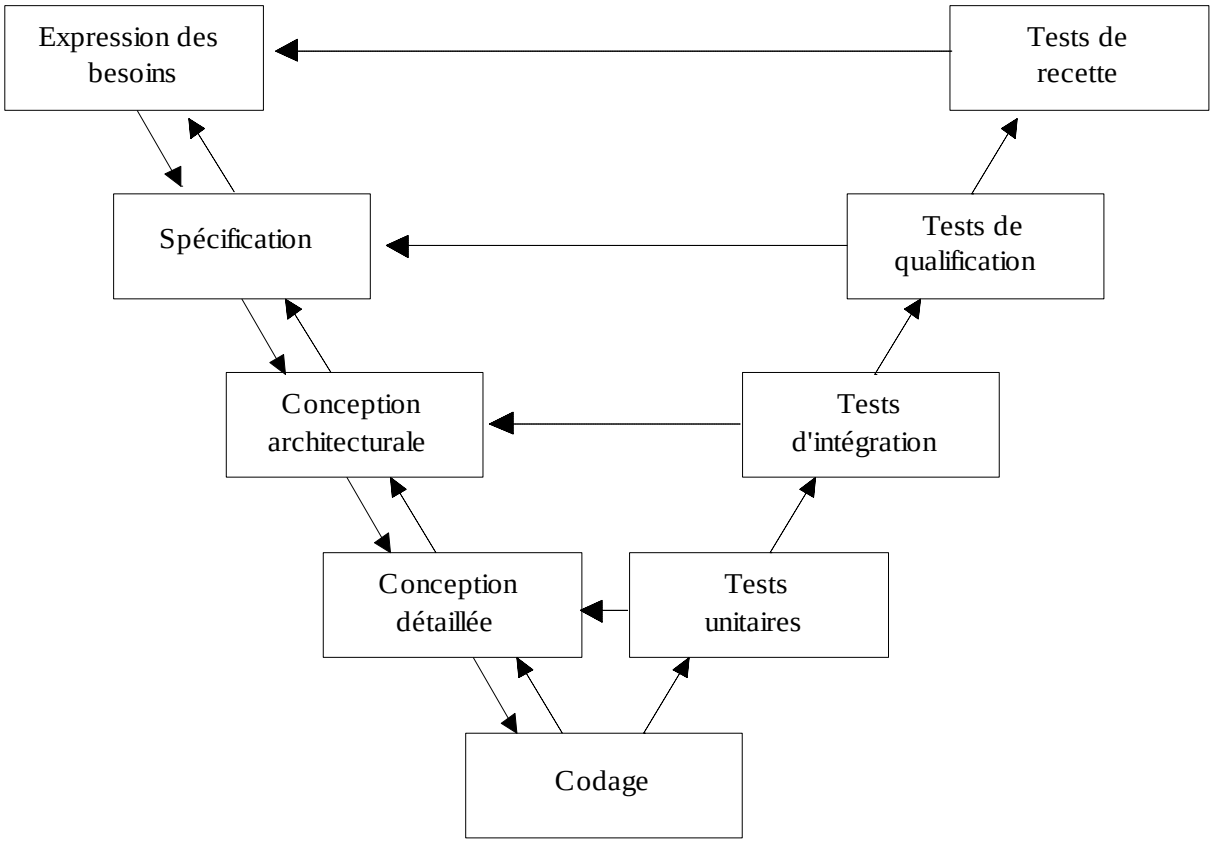


Figure 97: Modèle en V

8.1.4.13 Autres modèles de cycle de vie

Il existe de nombreuses autres façons de représenter le cycle de vie d'un logiciel. Notamment, outre les modèles « cascade » et « V », on trouve assez souvent celui dit « en spirale », adapté aux développements dits évolutifs ou incrémentaux (le produit final est obtenu par ajouts successifs au produit initial). Le cycle en spirale repose sur des maquettes (ou des simulations) successives d'un raffinement croissant.

8.2 Structures d'accueil

Une structure d'accueil de simulations ou structure d'accueil de modèles est un ensemble de méthodes, de logiciels et, parfois, de matériels, permettant de réaliser une simulation par intégration de modèles ou de simulation existants.

Terme anglais : *simulation framework* ou *simulation support environnement*.

Typiquement, une structure d'accueil fournit au développeur de simulation les services de base d'une simulation : moteur de simulation, gestion du temps, gestion des objets d'une simulation et de leurs interactions, accès au système d'exploitation, interface graphique, fonctions mathématiques et statistiques, journalisation des données, etc.

Elle comprend généralement une méthodologie de conception et d'utilisation[82], parfois réduite à une liste d'exigences ou de conseils, qui assure la réutilisabilité des modèles et leur compatibilité avec la structure d'accueil et permet ainsi de capitaliser ces modèles.

Les structures d'accueil permettent de diminuer considérablement les coûts et les délais de développement et d'améliorer notablement la qualité logicielle de la simulation.

En effet, on remarque que 30 à 60% du code d'une simulation est consacré à ce que l'on peut considérer comme des services de base (voire définition ci-dessus), qui sont communs à toutes les simulations. En les factorisant au sein d'une structure d'accueil, on économise non seulement le développement du code correspondant, mais en plus on améliore la **fiabilité** de ce code, puisque la structure d'accueil aura été maintes fois utilisée et testée, et on augmente les fonctionnalités, car on a d'emblée à sa disposition un ensemble de services élaboré. Le développeur de la simulation peut ainsi se concentrer sur son métier, à savoir la modélisation de systèmes (radar, missile…), et non sur des problèmes d'informatique pure. Cette factorisation permet enfin de découpler les modèles de la plate-forme logicielle et matérielle, et ainsi de leur conférer un haut degré de **portabilité**.

La Figure 98 montre une architecture typique, dans laquelle les modèles sont découplés de l'interface homme-machine et du système d'exploitation, et s'interfacent avec la structure d'accueil par une API (interface de programmation). Un changement de système d'exploitation ou d'IHM n'affecte en rien les modèles, bien qu'il puisse être souhaitable de ne pas multiplier les IHM, afin de minimiser la formation requise pour les utilisateurs.

[82] Car l'utilisation d'une structure d'accueil doit s'inscrire dans un processus global touchant tout le cycle de vie de la simulation.

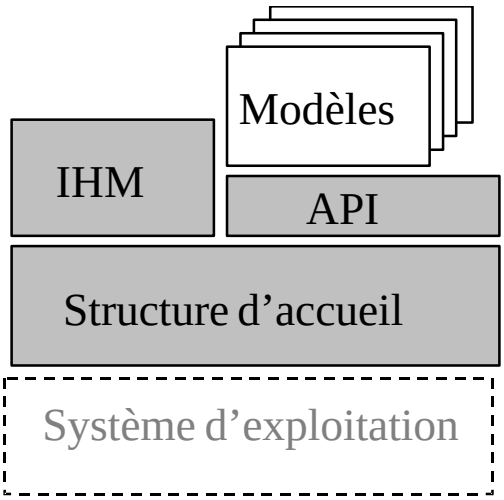


Figure 98: Architecture typique d'une simulation basée sur une SA

À moyen terme, l'utilisation d'une structure d'accueil favorise la **réutilisation** des modèles. En effet, en prenant soin de capitaliser les modèles au fur et à mesure de leur développement, on peut se constituer une bibliothèque de modèles dans laquelle on viendra « piocher » en cas de besoin. Par exemple, si on veut modéliser l'action d'un nouveau missile anti-radar, le développeur se contentera de modéliser le missile et aura la possibilité de prendre dans la bibliothèque un modèle de radar (pour la cible) et un modèle d'avion (pour le porteur), pour peu qu'il y en ait un qui convienne.

Ceci est également une source de gains importants en coûts, en délais, et en qualité, puisque le code du modèle aura déjà subi une **validation**, dans un **domaine de validation** donné. Néanmoins, cette **capitalisation des modèles** exige un effort certain de documentation et d'organisation.

8.2.1 Problèmes à résoudre

8.2.1.1 Organisation

Les structures d'accueil sont d'autant plus efficaces qu'elles s'insèrent dans une **politique globale** de haut niveau. Les bénéfices sont en effet fonction du nombre de développeurs l'ayant adoptée.

8.2.1.2 Les facteurs humains

Ils constituent souvent, hélas, le principal obstacle à l'adoption d'une structure d'accueil par un ensemble d'équipes de développeurs de simulation. La réticence des développeurs peut avoir de nombreuses origines, plus ou moins rationnelles :

- Sentiment d'inadéquation au besoin (les fonctionnalités du nouveau produit suggéré n'étant pas similaire à ceux auxquels on est habitués, on a tendance à soupçonner d'emblée des incompatibilités, des carences...).
- Résistance à l'évolution des habitudes (il faut se remettre en question, changer sa façon de travailler, se former...).
- Syndrome du « *Not Invented Here* » (on se méfie de ce qui vient de l'extérieur)

- Résistance à l'autoritarisme (l'informaticien se considère souvent comme un artiste ou un scientifique, non comme un ingénieur réalisant un produit industriel).
- Peur de l'inconnu (difficile d'abandonner ce qui marchait pour un nouveau produit, même si c'est l'intérêt commun : ceci a pu être observé lors du passage de DIS à HLA).
- Vision à court terme (l'adoption d'une structure d'accueil requiert d'importants efforts, mais sa rentabilité ne se fait sentir qu'au bout de quelques années).
- ...

8.2.1.3 Diversité des besoins

La simulation comporte de nombreux domaines aux besoins parfois très éloignés : entre une simulation technico-opérationnelle et un simulateur de vol piloté, les services et les modèles ne sont pas nécessairement les mêmes.

Une structure d'accueil répondant à tous les besoins risquerait donc d'être extrêmement complexe et difficile à maîtriser, ce qui n'est pas souhaitable. On peut en revanche définir un nombre limité de domaines homogènes pour le niveau de granularité des modèles et les exigences de performances, et définir une solution adéquate pour chacun de ces domaines.

8.2.1.4 Modèles hérités

Il n'est généralement pas envisageable financièrement de ne pas tenir compte des modèles et applications développés avant la mise en place de la structure d'accueil (*legacy models* ou modèles hérités). Une bonne structure d'accueil doit donc être capable d'intégrer des modèles qui n'ont pas été conçus pour elle.

8.2.1.5 Résistance aux modes

Lors de la conception d'une structure d'accueil, il est nécessaire de faire un certain nombre de choix techniques : plate-forme, langage, outils, etc.

Il faut garder à l'esprit qu'en informatique les solutions techniques ne durent pas éternellement. Ainsi, le système d'exploitation DEC VMS, très prisé dans le milieu américain de la simulation, est-il voué à disparaître d'ici peu, rendant de fait obsolètes nombre de produits.

La structure d'accueil doit donc être facilement portable d'un système à l'autre, ou d'un langage à l'autre, et ne pas dépendre de produits propriétaires dont la pérennité ne serait pas garantie.

Il est également nécessaire de bien découpler la conception des modèles de leur implémentation.

8.2.2 Conclusion

Les structures d'accueil permettent d'améliorer la qualité des simulations tout en diminuant les coûts et les délais de développement. En revanche, elle peuvent rapidement devenir complexes à développer et à mettre en œuvre, et nécessitent une approche rigoureuse de la conception et du développement des simulations pour être pleinement efficaces.

8.2.3 Exemples de structures d'accueil

8.2.3.1 ESCADRE

Utilisée au CAD depuis 1987 dans le cadre d'études technico-opérationnelles, Cette structure d'accueil est basée sur une méthodologie orientée objet (Booch). Ce produit comporte 300 000 lignes d'Ada95 et est disponible sous plusieurs systèmes d'exploitation (dont NT, Linux et Solaris). Plus d'une trentaine d'applications différentes ont été faites avec ESCADRE.

La version 5.1, disponible depuis début 2000, est pleinement orientée objet et intègre HLA. Elle fut d'ailleurs à la base de la première simulation française à passer avec succès les tests de certification HLA américains en juin 2000.

SOFIA, utilisée, entre autres, au CTSN, est une structure d'accueil très similaire à ESCADRE.

ESCADRE représente un bon exemple, avec un important retour d'expérience dû notamment à sa longévité et à l'usage intensif qui en a été fait, même s'il est limité à un domaine particulier de la simulation. C'est pourquoi nous le présenterons plus en détail.

8.2.3.2 SAGESSE

SAGESSE est destinée par le CEV à des simulations d'étude, temps réel et pilotées (RAFALE, TIGRE…). Les travaux ont débuté en janvier 1998, et les premiers éléments seront disponibles fin 2000.

Cette structure d'accueil a pour particularité d'inclure du matériel. Elle a des capacités de simulation distribuée et de simulation hybride. La conception des simulations se fait grâce à un langage de description formelle de simulation.

8.2.3.3 LEGOS

Cette structure d'accueil de l'ETBS est particulièrement orientée vers la réutilisation de modèles existants (hérités). Elle a ainsi une capacité multi-langage, et une grande importance a été accordée à sa portabilité et son ouverture.

Son architecture est du type client-serveur (Unix). LEGOS est capable de choisir le calculateur à utiliser en fonction de la charge, et ce de façon transparente pour l'utilisateur.

Les travaux, lancés en 1994, devraient aboutir en 2001.

8.2.3.4 CASTOR

Utilisée à l'ONERA, CASTOR est destinée à des simulations scientifiques, donc avec un niveau de modélisation très fin, nécessitant d'importantes puissances de calcul.

Le principal problème que l'ONERA a voulu traiter avec CASTOR était la difficulté de la mise en œuvre des modèles. Ainsi, la description du scénario et des modèles est faite via des fichiers texte dans un langage quasi-naturel, la simulation étant construite ensuite automatiquement.

8.2.3.5 JMASS

Le programme américain JMASS (*Joint Modeling and Simulation System*) vise à fournir une architecture pour la simulation numérique, utilisable dans des domaines variés (surtout l'acquisition), et permettant le développement, le test, la configuration et l'exécution (et l'analyse des résultats) de simulations interopérables entre elles à travers le standard HLA.

JMASS facilite également la réutilisation des composants logiciels en permettant la création de librairies cohérentes de modèles et de simulations. En effet, JMASS intègre un modèle de structure logicielle (SSM) qui permet au développeur de modèle de générer des modèles orientés objets (*Model Components*) avec une interface standard. Cette génération de code assure ainsi l'uniformité des interfaces entre modèles et des méthodes de développement.

Un type d'objet particulier, le *Port*, assure la communication entre les différents composants de haut niveau (*Players*) de la simulation.

La structure d'accueil JMASS existe depuis 1992, date à laquelle elle n'était que la fusion entre plusieurs structures d'accueil de moindre envergure. Utilisant initialement le langage Ada, elle passa au C++ en 1996, afin de s'ouvrir à un plus large public. Elle prit une nouvelle dimension avec la version J98 (qui ne fut vraiment au point qu'en 2000), complètement réécrite, qui apporte notamment la conformité à HLA et un fonctionnement convivial sur des matériels peu coûteux de type PC, en plus des stations de travail Unix (SGI, Sun).

Aujourd'hui, JMASS est utilisé dans une centaine de sites, essentiellement aux USA, pour des simulations de niveau système et sous-système. Cette structure d'accueil est devenue un élément clé de la politique SBA (voir §2.3.1.8) du département de la défense américain, avec dans l'optique d'améliorer la qualité des simulations tout en réduisant leur coût[83].

8.2.3.6 DEVS

DEVS (*Discrete Event System Specification*) est un formalisme de l'université d'Arizona pour la simulation à événements discrets, apparu en 1976, dont le maître à penser est B. Zeigler. Ce formalisme se focalise sur les changements des valeurs des variables.

[83] Notons que pour y arriver, les investissements furent conséquents, puisqu'à elle seule JMASS a coûté plus de 100 M$ en développement, soit 30 fois plus qu'ESCADRE...

Le formalisme DEVS définit comment générer de nouvelles valeurs pour les variables et le temps nécessaire pour que ces changements prennent effet.

Un modèle DEVS est de la forme : M = <X, S, Y, d_{int}, d_{ext}, d_{con}, l, ta>, avec :

- X : ensemble d'événements externes en entrée
- S : un ensemble d'états séquentiels
- Y : un ensemble de sorties
- d_{int}: S $\rightarrow$ S : fonction de transition interne
- d_{ext} : Q $\times$ X^b $\rightarrow$ S : fonction de transition externe, avec
  Q = { (*s*,*e*) | *s* $\in$ S, 0 $\leq$ *e* $\leq$ ta(*s*) }
  X^b est un sous-ensemble d'éléments de X
  d_{ext}(s,e,ϕ) = (s,e);
- d_{con} : S $\times$ X^b $\rightarrow$ S : fonction de transition confluente
- λ : S $\rightarrow$ Y^b : fonction de génération des évènements externes en sortie
- ta : S $\rightarrow$ Real : fonction d'avancement du temps
- *e* : temps écoulé depuis la dernière transition d'état

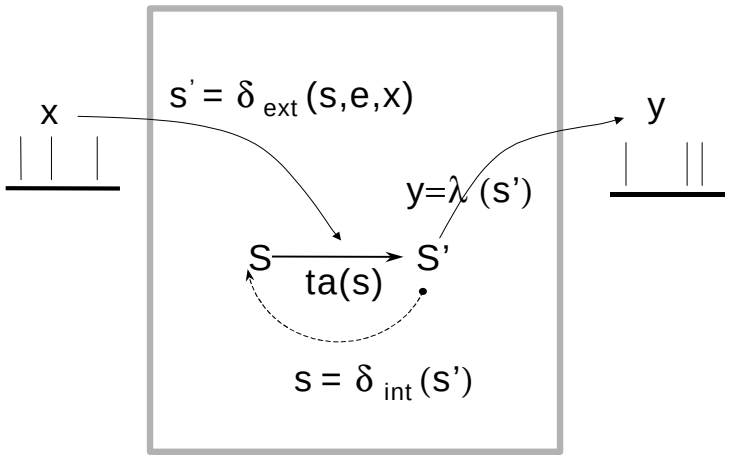


Figure 99: Représentation d'un modèle DEVS simple

La principale implémentation du formalisme DEVS est DEVS-C++[84], qui met à la disposition des développeurs deux principales classes de bases : *atomic* et *coupled*. La classe atomique réalise le niveau de base du formalisme DEVS, tandis que le modèle couplé est une composition hiérarchique de plusieurs modèles.

La Figure 99 représente schématiquement un modèle très basique. Les événements ont des origines internes ou externes. L'état d'un modèle, qui est persistant entre deux événements, est caractérisé par un ensemble de variables dans la représentation objet de DEVS. Par exemple, une file d'attente sera caractérisée, très classiquement, par sa longueur (nombre d'entités en attente dans cette file). Quand une nouvelle entité arrive, la variable longueur est incrémentée d'une unité.

[84] Il existe également un DEVSJAVA utilisant, comme son nom l'indique, le langage Java.

La fonction de transition externe (d_{ext}) analyse l'entrée et détermine l'état résultant, en fonction de l'état courant et le temps que le modèle a passé dans cet état.

Le temps séjour dans un état quelconque (*residence time)* est donné par la fonction d'avancement du temps ta. La fonction de transition interne (d_{int}) change l'état lorsque le temps de séjour s'est écoulé sans que le système n'ait été stimulé par une entrée. Par exemple, pour la file d'attente, ce temps de séjour pourra correspondre au temps de traitement d'un client dans la file. La fonction de transition interne retirera de la file les entités qui ont été traitées, ce qui réduira d'autant la longueur de la file.

La fonction de transition confluente (d_{con}) décide, elle, de ce qu'il faut faire lorsque le système est stimulé simultanément par des événements internes et externes. Une des approches possibles est d'appliquer les fonctions d_{ext} et d_{int} dans un ordre prédéterminé.

La fonction de sortie (λ) génère les sorties en fonction de l'état courant quand elle est activée juste avant un événement interne.

Pour un modèle couplé, il faut spécifier les modèles qui le composent et définir le couplage souhaité (d'où découlent les liens de communication à utiliser).

Les modèles couplés sont de la forme : DN = <X, Y, D, {M_i}, {I_i}, {$Z_{i,j}$}>, avec :

- X : ensemble d'événements externes en entrée
- Y : un ensemble de sorties
- D : un ensemble de noms de composants. Pour chaque *i* dans D :
  - M_i est un modèle (composant)
  - I_i est l'ensemble des influences pour *i*. Pour chaque *j* dans I_i :

    $Z_{i,j}$ est la fonction de translation des sorties de *i* vers *j*.

Les modèles couplés sont implémentés dans DEVS-C++ par une classe de modèles *coupled*. Une instance de cette classe contient les informations suivantes :

- La liste des composants
- La listes des ports d'entrée à travers lesquels les événements externes sont reçus
- La liste des ports de sortie à travers lesquels les événements destinés à l'extérieur sont envoyés
- Une spécification de couplage constituée de la liste des connexions entre ports d'entrée et de sortie des différents composants.

La Figure 100, tirée de [ZEIG98], donne la hiérarchie de base des classes de DEVS-C++. Pour plus de détails, on pourra se référer à [ZEIG98] par exemple.

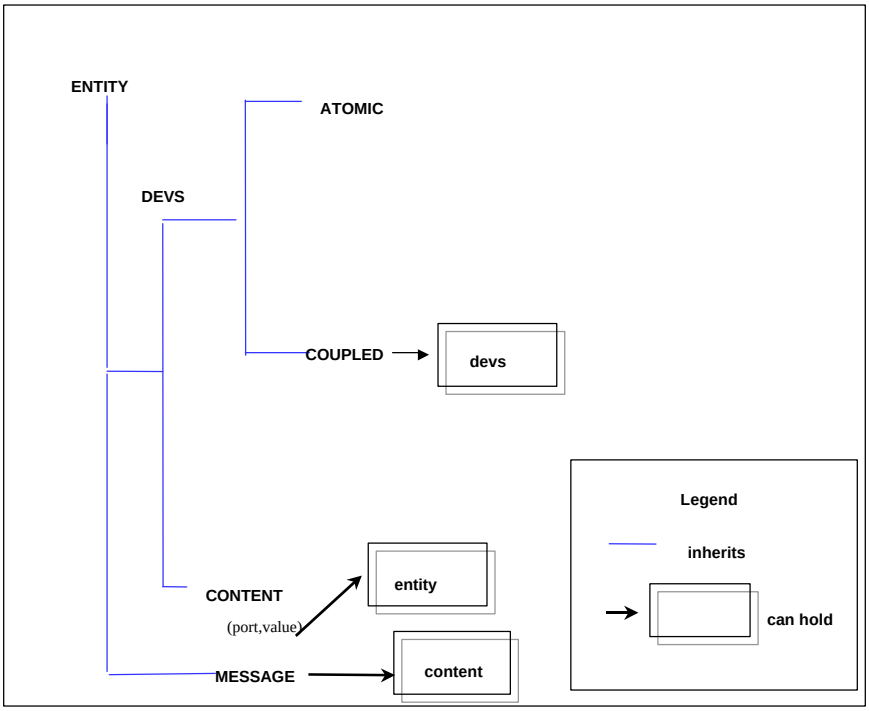


Figure 100: Hiérarchie des classes DEVS-C++

Un récent développement ouvre de vaste perspectives pour DEVS, puisqu'il s'agit d'un environnement conforme au standard HLA (voir §9.5.4.3) appelé DEVS/HLA.

DEVS/HLA est basé sur DEVS-C++ dont il réalise l'adaptation à l'API C++ du RTI HLA. Alors que HLA permet l'interopérabilité au niveau application de simulation, DEVS/HLA offre des fonctionnalités héritées de DEVS au niveau des modèles (formalisme générique pour modéliser des systèmes dynamiques, concept bien défini de couplage de composants, construction hiérarchique et modulaire, support de la modélisation approchée par événements discrets de systèmes continus, bases orientées objet pour la capitalisation de modèles, etc.). De plus, DEVS/HLA permet la construction de modèles de haut niveau avec le formalisme DEVS tout en masquant les détails informatiques sous-jacents, notamment les problèmes de programmation inhérents à tout application participant à une fédération HLA :

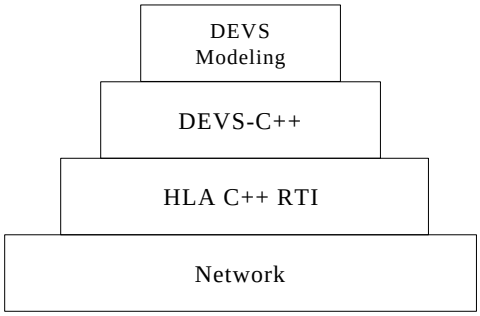


Figure 101: Niveaux techniques avec DEVS/HLA

DEVS/HLA agit ainsi comme un bus logiciel de haut niveau. La figure ci-dessous donne un exemple de fédération HLA, basée sur DEVS/HLA, pour la modélisation de systèmes de communication par satellites pour l'US Navy :

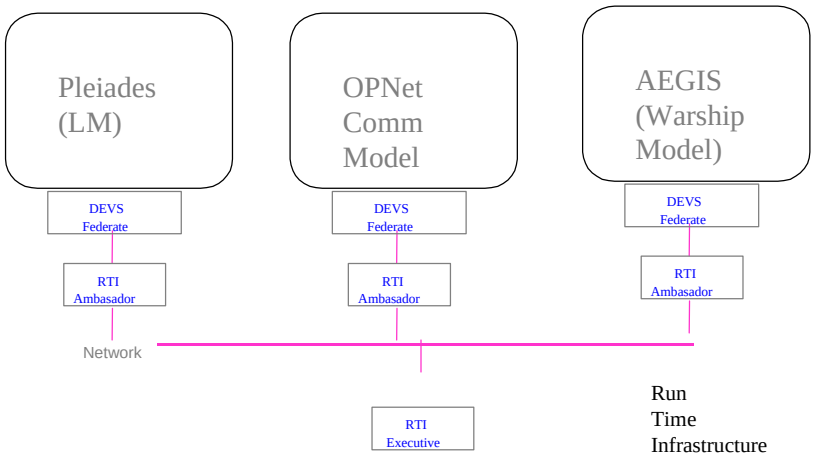


Figure 102: Exemple de fédération basée sur DEVS/HLA

Pleiades est un environnement de simulation basé sur DEVS réalisé par Lockheed Martin pour la modélisation d'interactions spatiales entre plusieurs plates-formes.

OPNET est un environnement de modélisation de réseaux de communication. Il est utilisé dans cette fédération pour modéliser un système de communication entre satellites.

AEGIS est une simulation de navire de combat.

DEVS, outre ses qualités intrinsèque, bénéficie de la notoriété de son principal concepteur, Bernard Ziegler. Mais il existe bien d'autres formalismes et structures d'accueil, dont certaines ont également des capacités de distribution.

En France, la structure d'accueil ESCADRE a connu un certain succès dans le domaine des simulations pour les études technico-opérationnelles. Nous allons examiner en détail ce produit dans la section suivante.

8.3 La structure d'accueil ESCADRE

8.3.1 Introduction

Afin de mieux comprendre ce qu'est une structure d'accueil, nous allons étudier de façon plus détaillée un exemple : ESCADRE[85].

ESCADRE est un produit destiné à supporter le cycle complet de vie d'une simulation (conception, codage, exécution). Il est disponible gratuitement, sur demande, avec son code source, sous une licence de type GPL[86]. Il fonctionne sur de nombreuses plates-formes (Windows NT, Solaris, Linux, AIX, IRIX, DEC/VMS[87], etc.).

Il est actuellement utilisé dans plusieurs dizaines de simulations, au ministère de la défense français, mais également chez des industriels et dans des universités. ESCADRE a été utilisé en Allemagne, évalué au Royaume-Uni et aux USA et une licence a été accordée à Singapour.

Un club d'utilisateurs ESCADRE permet de faciliter les interactions entre les développeurs et les utilisateurs du produit[88].

Cette section n'a pas pour objet de décrire exhaustivement ESCADRE (voir [SAUT01] pour cela), mais de donner un aperçu de ses concepts et du processus de modélisation qu'il préconise.

8.3.2 Historique

ESCADRE a été conçu en 1987 par le Centre d'Analyse de Défense (CAD) de la DGA, afin de développer et exploiter des simulations à des fins d'études techniques, technico-opérationnelles et opérationnelles, ainsi que, plus généralement, des simulations numériques constructives non temps réel.

La nécessité d'ESCADRE est partie notamment du constat suivant : 30 à 60% du code d'une simulation d'étude assure seulement des services élémentaires (interface graphique, statistiques…), et plus de la moitié du code restant modélise l'environnement naturel et tactique du système étudié.

De là vint l'idée de factoriser les services communs à toutes les simulations au sein d'une structure d'accueil et de mettre en place une méthodologie et une organisation pour permettre la capitalisation des modèles développés, et leur réutilisation d'une simulation à l'autre.

---

[85] Environnement de Simulation et de Conception orientée objet en Ada95 pour le Développement et la Réutilisation des Études.

[86] GNU Public Licence. Il s'agit de la licence utilisé pour des logiciels « open source » tels que EMACS ou le noyau LINUX.

[87] Le support de la plate-forme DEC/VMS a été arrêtée à partir de la version 4.6 d'ESCADRE.

[88] Le site web de ce club peut être trouvé à l'adresse suivante : http://escadre.cad.etca.fr:1815/

Les particularités des simulations technico-opérationnelles du CAD étaient les suivantes :

- Spécifications parfois imprécises et non figées.
- Nécessité de développement rapide (résultats attendus parfois sous quelques mois).

De plus, ont été ajoutées les contraintes suivantes :

- Les développeurs d'applications doivent se concentrer sur leur métier d'expertise (la modélisation de systèmes d'armes ou de sous-systèmes), et non sur des tâches d'informaticien (par exemple développer une interface graphique, choisir la position et la couleur d'un bouton…).
- Séparer les outils des modèles, afin non seulement de permettre une meilleure réutilisation du code développé, mais aussi de réduire les problèmes de sécurité (les modèles de systèmes sont classifiés, mais pas les outils génériques).
- Standardiser les outils et les interfaces.

Après examens des différentes options (en 1987), la voie d'une structure d'accueil en Ada83 et d'une méthodologie orientée objets dérivée de la méthode de Grady Booch a été choisie (implémentée toutefois à l'aide dans langage non-orienté objets, jusqu'à la version 5.0 qui utilisera pleinement les capacités objets d'Ada95).

ESCADRE a depuis beaucoup évolué :

- V1 (1987) : Évaluation de missiles antiradar (DEC / VMS). Première application (simulation de mission SEAD).
- V2 (1988): Nouvelle gestion du temps, traitement des données et exploitations statistiques (5 applications tournent).
- V3 (1991): Nouvelles plates-formes (Unix), affichage graphique en ligne (X-Window) et possibilité de rejeu d'une exécution.
- V4 (1994) : Améliorations diverses, passage au multi plates-formes Unix et DEC/VMS, IHM graphique basée sur Motif. Déjà, quelques dizaines d'applications basées sur ESCADRE tournent.
- V4.3 (1996) capacité multi-joueur (possibilité de créer des wargames), joystick virtuel pour le pilotage des acteurs mobiles
- V4.5 (1997) : Portage en Ada95 sans altération de l'API.
- V4.6 (1999) : Portage sur Windows NT, réorganisation du code des couches basses, industrialisation du produit.
- V5.0 (1999-2000) : Passage à la programmation orientée objets. Nouvelle API et révision de la méthode pour supporter les nouveaux mécanismes de spécialisation et de sous-composant.
- V 5.1 (2000) : Intégration du standard HLA pour interopérer avec d'autres simulations ou outils par simple ajout de service.

8.3.3 Architecture

ESCADRE est basé sur une architecture en couches. Notamment, une couche de services de base (tels que l'accès aux fonctions du système d'exploitation) et une couche graphique de haut niveau garantissent une excellente portabilité des applications : une simulation ESCADRE développée sous Unix peut être portée sous Windows NT par simple recompilation du code applicatif.

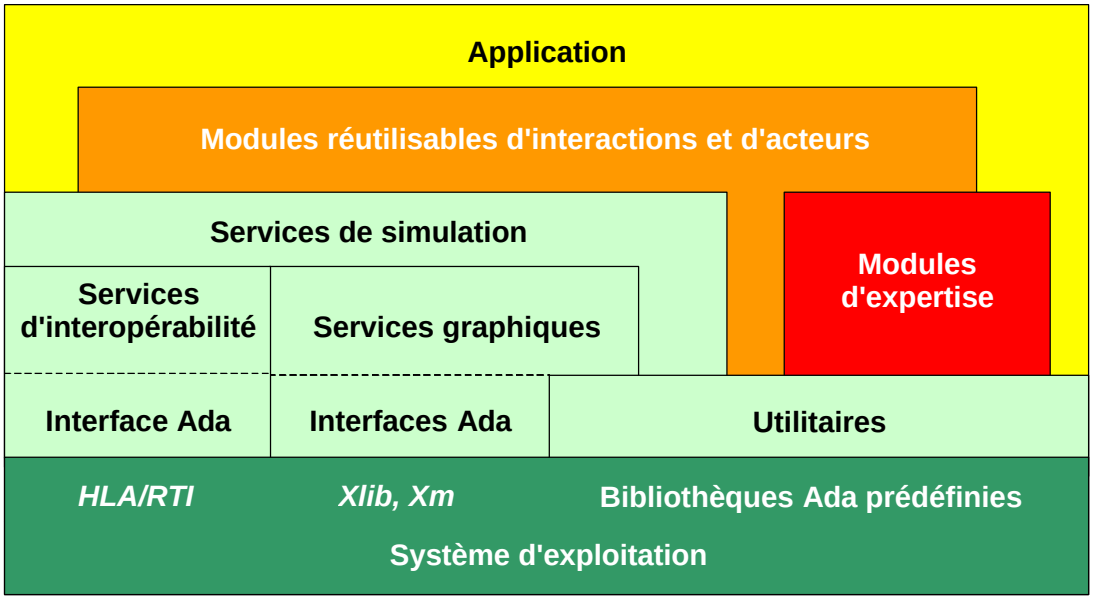


Figure 103: Structure d'ESCADRE

8.3.3.1 Interface de programmation (API)

ESCADRE comporte un ensemble de services, accessibles à partir d'une API Ada95. Ces services sont regroupés en 4 catégories :

- Services de base : accès au système, entrées-sorties du haut niveau, aide à la mise au point, manipulation de texte et données, fonctions mathématiques...
- Services graphiques[89] : boîtes de dialogues, affichage de cartes, animation, tracé de courbes, visualisation de données, joystick virtuel...
- Services d'interopérabilité fondés sur la HLA/RTI distribué par le DMSO équipé d'un binding Ada95[90].
- Services de simulation : moteur de la simulation (gestion du temps, des acteurs, des interactions...), visualisation des acteurs, recueil de données, interopérabilité HLA...

[89] Ces services sont en principe encapsulés dans la couche haute de simulation et sont appelés automatiquement par l'infrastructure d'exécution d'ESCADRE, bien qu'il soit possible pour des experts de les utiliser directement.
[90] Ce « binding » Ada95 a d'ailleurs été conçu dans le cadre du développement de la version HLA d'ESCADRE, puis adopté par la suite par le DMSO.

8.3.3.2 Interface utilisateur d'exécution

- Supervision de l'exécution des répliques (pas à pas, jusqu'à une date, automatique)
- Introspection de l'applications (hiérarchie de composition et spécialisation d'acteurs, Interaction aveugles, interopérabilité HLA, …)
- Commande interactives sur les objet de la simulation.

8.3.3.3 Outils

ESCADRE est livré avec des outils destinés à effectuer certaines tâches de préparation et d'exploitation des exécutions de simulations :

- Édition de cartes et de données techniques opérationnelles.
- Analyse statistique de résultats.
- Analyse de journaux.
- Conversions de formats de données.
- Rejeu graphique d'une exécution (ou réplique).
- ...

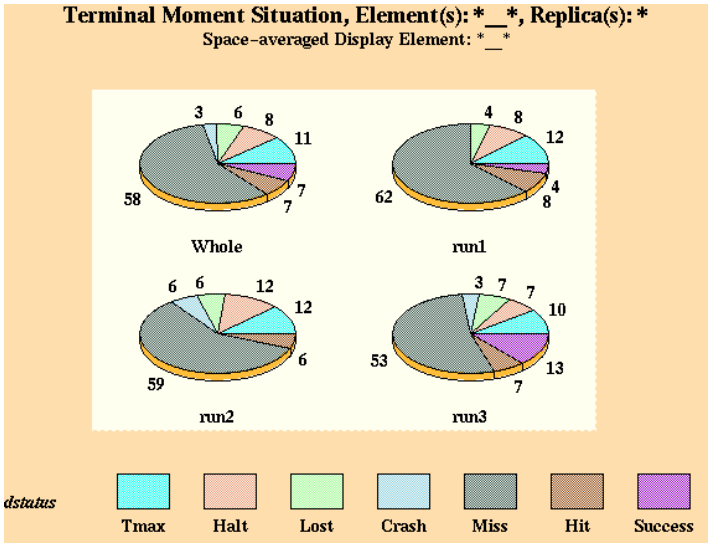


Figure 104: outil d'analyse statistique d'une série de répliques

8.3.4 Concepts de base

8.3.4.1 Terminologie ESCADRE

Un **acteur ESCADRE** est une classe d'objets de la simulation. Il peut être défini :

- Par spécialisation d'un acteur abstrait prédéfini (unique cas proposé par l'ancienne méthode). C'est le cas des acteurs *Battery*, *Radar* et *Patrol* dans l'exemple Air.
- Par spécialisation d'un acteur abstrait (non enregistré auprès d'ESCADRE) préalablement défini au niveau d'une famille d'application. C'est le cas des acteurs *SA_Missile*, *Aircraft* et *Bomb* qui spécialise l'acteur abstrait *Movable* (équipé des fonctions abstraites donnant position et vitesse).
- Par spécialisation d'un acteur concret (enregistré auprès d'ESCADRE) préalablement défini. C'est le cas de l'acteur *Bomber* qui spécialise l'acteur concret *Aircraft* et qui hérite de ses rôles vis à vis des propriétés.

La bibliothèque ESCADRE fournit :

- Un acteur racine abstrait avec ses accesseurs (*name_of*, *team_of*, *status_of*, *signal_of*, *parent_of*, *children_of*) et ses primitives (write_data, *create*, *start*, *suspend*, *resume*, *stop*, *delete*, *detach*, *reparent*).
- Un patron de propriété (interaction aveugle) instantiable sur la paire de type Question / Réponse. Ce patron définit la propriété abstraite avec sa primitive de service recevant la question et retournant la réponse

Les **sous-composants** (acteur léger) sont des agrégations (par valeur) d'éléments au profit d'un acteur container. Leurs données techniques et d'état sont gérées par l'acteur container. Ils peuvent définir des Événements et Dynamiques en s'identifiant par le container et collaborent au fonctionnement de ce dernier. Ils peuvent être spécifiques à l'acteur qui les utilise ou génériques (réutilisables).

