CHAPITRE 4

Choix des outils

Ce chapitre justifie puis présente en détail les choix technologiques adoptés pour une implantation du modèle acteur-observateur au sein d'environnements existants :

- **SmalltalkAgents** est une implantation moderne du langage Smalltalk avec des extensions réflexives et concurrentes permettant l'implantation rapide du modèle acteur-observateur.
- **MacOS** est un système d'exploitation doté de caractéristiques multimédias grâce à **Quicktime** et d'une gestion des interfaces en trois dimensions grâce à **Quickdraw3D**
- **Actalk** est une implantation de modèles d'objets concurrents dans Smalltalk.

4.1 Critères et choix d'un environnement

Le choix de l'environnement d'implantation du modèle Acteur-observateur visait, en 1993, à optimiser les critères suivants :

- implanter en un minimum de temps le modèle acteur-observateur;
- implanter en un minimum de temps des composants multimédias (musique, graphisme 3D);

- bénéficier d'un environnement graphique, dynamique et permettant une exécution en « temps-réel »[1].
- dépenser le moins d'argent possible[2].

En 1993, aucun environnement informatique ne maximisait simultanément ces quatre critères. Mon choix a donc consisté à comparer les systèmes d'exploitation, les langages de programmation et les machines disponibles alors pour parvenir à une combinaison maximisant les critères que je m'étais fixés.

4.1.1 Choix du système d'exploitation

Pour conduire en 1993 un projet de recherche faisant appel à du graphisme 3D, on avait le choix entre :

- utiliser une station de travail Unix dotée d'une bibliothèque graphique fiable et performante (OpenGL, PEX etc.);
- utiliser un micro-ordinateur doté d'une bibliothèque graphique peu fiable ou peu performante.

Au même moment, pour conduire un projet de recherche en informatique musicale, on avait le choix entre :

- utiliser une station de travail Unix, capable de traiter très rapidement un signal sonore mais incapable, à l'exception des stations Next, de gérer un système musical en temps réel;
- utiliser un Macintosh, doté d'une bonne gestion d'événements en temps réel, de très nombreux logiciels de composition musicale et de cartes accélératrices permettant un traitement du son très rapide.

1. Ce critère provient de contraintes et habitudes de travail de la production multimédia.

2. Cette contrainte est liée à mon voyage au Japon où j'ai effectué mon service national. L'acquisition d'une machine était rendue nécessaire dans la mesure où mon poste en Ambassade m'interdisait de faire appel aux ressources informatiques universitaires locales ou d'utiliser le matériel disponible sur mon lieu de travail.

Par ailleurs, plus de 90% de la production multimédia était réalisée en 1993 sur Macintosh : conception de CD-ROM, montage vidéo virtuel, retouche photographique, illustration, mise en page etc. En effet, le système MacOS gère en standard la musique, le son, l'image et la vidéo; des environnements de développement nombreux et parfois innovants étaient déjà disponibles.

4.1.2 Choix du langage

Pour implanter le modèle acteur-observateur, on peut envisager de :

- créer un langage puis développer, en C/C++, un interpréteur ou un compilateur;
- modifier un langage non-réflexif, à condition d'avoir accès au code source;
- adapter un langage réflexif.

Après avoir demandé conseil à des chercheurs expérimentés, j'ai éliminé les deux premières approches parce qu'elles me semblaient trop éloignées de mon objectif de thèse, c'est-à-dire de l'application du modèle acteur-observateur à la synthèse multimédia. Il valait mieux travailler dans un langage le plus standard possible afin de pouvoir bénéficier de l'expérience acquise par la communauté scientifique et d'éviter de pâtir des défauts d'un système en développement.

En 1993, les deux langages réflexifs les plus courrants étaient Common Lisp[1] [Stee84] et Smalltalk [Gold84]. Je me suis orienté vers Smalltalk en raison des 10 années d'expérience acquises par le projet Actalk [Brio89] en matière de programmation à objets concurrents.

