Conclusion et perspectives

Rappel de la problématique

Nous avons introduit dans cette thèse le terme d'*adaptabilité en entrée* comme une combinaison de trois propriétés : la *contrôlabilité*, qui caractérise la capacité d'un système interactif à exploiter efficacement des entrées enrichies, l'*accessibilité*, qui définit sa capacité à exploiter des entrées appauvries, et la *configurabilité*, qui décrit sa capacité à pouvoir être finement personnalisée du point de vue des entrées. Ces trois propriétés sont fortement liées et constituent trois facettes d'une propriété plus générale, qui caractérise des systèmes interactifs entièrement « ouverts en entrée ». Après avoir montré en quoi il était essentiel de tendre vers ce type de système, nous avons mis en évidence le retard considérable accumulé par les systèmes interactifs grand public actuels, retard largement dû à des outils de développement figés, exclusivement basés sur le modèle de l'interaction standard (souris, clavier et techniques d'interaction WIMP). Les rares applications faisant usage d'entrées non conventionnelles, comme les jeux vidéo ou certaines applications semi-professionnelles, nécessitent une programmation artisanale extrêmement coûteuse en temps et en énergie.

Si la complexité et l'importance des problèmes liés à l'interaction en entrée sont aujourd'hui universellement reconnues par la communauté de l'interaction homme-machine, ce domaine a encore fait l'objet de peu de travaux. Nous avons présenté dans notre état de l'art les principaux modèles et outils pertinents du point de vue de l'interaction en entrée, et avons montré en quoi certaines approches constituaient une avancée dans le domaine de l'interaction en entrée, bien qu'aucun modèle ou outil à ce jour ne réponde aux exigences de l'adaptabilité en entrée sous tous ses aspects.

La spécificité de notre démarche

Les contributions de la recherche les plus pertinentes du point de vue de notre problématique sont les boîtes à outils graphiques avancées, qui se sont données pour objectif de corriger les principales faiblesses des outils de développement existants. Les contributions comme Subarctic et Garnet/Amulet montrent qu'un modèle d'interaction en entrée standard assez général peut être en partie étendu pour décrire des techniques plus évoluées. Les boîtes à outils Post-WIMP ont, de leur côté, introduit de nombreux modèles nouveaux d'interaction, spécifiquement adaptés à des paradigmes comme l'interaction gestuelle, l'usage de modalités ambiguës, l'interaction multi-utilisateurs ou encore l'interaction bimanuelle avec des outils semi-transparents. Malgré leurs apports indéniables, les boîtes à outils avancées s'appuient sur des modèles événementiels conventionnels, où seuls les dispositifs standard ou un nombre fixe de dispositifs étendus peuvent être pris en compte. Par ailleurs, ces outils exploitent ces dispositifs de manière efficace mais figée, ou au mieux difficile à étendre.

Depuis l'introduction des standards comme GKS, bien des modèles et des outils continuent d'employer des abstractions dans lesquelles les dispositifs d'entrée sont regroupés en classes d'équivalence stéréotypées. Les tâches d'interaction ont subi des simplifications analogues, notamment par l'introduction des widgets génériques. Essentiellement motivée par des problèmes de portabilité et de réutilisabilité, ce type d'approche empêche cependant de tirer parti des capacités intrinsèques de chaque dispositif et des caractéristiques de la tâche. Notre démarche se distingue des approches existantes dans le sens où elle néglige (dans un premier temps) les problèmes de portabilité pour tenter de répondre à la question suivante : sur un système concret, qui comprend un ensemble donné de dispositifs physiques, l'utilisateur peut-il efficacement tirer parti de ces dispositifs pour contrôler une application donnée ?

Il nous a paru nécessaire pour répondre à cette question de substituer aux dispositifs logiques des *dispositifs concrets*, qui apparaissent au plus bas niveau possible, c'est-à-dire comme des ensembles de canaux bruts ; de même, les objets du domaine doivent pouvoir être exposés *sans a priori sur la façon dont ils seront contrôlés*, et nous devons enfin disposer d'un moyen flexible de relier les canaux d'entrée à l'application interactive pour pouvoir la contrôler. Notre démarche s'inspire d'approches couramment employées dans les outils de développement d'applications 3D. Comme certains de ces outils, nous proposons un paradigme de programmation visuelle pour l'assignation des canaux : cette idée est pertinente dans la mesure où les canaux ne comportent pas de sémantique, et l'accès à tout nouveau dispositif physique dans un langage textuel nécessiterait des requêtes préalables à l'API de bas niveau pour connaître l'interface du dispositif. À l'inverse, un nouveau dispositif peut apparaître explicitement dans un langage visuel, auquel cas ses canaux ne nécessitent qu'une interprétation sémantique de la part de l'utilisateur pour pouvoir être exploités.

ICON et son modèle d'interaction en entrée

Le modèle d'interaction en entrée que nous avons proposé dans cette thèse est basé sur le principe de *configurations d'entrée*, dont les *dispositifs généralisés* constituent les briques de base. Les dispositifs généralisés sont des modules (agents) qui comportent des canaux en entrée et en sortie et peuvent traiter de l'information. Ils regroupent les *dispositifs système*, qui décrivent principalement des dispositifs d'entrée physiques, les *dispositifs utilitaires*, modules de traitement de données et de feedback, et les *dispositifs d'application*, qui exposent indépendamment des entrées les dimensions directement contrôlables de l'application. Une configuration d'entrée décrit la façon dont des dispositifs système sont connectés aux dispositifs d'application à travers des dispositifs utilitaires. Dans ce paradigme, les dispositifs utilitaires peuvent être vus aussi bien comme des adaptateurs que comme des « morceaux » de techniques d'interaction. Les configurations d'entrée s'appuient sur un modèle à flot de données qui permet de décrire des canaux structurés et des dispositifs composables et spécialisables. Elles emploient en outre un modèle d'exécution qui s'inspire des langages réactifs.

Le système ICON (Input Configurator) a permis à la fois de valider et d'affiner notre modèle, et de montrer la faisabilité d'un outil de prototypage et de développement pour des interfaces adaptables en entrée. ICON est une *boîte à outils d'entrées*, qui ne traite que les aspects « entrées » indépendamment des aspects « sorties », contrairement aux boîtes à outils existantes où ces deux notions sont habituellement imbriquées. Développé dans le langage Java, ICON comprend un ensemble riche et aisément extensible de dispositifs système et utilitaires, les dispositifs d'application étant déclarés par chaque application à travers un mécanisme simple. Les applications Java/Swing existantes peuvent également être contrôlées de manière générique à travers des dispositifs « de boîte à outils graphique ». ICON

peut accéder aux entrées et contrôler les applications à plusieurs niveaux, et peut être employé pour redéfinir tout ou partie de l'interaction en entrée. Il est par exemple possible d'ajouter une prise en charge des dispositifs spécialisés à une application 3D existante tout en conservant sa gestion standard des entrées.

ICON offre en outre des outils pour assurer une portabilité minimale des configurations d'entrée d'un système à l'autre, par des mécanismes qui permettent de décrire des équivalences entre dispositifs concrets sans notion de classe a priori. Les mécanismes des *descripteurs de dispositifs* sont uniques dans la mesure où ils permettent de jouer sur la finesse des descriptions pour rendre des configurations standard plus tolérantes aux différences de plate-formes avant distribution, ou au contraire leur permettre de dissocier sur une plate-forme donnée deux dispositifs très proches mais employés différemment (souris main gauche et souris main droite, par exemple).

Décrire l'interaction avec des flots de données

Notre approche à flot de données est innovante dans le domaine des applications interactives 2D, où les rares éditeurs visuels de comportement emploient comme Thinglab des systèmes à base de contraintes, principalement pour décrire des relations géométriques entre les widgets dans les applications WIMP. Seul l'outil 2D Whizz'ed a servi à décrire quelques techniques d'interaction bimanuelle par des flots de données, mais l'approche n'avait pas encore été généralisée à d'autres dispositifs et d'autres techniques Post-WIMP ou d'accessibilité. Les flots de données ont, nous l'avons vu, connu plus de succès dans le domaine de la 3D, mais les interactions que ces outils permettent actuellement de décrire sont encore limitées. ICON permet de contrôler des objets plus variés que des attributs de formes 3D, d'exploiter des dispositifs d'entrée de nature bien plus hétérogène, et permet d'employer ou construire des techniques d'interaction plus évolués que le simple contrôle direct.

Notre approche à flot de données se situe par ailleurs à l'opposé des démarches de modélisation orientées contrôle comme les réseaux de Pétri ou les automates à états, adaptés à la description du multiplexage temporel et des modes, mais non à la représentation et au traitement de l'information. La description du contrôle dans ICON est possible mais se fait généralement au détriment de la lisibilité. Son éditeur interactif fournit malgré tout des outils (raccourcis de dispositifs, zoom, encapsulation par composition) qui se sont révélés efficaces pour gérer ce type de complexité.

Nous pensons cependant que le tout-visuel a ses limites, et prônons une approche mêlant l'emploi d'opérateurs simples et de techniques d'interaction monolithiques développées dans un langage conventionnel. Notre expérience a en effet montré qu'à un certain niveau de granularité, l'interaction en entrée pouvait en grande partie être décrite comme des flux unidirectionnels de données et qu'à ce niveau, la configurabilité l'emportait avantageusement sur la complexité. Idéalement, l'emploi d'ICON devrait se limiter à l'insertion et à la paramétrisation d'adaptateurs prédéfinis comme le dispositif gestuel (qui traduit des séries de positions en commandes booléennes et produit du feedback) ou le dispositif « contrôle clavier » (qui convertit quatre canaux booléens en positions). Le rôle des dispositifs de traitement plus simples (opérateurs mathématiques ou booléens, traitement de signal, branchements conditionnels, animation) reste toutefois essentiel, car ils permettent d'affiner les techniques d'interaction prédéfinies et de construire des techniques répondant à des situations très spécifiques.

Les apports essentiels de notre approche

Le premier avantage de notre modèle d'interaction en entrée est qu'il permet de décrire toute la chaîne de gestion des entrées de manière explicite, du dispositif physique jusqu'à l'application. Dans les systèmes interactifs conventionnels, les techniques d'interaction sont concrètement implémentées sur plusieurs niveaux d'abstraction, la plupart de ces niveaux étant inaccessibles au programmeur d'applications ; celui-ci dispose en particulier d'une visibilité très limitée quant aux traitements effectués aux niveaux bas. À l'inverse, notre modèle expose tous les aspects de la gestion des entrées de manière homogène et modulaire, d'une manière qui permet de saisir des aspects habituellement obscurs de l'interaction en entrée : dans notre paradigme, les techniques d'interaction sont simplement des adaptateurs ou des compositions d'adaptateurs dont la disposition en série traduit des cascades de traitement de données et de feedback.

Une spécificité de notre approche est qu'elle repose sur un modèle d'exécution réactif. Les langages réactifs, dans lesquels la propagation de l'information est conceptuellement instantanée, sont particulièrement adaptés à la description de systèmes qui réagissent immédiatement à l'environnement. Aujourd'hui, seuls les aspects très bas-niveau de l'interaction sont gérés de façon réactive (pilotes de dispositifs, animation du pointeur), le reste étant pris en charge par des modèles à événements qui sont de type conversationnel (l'utilisateur attend que les événements soient traités). Nous pensons cependant que la majeure partie de l'interaction en entrée gagne à être décrite de façon réactive. Les aspects asynchrones et non-déterministes de l'interaction, liés par exemple à des techniques de reconnaissance vocale ou gestuelle, s'intègrent en outre dans notre modèle de façon très naturelle.

La métaphore de la connexion logicielle et le principe d'adaptateurs qui caractérisent notre modèle constituent des paradigmes naturellement appropriés à la description d'applications entièrement configurables en entrée. L'éditeur interactif d'ICON instrumente ces paradigmes avec des techniques d'interaction étudiées pour faciliter la construction de configurations d'entrée et offrir une configurabilité à un public plus large que les seuls développeurs. Les utilisateurs avancés peuvent ainsi, selon leur degré d'expertise, personnaliser des configurations d'entrée prédéfinies pour les adapter à leurs besoins ou à ceux d'autres utilisateurs. L'éditeur interactif d'ICON est également un outil de prototypage qui permet aux développeurs de construire et tester pour leur application un grand nombre de techniques de contrôle en fonction des dispositifs physiques potentiellement disponibles.