Les **technologies** permettent, pour une spécification donnée d'un acteur, de définir des variantes (de fidélité ou de comportement) qui se distinguent :

- par des variantes de données techniques (vitesse de croisière, accélérations maxi…)
- par des variantes de code non extensibles (discriminant), par exemple : missile 3 ou 6 degrés de liberté (fidélité), radar à balayage mécanique ou électronique (comportement).
- par des variantes de code extensibles (spécialisation privée), par exemple : armement générique ou spécifique(s) (fidélité), missile balistique, tournoyant, manœuvrant, etc. (comportement).

Concrètement, les technologies sont traitées comme des variantes statiques d'un acteur assimilables à un second niveau de spécialisation interne. Ainsi, on peut avoir un acteur de type *Bomber* déclinable en plusieurs technologies (Tornado, F16, Mirage2000) qui indiqueront les caractéristiques de chaque modèle de bombardier (capacité d'emport, vitesse de croisière, accélérations, résistance aux éclats…). Les paramètres des différentes technologies sont contenus dans des fichiers texte tels que celui-ci :

```
Bomber                              -- nom de l'acteur lu par RTI

Base                                -- nom de la technologie lu par le RTI
   Rcs       =>  10                 -- champs lus par Airplane.Def.Read_Tno
   Speed_Km/H=> 800
   Tmax_H    =>   6
   Lethal    =>  50
   Step      =>  20
   Bomb_Tno  =>  Complex_Default    -- champs lus par Airplane.Bomber.Def.Read_Tno
   Bomb_Max  =>   8
```

La **filiation** entre acteurs permet à un acteur parent de faire collaborer un acteur enfant à la réalisation de sa mission. Les interactions se font dans les deux sens selon un protocole établi par l'acteur enfant (**interactions cognitives**). Ceci garantit la réutilisabilité de toutes sous hiérarchies de filiation. L'acteur enfant conserve une certaine autonomie et peut d'ailleurs changer de parent où devenir autonome. Sa durée de vie n'est pas nécessairement contrainte par celle de son parent.

La **propriété** définit un protocole d'échange (données d'entrées et de sorties) entre un acteur **client** initiateur de l'interaction et un acteur **serveur** qui s'est déclaré capable de subir l'interaction. Elle permet une interaction entre acteurs sans imposer la visibilité du client initiateur sur le serveur (d'où la dénomination d'**interaction aveugle**). De ce fait, la propriété assure le découplage (et donc la substitution) des acteurs qui interagissent selon ce mode (qui modélise souvent les interactions physiques du monde réel).

8.3.4.2 Cas pratique : l'exemple Air

ESCADRE est livré avec des exemples Air (simulation d'attaque air-sol et sol-air), Terre (combat terrestre chars et hélicoptères) et Mer (patrouilles de sous-marins).

L'exemple Air simule une mission SEAD (suppression des défenses aériennes ennemies) : une patrouille d'avions attaquent des radars qui peuvent se défendre à l'aide de batteries de missiles sol-air (voir Figure 105).

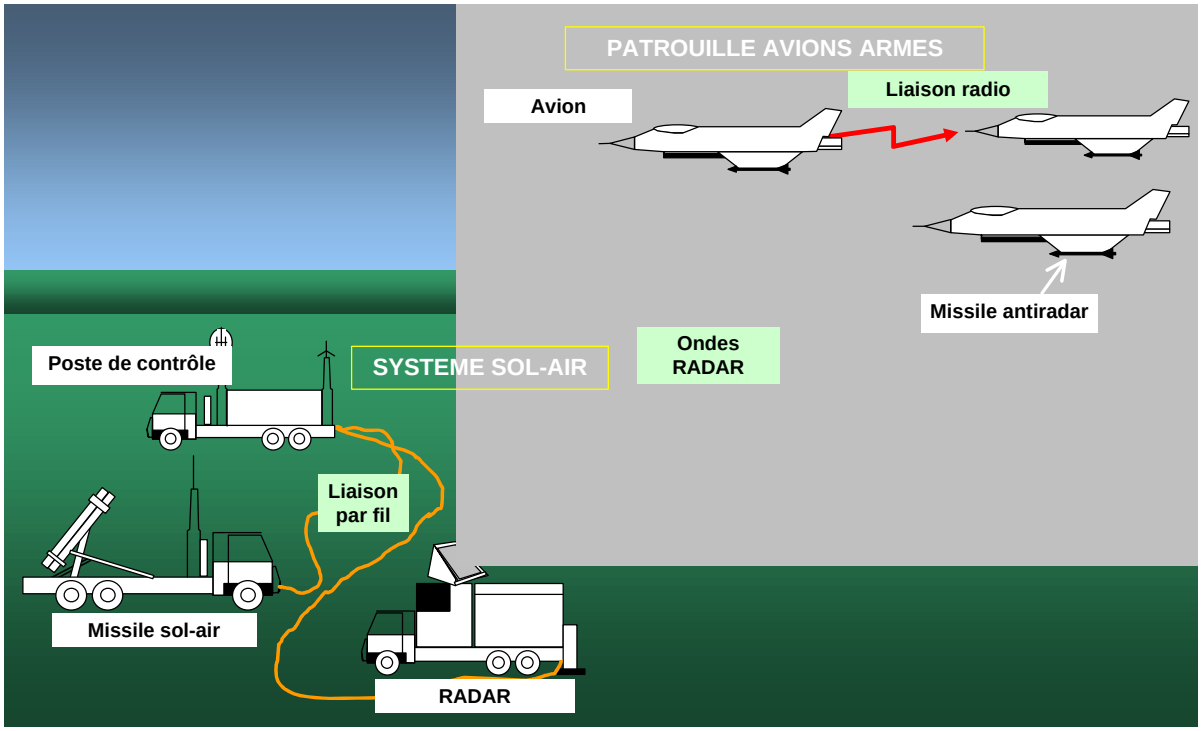


Figure 105: Exemple AIR (mission SEAD)

La première étape consiste à décrire textuellement l'environnement et la mission que l'on souhaite modéliser :

– Chaque patrouille de l'attaque *coordonne* un groupe d'avions *progressant* sur une route. Chaque avion *porte* des missiles antiradar et, *arrivé* à proximité de son objectif, *écoute* les émissions radar afin de *localiser* sa cible puis de *larguer* un missile antiradar. Chaque missile se *dirige* vers sa cible *guidé* par son autodirecteur passif afin de la *détruire*.

– Chaque système sol air de la défense se *compose* d'un radar qui *détecte* les cibles adverses, *élabore* des pistes et peut *suspendre* son émission, d'un poste de contrôle qui *recueille* et *classifie* ces pistes, puis *tire* des missiles sol air afin de les *détruire*.

Dans cette description textuelle, nous avons fait apparaître les acteurs et les *interactions*.

Dès lors, on peut en déduire aisément les acteurs de la simulation et leurs relations :

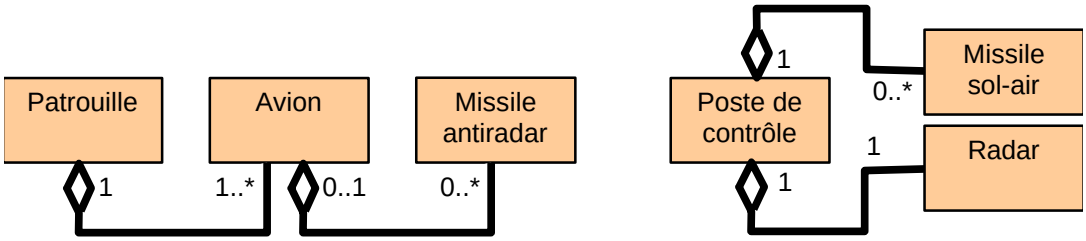


Figure 106: Diagramme des acteurs

Ainsi que les interactions entre ces acteurs :

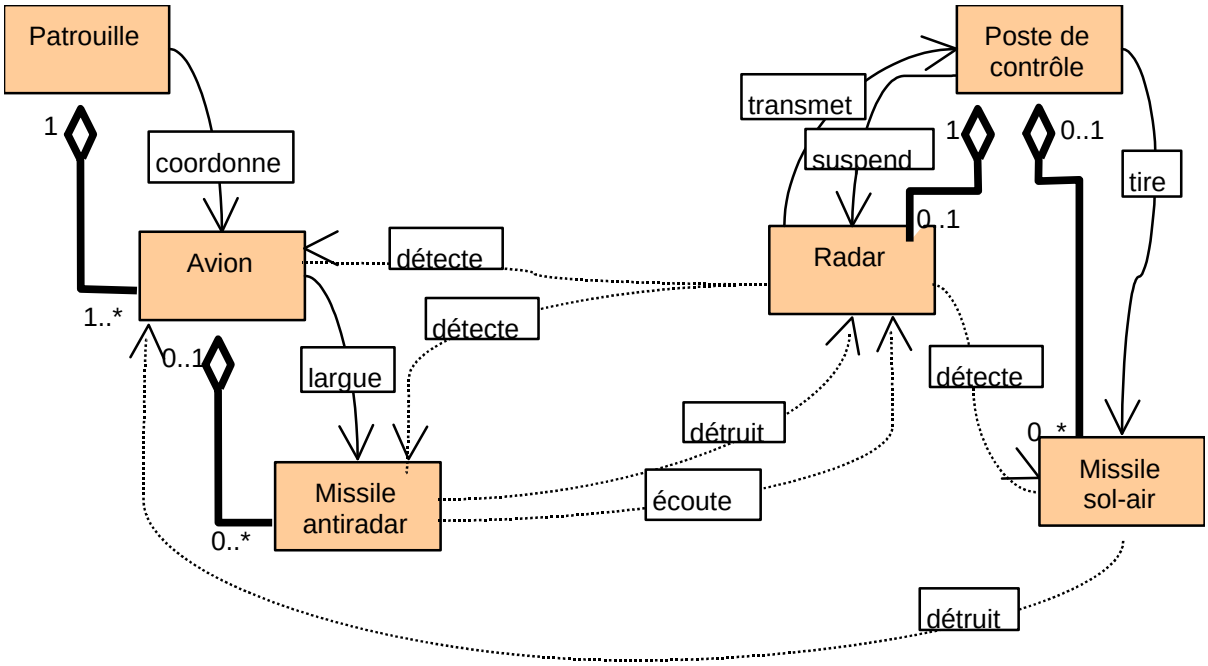


Figure 107: Diagramme des interactions entre acteurs

Ces diagrammes respectent le formalisme UML, choisi par la méthodologie ESCADRE (à partir de la version 5.0).

Outre les acteurs et les interactions, il faut définir les propriétés qui seront modélisées. Dans notre exemple, nous avons besoin de trois propriétés seulement :

– Réflexion radar : chaque acteur de type avion est serveur de cette propriété, qui permet de savoir quelle est la proportion de l'onde radar incidente qui est réfléchie par l'avion. Typiquement, le client est le radar émetteur. Notons que les missiles ne sont pas serveur de cette propriété, car, dans l'exemple Air, les radars ne tiennent pas compte des missiles en vol, et donc n'ont pas l'utilité de cette propriété pour les missiles.

– Émission radar : cette propriété est utilisée par le missile anti-radar pour savoir si, en fonction de sa bande d'écoute et de sa position, il peut détecter le radar serveur de cette propriété qu'il interroge.

– Vulnérabilité : cette propriété permet d'évaluer les dégâts occasionnés à un acteur par l'explosion d'une charge militaire (i.e. l'ogive d'un missile sol-air). On voit que les radars, les avions, les missiles sol-air et les missiles air-sol sont tous serveurs et donc supposés susceptibles d'être endommagés ou détruits.

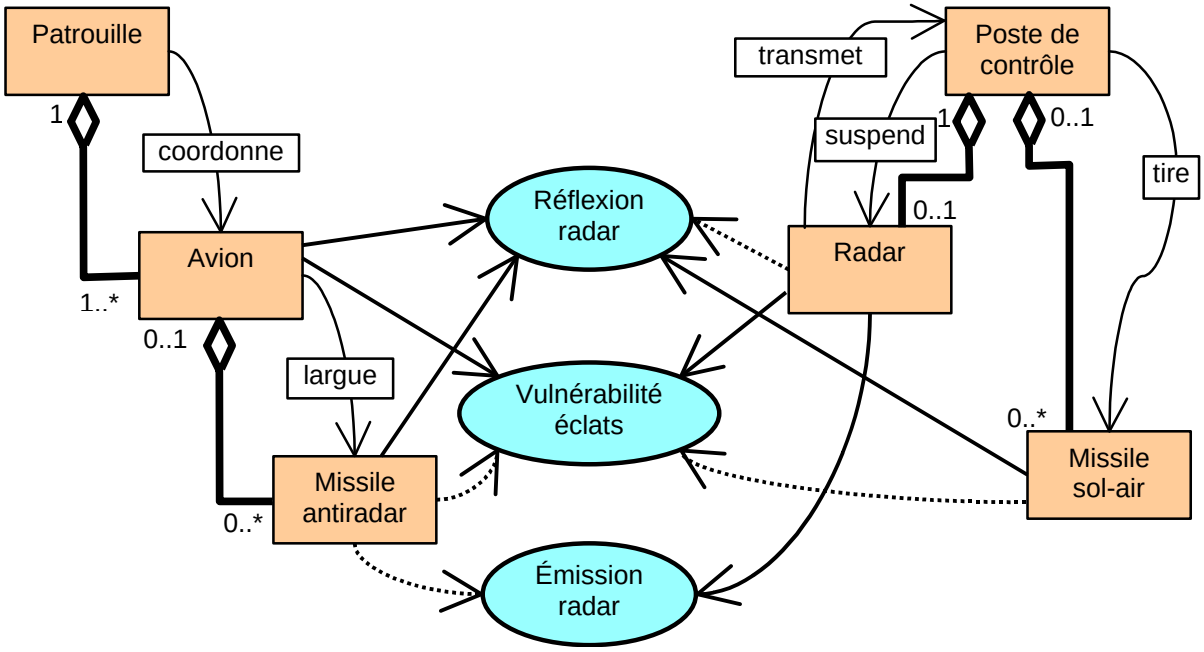


Figure 108: Diagramme des acteurs, interactions, propriétés

La Figure 108 donne la topologie complète de l'application.

Mais l'analyse du système ne s'arrête pas là, car chacun des composants de ce diagramme peut lui-même faire l'objet d'une analyse et d'une décomposition. Ainsi, on remarque qu'un missile comporte des sous-composants : autodirecteur, propulseur, charge militaire… La Figure 109 donne un exemple de décomposition.

À chaque Acteur, sous-composant, propriété, etc., correspondra par la suite un paquetage Ada95, donc une spécification et une implémentation (« body »).

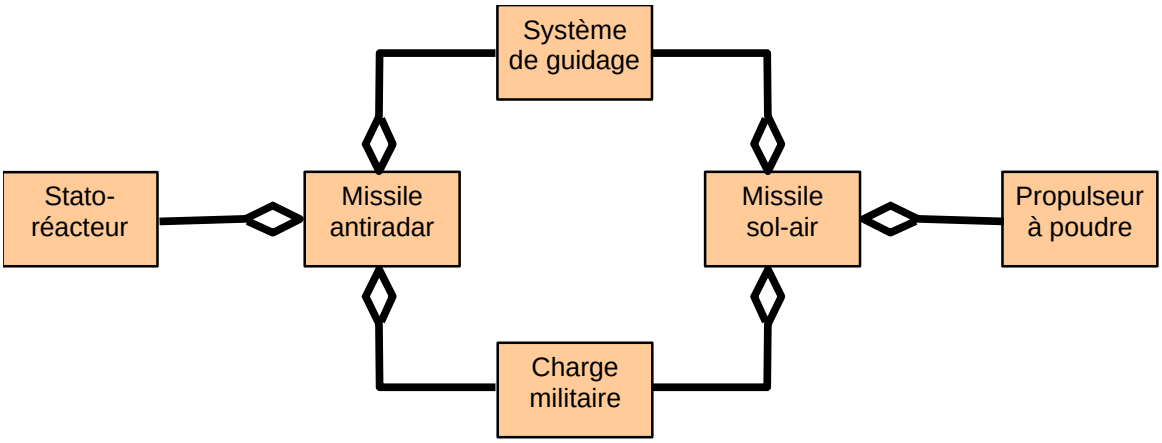


Figure 109: Sous-composants de l'acteur missile

Cette phase de conception des modèles nécessaires terminée, il reste à traduire ces modèles conceptuels en spécifications Ada95 puis à les implémenter.

Par exemple, voici la spécification du paquetage du sous-composant *Guidance* (guidage du missile sol-air) :

```
private package Sa_Missile.Guidance is
   type Rec_Tno is record ... end record;
   procedure Read_Tno  (Title: in String; Tno:    out Rec_Tno);
   procedure Write_Tno (Title: in String; Tno: in     Rec_Tno);

   type Rec_Data is private;
   type Ref_Data is access all Rec_Data;
   procedure Write_Data (Title: in String; Data: in Rec_Data);
end Sa_Missile.Guidance;
with Root.Movable.Def;
generic
   type Owner_Data is new Root.Movable.Def.Rec_Data with private;
   with function Tno_Of       (Owner: Owner_Data'Class) return Rec_Tno;
   with function Data_Of      (Owner: Owner_Data'Class) return Ref_Data;
   with function Target_Pos_Of(Owner: Owner_Data'Class) return Position_3d;
   with function Target_Vel_Of(Owner: Owner_Data'Class) return Velocity_3d;
package Sa_Missile.Guidance.Api is
   procedure Initialize  (Owner     : in out Owner_Data'Class;
                          Target_Pos: in     Position_3d;
                          Pos       : in     Position_3d;
                          Vel       :    out Velocity_3d);
   function Acc_Commanded(Owner: Owner_Data'Class) return Acceleration_3d;
end Sa_Missile.Guidance.Api;
```

Voici l'implémentation de la fonction réalisant la mise à feu d'un missile :

```
procedure Process_Launch (Data: in out Rec_Data'Class;
                          Target: in Object; Cmpt: in Component_Id)
   is
      Tno   : constant Aa.Ref_Tno := Aa.Tno_Of(Data);
      Dirvel: Velocity_3d;
   begin
      -- sous composants
      for Cmpt in 1..Tno.Nbrock loop
         Rocket.Initialize (Data, Cmpt);
      end loop;
      Seeker.Initialize (Data, Target);
      Guidance.Initialize(Data, Seeker.Target_Pos_Of(Data), Data.Illum_Pos,
Dirvel);
      Airframe.Initialize(Data, Data.Illum_Pos, Dirvel);
   ...
   end Process_Launch;
   ...
end Sa_Missile.Def;
```

Voici le résultat obtenu (application de démonstration Air) :

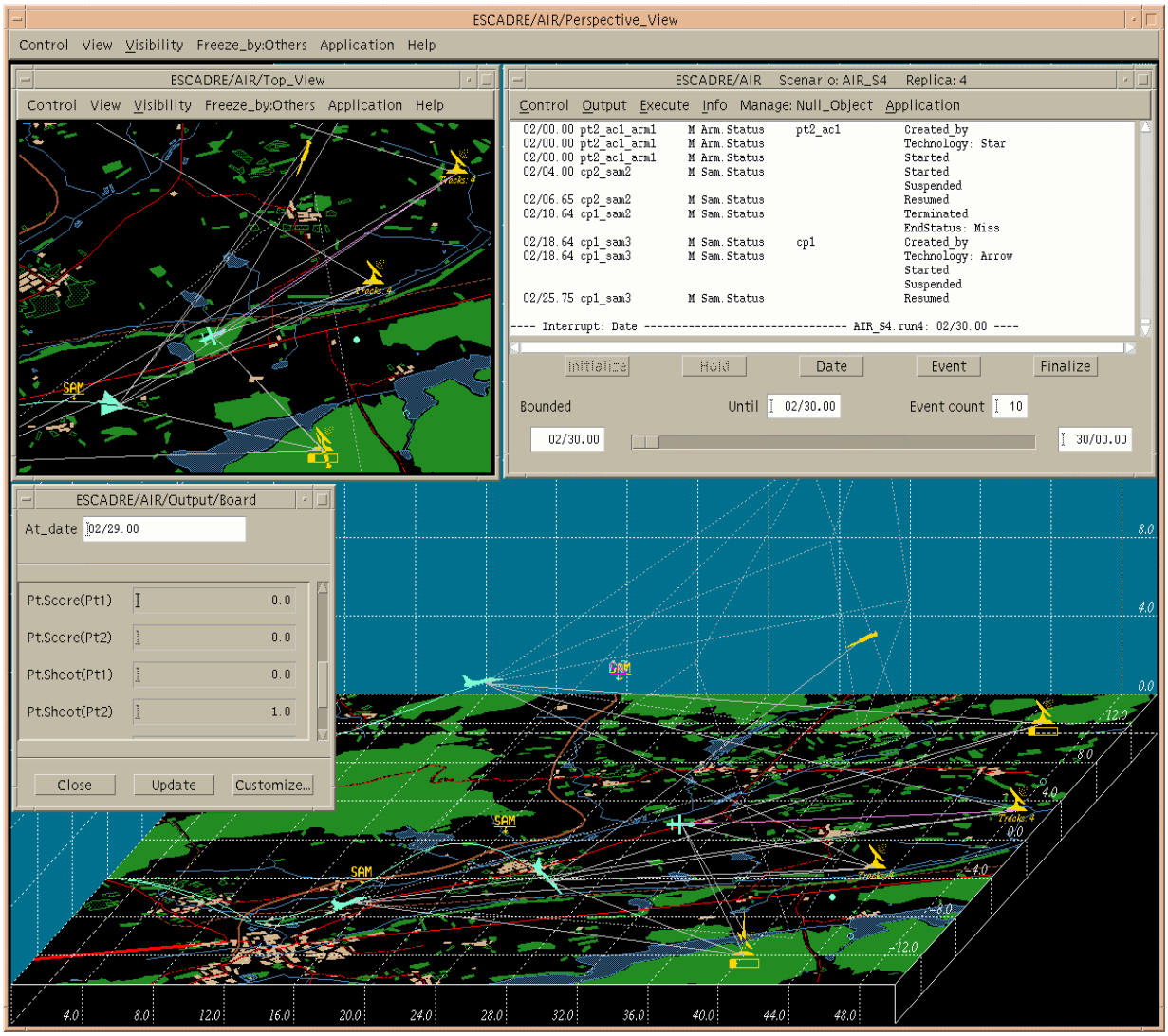


Figure 110: Exemple AIR d'ESCADRE

On note tout de suite la richesse de l'interface graphique et des menus, alors qu'à aucune moment nous n'avons travaillé sur autre chose que la modélisation du système : tout l'environnement d'exécution nécessaire est pris en charge par la structure d'accueil ESCADRE et ses outils. On comprend aisément sur cet exemple l'intérêt d'un tel produit.

Mais il faut garder à l'esprit que la factorisation de fonctions communes et la mise en œuvre d'une méthodologie de conception ne suffisent pas, encore faut-il avoir une politique et une organisation permettant une capitalisation efficace des modèles…

8.4 La capitalisation de modèles

L'utilisation d'une structure d'accueil permet donc de se focaliser sur le développement des modèles. Mais la démarche peut (et doit) aller encore plus loin dans la rationalisation des développements.

En effet, le développement d'un modèle reste parfois complexe, et coûteux, et nécessite une validation rigoureuse. Or, particulièrement dans la simulation à des fins d'études, on retrouve souvent d'une simulation à l'autre les mêmes modèles, éventuellement avec quelques variations (changement du type de radar, etc.). Il serait donc souhaitable de ne pas redévelopper plusieurs fois ce qui existe déjà.

D'où l'idée de réutiliser les modèles d'une simulation à l'autre, évidemment en remettant à chaque fois en cause la validation de ces modèles, afin de s'assurer que l'on reste toujours dans le domaine de validité du modèle, ou que les modifications apportées n'altèrent pas la validité de ce dernier.

La réutilisation des modèles ne peut être rentable, efficace et correcte que si un certain nombre de précautions sont prises :

- Chaque modèle doit être convenablement documenté : architecture, entités et objets manipulés (ceci peut être par exemple l'objet d'un SOM HLA), hypothèses utilisées, domaine de validité, etc.
- L'accès à la bibliothèque de modèles doit se faire à partir d'un accès unique (par exemple un moteur de recherche), même si leur stockage physique est distribué.
- Les modèles devront être disponibles sous forme de code source, pour permettre leur modification ou leur portage par un tiers.
- Si les modèles utilisent des structures d'accueil différentes, on gagnera à ce que celles-ci soient interopérables (par exemple par HLA ou CORBA).
- Une politique forte et réelle de capitalisation et de réutilisation des modèles doit être mise en place, c'est-à-dire qu'il faut que la hiérarchie soit impliquée à un haut niveau.
- …

Un processus de capitalisation, coûteux au départ, permet la constitution d'une librairie de composants qui, à terme, permettra un gain de temps et donc d'argent très important.

Les américains ont particulièrement mis l'accent avec HLA sur ce concept qu'ils appellent « component-based simulation ».

En outre, un effet secondaire de la capitalisation est la responsabilisation plus forte des développeurs : dans la mesure où leur code pourra être relu par leurs pairs et réutilisé, ils ont tendance à améliorer la qualité de leur code et de leur documentation. Cet effet est à l'origine de la qualité remarquable de nombreux logiciels libres, qui fournissent de nombreux exemple de réutilisation de composants à large échelle.

9. LES ENVIRONNEMENTS SYNTHETIQUES

9.1 Définition

Un **environnement synthétique** est un ensemble de moyens informatiques (logiciels, matériels, réseaux, données) mis en œuvre afin de recréer un **monde virtuel**.

Typiquement, un environnement synthétique comporte les trois composantes suivantes :

- une simulation ou un simulateur avec un modèle des fonctions auxquelles on s'intéresse ;
- un environnement naturel (terrain, atmosphère, océan…) ;
- un environnement tactique (ennemis…)

Notons que l'environnement tactique peut être fourni par un opérateur, par une autre simulation. Les CGF (voir p.240) permettent notamment à des simulations ou des simulateurs de disposer d'un environnement tactique élaboré.

Nous allons passer en revue dans ce chapitre les technologies utilisées pour réaliser ces environnements synthétiques : réalité virtuelle pour l'immersion sensitive, la représentation de l'environnement naturel, la simulation distribuée pour gérer la complexité, CGF pour simuler les comportements humains.

9.2 Réalité virtuelle

9.2.1 Définition

Le terme « réalité virtuelle » est employé avec une grande diversité de sens. Elle peut désigner un environnement synthétique, défini précédemment, ou une interface homme-machine (IHM, *GUI* en anglais) permettant à un opérateur de visualiser, de manipuler et d'interagir avec des données ou un environnement complexes.

Dans quelle mesure un environnement synthétique est-il alors un environnement de réalité virtuelle ?

Un jeu de critères généralement admis est celui du « triangle de Burdea » (voir [BURD93] et Figure 111) : un environnement de réalité virtuel doit satisfaire à la « règle des 3I » (**Immersion**, **Interactivité**, **Imagination**), dont nous allons détailler les composantes dans les paragraphes qui suivent.

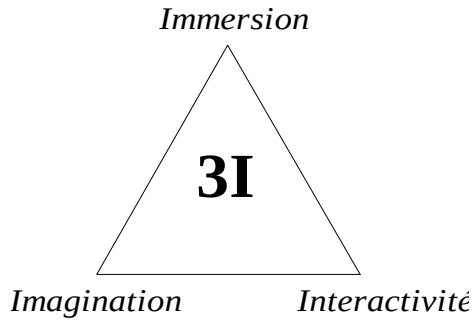


Figure 111: Triangle de Burdea (règle des 3I)

David Zeltzer, chercheur au MIT[91], a imaginé des critères d'évaluation d'un système de réalité virtuelle sous la forme d'un cube de 1 unité de côté. Chaque axe mesure la qualité du système dans les trois critères suivants :

- **Autonomie** : capacité du système à réagir aux stimuli. La valeur 0 caractérise un système purement passif, se contentant d'afficher du texte ou des images, tel que le cinéma. La valeur 1 indiquera que le système est capable de réagir à toutes les sollicitations de l'utilisateur (par exemple répondre aux actions sur un joystick).
- **Interaction** : mesure l'influence de l'utilisateur sur les paramètres du système. La valeur 0 indique un système où l'utilisateur ne peut intervenir sur le déroulement des opérations, qui sont pré-déterminées. Au niveau 1, il peut en revanche modifier en temps réel le comportement du système.
- **Présence** : mesure le nombre et la variété des stimuli (visuels, auditifs, etc.) envoyés à l'opérateur par le système.

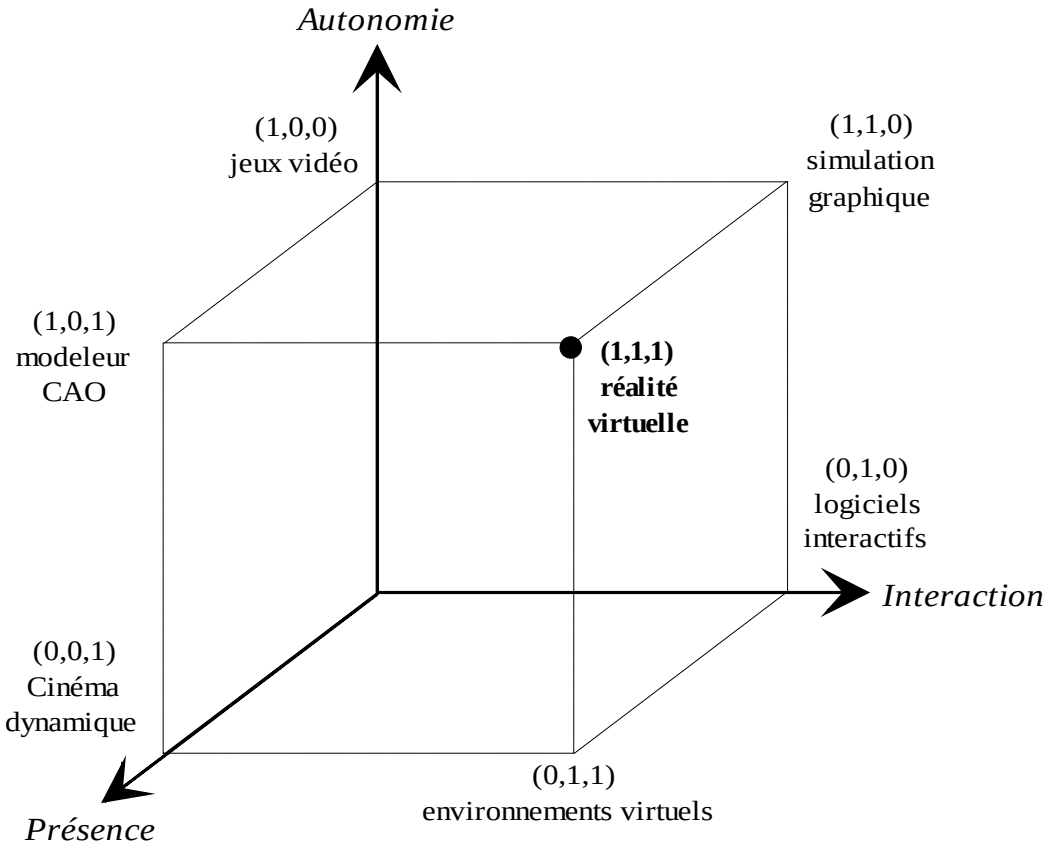


Figure 112: Cube AIP de Zeltzer (MIT)

(d'après [JOLI95])

[91] Massachusetts Institute of Technology, prestigieuse université américaine, très en pointe sur les questions d'intelligence artificielle et d'interface homme-machine.

Un système de réalité virtuelle parfait serait donc aux coordonnées (1,1,1). Actuellement, il n'y a pratiquement que les simulateurs de vols haut de gamme (ceux dits *full flight*) qui correspondent à cette définition. Pour ma part, je trouve ce cube intéressant, mais restrictif, et je lui préfère le triangle de Burdea (en plus, c'est plus facile à dessiner !).

9.2.2 Immersion

L'opérateur doit avoir l'impression qu'il est physiquement plongé dans l'univers virtuel. Typiquement, ceci est réalisé en s'assurant que tout son champ visuel est occupé par l'environnement virtuel, dans l'idéal à 360°.

Une simulation pilotée avec une cabine est un exemple typique de système de réalité virtuelle. En effet, comme nous l'avons vu au chapitre 3, les cabines de simulation pilotées fournissent un champ visuel généralement supérieur à 180°, voir parfois, avec les sphères de projection, à 360°. Partout où le pilote porte son regard, il voit donc les images issues du système de simulation.

Un wargame, par contraste, s'il utilise un environnement synthétique sophistiqué, ne constitue pas un système de réalité virtuelle, car il n'est pas immersif, entre autres.

Cette notion d'immersion est bien entendu très subjective. Ainsi, le jeu Doom sur micro-ordinateur (voir page 225) n'a qu'un graphisme médiocre, souvent en « fausse 3D » (utilisation d'un jeu de formes bitmap 2D dans un décor en 3D) et d'une résolution réduite, mais cela a permis une excellente fluidité du jeu sur la plupart des plates-formes. En effet, les auteurs se sont aperçus que le plus important pour immerger le joueur était le temps de réaction de l'image et sa fluidité, plus que son niveau de détails, l'imagination faisant le reste (nous y reviendrons).

L'immersion s'obtient donc à moindre coût en **abusant nos sens**. Dans les années 50, le producteur de cinéma Morton Heilig lança le système de cinéma expérimental Sensorama, qui ajoutait à l'image et au son du vent, de la chaleur, des odeurs, des vibrations. Il estimait en effet que les informations traitées par les spectateurs se répartissaient ainsi :

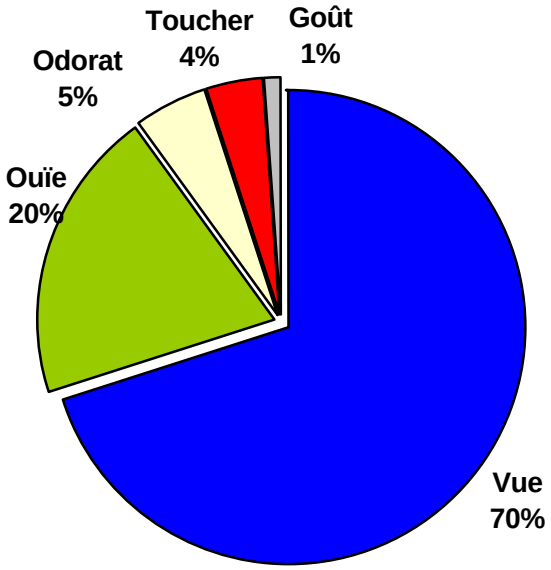


Figure 113: Répartition des informations sensitives

Les concepteurs de simulateurs de vol l'ont bien compris. Certains modèles destinés à la formation de pilotes d'avions commerciaux peuvent générer des odeurs (brûlé…) afin d'augmenter le réalisme et à donner aux pilotages des indices olfactifs sur l'origine d'une panne simulée. Certains parcs d'attractions ont également bien compris le gain en réalisme apporté par la stimulation de sens multiples. Ainsi, le spectacle Terminator 2000 des parcs Universal Studios (Floride et Californie) est remarquable de ce point de vue[92] : une partie des sièges est mobile, et des odeurs, fumées et brume humide viennent surprendre le spectateur. Dans une autre attraction, de Disney cette fois, un système placé sous les sièges vient caresser les pieds des spectateurs, au moment où, dans le spectacle, une nuée de souris est censée se répandre dans la salle. C'est rustique et peu coûteux à mettre en œuvre, mais l'effet est garanti !

Voyons maintenant quels types de matériels et de logiciels sont mis en œuvre pour permettre une telle immersion.

9.2.2.1 Systèmes de visualisation

Nous avons vu que la principale source d'information sensitive était la vue. Il est donc naturel que les efforts des développeurs se soient portés sur les systèmes de visualisation graphique, qui utilisent une grande variété de technologies pour réaliser l'immersion.

L'image est calculée par un ordinateur, équipé d'une **carte graphique** 3D performante, voire **d'un générateur d'images**ou d'un calculateur graphique dédié pour le très haut de gamme.

[92] Du point de vue immersif, bien entendu, puisque qu'il n'y a aucune interactivité dans le spectacle, l'observateur étant totalement passif (sauf quand il hurle…).

Le système de visualisation le plus répandu est bien entendu l'écran vidéo, qu'il soit à **tube cathodique** (CRT) ou à **cristaux liquides** (LCD). Les CRT ont de meilleures performances en contraste, résolution, luminosité... De plus, ils sont maintenant bon marché. En revanche, ils sont lourds, encombrants et gourmands en énergie. Le succès des ordinateurs portables a permis celui des écrans plats LCD, qui restent néanmoins beaucoup plus onéreux[93], surtout pour les grandes surfaces, en raison d'un processus de fabrication complexe et d'une important proportion de matériel défectueux en fin de chaîne de fabrication.

Figure 114: Lunettes à obturation

Bien que la surface de l'écran soit relativement restreinte et plate, de nombreux logiciels arrivent à donner l'illusion d'immersion à l'utilisateur (nous avons déjà discuté de l'exemple du jeu DOOM). Certains logiciels et cartes graphiques savent également gérer une image stéréoscopique. L'utilisateur porte des lunettes dites « à obturation » (*shutter glasses*) dont les verres, comportant des cristaux liquides, peuvent devenir alternativement opaques ou transparents sous l'effet d'un courant électrique. L'astuce consiste à afficher sur l'écran, avec une fréquence d'au moins 30 Hz par œil (soit 60 Hz en tout) l'image destiné à l'œil droit pendant que l'œil gauche est masqué, et vice versa, donnant au cerveau l'illusion du relief[94]. Ce système peu onéreux (quelques milliers de francs) est très efficace, mais la taille de l'objet est bien évidemment fonction de celle de l'image.

Cette taille de l'image est d'ailleurs le principal enjeu des systèmes de visualisation en réalité virtuelle. En effet, pour immerger l'opérateur ou le spectateur, il faut remplir son champ de vision. Pour cela, deux principales solutions : soit on élargit l'image, soit on rapproche l'image de l'œil.

Élargir l'écran, c'est facile : on sait maintenant faire des écrans cathodiques de pratiquement 1 m de diagonale, des écrans à plasma plus grands encore, et les projecteurs vidéo peuvent donner des images de haute qualité de plusieurs mètres de large. C'est un créneau que met en avant ces derniers temps la société SGI, avec ses offres de *Reality Center*.

[93] Alors qu'on trouve des écrans de 17 pouces CRT de bonne qualité pour 2500F, il faut encore compter près de 8000F pour l'équivalent LCD (15 pouces), avec une résolution souvent moindre (tarifs été 2000).

[94] Il existe une version économique de ce procédé, en affichant simultanément une image rouge pour l'œil gauche et une bleu pour l'œil droit. L'utilisateur utilise des lunettes avec des filtres de couleur appropriée. Si de telles lunettes coûtent quelques francs, l'image est monochrome et le relief nettement moins évident. Une meilleure technique consiste à utilise des verres polarisants, dont la direction de polarisation diffère, mais il faut alors également polariser la lumière de l'image que l'on observe, ce qui n'est pas possible avec un écran CRT (mais ce système est très populaire dans les cinémas de certains parcs d'attractions).