1. Common Lisp inclut un langage à objets réflexif : Common Lisp Object System (CLOS).

4.1.3 Choix du matériel

En 1993, à performances comparables, une station de travail était deux fois plus chère qu'un micro-ordinateur et les logiciels commerciaux pour Unix, lorsqu'ils existaient, étaient souvent plus de deux fois plus chers que ceux pour MacOS ou Windows. Par ailleurs, travailler sous Unix permettait d'avoir accès gratuitement au code source de nombreux logiciels expérimentaux développés par la communauté scientifique. Des contraintes de prix ne permettaient donc d'envisager en 1993 que deux solutions pour conduire un travail de recherche :

- Un micro-ordinateur sous système Windows ou MacOS et des environnements commerciaux de production ou de développement à la fois sophistiqués et bon marché;
- Un micro-ordinateur sous système Unix ou Linux utilisant les nombreux logiciels développés par la communauté scientifique, gratuits, performants mais peu fiables ou peu sophistiqués.

4.1.4 Synthèse

J'ai choisi en 1993 le système MacOS car c'était probablement le meilleur système d'exploitation pour conduire un projet de recherche en multimédia, si l'on excepte une faiblesse en graphisme 3D. En fonction d'informations provenant d'amis ingénieurs travaillant chez Apple en Californie, j'ai parié sur la disponibilité rapide d'une bibliothèque graphique 3D performante et fiable. J'ai alors choisi le produit commercial SmalltalkAgents car c'était le seul environnement Smalltalk sous MacOS doté d'une bonne intégration avec le système d'exploitation, d'une gestion de tâches préemptive et d'un environnement graphique de programmation.

4.2 SmalltalkAgents : un environnement dynamique

SmalltalkAgents est une implantation récente du langage à objets et de l'environnement dynamique Smalltalk. Sa principale caractéristique est une très forte intégration avec le système d'exploitation hôte, ce qui lui permet d'exploiter au mieux ses ressources [Quas93].

4.2.1 Le langage Smalltalk

Smalltalk est le langage à objets dont dérivent la plupart des systèmes « orientés objets » modernes. Conçu dans les années 70 par Alan Kay au Palo Alto Research Centre de Xerox(PARC), il a trouvé aux Etats-Unis une très large audience auprès de l'industrie et fait partie de l'offre standard d'IBM pour le développement multiplateforme d'applications client-serveur.

Le deuxième langage à objets

Smalltalk est historiquement le deuxième langage à objets, après SIMULA, et a introduit le paradigme anthropomorphique de l'envoi de message, décrit au chapitre 2. La sémantique de Smalltalk est celle d'un système de composants à évaluation séquentielle avec structures d'héritage et d'instanciation (voir chapitre 3). Cependant, contrairement à un système fondé sur le modèle acteur-observateur, les structures d'héritage et d'instanciation sont implicites dans la mesure où elle ne peuvent être manipulées directement en tant que telles mais évoluent comme effet de bord de l'envoi de message aux composants qu'elles structurent.

Lorsque l'on compare Smalltalk à d'autres langages à objets [Paep93], on met souvent en avant la pauvreté du modèle d'envoi de message, comparé à celui de fonction générique [Stee84] mais on loue aussi sa simplicité qui a facilité la réalisation d'un environnement graphique et interactif particulièrement agréable et encore très en avance sur ses concurrents[1].

Le langage graphique du Macintosh

Smalltalk est en effet à l'origine des interfaces utilisateur à fenêtre et souris. Dès sa première implantation au PARC, la modélisation aisée d'objets graphiques et la prise en compte d'un nouveau périphérique d'entrée, la souris, ont permis de réaliser un environnement de développement graphique et interactif avec fenêtres, *pane* et menus déroulants. C'est de cet environnement qu'est né , à la suite d'une visite au PARC de Steve Jobs alors président d'Apple, l'ordinateur Macintosh grâce auquel les interfaces graphiques sont devenues aujourd'hui le standard de la bureautique moderne.