Les possibilités offertes par ICON et son éditeur interactif sont extrêmement nombreuses, et nous n'en n'avons exploré qu'un sous-ensemble. Nous avons décrit dans cette thèse plusieurs exemples de configurations fonctionnant avec des applications Swing déjà existantes ou avec une application de dessin conçue pour être exclusivement contrôlée avec ICON. Ces exemples illustrent diverses façons d'exploiter des dispositifs standard (clavier, souris) ou évolués (tablettes graphiques, reconnaissance vocale, dispositifs 3D) avec des adaptateurs prédéfinis (contrôle positionnel au clavier, interaction gestuelle, outils semi-transparents,…) ou par combinaison d'opérateurs simples (interaction bimanuelle, commandes vocales, techniques de bas niveau inédites), pour contrôler des applications interactives de façon générique ou dédiée.

Un avantage décisif de l'approche à flot de données est qu'elle est particulièrement adaptée à la description d'interactions Post-WIMP fortement concurrentes, directes et amodales. ICON encourage en outre l'emploi de dispositifs physiques multiples et la description de techniques d'interaction novatrices et dédiées à la tâche. De plus, ICON fournit les outils nécessaires pour décrire la majorité des paradigmes d'interaction non conventionnels que nous avons évoqués dans le premier chapitre. Les styles d'interaction pour lesquels ICON est moins adapté sont de deux types : d'abord, nous l'avons

vu, les comportements essentiellement orientés contrôle. Ensuite, les paradigmes Post-WIMP qui font un usage évolué de modalités naturelles : interprétations sémantiques avancées, fusion de modalités hétérogènes et prise en charge de l'ambiguïté. Nous qualifions ces dernières techniques de *communicationnelles* (l'interface est un interlocuteur intelligent) par opposition aux techniques *instrumentales* (l'interface est un ensemble d'outils) pour lesquelles ICON est le mieux adapté. Même des modèles à événements sophistiqués du type Multimodal Subarctic gagneraient cependant à reposer sur une infrastructure ICON pour décrire les niveaux bas de l'interaction en entrée.

Pour finir, ICON constitue une solution originale et efficace aux problèmes liés à l'adaptabilité en entrée. Il permet de décrire des applications hautement *contrôlables* grâce à une prise en charge de dispositifs d'entrée multiples, l'emploi d'abstractions de bas niveau pour décrire à la fois ces dispositifs et les besoins spécifiques de chaque application en termes d'entrées, la facilité de spécification d'interactions fortement concurrentes, et la présence d'une librairie extensible de techniques d'interaction Post-WIMP prêtes à l'emploi. Une *accessibilité* minimale est assurée par la présence d'un ensemble aisément extensible de techniques d'accessibilité génériques reposant sur le principe des adaptateurs, et peut être améliorée par la construction de techniques d'accessibilité dédiées à la tâche. L'éditeur interactif d'ICON permet en effet au développeur d'applications de répondre efficacement à un grand nombre de situations en termes d'entrées, qu'elles soient enrichies ou appauvries. Une fois construites, les configurations d'entrée sont hautement *configurables*, et peuvent être personnalisées à différents niveaux selon le degré d'expertise de l'utilisateur.

Exploitation d'ICON et perspectives

La boîte à outils en entrées ICON est distribuée sur Internet et est actuellement employée dans le cadre du projet GINA (Géométrie Interactive et NAturelle). Elle a d'abord été expérimentée sur une application de reconstruction interactive de scènes 3D à partir de photographies, et est actuellement utilisée dans le développement d'une application Post-WIMP de modélisation 3D exploitant les techniques de dessin d'architectes. ICON a également servi à monter une expérimentation visant à comparer l'utilisabilité de plusieurs techniques d'interaction. En outre, deux projets sont actuellement en cours pour une intégration totale d'ICON avec des boîtes à outils graphiques Post-WIMP.

Si ICON semble avoir aujourd'hui atteint la maturité nécessaire pour pouvoir être employé dans des projets de développement, notre travail demeure essentiellement exploratoire, et demande à être affiné, complété et généralisé. Une validation exhaustive de notre approche nécessiterait notamment des études d'utilisabilité portant sur l'éditeur interactif et la librairie de programmation ICON, la mise en œuvre d'applications plus nombreuses et plus complexes, et une formalisation de son modèle à flots de données réactif. Notre approche ouvre par ailleurs de nombreuses perspectives d'amélioration et d'extension. Nous avons par exemple évoqué des stratégies d'intégration avec des approches complémentaires (formalismes orientés-contrôle, modèles d'interaction communicationnels de haut niveau, modèles multicouches pour le feedback graphique), ainsi que des extensions possibles de notre modèle visant notamment à incorporer les caractéristiques pragmatiques des dispositifs physiques mis en évidence par Buxton.

Une perspective peut-être lointaine mais des plus intéressantes serait une généralisation de notre paradigme de connexion logicielle aux niveaux matériel et système d'exploitation, dans le but de tendre vers une nouvelle génération de plate-formes intégralement ouvertes en entrée. Selon nous, l'avènement du standard USB a rendu ce type d'évolution possible, notamment en fournissant un accès unifié et de niveau bas à un grand nombre de dispositifs d'entrée, et en autorisant les connexions et

les déconnexions à chaud. Nos postes de travail devraient non seulement autoriser les connexions dynamiques de dispositifs physiques divers et variés, mais devraient également permettre aux utilisateurs de spécifier leur emploi de façon simple et naturelle. Notre expérience avec ICON nous a convaincu de la faisabilité d'un tel système, et nous avons déjà évoqué un certain nombre de caractéristiques qui nous semblaient pertinentes pour une version grand public et allégée de notre configurateur, combinant dans des proportions raisonnables simplicité d'utilisation et pouvoir d'expression.

Remerciements

Merci à Jean-Daniel Fekete, pour ses qualités d'encadrant, à la fois scientifiques et humaines, pour la patience dont il a su faire preuve pendant ces nombreuses années où j'ai souvent douté, et pour avoir fini par me transmettre le virus de l'IHM. Il s'est beaucoup investi dans ces travaux, a été à l'origine de nombreuses idées, et ce sont véritablement les fruits d'un travail commun qui sont présentés dans ce mémoire. Je remercie aussi Gérard Hégron, pour sa patience également, et son soutien.

Merci à Stéphane Huot, pour m'avoir continuellement stimulé par son enthousiasme sans bornes et sa passion du travail en équipe, ainsi qu'aux autres thésards de l'équipe CMI : Frédéric Jourdan, Didier Boucard et Mohammad Ghoniem, avec lesquels je me suis senti très proche. Nul doute que je n'aurais pu venir à bout de ma thèse sans les encouragements et le soutien (et aussi la bonne humeur) de Stéphane, Frédéric et des autres. J'ai eu également grand plaisir à côtoyer les membres et ex-membres du département informatique de l'EMN, je pense en particulier à Geoffrey Subileau, Mathias Braux, Mohammed Tounsi, Abdallah, Philippe, Narendra, Rémi, Christine, Cédric et Christian, sans oublier Vanessa et les autres membres du CRITE. Merci aussi à Pierre Cointe pour m'avoir accordé sa confiance et offert son appui lors d'une période difficile.

Merci à ceux qui en montrant de l'intérêt pour mes travaux m'ont beaucoup encouragé, c'est-à-dire aux membres de mon jury de thèse, à Caroline Appert, et aux autres. Merci également à ceux de mes nouveaux collègues Toulousains du LIIHS, du CENA, et d'Intuilab, qui m'ont accompagné à la fin de mon calvaire et m'ont aidé lors de mes répétitions de soutenance.

Merci à mes proches pour avoir longtemps supporté mon irritabilité lorsqu'il fallait aborder le délicat sujet de l'avancement de ma thèse : mes parents et mon frère Christian, ainsi que Pascal, Nadir et Fatima, Paul et Lucie, Jérémy, Thomas, Stéphane et Florence. Merci aussi au personnel du Groupe Scolaire de la Maison d'Arrêt avec qui j'ai eu plaisir à travailler pendant la durée de mon service national : Jean-Marc Allain et tous les instituteurs spécialisés, les emplois-jeunes, Raphaël, ainsi que les détenus.

Merci tout particulièrement à Béatrice, ainsi qu'à sa proche famille : Jean, Jeanine, Philippe, Bernadette, Xavier, Thérèse, Laurent et Anita.

Annexe A

Structure d'une configuration d'entrée

Sommaire

Notre but dans ces trois annexes n'est pas d'introduire un nouveau formalisme data-flow, mais de décrire la structure et les mécanismes spécifiques aux configurations d'entrée de façon suffisamment univoque pour qu'ils puissent être compris et implémentés dans un langage impératif. Cette structure et ces mécanismes constituent le modèle ICOM *(Input Configuration Model)*. Dans la mesure du possible, nous essaierons de décrire ce modèle indépendamment de toute implémentation : les choix liés au style de programmation (hiérarchie des classes par exemple) sont laissés libres. Par ailleurs, nous emploierons le plus souvent des schémas graphiques génériques pour décrire les structures de données, indépendamment du langage visuel décrit dans le chapitre 3. Enfin, dans certains passages nécessitant des explications plus détaillées et à chaque fois que le langage naturel se révélera peu adapté à la description d'un mécanisme, des notations mathématiques simples seront employées.

Dans cette partie, nous introduisons les éléments de base du modèle ICOM, en décrivant leur structure et leurs relations. Cette partie concerne les *aspects statiques* des configurations d'entrée.

A.1 Vue d'ensemble

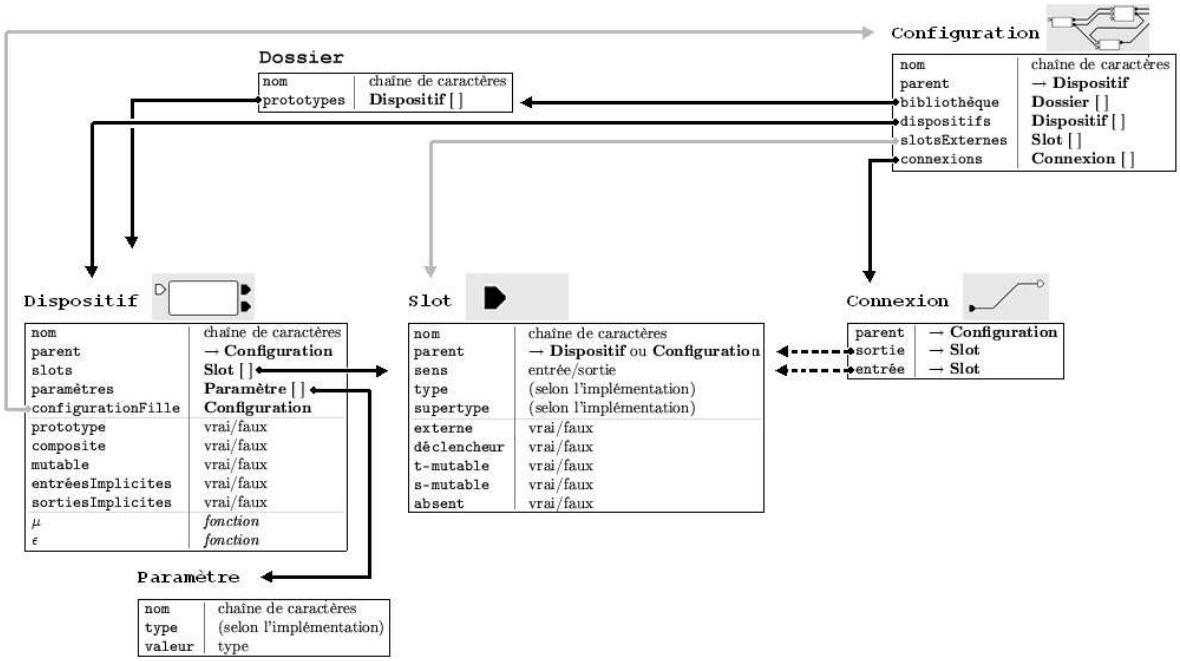


FIG. A.1 – Structure globale d'une configuration d'entrée, avec les briques de bases et leurs relations.