Oui, mais voilà : ces systèmes sont encombrants et coûtent cher. Un écran à plasma de 1 m de diagonale vaut actuellement plus de 10 000 €, et un vidéoprojecteur correct coûte de 7 000 € (petit modèle à panneau LCD peu lumineux) à plus de 25 000 € (projecteur à CRT, dit « tritube »).

Figure 115: BARCO Reality Center à Paris

Autre solution : réduire l'écran pour réduire son coût et son encombrement, et le rapprocher de l'œil de l'utilisateur. C'est le principe des casques d'affichage (*Head-Mounted Display*, HMD). On place devant chaque œil un minuscule écran vidéo (CRT ou LCD, voir un exemple de conception Figure 116), couleur ou monochrome, fixé sur un casque que porte l'utilisateur. Ne voyant plus son environnement réel, l'utilisateur est entièrement immergé. Afin d'avoir une immersion sur 360°, un ou plusieurs capteurs déterminent la direction du regard (ou la position de la tête) de l'utilisateur et le logiciel adapte l'image en conséquence. L'utilisateur peut alors regarder dans toutes les directions. C'est actuellement la technique d'immersion la plus poussée.

De nombreuses sociétés fabriquent de tels dispositifs HMD. Le bas de gamme, équipé de moniteurs LCD couleurs, ne coûte guère plus de 500 €, avec évidemment une qualité d'affichage médiocre et un champ de vision limité (typiquement 40 à 70°). Le haut de gamme, à base de CRT, atteint des résolutions de 1280x1024 en couleurs, voire plus, avec un contraste et une luminosité très supérieure au LCD, et un champ de vision de 60 à 120°. En revanche, leur prix se chiffre en milliers, voire en dizaines de milliers d'euros, ce qui limite leur utilisation aux professionnels (ingénieurs, designers, architectes, militaires...). Ainsi, le désormais classique casque SimEye de Kaiser Electro-Optics, qui permet de voir le monde réel à travers l'affichage, qui s'effectue sur une surface semi-transparente, est facturé dans les 80 000 €. Sa résolution est de 1024x768, et son champ de vision varie de 60° à 100° suivant le réglage des deux afficheurs. Un tel casque est destiné à des applications de **réalité augmentée**, où l'on superpose des informations synthétiques à la vision du monde réel (par exemple, pour un pilote, un horizon artificiel ou un représentation du couloir d'approche de l'aéroport).

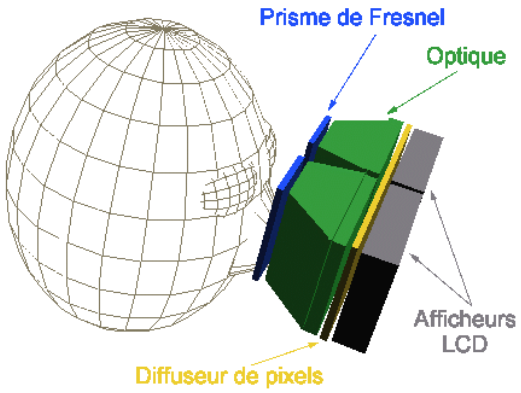


Figure 116: Principe d'un HMD (CRT)

Figure 117: HMD Datavisor (n-Vision)

Figure 118: HMD Datavisor-NVG (n-Vision)

Toutefois, ces casques se heurtent à un certain nombre de limitations :

- Leurs performances sont limitées (champ de vision, résolution...).
- Le fait de porter un HMD finit par être fatiguant pour l'utilisation en raison de son poids (particulièrement pour les modèles CRT).
- Le décalage, même à peine perceptible, entre les mouvements de la tête et la réponse du système (et donc de l'image) entraîne assez rapidement une désorientation du cerveau et des nausées. Ce phénomène, appelé « mal des simulateurs », est bien connu en simulation pilotée, et peut rendre l'utilisateur réellement malade. Toutefois, cet effet temps à se réduire au fur et à mesure qu'augmentent les performances des générateurs d'images.
- La plupart des modèles sont reliés à l'ordinateur par un assez gros câble (vidéo, alimentation...), ce qui n'est guère pratique.
- Enfin, l'immersion est une expérience purement personnelle, ce qui n'est pas adapté au travail en équipe.

C'est pourquoi les systèmes à base de vidéoprojection connaissent un succès croissant, alors qu'il y a quelques années tout le monde s'accordait à dire que l'avenir était aux HMD (et il l'est toujours... mais pour plus tard !). En effet, les vidéoprojecteurs donnent une image de bien meilleure qualité (c'est important quand on analyse un prototype virtuel de moteur par exemple), et il est possible de partage son expérience d'immersion avec d'autres personnes, par exemple une équipe d'ingénieur qui discute de la

conception d'un nouveau modèle de voiture. De plus, en utilisant plusieurs écrans disposés en arc de cercle, avec plusieurs projecteurs, il est possible d'obtenir une image à 180°, voire 360° (voir l'exemple du BARCO Reality Center, Figure 115).

L'exemple le plus connu d'un tel dispositif est le CAVE (*Cave Automatic Virtual Environment*) de l'Université d'Illinois (USA). Il s'agit d'un cube d'un peu moins de 2 mètres de côté, dont tous les côtés sauf le sol sont constitués d'écrans sur l'arrière desquels on va venir projeter une image, permettant une vision à 360° sur les côtés et environ 270° verticalement. Les utilisateurs portent des lunettes à obturation pour obtenir une vision en relief. Un capteur signale au logiciel CAVE la position de la tête de l'utilisateur. Ce dernier peut manipuler des objets à l'aide d'un dispositif de pointage. La Figure 119 montre une application d'interaction avec des modèles moléculaires de l'Argonne National Lab (USA).

Figure 119: Manipulation virtuelle de molécules dans le CAVE (ANL)

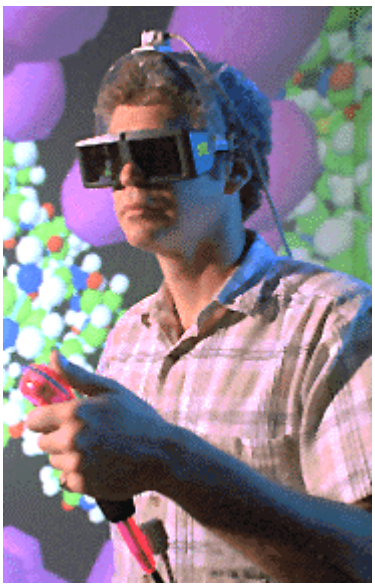


Figure 120: Capteur de position, lunettes à obturation et pointeur 3D du CAVE (ANL)

Les performances pratiques des CAVE sont réellement très impressionnants mais leur coût est exorbitant, puisque, outre les cinq vidéoprojecteurs haute résolution, il faut un calculateur graphique musclé pour produire dynamiquement les cinq images nécessaires à une fréquence d'au moins 60 Hz (puisqu'il s'agit d'un affichage stéréoscopique et qu'il faut au moins 30 Hz par œil pour éviter un scintillement trop évident). De fait, un CAVE coûte quelques centaines de milliers d'euros...

Figure 121: Simulateur full-flight de Mirage 2000 (dôme SOGITEC)

Le CAVE peut être vu comme une version simplifiée des simulateurs pilotés full-flight sous dôme (voir page).

Une autre alternative aux HMD est le moniteur binoculaire orientable de Fakespace (BOOM). Il s'agit tout simplement d'un double moniteur placé dans un boîtier équipé d'une optique adéquate, que l'utilisateur utilise comme une grosse paire de jumelles. Ce boîtier, évidemment assez lourd, est placé sur un bras articulé qui, en plus de faciliter sa manipulation, notamment grâce à un contrepoids, est équipé de capteurs de position.

La soufflerie virtuelle (*Wind Tunnel*) de la NASA (voir Figure 123) est l'une des plus célèbres applications du BOOM : elle permet de visualiser en trois dimensions l'écoulement de l'air autour d'un objet, calculé par un logiciel de simulation numérique, et, par l'intermédiaire d'un dataglove (voir §9.2.3), d'interagir avec le logiciel, par exemple pour zoomer sur un détail de l'image, ou pour mettre en évidence un filet d'air.

Le BOOM est un dispositif performant, dont le coût reste raisonnable (de l'ordre de 10 000 €).

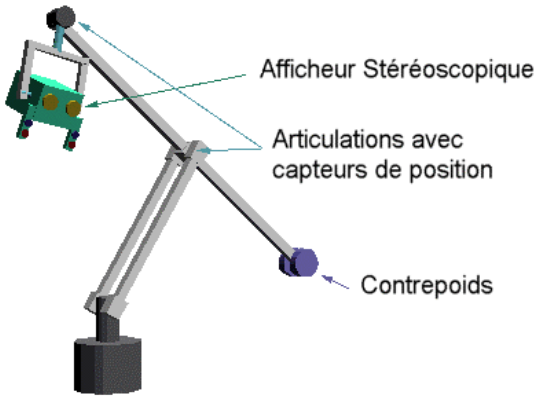


Figure 122: Le BOOM de Fakespace

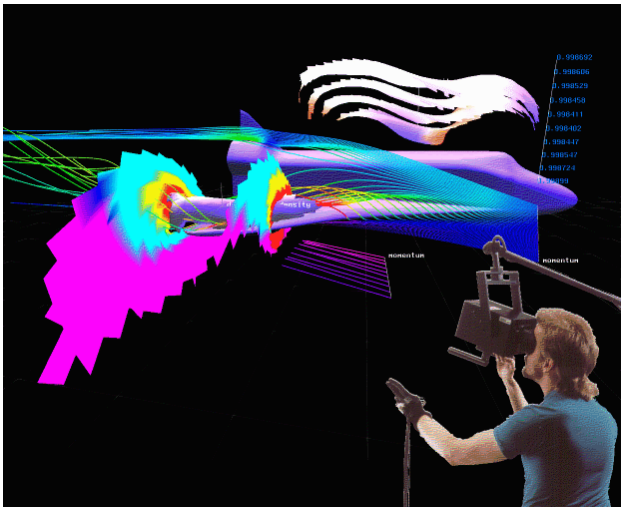


Figure 123: Utilisation du BOOM à la NASA

Le futur de la visualisation en réalité virtuelle réside peut-être dans les techniques de projections d'images directement sur la rétine à l'aide d'un faisceau lumineux, voire d'un laser (de très faible puissance, bien entendu). On s'affranchit ainsi des limitations inhérentes à la taille des pixels de la matrice LCD ou d'un tube cathodique et devrait permettre d'atteindre rapidement des résolutions très importantes à un coût modéré[95].

Ainsi, la société Microvision commercialise un système d'affichage par balayage rétinien (Retinal Scanning Display – RSD, voir Figure 124).

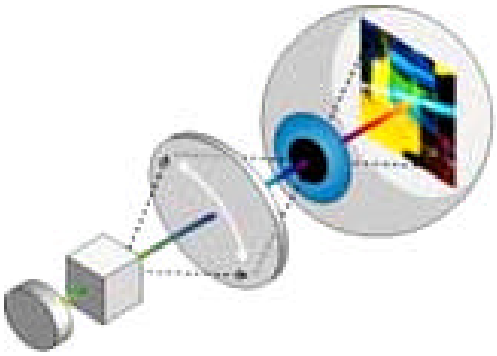


Figure 124: Affichage à balayage rétinien de Microvision

9.2.2.2 Systèmes de sonorisation

Comme nous l'avons vu, l'ouïe est, après la vue, le sens qui participe le plus à notre environnement sensoriel.

C'est pourquoi, de plus en plus, les systèmes de réalité virtuelle font intervenir des dispositifs sonores, monophonique ou multicanaux (plusieurs sources sonores distinctes). Alors qu'en monophonie la source du son est ponctuelle et localisable, les installations multicanaux (par exemple stéréophonique, avec deux canaux) permettent de multiplier les sources sonores ou de diffuser un son d'ambiance diffus. En effet, si vous prenez un son monophonique (par exemple l'enregistrement de l'ambiance d'un hall de gare), puis que vous l'envoyez sur deux enceintes suffisamment espacées placées de part et d'autre d'un auditeur, et que vous envoyez sur l'une l'enregistrement original et sur l'autre une version déphasé à 180° du signal sonore (opposition de phase), vous obtiendrez un son diffus, non localisable. L'auditeur, s'il ferme les yeux, aura l'impression d'être à l'intérieur du hall de gare.

[95] Cette technique pourrait permettre la projection d'images de 12 millions de pixel avec un champ de vision de 140°, pour un encombrement et un poids réduit du HMD.

Cette immersion par le son multicanal est très exploitée au cinéma et dans les installations de « cinéma chez soi » (home-theater), avec les systèmes Dolby Surround, Dolby Digital et DTS. En Dolby Digital, la bande son des films est codée numériquement sur 6 canaux : deux canaux à l'avant (gauche et droit), deux à l'arrière, un à l'avant centre et un caisson de basse ou *subwoofer* (qui ne restitue que les fréquences basses, par exemple inférieures à 80 Hz, c'est pourquoi on parle généralement de système 5.1 pour 5+1 canaux[96]). Les enceintes avant diffusent la musique et une bonne partie des effets sonores. Les enceintes arrières sont plutôt utilisées pour l'ambiance sonore. L'enceinte centrale, placée derrière l'écran de projection (les écrans de cinéma sont perméables aux sons) ou sur le téléviseur, restitue les dialogues. Ainsi, lorsqu'un acteur parle, le spectateur perçoit sa voix comme venant d'une source ponctuelle, localisée au niveau de l'image, ce qui est cohérent avec ce qu'il voit. Enfin, le caisson de basse permet de faire ressentir au spectateur des sensations physiques, par exemple lors d'explosions, en raison de la pression sonore des basses fréquences. L'oreille humaine effectuant la localisation des sons par les moyennes et hautes fréquences seulement, un seul caisson de graves placé n'importe où dans la pièce suffit. La Figure 125 représente une installation typique de home-theater en 5+1 canaux.

Le réalisme de l'immersion est tel que lorsque, dans un film, un avion survole l'action, le spectateur l'entend clairement traverser la pièce au-dessus de sa tête et il est parfois difficile de ne pas avoir le réflexe de lever la tête pour le voir passer.

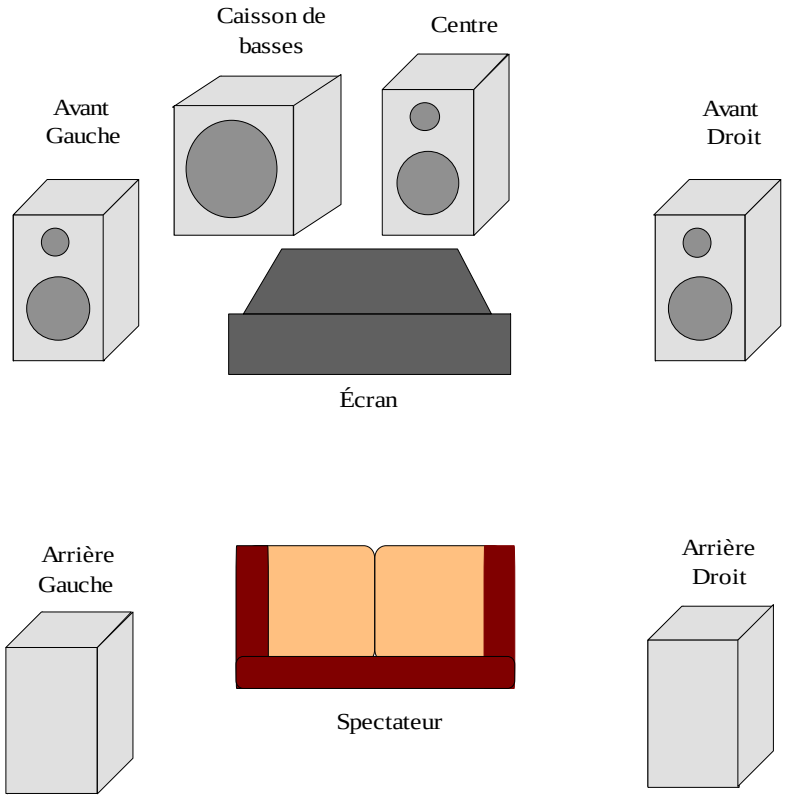


Figure 125: Installation de cinéma Dolby Digital®

Cette popularité des systèmes multicanaux chez le grand public a son importance. En effet, jusqu'ici, pour réaliser un environnement sonore réellement immersif, il fallait

[96] Le stéréo restitue 2 canaux, le surround 4 canaux (en fait 3, le quatrième étant généré artificiellement par un processeur sonore dit « pro-logic »).

faire appel à des technologies propriétaires, coûteuses et difficile à maintenir. Ainsi, depuis plusieurs années, de nombreux fabricants de cartes son pour micro-ordinateur, tels que la société Creative Labs, ont introduit des fonctions de « son 3D » sur leurs produits, mais sans qu'il n'y ait de standard, ce qui était très contraignant pour les éditeurs de jeux, qui devaient prévoir plusieurs générateurs sonores, en fonction de l'équipement du joueur.

Mais le succès du home-theater a popularisé les DTS, Dolby Digital et Dolby Surround, de sorte que désormais on voit apparaître des cartes (telles que la Sound Blaster Live, à partir de 450F seulement, qui pilote jusqu'à 5 enceintes et un caisson), qui gèrent ces standards. Gageons que rapidement nous verrons arriver des jeux sur 4 ou 6 canaux, avec une véritable immersion sonore, qui sera effective sur des systèmes bon marché.

Bien entendu, le cinéma n'est pas en soi un système de réalité virtuelle, puisque, s'il permet l'immersion visuelle et sonore, il n'est pas interactif (pour le moment !). Mais on comprend aisément l'intérêt de telles techniques pour la simulation d'entraînement…

9.2.3 Interactivité

Maintenant que l'on sait comment réaliser l'immersion de l'utilisateur dans l'environnement virtuel, il faut maintenant lui donner les moyens d'agir sur cet environnement.

9.2.3.1 Entrée de données

Les périphériques d'entrée de données sont désormais classiques : clavier, micro, etc., auxquels nous pouvons ajouter des entrées/sorties particulières, par exemple pour connecter du matériel dans la boucle.

Nous ne reviendrons pas en détail sur ces produits, car notre propos est ici de nous concentrer sur les interactions particulières entre l'homme et la machine que l'on voit en réalité virtuelle.

9.2.3.2 Périphériques de pointage

Les périphériques de pointage sont, en revanche, plus intéressants à étudier. En effet, si la souris permet de piloter un curseur sur un écran plat, comment s'y prend-on en trois dimensions ?

Dans certains système de CAO 3D, on utilisait des séries de boutons rotatifs, chacun permettant la translation ou la rotation suivant un axe. Évidemment, c'était fastidieux et peu naturel.

On a alors chercher des périphériques de pointage ayant, dans l'idéal, 6 degrés de liberté (3 translations, 3 rotations) et d'usage plus intuitifs.

Une première approche a consisté à adapter à la troisième dimension les périphériques classiques : souris, trackball, joystick...). Par le biais de capteurs de pression ou de position (voir §0), on peut situer le pointeur dans l'espace. Ainsi, la souris 3D de Logitech utilisait des faisceaux infrarouge pour se situer. Le GyroPoint de Gyration Inc. est un joystick 3D utilisant deux minuscules gyroscopes, qui lui donnent une précision de 0,1° pour un coût modique (de l'ordre de 100 €). Le spaceball (de Spaceball Technologies), qui est utilisé en CAO, est une boule fixe, que l'utilisateur va pousser ou tordre. 3 capteurs de forces (translation) et 3 capteurs de couple (rotation) vont permettre le déplacement avec 6 degrés de liberté d'un objet à l'écran. Ingénieux et peu

coûteux, ces dispositifs se révèlent en fait souvent imprécis et délicats à employer. Ainsi, il est difficile de ne pas imprimer de couple au spaceball lorsqu'on tente une translation et vice versa[97]...

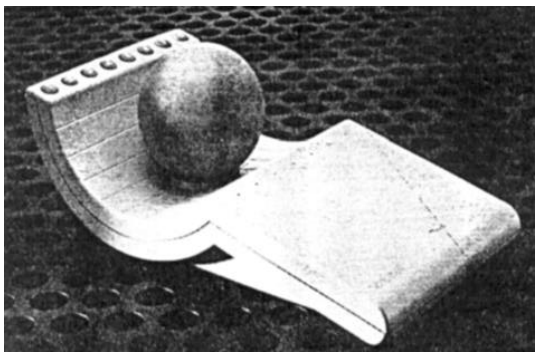


Figure 126: Spaceball

Figure 127: Joystick 3D

Certains s'avisèrent que la façon la plus intuitive de désigner un objet et de le manipuler, c'est finalement... à la main. C'est ainsi qu'apparurent à la fin des années 80 les **datagloves**.

L'un des pionniers fut le VPL DataGlove, apparut en 1987 et qui connut un grand succès pendant plusieurs années. Sur un gant en lycra étaient montées des boucles de fibres optiques (que l'on peut distinguer sur la Figure 128). La courbure de ces fibres provoquait l'atténuation de la transmission de la lumière à travers elles. En mesurant cette atténuation, on pouvait en déduire l'angle de flexion d'un doigt.

La position dans l'espace de la main était, elle, déterminée grâce à un capteur Polhemus. Les systèmes Polhemus (Fastrak, Isotrak...) utilisent des bobines que l'on attache à l'objet que l'on souhaite situer dans l'espace. Ces bobines, parcourues par un courant, produisent un champ magnétique qui est mesuré par un capteur fixe, plusieurs dizaines de fois par seconde, avec une précision de l'ordre de 0,1°. Très pratique, ce type de capteurs est néanmoins sensible aux perturbations extérieures et n'a qu'une portée limitée (quelques mètres).

Toutefois, les fibres optiques, fragiles, s'usaient rapidement et limitait la durée de vie (quelques mois en utilisation quotidienne !) d'un matériel relativement onéreux (7 000$ de l'époque). Dans un modèle ultérieur, le Cyberglove, VPL remplaça les fibres optiques par des jauges de contrainte, plus précises et plus endurantes. Néanmoins, les deux systèmes ont en commun la nécessité d'un calibrage fréquent du gant.

[97] Certains équipements sont dotés de commutateurs pour inhiber la translation ou la rotation, mais, bien évidemment, cela réduit d'autant l'intérêt du dispositif.

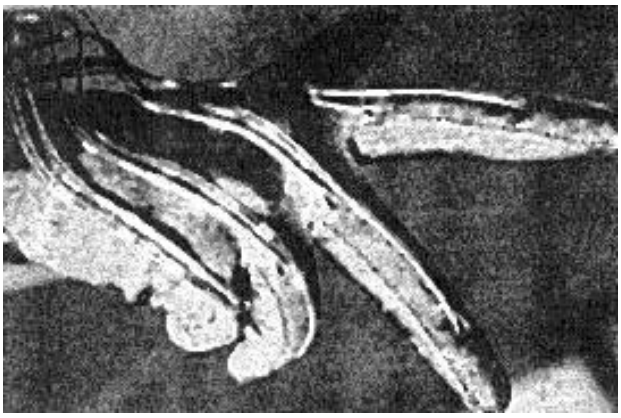


Figure 128: VPL DataGlove

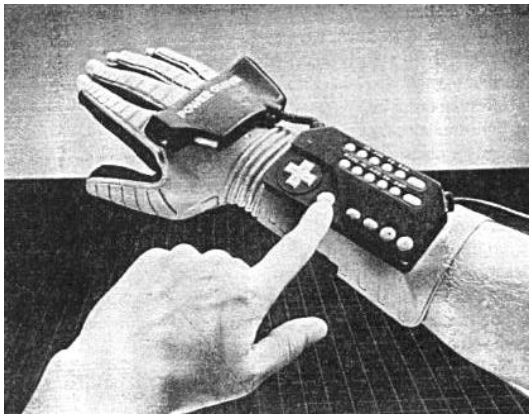


Figure 129: Mattel PowerGlove

Un autre modèle de dataglove remarquable, en raison de sa simplicité, de son faible coût et de sa popularité, fut le PowerGlove de Mattel, commercialisé en1987 pour le compte de Nintendo. Le système de mesure de la position de la main utilisait... des ultrasons. Pour la mesure de la flexion des doigts, Mattel choisit une technologie originale, à base d'encre conductrice : quand l'utilisateur pliait les doigts, la distance entre les gouttelettes d'encre augmente et la résistivité s'accroit. La précision était médiocre, mais le prix imbattable, et il s'en vendit des milliers. Le PowerGlove joua un grand rôle dans la démocratisation de la réalité virtuelle.

Il existe aujourd'hui de nombreux dataglove, dont les technologies n'ont guère évoluées. L'innovation la plus importante fut l'extension du principe du dataglove... au corps tout entier.

En effet, les sociétés d'animation de personnages de synthèses (héros de films d'animations, présentateurs virtuels[98], doublures numériques d'acteurs...) peinaient à obtenir des mouvements réalistes et donc complexes d'un corps entier dans des délais raisonnables.

[98] L'un des pionniers de l'utilisation du datasuit en France fut Medialab, filiale de Canal+. Vous souvenez-vous de la cyber-présentatrice Cléo ?

L'idée fut alors de coupler les mouvements du personnage virtuel à ceux d'un humain, d'un « marionnettiste » d'un nouveau genre. On testa différentes solutions, telles que la reconnaissance optique par des caméras. Mais finalement, ce qui eut du succès, c'est un costume (typiquement en lycra) équipé de capteurs, sur le modèle du gant de VPL : le datasuit.

Il est aujourd'hui possible d'animer en temps réel un personnage de synthèse, de le faire parler, danser, sourire, avec seulement deux animateurs (un avec un datasuit pour animer le corps, l'autre avec un dataglove pour animer le visage et notamment synchroniser les mouvements de la bouche avec les dialogues).

Figure 130: Exemple de datasuit

9.2.3.3 Périphériques haptiques

Le problème commun à tous ces systèmes d'interaction avec l'univers virtuel est l'absence totale du sens du toucher et de retour de force (sensation de résistance d'un objet). Lorsque, avec un dataglove par exemple, on saisit un objet virtuel, les doigts ne rencontrent que le vide, ce qui est extrêmement perturbant. Et parfois rédhibitoire, comme dans les simulateurs pilotés, où il est essentiel que le pilote fasse l'apprentissage de sensations physiques.

Pour le cas des simulateurs de vol, le problème a été résolu dans les simulateurs full flight : cabine montée sur vérins pour reproduire les mouvements de l'appareil, reproduction fidèle du cockpit et des commandes, combinaison gonflable pour simuler les accélérations, etc. Toutefois, le coût d'un tel système est prohibitif, et le réserve à un nombre restreint d'applications.

Or, il y a un besoin important actuellement de retour tactile, car la réalité virtuelle est de plus en plus utilisée pour l'entraînement à la chirurgie, ou pour le télépilotage de robots. Dans ces applications, il est essentiel que l'opérateur puisse doser avec précision ses gestes. Sans retour tactile, cela n'est pas possible, et pourrait conduire à des catastrophes.

C'est là qu'interviennent les périphériques dit haptiques, qui transmettent des sensations tactiles à l'utilisateur et permettent le retour de force.

Le retour de force peut être réalisé à l'aide d'un exosquelette motorisé, englobant une partie du corps (voir Figure 131), ou en rendant le périphérique de commande solidaire d'un bras articulé muni d'actionneurs d'actionneurs pneumatiques, électriques ou hydrauliques (voir Figure 132). Bien entendu, ces dispositifs doivent être couplés à des capteurs de position suffisamment précis.

Ces dispositifs (en particulier les exosquelettes) restent toutefois trop encombrants et onéreux pour le grand public, qui, lui, a un besoin important : les jeux. En effet, n'est-il pas frustrant de conduire une voiture sans ressentir les vibrations du moteur ? de mener un combat de boxe sans sentir le contact de l'adversaire sous ses poings ?

C'est pourquoi des fabriquants de matériels de jeux grand publics se sont penchés sur la question, et on sortis quelques modèles à retour de force. Ainsi, on peut maintenant

trouver des volants dont la résistance en rotation est commandée par le logiciel et qui sont équipés de vibreurs pour permettre au joueur de « sentir » le moteur, la route et les chocs. Ces mêmes vibreurs équipent certains joysticks ou souris. Les résultats sont généralement médiocres et décevants, mais il ne faut pas oublier que ces périphériques coûte rarement plus d'une centaine d'euros.

Figure 131: Dextrous Arm Master (Sarcos)

Figure 132: Le Phantom de SensAble Tech.

Figure 133: Joystick à retour de force de Microsoft

Le retour tactile, quant à lui, est un problème similaire, mais toutefois distinct. Le sens du toucher est également très important, car il permet de déterminer quantité d'informations sur la nature de l'objet saisi : état de surface (lisse ou rugueux), température, dureté...

Il existe cependant peu de périphériques de retour tactile. La sensation de contact est souvent reproduite grâce à des poches d'air implantées dans le gant (le TeleTact de Airmuscle Ltd. ne comporte pas moins de 30 poches sous les doigts et la paume, et fournit par la même occasion un certain retour de force). D'autres modèles utilisent de minuscules vibreurs. Des modèles expérimentaux recourent à des alliages à mémoire de forme : une matrice de micro-tiges est solidaire d'une série de fils en alliage de titane et de nickel. Lorsqu'un fil est parcouru par un courant, il chauffe par effet Joule et se contracte, déformant ainsi légèrement la tige qui vient appuyer contre les doigts, permettant ainsi de simuler un état de surface.

9.2.4 Imagination

Un système de réalité virtuelle n'est pas tenu de respecter la réalité. Équipé d'un casque de visualisation, un ingénieur de Renault peut faire flotter un véhicule dans l'espace et le faire tourner. Un joueur dans une galerie d'arcades peut combattre des dragons dans un décor médiéval-fantastique, ou se retrouver aux commandes d'un vaisseau spatial. Et pourtant, le sentiment d'immersion est parfois très importante chez les joueurs...

Le critère d'imagination est certes discutable, car très subjectif. Mais force est de constater son importance, surtout dans le domaine du jeu vidéo, où il faut que le joueur « s'y croit ». Les jeux de rôles sur micro-ordinateur des années 80 (comme Ultima ou Wizardry) peuvent faire sourire de nos jours par leur graphisme rudimentaire, quand ils n'étaient pas purement et simplement en mode texte, mais l'auteur de ces lignes peut

lui-même témoigner que, pour certains d'entre eux, on arrivait vraiment à se plonger dans l'aventure.

L'appel à l'imagination n'est donc pas à négliger.

9.2.5 Logiciels

Tous les matériels que nous venons de voir n'ont d'intérêt que couplés à des logiciels ad hoc. De fait, le logiciel représente souvent l'investissement le plus important d'un système de réalité virtuelle.

On trouve maintenant sur étagère des logiciels dédiés à la réalité virtuelle :

- Les boîtes à outils (*toolkits*) : elles offrent à l'utilisateur les fonctions nécessaires pour gérer les périphériques de réalité virtuelle (HMD, gant, souris...), d'importer les fichiers issus de modeleurs 3D ou de logiciels de CAO, et de définir la disposition et le comportement des objets virtuels (scénario...).
- Des éditeurs/modeleurs : pour pouvoir évoluer dans un monde virtuel, il est nécessaire de le modéliser en décrivant l'environnement graphique (terrain...) et sonore, ainsi que les objets virtuels (forme, comportement, interactions entre eux et avec l'opérateur).

L'utilisation combinée d'un toolkit et d'un modeleur peut suffire à développer des mondes virtuels complexes.

Les langages à objets (surtout C++, Java) sont particulièrement privilégiés par ces produits. Notons le succès de VRML (*Virtual Reality Markup Language*, langage de description de scènes 3D inspiré de HTML, le langage de description de pages web).

9.2.6 Exemples d'utilisation

9.2.6.1 Téléopération de robot

Un problème fréquent dans le pilotage de robots en milieu hostile (centrale nucléaire, espace, etc.) est la médiocrité du retour de l'information, généralement réduite à un écran vidéo. De 1949 (manipulateur maître-esclave de Goertz de l'Argonne National Lab.) à 1990, les améliorations n'ont essentiellement porté que sur la mécanique d'asservissement (avec ou sans retour de force). Puis vint la réalité virtuelle, qui permis de donner à l'opérateur l'illusion d'être à la place du robot, grâce à une immersion dans son environnement, ce à quoi on pourra ajouter un retour d'information tactile pour faciliter la télémanipulation d'objets. L'apport de la réalité virtuelle est donc non seulement ergonomique, mais traite également le problème du retour d'informations.

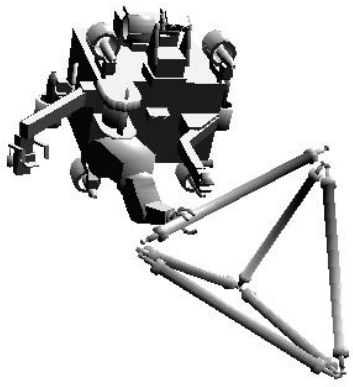


Figure 134: BAT assemblant une structure

Au Space System Laboratory de l'Université du Maryland, on a préparé la construction de la station spatiale internationale grâce au robot sous-marin BAT (*Beam Assembly Teleoperator*), l'eau simulant la micro-gravité. BAT est une plate-forme mobile, quipée de deux caméras couplées, une pour chaque œil, permettant ainsi la perception du relief. Dans la station de contrôle, l'opérateur récupère une vue en 3D par l'intermédiaire d'un

HMD, les mouvements des caméras du robot étant couplés à ceux du casque. Un dispositif mécanique asservit les mouvements du bras du robot à ceux de l'opérateur.

Grâce à ce type d'interface, il a été possible de piloter depuis le Ames Laboratory (USA) un robot situé... en Antarctique. La NASA a également utilisé ces techniques pour simuler le pilotage du robot martien Pathfinder. Quant à l'IFREMER, elle étudie également, depuis déjà quelques années, un système de téléopération de véhicule autonome sous-marin par réalité virtuelle.

9.2.6.2 Architecture et design

La réalité virtuelle offre la possibilité de modéliser des bâtiments avant même leur existence, et de se déplacer à l'intérieur de ceux-ci afin d'évaluer leur esthétique, leur sécurité et leur ergonomie.

C'est ainsi que la RATP a fait appel à Medialab et IBM pour son projet Concept 2000 de station de métro future. Une base de données CAO sous CATIA a été récupérée et optimisée avec le logiciel Explore de Medialab, puis les interactions ont été définies sous un logiciel auteur de réalité virtuelle de la même société, CLOVIS.

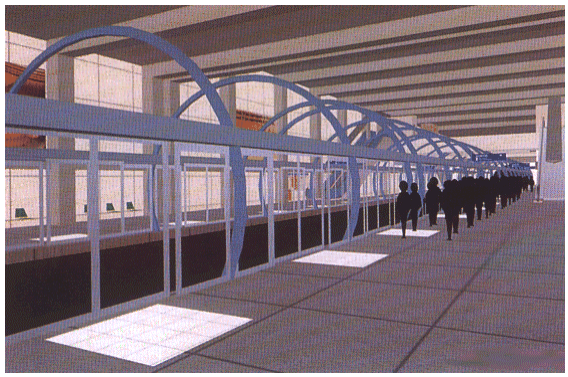


Figure 135: Modélisation de METEOR (Medialab/RATP)

Grâce à un HMD et un calculateur graphique SGI, il devint alors possible de se promener à l'intérieur des infrastructures du métro METEOR (actuelle ligne 14), alors que celle-ci n'était pas encore construite, et de juger, entre autres, la facilité des accès et la pertinence de la signalisation.

Un autre projet Medialab/IBM a été, en collaboration avec l'ENSAM, de reconstituer virtuellement l'Abbaye de Cluny. Le résultat, présenté à IMAGINA[99] en 1993, connut un large succès. Il s'agissait également d'une expérience de téléprésence, puisque deux opérateurs distants (l'un à Monaco, l'autre à Paris) pouvaient se voir[100] et interagir à l'intérieur de l'abbaye virtuelle par le biais d'une liaison Numéris (pour les données) et d'un simple téléphone (pour la voix).

[99] Imagina est le colloque phare des « nouvelles images » (images de synthèse, réalité virtuelle...). Il se déroule tous les ans à Monaco.

[100] En utilisant un avatar, c'est-à-dire une représentation symbolique d'un utilisateur dans un monde virtuel. Les avatars sont très utilisés par les lieux de rencontres virtuels qui fleurissent depuis quelques années sur Internet.

9.2.6.3 Marketing

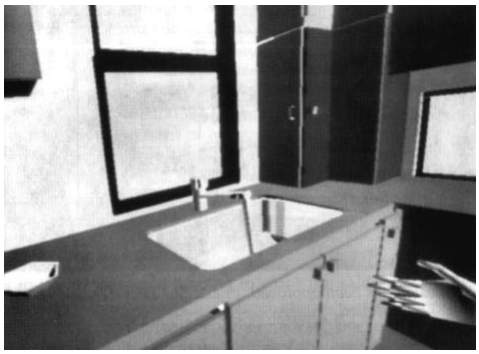


Figure 136: Visite d'une cuisine virtuelle

Comme pour l'ingénierie ou l'architecture, le fait de pouvoir visualiser et manipuler un objet qui n'existe pas encore a attiré certains spécialistes du marketing.

Au Japon, le géant Matsushita utilise un HMD relié à un calculateur graphique pour aider des clients potentiels à choisir leur cuisine. Lorsqu'un client ne trouve pas dans le hall d'exposition la cuisine de ses rêves, il est possible, à l'aide d'un logiciel de CAO spécialisé, de modéliser celle-ci et de permettre au client de naviguer à l'intérieur et même d'en ouvrir tiroirs et placards à l'aide d'un dataglove. Par la suite, le concept a été étendu à la réalisation de maquettes virtuelles de maisons entières, basées sur un catalogue de plus de 50 000 éléments. De nombreuses applications similaires sont aujourd'hui couramment utilisées de par le monde.

9.2.6.4 Recherche pharmaceutique

Figure 137: Visualisation en relief de molécules

Les entreprises de biotechnologie sont des clients importants des fournisseurs de systèmes de réalité virtuelle.

En effet, les molécules organiques qui sont à la base des derniers progrès de ce secteur sont particulièrement complexes. Leur manipulation sur un écran 2D ne permet pas de correctement reconnaître les différentes conformations spatiales. Aujourd'hui, de nombreux logiciels spécialisés permettent la visualisation et la manipulation en relief de telles molécules, avec une modélisation des propriétés physiques et chimiques des molécules (conformations, force des liaisons, sites actifs...).

9.2.6.5 Parcs à thèmes

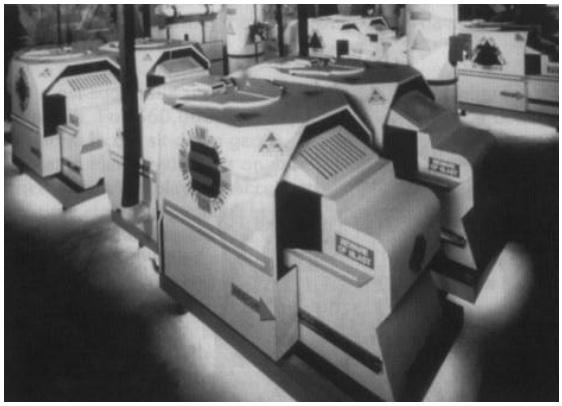


Figure 138: Simulateurs Battle Tech

Les parcs à thèmes (et à gros budgets !) comme Eurodisney investissent des sommes parfois considérables dans des attractions virtuelles, bien que la plupart du temps il ne s'agissent que de la projection d'un film avec animation des sièges des spectateurs par des vérins (sans interacton).

