CHAPITRE 9

Conclusions

La maîtrise de l'interaction est un enjeu informatique au moins aussi important que la disponibilité de modèles et algorithmes efficaces. L'histoire récente du multimédia montre en effet que ce n'est qu'une fois combiné au sein de systèmes complexes qu'un nouveau modèle ou un algorithme rapide donne lieu à des applications largement diffusées et utilisées. Aussi, des logiciels extensibles ainsi que des systèmes de composants sont proposés aux chercheurs et développeurs comme le meilleur moyen de mettre en oeuvre, tester et diffuser très rapidement de nouvelles idées d'algorithmes ou de produits.

Nous verrons dans ce chapitre que les architectures développées dans l'industrie proposent un cadre de travail (*framework*) rigide dans lequel peuvent se greffer des modules ou composants (*component*s). Cependant, elles souffrent d'une limitation majeure : il est impossible d'implanter des interactions entre composants qui n'ont pas été prévues lors de la conception du cadre de travail. Une telle limitation pourrait être levée en faisant appel au modèle acteur-observateur présenté dans ce travail de thèse.

9.1 Résumé : une théorie des systèmes modulaires

Le modèle acteur-observateur propose une théorie des systèmes modulaires fondée sur une sémantique opérationnelle réflexive des systèmes de composants. Nous avons mis en oeuvre le modèle acteur-observateur au sein d'environnements existants et l'avons illustré au moyen d'exemples issus de la synthèse multimédia.

Fondements théoriques

Au chapitre 2, nous avons construit une abstraction d'un système réflexif de composants indépendante de tout langage ou de tout système existant :

- Un **composant** est défini au paragraphe 2.2.2 comme un ensemble de champs, c'est à dire de couples (i,e) formés d'un identificateur i et d'une expression e. Les expressions font appel à trois primitives sémantiques : la lecture d'un champ, l'écriture d'un champ et l'évaluation d'un champ appelé aussi **envoi de message**.
- Un **système de composants** est défini au paragraphe 2.2.3 en étendant la sémantique de l'évaluation d'un champ pour permettre à un composant d'envoyer un message à un autre composant.
- Un **miroir** est introduit du paragraphe 2.3.1 au paragraphe 2.3.3. Le miroir permet aux composants d'un système non seulement de se donner un représentation d'eux-mêmes (quels sont mes champs, quels sont ceux de mon voisin etc.) mais aussi de modifier la sémantique des primitives de lecture, écriture et envoi de message. C'est pourquoi le système de composants présenté est qualifié de **réflexif**.
- On montre aux paragraphes 2.4.1 et 2.4.2 comment utiliser un miroir pour rendre un système concurrent ou distribué. On donne alors au paragraphe 2.4.3 une définition des composants actifs : les **acteurs**.

Au chapitre 3, nous avons montré que l'interaction au sein d'un système réflexif de composants pouvait elle-même être représentée par un composant du système.

- Nous avons montré à la section 3.1 que les modes de structuration les plus courants dans les systèmes multimédia correspondaient à une **extension de la sémantique** présentée au chapitre 2 et à l'introduction de **graphes d'interaction**.
- Nous avons alors proposé à la section 3.2 d'utiliser la réflexion du miroir pour implanter les extensions sémantiques et les graphes d'interaction. Nous avons finalement proposé de réunir au sein d'un même composant, appelé **observateur**, les extensions sémantiques et les graphes qui caractérisent toute interaction.

Mise en oeuvre et applications

Les chapitres 4 à 5 illustrent une mise en oeuvre du modèle acteur-observateur au sein d'environnements existants. Un important travail d'implantation a permis de réunir au sein d'un même système :

- un **environnement à objets dynamique** (décrit en 4.2) mettant en oeuvre des composants actifs (section 5.2) et le modèle acteur-observateur (section 5.3);
- une **bibliothèque graphique 3D** (décrite en 4.3.2) adaptée sous forme de classes (section 5.1);
- une **bibliothèque de synthèse sonore**, de génération musicale et de gestion du temps (décrite en 4.3.1) adaptée sous forme d'acteurs et d'observateurs (paragraphe 7.1.3);
- un générateur d'**interface utilisateur** et des outils de manipulation directe d'interactions (section 5.4).