Principales caractéristiques

Les autres caractéristiques essentielles du langage sont :

- **Un modèle de classe implicite**. Dans un souci de simplicité apparente, le modèle de programmation standard de Smalltalk est un modèle de classe implicite. En effet, bien qu'il soit possible de procéder autrement [CoinBrio89], l'environnement Smalltalk associe automatiquement à chaque classe une métaclasse, ou classe de classe, de manière bijective.
- **Typage dynamique**. Les variables Smalltalk ne sont pas typées. En revanche, les composants sur lesquels elles pointent sont typés par leur classe.
- **Monosélection**. La sélection d'une méthode dépend uniquement de la classe du composant auquel un message est envoyé mais aucunement du type de ses arguments.
- **Réflexion**. La classe *Object* de Smalltalk et la structure de métaclasse qui lui est associée offrent suffisamment de réflexion pour implanter un modèle acteur-observateur ou un système de composants distribués.

1. Ainsi, Gregor Kikzalès, principal inventeur du meta-object protocol de *Common List Object System*, écrivait début 1995 dans les *news*, c'est-à-dire avant l'annonce par Sun de Java, que Smalltalk est la seule alternative à C++ à la fois capable de proposer un modèle de programmation plus abouti et crédible d'un point vue industriel.

- **Gestion de la mémoire**. Il n'est pas nécessaire de gérer la suppression des composants inutiles. Un glaneur de cellules (gc) se charge de supprimer tous les composants qui ont cessé d'être référencés.

4.2.2 Environnement dynamique intégré

Une caractéristique essentielle de Smalltalk et de son environnement est la dynamicité (voir chapitre 2).

Compilation incrémentale

La programmation en Smalltalk est dite interactive dans la mesure où un mécanisme de compilation incrémentale permet de programmer et de compiler le code d'un programme en cours d'exécution. A l'instar d'un cuisinier préparant une sauce par ajouts successifs d'ingrédients et d'épices, la programmation en Smalltalk consiste à ajouter des objets et des méthodes à un environnement évoluant sans-cesse sous l'effet du gc ou de tâches exécutées simultanément.

Programmation persistante

Les modifications apportées à l'environnement, par ajout de composants et de méthodes, sont conservées en fin de session dans un fichier appelé image qui, chargé lors d'une session ultérieure, permet de retrouver un environnement dans l'état où on l'avait laissé; on parle alors de persistance des composants du système. L'utilisation de fichier image rend caduque la programmation fondée sur l'utilisation de multiples fichiers textuels.

Inspecteurs

Toute classe Smalltalk comporte une méthode appelée *inspect* permettant la visualisation et l'édition interactive d'une instance de cette classe par l'ouverture d'une fenêtre appelée inspecteur.

C'est ainsi que l'inspecteur de métaclasse en Smalltalk est tout simplement un éditeur de classe permettant de spécifier les méthodes et les champs d'une classe. Dans un même esprit, l'inspecteur d'un dictionnaire permet de visualiser l'ensemble des clés et valeurs du dictionnaire, de les éditer, d'en ajouter ou d'en modifier.

Débogueur

Enfin, l'environnement Smalltalk intègre un debugger graphique et interactif dont l'implantation repose en grande partie, mais pas complètement, sur les caractéristiques réflexives du langage.

4.2.3 Des extensions innovantes

L'implantation moderne SmalltalkAgents de Smalltalk est fondée sur le standard Smalltalk 80 et apporte quelques innovations majeures qui en font un des meilleurs Smalltalk disponibles aujourd'hui.

Multitâche préemptif

SmalltalkAgents définit une classe *Thread* pour implanter un mécanisme de tâches préemptives avec priorités. Chaque tâche est attachée à une clôture et un contexte initial (dans un sens plus large que celui défini au chapitre 2, car il fait aussi référence à des variables). Chaque tâche peut être contrôlée et déboguée séparément avec la possibilité de définir des points d'arrêt communs à toutes les tâches ou propres à l'une d'entre elles. Aussi, il n'est pas rare de voir apparaître plusieurs débogueurs à l'écran correspondant à des erreurs survenues sur des processus concurrents.