La figure A.1 fournit une vue globale de l'organisation structurelle d'une configuration d'entrée. Chaque entité est décrite par un ensemble d'attributs typés, c'est-à-dire une liste de noms (à gauche, en gras) et de types (à droite). Les flèches pleines décrivent les relations d'*agrégation* (X comporte un ou plusieurs Y) et les flèches en pointillés des relations de *référence* (X référence Y). Les flèches grisées traduisent des relations d'agrégation uniquement employées pour la composition et seront abordées plus tard.

La structure de base est la suivante : une configuration comporte un ensemble de *dispositifs* et

de *connexions*, ainsi que des *dossiers de dispositifs-prototypes*. Chaque dispositif comporte un ensemble de *slots* et de *paramètres*, et chaque *connexion* fait référence à deux slots. Un dispositif de type composite peut comporter une configuration dite *fille*, elle-même pouvant comporter des slots dits *externes*.

Dans la suite, nous décrivons les attributs des entités élémentaires de notre modèle dans l'ordre de la figure, de gauche à droite : le *dispositif*, les *paramètres*, le *dossier de prototypes*, le *slot*, la *connexion*, et la *configuration*.

A.2 Les quatre briques de base et leurs attributs

A.2.1 Dispositifs

La structure d'un dispositif est définie par un ensemble d'attributs et deux fonctions, qui sont énumérés dans le tableau A.2.1 puis décrits par la suite.

Dispositif	
`nom`	chaîne de caractères
`parent`	→ **Configuration**
`slots`	**Slot** [ ]
`paramètres`	**Paramètre** [ ]
`configurationFille`	**Configuration**
`prototype`	vrai/faux
`composite`	vrai/faux
`mutable`	vrai/faux
`entréesImplicites`	vrai/faux
`sortiesImplicites`	vrai/faux
μ	*fonction*
ε	*fonction*

TAB. A.1 – Structure d'un dispositif. La colonne de gauche liste le nom des attributs, celle de droite précise leur type.

- `nom :` Cet attribut contribue à identifier la fonction du dispositif, par exemple le type de traitement de données qu'il effectue. Il peut également refléter le rôle spécifique du dispositif au sein de la configuration.
- `parent :` Une référence à la configuration à laquelle appartient le dispositif, ou ou `null` s'il s'agit d'un dispositif-prototype.
- `slots :` Les *slots* d'un dispositif lui permettent de communiquer avec les autres dispositifs lors de la phase d'exécution (→ section A.2.2 page 199).
- `paramètres :` Un dispositif possède un ensemble éventuellement vide de *paramètres*. Les valeurs de ces paramètres déterminent certains aspects du comportement du dispositif pendant la phase de construction et d'exécution. Un dispositif ne peut pas comporter deux paramètres portant le même nom (→ section A.2.1 page suivante).
- `configurationFille :` La configuration-fille du dispositif, lorsqu'il s'agit d'un dispositif composite ; `null` sinon.

- `prototype :` Spécifie si le dispositif est un *dispositif-prototype* (→ section A.2.1).
- `composite :` Spécifie si le dispositif est de type *composite* (→ section A.4 page 204).
- `mutable :` Un dispositif est dit *mutable* s'il est susceptible de se spécialiser en fonction de la valeur prise par certains de ses attributs (→ annexe B).
- `entréesImplicites :` Indique si le dispositif reçoit des informations de l'environnement, c'est-à-dire des données autres que celles en provenance de ses slots d'entrée) (→ section C.5.1 page 232).
- `sortiesImplicites :` Indique si le dispositif émet des informations vers l'environnement, c'est-à-dire ailleurs que sur ses slots de sortie (→ section C.5.1 page 232).
- μ : μ est la *fonction de mutation* du dispositif, qui détermine le comportement de celui-ci pendant la phase de construction (→ annexe B).
- ε : ε est la *fonction d'exécution* du dispositif, qui détermine le comportement de celui-ci pendant la phase d'exécution (→ annexe C).

Paramètres

La structure d'un paramètre, décrite dans le tableau A.2.1, est simplement celle d'un attribut. Les paramètres peuvent être ainsi vus comme des attributs de dispositif supplémentaires.

Paramètre	
`nom`	chaîne de caractères
`type`	(selon l'implémentation)
`valeur`	type

TAB. A.2 – Structure d'un paramètre.

Dossiers de prototypes

Un dispositif-prototype est en tous points semblable à un dispositif, à ceci près qu'il ne fait pas partie d'une configuration, ne mute pas (il est obligatoirement dans un état consistant (→ section B.2.4 page 214)) et n'est jamais exécuté. Uniquement destiné à être dupliqué, celui-ci possède un état statique qui définit un paramétrage initial (valeur par défaut des paramètres) et un comportement (fonctions de mutation et d'exécution).

Les dispositifs-prototypes sont regroupés dans un dossier de prototypes dont la structure est la suivante :

Dossier	
`nom`	chaîne de caractères
`prototypes`	**Dispositif** [ ]

TAB. A.3 – Structure d'un dossier de prototypes.

A.2.2 Slots

Un *slot* est un état du dispositif qui est accessible aux autres dispositifs. Ces derniers ont la possibilité d'écrire sur le slot s'il s'agit d'un *slot d'entrée*, ou de lire sa valeur s'il s'agit d'un *slot de sortie* (figure 3.11 page 103). Ces échanges se font par l'intermédiaire des connexions (→ section A.3 page 201).

Les attributs d'un slot sont énumérés dans le tableau ci-dessous, et détaillés par la suite.

Slot	
`nom`	chaîne de caractères
`parent`	→ **Dispositif** ou **Configuration**
`sens`	entrée/sortie
`type`	(selon l'implémentation)
`supertype`	(selon l'implémentation)
`externe`	vrai/faux
`déclencheur`	vrai/faux
`t-mutable`	vrai/faux
`s-mutable`	vrai/faux
`absent`	vrai/faux

TAB. A.4 – Structure d'un slot.

- `nom :` Cet attribut identifie le slot au sein du dispositif. Le nom de chaque slot d'entrée d'un dispositif est *unique*, de même pour ses slots de sortie. Les noms comportant des points « . » définissent des slots structurés (structure cependant non explicite dans ICOM).
- `parent :` Une référence au dispositif ou à la configuration à laquelle appartient le slot, selon qu'il s'agisse d'un slot de dispositif ou d'un slot externe.
- `sens :` Cet attribut vaut `entrée` ou `sortie`, selon qu'il s'agisse d'un *slot d'entrée* ou d'un *slot de sortie*. Ce critère restreint l'ensemble des connexions possibles entre slots (→ section A.3 page 201).
- `type :` Le *type* d'un slot décrit le type de données qui y transitera lors de la phase d'exécution. Dans la phase de construction, celui-ci détermine simplement la *compatibilité* des connexions (→ section A.3.2 page 201).
- `supertype :` Le supertype d'un slot décrit l'ensemble des types que peut posséder un slot t-mutable. (→ annexe B)
- `externe :` Un slot *externe* est un slot isolé n'appartenant à aucun dispositif. Il sert à communiquer avec la configuration parente dans les dispositifs composites. Un slot non externe est également appelé *slot de dispositif*. (→ section A.4.1 page 204)
- `déclencheur :` Un slot est dit *déclencheur* s'il est susceptible de provoquer une mutation de son dispositif parent. Cet attribut vaut `faux` pour les slots absents. (→ annexe B)
- `t-mutable :` Un slot est dit *t-mutable* s'il est susceptible de changer de type. (→ annexe B)
- `s-mutable :` Un slot est dit *s-mutable* s'il est susceptible d'être ajouté ou supprimé du dispositif. Un slot ne peut pas être simultanément s-mutable et t-mutable. En outre, un slot déclencheur ne peut pas être s-mutable. (→ annexe B)
- `absent :` Spécifie si le slot est un slot *absent* ou un slot *présent*. Les slots présents sont créés par le dispositif, et les slots absents sont créés par les mécanismes de mutation. (→ annexe B)

A.2.3 Connexions

Une *connexion* est une relation liant deux slots, indiquant que ces deux slots partageront la même valeur lors de la phase d'exécution. La structure d'une connexion est décrite dans le tableau A.2.3.

Connexion	
`parent`	→ **Configuration**
`sortie`	→ **Slot**
`entrée`	→ **Slot**

TAB. A.5 – Structure d'une connexion.

- `parent :` Une référence à la configuration parente de la connexion.
- `sortie :` Une référence à un slot de sortie appartenant à `parent` ou à l'un de ses dispositifs-fils.
- `entrée :` Une référence à un slot d'entrée appartenant à `parent` ou à l'un de ses dispositifs-fils.

Les connexions sont orientées *de la sortie vers l'entrée*, et seront parfois notées **s** → **e**, *s* étant le slot de sortie et *e* le slot d'entrée.

A.2.4 Configuration d'entrée

Une configuration d'entrée est principalement constituée d'un ensemble de *dispositifs* et de *connexions*. Elle peut éventuellement être encapsulée dans un dispositif composite, auquel cas elle possède un dispositif parent. La structure d'une configuration d'entrée est décrite ci-dessous :

Configuration	
`nom`	chaîne de caractères
`parent`	→ **Dispositif**
`bibliothèque`	**Dossier** [ ]
`dispositifs`	**Dispositif** [ ]
`connexions`	**Connexion** [ ]
`slotsExternes`	**Slot** [ ]

TAB. A.6 – Structure d'une configuration d'entrée.

- `nom :` Ce nom contribue à identifier la fonction et le rôle de la configuration d'entrée.
- `parent :` Une référence au dispositif parent de la configuration, lorsque celle-ci est encapsulée dans un dispositif composite, `null` sinon.
- `bibliothèque :` La bibliothèque de dispositifs de la configuration, composée d'un ensemble de dossiers de prototypes.
- `dispositifs :` L'ensemble des dispositifs de la configuration d'entrée.
- `connexions :` L'ensemble des connexions qui relient les dispositifs entre eux. Les connexions seront abordées en détail dans la section suivante.
- `slotsExternes :` Une configuration peut comporter des slots isolés dits *externes*, qui lui permettent de communiquer avec l'extérieur. Ces objets sont décrits dans la section A.4 page 204.

A.3 Les connexions

A.3.1 Contraintes sur les connexions

Tout ensemble de connexions doit obéir aux trois contraintes suivantes :

Couplage : Une connexion appartenant à une configuration *C* peut relier soit :

1. Un slot de sortie appartenant à un dispositif de *C* à un slot d'entrée appartenant à un dispositif de *C* ;
2. Un slot de sortie appartenant à un dispositif de *C* à un slot d'entrée externe appartenant à *C* ;
3. Un slot de sortie externe appartenant à *C* à un slot d'entrée appartenant à un dispositif de *C* ;

En résumé, une connexion doit relier deux slots de sens opposés et être orientée du slot de sortie vers le slot d'entrée, et ne peut relier deux slots externes.

Unicité en entrée : Il existe au plus une connexion comportant un élément `entrée` donné. Ceci implique notamment que les connexions multiples sur un slot d'entrée sont interdits, ainsi que les connexions redondantes.

Acyclisme : Un ensemble de connexions ne doit pas générer de dépendances cycliques : pour toute configuration *C*, le *graphe de dépendances* (→ section A.3.5 page 204) de la *décomposition totale* (→ section A.4 page 204) de *C* doit être acyclique.

A.3.2 Types et sous-typage

Hormis les contraintes énumérés précédemment, toute connexion est autorisée indépendamment des types des slots. Cependant, notre modèle de connexion est typé et différencie les connexions *compatibles* des connexions *incompatibles*.

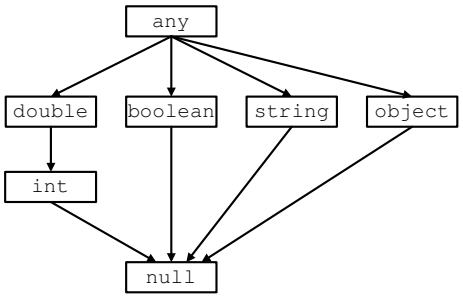


FIG. A.2 – Exemple de graphe de types pour les slots.