Les chapitres 6 et 7 analysent des algorithmes d'animation et de contrôle de la synthèse sous forme de système acteur-observateur. On a montré en particulier que la création d'observateurs permet de faire interagir ou d'hybrider plusieurs algorithmes d'animation ou de synthèse sonore déve-

loppés de façon indépendante. L'interaction d'interactions permet par exemple :

- de combiner animation dynamique et animation comportementale (section 6.2);
- de produire automatiquement la bande son d'un film d'animation (paragraphe 6.3.1);
- d'introduire de la mécanique dans un modèle de contrôle de la synthèse par règles (paragraphe 7.2.2).

Le chapitre 8 propose d'appliquer le modèle acteur-observateur à la programmation d'interfaces utilisateur et au génie des jeux vidéo.

9.2 Perspectives : les systèmes modulaires industriels

Presque tous les grands logiciels édités de nos jours sont conçus comme des chefs d'oeuvres monolithiques, virtuoses d'un nombre restreint de traitements de données mais incapables de s'ouvrir aux nouveaux algorithmes que pourrait intégrer un développeur indépendant. C'est le cas des logiciels de bureautique tels que ClarisWorks [Clar95].

Conscients des limitations inhérentes à un tel degré de fermeture, les éditeurs de grands logiciels ont proposé depuis quelques années la notion de module (*plug-in*) afin de permettre à un développeur indépendant d'accroître les capacités d'un logiciel. Par ailleurs, les fabricants de systèmes d'exploitation tentent actuellement d'imposer un nouveau mode de programmation et d'utilisation de l'informatique avec l'émergence des systèmes industriels de composants.

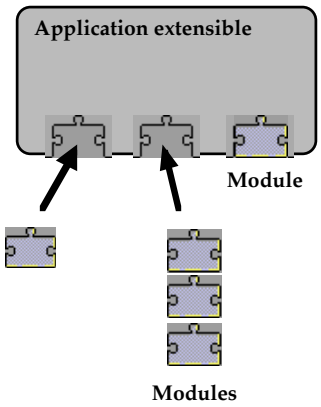
Application extensible

Une application extensible est un logiciel monolithique dans lequel une partie des fonctionnalités est gérée de façon générique par des modules liés dynamiquement à la demande de

l'utilisateur (voir figure 48 page 151). L'application communique à sens unique avec ses modules par envoi de messages (selon une sémantique proche de celle décrite page 23). Donnons deux exemples d'applications extensibles :

- **Photoshop 3.0** [Adob95]. Photoshop est un logiciel de retouche photographique et de traitement d'images. Son éditeur a publié une API (*Application Programming Interface*) décrivant comment écrire un module d'acquisition, d'importation, d'exportation ou de traitement d'images. De nombreux modules ont été développés, aussi bien par des chercheurs que par des développeurs professionnels, pour lire une image provenant d'une caméra, exporter une image dans un format de données rare, transformer une image en fonction d'un modèle géométrique 3D etc.
- **Navigator 2.0** [Nets95]. Navigator est un butineur, essentiellement monolithique, utilisé pour parcourir Internet. Son éditeur a publié une API (*Application Programming Interface*) décrivant comment écrire un module pour afficher au sein du butineur des données téléchargeables dans un format non reconnu en standard. Des modules ont été développés par des chercheurs et éditeurs indépendants pour afficher par exemple des séquences d'animation, écouter une émission radiodiffusée sur le réseau ou encore représenter le contenu d'un site Internet sous forme d'un modèle géométrique tridimensionnel.