Interface système

SmalltalkAgents innove en matière d'intégration avec le système hôte sur trois points :

- La définition de **méthodes externes** permet d'appeler des routines du système hôte directement au sein d'expressions Smalltalk.
- Le modèle de **gestion de la mémoire** permet de séparer les champs d'un composant en une partie stockée dans la mémoire allouée et gérée par le gc et une partie stockée dans la mémoire allouée et gérée par le système hôte.
- La classe *Callback* permet de simuler des **pointeurs de fonction** appelés par les routines du système hôte et de les traduire en envois de messages capables d'interrompre une tâche en cours d'exécution.

Grâce à cette approche, il est concevable de manipuler des structures (au sens du langage C) comme des objets Smalltalk et de les passer en paramètre à des routines du système ou à des bibliothèques partagées, ces appels pouvant en retour déclencher l'évaluation d'une expression Smalltalk.

Messages sémantiques

Le modèle de message de SmalltalkAgents étend celui de Smalltalk par l'ajout de messages sémantiques. Lorsqu'une instance de classe héritant de la classe *Component* reçoit un message sémantique, celui-ci est redirigé automatiquement vers un ensemble d'objets spécifiés dans un dictionnaire, appelé dictionnaire sémantique, associé à l'instance. Les messages sémantiques servent de fondement à la réalisation d'applications multifenêtre en offrant une gestion simple et efficace de contraintes de contrôle.

Méthodes spécifiques d'instance

SmalltalkAgents étend le modèle à objets de Smalltalk 80 en autorisant la création de méthodes spécifiques à une instance. Cette caractéristique, combinée aux messages sémantiques, permet d'associer au sein d'un même environnement les avantages des langages à objets et des langages de prototypes.

Réflexion étendue

SmalltalkAgents étend les caractétistiques réflexives de Smalltalk 80 en permettant d'intercepter les envois de message ainsi que l'accès aux champs des objets. Ces extensions sont particulièrement utiles et pratiques lorsque l'on souhaite modifier le modèle d'objet du langage ou implanter le calcul distribué.

Hiérarchie d'espaces de noms

La notion de variable globale est étendue par un mécanisme d'espaces de noms imbriqués. L'étendue lexicale d'une variable à partir d'un espace de nom donné correspond à tous les espaces de noms que l'on peut atteindre en parcourant le sous-arbre d'imbrication.

Cette approche permet de restreindre la définition de méthodes ou de classes à une partie du système. Elle s'avère particulièrement utile lorsque l'on souhaite modifier une classe système et ne rendre cette modification accessible qu'à une partie de l'environnement. Elle prouve aussi son utilité en matière de protection mutuelle d'espaces d'exécution pour éviter la propagation accidentelle de virus lors d'un accès réseau.

Archivage de paquets d'objets

La notion de fichier image est étendue par la notion de paquet d'objets ou PIPO. Elle revient à n'archiver que l'ensemble des objets dépendant d'un objet donné et appartenant au même espace de noms. Elle permet l'échange de données actives ou de micro-applications par le réseau.

Composants d'interface utilisateur

SmalltalkAgents propose un outil de construction d'interface utilisateur par manipulation directe partielle qui abandonne le modèle MVC pour le remplacer par la notion de *Component* associé à une gestion de messages sémantiques.

Stratification de l'offre en fonction des besoins finaux

SmalltalkAgents est proposé en deux versions :

- **VisualAgents**, un environnement d'écriture et d'adaptation d'applications à partir de composants prédéfinis.
- **SmalltalkAgents Pro**, un environnement de développement de composants et de *frameworks*.

La programmation en SmalltalkAgents est décomposée en strates de niveau d'utilisation.