L'une des applications les plus évoluées de la réalité virtuelle aux parcs à thèmes, qui fut aussi l'une des premières, est le Battle Tech Center de Chicago, un réseau de simulateurs de robots de

combat futuristes créé par Virtual World Entertainement. Dans son cockpit, le joueur dirige son robot et combat d'autres joueurs. Il est possible de monter des alliances pour former une équipe, des microphones assurant la communication de chacun avec ses partenaires. Le succès fut tel que plusieurs autres centres furent ouverts aux États-Unis et au Japon. L'investissement à chaque fois approchait les 20 M€, mais la rentabilité était bonne : en un mois, le Battle Tech Center de Yokohama (Japon) a réalisé 400 000 entrées. À 9 € la séance de 25 minutes, l'optimisme était permis...

9.3 Bases de données d'environnement

La représentation de l'environnement physique est critique pour l'obtention d'environnements synthétiques de qualité. Or, la quantité de données à gérer peut être très importante : une base de données terrain peut comporter des millions de données. C'est donc un élément à ne pas négliger dans une simulation (voir Figure 12, page 34), d'autant que, comme nous le verrons, il peut être la source de nombreux dysfonctionnements d'une simulation.

9.3.1 Terrain statique et dynamique

Le terrain peut être **statique** ou **dynamique**. L'impact sur le réalisme est important. Ainsi, j'ai entendu une fois un capitaine de l'US Army critiquer vivement un simulateur de char parce que le terrain ne gardait pas la trace du passage des chars, alors que les traces laissées sur le sol sont un élément décisionnel non négligeable.

Je prendrai un autre exemple dans le monde du jeu (de loisir) sur ordinateur. Dans le jeu Wizardry, best-seller du jeu de rôle sur micro-ordinateur dans les années 80 que j'ai déjà mentionné, le joueur se promène dans un labyrinthe souterrain (« donjon »), enfonce des portes, combat des monstres et récupère des trésors[101]. Une minuscule vue en 3D subjective des couloirs très schématique permet de s'orienter. Malgré la rusticité du graphisme, ce jeu était extrêmement prenant. Néanmoins, j'ai toujours été troublé par le fait que si on défonçait une porte et qu'on tuait les monstres dans la pièce pour leur voler leur or et que l'on ressortait pour rentrer à nouveau, on retombait sur les mêmes monstres et le même trésor. Le verrou de la porte semblait d'ailleurs s'être réparé instantanément ! Ceci était, à mes yeux, des plus choquant du point de vue réalisme. Dans le jeu plus moderne Doom, le graphisme est également frustre (pour notre époque), mais fluide, et le terrain est devenu dynamique : les cadavres restent en place et les objets déplacés… restent déplacés. Du coup, si vous tournez en rond dans le labyrinthe, vous vous en apercevez immédiatement. Non seulement cela augmente le réalisme, mais cela fournit une aide au joueur.

Dans le cas d'un terrain **statique**, le nombre de données à gérer reste important :

- Données **altimétiques** (altitudes et profondeurs, courbes de niveau…). Généralement on effectue un **maillage du terrain** (tel que celui de la Figure 139), dont la résolution dépend du niveau de détails recherché. Il peut arriver que des fossés ou des crevasses ne soient pas prises en compte par le terrain numérique, mais un champ de manœuvre de 50x50 km peut difficilement être numérisé avec une résolution métrique[102] !
- **Nature du terrain** (prairie, forêt, sable, marécage…). Ce terrain peut être caractérisé de façon plus précise : taille des arbres (quelle est la visibilité à l'intérieur de la forêt ?), dureté du sol (un véhicule d'un certain poids va-t-il s'enfoncer ?), etc.

[101] Il s'agissait de l'une des premières adaptations du jeu de rôle « Donjons & Dragons », très populaire alors (et qui l'est toujours sous sa forme « avancée »).
[102] En effet, cela ferait 2,5 milliards de mailles !

– **Éléments planimétriques** (maisons, routes, ponts, rivières, voies ferrées, rochers…). Là encore, chacun peut avoir des caractéristiques particulières, par exemple son nom (toponyme), ou la charge maximale admise (pour un pont).

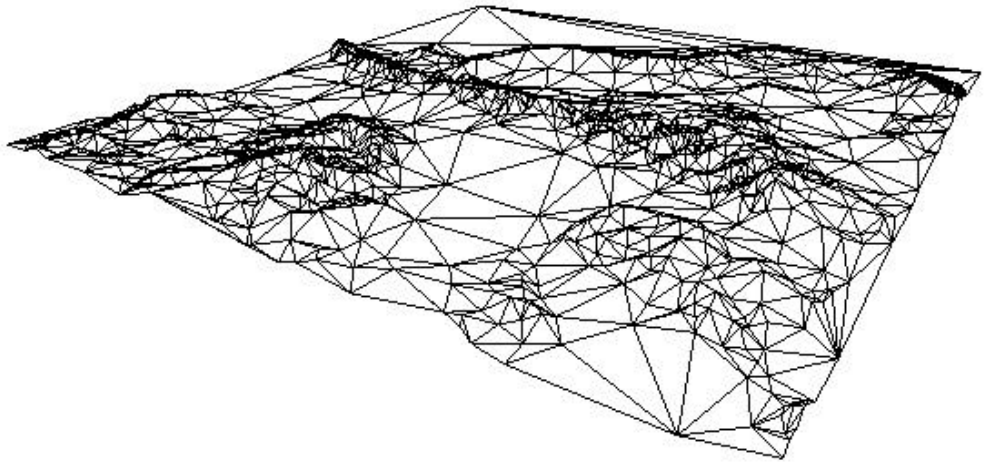


Figure 139: Exemple de maillage de terrain à l'aide de triangles irréguliers (Thomson T&S)

9.3.2 Production d'un terrain numérique

L'établissement d'une carte n'est pas une mince affaire. S'il n'existe pas de base de données géographique ou de carte du terrain considéré à la résolution requise, l'opération peut s'avérer longue et coûteuse. Cette incompatibilité avec l'urgence a amené les américains, notamment au travers de la Defense Mapping Agency, à investir des sommes importantes pour numériser tous les théâtres d'opération potentiels du monde entier, afin de pouvoir rapidement mettre en place des simulations de planification de missions.

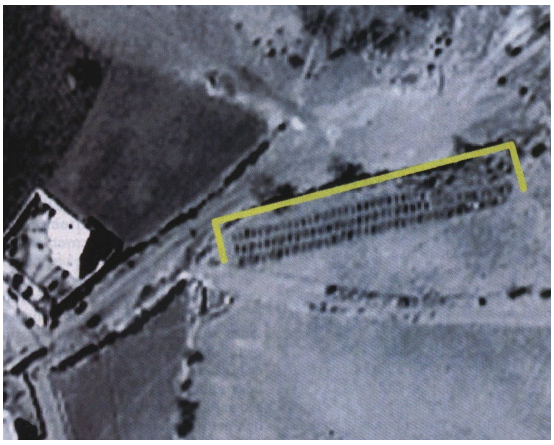


Figure 140: Tombes fraîches au Kosovo, vues par un satellite US (NIMA/NRO)

Les sources de données géographiques sont, outre les cartes préétablies, les instituts spécialisés (Institut Géographique National, Centre Géographique Interarmées, etc.), les satellites d'observation[103] (SPOT, HELIOS), les avions de reconnaissance (Mirage F1CR, drônes), les relevés sur le terrain. L'importance de disposer de données géographiques fiables est illustrée par exemple par les campagnes allemandes en France en 1940 : les armées allemandes disposaient de cartes de la France détaillées et mises à jour, ce qui a contribué au succès de ces troupes. Dans l'ex-URSS, les cartes disponibles étaient volontairement fausses, afin d'éviter de fournir des renseignements à l'ennemi. Malheureusement, les satellites permirent une cartographie précise, de sorte que rapidement les occidentaux pouvaient se procurer des cartes exactes de l'URSS dans n'importe quelle librairie.

Une nouvelle méthode pour produire des terrains en 3D est la **photogrammétrie**. À partir de photos 2D (reconnaissance aérienne, satellite...) et des coordonnées de quelques points, les logiciels de photogrammétrie peuvent reconstituer le relief du terrain et des structures qu'il comporte (bâtiments...). Il est ainsi possible de produire en quelques jours, voire quelques heures, un modèle 3D d'une zone d'une qualité tout à fait acceptable pour, par exemple, des applications de répétition de mission ou d'urbanisme (étude d'impact environnemental...). Notons que la photogrammétrie est également utilisée en CAO pour produire des modèles en 3D de pièces, installations... Un exemple de logiciel exploitant la technique de photogrammétrie est RapidScene, d'Evans & Sutherland.

Une fois acquises les données géographiques, un nouveau problème survient : comment stocker et échanger ces données ? Il faut pour cela choisir un format de données. Pas de chance, il existe plusieurs standards, comme nous le verrons par la suite…

9.3.3 Définitions

On appelle **environnement** d'un système les objets, variables et processus externes au système qui influencent le comportement de ce système.

Le terme environnement (ou **environnement naturel**) est aussi utilisé pour désigner les détails de représentation du domaine dans lequel évolue le système (i.e. le relief du terrain, les entités planimétriques, les conditions météorologiques, alternance jour/nuit[104], etc.). Cet environnement peut être statique (cas du relief du terrain) ou dynamique (cas de la météo).

[103] On atteint maintenant des résolutions de moins d'un mètre sur des satellites commerciaux. La Figure 140 donne une idée de ce qu'obtiennent les satellites militaires, sachant que cette image d'un satellite KH-11 américain a été brouillée pour cacher sa vraie résolution...

[104] La nuit est souvent traitée dans les simulations comme une condition météorologique particulière. C'est en tout cas un facteur environnemental à ne pas négliger, notamment dans les simulations militaires : les opérations au Kosovo ont montré l'importance du combat de nuit dans la guerre moderne.

On appellera **données d'environnement** les données décrivant cet environnement : puissance d'un brouillage électromagnétique, données altimétriques, résistance d'un pont, température de l'air, vitesse du vent, etc.

9.3.3.1 Environnement synthétique

Un **environnement synthétique** est une représentation du monde réel dans une simulation. Il est composé d'entités de la simulation, de l'environnement naturel (terre, mer, atmosphère, espace), et des interactions avec les entités de la simulation (d'après [IEEE94].

On parle également de « monde virtuel ».

Le département de la défense américain fait intervenir dans son glossaire M&S (voir [DOD97] la notion de **haut niveau de réalisme** pour les environnements synthétiques : ceux-ci doivent permettre l'immersion dans la simulation, et sont appuyés par des modèles comportementaux et des effets spéciaux (visuels, sonores, sensitif…). Les SE peuvent être créés à l'aide d'une seule machine ou d'un réseau de stations distantes. Cette définition est très discutable : elle semble avoir été rédigée par un aficionado de la simulation pilotée, et s'attarde un peu trop sur les moyens (effets spéciaux visuels, distribution…). Mais elle a le mérite de rappeler qu'un SE peut être effectivement distribué sur plusieurs ordinateurs, par exemple à l'aide du standard HLA.

9.3.3.2 Base de données d'environnement

Les bases de données d'environnement sont des systèmes de bases de données spécifiques que les simulations viennent interroger pour connaître leur environnement : terrain, météo, etc.

Figure 141: Exemple de modification d'environnement

L'utilisation d'un serveur de BDE unique pour l'ensemble d'une fédération de simulations est idéale, permettant d'assurer la cohérence des environnements naturels entre toutes les simulations, car si cela n'est déjà pas si simple avec un environnement statique, par exemple en raison des différences représentation ou d'interprétation du terrain d'une simulation à l'autre, cela devient particulièrement complexe avec un environnement dynamique (météorologie changeante, altération du terrain…). La Figure 141 donne un exemple d'altération dynamique de l'environnement : la destruction d'un pont (image tirée d'un projet européen EUCLID RTP 11.10 traitant justement de la gestion dynamique de l'environnement naturel d'une fédération de simulations).

L'utilisation d'une BDE unique (utilisant SEDRIS par exemple) résout nombre de difficultés, mais pose, en revanche, un problème de performances certain (qui peut être en partie contourné en utilisant des caches ou des copies locales de l'environnement).

9.3.3.3 Système d'information géographique

La vocation d'un système d'information géographique (SIG) est de relier des données (démographiques, géologiques, économiques, etc.) à un espace géographique.

Ce type de système est particulièrement utilisé pour l'aide à la décision : quel est le meilleur endroit pour chercher du pétrole ? Où implanter un centre commercial ? Dans quels départements faut-il intensifier la campagne électorale ? Comment peut-on améliorer le plan d'occupation des sols ? Par où faire passer l'autoroute ? etc.

Traiter des SIG n'est pas notre propos ici, même si la simulation est un outil parfois utilisé par ces systèmes, par exemple pour des études prospectives (simulation de l'évolution de l'implantation de l'algue *Taxifolia* en Méditerranée, de l'évolution de la pollution d'une zone industrielle...). En revanche, ces SIG sont des sources importantes de données géographiques et d'outil de traitement de ces données, qu'il convient de ne pas négliger.

9.3.4 Problèmes potentiels avec une base de données d'environnement

Au sein d'une base de données d'environnement coexistent diverses objets, dont la cohérence et le réalisme peuvent poser divers problèmes :

- Précision insuffisante pour la simulation qui les utilise (imprécision de la simulation).
- Au contraire, précision trop importante (gaspillage de ressources de stockage et de temps de calcul, diminuant les performances du système).
- Erreur sur les données : route passant sur une rivière sans pont, mauvaise coordonnée en altitude (d'ou un « pic » incohérent)…

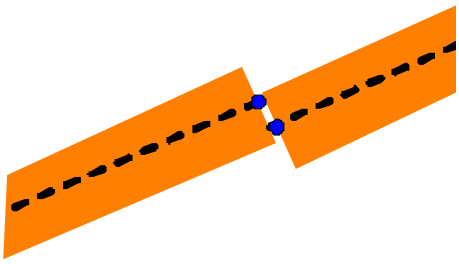


Figure 142: Problème d'interconnexion de deux routes

- Imprécision des éléments topographiques : deux tronçons de route qui devraient se rejoindre ne sont pas connectées (voir Figure 142), routes traversant un obstacle (au lieu de le longer)…
- Non respect des lois physiques : une rivière qui remonte une pente…

- Erreur de maillage : « trous » dans le maillage (toute la surface du terrain n'est pas décrite), polygones non raccordés, se recouvrant ou dupliqués.
- Ambiguïté conceptuelle : un élément d'environnement peut être vu de diverses façons. Ainsi, un barrage peut-il être un obstacle sur une rivière ou un pont routier (Voir Figure 143).
- …

Figure 143: Barrage ou route ?

Ceci explique pourquoi les producteurs et les grands utilisateurs de bases de données d'environnement se dotent de services de contrôle de la qualité. C'est par exemple le cas en France du centre d'études en géographie numérique (CEGN), dont le service qualité vérifie la cohérence des terrains vectorisés par rapport à la réalité, grâce à des outils spécialisés, des photos satellite et… au coup d'œil de l'opérateur.

Le logiciel ModSAF (un CGF, voir page 240) nous fournit de bonnes illustrations des effets pervers de ces erreurs dans la base de données. Ainsi, dans le cas de nos deux routes qui ne sont pas connectées, les véhicules vont se retrouver bloqués, comme s'il y avait un mur au niveau de la jonction.

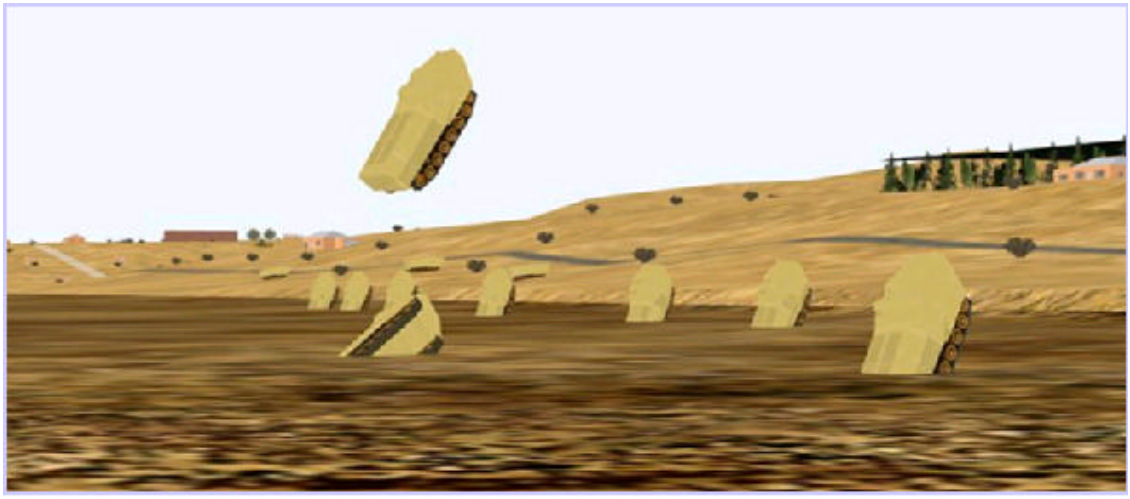


Figure 144: Curieux comportement de ModSAF !

Lors de l'utilisation de simulations distribuées, se pose des problèmes supplémentaires de cohérence entre les bases de données. La Figure 144 illustre ce qui peut se produire : visiblement, la base de données de ModSAF et celle de l'outil de visualisation graphique ne sont pas cohérente : les transports de troupes, gérés par ModSAF, croient visiblement qu'ils sont arrivés sur le rivage et qu'ils grimpent la pente. Mais pour l'outil de visualisation, le rivage est situé une centaine de mètres plus loin… De tels bizarreries étaient par exemple courantes dans SIMNET, où l'on voyait des chars voler ou croiser des avions volant au ras du sol !

Ces erreurs peuvent avoir une influence direct sur les résultats. Prenons comme exemple deux simulateurs de chars différents, connectés entre eux. L'un utilise une base de données d'une précision de 20m, l'autre une base d'une précision de 50m. Une butte de 30 mètres sera prise en compte par le premier, mais pas forcément par le second : ce dernier pourra voir un ennemi masqué par la butte, puisque pour lui celle-ci n'existe pas ! L'ennemi, en revanche, pensera qu'il est à l'abri.

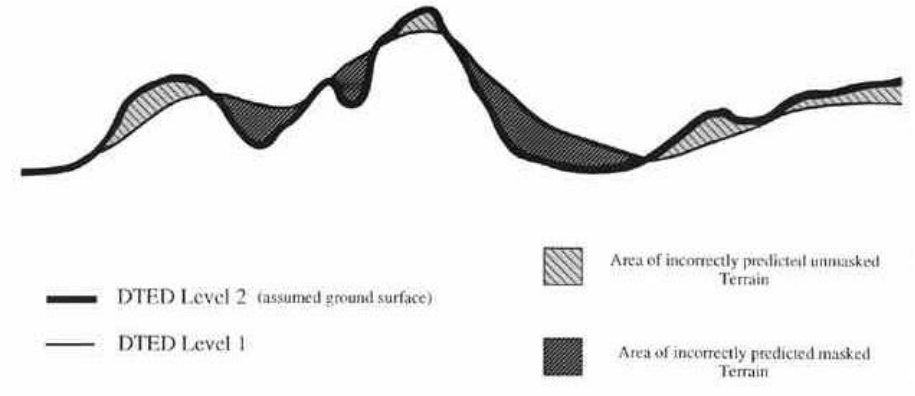


Figure 145: Erreur de LOS due aux DTED 1 et 2

La Figure 145 (tirée de [FATA94]) donne un autre exemple d'erreur : le format de données DTED peut avoir plusieurs niveaux de précision, mais même le plus élevé n'est pas une représentation exacte à 100% du terrain réel[105]. Le graphique donne un aperçu de l'étendue de l'erreur induite lors du calcul du terrain visible ou masqué (LOS) pour une entité d'une simulation. Bien entendu, il est impossible d'avoir une précision à 100%, mais il faudra choisir avec pertinence la résolution adéquate pour obtenir la précision requise des résultats, sachant bien entendu que plus la résolution est élevée, plus la base de données terrain sera grande et donc lourde à traiter. Pour donner un ordre d'idées, voici ce que serait une base de données DTED décrivant les élévations d'un terrain d'environ 200 km² :

Niveau de DTED	Finesse (*post-spacing*)	Nombre de points	Nombre de données	Taille de la base
DTED 1	~ 100 m	90 000	1 442 401	5 Mo
DTED 2	~ 30 m	810 000	12 967 201	54 Mo
DTED 3	~ 10 m	5 000 000	144 024 001	583 Mo !
DTED 4	~ 3 m	21 250 000	1 296 072 001	6,3 Go !!
DTED 5	~ 1 m	506 250 000	11 660 000 001	68 Go !!!

Il va sans dire qu'au-delà du DTED 2, les bases de données disponibles « sur étagère » sont très rares !

[105] Il ne s'agit pas d'un défaut du DTED, mais d'un problème inhérent à toute donnée géographique, la précision se payant parfois très cher, comme on le voit ici. Pensons à la mesure des côtes de la Bretagne, célèbre problème mathématique sur les courbes fractales…

9.3.5 Rappel : systèmes de coordonnées

Un autre problème que l'on peut rencontrer est celui de l'utilisation de différents systèmes de coordonnées entre les différentes bases, ou de différentes unités. La perte récente d'une sonde américaine lors de son approche de la planète Mars à la suite d'une confusion entre les kilomètres (système métrique) et les miles (système dit « impérial ») nous montre qu'une telle erreur est non seulement possible, mais probable.

9.3.5.1 Rotondité de la Terre et ellipsoïde de référence

Tout d'abord, faut-il tenir compte de la rotondité de la Terre ? On peut probablement s'en passer pour une zone de quelques kilomètres, mais si le terrain dans lequel se déroule la simulation couvre des milliers de kilomètres carrés, il faudra tenir compte de l'horizon. Par exemple, un radar verra son champ de détection limité pour les basses altitudes.

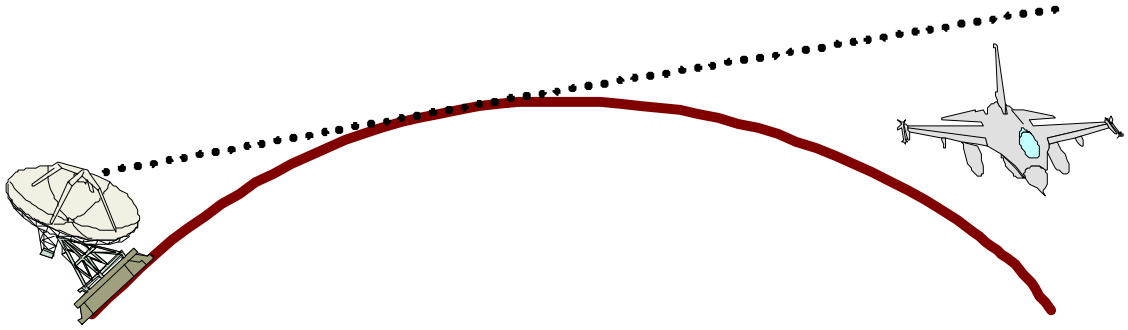


Figure 146 : Le radar ne peut voir l'avion sous l'horizon

Si la précision requise pour le terrain est grande, on peut également tenir compte du fait que la terre n'est pas ronde : elle n'est même pas régulière[106], et ne peut être représentée par un volume classique, c'est pourquoi on parle de « géoïde ». Toutefois, quand la précision requise n'est pas excessive, on assimile la terre à un ellipsoïde dit « de référence », pour lequel on définira un facteur d'aplatissement (voir Figure 147 et Figure 148). Le système WGS84[107] (*World Geodetic System*) prend comme valeur $1/f = 298{,}257\ 223\ 563$.

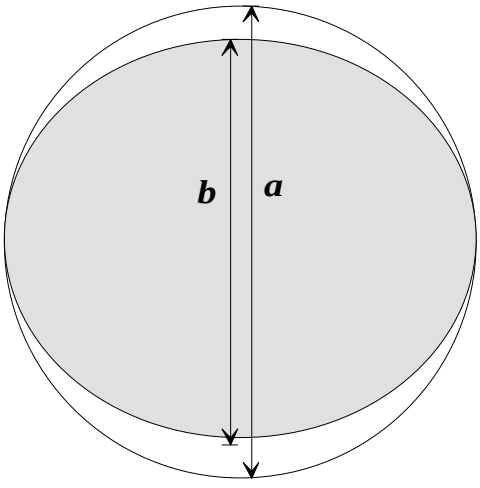


Figure 147: Ellipsoïde de référence : f = (a-b)/a

Les écarts entre un ellipsoïde et une approximation sphérique peuvent paraître minimes, mais sont suffisants par exemple pour qu'un véhicule terrestre se mette à voler si l'on connecte deux simulations qui n'ont pas le même système de référence…

[106] Elle est notamment aplatie aux pôles, avec un facteur d'aplatissement $f \approx 0{,}34\%$: le rayon polaire est 21 385 km plus petit que le rayon équatorial.

[107] Le WGS84 est très utilisé dans le monde de la simulation. C'est le système utilisé dans DIS, par exemple.

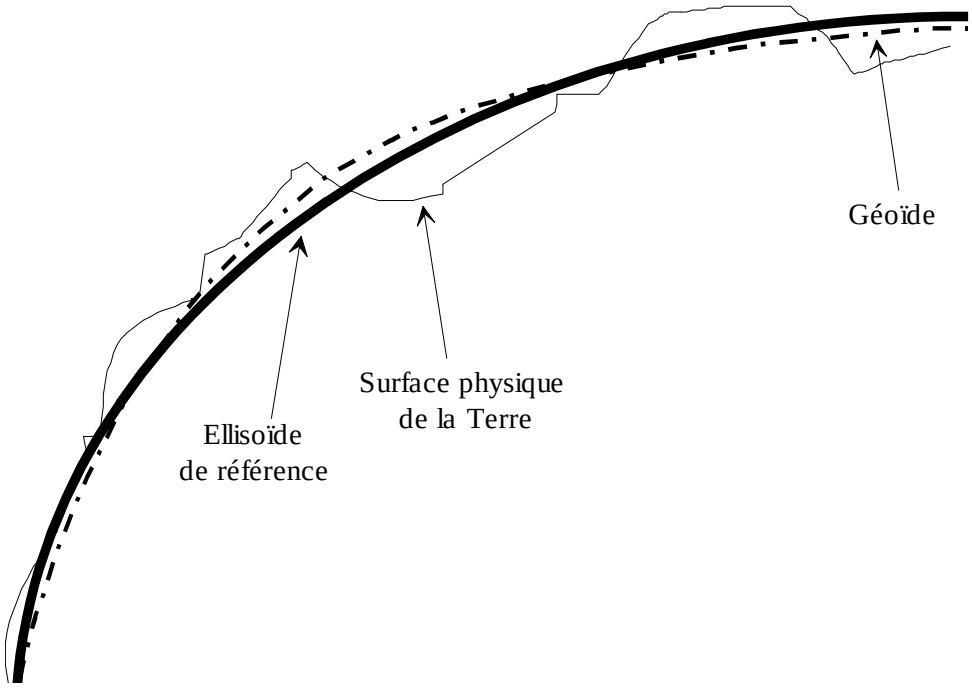


Figure 148: Ellipsoïde, géoïde et surface terrestre

9.3.5.2 Changement d'unités

Nous l'avons vu, les américains ont tendance à travailler avec les unités « impériales » (mile, pied, once…) et les européens en système métrique (mètre, kilogramme…), d'où des confusions possibles.

Une solution adoptée par les systèmes de simulation comme ESCADRE ou SOFIA est de travailler en interne en système MKSA[108] (en utilisant les unités internationales de base), et en appliquant systématiquement un coefficient pour convertir vers (pour l'affichage) ou depuis (pour lire les données) l'unité que l'utilisateur souhaite utiliser. Une bonne pratique est, dans les fichiers de données, de préciser l'unité à chaque fois.

9.3.5.3 Coordonnées polaires ou cartésiennes

Les coordonnées d'un point sur le globe peuvent être données soit en coordonnées polaires (longitude j, latitude l, altitude h) soit en coordonnées cartésiennes (x, y, z). Les changements de repères sont par conséquent des fonctions de base de toute simulation traitant des données géographiques (voir par exemple le cas de l'API SEDRIS, page 238 et dans [DMSO98]).

[108] Mètre, Kilogramme, Seconde, Ampère.

9.3.5.4 Projections cartographiques

Une carte, qu'elle soit sur un papier ou sur un écran d'ordinateur, est plate. La Terre, en revanche, est sphérique (en première approximation !). Or, il n'est pas possible de projeter la surface d'une sphère sur un plan sans provoquer des déformations.

C'est pourquoi les cartographes sont obligés de faire des compromis lorsqu'ils choisissent un système de projection, en fonction de la destination de la carte, de la précision requise, de la zone couverte et de la latitude.

L'une des plus utilisées est la projection cylindrique, obtenue en projetant la surface du globe terrestre sur un cylindrique dont l'axe est soit parallèle (aspect direct) soit perpendiculaire (aspect transverse, voir Figure 149) à l'axe de rotation de la terre. Cette projection est dite conforme lorsque les parallèles s'écartent de plus en plus pour compenser les défauts de linéarité. Une projection cylindrique conforme[109] est dite de Mercator.

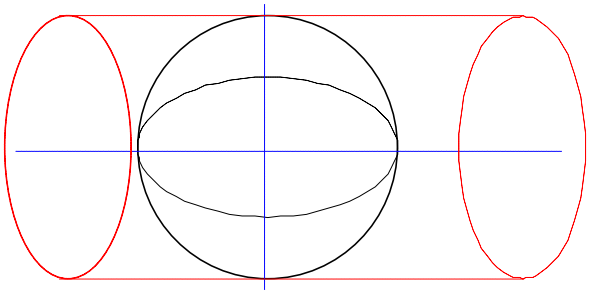


Figure 149: Projection cylindrique transverse

La représentation **UTM** (***Universal Transverse Mercator***) est une représentation de Mercator transverse limitée à 3° d'amplitude de part et d'autre du méridien central. La Terre est ainsi découpée en 60 faisceaux de 6°. Le fuseau n°31 commence au méridien de Greenwich (λ=0°), le n°32 à λ = 6° Est, etc.

Les coordonnées UTM utilisent ce système de projection. Les militaires se basent sur ces coordonnées UTM en définissant un quadrillage, avec des carreaux de 100 km de côté.

L'IGN se base sur une projection conique conforme, dite de Lambert, qui donne une bonne précision pour les latitudes métropolitaines. La France est divisée en 4 zones (Nord, Centrale, Sud et Corse) pour lesquelles sont utilisées des valeurs optimales de φ_0.

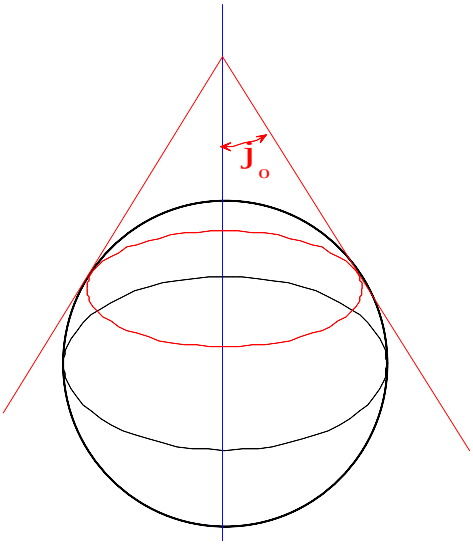


Figure 150: Projection de Lambert

Le passage d'une de ces représentations à l'autre n'est pas immédiat, car il implique des calculs trigonométriques qui, s'ils ne sont pas très compliqués, entraînent une charge importante pour l'ordinateur, lorsqu'ils sont appliqués à des millions de coordonnées par seconde.

[109] Une projection est dite **conforme** si elle conserve les angles.

Si les projections conformes sont favorisées par les bases de données géographiques, les cartes peuvent utiliser des projections équivalentes[110] : projection Albers, très utilisée pour les cartes au 1/25 000 aux USA, projection de Peters, projection gnomonique, projection d'Eckert…

Bien évidemment, il n'y a pas un système standard unique pour toutes les applications, ce qui implique souvent des travaux d'écriture de filtres de conversion d'un système de coordonnées à l'autre.

SEDRIS, pour pouvoir prétendre à l'universalité, a dû supporter un nombre impressionnant de systèmes de coordonnées, comme le montre la Figure 151 (SM et GEI ne sont pas pris en compte actuellement dans SEDRIS 3.0, mais devraient l'être dans une prochaine version de SEDRIS). SEDRIS est en outre capable de convertir n'importe quelle coordonnée d'un système vers l'autre.

Arbitrary	**LSR**	Local Space Rectangular Coordinate System
Earth-Surface, Global	**GDC**	Geodetic Coordinate System
Earth-Centered, Earth-Fixed	**GCC**	Geocentric Coordinate System
Earth-Surface, Projected	**M**	Mercator PCS
	OM	Oblique Mercator PCS
	PS	Polar Stereographic Projected Coordinate System (PCS)
	LCC	Lambert Conformal Conic PCS
	TM	Transverse Mercator PCS
	UTM	Universal Transverse Mercator PCS
Earth-Surface, Local (Topo centric)	**GCS**	Global Coordinate System
	LTP	Local Tangent Plane Coordinate System
Earth-Centered, Rotating (Inertial & Quasi Inertial)	**GM**	Geomagnetic Coordinate System
	GEI	Geocentric Equatorial Inertial Coordinate System
	GSE	Geocentric Solar Ecliptic Coordinate System
	GSM	Geocentric Solar Magnetospheric Coordinate System
	SM	Solar Magnetic Coordinate System

Figure 151: Systèmes de coordonnées supportés par SEDRIS

9.3.6 Formats et standards les plus utilisés

Je n'ai pas l'ambition d'être exhaustif ici, car il existe une multitude de standards différents, et j'en découvre régulièrement. Il s'agit surtout de montrer la diversité et le besoin de rationalisation, et l'espoir que peut susciter SEDRIS dans ce domaine.

9.3.6.1 DLMS

Le *Digital Land Map System* a été créé en 1976 par l'Italie, la France, l'Allemagne et le Royaume-Uni, pour les besoins de préparation de missions du chasseur-bombardier Tornado. Les données DLMS (élévation et éléments topographiques) sont régulièrement

[110] Les surfaces projetées sur la carte sont proportionnelles aux surfaces correspondantes sur la sphère.

mises à jour par un groupe multinational, d'échange de données topographiques numérisées (GMEDTN).

9.3.6.2 OTAN

Le standard OTAN (STANAG) 7074 intitulé *Feature and Attribute Coding Catalog* (FACC) donne une typologie des éléments de l'environnement et des attributs à prendre en compte. Il a été développé par le *Digital Geographic Information Working Group* (DGIWG), lequel a également produit la norme DIGEST (*Digital Geographic Exchange Standard*), permettant l'échange de données vectorisées (éléments topographiques) et maillées (données d'élévation).

Notons que l'un des documents du standard IEEE 1278.1 (DIS, voir page 248), qui est devenu un STANAG OTAN, est une énumération des objets que l'on peut intégrer dans une simulation, ceci comprenant des éléments d'environnement.

9.3.6.3 DTED

Les militaires américains ont développé le standard DTED (*Digital Terrain Elevation Data*), utilisé pour les données produites par la National Imagery and Mapping Agency (NIMA), que l'on peut utiliser en combinaison avec les standards de données topographiques DFAD (*Digital Feature Analysis Data*) et ITD (*Interim Terrain Data*).

Le DTED est constitué d'une grille de données altimétriques pour une zone géographique. Il est divisé en plusieurs niveaux, suivant la précision (voir tableau page 231). Les plus courants sont le DTED 1 (maillage d'environ 100 m) et le DTED 2 (maillage de 1×1 seconde d'arc entre 50° de latitude sud et 50° de latitude nord, 1×2 secondes d'arc ailleurs, ce qui donne une maillage de l'ordre de 30 m).

L'ITD, un format initialement censé être temporaire, décrit les éléments du terrain classées en 6 catégories (végétation, composition du sol, pente, drainage, obstacles, voies de transport). Les éléments peuvent être de type ponctuel, linéaire ou surfacique, et comportent de nombreux attributs.

Le DFAD est similaire à l'ITD, mais est mieux adapté à la description des zones urbaines.

9.3.6.4 SIF

Le standard SIF (Standard Simulator Database Interchange Format) fut élaboré par l'US Air Force, dans le cadre du projet 2851, afin de tenter de résoudre le problème d'échange de données. Ce projet engendra deux standards militaires : MIL-STD 1821 (SIF) et MIL-STD 1820 (GTDB, *Generic Transformed Data Base*).

Par la suite, l'US Army l'étendit pour les besoins de son entraîneur tactique CCTT (Close Combat Tactical Trainer), pour obtenir la spécification dite SIF++.

Mais tous les deux sont handicapés par l'accent qu'ils mettent sur le format des données plutôt que leur représentation, ce qui grève leur souplesse. De plus, de par leur origine, ils sont fortement orientés vers la simulation virtuelle (plates-formes pilotées). Ils devraient céder peu à peu la place à SEDRIS (voir §9.3.7).

Pour les besoins de construction d'une base de données française pour les simulateurs de Mirage 2000D, réutilisable avec d'autres types d'avions, il fut créé un format dérivé de SIF, SIF-France, propriété de la DGA et compatible avec le format d'origine. Il

comporte quelques améliorations, notamment il est plus précis sur certains aspects visuels.

9.3.6.5 Autres formats

Il existe d'autres formats de données qui sont utilisés pour décrire des cartes ou des données d'environnement :

Formats vectoriels :

- Militaires : VMAP, MIL-STD-2407, Vector Product Format (VPF), DNC...
- Civils : SHP, DXF...

Formats raster/bitmap :

- Militaires : MIL-STD-2411A, Raster Product Format (RPF), ASRP, CADRG, CIB, NITF, CRP...
- Civils : TIFF, JPEG, GIF, BMP...

Formats matriciels (maillages) :

- Militaires : DTED...
- Civils : GRID...

Autres formats d'échanges de données :

- Data Interchange Format
- Geospatial Data Interchange
- Atmospheric & Oceanographic Data Interchange - WMO Standards
- ...

9.3.7 SEDRIS

SEDRIS est aux bases de données d'environnement (BDE) ce que HLA[111] est aux simulations : un standard d'interopérabilité universel, destiné à faciliter l'échange, sans perte d'information, de données d'environnement entre des applications très différentes.