FIGURE 48 Une application extensible utilise des modules pour accroître ses fonctionnalités



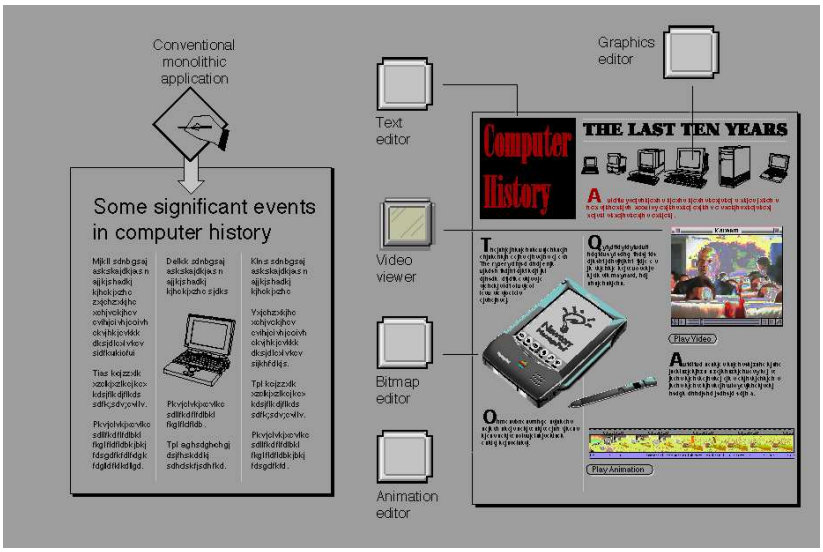
Systèmes de composants

Un système de composants utilise un modèle unifié de programmation à objets pour faire interagir des composants indépendants au sein d'un cadre prédéfini (voir figure 50 page 154). Les composants communiquent entre eux par envoi de messages (selon une sémantique proche de celle décrite page 23) et ont parfois accès à leurs champs respectifs. Donnons deux exemples de systèmes de composants :

- **OpenDoc** [Open94]. OpenDoc est le fruit d'une collaboration entre Apple et IBM visant à casser le monopole des grands éditeurs d'applications monolithiques[1]. OpenDoc propose un cadre de travail pour faire interagir au sein d'un même document des données actives gérées par des composants développés indépendamment les uns des autres (voir figure 49 page 153). Les composants d'OpenDoc utilisent le modèle unifié de programmation à objets SOM [DanfForm89]. La sémantique de SOM comprend à la fois des éléments similaires aux primitives d'un système de composants (voir table 3 page 16) et des caractéristiques de réflexion semblables à celles du miroir (voir table 12 page 30).

1. OpenDoc a été appliqué récemment au développement d'un logiciel de communication modulaire, Cyberdog, où chaque protocole de communication (transfert de fichier, courrier électronique, consultation de sites Internet etc.) est géré par un composant spécialisé. C'est pourquoi, contrairement à Navigator, seuls les composants utiles lors d'une communication sont utilisés.

FIGURE 49 Comparaison d'une application monolithique et d'un système de composants pour la gestion des documents composites (crédits Apple)



- **Java** [Java95]. Conçu initialement pour le contrôle des petits systèmes embarqués et des produits électroniques grand public, Java est devenu un moyen privilégié de diffusion de composants applicatifs sur Internet[1]. A l'instar de Smalltalk (voir paragraphe 4.2.1 page 67), Java est fondé sur un trio composé d'un langage à objets faiblement réflexif, d'une bibliothèque standard de classes s'interfaçant avec le système hôte et d'une machine virtuelle capable de transcrire rapidement un code binaire standardisé dans le langage machine d'un processeur hôte[2]. Si l'on tente de comparer Java à OpenDoc, la machine virtuelle

1. Java a pu conquérir très rapidement une large communauté de développeurs confortés dans leur choix par sa gratuité sur les trois systèmes d'exploitation les plus courants et par le développement en cours de cadres de travail multimédia s'interfaçant avec Java.

Java a été choisi par le groupe de standardisation du format VRML 2.0 (*Virtual Reality Modeling Language*) pour intégrer du comportement autonome distribué au format de réalité virtuelle VRML.