La figure A.2 représente le graphe de types que nous utilisons dans notre implémentation en Java, avec les relations de sous-typage. Les différents types de slots sont les `int` (nombres entiers), `boolean` (booléens), `double` (nombres flottants), `string` (chaînes de caractères), et `object` (objets). À cela s'ajoutent les types spéciaux `any` (qui représente tous les types possibles) et `null` (qui ne représente

aucun type). Ce graphe, comportant notamment le type `object`, est adapté à un langage à objets tel que Java.

Notre modèle ne nécessite pas la définition explicite d'un graphe de types. Au lieu de cela, il suppose qu'il existe un graphe de types avec à son sommet le type `any`, et à sa base le type `null`, et définit sur ce graphe les relations et opérations suivantes :

• Nous dirons que t_s est un ***sous-type*** de t (relation notée $t_s \subseteq t$) *ssi* t_s est successeur inclusif de t dans le graphe, c'est-à-dire ssi $t_s = t$ ou t_s est successeur direct ou indirect de t. De même, nous nommerons ***union*** de plusieurs types leur premier prédécesseur inclusif commun dans le graphe, et ***intersection*** de plusieurs types leur premier successeur inclusif commun dans ce même graphe. Les opérations d'union et d'intersection seront respectivement notées $\bigcup(t_1,\ldots,t_n)$ et $\bigcap(t_1,\ldots,t_n)$.

A.3.3 Attributs de connexions dérivés des slots

Nous nommons *attribut dérivé* un attribut d'un objet dont la valeur est fonction de celle des autres attributs de cet objet ou de ses objets accessibles (objets-fils ou objets référencés). Ils seront précédés d'un triangle ► . Ces attributs n'apportent pas d'information supplémentaire mais peuvent néanmoins servir à introduire des définitions utiles. Nous verrons également que lorsque la valeur de certains attributs est fixée dans un objet, d'autres attributs initialement libres peuvent devenir des attributs dérivés.

Les connexions comportent des caractéristiques de compatibilité, ainsi que d'autres attributs dérivés dont les valeurs sont fonction de leurs slots d'entrée et de sortie. Nous décrivons ces attributs, résumés dans le tableau A.3.3, en introduisant de nouveaux termes ainsi que leurs définitions.

Connexion	
`externe`	vrai/faux
`compatibilité`	compatible/incompatible
`compatibilitéStatique`	compatible/incompatible/mutable

TAB. A.7 – Attributs dérivés dans une connexion

► `externe :` Une *connexion externe* est une connexion reliant un slot de dispositif (slot pour lequel `externe=faux`) à un slot externe (slot pour lequel `externe=vrai`). Une connexion non externe, ou *connexion de dispositifs* est une connexion reliant deux slots de dispositifs. Une connexion ne peut relier deux slots externes.

► `compatibilité :` Une connexion est dite *compatible ssi* le type de son slot de sortie est un sous-type de celui de son slot d'entrée. Dans le cas contraire, elle est *incompatible*.

► `compatibilitéStatique :` Une connexion est dite *statiquement compatible ssi* le supertype de son slot de sortie est un sous-type de celui de son slot d'entrée, et que son slot d'entrée est ni s-mutable ni t-mutable. Une connexion est dite *statiquement incompatible ssi* le supertype de son slot de sortie n'est pas un sous-type de celui de son slot d'entrée, et que son slot de sortie est ni s-mutable ni t-mutable. Dans tous les autres cas, la connexion est dite *mutable*. Une connexion mutable est soit compatible soit incompatible, mais sa compatibilité peut changer suite à une mutation (→ annexe B). À l'inverse, une connexion statiquement compatible est toujours compatible, et une connexion statiquement incompatible est toujours incompatible.

La *compatibilité de types* n'entre pas en ligne de compte dans les contraintes de connexion vues précédemment car, comme nous le verrons dans l'annexe B, une connexion compatible peut très bien devenir incompatible au cours de l'édition et vice-versa. La *compatibilité statique* décrit cependant les évolutions possibles de cette compatibilité, et il est tout à fait possible (bien que nous ne le faisions pas) de refuser les connexions statiquement incompatibles.

A.3.4 Attributs de slots dérivés des connexions

De même que les slots induisent un certain nombre de propriétés sur les connexions, certaines propriétés caractérisent les slots en fonction des connexions existantes. Nous décrirons ces attributs dérivés tout en introduisant de nouveaux termes qui nous seront utiles dans les deux prochaines annexes. Ces attributs sont résumées dans le tableau A.3.4.

Slot	
`connexions`	**Connexion** [ ]
`slotsConnectés`	**Slot** [ ]
`connecté`	vrai/faux
`typeConnecté`	(selon l'implémentation)

TAB. A.8 – Attributs dérivés dans un slot

- `connexions :` Les *connexions d'un slot S* sont les connexions comportant *S* en entrée ou en sortie. Pour un slot d'entrée, cet attribut comporte au plus un élément.
- `slotsConnectés :` Les *slots connectés à S* sont les slots reliés à *S* par des connexions. Pour un slot d'entrée, cet attribut comporte au plus un élément.
- `connecté :` Un slot est dit *connecté* ssi il comporte au moins une connexion.
- `typeConnecté :` Le *type connecté* d'un slot de sortie est l'intersection des types de ses slots connectés, ou `any` s'il n'est pas connecté. Le *type connecté* d'un slot d'entrée est le type de son slot connecté, ou `null` s'il n'est pas connecté.

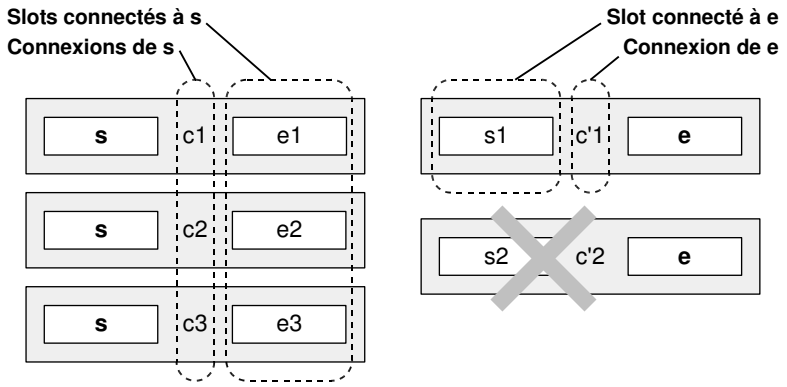


FIG. A.3 – Exemples de connexions : le slot de sortie *s* comporte trois connexions *c1*, *c2* et *c3* qui le relient aux slots d'entrée *e1*, *e2* et *e3*. Le slot d'entrée *e* comporte une connexion *c'1* qui le relie au slot de sortie *s1*. *e* ne peut comporter d'autre connexion.

A.3.5 Graphe de dépendances

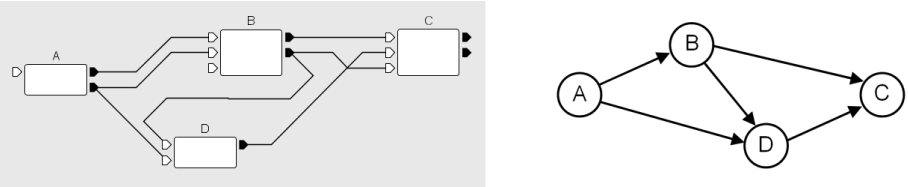


FIG. A.4 – Graphe de dépendances d'un ensemble de connexions.

Les connexions de dispositifs sont des arcs orientés reliant deux slots qui ont chacun un dispositif pour parent. Il est donc possible, à partir d'un ensemble de connexions de dispositifs, de construire un graphe orienté exprimant des dépendances entre les dispositifs (figure A.4). Ce graphe nous permet d'introduire les deux relations suivantes sur les dispositifs :

• Un dispositif D_1 est *successeur direct* d'un dispositif D_0 ssi il existe un arc $D_0 \rightarrow D_1$ dans le graphe de dépendances.

• Un dispositif D_1 est *successeur indirect* d'un dispositif D_0 ssi il existe un chemin qui va de D_0 à D_1 dans le graphe de dépendances. Cette relation est notée $D_0 \rightsquigarrow D_1$.

Dans les configurations ne comportant pas de dispositif composite, en particulier dans les configurations résultant d'une décomposition totale (→ section A.4), le graphe de dépendances est toujours un *graphe orienté acyclique*, ou DAG.

A.4 Composition de dispositifs

Les dispositifs sont *composables* : un ensemble de dispositifs interconnectés peut être décrit de façon équivalente par un dispositif unique appelé *dispositif composite*.

Un dispositif composite est essentiellement décrit par une configuration d'entrée, sa *configuration fille*. De façon générale, toute configuration d'entrée peut être encapsulée dans un dispositif composite, appelé *dispositif parent*. Ces deux relations de composition induisent également des relations hiérarchiques entre configurations et dispositifs : une configuration peut être *fille* ou *parente* d'une autre configuration du point de vue de la composition, de même que les dispositifs (figure A.5 page ci-contre). La communication d'un niveau à l'autre se fait par l'intermédiaire des *slots externes*.

A.4.1 Slots externes

Un *slot externe* est un type particulier de slot qui n'appartient pas à un dispositif mais à une configuration. Du point de vue de cette configuration, ce sont des points d'entrée et de sortie lui permettant d'échanger de l'information avec l'extérieur. Au niveau de la configuration parente, ils représentent les slots d'un dispositif composite : à chaque slot externe est associé un slot de sens opposé sur le dispositif composite (voir figure A.6 page suivante).

Un slot externe est uniquement défini par un *sens*, un *nom* et un *index*. Ses autres attributs sont

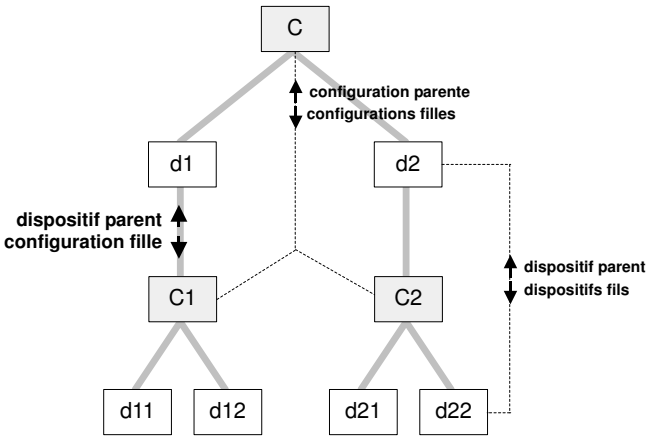


FIG. A.5 – Exemple de hiérarchies compositionnelles entre configurations et dispositifs : une configuration C comprend deux dispositifs composites $d1$ et $d2$ respectivement décrits par les configurations $C1$ et $C2$. Ces dernières comprennent chacun deux dispositifs.

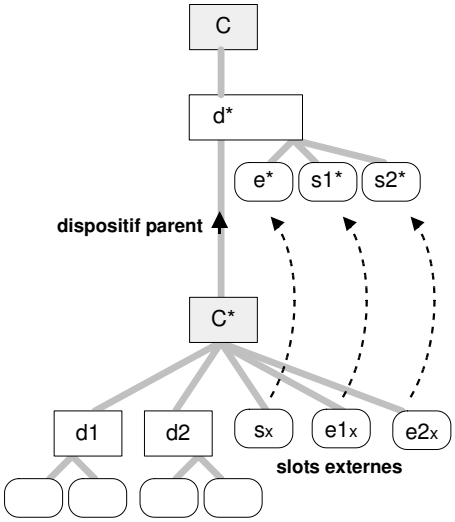


FIG. A.6 – La configuration $C^\star$ comprend trois slots externes s_x, $e1_x$ et $e2_x$ qui décrivent les slots de son dispositif parent d. À chaque slot d'entrée (resp. de sortie) externe est associé un slot de sortie (resp. d'entrée) sur le dispositif composite.

directement fonction des attributs de ses slots connectés (voir section suivante sur les connexions externes). Dans la liste qui suit, ces attributs dérivés sont précédés d'une flèche :

- `nom :` Cet attribut spécifie le nom du slot externe au sein de la configuration, ainsi que le nom du slot correspondant sur le dispositif composite. Dans une configuration d'entrée, les slots externes d'entrée ont des noms distincts, de même que les slots externes de sortie.
- `parent :` La configuration parente du slot composite.
- `sens :` Cet attribut vaut `entrée` ou `sortie`. Il spécifie si le slot externe est un slot externe *d'entrée* ou *de sortie*.
- ► `type :` Cet attribut a pour valeur le *type connecté* du slot externe (voir section suivante).
- ► `supertype :` Cet attribut a pour valeur `any`.
- ► `externe :` Cet attribut vaut `vrai`.
- ► `déclencheur :` Vaut `vrai` *ssi* il existe un slot connecté au slot externe pour lequel `declencheur=vrai`.
- ► `s-mutable :` Vaut `vrai` *ssi* il existe un slot connecté au slot externe pour lequel `s-mutable=vrai`.
- ► `t-mutable :` Vaut `vrai` *ssi* `s-mutable=faux` et il existe un slot connecté au slot externe pour lequel `t-mutable=vrai`..