SEDRIS est un des éléments devant permettre, aux yeux du DoD américain, d'obtenir des environnements synthétiques de haute qualité. En effet, HLA est une condition nécessaire mais non suffisante d'interopérabilité des systèmes de simulation : il faut également s'assurer, entre autres choses, que les différents composants d'une fédération partagent le même environnement.

Or, nous l'avons vu, il existe nombre de formats de données et de modèles d'environnement. Jusqu'ici, lorsque deux simulations devaient partager leur environnement, il fallait écrire un filtre permettant de passer d'une représentation à l'autre, ou alors modifier l'une des deux simulations. C'est encore jouable… Mais si on veut créer des environnements synthétiques complexes, avec 3, 4, …, N applications, il faut autant de filtres de conversion que de flux d'information possibles, soit jusqu'à N^2 conversions (voir Figure 152). Une solution serait d'avoir un modèle d'environnement unique. Mais le problème est que la simulation, comme nous l'avons largement rappelé, est très diversifiée, et les besoins varient d'un domaine à l'autre, de sorte qu'il est illusoire de

[111] Voir §9.5.4.3.

penser définir un format de données figé et universel pour les données d'environnement[112].

SEDRIS a l'ambition de satisfaire les besoins de tous les types d'applications de simulation (virtuelles, constructives, CGF…) en données d'environnement terrestre, atmosphérique, océanique et même spatial. Ce standard est composé principalement :

- D'un modèle de représentation de données commun (DRM) et d'un dictionnaire de données associé.
- D'une spécification d'interface.

Le modèle de données est en fait un méta-modèle de données, permettant à l'utilisateur de décrire ses propres données. Une originalité de SEDRIS est de permettre le polymorphisme des données, c'est-à-dire que l'information issue d'une description d'un élément pourra varier suivant le contexte. Ainsi, le nombre de dimension décrites pour un bâtiment pourra être de deux si la donnée est utilisée par *Plan view display* (affichage d'une situation tactique sous forme de carte plane), ou de trois pour une visualisation 3D.

L'interface de programmation (API) de SEDRIS, quant à elle, permet d'accéder en lecture et écriture[113] au médium de transmission SEDRIS. L'interface répond en tenant compte du contexte de l'application, et est capable, entre autres, d'effectuer des conversions de coordonnées. Cette API permet de découpler l'application de la base de données, l'architecture de cette dernière devenant alors transparente pour l'application. En outre, cela permet d'envisager plus facilement la réutilisation de la base de données pour une autre applications, ou le fonctionnement de l'application avec une autre base de données.

SEDRIS fournit également un format de transmission de données (STF), qui est indépendant de la plate-forme et permet l'échange de fichiers de données d'environnement entre différents systèmes.

La Figure 153 illustre bien le rôle de SEDRIS dans les échanges de données, et la simplification qu'il apporte (on passe d'un nombre de liaisons $n(n+1)/2$ à $2n$). Le parallèle avec HLA, le grand standard d'interopérabilité des simulations, est assez aisé à faire, tant dans l'architecture que dans les objectifs…

[112] L'une des raisons de la disgrâce du protocole DIS pour l'interopérabilité des simulations est justement qu'il se voulait universel, et codifier tous les objets pouvant évoluer au sein d'une simulation (document « DIS Enumeration »). Évidemment, la liste s'allongeait au fil des ans et il devint évident que le but ne serait jamais atteint. SIF et SIF++ subirent des critiques similaires.

[113] Actuellement, cette possibilité n'est exploitée que de façon statique, notamment en raison des performances limitées des systèmes actuels, mais à terme il est prévu d'utiliser SEDRIS dynamiquement : une application pourra ainsi modifier l'environnement de toute une fédération de simulations, par exemple en détruisant un pont sur une coupure topographique (rivière…).

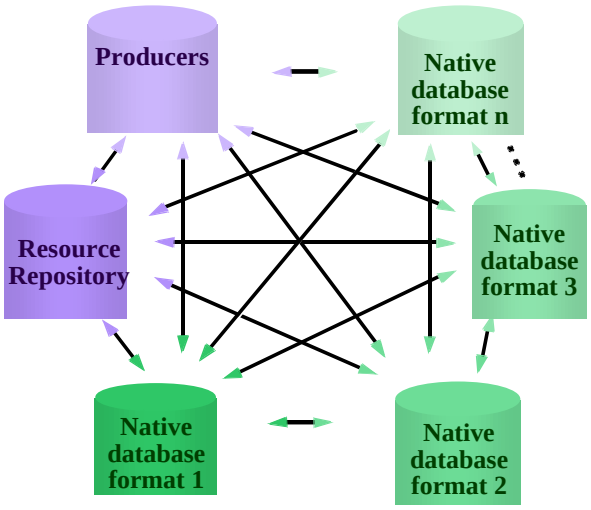


Figure 152: Avant SEDRIS, il fallait prévoir un convertisseur entre chaque BDE

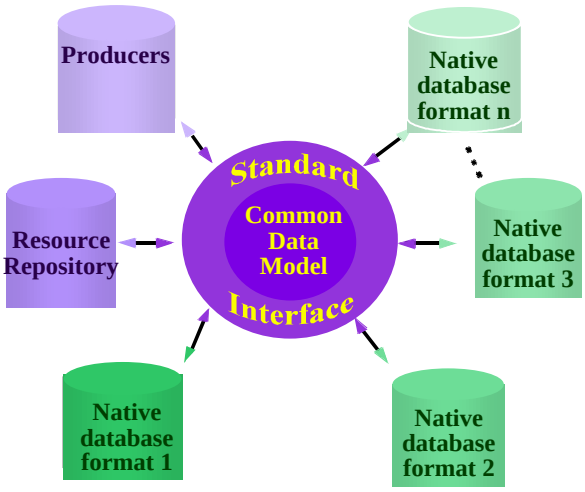


Figure 153: SEDRIS fournit une interface commune à toutes les sources de données d'environnement

9.4 Automatisation des acteurs

9.4.1 CGF et SAF

9.4.1.1 Définitions

Les **Computer Generated Forces** (forces synthétiques) sont des entités d'une simulation contrôlées par l'ordinateur, utilisant un modèle de comportement humain assez sophistiquées pour qu'elles puissent entreprendre des actions de façon autonomes (i.e. sans humain dans la boucle).

On les appelle aussi SAF (Semi-Automated Forces), bien que ce dernier terme traduit une autonomie partielle des entités (par exemple, un opérateur doit leur donner des directives de temps à autre, car leur initiative est réduite).

9.4.1.2 Utilisations

Les CGF représentent la composante comportementale et décisionnelle humaine. Elle sont essentielles dans toute simulation faisant intervenir une telle composante, notamment pour l'environnement tactique d'une simulation, qu'elle soit militaire (jeu de guerre…) ou civil (comportement des voyageurs dans une gare…).

Les CGF peuvent travailler au niveau pion individuel (par exemple, dans un jeu de guerre tactique, simulation d'un combattant ou d'une plate-forme et de son équipage) ou agrégé (capacité à simuler des décisions au niveau du commandement d'une section, d'une compagnie, etc.). Les modèles de comportement humain prennent en compte la doctrine (règles de déplacement, d'engagement...), et peuvent ou non tenir compte des facteurs humains (peur, stress, moral...). Ce dernier point est encore aujourd'hui mal traité, car il est difficile de valider des modèles de facteurs humains (comment valider un modèle émotionnel ?), mais il est clair qu'il représente un enjeu important dans la mesure où ces facteurs, dans la réalité, peuvent influer considérablement l'issue des combats, particulièrement à des niveaux tactiques. Notons que, dans le civil, il existe des simulations de comportement humain, par exemple pour le marketing, qui prennent en compte ces facteurs humains.

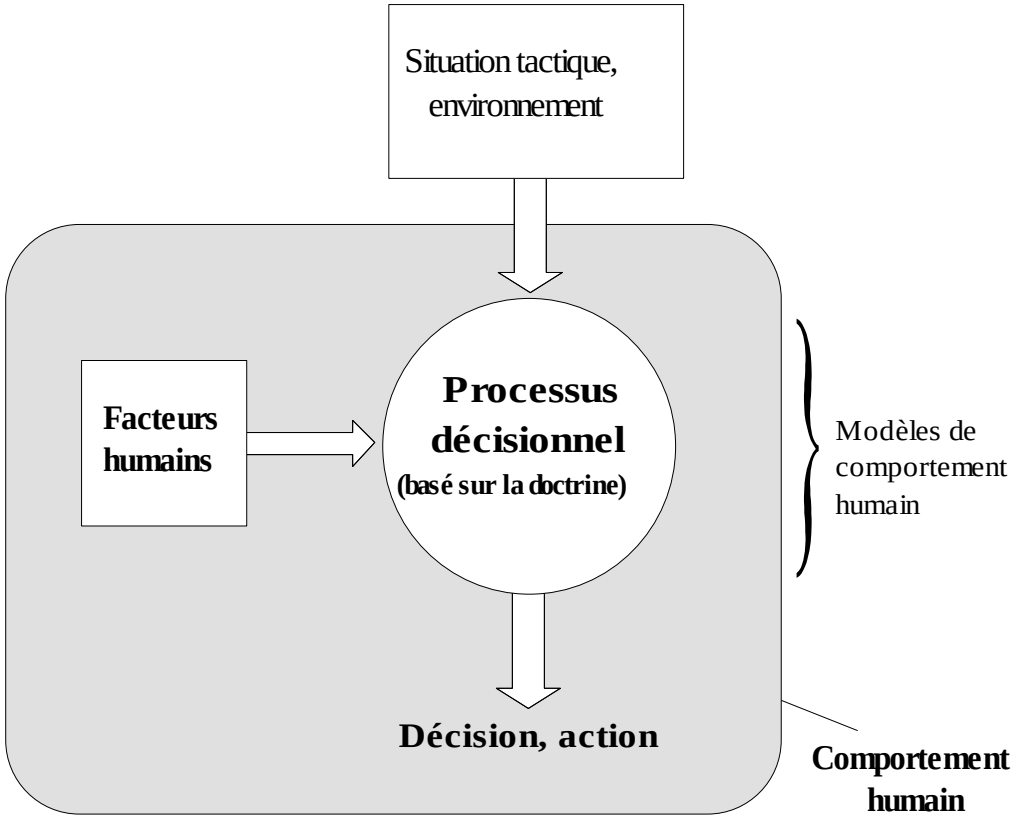


Figure 154: Modélisation du comportement humain

Les forces synthétiques permettant de simuler les décisions humaines, elles autorisent l'automatisation de la simulation. Par exemple, dans un jeu de guerre destiné à l'entraînement d'un état-major, le ou les entraînés jouent le camp ami (traditionnellement de couleur bleu), tandis qu'une cohorte d'officiers (parfois plusieurs dizaines ou centaines) commandent les unités ennemies et les alliés qui doivent réagir aux ordres ou qui ne sont pas directement sous le commandement de l'entraîné. Ceci est particulièrement lourd, aussi cherche-t-on à réduire le nombre de ces officiers, qui constituent ce que l'on appelle une **cellule réponse**, en leur fournissant des outils automatisant leur travail. Notons que pour un niveau d'entraînement *N* (par exemple, pour l'armée de terre, division/brigade), les cellules réponses seront au niveau *N-1* (par ex. régiment) et les CGF au niveau *N*-2 (par ex. compagnie/section).

9.4.1.3 Exemple de CGF : ModSAF

ModSAF (*Modular Semi-Automated Forces*) est un ensemble de modules logiciels et d'applications qui permettent de construire des simulations intégrant des CGF. Son développement a été financé par le DoD américain, plus exactement par l'Army Simulation, Training and Instrumentation Command (STRICOM). C'est un des CGF les plus populaires dans la simulation de défense, et il est très utilisé non seulement dans l'US Army, mais aussi dans les autres armées américaines (Navy, Marines Corps, Air Force) et dans plusieurs pays étrangers (Royaume-Uni, France...).

ModSAF comprend de nombreux types d'entités pouvant être gérées par l'ordinateur : véhicules terrestres et aériens, infanterie à pied, missiles, structures dynamiques, etc.

Ces entités peuvent interagir entre elles, mais aussi avec des entités d'autres simulations via les standards DIS ou HLA. Des comportements représentés comprennent le mouvement, le tir, la perception, la réaction à l'environnement tactique, la communication, etc. La fidélité du comportement des entités est sélective, de façon à pouvoir adapter le coût en puissance de calcul en fonction du nombre d'objets et des performances requises.

Cependant, ModSAF a de nombreuses limitations. Ainsi, les comportements inclus dans le produit ne sont pas toujours satisfaisants. ModSAF simule les événements et résout les combats à l'aide de modèles stochastiques. Les comportements, eux, sont déterministes (dans une situation initiale donnée, le comportement du CGF sera toujours le même). Il n'y a pas de prise en compte des facteurs humains, de la psychologie du combat, de sorte que les décisions sont très « mécaniques ».

Il est possible de programmer ses propres comportements à l'aide d'une machine à états finis, ce qui n'est pas simple mais rend le produit ouvert.

Toutefois, dans la mesure où les comportements sont définis par des automates à états finis, il est difficile d'avoir des modèles de comportement globaux (i.e. de haut niveau d'agrégation). De fait, ModSAF est utilisé pour modéliser des combattants et des plates-formes individuels et un opérateur est nécessaire pour les comportements de plus haut niveau. Cet opérateur devra alors avoir une connaissance pointue de toutes les entités qu'il contrôle ainsi de des organisations qu'il est censé simuler par ses actions.

Un CGF de nouvelle génération, OneSAF, devrait remplacer d'ici quelques années ModSAF, avec des capacités plus étendues (modélisation à des niveaux régiment/brigade, plus grande automatisation, meilleure modularité, meilleure interopérabilité).

ModSAF et OneSAF peuvent se connecter à d'autres simulations, en utilisant des standards tels que DIS ou HLA.

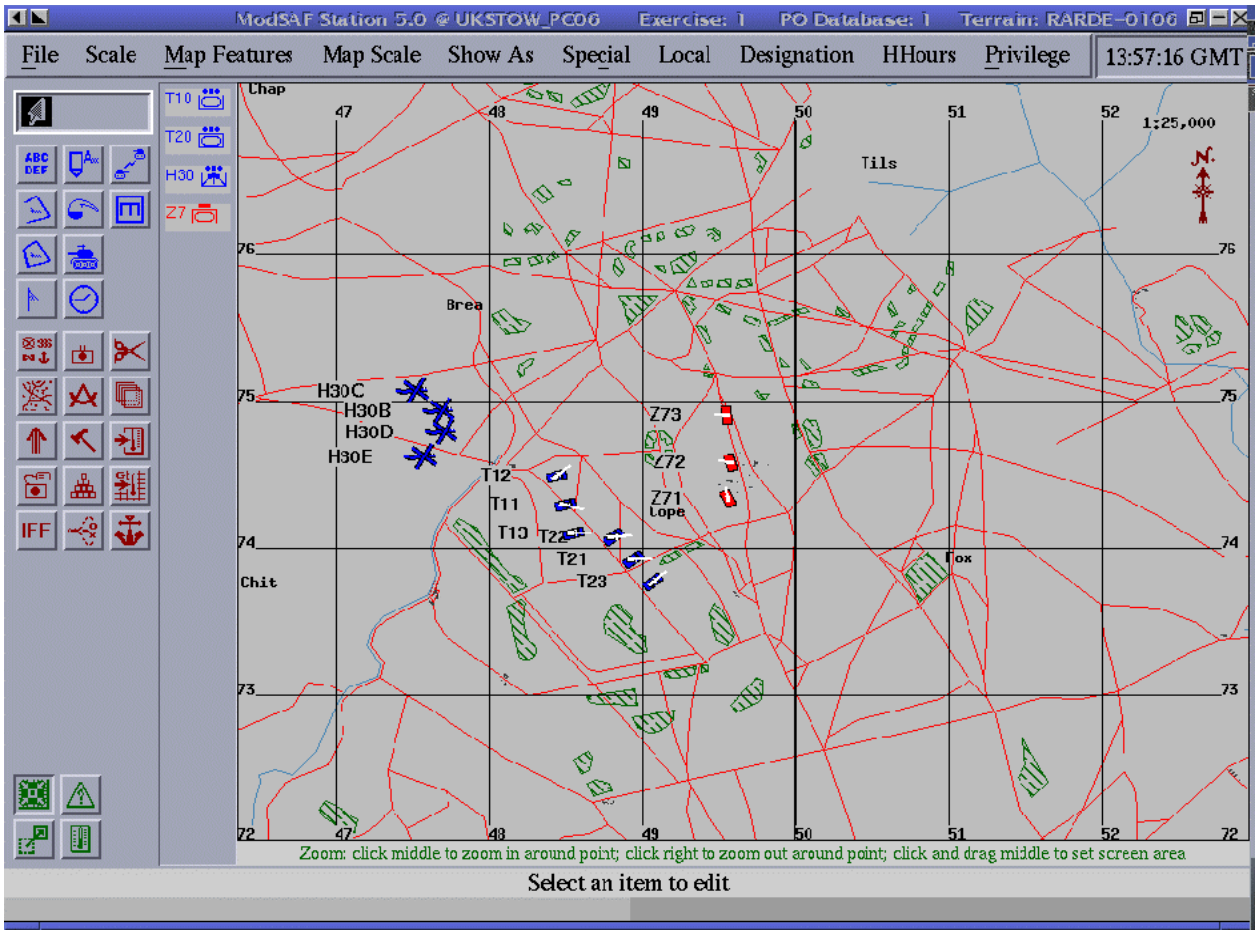


Figure 155 : Copie d'écran ModSAF 5.0 (d'après DERA)

9.4.1.4 Performances présentes et futures

Ces CGF, on s'en doute bien, font appel à des technologies de pointe, particulièrement dans le domaine de l'intelligence artificielle. De nombreuses approches différentes existent, et des dizaines de produits existent sur le marché (ModSAF, Sequoia, SETHI, ...).

Toutefois, il est vrai qu'on en est encore qu'aux balbutiements et que ces produits sont, pour la plupart décevants, qu'ils soient civils ou militaires[114]. En fait, ils sont souvent limités, soit en réalisme ou possibilités, soit en domaine d'application. L'enjeu est toutefois suffisamment important pour que de nombreux programmes de recherches soient menés.

Dans les années à venir, les CGF devraient être un domaine très dynamique de la simulation. Voici quelques-unes des qualités qui seront requises pour ces produits :

- Autonomie : un grand nombre d'entités doivent pouvoir accomplir les missions que l'opérateur leur a assignées au départ sans que celui-ci n'ait à intervenir durant l'exécution. Chaque entité synthétique devra pouvoir

[114] L'auteur de ces lignes a néanmoins vu des choses fort intéressantes dans le domaine du jeu vidéo. Peut-être qu'un jour certains généraux utiliseront le même moteur d'IA sur leur station de travail que leur petit-fils sur sa console de jeux !

réagir en temps réel aux informations qui lui parviennent, et de replanifier éventuellement le déroulement de sa mission si le contexte le nécessite.

- Utilisation du terrain : déplacements à couvert dans les zones dangereuses, choix d'une position de tir où l'on peut se dissimuler, etc.
- Comportements collectifs : l'entité doit pouvoir agir en groupe, en se coordonnant avec les autres entités du groupe, et en ayant « conscience » de l'impact de ses actions sur elles. Les décisions et comportements individuels doivent pouvoir être agrégés en comportements collectifs (i.e. d'une entité de plus haut niveau) et, inversement, une décision de haut niveau doit pouvoir être désagrégée en décisions individuelles adaptées[115].
- Psychologie : les entités synthétiques devront pouvoir, à terme, prendre en compte les facteurs humains (voir § 9.4.2) de ses composants et des entités de l'adversaire. Les forces synthétiques ne seront plus des robots sans émotions. Ainsi, une unité ayant subit de lourdes pertes peut décider de se retirer, voire de se rendre, plutôt que de combattre jusqu'au dernier homme comme on le voit souvent. De même, une unité devant mener une action offensive pourra rechercher l'effet de surprise et la panique et la désorganisation qui peuvent alors se développer chez l'adversaire.

Rendez-vous d'ici quelques années donc...

9.4.2 Facteurs humains

Quand on modélise la composante humaine dans un système, il n'y a pas que les aspects cognitifs et décisionnels. Il y a également les **facteurs humains**, c'est-à-dire l'influence de la réponse physiologique ou psychologique de l'homme à son environnement : stress, fatigue, faillibilité, motivation...

Là encore, on commence à disposer de certains travaux de modélisation, mais les facteurs humains ne sont pas encore très développés dans les simulations, car leur quantification est très délicate et subjective. En outre, la validation des modèles de facteurs humains n'est guère aisée.

Néanmoins, on trouvera particulièrement de tels modèles dans des simulations réalisées dans le cadre d'études d'ergonomie (adaptation d'un poste de travail en fonction du stress et de la fatigue provoquée...).

[115] Ceci peut être complexe, notamment à un très haut niveau (commandants de théâtre, organisations, etc.), car il est alors nécessaire de faire preuve d'une grande abstraction dans le traitement des informations montantes. Cette abstraction implique une compréhension globale de la situation, ce qui va au-delà de la simple fusion de comportements ou d'informations.

9.5 Simulation distribuée

9.5.1 Les enjeux

Nous avons déjà eu l'occasion de rappeler à plusieurs reprises l'intérêt de la simulation pour résoudre des problèmes complexes ou pour effectuer des entraînements avec un haut degré de réalisme.

Jusqu'au milieu des années 80, le modèle d'architecture informatique qui prévalait était celui du supercalculateur, desservant des terminaux clients passifs, comme le montre la Figure 156. Les terminaux ne servent qu'à l'affichage et à l'entrée des données par l'utilisateur, tous les traitements étant réalisés au niveau du calculateur central.

Un tel système avait ses avantages, tel qu'une administration simplifiée et une bonne sécurisation. Malheureusement, ces « monstres » qu'étaient les gros calculateurs coûtaient fort cher (parfois des dizaines de millions de francs). L'apparition des mini-ordinateurs, puis des stations de travail et des micro-ordinateurs, permit l'apparition d'un nouveau modèle, où les traitements sont distribués sur un réseau (de type « bus » ou « anneau ») où toutes les machines sont au même niveau (Figure 157).

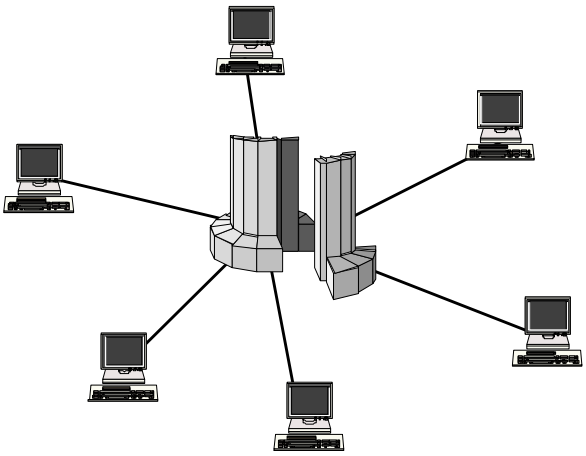


Figure 156: Informatique centralisée

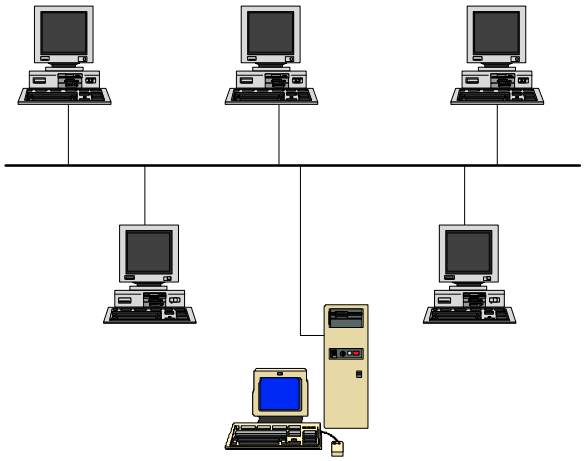


Figure 157: Informatique distribuée

Cette architecture permettait d'effectuer les traitements sur des machines bien moins coûteuses. Les serveurs diminuèrent en taille, leur tâche se limitant à des services de fichiers, de bases de données ou d'impression, et à la gestion de la sécurité.

De nombreux scientifiques mirent au point des systèmes permettant d'obtenir de grandes puissances de calcul par la mise en commun des ressources de plusieurs stations de travail sur un réseau[116]. Un exemple flagrant est la compétition permanente que se livrent diverses équipes sur Internet pour « casser » des codes de chiffrement (DES,

[116] Un célèbre fabriquant de stations de travail avait imaginé le slogan : « le réseau est l'ordinateur ».

RSA…) en utilisant des centaines, voire des milliers d'ordinateurs, répartis sur toute la planète[117].

La simulation, jusqu'au début des années 90, était basée elle aussi sur un modèle centralisé. Il n'était pas rare de voir un jeu de guerre monopoliser un supercalculateur ! Mais la complexité des simulations et les exigences des utilisateurs étant sans cesse croissants, et les coûts des réseaux et des stations de travail diminuant, les ingénieurs finirent par réfléchir à une nouvelle architecture pour leurs simulations, dans laquelle le système de simulation est la réunion du potentiel de plusieurs applications, réparties sur différents ordinateurs.

9.5.2 Historique

Pendant longtemps, les simulations étaient des programmes monolithiques, bien qu'il arrivât que certaines formes simples de distribution soit utilisées, par exemple le moteur sur un calculateur, et l'affichage graphique sur différentes stations de travail. Ou un camp d'un jeu de guerre sur un ordinateur, et l'autre camp sur un autre.

Ce sont une fois de plus les américains, toujours très à la pointe en matière de virtuel, qui fait le premier pas avec le projet SIMNET.

SIMNET (*Simulation Network*) fut lancé dans les années 80 par la DARPA[118]. Le modèle était distribué : il n'y avait pas de système central, mais une fédération de simulateurs, chacun gérant son état (dommages…) et ses ressources (carburant, munitions…) et communiquant avec les autres simulateurs à l'aide d'un protocole basé sur des messages véhiculés par un réseau informatique.

SIMNET devait permettre à des simulateurs pilotés (chars notamment) de manœuvrer sur un même terrain virtuel tout en se trouvant dans des lieux distincts. On pouvait ainsi s'entraîner à manœuvrer en pelotons, ce qui correspondait mieux à la réalité.

SIMNET avait, certes, de nombreuses imperfections, car la simulation distribuée est une technologie très complexe, longue à maîtriser. Il doit être vu comme un « laboratoire » qui permit aux américains de passer à l'étape suivante, DIS.

9.5.3 Qu'est-ce que l'interopérabilité ?

Avant de poursuivre, il importe de clarifier un vocabulaire de l'interopérabilité, comportant niveaux, chacun ayant ses propres termes. Bien entendu, il est courant de jouer sur ces mots pour présenter sous un jour plus favorable une solution technique de simulation distribuée.

9.5.3.1 Conformité

Une simulation distribuée est dite **conforme à un standard** (*compliant* en anglais) si elle implémente ce standard en respectant entièrement ses exigences.

[117] Je tiens à préciser ici, bien que nous y reviendrons, que la distribution n'est pas une panacée : en effet, elle peut également s'avérer coûteuse (gestion des communications, synchronisation des processus, etc.). C'est pourquoi un de mes collègues disait souvent : « la distribution n'est pas une fin en soi, c'est une contrainte ».

[118] Defense Advanced Research Project Agency, finançant de nombreuses recherches militaires et civiles. C'est la DARPA qui, par exemple, fut à l'origine du réseau Internet (qui débuta sous l'appellation « ARPANET »).

Ainsi, on dira qu'une simulation est « conforme au standard HLA » (*HLA compliant*). On fera ainsi, pour vérifier cette propriété, des **tests de conformité** à HLA.

9.5.3.2 Compatibilité

On parlera de **compatibilité avec un protocole** lorsque cette simulation implémente un protocole d'échange lui permettant de dialoguer directement avec une autre simulation à l'aide de ce protocole.

Ainsi, on dire qu'une simulation est compatible avec le protocole DIS. Notons que cela signifie également que cette simulation est conforme au standard DIS[119].

En revanche, parler de compatibilité HLA n'a pas de sens, puisque ce standard, comme nous le verrons par la suite, ne spécifie pas de protocole de communication entre les simulations, mais uniquement des services. Contrairement à DIS, faire coopérer effectivement deux simulations HLA nécessite de modifier ou de reconfigurer l'application (nouvelle édition de lien pour intégrer un nouveau RTI, création d'une FOM...).

9.5.3.3 Interopérabilité

> Capacité d'une simulation (ou d'un modèle) à fournir et à accepter des services d'autres simulations et à utiliser les services ainsi échangés pour leur permettre de fonctionner effectivement ensemble pour parvenir à un objectif donné (en terme d'étude, d'entraînement, etc.)[120].

Ainsi, une simulation conforme à HLA n'est pas forcément interopérable avec une autre simulation conforme à HLA : il faut qu'elles utilisent le même RTI (voir page 254), qu'une FOM commune puisse être écrite, qu'elles partagent la même vision de leur environnement naturel, etc.

On peut considérer la hiérarchie suivante :

Interopérabilité > Compatibilité > Conformité

La Figure 158 donne une définition de niveaux supplémentaires des simulations utilisées dans le cadre du programme de simulation CCTT (*Close Combat Tactical Trainer*) de l'US Army.

La méthode MADIOS v1.0, démarche visant à uniformiser l'administration et l'interopérabilité des données opérationnelles des armées françaises, définit trois niveaux d'interopérabilité entre systèmes :

- le niveau opérationnel, correspondant aux procédures opérationnelles et à la signification opérationnelles des données (c'est un niveau sémantique, traité au niveau des applications de simulation) ;
- le niveau procédural, relatif à la façon d'échanger les données (c'est le niveau de HLA) ;

[119] Cette notion se retrouve couramment en informatique. Ainsi, un ordinateur est dit « compatible » avec un matériel ou un logiciel s'il peut le recevoir directement, sans modification.

[120] Cette définition est dérivée d'un document OTAN sur les systèmes d'information et de commandement. L'ISO a quant à elle la définition suivante : « [l'interopérabilité est] la possibilité de communication, d'exécution de programmes ou de transfert de données entre unités fonctionnelles différentes, de telle manière que l'utilisateur n'ait que peu ou pas de besoin de connaître les caractéristiques propres à chaque unité ».

- le niveau technique, relatif aux couches de communication support des échanges (c'est le niveau pris en charge par le RTI).

Interoperability Level	Definition
Non-invasive:	A simulation/simulator system is said to be non-invasive if it is able to operate on the local area network (LAN) in the same exercise with the CCTT system without degrading the performance of the CCTT system.
Compliant:	A simulation/simulator system is said to be compliant if it is non-invasive and it implements the Distributed Interactive Simulation (DIS) protocols in accordance with the IEEE Standard 1278.1-1995. A specific compliance determination must be made regarding each Protocol Data Unit (PDU) generated and interpreted by the simulation system.
Compatible:	A simulation/simulator system is said to be compatible with CCTT if (1) it is compliant; (2) its models and databases send and interpret PDUs in support of the realization of a common synthetic environment (coherent in space and time); and (3) it is managed in a way that is consistent with CCTT.
Interoperable:	A simulation / simulator system is said to be interoperable with CCTT if it is compatible and, for a given exercise, its performance characteristics support the fidelity required for the CCTT interoperability exercise.
Fully Correlated:	A simulation / simulator system is said to be fully correlated if it is interoperable and provides identical representations in all aspects of the synthetic environment, data sets, and algorithms as CCTT.

Figure 158 : Niveaux d'interopérabilité définis pour le CCTT de l'US Army

L'interopérabilité des simulations est, depuis la fin des années 80, l'enjeu de vastes études et développements dans les pays fortement utilisateurs de simulations, et tout particulièrement aux États-Unis, donnant naissance à différentes solutions et standards, que nous allons maintenant passer en revue.

9.5.4 La course aux standards

9.5.4.1 DIS

En 1990, les américains lancèrent le projet DIS (*Distributed Interactive Simulation*). Cette fois, il s'agissait d'établir un protocole standard en utilisant une procédure ouverte, dev ant mener à un standard IEEE.

Deux fois l'an, des ateliers (*DIS Workshops*)étaient organisés à Orlando, en Floride, auxquels participaient environ un millier de personnes (chiffre de 1995), venus essentiellement des USA, bien entendu, mais aussi d'Europe, du Japon, d'Australie, etc., dénotant l'intérêt suscité par l'initiative du DoD.

Des groupes de travail rédigèrent les documents nécessaires, faisant grand usage d'Internet pour poursuivre les débats entre les ateliers[121]. DIS 1.0 fut publié en 1992 et standardisé peu après (IEEE 1278.1). Il devint par la suite un standard OTAN (STANAG 4482) et par là même était consacré comme LE standard de simulation distribuée chez les militaires occidentaux.

DIS est un **standard de protocole de communication** entre plusieurs simulations, lesquelles, devenant interopérables, peuvent partager leur environnement tactique et interagir. Il est basé sur l'échange de messages de bas niveaux (le standard précise le

[121] Ce qui était un bon exemple de « distribution du travail » !

codage des octets du message au niveau binaire), les PDU (*Protocol Data Unit*), que s'échangent entre elles les **entités**[122] de différentes simulations.

Ces messages peuvent servir à informer les autres entités de l'**état** d'une entité (*Entity State PDU*), ou effectuer une **interaction** (*Collision.*…). Afin de limiter le trafic sur le réseau, une entité n'envoie pas systématiquement une mise à jour de sa position. Elle ne le fait que si on lui demande ou que les paramètres de sa dynamique sont modifiés (par exemple si elle vire ou accélère) ou sont trop anciens. Entre deux mises à jour, sa position est extrapolée à l'aide d'un algorithme dit de ***dead reckoning***.

Il existe d'autres types de PDU, par exemple pour la gestion de la simulation (SIMAN, *Simulation Management* : démarrage/arrêt de l'exercice, création d'une entité, etc.), la logistique (*Resupply*), l'émission de signaux, les communications radio (encapsulation de paquets contenant une communication phonie numérisée…), etc.

Particulièrement destiné aux applications temps réel (simulateurs pilotés), il fut également utilisé pour de nombreux jeux de guerre tels que WARSIM, JANUS…

L'expérimentation STOW-E (*Synthetic Theater of War – Europe*), menée de 1994 à 1997, fournit une brillante démonstration de son potentiel. Cet exercice interarmées, faisant intervenir des simulateurs pilotés (simulation « virtuelle »), des jeux de guerre (simulation « constructive »[123]), des SIC et des unités sur le terrain (simulation « live »), comportait des dizaines de milliers d'entités, réparties géographiquement entre les USA, le Royaume-Uni et l'Allemagne.

Simple à mettre en œuvre et efficace, DIS fut assez populaire, bien qu'il ne réussit pas à s'imposer dans le civil.

9.5.4.2 ALSP

ALSP (*Aggregate Level Simulation Protocol*) est un protocole d'interopérabilité de simulations constructives non temps réel.

Ce protocole a été réalisé par la MITRE Corporation au début des années 90 pour le compte du STRICOM (*Simulation, Training and Instrumentation Command* de l'US Army).

Avec ALSP, les simulations sont des **acteurs** interconnectés au sein de **confédérations**. Ces simulations manipulent des entités appelées **objets**, échangeant des informations sur eux à travers une architecture logicielle, l'AIS (*ALSP Infrastructure Software*).

ALSP est un protocole de haut niveau, utilisant des messages textuels, ce qui est plus souple que les messages binaires de DIS, et facilite leur interprétation humaine, par exemple pour la mise au point des acteurs, mais alourdit considérablement le trafic réseau. Aucune hypothèse n'est faite sur les protocoles réseaux utilisés, la seule limitation étant leur implémentation au sein des composants logiciels de l'AIS. Les fédérations ALSP ont pu ainsi utiliser indifféremment les protocoles réseaux TCP/IP ou DecNET.

L'AIS comprend trois composants essentiels : le translator, l'ACM et l'ABE (voir Figure 159).

[122] Une entité est un objet de la simulation : char, avion, fantassin, pont…

[123] Notons que certains jeux de guerre participant à STOW utilisaient ALSP.

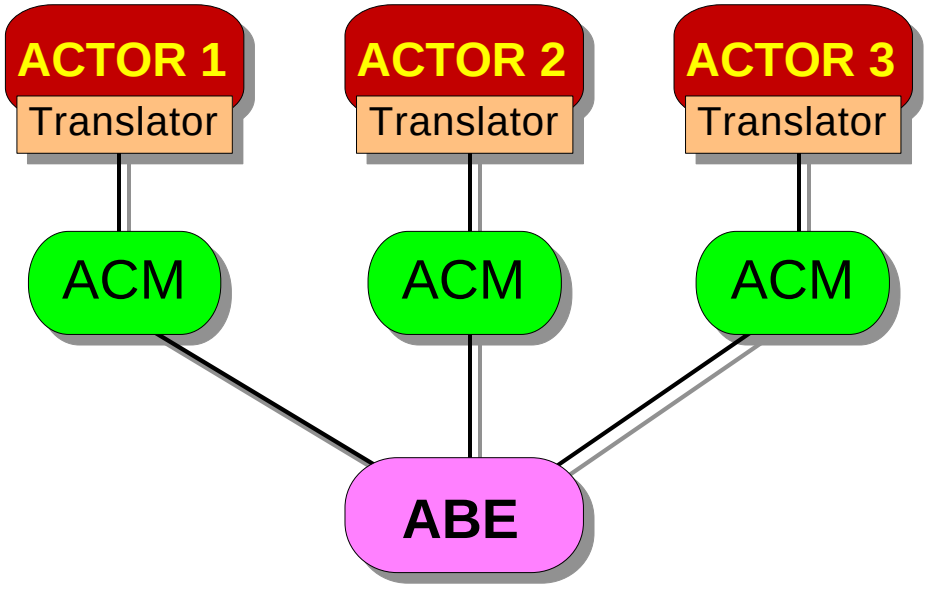


Figure 159 : Architecture d'une confédération ALSP simple

Le **translator** (traducteur) est une portion de code qui complète l'acteur et le rend compatible ALSP[124]. Il peut être intégré au code de l'acteur, ou extérieur, voire même situé sur une autre machine. Il peut s'agir, par exemple, d'une bibliothèque liée au code de l'acteur lors de l'édition de liens de l'application. Le translator, comme son nom l'indique, convertit les événements internes en messages ALSP et vice versa. Il s'occupe également de l'adhésion de l'acteur à la confédération. Il coordonne l'horloge de l'acteur avec le temps de la confédération.

L'**ACM** est l'ALSP Common Module. Bien que ce composant logiciel soit le même pour tous, il en faut une copie séparée pour chaque acteur, mais pouvant se situer sur une autre machine. Ce mécanisme permet d'avoir une interface commune pour tous les translators et permet à un couple acteur/translator de rejoindre une confédération sans se préoccuper des autres acteurs. L'ACM coordonne les adhésions et les démissions de la confédération, gère le temps, filtre les messages arrivant et gère la propriété et les droits d'accès aux attributs.