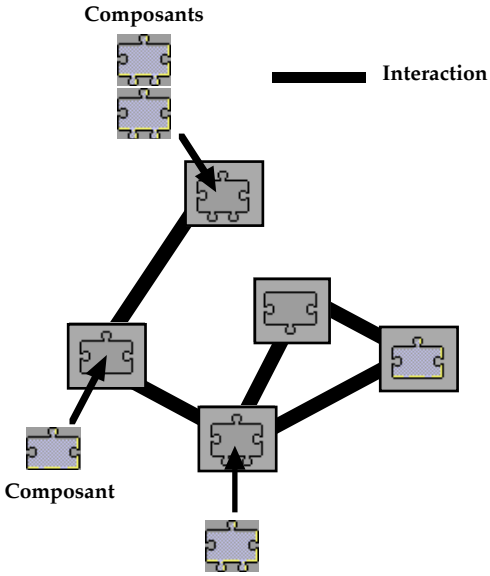
Par ailleurs, Apple développe un jeu de classes transformant Java en un environnement de production multimédia comparable à ScriptX [Appl94].

2. L'apparition prochaine de processeurs optimisés pour l'exécution de code binaire Java standardisé pourrait réduire la portée de cet argument ou annoncer un renouveau du développement de processeurs spécialisés permettant plus d'efficacité que les technologies RISC ou VLIS.

Java correspond à SOM et la bibliothèque standard de classes correspond au cadre de travail défini par OpenDoc. Les primitives de la machine virtuelle de Java ressemblent à celles d'un système de composants non réflexif (voir table 3 page 16).

FIGURE 50

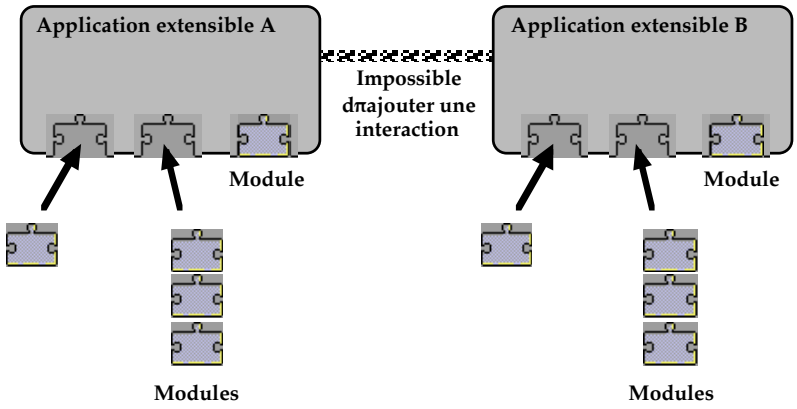
Un système de composants fait interagir des composants indépendants au sein d'un cadre prédéfini



Limitations

Les limitations des applications extensibles ont été suggérées en introduction : l'ajout de modules ne permet pas, en général, de créer des interactions entre applications (voir figure 51 page 155).

FIGURE 51 Deux applications extensibles ne peuvent pas interagir



Les limitations des systèmes de composants sont moins évidents car elles tiennent aux choix qui ont prévalu lors de leur implantation. Le cadre de travail d'OpenDoc ou de Java est rigide :

- Les interactions mises en oeuvre par OpenDoc sont décrites implicitement dans un ouvrage de référence. La figure 52 montre par exemple comment sont gérées les hiérarchies contenant-contenu dans OpenDoc : on distingue plusieurs graphes représentant implicitement autant d'interactions entre composants. Or, il n'existe pas d'API pour modifier ou étendre ces interactions, car elles sont figées par les interfaces immuables des classes standard de composants. L'interaction dans OpenDoc est gérée par une implantation rigide similaires aux extensions sémantiques décrites aux paragraphes 3.1.1 à 3.1.4. C'est pourquoi, il est pratiquement impossible d'ajouter de nouvelles interactions dans un document OpenDoc[1].
- Les interactions entre composants dans Java sont aussi décrites implicitement au sein d'interfaces statiques[2]. Elles sont directement mises en oeuvre par l'envoi de messages de la machine virtuelle Java, ce qui induit une architecture

1. Néanmoins, un mécanisme de message sémantique restreint et de script permettent de développer des interactions simples ou peu performantes.

2. Les interfaces de Java défissent un système de types.

très rigide en raison du manque de réflexivité du langage. C'est pourquoi, il est difficile de faire interagir deux composants applicatifs Java développés séparément et dont on ne disposerait pas du code source.[1].