A.4.2 Connexions externes

Comme nous l'avons vu précédemment, les connexions comportant un slot externe sont nommées *connexions externes*. Il existe deux types de connexions externes : les connexions externes *d'entrée* qui relient un slot d'entrée de dispositif à un slot externe de sortie, et les connexions externes *de sortie* qui relient un slot de sortie de dispositif à un slot externe d'entrée.

Les règles de connexion ainsi que les attributs et les relations définies dans la section A.3 page 201 sont les mêmes pour les connexions externes. Les contraintes d'unicité relatives aux connexions sur les slots d'entrée (section A.3 page 201) s'appliquent aux slots externes de sortie. Notons qu'une connexion externe est toujours compatible, car elle relie deux slots de même type.

A.4.3 Dispositifs composites

Un dispositif composite est entièrement défini par sa configuration fille. Hormis l'attribut `parent`, l'ensemble de ses attributs (précédés d'une flèche dans la liste ci-dessous) est fonction des attributs de cette configuration fille :

- `parent :` La configuration à laquelle appartient le dispositif, ou `null`.
- `configurationFille :` La configuration d'entrée qui décrit le dispositif composite.
- ► `nom :` Le nom d'un dispositif composite est le nom de sa configuration fille.
- ► `slots :` Les slots d'un dispositif composite correspondent aux slots externes *utilisés* dans la configuration fille, dans l'ordre indiqué par leur attribut `index`. Chaque slot du dispositif composite hérite des attributs du slot externe qui lui est associé.
- ► `paramètres :` Un dispositif composite ne comporte pas de paramètre.
- ► `composite :` Cet attribut vaut `vrai`.

► `prototype :` Cet attribut vaut `faux`.

► `mutable, entréesImplicites, sortiesImplicites :` Chacun de ces attributs vaut `vrai` *ssi* il est vrai pour au moins un dispositif dans la configuration fille.

A.4.4 Composition et décomposition

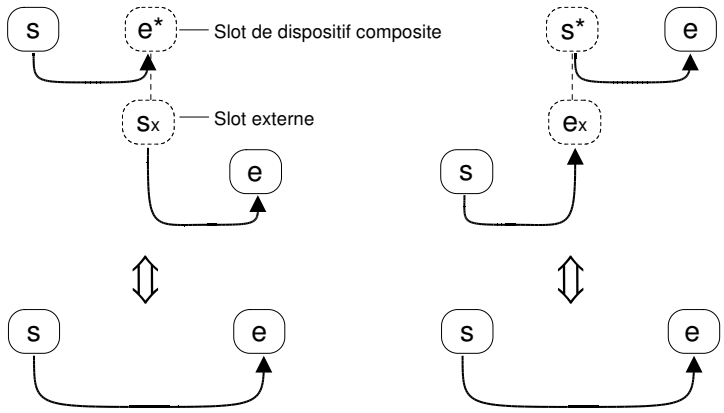


FIG. A.7 – Deux types de connexions inter-niveaux équivalentes à la connexion $s \rightarrow e$. À gauche, la connexion inter-niveaux *descendante* $(s \rightarrow e^{\star}, s_x \rightarrow e)$. À droite, la connexion inter-niveaux *ascendante* $(s \rightarrow e_x, s^{\star} \rightarrow e)$. Les slots intermédiaires permettent à la connexion $s \rightarrow e$ de remonter ou de redescendre d'un niveau dans l'arbre des configurations.

Nous appellerons *connexion inter-niveaux* toute paire de connexions reliant deux slots par l'intermédiaire d'un slot externe et de son slot de dispositif composite associé. Si cette connexion passe par le slot du dispositif composite puis par le slot externe, nous sommes dans le cas d'une connexion inter-niveaux *descendante* ; dans le cas contraire, il s'agit d'une connexion inter-niveaux *ascendante* (figure A.7). Dans les deux cas, la connexion inter-niveaux est équivalente du point de vue de l'exécution à une connexion directe entre le deux slots.

Des opérations de *composition* et de *décomposition* permettent de construire des structurations alternatives de l'arbre des configurations. La figure A.8 page suivante montre un exemple de composition ajoutant un niveau de configuration. Les dispositifs $d2$, $d3$ et $d4$ sont regroupés au sein d'un dispositif composite, qui peut ensuite être aplati pour redonner la configuration initiale.

La composition d'un ensemble de dispositifs Δ dans une configuration C s'obtient par l'algorithme A.1 page 209. L'opération inverse, consistant à décomposer un dispositif composite $d^{\star}$ dans C s'obtient par l'algorithme A.1 page 209. Ces algorithmes emploient la factorisation des connexions multiples sur un slot de sortie (figure A.9 page suivante).

Enfin, la *décomposition totale* d'une configuration C, utilisée pour le calcul des dépendances cycliques, s'obtient en appliquant récursivement l'algorithme A.1 page 209 sur l'ensemble des dispositifs de la configuration C.

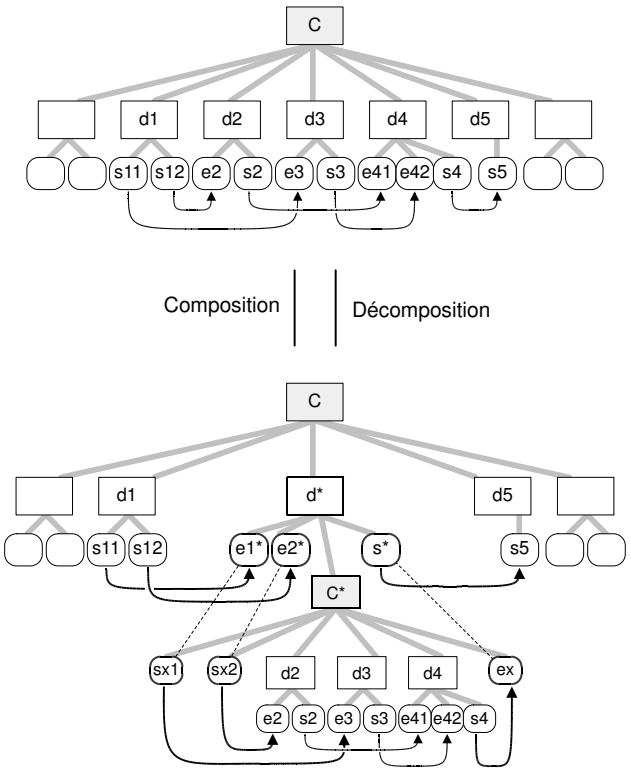


FIG. A.8 – Exemple de composition et de décomposition d'une configuration d'entrée. Les dispositifs $d1$, $d2$ et $d3$ sont déplacés dans la configuration-fille $C^\star$ et remplacés par le dispositif composite $d^\star$. Les connexions scindées telles que $s11 \rightarrow e3$ sont remplacées par des connexions inter-niveaux, tandis que les connexions internes telles que $(s2 \rightarrow e41)$ sont préservées (voir l'algorithme A.1). L'opération de décomposition supprime le dispositif composite et la configuration-fille, après avoir replacé les dispositifs-fils dans la configuration initiale et remplacé les connexions inter-niveaux par des connexions simples (voir l'algorithme A.2).

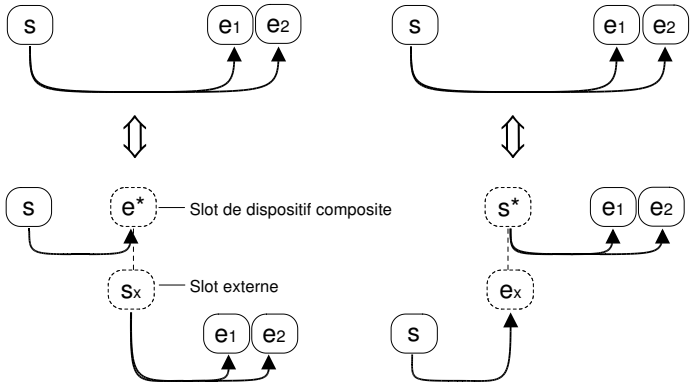


FIG. A.9 – Formes factorisées de connexions inter-niveaux multiples sur un slot de sortie.

1. Créer une configuration vide $C^\star$ contenant la même bibliothèque que C.
2. Créer un dispositif composite $d^\star$ de configuration-fille $C^\star$ et l'ajouter à C ;
3. Déplacer les dispositifs Δ vers $C^\star$;
4. Soit Σ l'ensemble des slots appartenant aux dispositifs Δ. Déplacer de C vers $C^\star$ toutes les connexions $s \rightarrow e$ de C vérifiant $s \in \Sigma$ et $e \in \Sigma$;
5. Pour toute connexion $s \rightarrow e$ de C vérifiant $s \notin \Sigma$ et $e \in \Sigma$:
   (a) s'il n'existe pas dans C de connexion $s \rightarrow e^\star$ avec $e^\star$ appartenant à $d^\star$:
   - ajouter un slot externe de sortie s_x dans $C^\star$, qui sera associé à un nouveau slot d'entrée $e^\star$ dans $d^\star$;
   - ajouter la connexion $s \rightarrow e^\star$ dans C ;
   - ajouter la connexion $s_x \rightarrow e$ dans $C^\star$.

   (b) s'il existe dans C une connexion $s \rightarrow e^\star$ avec $e^\star$ appartenant à $d^\star$:
   - ajouter la connexion $s_x \rightarrow e$ dans $C^\star$, s_x étant le slot externe associé à $e^\star$.
6. Pour toute connexion $s \rightarrow e$ de C vérifiant $s \in \Sigma$ et $e \notin \Sigma$:
   (a) s'il n'existe pas dans $C^\star$ de connexion $s \rightarrow e_x$ avec e_x slot externe :
   - ajouter un slot externe d'entrée e_x dans $C^\star$, qui sera associé à un nouveau slot de sortie $s^\star$ dans $D^\star$;
   - ajouter la connexion $s^\star \rightarrow e$ dans C ;
   - ajouter la connexion $s \rightarrow e_x$ dans $C^\star$.

   (b) s'il existe dans $C^\star$ une connexion $s \rightarrow e_x$ avec e_x slot externe :
   - ajouter la connexion $s^\star \rightarrow e$ dans C, $s^\star$ étant le slot de $d^\star$ associé à e_x.

Algorithme A.1: Composition des dispositifs Δ dans C.

A.4.5 Slots i-connectés

Il est utile d'introduire une définition alternative de la connexion, la *i-connexion*, qui ne prend en compte que les connexions effectives (éventuellement inter-niveaux) entre slots de dispositifs. Nous dirons ainsi qu'un slot de dispositif S' est *i-connecté* à un slot de dispositif S si S' est relié à S soit par une connexion, soit par une ou plusieurs connexions inter-niveaux successives.

Soit un slot S appartenant à un dispositif dans une configuration C. Soit C_0 la racine de l'arbre de configurations dont fait partie C.

• Le slot S' est *i-connecté* à S ssi S' est connecté à S dans la décomposition totale de C_0.

• *L'ensemble des slots i-connectés à S* est l'ensemble des slots connectés à S dans la décomposition totale de C_0.