L'**ABE**, ALSP Broadcast Emulator, est un processus destiné à faciliter la communication entre les ACM. Sa principale fonction est de redistribuer les messages sur ses différents canaux de communication. Il est possible de relier deux ABE par un tel canal (appelé *tie-link*), ce qui donne une grande souplesse dans l'organisation et l'optimisation d'un réseau de simulation distribuée basé sur ALSP.

Grâce à cette architecture, on peut optimiser la confédération en fonction des performances des liaisons de communication une confédération complexe, comme le montre la Figure 160, qui représente la Joint Training Confédération (JTC), telle qu'elle était en 1996. On y trouve:

- Air Warfare Simulation (AWSIM), de l'US Air Force
- Corps Battle Simulation (CBS), de l'US Army

[124] Il s'agissait ainsi de pouvoir facilement réutiliser des simulations existantes (*legacy simulation*) en les adaptant à ALSP à moindre coût.

- MAGTF Tactical Warfare Simulation (MTWS), de l'US Marine Corps
- Research, Evaluation and Systems Analysis (RESA), de l'US Navy
- Tactical Simulation Model (TACSIM), de l'US Army (renseignement)
- Joint Electronic Combat-Electronic Warfare Simulation (JECEWSI)
- Combat Service Support Training Simulation System (CSSTSS), de l'US Army
- Portable Space Model (PSM), une simulation d'alerte missile ballistique.

Cette confédération fait évoluer 19 types d'objets publics (unités, navires, hélicoptères, sous-marins...), chacun avec 43 attributs (localisation, identité, mission, état, ...) et pouvant interagir de 58 façons différentes!

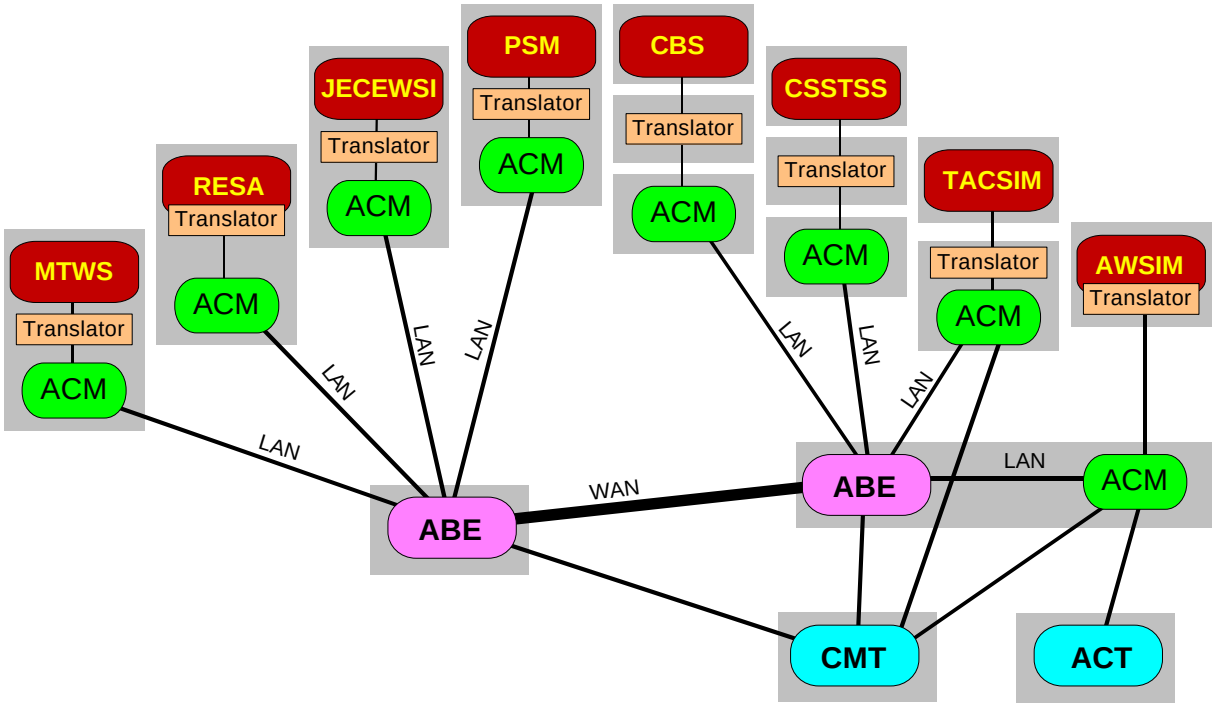


Figure 160 : JTC ALSP Confederation (1996)

Note : Le CMT (*Confederation Management Tool*) et l'ACT (*ALSP Control Terminal*) sont des outils complémentaires pour l'administration de la confédération.

Les acteurs communiquent entre eux via des messages textuels permettant la réalisation de divers services et requêtes :

- Initialisation (`join`, `register_attrs`, ...)
- Filtrage (`filter_xxxx`, ...)
- Interaction (`interaction`, ...)
- Gestion objets (`register_object`, ...)
- Gestion du temps (`grant_advance`, ...)
- Sauvegardes (`save_request`, ...)

L'une des caractéristiques les plus importantes d'ALSP par rapport à DIS est sa gestion du temps, non temps réel. En effet, ALSP **coordonne le temps** entre les acteurs d'une confédération, qui n'ont pas forcément le même temps.

La méthode de synchronisation est celle du rendez-vous: chaque acteur avance d'un certain quantum de temps, puis attend que tous les autres acteurs l'aient rattrapé avant d'avancer de nouveau. En pratique, il demande, via l'ACM, l'autorisation d'avancer (message `advance_request`) et attend l'accord (message `grant_request`).

L'acteur suppose que la date courante de la confédération est la dernière date qui lui a été accordée par l'ACM.

La confédération suppose que la date courante de l'acteur est la dernière date qu'il a demandée, plus un temps de latence (*lookahead*), précisé par celui-ci lors de l'adhésion à la confédération. L'ACM date tous les messages à destination de la confédération avec cette valeur (*timestamp*).

Une autre caractéristique intéressante est l'absence de standardisation de la hiérarchie des objets manipulés, celle-ci étant définie de façon propre à chaque confédération dans un fichier. Ceci est plus souple que les énumérations de DIS, mais nécessite un travail de préparation substantiel pour chaque confédération[125].

Chaque acteur dispose d'une base de données locale des objets déclarés au sein de la confédération. Ces objets peuvent soit être locaux, soit gérés par un autre acteur. Dans ce dernier cas, la simulation crée une image locale (*ghost*) de l'objet. On comprend aisément que ce mécanisme se révéla particulièrement utile lors de l'adaptation à ALSP de simulations non conçues initialement pour la distribution.

Au milieu des années 90, on avait donc deux solutions concurrentes pour l'interconnexion de simulations : DIS, plus orienté temps réel, et ALSP, pour les simulations constructives non temps réel, les deux étant totalement incompatible entre eux bien entendu. La communauté de la simulation de défense n'étant pas très grande, cela faisait indéniablement trop de standards...

[125] Malgré cette contrainte, c'est cette solution qui fut retenue pour le standard unitificateur, HLA, avec la FOM propre à chaque fédération.

9.5.4.3 HLA

DIS, malgré un succès indéniable, finit par s'enliser dans des considérations techniques de **bas niveau**, et ne remplit pas ses promesses d'interopérabilité, car sa rigidité incita nombre de développeurs à créer leurs propres extensions propriétaires.

En outre, DIS était mal adapté aux besoins des simulations agrégées non temps réel (typiquement, les jeux de guerre). Certes, il y avait ALSP, mais il s'agissait d'un protocole propriétaire, destiné à une communauté bien précise[126].

Pendant ce temps, le DoD voyait exploser les **coûts de la simulation**. Il fut alors décider qu'il était temps d'introduire dans ce domaine des règles drastiques de **rationalisation** et de **qualité logicielle**. Le DoD chargea alors le **DMSO**[127] de développer une nouvelle architecture de haut niveau, générique et ouverte, incorporant un volet méthodologique. En 1995, la première version du standard HLA fut publiée.

La version actuelle est la 1.3 DoD. Avec quelques modifications mineures, elle a été standardisée fin 2000 par l'IEEE (standard n°1516) et un standard OTAN (STANAG) est en cours de rédaction. HLA a été également adopté par l'OMG[128] comme standard pour les simulations distribuées.

Contrairement à DIS, HLA n'est pas un protocole de communication, mais un standard d'interopérabilité générique, applicable à tous types de simulations.

Il part du principe qu'aucune simulation monolithique ne permet jamais de satisfaire tous les besoins des utilisateurs à des fins d'entraînement, d'études, etc. En outre, nul ne peut être expert dans tous les domaines, et il faut donc être capable d'utiliser des modules de simulation développés par d'autres. Enfin, il est illusoire de croire que l'on peut prédire les évolution à long terme, et même à moyen terme, de la simulation, dans la mesure où elle est fortement liée à des technologies informatiques dont les progrès sont particulièrement rapides. Par conséquent, il faut pouvoir intégrer des composants « étrangers » dans une simulation, tout en conservant ouverture et évolutivité. Il fallait pour cela un standard de plus haut niveau que DIS, trop rigide, et plus versatile que ALSP, trop spécialisé.

Le DoD préconise donc, avec HLA, une approche de **conception par composants**[129] : une simulation HLA est une **fédération** de simulations. Les membres d'une fédération HLA (**fédérés**) peuvent en fait être autre chose que des simulations, par exemple un outil de visualisation 3D, ou un matériel dans la boucle (simulation *live* ou hybride).

En plus de l'**interopérabilité des simulations**, HLA développe la **réutilisabilité des simulations**.

Le standard HLA comprend trois parties, correspondant à trois documents :

---

[126] Ces inconvénients ont entraîné une rapide disparition d'ALSP à la fin des années 90. Cependant, nombre de principes utilisés dans ALSP ont été repris et adaptés dans HLA, justifiant ainsi l'intérêt que nous avons porté à ce défunt standard. Notons que, en revanche, il existe encore de nombreuses simulations DIS dans le monde, notamment dans le domaine de la simulation pilotée, probablement appelées à perdurer de nombreuses années encore en raison des importants investissements qu'elles ont nécessité.

[127] *Defense Modeling and Simulation Office.*

[128] Object Management Group, association d'éditeur de logiciels et de constructeurs, qui conçu notamment le standard CORBA d'interopérabilité d'applications.

[129] Principe du *Component Based Design*. HLA n'est pas une révolution, mais une évolution, reprenant les concepts actuellement utilisés pour le génie logiciel.

- **Les règles HLA** : ce sont 10 principes que les fédérés et les fédérations doivent respecter pour arriver à assurer l'interopérabilité d'applications via HLA. Elles définissent le rôle de chaque intervenant dans une fédération.
- **Le formalisme de trames objets (OMT)** : l'*Object Model Template* est une méthode de description orientée objets (voir **Erreur ! Source du renvoi introuvable.**) des entités et des interactions manipulées par les fédérés.
- **La spécification d'interface (IFSPEC)** : Il s'agit de l'interface de programmation (API) qui permet aux fédérés d'accéder aux services de l'infrastructure d'exécution de HLA (**RTI**). L'IFSPEC décrit également l'interface que doivent fournir ces fédérés au RTI.

C'est le respect de ces trois documents[130] qui détermine la **conformité à HLA**.

On peut rajouter également le **processus de développement et d'exécution de fédération (FEDEP)**, qui est souvent traité comme s'il faisait partie du standard (voir [DMSO99] et Figure 161).

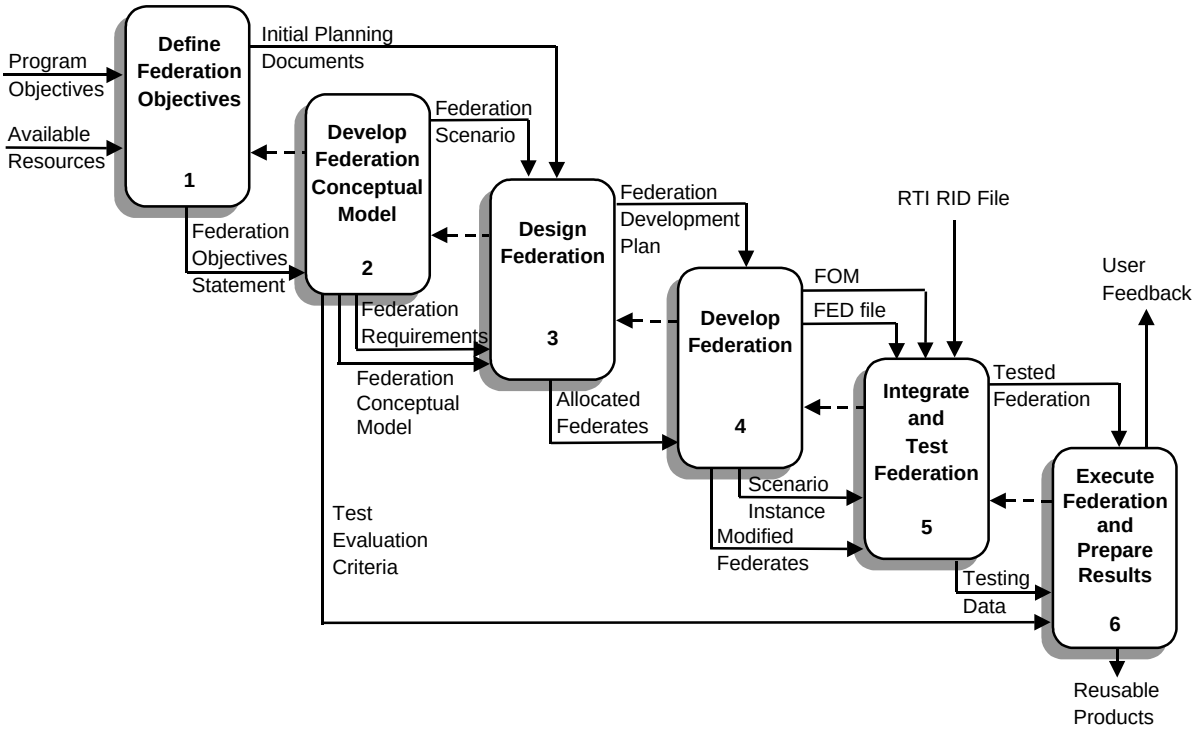


Figure 161 : Processus FEDEP en 6 étapes du DMSO (v1.4)

La Figure 161 représente la version 1.4 du FEDEP. Le processus général FEDEP inclut depuis la version 1.5 un rebouclage de chaque étape sur les précédentes, afin de permettre des actions correctives et un **développement incrémental** (par versions successives), approche de plus en plus courante en simulation.

[130] On peut, à la rigueur, considérer que le respect des règles HLA est une condition nécessaire et suffisante pour la conformité à HLA, puisque ces règles impliquent le respect de l'IFSPEC et de l'OMT.

Les fédérés (voir **Erreur ! Source du renvoi introuvable.**) peuvent être :

- Des applications de simulation : jeux de guerre, simulateurs, simulations scientifiques…
- Des utilitaires : visualisation passive 2D (*Plan View Display*) ou 3D (*Stealth Viewer*), enregistreurs de données (*Data Logger*), contrôleurs de fédération, etc. (voir §9.5.5).
- Des matériels réels : sous-systèmes (simulation hybride), plates-formes instrumentées sur le terrain (simulation dite « live »).

Ces fédérés fournissent une interface au RTI appelé *Federate Ambassador*, ainsi qu'un certain nombre de services documentés dans l'IFSPEC, par exemple fournir l'état d'une entité de la simulation.

À chaque fédéré est associée une description des objets (entités, interactions) qu'il est susceptible de traiter, appelée **SOM** (*Simulation Object Model*). Ce SOM indique ce dont est potentiellement capable l'application, et peut dépasser les besoins de la fédération pour laquelle elle a été conçue. En effet, la philosophie connexe à HLA consiste à archiver les applications développées pour une réutilisation ultérieure. Le SOM fait ainsi partie de la documentation de l'application devenue composant logiciel.

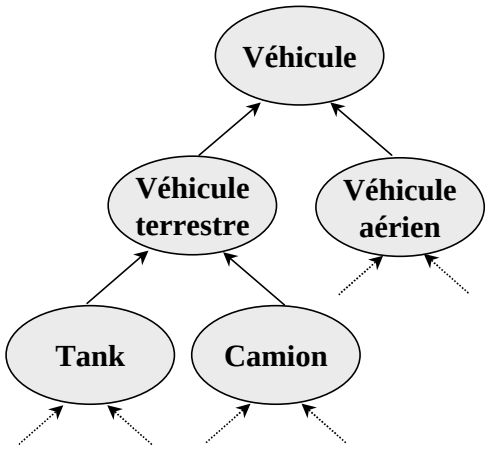


Figure 162: Exemple de hiérarchie d'objets

Les sous-ensembles des SOM des différents fédérés qui seront effectivement utilisés dans la fédérations sont, eux, décrits dans le **FOM** (*Federation Object Model*). Ce FOM contient la liste de tous les objets et interactions qui vont être échangés par les fédérés, ainsi que des informations sur la fédération et une table des espaces de routage des données. Il est fourni en paramètre au RTI (*RunTime Infrastructure*).

Le RTI est un logiciel de type *middleware*, c'est-à-dire un intermédiaire entre les applications et les couches basses de communication du système d'exploitation. Il permet, en masquant ces dernières, de libérer le développeurs de nombreux aspects techniques de bas niveau, lui permettant de se concentrer sur les fonctionnalités de son application.

Il fournit **différents services aux fédérés**, tels que la gestion de la fédération, des objets, du temps de la fédération, des déclarations (créations d'objets…), de propriétés (transfert d'un objet ou d'un attribut d'objet d'un fédéré à l'autre), de la distribution des données (DDM)…

Ces services sont offerts aux fédérés par l'intermédiaire d'un module appelé *RTI Ambassador*, dont l'interface de programmation est conforme à l'IFSPEC.

Le RTI est bien entendu un **composant clé de la fédération**, qui a une influence déterminante sur la qualité et les performances de l'ensemble.

Or, si l'interface du RTI vers les applications est bien documentée dans l'IFSPEC, le standard est muet sur l'implémentation des services qu'il fournit, et notamment sur le type de support de communication utilisé.

Ceci peut paraître surprenant, surtout quand on compare à son prédécesseur DIS, qui était un protocole de communication.

En réalité, il s'agit d'un choix délibéré, qui permet à HLA de pouvoir prétendre à l'universalité. HLA n'est pas une architecture physique, mais une **architecture logicielle**.

Le développeur de fédération peut alors **choisir parmi plusieurs implémentations du RTI** celle qui lui convient le mieux. Ainsi, si la plupart des RTI actuellement disponibles utilisent les services de communication TCP/IP[131], lequel a des performances limitées, certains, plus spécialisés, utilisent des réseaux hauts débits dédiés ou même de la mémoire partagée.

Le corollaire est en revanche que, en principe, **seuls des fédérés utilisant le même RTI peuvent participer à une fédération**.

Actuellement, le RTI le plus utilisé est celui qui est fourni gratuitement par le DMSO sur son site Internet (http://hla.dmso.mil). Toutefois, ce produit ne devrait plus évoluer notablement, le DMSO souhaitant que des RTI commerciaux prennent le relais. À moins que certains des RTI libres (fournis gratuitement avec leur code source) qui sont développés dans les milieux académiques n'arrivent à une qualité industrielle et ne viennent bouleverser le marché.

En outre, dès 1997, le DMSO a fourni à la communauté de la simulation, en plus du RTI, des outils d'aide à la conception, à la mise au point et à l'exécution de fédérations HLA : rédaction de SOM/FOM, contrôle de la fédération, capture de messages, etc. (voir §9.5.5). Depuis, des produits commerciaux sont apparus sur le marché et devraient prendre peu à peu la relève des outils gratuits du DMSO.

[131] Protocole standard du réseau Internet.

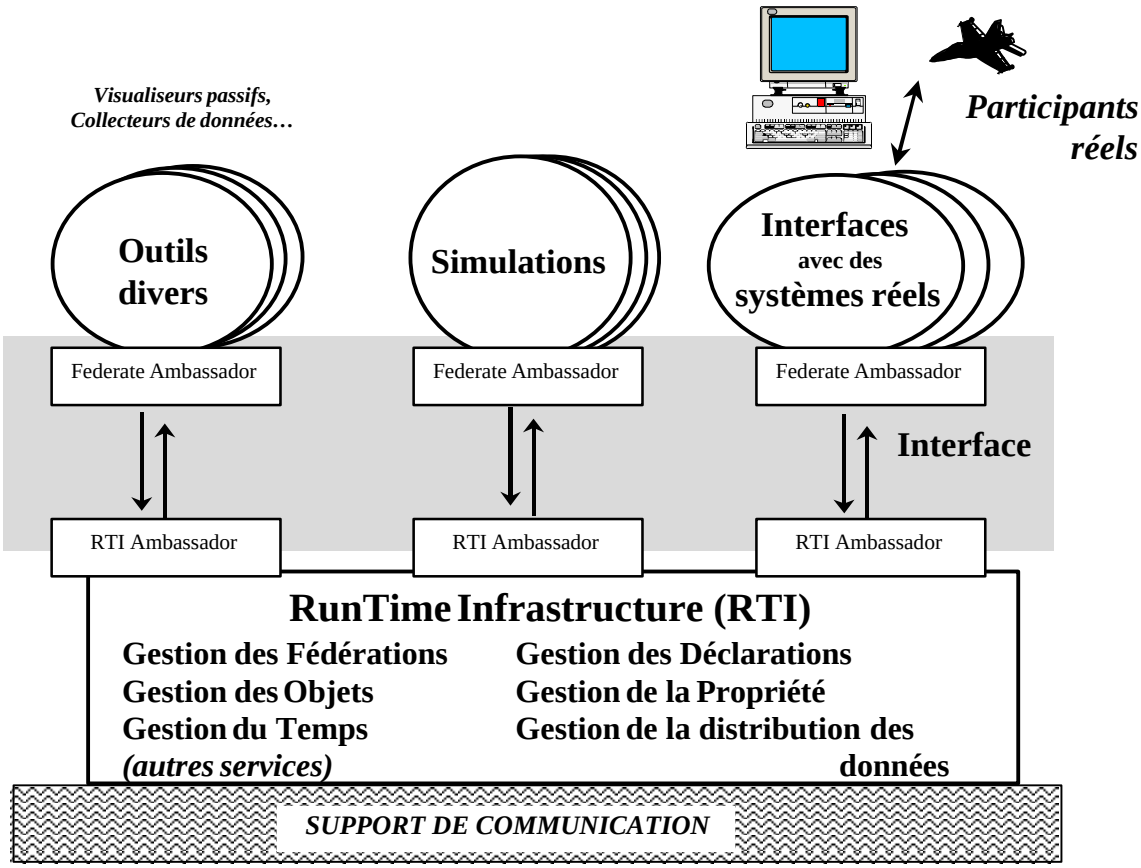


Figure 163: Architecture d'une fédération HLA

9.5.4.4 CORBA

Dans les années 80, les logiciels devenaient de plus en plus complexes et coûteux (chanson que vous avez déjà entendue, n'est-ce pas ?). Lors de cette période, on vit se développer de nombreuses techniques et méthodologies de génie logiciel (voir §9.5.4.4), et notamment celles permettant la réutilisation du logiciel, telles que la programmation orientée objets[132].

En 1989, l'OMG[133], un large consortium d'industriels, fut créé avec pour objectif de « maximiser la portabilité, la réutilisabilité et l'interopérabilité des logiciels par la définition de standards industriels pour le développement d'applications commerciales orientées objets ». L'OMG développa, entre autres, un modèle objet et une architecture de communication client/serveur entre composants basée sur un « bus logiciel à objets »[134] : CORBA[135].

[132] C'est à cette époque qu'apparurent Ada (Ada83 n'était pas pleinement orienté objet, mais permettait une bonne réutilisation des paquetages logiciels) et C++ notamment. Java n'apparut que dans les années 90, en intégrant les problèmes particuliers liés à l'utilisation de composants logiciels à travers Internet (portabilité au niveau code, distribution, sécurité…).

[133] Object Management Group.

[134] Toute ressemblance avec HLA pourrait ne pas être fortuite !

[135] CORBA™: Common Object Request Broker Architecture.

CORBA permet à des composants logiciels d'interagir par l'envoi de messages, indépendamment du système d'exploitation, de la plate-forme matérielle, du langage de programmation ou du réseau.

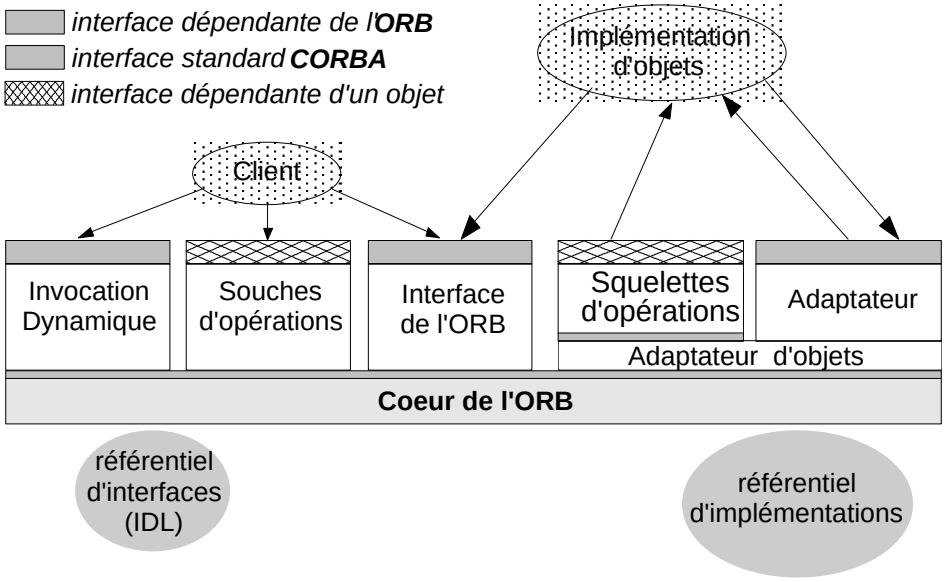
La description des interfaces entre les composants est réalisée à l'aide d'un langage de description d'interface (IDL) spécifique, ce qui permet d'effectuer la déclaration des interfaces indépendamment du langage utilisé, un traducteur se chargeant ensuite de produire le code requis.

Le standard CORBA comprend :

- un modèle objet de référence ;
- les spécifications d'un mécanisme de communication entre composants ;
- la description des API des services ;
- la définition du langage IDL ;
- la correspondance entre IDL et les langages de programmation (mapping).

Comme pour le RTI de HLA, il n'y a pas d'implémentation standard du middleware (l'ORB[136]) : deux ORB d'origine différentes ne sont pas forcément compatibles.

La filiation entre CORBA et HLA est évidente, et d'ailleurs l'implémentation du RTI qui a été réalisée par le DMSO est basée sur CORBA.



(d'après [PINA96])

Figure 164: Architecture CORBA

Aujourd'hui, CORBA est largement utilisé, bien que parfois très discret. On le trouve ainsi au cœur de l'interface graphique GNOME, très populaire chez les utilisateurs du système d'exploitation Linux. COBRA a largement bénéficié ces dernières années de l'apparition d'implémentations gratuites et libres, ainsi que du développement des services en ligne sur Internet. Le protocole IIOP (Internet Inter-Orb Protocole) a été adopté par de nombreuses sociétés (Netscape, Oracle, IBM...) pour le développement d'applications sur Internet.

[136] Object Request Broker.

Le principal concurrent direct de CORBA, pour un usage générique, est probablement la technologie DCOM propriétaire de Microsoft, qui est assez proche. Toutefois, l'utilisation de DCOM est très liée à l'environnement Windows, bien que Microsoft ait pris l'initiative de l'étendre à d'autres plates-formes. En revanche, la popularité de Windows a entraîné la disponibilité d'un grand nombre de composants DCOM.

Pour résoudre les problèmes de communication entre les deux architectures (bien évidemment incompatibles entre elles), des passerelles et un mapping DCOM/CORBA ont heureusement été développés.

D'autres architectures ou protocoles peuvent être une alternative (Java RMI, moniteurs transactionnels, etc.) mais n'ont pas tous les avantages de CORBA. D'ailleurs, souvent, ces techniques peuvent être vues comme un complément à CORBA. Ainsi, Sun a réalisé des composants Java (EJB[137]) au-dessus de CORBA et masquant ce dernier pour ne conserver que l'interface Java standard.

Notons que l'OMG a récemment adopté HLA comme standard d'interopérabilité pour les simulations.

[137] Enterprise Java Beans.

9.5.4.5 Comparaison des standards

Dans un mémoire de DESS **[PINA96]**, Jean-Marc Pinard compare avec pertinence ces quatre standards. Il donne un tableau résumant les particularités de chacun :

	ALSP *Aggregate Level Simulation Protocol*	**DIS** *Distributed Interactive Simulation*	**HLA** *High Level Architecture*	**CORBA** *Common Object Request Broker Architecture*
Domaine d'application	simulations constructives (« jeux de guerre »)	simulations d'entraînement en temps réel	tous types de simulations et interface aux C4I	service général, non dédié à la simulation
Niveau d'intégration	protocole (*+ quelques services*)	protocole	architecture + API	« bus » à objets + API
Objectifs majeurs	interopérabilité de simulations non temps réel	interopérabilité de simulations temps réel (+matériel)	interopérabilité et réutilisation de simulations tous types	interopérabilité d'objets en architecture client/serveur
Mode de diffusion	MITRE (limité à la défense US)	norme IEEE 1278	public via l'Internet *(restrictions pour la diffusion du RTI)* norme IEEE 1516	implémentations commerciales (*ex: ORBIX*)
Point de départ	projet ARPA (90)	SIMNET en 88, document DIS Vision (10/93)	Master Plan M&S DoD 5000-59 (10/95)	regroupement d'industriels de l'informatique (89)
Méthodologie explicite			oui (via OMT et FEDEP)	description des interfaces en IDL
Transport des informations	chaînes ASCII (syntaxe)	messages « PDU » (binaire IEEE)	m.a.j valeurs d'attributs via services	requêtes entre objets
Service commun d'exécution (« tuyau »)	AIS (ALSP Infrastructure Software)	rien (mais qq. PDU de gestion de simul.)	RTI (RunTime Infrastructure)	ORB + services objets (Object Request Broker)
Couplage informatique	via le traducteur	coupleur DIS *(utilisé via des API)*	appels à API *(+Common Software?)*	souches et squelettes + appels à API
Images locales d'objets distants	objets fantômes (« ghost »)	extrapolées («dead reckoning »)	attributs répercutés	proxy agent
Gestion du temps	explicite (par événements coordonnés)	rien (temps réel, aucune coordination)	explicite (a priori, tout les types de combinaisons)	rien sauf service « temps » (serveur de temps)
Outils communs	ACT, CMT	produits commerciaux *(ex : « espion 3D »)*	Nombreux outils gratuits ou commer-ciaux disponibles	utilitaires communs (common facilities)
Organisme de tutelle	STRICOM	IST	DMSO	OMG
Version courante (mi-2000)	-	2.1.4	1.3	3.0
Support actuel (année 2000)	Abandonné	Abandonné pour les nouvelles applications, mais le parc installé reste important	Forte expansion	En expansion

Figure 165 : Comparaison des standards DIS, ALSP, HLA, CORBA

9.5.4.6 Conclusion de cette comparaison

On le voit, l'interopérabilité des simulations s'est beaucoup cherchée (il ne faut pas oublier SIMNET et toutes les architectures propriétaires et expérimentales qui ont précédé).

Aujourd'hui, un certain consensus semble se faire aujourd'hui autour de HLA, dont l'adoption de plus en plus vaste par la communauté internationale est plutôt engageant. En effet, HLA a opéré la synthèse de ses prédécesseurs, et propose une architecture quasi universelle, ouvrant de nouveaux horizons en matière d'interopérabilité et de réutilisation de modèles et simulations.

9.5.4.7 Quel avenir pour l'interopératibilité des simulation ?

Après SIMNET, DIS et ALSP, il est difficile de garantir à 100% que HLA ne sera pas remplacé à long terme par un autre standard. Néanmoins, il s'est stabilisé depuis 1997 (version 1.3) et sa généricité le rend adaptable à pratiquement tous les besoins. Les faiblesses actuelles de l'implémentation la plus répandue (celle du DMSO), et notamment ses performances en temps réel et pour les fédérations communicant de façon intensive, peuvent être résolues par des implémentations améliorées (la nouvelle version NG du RTI du DMSO a ainsi notablement augmenté les performances) ou optimisé pour un emploi donné. De plus, HLA est désormais obligatoire pour toute simulation de défense aux USA et d'importants investissements ont été faits. Un test déterminant pour HLA sera son adoption éventuellement par la communauté de la simulation civile.

L'effort est actuellement porté sur l'encouragement de l'offre commerciale de RTI et d'outils HLA, le développement de l'aspect méthodologique (FEDEP, VV&A[138]…), l'élargissement de la standardisation (OMG, IEEE, OTAN, ISO…), l'ouverture vers le civil, la capitalisation de modèles et de ressources (MSRR), et les aspects de l'inter-opérabilité non traités dans HLA.

Car (on ne le répétera jamais assez) **être conforme à HLA est une condition nécessaire mais non suffisante d'interopérabilité**. Nous avons déjà vu la limitation due aux différents types de RTI. Mais il faut également, entre autres, que les simulations partagent la même vision de leur environnement naturel et tactique. D'où le développement d'un standard pour l'interopérabilité des bases de données d'environnement, SEDRIS (voir §9.3.7), et d'un modèle conceptuel de l'espace de mission (CMMS) commun.

La route de l'interopérabilité est donc encore longue et difficile, mais le chemin parcouru est déjà appréciable et la vision du but à atteindre plus précise : la réalisation d'environnements synthétiques complexes et très proches de la réalité.

[138] Vérification, validation et accréditation des simulations.

9.5.5 Quelques outils pour la simulation distribuée par HLA

9.5.5.1 Stealth Viewer

Un *stealth viewer* (ou *stealth*) est un logiciel de visualisation 3D passif qui s'intègre comme fédéré à une fédération HLA et entretien un bilan de la situation de chaque objet. Il peut alors présenter à son utilisateur une représentation 3D dynamique et temps réel du milieu dans lequel évoluent les objets de la fédération (voir Figure 166). L'utilisateur peut se déplacer à son gré, comme s'il se promenait au milieu de l'environnement synthétique. En revanche il ne peut interagir avec les entités qu'il rencontre, d'où le terme de « stealth » (invisible).

Les stealth viewers sont très pratiques pour analyser une situation, mais sont également de plus en plus utilisés comme logiciels de visualisation 3D « sur étagère » pour les simulations (i.e. à la place d'une interface graphique propriétaire).

Figure 166: MäK Stealth Viewer

Figure 167: MäK Plan View Display

9.5.5.2 Plan view display

Le plan view display est une visualisation en 2D du domaine de la simulation, offrant une vision globale de la répartition et des mouvements des entités sur le terrain, typiquement en plaçant des symboles représentant les entités de la simulation (éventuellement agrégées) sur une carte.

Dans un jeu de guerre, lorsque le PVD permet de visualiser les deux camps à la fois, on parle également de « *God's eye* ».

9.5.5.3 Data logger

Le data logger est un enregistreur de message, qui stocke dans un fichier sur disque tous les échanges réalisés entre les membres d'une fédération (DIS ou HLA). Cela permet ultérieurement d'analyser finement le déroulement de la fédération, voire de rejouer la simulation ou de la restaurer à une étape donnée.

L'un des outils disponibles à cet effet pour HLA est le DCT (*Data Collection Tool*).

9.5.5.4 Outils de gestion de fédération

La gestion d'une fédération (*simulation management* ou *SIMAN*) comprend des opérations telles que la déclaration de nouveaux membres, la restauration/sauvegarde, le test des liaisons entre les fédérés, etc.

- FMT (*Federation Management Tool*)
- FVT (Federation Verification Tool)

9.5.5.5 Outil d'aide à la conception de fédération

Ces outils sont très spécifiques à un standard donné. Pour HLA, citons ceux promus par le DMSO américain :

- FEPW : *Federation Execution Planner's Workbook Editor*, éditeur de planification de l'exécution d'une fédération. Utilisé lors de la phase de conception et développement du processus de développement et d'exécution de fédération (FEDEP), le FEPW décrit la configuration et les conditions d'exécution d'une fédération. Cette information sert alors de base pour la description de la structure physique de la fédération, de ses performances et de ses besoins en ressources. L'éditeur de FEPW utilise un format de fichier FEPW-DIF pour pouvoir échanger des données avec les autres outils.
- OMDT : *Object Model Development Tool*, outil de développement de modèles objet. Il permet aux développeurs de fédération et de fédérés de documenter leurs modèles objet HLA, en s'assurant de la conformité de ces derniers avec les exigences du standard HLA (notamment la partie OMT). L'OMDT est capable de faire des contrôles de syntaxe et de cohérence, d'accéder à la librairie online de modèles objets (OML), de générer des fichiers de descriptions des détails d'exécution de la fédération (*Federation Execution Details*, FED) pour le RTI, supporte les modèles objet de gestion (MOM), etc.
- OMDD : *Object Model Data Dictionary*, dictionnaire de données pour modèles objet.
- OML : *HLA Objet Model Library*, bibliothèque de modèles objets HLA (SOM et FOM) disponible en ligne sir Internet[139]. Ces modèles objets sont stockés au format HLA OMT DIF, lisible par l'outil OMDT, qui peut accéder en lecture et écriture directement à cette bibliothèque.
- RID Editor : éditeur de données d'initialisation du RTI. Cet utilitaire permet d'éditer les fichiers RID qui permet de paramétrer l'exécution du RTI.
- FVT : *Federation Verification Tool*, outil de vérification de fédération. Il assiste les développeurs de fédération dans la phase de test et d'intégration du processus de développement et d'exécution de fédération (FEDEP). Il vérifie notamment que chaque fédéré remplit son contrat de mise à jour et de publication d'attributs, d'envoi et de réception d'interactions, d'invocation de services HLA. Les données peuvent être entrées

[139] http://www.omlibrary.epgc4i.com/

directement dans le FVT, ou importées à partir de fichiers au format FEPW DIF.

- DCT : *Data Collection Tool*, outil de collecte et d'analyse de données. Il s'agit d'un fédéré qui capture les données échangées durant l'exécution d'une fédération. Ces données peuvent être ensuite examinées et analysées. L'outil DCT peut également générer une base de données correspondante.
- FMT : *Federation Management Tool*, outil d'administration de fédération. Cet outil est destiné à aider son utilisateur à gérer l'exécution d'une fédération et des fédérés y participant, par exemple en contrôlant la synchronisation des fédérés entre eux.

Ces outils sont disponibles gratuitement en téléchargement sur le site du DMSO[140] (moyennant un enregistrement préalable).