TABLE 24 Niveau d'utilisation d'un environnement de programmation SmalltalkAgents

Rôle	Ce que l'on construit	Ce que l'on a besoin de savoir
Auteur	Application spécialisée ou formulaires d'entrée	Dessiner une application à la souris Connecter des composants entre eux
Constructeur	Application et composants hybrides	Méthodes de composition de composants par câblage graphique et écriture de routines[a]
Spécialisateur	Nouveaux *frameworks* et applications	L'ensemble de l'architecture de SmalltalkAgents
Architecte	Nouveaux *frameworks* absolument pas pris en compte dans la bibliothèque de classe	L'ensemble de l'architecture de SmalltalkAgents et le système hôte

a. On notera qu'un observateur s'apparente à un agent de câblage automatisé.

4.3 MacOS : une plate-forme multimédia

MacOS est le système d'exploitation le plus avancé en termes d'intégration multimédia grâce à deux gestionnaires : Quicktime et Quickdraw 3D. Il est aussi le système d'exploitation d'une console de jeu et d'accès à *Internet*.

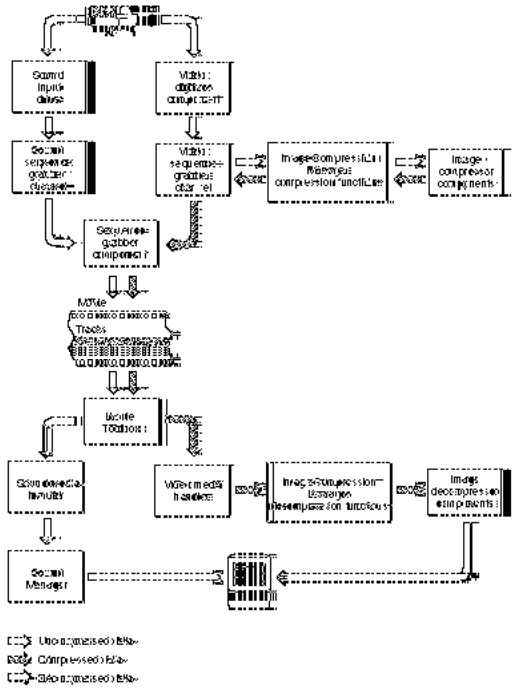
4.3.1 Quicktime : une gestion générique du temps

Quicktime est un gestionnaire multimédia dont l'objectif est de gérer la compression, l'acquisition et la restitution en temps réel de données multimédia [Appl93].

Quicktime est fondé sur une architecture de composants reliés selon une structure d'interaction statique (voir figure 26 page 75). Les principaux composants sont chargés de :

- **Entrée-sortie**. L'acquisition de données sonores ou visuelles ainsi que leur rendu final est géré de façon générique par des composants d'entrée ou de sortie en fonction de l'équipement disponible.
- **Compression, décompression et restitution**. Quicktime propose en standard la compression-décompression d'images (format JPEG), d'animations (format Cinepak, MPEG…), de sons (format μLaw, IMA, MIDI…). Lors de la restitution d'une séquence multimédia, des composants sont utilisés pour lire les données de chaque piste et les restituer sous forme de son ou d'image.
- **Gestion du temps**. Quicktime propose plusieurs méthodes de mesure du temps au moyen de composants (horloge interne, signal SMPTE…) afin d'assurer une synchronisation avec un équipement externe ou sur un réseau. La notion de base de temps permet de gérer simultanément plusieurs unités de mesure du temps et de synchroniser le déroulement de tâches concurrentes au sein d'une machine. Les bases de temps sont organisées selon une hiérarchie de relations maître-esclave.

FIGURE 26 Représentation partielle de la structure d'interaction des composants dans Quicktime.



Gestion de séquences multimédia génériques

Le format de séquence générique Quicktime permet de spécifier l'ordonnancement de données temporelles génériques et leur restitution grâce à des composants adéquats.