FIGURE 52 Exemple d'interactions implicites entre composants (crédits Apple)

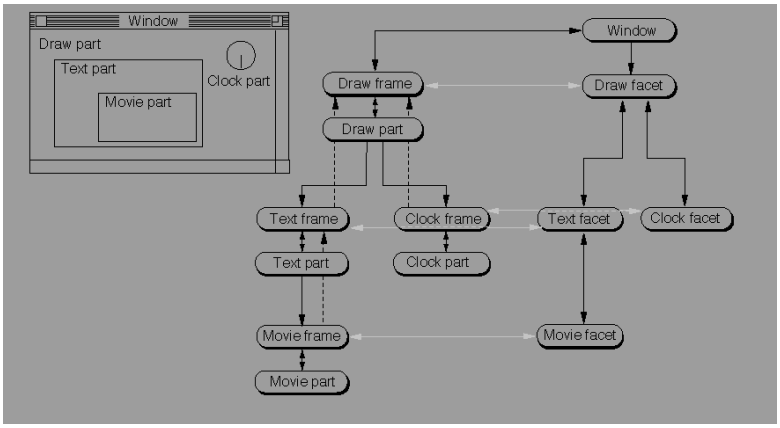
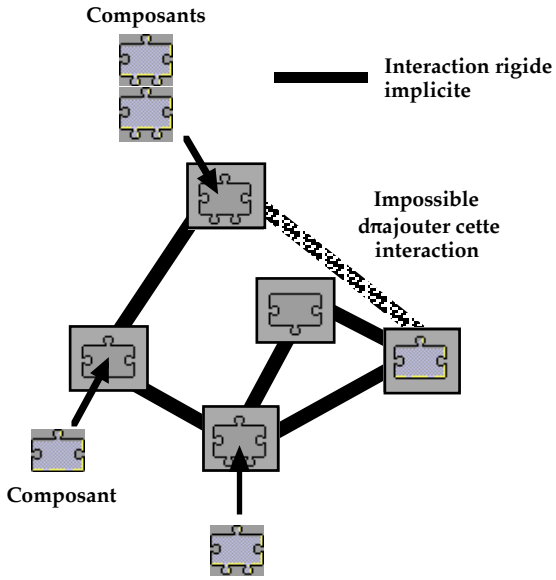


FIGURE 53 Il n'est pas possible d'ajouter de nouvelles interactions dans une architecture rigide



1. Sun Microsystems, concepteur de Java, reconnaît cette limitation dans un rapport (*Java Beans White Paper*) consacré aux applications de Java à la programmation par composants. C'est pourquoi Sun envisage développer d'intégrer à Java un mécanisme similaire aux messages sémantiques d'OpenDoc.

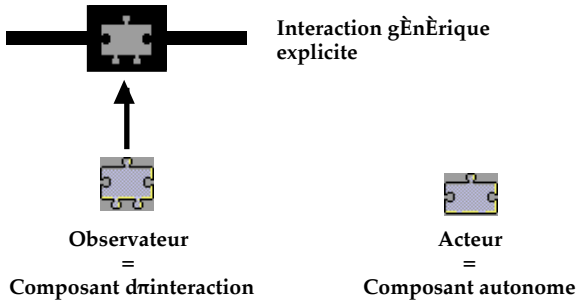
9.3 Acteur-observateur : une formalisation réflexive de l'interaction

Le modèle acteur-observateur lève les limitations des systèmes industriels de composants. Nous avons montré en effet aux chapitres 2 et 3 qu'un cadre de travail réflexif, associé aux propriétés de réflexion d'un système de composants sous-jacent, permet la combinaison sans limites de structures d'interactions quelconques au sein d'un même système. En pratique, cela signifie qu'en construisant un système de composants dans lequel les interactions sont explicitement représentées et mises en oeuvre par d'autres composants, on se donne les moyens de bâtir des architectures d'interaction souples et extensibles.

Observateur : un composant d'interaction

En analysant la figure 50, on constate en effet que le seul élément de modularité des systèmes industriels de composants consiste à pouvoir remplacer les composants placés sur l'un des noeuds du graphe d'interaction. On ne peut pas modifier le type d'interaction associé aux arêtes du graphe ou étendre le graphe pour ajouter de nouvelles interactions. L'idée des observateurs suit alors tout naturellement : il suffit d'utiliser des composants pour construire les arêtes du graphe d'interaction (voir figure 54 page 157).