1. Déplacer les dispositifs de $C^\star$ vers C ;
2. Déplacer toutes les connexions de dispositifs $s \rightarrow e$ de $C^\star$ vers C ;
3. Pour tout slot externe de sortie s_x de $C^\star$ et $e^\star$ son slot associé sur $d^\star$:
   (a) s'il n'existe pas de connexion $s \rightarrow e^\star$ dans C :
   - supprimer toutes les connexions $s_x \rightarrow e$ de $C^\star$.
   - supprimer s_x dans $C^\star$.

   (b) s'il existe une connexion $s \rightarrow e^\star$ dans C :
   - supprimer $s \rightarrow e^\star$ dans C ;
   - pour toute connexion $s_x \rightarrow e$ de $C^\star$:
     - supprimer $s_x \rightarrow e$ dans $C^\star$;
     - ajouter la connexion $s \rightarrow e$ à C ;
   - supprimer s_x dans $C^\star$.
4. Pour tout slot externe d'entrée e_x de $C^\star$ et $s^\star$ son slot associé sur $d^\star$:
   (a) s'il n'existe pas de connexion $s \rightarrow e_x$ dans $C^\star$:
   - supprimer toutes les connexions $s^\star \rightarrow e$ de C.
   - supprimer e_x dans $C^\star$.

   (b) s'il existe une connexion $s \rightarrow e_x$ dans $C^\star$:
   - supprimer $s \rightarrow e_x$ dans $C^\star$;
   - pour toute connexion $s^\star \rightarrow e$ de C :
     - supprimer $s^\star \rightarrow e$ de C ;
     - ajouter la connexion $s \rightarrow e$ à C.
   - supprimer e_x dans $C^\star$.
5. Supprimer le dispositif composite $d^\star$ et sa configuration-fille $C^\star$.

Algorithme A.2: Décomposition du dispositif $d^\star$ dans C.

Annexe B

Aspects dynamiques d'une configuration d'entrée

Sommaire

Dans cette partie, nous décrivons les principaux aspects dynamiques du modèle ICoM, qui traduisent le comportement d'une configuration d'entrée durant la phase de *construction* et d'*édition*. La plupart des notions abordées ici s'appuient sur les structures décrites dans l'annexe A.

B.1 Vue d'ensemble

L'essentiel du comportement en édition d'une configuration d'entrée repose sur les mécanismes de *mutation*. Les *dispositifs mutables* sont des dispositifs capables de se spécialiser ou se restructurer en modifiant la valeur de certains de leurs attributs en fonction de la valeur d'autres attributs, selon un mécanisme appelé *mutation*. Lors d'une mutation, le type de certains slots peut changer : ces slots sont nommés *t-mutables*. Une mutation peut également restructurer un dispositif en créant ou en supprimant des slots dynamiques, nommés *s-mutables*.

L'ensemble des attributs qui provoquent ces spécialisations et l'ensemble des attributs spécialisables constituent respectivement le *paramétrage* et le *m-paramétrage* du dispositif (figure B.1) :

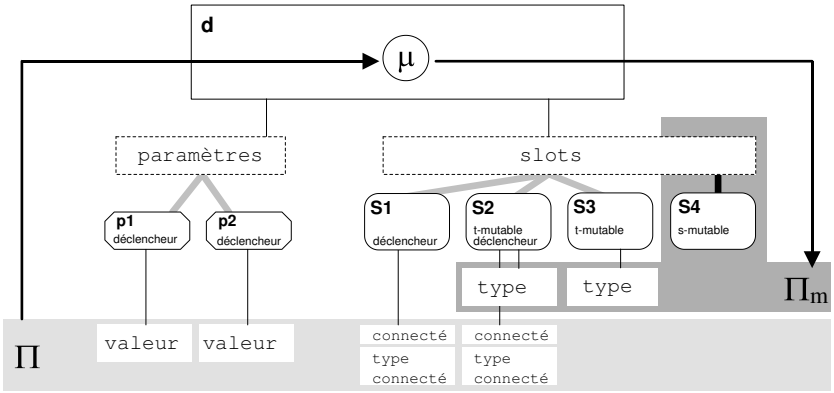


FIG. B.1 – Fonction de mutation μ d'un dispositif d comportant les paramètres $p1$, $p2$ et les slots $S1$, $S2$, $S3$ et $S4$. Π, la source de μ, est composé des valeurs prises par ses paramètres et des attributs de connexion de ses slots déclencheurs. Π_m, la cible de μ, décrit les types des slots t-mutables de d, ainsi que de l'ensemble des slots s-mutables que possède le dispositif après mutation.

1. Le *paramétrage* Π d'un dispositif décrit les valeurs prises par ses paramètres, et spécifie si ses slots déclencheurs sont connectés, ainsi que leurs types connectés.
2. Le *m-paramétrage* Π_m d'un dispositif décrit l'ensemble de ses slots s-mutables, ainsi que l'ensemble des types pris par ses slots t-mutables.
3. Chaque dispositif comporte une fonction de mutation $\mu : \Pi \mapsto \Pi_m$ qui à un paramétrage associe un m-paramétrage.

Après une introduction préliminaire sur les notions de signature et de valuation, nous définirons plus précisément Π, Π_m et la fonction μ, puis nous décrirons les mécanismes de base qui permettent d'appliquer et de propager les fonctions de mutation.

B.2 Définitions préliminaires

B.2.1 Types de slots

Un slot structurellement mutable ou *s-mutable* est un slot pour lequel l'attribut `s-mutable = vrai`. Un slot *t-mutable* est un slot pour lequel `t-mutable = vrai`, et un slot absent est un slot pour lequel `absent = vrai`.

Les slots s-mutables sont uniquement définis par leur `parent`, leur `nom`, leur `sens` et leur `type`. Les slots absents sont uniquement définis par leur `parent`, leur `nom`, et leur `sens`.

Un slot ne peut pas être simultanément s-mutable et t-mutable. En outre, *un slot déclencheur ne peut pas être s-mutable*.

B.2.2 Identifiants et valuations

Les définitions qui suivent s'appliquent à tout dispositif, qu'il soit mutable ou non.

• L'***identifiant*** $V(S)$ d'un slot S de dispositif est un couple formé de la valeur de ses attributs `nom` et `sens`, permettant de caractériser de façon unique ce slot sur son dispositif parent :

$V(S) = \langle S.nom, S.sens \rangle$

Tout couple de la forme $\langle nom, sens \rangle$ dont les éléments constitutifs ont pour types respectifs ceux des attributs de slots `nom` et `sens` est un *identifiant de slot*. Un identifiant de slot $\langle nom, sens \rangle$ ***désigne*** le slot S *ssi* $V(S) = \langle nom, sens \rangle$.

Soient $a1, \ldots, an$ des attributs de slots distincts, et différents des attributs `nom` et `sens`.

• La ***valuation*** $V_{a_1,\ldots,a_n}(S)$ d'un slot de dispositif S est un n-uplet comportant l'identifiant de S et les valeurs de ses attributs $a_1, \ldots, a_n$:

$V_{a_1,\ldots,a_n}(S) = \langle S.nom, S.sens, S.a_1, \ldots, S.a_n \rangle$

Tout n-uplet de la forme $\langle nom, sens, a_1, \ldots, a_n \rangle$, dont les éléments constitutifs ont pour types respectifs ceux des attributs de slots `nom`, `sens`, a_1, ..., a_n, est *une valuation de slot*, notée $V_{a_1,\ldots,a_n}$. Le couple $\langle nom, sens \rangle$ en constitue *l'identifiant*. Nous dirons qu'une valuation de slot *désigne* le slot S *ssi* son identifiant désigne le slot S.

De façon analogue, pour les paramètres :

• L'identifiant $V(P)$ d'un paramètre P est la valeur de son attribut `nom`, et la valuation $V_{valeur}(P)$ de ce paramètre est le couple formé par la valeur de ses attributs `nom` et `valeur`.

B.2.3 Paramétrages et m-paramétrages de dispositifs

Les définitions qui suivent s'appliquent à tout dispositif, qu'il soit mutable ou non.

• Le ***paramétrage*** d'un dispositif d est le couple :

$\Pi(d) = \langle V_{Pd}, V_{Sd} \rangle$

où V_{Pd} est l'ensemble des valuations V_{valeur} des *paramètres* de d :

$V_{Pd} = \{\langle P_1.nom, P_1.valeur\rangle, \ldots, \langle P_n.nom, P_n.valeur\rangle\}_{P_i \in d.parametres}$

et V_{Sd} est l'ensemble des valuations $V_{connecte,typeConnecte}$ des *slots déclencheurs* de d :

$V_{Sd} = \{\langle S_1.nom, S_1.sens, S_1.connecte, S_1.typeConnecte\rangle, \ldots,$
$\langle S_{n'}.nom, S_{n'}.sens, S_{n'}.connecte, S_{n'}.typeConnecte\rangle\}_{S_i \in d.slots\ et\ S_i.declencheur=vrai}$

• Le ***m-paramétrage*** d'un dispositif d est le couple :

$\Pi_m(d) = \langle V_{Ss}, V_{St}\rangle$

où V_{Ss} est l'ensemble de valuations V_{type} des slots *s-mutables* de d :

$V_{Ss} = \{\langle S_1.nom, S_1.sens, S_1.type\rangle, \ldots,$
$\langle S_{n''}.nom, S_{n''}.sens, S_{n''}.type\rangle\}_{S_i \in d.slots\ et\ S_i.s-mutable=vrai}$

et V_{St} est l'ensemble de valuations V_{type} des slots *t-mutables* de d :

$V_{Ss} = \{\langle S_1.nom, S_1.sens, S_1.type\rangle, \ldots,$
$\langle S_{n'''}.nom, S_{n'''}.sens, S_{n'''}.type\rangle\}_{S_i \in d.slots\ et\ S_i.t-mutable=vrai}$

• Un ***m-paramétrage valide*** pour un dispositif d est un couple de la forme :

$\Pi_m = \langle V_{Ss}, V_{St}\rangle$

où V_{Ss} est un ensemble de valuations V_{type} :

$V_{Ss} = \{\langle nom_1, sens_1, type_1\rangle, \ldots, \langle nom_{n''}, sens_{n''}, typen_{n''}\rangle\}$

dont les identifiants sont distincts et ne désignent pas de slot présent non s-mutable de d (slots pour lesquels $absent = faux$ et s-mutable$= faux$).

et V_{St} est un ensemble de valuations V_{type} :

$V_{St} = \{\langle nom_1, sens_1, type_1\rangle, \ldots, \langle nom_{n'''}, sens_{n'''}, typen_{n'''}\rangle\}$

dont les identifiants sont distincts et désignent tous des slots t-mutables de d. En outre, chaque élément $\langle nom, sens, type\rangle$ de V_{St} identifiant S vérifie $type \leqslant S.supertype$.

Notons qu'un slot s-mutable étant entièrement défini par ses attributs `parent`, `nom`, `sens` et `type`, un ensemble de valuations V_{type} suffit bien à décrire l'ensemble des slots s-mutables d'un dispositif.

B.2.4 Fonction de mutation et consistance

Les définitions qui suivent s'appliquent à tout dispositif, qu'il soit mutable ou non.

Soit d un dispositif, $\mathcal{P}$ l'ensemble de ses paramétrages possibles, et $\mathcal{P}_m$ l'ensemble de ses m-paramétrages valides.

Le dispositif d comporte une ***fonction de mutation*** μ, qui à chaque paramétrage Π de $\mathcal{P}$ associe un m-paramétrage Π_m de $\mathcal{P}_m$:

$$\mu : \left| \begin{array}{l} \mathcal{P} \rightarrow \mathcal{P}_m \\ \Pi \mapsto \Pi_m \end{array} \right.$$

• Un dispositif est dans un ***état consistant*** *ssi* il est non mutable, ou s'il satisfait à sa fonction de mutation :

$\Pi_m(d) = \mu(\Pi(d))$

B.3 Algorithmes

B.3.1 Mutation d'un dispositif

La *mutation* d'un dispositif consiste en la réévaluation des valeurs de ses attributs à la suite d'une modification de son paramétrage, dans le but d'assurer sa consistance. Cette réévaluation n'est nécessaire que dans la mesure où ce dispositif est mutable. Le mécanisme de mutation d'un dispositif est décrit par l'algorithme B.1.