Multiplateforme

Quicktime est disponible sous MacOS, Windows et IRIX. C'est le standard *de facto* de la production multimédia non interactive. De nombreuses extensions permettent d'améliorer l'interaction avec l'utilisateur (Quicktime VR, Quicktime Interactive) et d'ajouter de nouveaux types de données (*sprites*, animation 3D).

4.3.2 Quickdraw 3D : une bibliothèque 3D portable

Quickdraw 3D est selon certain spécialistes la bibliothèque graphique 3D la plus complète et la plus avancée disponible

aujourd'hui. Elle reprend de nombreuses caractéristiques d'*OpenGL* [Open93] et *Open Inventor* [Wern94] en les étendant [Appl95].

Gestion de l'affichage

Quickdraw 3D est une bibliothèque graphique orientée objets dans la mesure où l'affichage d'un objet graphique passe par une étape de création d'un objet suivie de l'appel d'une fonction d'affichage immédiat. A cette fin Quickdraw 3D gère un grand nombre de classes graphiques comme :

- **Camera**. Définition d'un plan et d'un mode de projection d'une scène.
- **View**. Définition d'un contexte d'affichage complet (mode de rendu, fenêtre de rendu, scène, caméra…)
- **Nurbs**. Courbes et surfaces paramétrées rationnelles.
- **Mesh**. Gestion de polyèdres topologiques.
- **Attributes**. Gestion des paramètres de rendu locaux ou globaux.
- **Picking**. Gestion des périphériques d'entrée et de la sélection des objets d'une scène.
- etc.

Métaprogrammable et extensible

De nouvelles classes peuvent être ajoutées à Quickdraw 3D pour étendre les attributs de rendu, les algorithmes de rendu, gérer un accélérateur graphique etc. Un métaprotocole est fourni pour manipuler ou étendre le jeu de classes standard.

Gestion de base de données

Quickdraw 3D définit une gestion de base de données graphiques ainsi qu'un format de fichier générique permettant le copier-coller ou l'échange de modèles, y compris lorsque ceux-ci font appel à des classes étendant les fonctionnalités de Quickdraw 3D.

Multiplateforme

Quickdraw 3D est conçu pour être facilement portable. Il est disponible sous MacOS aujourd'hui et en partie sous Windows. Le format de fichier de Quickdraw 3D est déjà pris en compte par un grand nombre de constructeurs de logiciel en raison de ses qualités intrinsèques de généricité et de portabilité.

4.3.3 Pippin : un lecteur multimédia générique

Pippin est une console de jeu bon marché ayant MacOS pour système d'exploitation.

Un matériel puissant, bon marché et extensible

Pippin coûte moins de 500$ et comprend entre autres :

- Un processeur RISC 32 bits d'une puissance d'environ 100 MSpecFp.
- Un lecteur de CD-ROM quadruple vitesse.
- Une gestion du son en 16 bits.
- Une gestion d'image double buffer en 16 bits.

Pippin peut être étendue par l'ajout de cartes PCI afin d'assurer une meilleure gestion de la compression d'images, du rendu en trois dimensions ou de l'accès réseau. Le développement d'applications sur CD-ROM fait appel à des stations de travail MacOS et des environnements graphiques de développement particulièrement peu onéreux.

Un vecteur de travaux de recherche

Pippin constitue un équipement suffisamment simple et bon marché pour constituer un vecteur de démonstration, d'expérimentation publique ou de finalisation de travaux de recherche intégrant de la visualisation de données ou du multimédia. En effet, un travail de recherche développé sous MacOS peut être adapté très simplement aux caractéristiques de Pippin. Compte tenu de son faible coût, d'une interface utilisateur simplifiée et d'un encombrement réduit,

Pippin permet d'aboutir à un démonstrateur plus aisément qu'avec une station de travail ou un micro-ordinateur.

4.4 Actalk : des acteurs dans un système à objets

Actalk est une bibliothèque de classes permettant de créer et manipuler des composants actifs, ou acteurs, en Smalltalk.