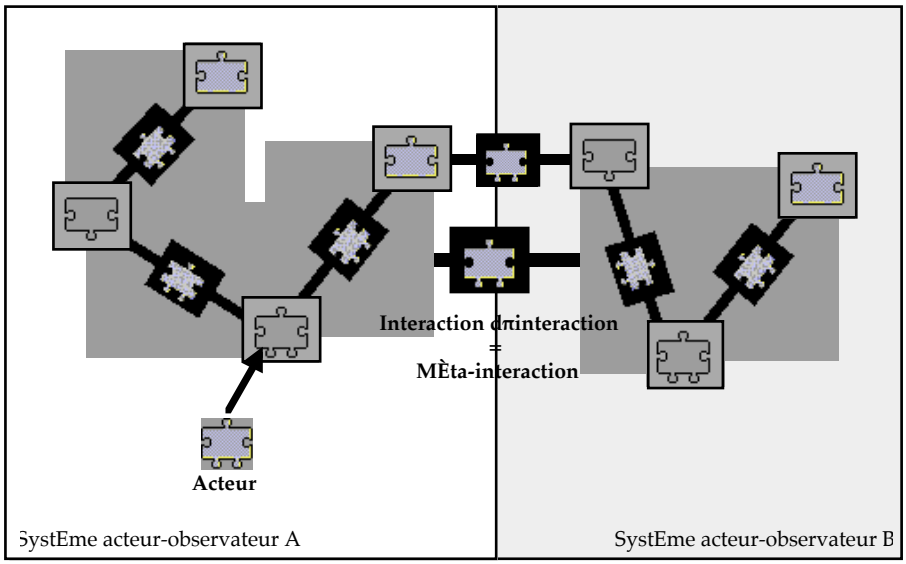
FIGURE 54 Observateur : un composant d'interaction



Une architecture extensible pour une meilleure réutilisabilité du code

On peut ainsi étendre les modes d'interaction entre composants d'un système de la même façon que l'on peut étendre les fonctionnalités d'un logiciel extensible par ajout de modules. Par ailleurs, lorsque l'on souhaite faire interagir deux systèmes hétérogènes à acteurs-observateurs, on peut non seulement créer des interactions entre les deux systèmes mais aussi mettre en oeuvre des interactions d'interactions susceptibles de corriger les incompatibilités qui pourraient naître (voir figure 55 page 158).

FIGURE 55 Interactions d'interactions



Reprenons l'exemple du mélange de la dynamique et de contraintes sur des systèmes articulés développé au chapitre 6. Dans un style de programmation traditionnel, l'écriture d'un tel système consiste à formuler un algorithme unique et intégré de résolution simultanée des équations de la dynamique et des contraintes mécaniques puis à l'implanter complètement sous forme d'un logiciel monolithique. Il est en effet difficile d'envisager de faire appel à des bibliothèques indépendantes de résolution de la dynamique d'une part, des

contraintes mécaniques d'autre part, car leur utilisation simultanée tendrait à faire diverger le système.

En revanche, dans un système d'acteurs-observateurs, l'intégration de la dynamique et des contraintes est obtenue simplement par création et ajout d'un observateur capable de corriger périodiquement les paramètres de vitesse de la dynamique en fonction des changements de position imposés par la résolution de contraintes mécaniques. Bien entendu, dynamique et contraintes mécaniques sont représentées par des observateurs qui ont pu être développés de façon indépendante dans deux équipes différentes.

Cet exemple illustre le bon degré de réutilisabilité d'un code écrit sous forme d'acteurs et d'observateurs.

Un cadre de travail pour la synthèse multimédia

La souplesse et l'extensibilité du modèle acteur-observateur en font un cadre de travail adapté à la synthèse multimédia. En effet, le modèle acteur-observateur réunit deux caractéristiques jusqu'à présent considérées comme exclusives :

- Il permet, avec des outils de haut niveau (MetaFinder, MetaBrowser) de formuler, visualiser et éditer une structure d'interaction générique.
- Il ne limite pas le chercheur, le créateur ou le producteur à des structures interactives figées.