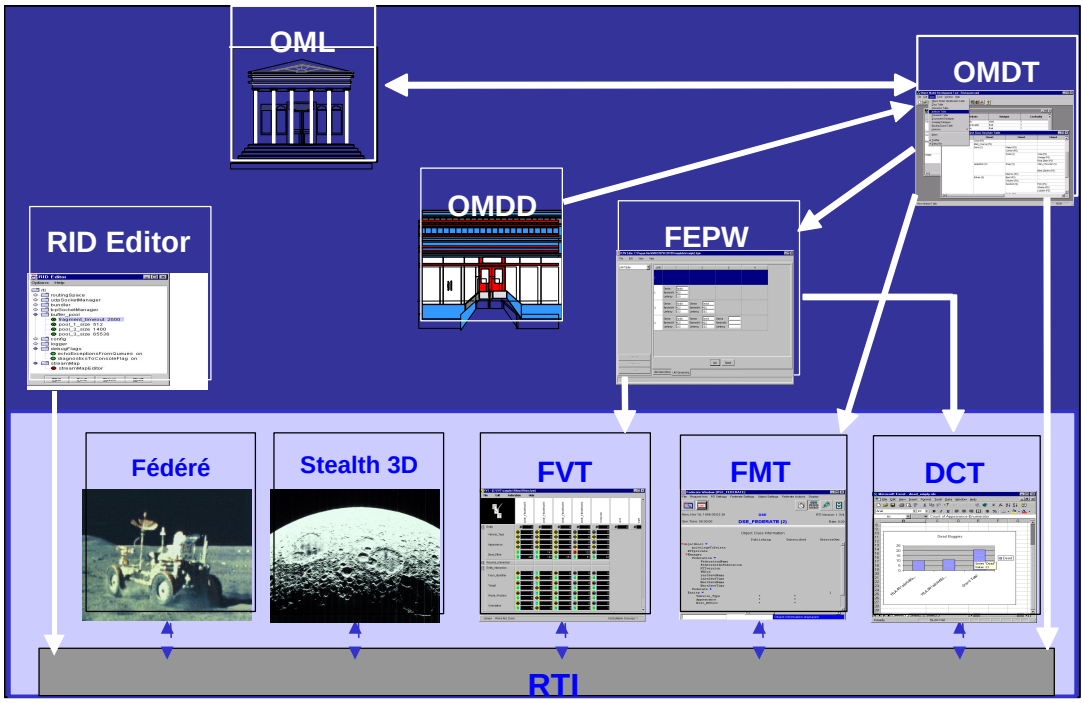


Figure 168: Relations entre les outils HLA

9.6 La sécurité dans les simulations

Les simulations contiennent à travers leurs modèles et leurs données une expertise précieuse. Les modèles, et particulièrement les prototypes virtuels, sont des représentations parfois très précises du système réel, dont l'analyse peut révéler nombre de secrets.

Jusqu'ici, les simulations tournaient de façon isolées, dans des locaux dont la sécurité était adaptée. On retenait qu'un modèle suffisamment précis devait être aussi protégé que le système réel. Maintenant, avec les environnements synthétiques distribuées, la

[140] http://hla.dmso.mil

question de la sécurité se pose de façon beaucoup plus complexe, et la réponse est loin d'être évidente.

En effet, lorsque deux simulations sont connectées en réseau, les données qu'elles s'échangent peuvent être interceptées par un tiers. Si ces données contiennent par exemple la réponse d'un avion de combat éclairé par un radar, on comprend aisément les informations que l'on peut ainsi obtenir tant sur le radar (caractéristiques des émissions, ce qui est utile à l'ennemi pour détecter, identifier et brouiller ce radar) que sur l'avion (SER[141], caractéristiques de furtivité).

Le tiers indélicat peut être également contrôler un fédéré de la simulation. Cette situation peut se produire dans le cadre d'une fédération internationale, ou d'une fédération regroupant plusieurs industriels d'un GIE. Il a alors la possibilité non seulement d'intercepter toutes les données échangées, mais aussi de stimuler volontairement un modèle « étranger » pour analyser sa réponse. Par exemple, il peut éclairer avec un radar qu'il contrôle un avion appartenant à une autre simulation et en déduire sa SER.

Nous n'allons pas rentrer dans le détail de la problématique de la sécurité des systèmes d'information, mais, en revanche, voici quelques solutions parmi les plus fréquemment mises en œuvre pour sécuriser les simulations distribuées :

- Protection des liaisons par chiffrement : le cryptage des données entre deux nœuds du réseau empêchera un tiers de prendre connaissance des données (sensibles) échangées. Le chiffrement peut intervenir au niveau des couches bases (par exemple chiffrement au niveau de la couche Ethernet sur un réseau local) ou hautes (chiffrement IP, voire applicatif). La tendance actuelle est d'utiliser un chiffrement de haut niveau, qui est plus compatible avec l'utilisation de réseaux de communication publics.
- Filtrage des données : un composant logiciel filtre les données destinées aux autres fédérés, en bloquant celles dont le degré de sensibilité ne correspond pas au niveau d'habilitation ou au besoin d'en connaître du destinataire.
- Fidélité variable : la précision des données issues d'un modèle peut être modulée en fonction du destinataire. Ainsi, pour un destinataire non habilité, on pourra dégrader la précision des résultats, voire introduire des erreurs aléatoires, de façon à ce que les données soient suffisamment correctes pour permettre leur utilisation dans le cadre de la fédération, mais assez biaisées pour ne pas trop donner d'indications sur les performances du système réel.
- …

[141] Surface Équivalente Radar.

10. L'AVENIR

Sur le plan technologique, l'avenir est relativement prévisible : ce sera celui du « toujours plus »[142]. Plus de puissance de calcul, plus de réalisme, plus de capacité mémoire… Certes, il peut toujours y avoir une rupture technologique majeure, comme le furent le microprocesseur, Internet ou la simulation distribuée, mais nous disposons d'ores et déjà de bases matérielles et logicielles solides.

La vraie révolution se fera au niveau de l'emploi par l'ingénieur de ces technologies. En effet, les environnements synthétiques commence à sortir réellement du domaine du laboratoire ou du spécifique, et se démocratisent. Et là, on peut s'attendre à un bouleversement des méthodes de spécification et conception des systèmes. Rappelons que l'utilisation systématique de la simulation est un facteur important du succès d'Airbus : des avions mieux spécifiés, mieux optimisés.

Cette révolution impliquera une transformation profonde des méthodes de travail en ingénierie. Il faudra apprendre à penser prototypage virtuel, validation, capitalisation, composants de simulation, distribution, sécurité…

Tel sera le prix à payer à court ou moyen terme pour surfer au sommet de la vague de l'ingénierie système.

Quant au long terme… qui sait ? Quand je relis les prévisions des experts des années 50 ou 60 (voire 70 ou 80 !) sur ce que sera l'informatique de l'an 2000, je ne peux m'empêcher de sourire, tout en sachant parfaitement que nul ne peut réellement être prophète à long terme en informatique. Mémoires holographiques, ordinateurs quantiques, connexion directe au cerveau, ordinateur intelligent, réalité virtuelle totale, tout ceci est annoncé depuis bien longtemps, mais reste toujours au stade du laboratoire dans le meilleur des cas. Certes, cela viendra, mais en attendant l'*holodeck* de Star Trek, les prévisions à 20 ou 30 ans relèvent de la pure spéculation… ou de la science-fiction !

[142] Logique dictée au moins autant par des impératifs commerciaux que par les besoins des utilisateurs : employer un processeur capable de faire plusieurs centaines de millions d'opérations par seconde pour rédiger son courrier et envoyer quelques messages sur Internet, est-ce bien raisonnable ?

ANNEXES

11. GLOSSAIRE

Ce glossaire n'est évidemment pas exhaustif. Il s'agit simplement des termes et acronymes que je rencontre couramment dans le monde de la simulation. J'espère néanmoins qu'il vous aidera dans votre vie professionnelle, qu'elle soit militaire ou civile.

2D : Se dit d'une représentation ou d'un affichage graphique en deux dimensions (typiquement X et Y). Exemple : plan view display.

3A : Voir AAA.

3D : Se dit d'une représentation ou d'un affichage graphique en trois dimensions (volume). Exemple : affichage d'un simulateur de vol.

AAA : Analyse après action (*After Action Analysis*), dépouillement d'un exercice ou d'une mission, réel ou simulé.

Accréditation : Approbation officielle d'un modèle ou d'une simulation pour un usage donné.

Activité : Toute cause qui provoque un changement d'état du système. Une activité peut être interne au système (endogène) ou externe au système (exogène).

AFNOR : Agence Française de NORmalisation, organisme établissant les normes en France (équivalent de DIN en Allemagne et ANSI aux USA).

Agent : Entité logicielle autonome, ayant des capacités d'adaptation à son environnement, collaborant avec d'autres agents pour accomplir une mission donnée. Les technologies à base d'agents sont très prisées actuellement, pour la recherche et le traitement d'informations. Les agents sont également de plus en plus utilisés dans le cadre des forces synthétiques (CGF).

AGL : Atelier de Génie Logiciel. Programme d'aide à la spécification, à la conception et au développement de logiciels complexes, permettant l'amélioration de la productivité et de la qualité du logiciel, par exemple en supportant une méthodologie de développement (Booch, SADT, HOOD, OMT…), en permettant une description de haut niveau du programme (UML…), et en fournissant des outils d'aide à la gestion de configuration (SCCS…). Exemple d'atelier de génie logiciel : StP (Software Through Pictures).

Agrégation : Regroupement d'entités individuelles (par ex. des fantassins) en un groupe d'entités de plus haut niveau (par ex. une compagnie d'infanterie), afin de simplifier les calculs, la prise en compte de comportements de plus haut niveau ou la représentation, tout en conservant les effets des comportements individuels et des interactions entre entités simples. Le processus inverse est appelé désagrégation.

Aléatoire : Qui dépend du hasard. Ex. : variable aléatoire.

ALSP : *Aggregate Level Simulation Protocol*. Protocole non temps-réel (contrairement à DIS), destiné aux simulations agrégées (i.e. jeux de guerre).

API : *Application Programmation Interface*, interface destiné à l'utilisation d'un composant logiciel (bibliothèque, etc.) à partir d'un programme développé par l'utilisateur dans un langage donné. Ainsi, les services du RTI de HLA peut-ils être invoqués par l'intermédiaire d'une API depuis les langages C++, Ada, Java…

Attribut : Propriété ou caractéristique d'une entité : vitesse, masse, quantité de carburant restant, etc. Ces caractéristiques ne sont pas nécessairement toutes physiques. Ainsi, il peut y avoir des attributs administratifs, indiquant par exemple quel fédéré est propriétaire de l'entité.

Avatar : Représentation (pas forcément ressemblante) d'un utilisateur dans un monde virtuel.

Base de données d'environnement : ensemble des données décrivant l'environnement (généralement naturel) d'un système simulé.

Benchmark : Évaluation des performances d'un système, en termes de performances (vitesse d'exécution…) ou de résultats (fidélité…). Désigne également le protocole de test, ainsi que le matériel ou logiciel utilisé.

Bibliothèque : (*library)* Collection de fichiers, programmes ou routines. En programmation, une bibliothèque est un ensemble de morceaux de codes qu'il est possible de lier à un code développé afin d'en étendre les possibilités. Ainsi, une bibliothèque graphique fournira des routines pour effectuer des affichages graphiques sur un écran.

Bitmap : Se dit d'une image stockée sous la forme d'une matrice de points (pixels), avec une résolution fixe. Voir aussi : vectoriel.

C3I : Command, Control, Communications, and Intelligence (Voir SIC, C4I).

C4I : Command, Control, Communications, Computers and Intelligence (Voir SIC, C3I).

Calage : Affinage des paramètres d'un modèle à l'aide d'essais afin que sa fidélité soit conforme au besoin.

CAX : *Computer-Aided Exercise*, exercice assisté par ordinateur. Terme utilisé pour les entraînements des états-majors de l'OTAN dans lesquels interviennent des simulations, telles que des jeux de guerre.

Certification (des données) : approbation des données (par l'utilisateur ou le producteur) après validation et vérification de celles-ci.

CGF : *Computer Generated Forces*, entités d'une simulation gérées par un logiciel informatique. Désigne également ce logiciel, de type simulation constructive, modélisant ces entités et les processus décisionnels des différents échelons de commandement impliqués. Voir SAF, forces synthétiques.

Chaîne de Markov : Processus de Markov discret.

Classe : Ensemble d'objets partageant une structure commune (attributs) et un comportement commun (méthodes). Une classe est composée de son interface (visible de l'utilisateur et des autres classes) et de son implémentation (visible uniquement par le développeur).

CMMS : *Conceptual Model of the Mission Space*, modèle conceptuel de l'espace de mission. Une des trois composantes de l'infrastructure technique commune du DoD, le CMMS décrira notamment les entités utilisées dans la simulation militaire, les actions et les interactions dans différents domaines (analyse, entraînement, acquisition…). Le CMMS permettra aux experts militaires et aux développeurs de simulation d'avoir la même vision (conceptuelle) des systèmes, de leur emploi et de leur environnement.

Combined : Terme anglais pour interarmes.

Compilateur : Programme traduisant un programme informatique écrit dans un langage de programmation évolué (tel que C++, Ada ou Java) en un langage de plus bas niveau, interprétable directement par le processeur (cas du langage machine) ou par une machine virtuelle (cas du pseudo-code de Java).

Component-Based Design : Approche de la conception des logiciels dans laquelle on créé un produit (par exemple une simulation) à partir de composants logiciels sur étagère. Lorsqu'un composant n'est pas disponible et qu'il doit être développé, toutes les précautions sont prises pour faciliter sa réutilisation (voir réutilisabilité, HLA).

Composant logiciel : Entité logicielle réutilisable, définissable indépendamment de son contexte d'utilisation, et dont l'interface est définie et stable. Voir Component-Based Design.

Conception : Phase du cycle de vie d'un système, ayant pour objectif de définir de façon précise les fonctions et l'architecture du système, à partir de ses spécifications.

Confédération : Voir Fédération.

Configuration : Composition matérielle et logicielle d'un système, précisée par la nature, le nombre, les interconnexions et les caractéristiques essentielles de ce système.

Constante : Une donnée dont la valeur ne varie pas durant la durée de vie du programme. Exemple : version du système d'exploitation, nombre maximum d'entités pouvant être créées…

Continu : Qualifie une variable dont la valeur change de façon continue. Se dit également d'un modèle dont les sorties varient de façon continue. Voir discret.

CORBA : *Common Object Request Broker Architecture*, architecture distribuée définie par l'OMG pour standardiser les communications entre objets logiciels. L'architecture CORBA comprend un middleware, l'ORB (*Object Request Broker*), qui distribue les requêtes des objets des clients vers les serveurs. Les interfaces entre les objets sont définies à l'aide d'un langage, l'IDL.

COTS : Commercial Off The Shelf, produit sur étagère. Se dit d'un produit que l'on va acheter à une société qui l'a déjà développé, par opposition à un développement spécifique. Le COTS n'est pas toujours totalement adapté au besoin, mais présente l'avantage d'un coût d'acquisition et de possession très inférieur aux produits dédiés. Voir aussi : GOTS.

Couplage : Combinaison de deux modèles. Par exemple, couplage entre un modèle d'écoulement de l'air et un modèle de calcul de déformation de structure pour une étude d'aéroélasticité.

Cycle de vie : Qualifie un modèle des phases de la vie d'un système, depuis la première expression du besoin jusqu'à son retrait du service actif. Comprend, entre autres, les phases de spécification, conception, définition, qualification, utilisation / maintien en condition opérationnelle (MCO), retrait du service. Suivant les choix de modélisation, ce cycle de vie peut prendre différentes formes : en cascade, en V, en spirale...

Deadlock : Situation d'interblocage de deux processus ou ressources qui s'attendent mutuellement. C'est un problème type de la programmation multiprocessus ou distribuée.

DES : *Discrete Event Simulation*, simulation à événements discrets.

Démonstration : Opération visant à prouver qu'un produit est conforme aux besoin du client. Une démonstration peut être faite à l'aide de calculs, de simulation, de méthode formelle, d'essais...

Désagrégation : Voir agrégation.

Déterministe : Se dit d'un système qui, placé dans un état donné, évoluera toujours de la même façon (par opposition à un système stochastique).

DIS : *Distributed Interactive Simulation*. Désigne les simulations interactives distribuées en général, mais aussi le protocole de communication DIS (IEEE 1278) dédié à ce type de simulation.

Discret : Une variable est dite discrète lorsque sa valeur peut changer de façon discontinue à certains instants. Par extension, qualifie un modèle dont les sorties prennent des valeurs discrètes ou d'une simulation utilisant un tel modèle.

Distribué : Qualifie un logiciel dont les composants peuvent être traités par des processus différents. Ces composants, réalisant un ensemble de fonctions donné au sein de l'application distribuée, peuvent ou non résider sur des machines différentes et bénéficient d'une certaine autonomie. Ne pas confondre avec parallèle, qui concerne l'exécution simultanée de portions de code n'ayant aucune autonomie.

DMT : *Distributed Mission Training*. Concept d'entraînement collectif à l'accomplissement de missions par l'utilisation de simulations distribuées.

DSI : Defense Simulation Internet, réseau de la défense américaine dédié à la simulation (notamment avec le protocole DIS). DSI était sécurisé (chiffrement) et comportait des nœuds aux USA, bien entendu, mais aussi en Allemagne, en Grande-Bretagne et en Corée du Sud. Trop onéreux, peu rentable, il a été intégré au bout de quelques années à un réseau plus général.

EAO : Enseignement assisté par ordinateur.

Éléments finis : Technique de calcul sur des structures discrétisées, particulièrement pour la résolution des équations aux dérivées partielles rencontrées en physiques des milieux continus.

Endogène : Interne à un système. Exemple : la vitesse d'un système est une variable d'état endogène de ce système.

Entité : Au sein d'une simulation, désigne tous les éléments élémentaires participant à la simulation (systèmes, unités, humains, matériels, etc.) et à propos desquels sont stockés et gérées des informations.

Entraînement : Entretien et/ou amélioration d'une compétence, i.e. acquise par le biais d'une formation.

Entraîneur : Simulateur piloté simplifié, par exemple sans cabine mobile ni sphère de projection. Peu coûteux en comparaison d'un simulateur Full-flight, il permet la formation initiale de pilotes, la répétition de mission ou l'apprentissage de procédures. Voir simulation virtuelle, simulation pilotée, simulateur.

Environnement : Le domaine naturel (terrain, atmosphère, océan, espace), les objets (i.e. autres systèmes) et les processus (météo…) extérieurs au système étudié mais pouvant influencer son comportement.

Environnement naturel : Le sous-ensemble de l'environnement d'une simulation représentant le domaine naturel (terre, océan, atmosphère, espace).

Environnement synthétique : Représentation du monde réel à travers une simulation. Comprend les entités de la simulation (y compris éventuellement du matériel et des personnels réels), leur environnement naturel, leur environnement tactique, leurs interactions.

Environnement tactique : Le sous-ensemble de l'environnement d'une simulation représentant les entités extérieures au système étudié.

Ethernet : Protocole réseau de bas niveau (liaison de données et physique) utilisé pour les réseaux locaux (LAN). Sa topologie est de type bus. Les débits les plus courants sont 10 Mbit/s (ethernet standard) et 100 Mbit/s (fast ethernet), et le gigabit arrive.

Événement : Changement discontinu de l'état d'un système. Exemples : survenue d'une panne, collision…

Évolutivité : Capacité d'un modèle ou d'une simulation à être modifié dans des délais et avec des efforts raisonnables pour lui permettre de traiter des problèmes différents de ceux pour lesquels ils ont été conçus initialement. Attention toutefois : toute évolution doit s'accompagner d'une nouvelle validation du modèle ou de la simulation, dans son nouveau domaine d'emploi. Ne pas confondre avec portabilité.

Exogène : Dont l'origine est extérieure au système. Par exemple, la température de l'air dans une simulation pilotée est une variable exogène.

Exploitation : L'exploitation d'une exécution d'une simulation (*run*) est la phase consistant à dépouiller et analyser les résultats de la simulation.

Facteur humain : Désigne les influences humaines (biomédicales, psychologiques, comportementales…) sur un système ou un processus. Par exemple, la fatigue d'un opérateur ou la mauvaise compréhension d'un ordre par un soldat.

Fédération : Regroupement d'applications (simulation, outils...) interopérant entre elles pour mener à bien un objectif donné, par exemple réaliser un entraînement collectif. Le terme fédération est utilisé pour DIS et HLA. La terminologie ALSP utilise le terme confédération.

Fédéré : Une application membre d'une fédération. Ce peut être une simulation, mais aussi un outil tel qu'un visualiseur 3D (« stealth viewer ») ou un enregistreur de messages (« data logger »).

Fiabilité : Caractéristique d'un dispositif, exprimée par la probabilité qu'il accomplisse une fonction requise dans des conditions données pendant une durée donnée.

Fidélité : Similitudes fonctionnelles et physiques entre un modèle et le système réel qu'il représente.

FIFO : *First In, First Out*, qualifie une liste d'attente dans laquelle la première entité arrivée est celle qui accédera la première au service.

File d'attente : Liste d'entités attendant la disponibilité d'un service. On parle aussi de queue. Voir aussi : FIFO.

Finesse : Précision temporelle (pas de calcul) et spatiale (résolution du terrain) d'un modèle. Voir granularité.

Fog of war : « brouillard de la guerre ». Désigne le manque d'information des combattants et commandants durant les combats (unités ennemies non visibles, identification incertaine des unités découvertes, etc.).

FOM : Federation Object Model. Dans HLA, description des classes d'objets utilisées au sein d'une fédération, ainsi que de leurs attributs et de leurs interactions, dans le formalisme de l'OMT.

Forces synthétiques : Voir CGF.

Formation : Apprentissage initial d'une tâche, d'un métier. Voir entraînement.

Full fight Se dit d'un simulateur de vol ayant un haut degré de réalisme.

GCL : Générateur aléatoire à congruence linéaire, utilisé pour produire des suites de nombres pseudo aléatoires.

Générateur aléatoire : Dispositif logiciel ou matériel permettant d'obtenir des nombres aléatoires, suivant une loi probabiliste donnée. Voir GCL.

Générateur d'images : Dispositif matériel ou logiciel permettant la production d'images de synthèse en trois dimensions en temps réel. Ce terme est généralement plutôt appliqué aux matériels haut de gamme, utilisés par exemple dans les simulateurs de vol.

Génie logiciel : Application de méthodes scientifiques au développement des logiciels, afin de favoriser la production d'un logiciel de qualité, répondant aux besoins du client au meilleur coût.

Gestion du temps : *Time Management*. Désigne les mécanismes et les services destinés à contrôler l'avance du temps au sein d'une simulation ou d'une fédération de simulations lors de son exécution. Ceci inclut la préservation de la causalité des événements.

GOTS : Governement Off The Shelf, se dit d'un produit développé par le gouvernement et que l'on va réutiliser, par opposition à un nouveau développement. Voir aussi : COTS.

Granularité : Taille des objets manipulés par un modèle ou une simulation. Par exemple, un jeu de guerre pourra gérer des plates-formes (char, fantassin individuel) ou directement des unités entières (ensemble de plates-formes). De même, pour le temps, une granularité d'une seconde signifie que la seconde est l'unité de temps minimale disponible (mais la précision d'une mesure peut être supérieure, de l'ordre de la minute par exemple). Voir finesse, niveau d'une simulation.

Ground truth : « Réalité du terrain ». Représentation exact d'une situation, sans les erreurs introduites par les capteurs ou la perception et le jugement humain. Voir perceived truth.

GUI : Graphic User Interface, voir IHM.

Haptique : Qualifie le sens du toucher et des efforts. En réalité virtuelle, ce terme est appliqué aux équipements utilisés pour fournir un retour de force à l'utilisateur.

HBR : *Human Behaviour Representation*, représentation du comportement humain.

HITL : *Hardware In The Loop*, avec du matériel dans la boucle (voir simulation hybride).

HLA : High Level Architecture. Standard de simulation distribuée créé par le département de la défense américain. La conformité à HLA passe par le respect de dix règles, d'un formalisme objet et d'une spécification d'interface. HLA correspond à une approche « par composants » du développement des simulations (voir component-based design). Ce standard a remplacé ALSP et DIS.

IA : Intelligence artificielle.

IEEE : Institute for Electrical and Electronics Engineers, association internationale de standardisation dans le domaine de l'électricité et de l'électronique (voir page 302).

IHM : Interface homme-machine (GUI en anglais).

Immersif : Un environnement de réalité virtuelle est dit immersif s'il donne la sensation à l'utilisateur d'être plongé dans cet environnement (et par là même coupé du monde réel).

Implémentation : Codage informatique d'un algorithme ou d'un modèle.

Ingénierie système : Discipline ayant pour but d'organiser et de transformer les exigences client et les contraintes industrielles en une spécification et une architecture de système (matériels, logiciels et humains) satisfaisant les critères d'optimisation coûts-délais-performances.

Intelligence : Terme anglais pour le renseignement militaire, sous la responsabilité de la DRM en France et de la DIA aux USA (voir C4I).

Intelligence artificielle : branche de l'informatique traitant de la reproduction, par des machines, de certains aspects de l'intelligence humaine

Interaction : Action entreprise par un objet d'une simulation, qui peut être dirigé contre un autre objet ou son environnement. Exemple : collision, détonation, brouillage, détection…

Interarmées : Ce qui est commun à plusieurs armées (marine, armée de terre, armée de l'air). Le terme anglais est joint. Exemple : état-major interarmées. Ne pas confondre avec interarmes.

Interarmes : Ce qui est commun à plusieurs armes (infanterie, cavalerie, artillerie…). Le terme anglais est combined. Ne pas confondre avec interarmées.

Interface homme-machine : Partie d'un programme permettant à l'utilisateur de dialogue avec le système. Windows est un exemple d'IHM sur un système d'exploitation. Abréviation : IHM, GUI.

Internet : « L'Internet » (avec une majuscule) est Fédération de milliers de réseaux à l'échelle planétaire, utilisant le protocole TCP/IP. Un « internet » est un réseau basé sur les protocoles et services utilisés sur l'Internet (TCP/IP avec les services email, newsgroups, web, FTP, etc.).

Interopérabilité : Capacité d'une simulation (ou d'un modèle) à fournir et à accepter des services d'autres simulations et à utiliser les services ainsi échangés pour leur permettre de fonctionner effectivement ensemble.

Interpréteur : Logiciel exécutant directement un programme écrit dans un langage informatique de haut niveau, sans passer par une étape de compilation (voir compilateur). Les langages interprétés sont plus souples mais moins performants. Exemples : Postscript, Perl, Python, langages de scripts Unix (sh, awk…).

IIOP : *Internet Inter-ORB Protocol*. Protocole de communication entre ORB de CORBA.

ISO : International Standardization Organisation, organisme de standardisation international (voir page 301).

Java/Beans : Système d'objets distribués basé sur l'utilisation du langage Java. Proposé par la société Sun, il est fortement inspiré de CORBA.

Jeu de guerre : Voir wargame.

Joint : Terme anglais pour interarmées.

Kriegspiel : Jeu de guerre prussien, utilisant une caisse à sable pour faire évoluer des représentations des unités.

Langage orienté objets : Langage de programmation intégrant de façon directe les concepts d'encapsulation, d'héritage et de polymorphisme. Parmi les principaux langages à objets d'usage général, citons C++, Java, Ada95.

Langage formel : Langage à la syntaxe et la sémantique définies formellement, s'appuyant sur une base théorique bien identifiée et rigoureuse : mathématiques (théorie des ensembles, algèbres), logique... Un langage formel est généralement associé à une méthode formelle donnée.

LCG : *Linear Congruence Generator*. Voir GCL.

Live simulation : Voir simulation instrumentée.

LOD : *Level Of Detail*, niveau de détail (d'une base de données terrain).

Lookahead : Valeur associée généralement à un fédéré indiquant le délai minimal qu'il faudra à ce dernier pour émettre un message, i.e. si le temps du fédéré est t et que le lookahead est l, le time stamp du prochain message généré par le fédéré sera au moins $t+l$. Le lookahead est souvent lié au temps de traitement interne du fédéré, mais peut avoir d'autres causes.

LOS : *Line Of Sight*. Détermine le terrain et les objets directement visibles par une entité d'une simulation (i.e. qui ne sont pas physiquement masqués).

M&S : Abréviation de « modélisation et simulation », particulièrement usitée chez les anglo-saxons. En France on aura tendance à utiliser « la simulation » en lieu et place.

Maillage : Décomposition de l'espace en éléments discrets, suivant une grille régulière ou non.

Maintenabilité : Capacité d'un logiciel à être maintenu en condition opérationnelle. Voir MCO.

Maquette : Reproduction non fonctionnelle d'un système, servant généralement à valider des spécifications ou un design. Par exemple, on pourra faire une maquette d'un logiciel en se limitant à son interface graphique, afin d'en valider les spécifications. On pourra également faire une maquette à échelle réduite de plusieurs propositions architecturales pour un bâtiment afin de recueillir l'avis du client. À ne pas confondre avec un prototype, qui, lui, est fonctionnel.

MCO : Maintien en condition opérationnelle. Ce terme comprend les évolutions d'un logiciel ou d'un matériel, les corrections des défauts de conceptions (bugs…), les évolutions de ses constituants (changement de plate-forme par exemple), les évolutions des spécifications (rajout de fonctions).

Mémoire : Espace de stockage des données dans un ordinateur. La mémoire peut être sous forme de composants électroniques stockant des signaux électriques (mémoire **vive**) ou de dispositifs magnétiques (disques durs, bandes…) ou optiques (CDROM, DVD). Dans ce dernier cas, on parle de **mémoire de masse**, en raison de la grande capacité de ces supports.

Mémoire dynamique : mémoire allouée à un programme au fur et à mesure de ses besoins, et libérée après usage. La mémoire dynamique permet d'optimiser l'usage de la mémoire d'un ordinateur et de créer des structures de données dynamiques, mais sa manipulation est plus délicate que la **mémoire statique**, allouée une fois pour toute en quantité fixée lors du lancement du programme.

Méta-langage : Langage permettant de décrire d'autres langages.

Méthodes : En programmation orientée objets, sous-programmes s'appliquant à une classe d'objets.

Micro-terrain : Qualifie une zone du terrain d'une simulation où la résolution est plus élevée qu'ailleurs. Par exemple, alors que le terrain est modélisé avec une résolution de 100m, une zone urbaine utilisera une résolution inférieure à 10m pour permettre le positionnement précis des unités par rapport aux bâtiments.

Middleware : Logiciels utilisés entre un client et un serveur, par exemple pour assurer la traduction ou le routage de requêtes. L'ORB de CORBA et le RTI de HLA sont des exemples typiques de middleware.

Mission rehearsal : Répétition de mission. Consiste à faire effectuer une mission en simulateur avant de l'effectuer sur le terrain réel. Ainsi, les pilotes de bombardiers connaissent à l'avance l'objectif et leur zone d'opération, ce qui augmente considérablement leur efficacité… si toutefois la situation numérisée pour la simulation correspond bien à la réalité. Derrière la répétition de mission se cachent des besoins très lourds en moyens de simulation, production rapide et gestion de bases de données terrain et recueil de renseignements.

Mission space : Espace de mission d'un système. Comprend typiquement les missions et les matériels auxquels peut être confronté un système. Voir CMMS.

MITL (Man In The Loop) : « homme dans la boucle ». Se dit d'une simulation sur le déroulement de laquelle un opérateur humain peut agir. Exemples : un simulateur de vol, un jeu de guerre d'entraînement… (voir HITL)

Modèle : Représentation physique (maquette...) ou abstraite (mathématique, logique...) de la réalité (système, processus ou phénomène physique). Cette représentation peut être plus ou moins fidèle. Voir modélisation.

Modèle analytique : Modèle composé d'un ensemble d'équations décrivant les évolutions du système.

Modèle comportemental : Modèle décrivant le processus de décision d'un système réagissant à son environnement. Typiquement, il s'agit du comportement de la composante humaine du système : réactions cognitives (application d'un règlement, d'une doctrine…), physiologiques (stress, fatigue, douleur…) ou psychologiques (peur, soumission, colère, moral…).

Modèle conceptuel : Formulation du contenu et de la représentation interne du modèle combinant les concepts de l'utilisateur et du développeur. Il inclut la logiciel et les algorithmes et identifie les hypothèses et les limitations.

Modèle objet : Une spécification des objets intrinsèque à un système donné, incluant une description des caractéristiques (attributs) des objets et une description des relations statiques et dynamiques (interactions…) qui existent entre ces objets.

Modélisation : Action de réaliser et d'utiliser un modèle, en vue d'un objectif donné.

Monte-carlo : Méthode utilisée dans les simulations impliquant un modèle ou un processus stochastique, consistant à exécuter un grand nombre de répliques d'un scénario et à effectuer des calculs statistiques sur les résultats pour en tirer des lois générales.

MSCO : NATO Modeling and Simulation Coordination Office.

MSMP : Modeling and Simulation Master Plan, schéma directeur sur la simulation. Ce terme est généralement employé pour désigner le MSMP de l'OTAN ou celui du DoD.

MSRR : Modeling & Simulation Resource Repository de la défense américaine.

Niveau d'une simulation : Taille du système modélisé dans la simulation. Les niveaux couramment utilisés sont : phénomène physique, composant, sous-système, système, tactique, opérationnel, stratégique. Voir granularité, fidélité.

Objet : Élément de base de la représentation conceptuelle d'un système. Désigne typiquement une entité d'une simulation, notamment dans une simulation orientée objets. En programmation orientée objets, instance d'une classe.

OMG : Objet Management Group, voir page 302.

OMT : *Objet Model Template*, formalisme (méta-modèle) objet de HLA. Attention, en génie logiciel, OMT désigne aussi une technique de modélisation objet.

OOTW : Opérations autres que la guerre (maintien de la paix, aide humanitaire, lutte contre le terrorisme et le trafic de drogue…).

PADS : *Parallel And Distributed Simulation*, terme générique pour les simulation distribuées, qu'elles soient interactives ou non (voir DIS).

Pas d'intégration : Pas de temps utilisé pour la résolution des équations d'état d'un système (i.e. décrivant son évolution, son mouvement…).

Pas de temps : Temps entre deux itérations d'un calcul. Dans une simulation, il s'agit typiquement du temps entre deux calculs successifs de l'état du système.

Pathfinder : Terme très utilisé chez les anglo-saxons pour nommer un projet exploratoire, par exemple une expérimentation pour évaluer des technologies avancées. C'est ainsi que le programme OTAN de fédération HLA expérimentale pour les CAX a-t-il été baptisé Pathfinder.

PDU : Protocol Data Unit, messages échangés par les simulations dans le cadre du protocole DIS. Exemple de PDU : entity state, detonation…

Perceived truth : Un sous-ensemble de la réalité (ground truth) acquise, et éventuellement altérée, par des capteurs ou la perception et le jugement humain.

Photogrammétrie : Technique permettant d'obtenir rapidement des modèles 3D de terrains et structures à partir de photos 2D (satellites, reconnaissance aérienne...) et d'un nombre limité de relevés topographiques.

Pixel : Picture Element, désigne un point individuel faisant partie d'une image bitmap. Exemple : le mode vidéo graphique SVGA correspond à une résolution de 800x600 pixels.

Planification : Organisation d'une opération (militaire). On distingue la « planification à froid », dans laquelle on détermine en période de paix des plans de déploiement et d'actions des forces pour des scénarios types, et la « planification à chaud », qui se déroule lors du lancement d'une opération militaire réelle. Les contraintes de délais n'étant pas les mêmes dans les deux cas, les méthodes et les outils peuvent différer sensiblement.

POO : Programmation orientée objets (*OOP* en anglais).

Portabilité : Aptitude d'un logiciel à être adapté dans des délais et avec des efforts raisonnables sur des plates-formes informatiques différentes (matérielles et logicielles) avec des fonctionnalités identiques et des performances comparables. Ne pas confondre avec évolutivité.

Processus : Ensemble des états pris par une entité au cours du temps.

Processus de Markov : Processus stochastique dans lequel la probabilité d'occurrence d'un événement ne dépend que des événements qui ont précédé.

Processus stochastique : Processus mettant en jeu des événements dont le survenue dans le temps ne peut être décrit qu'en termes probabilistes.

Programmation orientée objets (POO ou *OOP*) : utilisation d'un système de programmation dans lequel les programmes sont organisés en un ensemble d'objets représentant des instances d'une classe particulière. Chacune de ces classes font partie d'une hiérarchie de classes, reliées entre elles par des mécanismes d'héritage. Pour la POO, on utilise de préférence un langage orienté objets.

Programmation par contraintes (PPC) : Méthode de résolution de problèmes combinatoires par la descriptions des exigences sous forme de règles (contraintes sur les variables du problème) qui sont ensuite analysées par un solveur spécifique qui calcule la meilleure solution. La PPC a de nombreux domaines d'applications, en particulier la planification et la logistique.

Protocole : Ensemble de conventions définissant le format logique, physique et temporel des messages échangés entre des matériels ou des logiciels se communiquant des données (DIS et ALSP sont des protocoles, mais pas HLA puisque le format des messages échangés ne fait pas partie du standard).

Prototype : Exemplaire d'un système destiné aux tests et à la mise au point de la version de série. Contrairement à la maquette, un prototype est tout à fait fonctionnel. Certains prototypes ont d'ailleurs pu, moyennant une mise à niveau, devenir des systèmes opérationnels (exemple : prototypes du bombardier furtif B2 américain).

Prototype virtuel : Une simulation d'un système placée dans un environnement synthétique et utilisée pour la mise au point, les tests et l'évaluation des spécifications, de la conception ou du maintien en condition opérationnelle du système.

Qualité : Ensemble de méthodes visant à assurer l'adéquation d'un produit au besoin, au meilleur coût. Quand le processus concerné est le développement d'un programme informatique, on parle de « qualité logicielle ». La qualité d'une simulation est assurée par le processus de VV&A.

Queue : Voir file d'attente.

Réalité augmentée : Superposition d'images synthétiques à la vision du monde réel. Un dispositif de réalité augmentée n'est donc pas immersif.

Réalité virtuelle : Qualifie un environnement synthétique immersif, doté d'une IHM permettant à un opérateur de visualiser, de manipuler et d'interagir avec des données ou un environnement complexes.

Recherche opérationnelle : Techniques d'analyse et de résolution de problèmes, concernant notamment les activités économiques (organisation d'une chaîne de production...) et militaires (planification...), et visant à déterminer la meilleure décision pour parvenir à un résultat donné. La simulation est un des outils de la R.O.

Rehearsal : Voir Mission rehearsal.

Répétition de mission : Voir Mission rehearsal.

Réplique : Une exécution d'une simulation (notamment avec l'intention d'en faire plusieurs, par exemple pour effectuer une analyse des résultats par Monte-Carlo). En anglais : *run* ou *replication*.

Repository : Structure permettant le stockage à des fins de consultation et réutilisation de documentation, modèles et simulations informatiques. Peut prendre la forme d'une bibliothèque de modèles. Exemple : *M&S ressource repository* (MSRR) du DoD.

Réseau neuronal : Système informatique dont le fonctionnement est basé sur celui des neurones du cerveau : des unités neuronales sont stimulées en entrée par des connections (pondérées) avec les unités voisines. Ces stimuli, en fonction des seuils programmés, déclenchent les sorties. Le réseau neuronal évolue donc jusqu'à atteindre l'équilibre, l'état final donnant la réponse au problème posé (par exemple la récupération d'informations).