4.4.1 La plateforme Actalk

La programmation concurrente a connu un développement remarquable dans les années 80 grâce à l'effort scientifique stimulé par le programme japonais d'ordinateurs de cinquième génération. Le langage ABCL développé à l'université de Tokyo [Yone90] dans le cadre de ce programme pourrait fournir prochainement la base d'un système d'exploitation adapté aux nouvelles architectures parallèles à mémoire distribuée et communication asynchrone développées dans le cadre du programme RWC[1]. La programmation à objets concurrents est aussi considérée, sous le nom de programmation à agents, comme une voie d'avenir pour le multimédia [Gene93] ou les interfaces utilisateurs des environnements de travail de groupe (CSCW[2]).

Un besoin d'unification des langages existants

Cependant, l'expérience acquise en programmation à objets concurrents reste encore insuffisante pour pouvoir choisir un modèle ou un langage plutôt qu'un autre. La plateforme Actalk est née de ce constat en proposant sous la syntaxe et au sein de l'environnement unique Smalltalk de manipuler des objets concurrents issus de modèles et de langages multiples.

1. RWC est un programme international de R&D financé par le MITI japonais.
2. *Computer Supported Cooperative Work*.

De nombreuses problématiques de recherche

L'expérience du projet Actalk a conduit à l'étude de nombreuses problématiques de recherche comme :

- La mise au point de débogueur [Brio90].
- La mise en oeuvre de mécanismes concurrents réflexifs [Brio90].
- L'intégration avec une bibliothèque d'objets et le modèle MVC d'interface utilisateur [Brio90].

4.4.2 Les modèles concurrents d'Actalk

Actalk est constitué d'une part par un noyau qui fournit les mécanismes élémentaires de création et de gestion d'objets concurrents. Ce noyau est complété d'autre part par une bibliothèque de classe pour proposer plusieurs sémantiques de la concurrence ainsi que des mécanismes de gestion de la synchronisation entre processus concurrents.

Langages et modèles de synchronisation

Actalk intègre un certain nombre de classes permettant de simuler la sémantique de divers langages concurrents ou de techniques de contrôle et de synchronisation de processus concurrents.

TABLE 25 Les modèles simulés par Actalk

Modèle / langage	Envoi de message	Activité	Constructeurs	Synchronisation
Par défaut	asynchrone & unidirectionnel	réactive & sérialisée	non	non
Actors	asynchrone & unidirectionnel	réactive & sérialisée	substitution de comportement	substitution de comportement
ABCL/1	3 types : maintenant, passé & futur 3 modes : standard & express	réactive & sérialisée	*wait for*, *sender*, *atomic*, *non resume*	*wait for* (*where* contrainte)
POOL2	synchrone & routines publiques	corps autonome & sérialisée	*answer* (serve), *unblocking answer*, *postactions*	*answer*

Modèle / langage	Envoi de message	Activité	Constructeurs	Synchronisation
Concurrent Eiffel	asynchrone & futur	corps (routine active) & sérialisée	unblocking serve	unblocking serve
Concurrent Smalltalk II	asynchrone, synchrone & futur	réactive & sérialisée	relinquish, post-actions	relinquish
ACT++	asynchrone & futur	réactive & sérialisée	transition d'états abstraits	états abstraits
OCore	asynchrone & futur (q-structure)	réactive & sérialisée	événements définis par l'utilisateur	événements définis par l'utilisateur et métaniveau
Etats abstraits	quelconque	quelconque	non	états abstraits
Gardes	quelconque	sérialisée	non	gardes
Gardes et compteurs de synchronisation	quelconque	concurrent	non	gardes
Invocations génériques	quelconque	concurrent	non	gardes sur invocations génériques
Hybride	quelconque	concurrent	non	états abstraits sur gardes

Chaque modèle concurrent est associé à un exemple simple de mise en oeuvre. Actalk intègre de plus des exemples permettant de faciliter l'apprentissage de la programmation à objets concurrents, dont le problème des Buffer bornés et celui de Philosophes.