La mise en oeuvre du modèle acteur-observateur au sein d'un système suffisamment abouti et performant pourrait donc permettre aux producteurs d'ouvrages électroniques ou de jeux interactifs d'accroître considérablement le champ potentiel de leur créativité, limité aujourd'hui par la lourdeur des investissements à consentir pour l'implantation au moyen d'environnements de programmation traditionnels d'une structure d'interaction multimédia quelque peu originale. Par ailleurs, le modèle acteur-observateur propose un définition abstraite et universelle de la notion d'interaction, souvent utili-

sée dans le discours sur le multimédia mais rarement définie avec précision.

Un cadre de travail pour la conception de grand systèmes

Le fait d'avoir développé aux chapitres 2 et 3 une sémantique opérationnelle du modèle acteur-observateur indépendante de tout langage permet d'envisager son application et sa mise en oeuvre au sein de n'importe quel système de composants suffisamment réflexif. Le modèle acteur-observateur pourrait ainsi devenir un outil d'analyse des grands systèmes susceptible de mieux prendre en compte leurs évolutions imprévues. L'application à la synthèse multimédia a montré comment séparer un système en acteurs et observateurs.

- On commence par casser le système en briques élémentaires les plus indépendantes possibles (point orienté, base de temps, forme d'onde formantique etc.) : ce sont les acteurs.
- On tente ensuite d'analyser les interactions survenant entre acteurs en essayant de les ramener à une forme la plus générique possible (hiérarchie de parties, délégation etc.).
- On implante alors les interactions au sein d'observateurs en s'interdisant de modifier directement le code des acteurs élémentaires déjà programmés (hiérarchie temporelle, dynamique, voyelle etc.).

Développé sur ce principe, un système pourrait être plus aisé à maintenir qu'un système à classes et objets. En effet, tout le code de mise en interaction des composants est concentré aux sein d'observateurs au lieu d'être dispersé dans l'implantation de classes. Par ailleurs, la mise en interaction d'interactions permet de corriger les incompatibilités qui pourraient certainement naître lorsque, à la suite d'une longue évolution du système et de nombreux ajouts d'acteurs-observateurs, deux observateurs entrent en conflit.

9.4 De l'interaction multimédia aux vies réticulées

Le livre et le jeu électronique ont certainement été à l'origine de l'essor du multimédia en 1993. Depuis1995, c'est l'essor d'Internet qui semble désormais stimuler le développement du multimédia. C'est donc sur le terrain des réseaux mondiaux que doivent être envisagées des applications éventuelles du modèle acteur-observateur. Contrairement au CD-ROM qui permet une diffusion espacée dans le temps de grandes quantités d'information, Internet propose aujourd'hui une diffusion répétée et rapprochée de petites quantités d'information. Ce mode de diffusion est tout particulièrement adapté à la diffusion de composants logiciels en constante évolution venant s'adapter à un cadre de travail relativement stable jouant le rôle anciennement dévolu au système d'exploitation.

C'est pourquoi, la vie artificielle[1] nous semble pouvoir devenir un domaine d'application de l'interaction entre composants à l'échelle de la planète. La mise en oeuvre d'algorithmes de vie artificielle butte en effet sur deux écueils : un besoin de puissance de calcul immense et un choix judicieux des stimulus. Or, en considérant Internet comme une machine multiprocesseur mondiale, les utilisateurs d'Internet comme autant de sources de stimulus et le modèle acteur-observateur comme un moyen de gérer des interactions concurrentes et distribuées à grande échelle, on pourrait envisager la synthèse de formes de vie artificielle dépassant le cadre limité des expériences menées aujourd'hui. Une étude approfondie pourrait faire l'objet de nouveaux travaux.

1. La vie artificielle a pour objet l'étude d'algorithmes concurrents de simulation de systèmes vivants et l'émergence de comportements caractéristiques. La vie artificielle a déjà permis de synthétiser des composants électroniques, des nageurs ou des chorales d'animaux sans faire appel à aucune analyse mathématique.