- Soit $\Pi_m = \langle V_{Ss}, V_{St} \rangle$ l'image de $\Pi(d)$ par μ.
- Soit $\Sigma^{=}$ l'ensemble des slots s-mutables de d désignés par une des valuations de V_{Ss}.
- Soit Σ^{-} l'ensemble des slots s-mutables non connectés de d, non désignés dans V_{Ss}.
- Soit Σ_a^{+} l'ensemble des slots s-mutables connectés de d, non désignés dans V_{Ss}.
- Soit Σ_a^{-} l'ensemble des slots absents de d, désignés dans V_{Ss}.
- Soit V^{+} l'ensemble des valuations de V_{Ss} ne désignant aucun slot de d.
- Soit Σ_t l'ensemble des slots t-mutables de d désignés dans V_{St}.
- Soit Σ un ensemble de slots initialement vide.

1. Pour chaque slot S de $\Sigma^{=}$:
   - mettre à jour l'attribut *type* de S avec la valeur x de la valuation $\langle S.nom, S.sens, x \rangle$ de V_{Ss} ;
   - si la nouvelle valeur de $S.type$ est différente de l'ancienne, ajouter S à Σ.
2. Pour chaque slot S de Σ^{-} :
   - supprimer le slot S du dispositif d.
3. Pour chaque slot S de Σ_a^{+} :
   - mettre à jour les attributs de S :
     $S.s-mutable = faux$, $S.absent = vrai$, $S.type = null$ si $S.sens = entre$ et $S.type = any$ si $sens = sortie$;
   - si la nouvelle valeur de $S.type$ est différente de l'ancienne, ajouter S à Σ.
4. Pour chaque slot S de Σ_a^{-} :
   - mettre à jour les attributs de S :
     $S.s-mutable = vrai$, $S.absent = faux$, $S.type = x$ avec $\langle S.nom, S.sens, x \rangle$ valuation de V_{Ss} ;
   - si la nouvelle valeur de $S.type$ est différente de l'ancienne, ajouter S à Σ.
5. Pour chaque valuation $\langle x, y, z \rangle$ de V^{+} :
   - créer un nouveau slot s-mutable $S_{[parent=d,nom=x,sens=y,type=z]}$ et l'ajouter à d.
6. Pour chaque slot S de Σ_t :
   - mettre à jour l'attribut *type* de S avec la valeur x de la valuation $\langle S.nom, S.sens, x \rangle$ de V_{St} ;
   - si la nouvelle valeur de $S.type$ est différente de l'ancienne, ajouter S à Σ.

Algorithme B.1: Mutation d'un dispositif d de fonction de mutation μ, et construction de l'ensemble Σ des slots qui ont changé de type.

L'algorithme de mutation met à jour les slots du dispositif en fonction du m-paramétrage, en supprimant, en ajoutant, ou en mettant à jour le type des slots. L'emploi de slots *absents* évite la suppression de connexions à la suite d'une mutation. En particulier, un slot supprimé puis créé à nouveau conservera ses connexions.

L'algorithme distingue six cas possibles pour chaque slot : Les slots s-mutables de d qui sont absents du m-paramétrage sont supprimés s'ils ne sont pas connectés (slots Σ^-) et transformés en slots absents s'ils sont connectés (slots Σ_a^+). Les nouveaux slots s-mutables décrits par le m-paramétrage sont ajoutés s'ils n'existent pas en tant que slots absents (valuations V^+), et dérivés des slots absents dans le cas contraire (slots Σ_a^-). Enfin, l'algorithme met à jour les types des slots t-mutables (slots Σ_t) et des slots s-mutables qui sont présents à la fois sur le dispositif et dans le m-paramétrage (slots $\Sigma^=$). À la fin de l'algorithme, Σ comprend l'ensemble des slots ayant changé de type, qui servira à la propagation des mutations.

B.3.2 Propagation des mutations

La mutation d'un dispositif d est provoquée par une ou plusieurs des causes élémentaires suivantes :

1. la modification du type connecté d'un des slots déclencheurs de d ;
2. une connexion ou une déconnexion sur l'un des slots déclencheurs de d ;
3. la modification de la valeur d'un des paramètres de d.

En outre, la mutation de d a pour résultat une ou plusieurs conséquences élémentaires qui peuvent être classées en quatre catégories :

1. la modification du type d'un slot t-mutable ou s-mutable de d ;
2. la suppression d'un slot s-mutable connecté de d ;
3. la suppression d'un slot s-mutable non connecté de d ;
4. l'ajout d'un slot s-mutable à d.

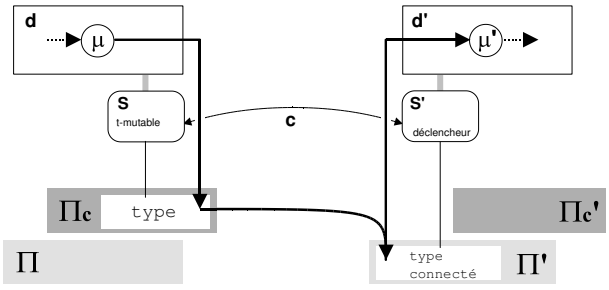


FIG. B.2 – Propagation d'une mutation : Le slot S, t-mutable, et le slot S', déclencheur, sont reliés par une connexion c de sens quelconque. À la suite d'une mutation du dispositif d, le type de S est modifié, ce qui a pour effet de modifier également le type connecté du slot S' et provoquer une mutation de d'.

Par l'intermédiaire des connexions, les conséquences de type 1 et 2 peuvent provoquer sur d'autres dispositifs des causes de type 1 et y déclencher des mutations : la mutation est alors *propagée* (figure B.2 page ci-contre). La mutation d'un dispositif est susceptible d'être propagée à tous les dispositifs directement connectés, *indépendamment du sens des connexions*. Ce mécanisme de propagation est décrit par l'algorithme B.2.

Répéter les étapes suivantes jusqu'à ce que Δ soit vide :

1. Copier Δ dans Δ' et effacer le contenu de Δ.
2. Pour tout dispositif d de Δ' :
   (a) Appliquer l'algorithme B.1 page 215 à d.
   Soit Σ l'ensemble des slots de d ayant changé de type.
   (b) Si $d \in \Delta$, retirer d de Δ.
   (c) Pour tout slot s appartenant à Σ, pour tout slot s' i-connecté à s :
   – Soit d' le dispositif parent de s'. Si s' est un slot déclencheur et si $d' \notin \Delta$, ajouter d' à Δ.

Algorithme B.2: Mutation avec propagation des dispositifs Δ dans une configuration C.

L'algorithme B.2 consiste à appliquer l'algorithme de mutation sur un ensemble de dispositifs, en se servant de la valeur de retour Σ pour déterminer l'ensemble des dispositifs vers lesquels les mutations doivent être propagées. L'algorithme est à nouveau appliqué sur ce dernier ensemble, et ainsi de suite jusqu'à ce qu'il n'y ait plus de dispositifs candidats. Notons que la propagation des mutations peut monter ou descendre dans l'arbre des configurations, à travers les slots i-connectés (voir section A.4.5 page 209).

Il est également important de noter que cet algorithme récursif opère en *largeur d'abord* plutôt qu'en *profondeur d'abord* : les dispositifs cibles de la propagation sont stockés au lieu d'être mutés immédiatement. Au niveau du slot, c'est l'algorithme de mutation (algorithme B.1 page 215) qui se charge du stockage des propagations. La propagation en largeur d'abord a pour avantage d'éviter des déclenchements inutiles de mutations (figure B.3).

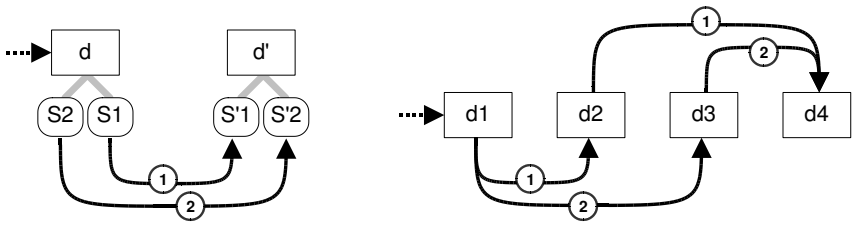


FIG. B.3 – Exemples de mutations inutiles. À gauche, une propagation en profondeur d'abord au niveau du slot : le dispositif d modifie son slot $S1$ puis son slot $S2$. La propagation instantanée des mutations déclenche deux mutations successives sur le dispositif d'. À droite, une propagation en profondeur d'abord au niveau du dispositif : le dispositif $d1$ déclenche une mutation sur $d2$ puis sur $d3$. Là aussi, la propagation instantanée déclenche deux mutations successives de $d4$.

B.3.3 Propagation bornée

Bien que les connexions d'une configuration d'entrée ne génèrent pas de dépendance cyclique, ce n'est pas le cas des mutations, qui se propagent indépendamment de l'orientation des connexions. Lorsque les mutations comportent des dépendances cycliques, l'algorithme B.2 page précédente peut ne pas se terminer, mais peut également converger.

L'algorithme B.3 est une variante de l'algorithme B.2 page précédente, qui se termine toujours. Le nombre de mutations sur chaque dispositif est limité à i_{max}. Lorsqu'un dispositif a muté i_{max} fois, celui-ci est considéré comme non stabilisé. À terme, l'algorithme fournit dans Δ_{NS} l'ensemble des dispositifs non stabilisés. Si cet ensemble est non vide, la propagation est considérée comme non stabilisée dans son ensemble.

De par sa propriété de terminaison, l'algorithme B.3 est de loin préférable à l'algorithme B.2 page précédente. Deux stratégies sont possibles pour l'exploiter : soit interdire les connexions qui induisent une propagation non stabilisée, soit autoriser provisoirement (hors exécution) les dispositifs non stabilisés dans une configuration. Un éditeur interactif pourrait par exemple mettre en évidence les « erreurs » dans l'espoir que l'utilisateur les corrige. Une manière de stabiliser la partie problématique de la configuration consiste à insérer des dispositifs « typeurs », dont les types en entrée et en sortie sont fixés par paramétrage : cela équivaut à ajouter des contraintes à un problème sous-contraint.

- Soit $i : \Delta'' \rightarrow \mathbb{N}$ une fonction qui associe un entier à des dispositifs, et dont l'ensemble de départ est initialement vide et Δ_{NS} un ensemble de dispositifs initialement vide également.
- Répéter les étapes suivantes jusqu'à ce que Δ soit vide :
  1. Copier Δ dans Δ' et effacer le contenu de Δ.
  2. Pour tout dispositif d de Δ' :
     (a) Si d ne fait pas partie de l'ensemble de départ de i, ajouter $d \mapsto i_{max}$ à i.
     (b) Si $i(d) = 0$, ajouter d à Δ_{NS}. Sinon :
        i. Appliquer l'algorithme B.1 page 215 à d.
        Soit Σ l'ensemble des slots de d ayant changé de type.
        ii. Décrémenter $i(d)$ dans i.
        iii. Si $d \in \Delta$, retirer d de Δ.
        iv. Pour tout slot s appartenant à Σ, pour tout slot s' i-connecté à s :
        – Soit d' le dispositif parent de s'. Si s' est un slot déclencheur et si $d' \notin \Delta$, ajouter d' à Δ.

Algorithme B.3: Mutation avec propagation bornée à i_{max} itérations des dispositifs Δ dans une configuration C. Cet algorithme construit également l'ensemble Δ_{NS} des dispositifs non stabilisés.

B.3.4 Exemples de fonctions de mutation

Le système de mutations, très général, peut se décliner en un certain nombre de mécanismes de spécialisation concrets. Parmi ceux-ci, nous donnerons un exemple de typage, de spécialisation structurelle par paramétrage, et d'adaptation bi-directionnelle. D'autres exemples peuvent être donnés, tels que la spécialisation structurelle par connexion, ou le typage par paramétrage.