Retour de force : *Force feedback*. Utilisation de périphériques haptiques dans le cadre d'une simulation afin de donner à l'utilisateur des informations tactiles (sens du toucher, résistance d'un manche à balai, etc.).

Réutilisabilité : Capacité d'un logiciel à être réemployé tout ou partie pour des objectifs différents de ceux pour lesquels il a été initialement conçu et développé.

RPR-FOM : Modèle objet (voir FOM, OMT) pour une fédération HLA facilitant l'intégration de simulations DIS.

RTI : Run-Time Infrastructure. Dans le standard HLA, logiciel de type « middleware » fournissant les services d'interface durant l'exécution d'une fédération.

SACOD : Simulation pour l'aide à la conception de l'outil de défense.

SAF ou **SAFOR** : *Semi-Automated Forces*. Simulation d'entités de type CGF, généralement à des niveaux plates-formes, modélisant les décisions de façon plus ou moins complète les décisions de l'équipage (engagement d'un ennemi découvert, contournement d'un obstacle…). Toutefois, les décisions de haut niveau (commandement, mission, conduite à tenir…) ou le maintien du réalisme d'ensemble requièrent la présence dans la boucle d'un opérateur humain, d'où le qualificatif de « semi-automatique ».

SBA : Simulation Based Acquisition. Voir Simulation pour l'acquisition.

Scalability : Capacité d'un logiciel à supporter une augmentation de sa charge (par exemple le nombre d'entités et d'interactions pour une simulation) tout en restant opérationnel, i.e. en conservant un niveau de performances convenable.

Scénario : Description des conditions initiales et des événements programmés durant l'exécution d'une simulation. Déroulement d'un exercice.

SEDRIS : *Synthetic Environment Data Representation and Interchange Specification*, standard permettant l'interopérabilité des bases de données d'environnement entre elles et avec des simulations.

SI : Abréviation pour désigner le système métrique international.

SIC : Système d'information et de commandement. Désigne des moyens utilisés pour faire remonter de l'information obtenue suivant différentes sources (observateurs sur le terrain, bases de données, satellites, etc.) vers des dirigeants (généraux ou directeurs de sociétés), leur présenter de façon synthétique, les aider à prendre une décision et à faire redescendre cette décision vers les échelons d'exécution. Voir C3I. Attention, le sigle « SIC » est également utilisé dans le sens de « systèmes d'informations et de communication », qui est beaucoup plus large.

SIF : Standard simulation database Interchange Format, destiné à l'échange de données d'environnement entre des simulations. Voir SEDRIS.

SIMNET : Démonstrateur américain de réseau de simulations (1987). Dédié au temps réel (simulation pilotée), SIMNET avait de nombreuses limitations et imperfections. Néanmoins, ce fut un laboratoire qui permit aux américains d'acquérir les connaissances techniques nécessaires pour les projets qui s'ensuivirent (DIS, ALSP, HLA).

Simulateur : Dispositif de simulation pilotée.

Simulation : Implémentation dynamique (dans le temps ou l'espace) d'un ou plusieurs modèles dans un but déterminé. Désigne également l'activité d'utilisation de ces modèles en vue d'un objectif donné, ainsi que l'ensemble des techniques de modélisation et simulation (ce que les anglo-saxons appellent M&S).

Simulation constructive : Simulation numérique faisant intervenir une modélisation du facteur humain. Un opérateur peut être dans la boucle, mais sans être la source principale de stimuli de la simulation.

Simulation distribuée : Simulation constituée de composants autonomes. Cette autonomie est généralement destinée à permettre leur fonctionnement sur des machines différentes, afin, par exemple, de répartir les calculs. Elle permet également une approche par composants de la conception des simulations (voir HLA, DIS).

Simulation à événements discrets : Simulation dont l'évolution est basée sur l'arrivée d'événements (exemples : arrivée d'un client, panne, détection d'une cible…). Voir DES.

Simulation hybride : Simulation comprenant du matériel réel, typiquement à des fins d'essais de ce matériel (qualification, test…).

Simulation interactive : Simulation dont certaines des entrées sont fournies par l'opérateur humain (voir simulation constructive, simulation pilotée).

Simulation instrumentée : Simulation faisant intervenir des équipements réels sur le terrain, dûment équipés. Par exemple, des fantassins dont les fusils sont munis d'émetteurs lasers pour simuler les tirs. Parfois désignée par l'anglicisme « simulation vivante » (live simulation).

Simulation numérique : Simulation entièrement logicielle.

Simulation numérique fermée : Simulation numérique dans laquelle aucun opérateur humain n'est dans la boucle. Voir simulation scientifique.

Simulation opérationnelle : Simulation destinée à être employée par les armées. Voir simulation technique.

Simulation pilotée : Simulation dans laquelle la principale source d'entrées vient d'un opérateur humain, et dont l'interface reproduit celle du système réel. Exemple : simulateur de vol. Voir simulateur, entraîneur.

Simulation pour l'acquisition : Doctrine d'emploi de la simulation dans laquelle celle-ci est intégrée d'une façon cohérente tout au long du cycle de développement d'un produit. Cela implique notamment une méthodologie commune à tous les stades de ce cycle et un fort taux de réutilisation des outils, des modèles ou des résultats de ces modèles d'un stade à l'autre.

Simulation scientifique : Se dit d'une simulation numérique fermée, modélisant finement un phénomène physique. Exemple : écoulement d'un fluide, rayonnement électromagnétique…

Simulation technique : Se dit d'une simulation utilisée à des fins d'ingénierie. Voir simulation opérationnelle, simulation pour l'acquisition.

Simulation technico-fonctionnelle : Ce type de simulation vise à évaluer un système ou un sous-système dans son contexte d'utilisation opérationnelle, à des fins d'ingénierie, par exemple pour évaluer ou qualifier un produit. Voir simulation technico-opérationnelle.

Simulation technico-opérationnelle : simulation d'un système (de niveau généralement système, tactique, voire opérationnel), afin d'en évaluer l'efficacité ou la vulnérabilité militaire, dans un contexte opérationnel défini par un scénario d'emploi et une menace. Les simulations T.O. sont notamment utilisées pour évaluer des concepts, aider à la spécification de systèmes et mettre au point les doctrines d'emploi. Voir simulation technico-fonctionnelle, SACOD

Simulation virtuelle : Traduction de l'anglais *Virtual Simulation*. Synonyme de simulation pilotée, bien que le terme « virtual simulation » soit plus vaste (par exemple, une simulation de SIC entre dans cette catégorie, bien qu'elle ne soit pas une simulation pilotée).

SISO : Simulation Interoperability Standards Organization. Voir page 301.

SOM : *Simulation Object Model*. Dans HLA, ensemble de tables suivant le formalisme des OMT, et décrivant les objets, attributs, interactions et paramètres qu'il est susceptible de partager avec le reste de la fédération.

Sous-système : Partie d'un système réalisant une fonction donnée. Ex. : le sous-système « guidage-navigation » d'un missile.

Spécification : Expression du besoin d'un utilisateur pour un nouveau système ou pour l'évolution d'un système existant.

STANAG : Standard OTAN. Voir DIS, HLA.

Stimulation : Dans un CAX, phase préliminaire consistant à transférer les connaissances opérationnelles nécessaires de la situation tactique et géopolitique d'un scénario aux entraînés, par l'intermédiaires d'outils de simulation, de SIC, de vidéos, de messages, etc.

Stochastique : Qualifie un processus, une variable ou un modèle dont le résultat dépend du hasard (par opposition à déterministe).

STOW : *Synthetic Theater Of War*, démonstrateur technologique américain, auquel participent également les britanniques, afin de démontrer et d'étudier le potentiel de la simulation distribuée. STOW est un projet extrêmement ambitieux, dans lequel sont créés des environnements synthétiques répartis sur plusieurs sites à travers le territoire américain et en Europe, et comportant plusieurs milliers d'entités. Plusieurs démonstrations concluantes ont déjà eu lieu.

Structure d'accueil : ensemble de méthodes, de logiciels et, parfois, de matériels, permettant de réaliser une simulation par intégration de modèles ou de simulation existants (en anglais : *simulation framework* ou *simulation support environnement*). On peut distinguer les structures d'accueil de modèles et les structures d'accueil de simulations.

Sur étagère : Se dit d'un produit existant déjà et que l'on peut réutiliser. Voir COTS, GOTS.

Synthetic environment : Voir environnement synthétique.

Système : Ensemble de matériels, logiciels, personnels et méthodes organisés de façon à remplir une mission donnée.

Système complexe : Système dont le nombre d'éléments ou le nombre d'interactions entre ces éléments ou l'intervention d'un facteur complexifiant (composante humaine, caractère dynamique...) rendent difficile, voire impossible, à appréhender par les procédés d'ingénierie habituels. Typiquement, un système complexe est un système dont les propriétés émergeantes (notamment celles issues des couplages rétroactifs) deviennent prépondérantes, de sorte qu'il n'est plus analysable par décomposition.

Système expert : Système informatique capable de collecter de l'information (base de faits) et des connaissances (généralement représentées sous la forme d'une base de règles), et, grâce à un moteur d'inférence, d'en déduire de nouvelles connaissances, dans le but de répondre à des requêtes. Par exemple, à partir de l'information « Paul est le fils de Pierre » et « Pierre est le fils de Jean » et des règles « Si A est fils de B, alors B est père de A » et « Si A est père de B et B père de C, alors A est grand-père de C », le système expert en déduira que Jean est le grand-père de Paul.

TCP/IP : Transport Control Protocol / Internet Protocol. Protocoles de communication utilisés dans les réseaux de type internet, respectivement de niveaux transport (constitution des paquets de données) et réseau (routage des paquets d'un ordinateur à l'autre).

Temps logique : Temps interne d'une simulation (ou d'un membre d'une fédération de simulations). Ce temps logique est le temps utilisé par cette simulation. Il peut différer du temps de la fédération ou du temps réel.

Temps processeur : ou « temps CPU ». Temps pendant lequel un processus a utilisé un processeur d'un ordinateur, dans un environnement multitâches. Par exemple, si un logiciel de tri a mis 10 secondes de temps réel pour s'exécuter en monopolisant 50% du temps le processus d'un ordinateur monoprocesseur, le temps CPU sera de 5 secondes. Attention en revanche, sur une machine quadriprocesseur, ce temps CPU serait de 20 secondes (5×4). Le temps CPU, toujours inférieur ou égal au temps réel sur une machine monoprocesseur, peut être supérieur si l'on a plusieurs CPU. Cette notion est importante pour l'analyse des performances d'un système informatique.

Temps réel : En simulation, qualifie une application pour laquelle le temps logique s'écoule à la même vitesse que le temps du monde réel. Ainsi, si on exécution un simulateur pendant dix secondes réelles (mesurées avec un chronomètre), le temps logique du simulateur aura avancé de dix secondes.

Temps simulé : Le temps tel qu'il est représenté au sein d'une simulation.

Time management : Mécanisme de gestion du temps d'une simulation.

Time stamp : « Timbre à date », attribut attaché à un objet (événement, message, PDU…) indiquant, par exemple, la date de sa génération.

Time Stamp Order (TSO) : Ordre chronologique.

UML : Unified Modeling Language, un langage pour spécifier, visualiser (grâce à une symbolique graphique), construire et documenter les systèmes à base de logiciels. UML est aujourd'hui très répandu, et est supporté par un large nombre d'outils de génie logiciel.

Validation : Processus visant à s'assurer qu'un modèle ou une simulation représente le monde réel d'une façon suffisamment précise pour remplir les besoins d'une utilisation donnée. Il est très important de noter que la validation n'est valable que pour un domaine d'emploi donné et doit être remise en question pour toute nouvelle utilisation sortant de ce domaine.

Variable : Une donnée dont la valeur peut changer. Exemple : position d'une entité, commande entrée par l'utilisateur, quantité de mémoire restante… (voir constante).

Variables d'état : Variables décrivant l'état du système (vitesse, position, température…). En particulier, si on sauvegarde lors d'une simulation l'ensemble des variables d'état d'un système simulé, la restauration de ces variables doit permettre de retrouver le système dans l'état exact où il se trouvait.

Vectoriel : Type de représentation d'une image dans lequel l'objet ou la scène sont décomposés en segments de droites et en surfaces élémentaires. Ce format a l'avantage de prendre peu de place et de pouvoir être agrandi à volonté sans perte de qualité, contrairement au bitmap.

Vérification : Processus visant à s'assurer que l'implémentation d'un modèle ou d'une simulation correspond bien à la spécification qui en a été faite. La vérification se fait par examen du code et par des procédés basés sur les techniques de génie logiciel. De préférence, la vérification doit se faire tout au long du développement, et non uniquement a posteriori.

Virtual simulation : Voir simulation virtuelle.

Virtuel : Qualifie un objet ou un environnement simulé informatiquement.

VV&A Vérification, Validation et Accréditation (de simulations).

VV&C Vérification, Validation et Certification (de données).

Wallclock time : Littéralement « temps de la pendule ». Temps issu d'une horloge matérielle (i.e. l'horloge interne d'un ordinateur).

WAN : Wide Area Network, réseau interconnectant des ordinateurs situés à de grandes distances les uns des autre (typiquement au moins plusieurs kilomètres). Voir aussi : LAN.

Wargame : Jeu de guerre. Simulation de combat dans laquelle le comportement humain n'est pas entièrement simulé, mais est assuré (totalement ou en partie) par la présence de joueurs. Les principales utilisations des jeux de guerre sont la formation, l'entraînement ou l'analyse de situation tactique ou stratégique. Ils sont souvent constitués de camps qui s'affrontent sur un terrain donné. Ces wargames peuvent être informatiques (ce sont alors des simulations constructives) ou sur plateau (carte papier, pions pour représenter les combattants, dés pour simuler le hasard...).

XML : *eXtensible Markup Language*. Langage standard de description de documents et données structurées.

12. PRINCIPAUX ACTEURS FRANÇAIS DE LA SIMULATION

Il est difficile de lister tous les acteurs français de la simulation, tant il y en a. Néanmoins, voici une liste qui peut servir de point de départ, et que j'espère compléter avec le temps.

12.1 Centres du Ministère de la Défense

12.1.1 DGA

CAD : Le Centre d'Analyse de Défense est en charge de la simulation techni-opérationnelle. Il travaille directement au profit des états-majors, des architectes de systèmes de forces (ASF) et des services de programmes. C'est un centre d'expertise où la simulation joue un rôle de premier plan.

DCE : La Direction des Centres d'Expertise et d'essais utilise beaucoup la simulation à des fins d'expertise, d'évaluation et des qualification des matériels, utilisant des techniques telles que la simulation scientifique, la simulation technico-fonctionnelle et le prototypage virtuel.

Services de programmes : Les services de programmes de la DGA gèrent les programmes d'armement, lesquels peuvent comprendre des simulations. Ainsi, le SPOTI a-t-il des activités dans la simulation interarmées. Ce service est responsable de l'ambitieux projet SCIPIO, jeu de guerre destiné à l'entraînement d'état-major de l'armée de terre. Le SPART, le SPAé et le SPN pilotent les programmes de simulateurs d'entraînement dans les domaines terrestres, aériens et marins respectivement.

STTC : Le département d'ingénierie des systèmes complexes (SC) du Service Technique des Technologies Communes a parmi ses mission l'organisation et la coordination de la simulation destinée à l'acquisition. Dans la politique technique qu'il a définie pour son domaine, il ambitionne de créer une infrastructure technique commune de simulation, afin de rationaliser les développements en matière de simulation. Plusieurs projets ont été lancés dans ce domaine, notamment sous forme de programmes d'études amont (PEA). En outre, SC anime le réseau d'échange sur la simulation (RESIM), qui permet à des experts et spécialistes, essentiellement de la DGA et de l'ONERA, mais aussi des états-majors, de dialoguer et d'échanger des informations.

12.1.2 Armée de Terre

CROSAT : Le Centre de Recherche Opérationnelle de l'Armée de Terre mène des études dans le domaine des jeux de guerre, et de l'analyse des systèmes d'armes et de forces. Il est situé à Paris.

CENTAC : Le Centre d'Entraînement Tactique à Mailly est remarquable par ses moyens, et par son utilisation de la simulation instrumentée, encore rare en France.

Autres : L'armée de terre dispose de nombreux centres de formation de taille moindre que le CENTAC, équipés de simulateurs, par exemple dans les écoles d'applications (EA de l'Artillerie à Draguignan, EA de la Cavalerie à Saumur, etc.).

12.1.3 Armée de l'Air

CEC/CEAM : Le Centre d'Entraînement au Combat, à Mont-de-Marsan, forme les pilotes de chasse et participe à certaines études et recherches sur les tactiques de combat aérien et la mise au point des simulations.

CASI : La Cellule d'Analyse, de Simulation et d'Innovation, situé à Paris, mène des études technico-opérationnelles et assure la cohérence des activités de simulation opérationnelle de l'armée de l'air.

CASPOA : Centre d'Analyse, de Simulation et de Planification des Opérations Aériennes.

Autres : Simulateurs et entraîneurs des centres de formation et d'entraînement.

12.1.4 Marine

ANPROS : Antenne de Planification, de Recherche Opérationnelle et de Simulation de la Marine.

Autres : Comme l'armée de terre et l'armée de l'air, la marine dispose de nombreux simulateurs dans ses centres de formation, par exemple les simulateurs de conduite de sous-marins à Brest.

12.1.5 Autres

ONERA : L'Office National d'Études et de Recherches Aérospatiale utilise largement la simulation, notamment scientifique, dans ses études. L'ONERA a également des experts en simulation distribuée et en sécurité.

12.2 Autres centres étatiques

De nombreux centres étatiques pratiquent la simulation, mais, pour l'instant, leur influence est assez limitée en matière méthodologique ou technologique. En effet, leur activité est essentiellement dans le domaine de la simulation scientifique[143], traditionnellement peu structurée. Il n'est pas possible d'en faire une liste exhaustive, car ils sont nombreux (particulièrement dans le milieu académique) mais citions tout de même l'INRIA[144], le LAAS (à Toulouse) et l'IMAG (à Grenoble), qui ont réalisé un nombre important de travaux dans les technologies de simulation.

Toutefois, il semble qu'il y ait une prise de conscience des enjeux de la simulation et de la nécessité « de mettre de l'ordre ». Ainsi, l'AFNOR a créé un groupe de travail pour rédiger une norme sur « les démonstrations autres que les essais » (typiquement les simulations). L'AFIS[145], quant à elle, accorde une grande importance à la simulation comme outil d'ingénierie système.

12.3 Industriels

De plus en plus d'industriels s'intéressent à la simulation et développent des activités dans ce domaine, de sorte qu'il est difficile d'être exhaustif dans ce domaine. Notamment, tous les grands groupes industriels proposent des services en matière de simulation.

Je m'abstiendrai donc d'essayer d'en dresser une liste, même partielle. Si vous souhaitez prospecter des sociétés susceptibles de travailler dans la simulation, je vous encourage à vous rendre dans les salons spécialisés (MiCAD[146], ITEC[147], I/ITSEC[148]…), à lire la presse spécialisée de votre domaine, et à effectuer des recherches sur Internet.

[143] Notamment, il y a peu d'activité dans le domaine des technologies générales de simulation. Les principaux efforts portent sur les architectures parallèles, les techniques de maillage et les solveurs d'équations.

[144] Qui est un acteur majeur du développement de l'atelier de simulation Scilab.

[145] Association française d'ingénierie système.

[146] Salon de la conception assisté par ordinateur, une fois par an à Paris.

[147] Salon/Conférence de l'industrie de simulation, une fois par an à La Haye (Pays-Bas).

[148] Salon/Conférence de la simulation, une fois par an aux USA.

12.4 Internationaux

12.4.1.1 CEPA11

Le programme EUCLID (*European Co-operation for the Long Term In Defense*) a été lancé en 1990, avec le but de développer des bases technologiques européennes pour la défense en Europe (UEO[149], et pas simplement CEE).

Les projets EUCLID (les RTP[150]) sont élaborés au sein de comités directeurs des CEPA (*Common European Priority Area*). Le CEPA 11 s'occupe des facteurs humains, dont la simulation, notamment pour l'entraînement, fait partie.

Le CEPA 11 a ainsi lancé plusieurs études sur les environnements synthétiques, surtout pour la simulation pilotée. Le dernier projet en date, le RTP 11.13, regroupe quinze pays, et vise à réaliser un démonstrateur de simulation distribuée très ambitieux.

12.4.2 Groupes OTAN

MSCO : Le Modeling & Simulation Coordination Office est le point focal de la simulation à l'OTAN. Il est chargé de coordonner l'exécution du schéma directeur (voir [OTAN98]) et de fournir un certain nombre de services à la communauté M&S de l'OTAN. Il identifie et facilite les opportunités de coopération dans son domaine et coordonne les activités de standardisation liées.

SGMS : Le *Steering Group on NATO M&S policy and applications* a été notamment chargé de collecter le besoin en matière de simulation auprès de sous-groupes militaires (MPSG), gouvernementaux (GPSG) et industriels (IPSG), afin de développer le schéma directeur M&S de l'OTAN.

NMSG : Le *NATO Modeling & Simulation Group* est chargé de la gestion de la politique M&S et de la coordination des activités dans ce domaine. Il supervise l'implémentation du schéma directeur (voir [OTAN98]), soutient les activités M&S, propose des financements pour les coopérations et identifie les besoins afin de faire évoluer les développements. Enfin, il supervise l'exécution du plan de travail du MSCO.

RTO : La *Research & Technology Organisation* est le point focal à l'OTAN de la recherche et de la technologie pour la défense. Sa mission est de conduire et de promouvoir les recherches en coopération et les échanges d'informations. Elle comprend le RTB (*R&T Board*), qui est l'organe décisionnel de haut niveau, et la RTA (*R&T Agency*), basée en France, à Neuilly. Les activités de la RTO sont couverts par 7 *panels*, dont le SAS (*Studies, Analysis and Simulation*), qui s'occupe de plusieurs projets et études dans le domaine de la simulation.

[149] Union de l'Europe Occidentale.
[150] Research and Technology Project.

NC3A : La *NATO C3I*[151] *Agency* n'est pas à la base une agence de simulation, mais est fortement impliquée dans les exercices assistés par ordinateurs, les CAX, qui sont utilisés pour les entraînements des états-majors de l'OTAN, et font largement appel à la simulation (jeux de guerre, CGF…).

12.4.3 AFDRG/WG11/SG3

Le sous-groupe 3 du groupe de travail 11 de l'*Anglo-French Defence Research Group* est consacré aux environnements synthétiques. Il a été créé afin de coordonner les activités de simulations au sein de l'AFDRG. Structure d'échange, le SG3 a aussi réalisé des études (simulation distribuée, VV&A).

À la fin de l'année 2000, l'AFDRG a été réorganisé. Les travaux du WG11/SG3 sont maintenant sous la responsabilité du WG6.

12.4.4 SISO

La SISO (Simulation Interoperability Standards Organization) a pour objectif de promouvoir l'interopérabilité entre les simulations et la réutilisation de composants, notamment par la standardisation. Les ateliers SISO, qui se tiennent deux fois l'an à Orlando, aux USA, sont l'occasion d'échanges très denses dans le domaine de la simulation distribuée. Outre l'organisation de cette grand-messe de la simulation, la SISO publie des documents et met sur pied des formations sur des thèmes tels que HLA.

12.4.5 ISO

L'International Standardization Organisation est un organisme international en charge de l'élaboration et de la diffusion de standards. Parmi les standards ISO : ISO-8859 (jeu de caractère), ISO/IEC/8652:1995 (langage informatique Ada95), ISO/IEC DIS 9899 (Langage C), ISO-35.100 (couches réseaux OSI)…

12.4.6 SCS

La SCS (Society for Computer Simulation) a été fondée en 1952. Elle compte parmi ses membres de multiples professions (ingénieurs, chercheurs, enseignants, étudiants, businessmen, cadres dirigeants) qui s'intéressent aux diverses applications de la simulation, ainsi qu'aux outils de simulation (logiciels, langages, matériels…).

L'association publie plusieurs périodiques, dont la revue *Simulation*, ainsi que de nombreux ouvrages, dont une grande partie sont des actes de colloques (« transactions » ou « proceedings » en anglais).

[151] Command, Control, Communications and Intelligence, terme équivalent au sigle français SIC (système d'information et de commandement), avec la dimension renseignement en plus.

12.4.7 IEEE

Fort de plus de 300 000 adhérents dans 150 pays, l'IEEE (Institute of Electrical and Electronics Engineers) promeut l'innovation dans les domaines de l'électricité, de l'électronique et de l'informatique. Cette organisation vieille de plus d'un siècle (bien que n'ayant pris sa forme actuelle qu'en 1963) publie de nombreuses revues et ouvrages, et organise plus de 300 conférences par an. Elle publie également des standards, tels que l'IEEE 1278 (DIS), le 802.3 (Ethernet) ou le 1394 (interface Firewire).

12.4.8 OMG

L'*Object Management Group* a été fondé en 1989 par onze compagnies, essentiellement d'informatique (HP, Sun, 3Com…) mais aussi des utilisateurs (American Airlines). Aujourd'hui fort de 800 membres, ce consortium cherche à développer l'utilisation de composants objets par l'industrie, notamment en rédigeant des spécifications pour le logiciel indépendantes des fournisseurs.

Son produit phare est CORBA, un *middleware* permettant la communication entre des applications. L'OMG a également développé un protocole de communication sur Internet, l'IIOP (*Internet Inter-ORB Protocol*), qui est utilisé par des sociétés comme Netscape, Oracle ou IBM.

12.4.9 ISAG

L 'International Simulation Advisory Group regroupe une trentaine de pays ayant un intérêt dans le domaine M&S. L'un des principaux buts de l'ISAG est de suivre le processus de standardisation entrepris aux États-Unis par le SISO et de s'assurer que les intérêts de la communauté internationale sont bien pris en compte (et pas seulement les besoins des USA).

12.5 Etats-Unis

L'organisation actuelle (1999) de la simulation aux USA est définie par la directive DoD 5000-59 (voir [DoD95]).

12.5.1 USD A&T

La simulation de défense américaine est placée sous l'autorité du sous-secrétaire d'état à l'acquisition et à la technologie (USD A&T), qui chapeaute donc l'EXCIMS et le DMSO.

12.5.2 EXCIMS

Le Conseil Exécutif pour la Modélisation et la Simulation (EXCIMS) est notamment chargé de définir et réviser les objectifs (données par le schéma directeur de la simulation), et de superviser l'application de la politique de simulation.

12.5.3 DMSO

Le *Defense Modeling and Simulation Office* sert de secrétariat à l'EXCIMS et fournit un point focal permanent pour les activités M&S : promotion de la politique de l'EXCIMS, diffusion de documentation, coordination des coopérations, organisation de séminaire, pilotage de projets…

Le site web du DMSO est notamment un point de départ incontournable de toute recherche sur les standards de simulation.

12.6 Autres pays

Il est difficile de détailler ici l'organisation de tous les pays en matière de simulation, car elle est très variée !

Citons néanmoins la DERA (Defence Evaluation & Research Agency) britannique, le TNO des Pays-Bas et l'IABG en Allemagne, qui sont les leaders de la simulation de défense dans leur pays respectif.

Signalons également que le Royaume-Uni a mis sur pied un bureau spécial pour les environnements synthétiques[152], montrant l'importance que cette nation, traditionnellement très influencée par les USA, attache à la mise en place d'une démarche globale en matière de simulation.

Notons enfin qu'il existe de nombreuses manifestations nationales ou internationales dans le domaine de la simulation, dont il est difficile de faire un inventaire exhaustif. Toutefois ces événements sont largement documentés sur Internet et dans la presse spécialisée.

[152] Synthetic Environments Co-ordination Office (SECO)

13. BIBLIOGRAPHIE

La liste d'ouvrages et d'articles traitant de la simulation est très longue et pourrait s'étaler sur des centaines de pages. Je me limiterai donc ici à citer les ouvrages et articles qui m'ont réellement aidé à réaliser ce cours, ou que l'on peut considérer comme des références.

[ADEL00] *Modélisation et simulation de systèmes complexes*
M. Adelantado, G. Bel, P. Siron, formation SAE S15, 2000.

[AGAR96] *Flight Simulation, Where are the Challenges ?*
NATO AGARD Conference proceedings 577, 1996.

[ANDR95] *Histoire de l'informatique, quatrième colloque sur l'histoire de l'informatique, Rennes, 14-16 novembre 1995*
J. André, P.-E. Mounier-kuhn et divers, IRISA/INRIA-Rennes, 1995.

[ARNO97] *Rapport d'étude sur la simulation dans la Défense*
ICETA J. Arnold et ICETA D. Tanfin, mémoire EMS2, DGA, 1997.

[BANK95a] *Discret-Event System Simulation*
J. Banks, B. Nelson, J. Carson, Prentice Hall, 1995.

[BANK95b] *Getting Started with GPSS/H*
J. Banks, J. Carson, J. Sy, Wolverine Software Corp., 1995.

[BANK98] *Handbook of Simulation*
J. Banks & divers, Wiley-Interscience 1998.

[BART98] *Guide to Constraint Programming*,
R. Barták, 1998, *http://kti.ms.mff.cuni.cz/~bartak/constraints/*.

[BROO75] *The Mythical Man-Month*
F. P. Brooks, Addison-Wesley, 1995 (éd. originale 1975).

[BURD93] *La Réalité Virtuelle*
G. Burdea, P. Coiffet, Hermès 1993.

[BURD94] *Virtual Reality Technology*
G. Burdea, P. Coiffet, John Wiley & Sons, 1994.

[BURD96] *Force & Touch Feedback for Virtual Reality*
G. Burdea, John Wiley & Sons, 1996.

[CANT94] *La réalité virtuelle et ses applications, tomes I (applications militaires)*
P. Cantot, Centre d'Analyse de Défense 1994

[CANT95] *La réalité virtuelle et ses applications, tomes II (applications civiles*
P. Cantot, Centre d'Analyse de Défense 1995

[CANT00] *Modélisation et simulation : une nouvelle ère commence*
P. Cantot, P. Dropsy, Revue scientifique et technique de la défense, ADDIM, octobre 2000.

[CARP98] *La recherche aéronautique et les progrès de l'aviation*
J. Carpentier, revue scientifique et technique de la défense (n°40), 1998.

[CAST97] *Would-Be Worlds: how simulation is changing the frontiers of science*
J. L. Casti, Wiley, 1997.

[DENE96] *Les systèmes d'information géographique,*
J. Denègre, F. Salgé, PUF, 1996.

[DGA97] *La simulation de défense – Guide dictionnaire (édition Avril 1997)*
édité par la DGA (STTC/DT/SC)

[DOCK93] *The Military Landscape : Mathematical Models of Combat*
J. T. Dockery, A. E. R. Woodcock, Woodhead Publishing Ltd, 1993.

[DOD95] *Modeling and Simulation Master Plan (directive 5000.59)*
US Under Secretary of Defense for Acquisition and Technology, 1995.

[DOD96] *DoD Modeling and Simulation (M&S) Verification, Validation and Accreditation (VV&A) (Instruction 5000.61)*
US Under Secretary of Defense for Acquisition and Technology, 1996 (remise à jour en 2000).

[DOD97] *DoD Modeling and Simulation (M&S) Glossary.*
Appendice M de [DOD95], 1997.

[DMSO98] *SEDRIS documentation set*
US Defense Modeling and Simulation Office, 1998.

[DMSO99] *Federation Development and Execution Process (FEDEP) Model, version 1.54*
US Defense Modeling and Simulation Office, 1999.

[DUNA87] *Maîtriser la qualité et les coûts des produits et projets*
M. Dunaud, Masson, 1987.

[DUNN92] *The Complete Wargame Handbook : How to Play, Design and Find Them*
J. F. Dunnigan, William Morrow & Co, New York, 1992.

[EHRH97] *Cours de simulation*
ICA Ehrhard, cours ENSIETA 1997-98.

[FISH96] *Monte Carlo: Concepts, Algorithms, and Applications*
G. S. Fishman, Springer-Verlag, 1996.

[FOUR87] *Probabilités et statistiques*
J. Fourastié, J.-F. Laslier, Dunod 1987.

[GIRA98] Guide de développement pour les simulations (M&S, VV&A)
D. Girardot, Centre d'Analyse de Défense, 1998.

[HILL93] *Analyse orientée objets et modélisation par simulation*
David Hill, Addison-Wesley France, 1993.

[IEEE94] *IEEE Standard Glossary of Modeling and Simulation Terminology*
Institute for Electrical and Electronics Engineers, 1994.

[IGAR95] *Modèles et Simulations*
J.-L. Igarza, Cours de DESS à l'Université de Versaille-St Quentin, 1995.

[IST94] *The DIS Vision : A Map to the Future of Distributed Simulation,*
Institute for Simulation and Training, 1994.

[JACQ93] *Quelques aspects de la mécanique des fluides numérique : maillages, schémas et modèles,*
O.-P. Jacquotte, L. de Chanterac, A. Ormancey, Revue Scientifique et Technique de la Défense (93-3), 1993.

[JOLI95] *La simulation et ses techniques*
B. Jolivalt, Presses Universitaires de France, 1995.

[KELT99] *Simulation Modeling and Analysis*
D. W. Kelton, A. M. Law, McGraw-Hill Higher Education, 1999.

[KNUT68] *The Art of Computer Programming, vol.1 : Fundamental Algorithms*
D. Knuth, Addison Wesley, 1968.

[KNUT71] *The Art of Computer Programming, vol.2 : Seminumerical Algorithms*
D. Knuth, Addison Wesley, 1971.

[KUHL99] *Creating Computer Simulation Systems*
F. Kuhl, R. Weatherly, J. Dahmann, Prentice Hall, 1999.

[LIAR97] *Les wargames commerciaux américains, des années soixante à nos jours, entre histoire militaire et simulation, une contribution à l'étude de la décision (thèse de doctorat)*
J.-P. Liardet, Université Paul Valéry – Monpellier III, 1997.

[MOOR98] *A brief history of Aircraft Simulation*
K. Moore, *http://www.bleep.demon.co.uk.*

[NATO98] *NATO Modeling and Simulation Master Plan version 1.0*
NATO AC/323 (SGMS)D/2, 1998.

[NRC97] *Opportunities for Collaboration Between the Defense and Entertainment Research Communities*
Committee on Modeling and Simulation (National Research Council), National Academy Press, 1997.

[NRC98] *Modeling Human and Organizational Behavior. Application to Military Simulations*
Panel on Modeling Behavior and Command Decision Making (National Research Council), National Academy Press, 1998.

[OFTA97] *Application des techniques formelles au logiciel*
Observatoire Français des Techniques Avancées, 1997.

[OFTA00] *Architecture de logiciels et réutilisation de composants*
Observatoire Français des Techniques Avancées, 2000.

[OTA94] *Virtual Reality and Technologies for Combat Simulation,*
Office Of Technology Assessment, 1994.

[PERL90] *The Art of Wargaming*
P. Perla, Naval Institute Press, 1990.

[PIME93] *Virtual Reality: Through the New Looking Glass*
K. Pimentiel, K. Teixeira, McGraw Hill, 1995.

[PINA96] *Étude des standards d'interopérabilité des simulations*
J.-M. Pinard, mémoire DESS, Univ. Versailles St Quentin-en-Y., 1996.

[POOC92] *Discret Event Simulation*
U. W. Pooch, J. A. Wall, CRC Pr., 1992.

[PRIN98] *Puissance et limites des systèmes informatiques*
J. Prinz, Hermès 1998.

[QUEA86] *Éloge de la simulation*,
Philippe Quéau, Champ Vallon, 1986.

[RAYM97] *The Cathedral and the Bazaar*,
Eric S. Raymond, 1997.

[ROUS96] *Cours de géographie numérique*
Thierry Rousselin, ENSIETA 1996

[RTO99] *Computer Generated Forces Technology (RTO TR-11)*
Studies, Analayse and Simulation Panel, NATO Research & Technology Organisation, 1999.

[SABO86] *Éléments finis et CAO*
J.-C. Sabonnadière, J.-L. Coulomb, Hermès, 1986.

[SAUT01] *Documentation ESCADRE 5.0*
C. Sautereau, DGA/CAD, 2001, *http://escadre.cad.etca.fr:1815* .

[SCHN93] *The Encyclopedia of Virtual Environments*
B. Schneiderman (et ses étudiants), document HTML, Univ. of Maryland, 1993, *http://www.hitl.washington.edu/scivw/EVE/* .

[SEDG91] *Algorithmes en langage C*
R. Sedgewick, Intereditions, 1991.

[THAL99] *Advanced Virtual Reality Systems and TelePresence*
Daniel Thalmann, *http://ligwww.epfl.ch/~thalmann/* .

[ZEIG98] *The DEVS/HLA Distributed Simulation Environment And Its Support for Predictive Filtering*
P. Zeigler, G. Ball, H. Cho, J.S. Lee, H. Sarjoughian, Univ. of Arizona, 1998.

[ZEIG00] *Theory of Modeling and Simulation (2nd edition)*,
B. Zeigler, H. Praehofer, T.G. Kim, Academic Press, 2000.

14. QUELQUES RESSOURCES SUR INTERNET

Les sites WWW sont par essence très volatils, aussi je n'ai pas souhaité être exhaustif ici. Mais néanmoins, voici quelques points de départ pour un surf thématique sur la simulation.

14.1 Sites web

- SISO : http://siso.sc.ist.ucf.edu/ et http://www.sisostds.org
- HLA : http://hla.dmso.mil
- DMSO : http://www.dmso.mil
- Groupe ADIS : http://www.cert.fr/adis
- Simulation Optimization and Sensitivity Analysis Resources : http://ubmail.ubalt.edu/~harsham/ref/Refsim.htm

14.2 Newgroups

- Simulation générale : news:comp.simulation
- Simulation militaire : news:comp.simulation.military
- Simulateurs de vol : news:rec.aviation.simulators
- Simulation en aéronautique : news:sci.aeronautics.simulation
- Jeux de guerre ludiques : news:alt.games.wargames
- Jeux de guerre ludiques, francophone : news:fr.rec.jeux.wargames

15. INDEX

Cet index a pour objet de permettre une recherche d'un sujet par mot clé. En gras on trouvera l'endroit où le sujet est particulièrement développé, i.e. s'il est l'objet d'un paragraphe. ***Il ne tient pas compte des annexes*** *(et donc, notamment, du glossaire).*

Il a fallu faire des choix pour limiter la taille de l'index. Ainsi, les termes ESCADRE (acteurs, technologies...), trop spécifiques, ne s'y trouvent pas. Lorsqu'un terme anglo-saxon n'avait pas de traduction claire et couramment admise en français, il a été conservé.

Attention à tenir compte des différentes façons possibles d'écrire un mot clé, par exemple « domaine de validité » peut aussi être sous la forme « validité, domaine de ». De même, il est possible de trouver « High Level Architecture » ou HLA. Un choix est alors fait, avec éventuellement un renvoi d'une entrée vers une autre.

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Z