Exemple de typage

Voici un exemple de dispositif dont les slots prennent le type `int` lorsque les slots d'entrée sont connectés à des entiers et le type `double` dans les autres cas :

Soit un dispositif mutable *add* ayant deux slots d'entrée nommés *in*1 et *in*2, un slot de sortie nommé *out*, et ne comportant pas de paramètre. *in*1, *in*2 et *out* sont t-mutables et possèdent comme supertype `double`. Les slots déclencheurs sont *in*1 et *in*2. La fonction de mutation μ_{add} de ce dispositif est la suivante :

$$\mu_{add} : \left| \begin{array}{l} \mathcal{P} \rightarrow \mathcal{P}_m \\ \Big\langle \varnothing, \big\{ \langle in1, entrée, c_1, x_1 \rangle, \langle in2, entrée, c_2, x_2 \rangle \big\} \Big\rangle \mapsto \\ \quad \Big\langle \varnothing, \big\{ \langle in1, entrée, y \rangle, \langle in2, entrée, y \rangle, \langle out, sortie, y \rangle \big\} \Big\rangle \end{array} \right.$$

avec :

$$\left| \begin{array}{l} y = int \text{ si } x_1 = int \text{ et } x_2 = int \\ y = double \text{ sinon.} \end{array} \right.$$

Exemple de spécialisation structurelle

Voici un exemple de dispositif dont le nombre de slots de sortie et d'entrée est paramétrable :

Soit un dispositif mutable *multipass* comportant deux paramètres respectivement nommés *inCount* et *outCount*. Le dispositif ne comporte pas d'autre slot que les slots s-mutables générés par la fonction de mutation. La fonction de mutation $\mu_{multipass}$ de ce dispositif est la suivante :

$$\mu_{multipass} : \left| \begin{array}{l} \mathcal{P} \rightarrow \mathcal{P}_m \\ \Big\langle \big\{ \langle inCount, x_1 \rangle, \langle outCount, x_2 \rangle \big\}, \varnothing \Big\rangle \mapsto \big\langle V_{Ss}^1 \cup V_{Ss}^2, \varnothing \big\rangle \end{array} \right.$$

avec :

$$\left| \begin{array}{l} V_{Ss}^1 = \varnothing \quad \text{si } x_1 \leq 0, \\ V_{Ss}^1 = \big\{ \langle in1, entrée, int \rangle, \ldots, \langle in_{x_1}, entrée, int \rangle \big\} \quad \text{sinon.} \end{array} \right.$$

$$\left| \begin{array}{l} V_{Ss}^2 = \varnothing \quad \text{si } x_2 \leq 0, \\ V_{Ss}^2 = \big\{ \langle out1, sortie, int \rangle, \ldots, \langle out_{x_2}, sortie, int \rangle \big\} \quad \text{sinon.} \end{array} \right.$$

Exemple d'adaptation bi-directionnelle

Enfin, voici un exemple de dispositif de type « adaptateur », dont les types s'adaptent en entrée et en sortie :

Soit un dispositif mutable *autocast* comportant un slot d'entrée et un slot de sortie respectivement nommés *in* et *out*, et ne comportant pas de paramètre. *in* et *out* sont déclencheurs et t-mutables, et possèdent comme supertype `any`. La fonction de mutation $\mu_{autocast}$ de ce dispositif est la suivante :

$$\mu_{autocast} : \left| \begin{array}{l} \mathcal{P} \rightarrow \mathcal{P}_m \\ \Big\langle \varnothing, \big\{ \langle in, entrée, c_1, x_1 \rangle, \langle out, sortie, c_2, x_2 \rangle \big\} \Big\rangle \mapsto \\ \quad \Big\langle \varnothing, \big\{ \langle in, entrée, x_1 \rangle, \langle out, sortie, x_2 \rangle \big\} \Big\rangle \end{array} \right.$$

B.4 Opérations sur les configurations

La structure d'une configuration d'entrée est susceptible d'évoluer dans le temps : elle peut être éditée. Cette évolution est réglée par un ensemble déterminé d'*opérations*, qui décrivent les évolutions possibles d'une configuration tout en imposant des contraintes sur celles-ci.

B.4.1 Opérations élémentaires

Une *opération élémentaire* sur une configuration est une fonction qui à une configuration C et un ensemble de dispositifs Δ associe une nouvelle configuration C' et un nouvel ensemble Δ'. Δ' comprend les dispositifs de Δ plus les dispositifs susceptibles d'être non consistants dans C', c'est-à-dire les dispositifs pouvant nécessiter une mutation.

Quatre opérations élémentaires sont énumérés par la suite, et décrits chacun par un algorithme simple montrant comment C' et Δ' s'obtiennent à partir de C et Δ.

Clonage de dispositifs

Le clonage d'un dispositif non composite d dans une configuration consiste à créer puis à ajouter dans cette configuration un nouveau dispositif possédant les mêmes valeurs d'attributs, mais ne comportant pas de connexion.

$Op_{cloner(d)} : (C, \Delta) \mapsto (C', \Delta')$

avec $d \in C.dispositifs$.

1. Création d'un dispositif d' vide ;
2. Copie dans d' des valeurs des attributs atomiques et des fonctions de d ;
3. Copie dans d' des paramètres et des slots non s-mutables et non absents de d ;
4. Ajout de d' à $C.dispositifs$;
5. Ajout de d' à Δ.

Ajout d'un dispositif

L'ajout d'un nouveau dispositif dans une configuration consiste en la copie d'un des dispositifs prototypes présents dans un des dossiers de prototypes de la configuration. L'opération d'ajout est la même que celle de clonage (même algorithme), à ceci près que le dispositif cloné appartient à l'un des dossiers de prototypes de la configuration.

$Op_{ajouter(p)} : (C, \Delta) \mapsto (C', \Delta')$

avec $p \in D.prototypes$ et $D \in C.bibliotheque$.

1. Appliquer $Op_{cloner(p)}$ à (C, Δ) ;
2. $p.parent = C$;

Connexion

Une opération de connexion consiste à relier un slot de sortie à un slot d'entrée par une nouvelle connexion. Elle est définie ainsi pour deux slots non externes :

$Op_{connecter(s,e)} : (C,\Delta) \mapsto (C',\Delta')$

avec :

- $s.parent \in C.dispositifs$
- $e.parent \in C.dispositifs$
- $s \rightarrow e$ obéit aux règles de couplage, d'unicité et d'acyclisme (→ section A.3 page 201).

1. Création d'une nouvelle connexion $x = s \rightarrow e$;
2. Ajout de x à $C.connexions$;
3. Ajout de $\{d_1, d_2\}$ à Δ.

Déconnexion

Une opération de déconnexion entre deux slots consiste à supprimer la connexion correspondante dans la configuration.

$Op_{deconnecter(s,e)} : (C,\Delta) \mapsto (C',\Delta')$

avec :

- $s.parent \in C.dispositifs$
- $e.parent \in C.dispositifs$
- $s \rightarrow e \in C.connexions$.

1. Suppression de la connexion $s \rightarrow e$ de $C.connexions$;
2. Ajout de $\{d_1, d_2\}$ à Δ.

Suppression d'un dispositif

La suppression d'un dispositif dans une configuration implique la suppression de toutes les connexions qui lui sont liées.

$Op_{supprimer(d)} : (C,\Delta) \mapsto (C',\Delta')$

avec $d \in C.dispositifs$

1. Pour toute connexion $s \rightarrow e$ de C telle que $s \in d.slots$ ou $e \in d.slots$, appliquer $Op_{deconnecter(s,e)}$ à (C,Δ) ;
2. Suppression de d dans $C.dispositifs$.

B.4.2 Opérations consistantes

Une *opération consistante* sur une configuration est une fonction qui à une configuration C associe une nouvelle configuration C' dont les dispositifs sont consistants. Une opération consistante peut être définie à partir d'une suite d'opérations élémentaires, de la manière suivante :

$Op_{(Op_1,\ldots,Op_n)} : C \mapsto C'$

1. Soit $\Delta = \varnothing$;
2. Appliquer dans l'ordre les Op_i sur (C, Δ) ;
3. Appliquer l'algorithme de mutation avec propagation sur Δ.

Annexe C

Exécution d'une configuration d'entrée

Sommaire

C.1 Introduction

Nous décrivons ici la partie exécution du modèle ICOM, c'est-à-dire le comportement des configurations d'entrée lors de la phase de lancement et d'exécution. Notre algorithme d'exécution est de type *réactif* ce qui signifie que chaque modification des entrées est répercutée et propagée dans la configuration en un temps conceptuellement nul (voir 3.3 page 99). La plupart des notions abordées ici s'appuient sur les structures décrites dans l'annexe A.

C.2 Définitions préalables

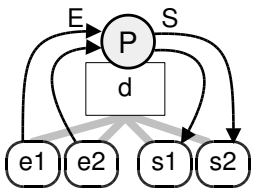


FIG. C.1 – Schéma d'un processeur.

Le comportement en exécution d'une configuration d'entrée est essentiellement décrit par des *processeurs*. Un processeur est une fonction qui aux valeurs prises par les slots d'entrée associe des valeurs aux slots de sortie (figure C.1). Lorsqu'une configuration est lancée, des structures sont allouées pour stocker la valeur de chaque slot, et chaque dispositif crée un processeur en fonction de son paramétrage, selon sa *fonction d'exécution* ε.

Les valeurs manipulées par les processeurs sont des valeurs de variables classiques associées à un booléen appelé *signal* : il s'agit de *signaux valués*. Un signal valué est propagé aux autres processeurs seulement si le signal est présent (booléen à vrai). Des modifications de la valeur d'une variable sont nécessairement propagées, mais une valeur non modifiée peut également être propagée. Un signal valué peut par conséquent représenter aussi bien un état qu'un événement.

Dans cette section, nous introduisons les définitions relatives aux signaux valués, aux processeurs, et aux fonctions d'exécution.

C.2.1 Signaux valués

Une *variable* est définie par ensemble appelé type et un élément de cet ensemble appelé valeur :

$V = \langle X, x \rangle, x \in X$

Un *signal valué* est la combinaison d'une variable et d'un booléen appelé *signal* :

$S = \langle V, s \rangle = \big\langle \langle X, x \rangle, s \big\rangle, x \in X, s \in \{vrai, faux\}$

La *valeur* d'un signal valué est un couple formé par la valeur x de sa variable V (appelée *valeur de variable* de S) et la valeur de son signal :

$v = \langle x, s \rangle$

L'ensemble des *valeurs possibles* de S, noté $\mathcal{V}(S)$, est $\mathcal{V}(S) = \{v\}_{v \in X \times \{vrai, faux\}}$.

C.2.2 Historiques

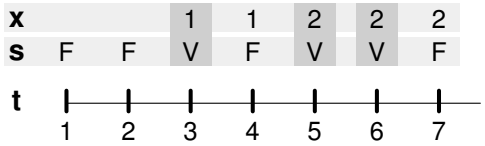


FIG. C.2 – Évolution possible d'un signal valué au cours du temps.

La valeur d'un signal valué évolue au cours du temps, que nous supposons discret et représenté par un entier strictement positif. Nous noterons $v^t = \langle x^t, s^t \rangle$ la valeur d'un signal valué à l'instant $t \in \mathbb{N}^*$.

Un signal valué évolue en accord avec la condition suivante :

$\forall t \in \mathbb{N}^*, x^{t+1} \neq x^t \Rightarrow s^{t+1} = vrai$

Autrement dit, le changement de la valeur de variable est une condition suffisante (mais non nécessaire) pour que le signal soit à vrai. En outre, un signal valué est considéré comme *n'ayant pas de valeur de variable* tant que son signal n'a pas reçu la valeur vrai au moins une fois. La figure C.2 illustre une évolution possible d'un signal valué au cours du temps, obéissant à ces deux conditions.

Un *historique* d'un signal valué S à l'instant t, noté $h^t(S)$, décrit une évolution de sa valeur depuis $t = 1$ jusqu'à t strictement positif :

$h^t(S) = \langle v^1, \ldots, v^t \rangle$

La condition de changement de valeur de variable énoncée plus haut est traduite par la définition de l'ensemble des historiques possibles $\mathcal{H}(S)$. La condition d'absence de valeur de variable est pour sa part traduite par la définition de la relation d'équivalence $\equiv$ entre deux historiques de $\mathcal{H}(S)$.

L'ensemble des *historiques possibles* $\mathcal{H}(S)$ d'un signal valué S est :

$\mathcal{H}(S) = \left\{ \langle v^1, \ldots, v^t \rangle \right\}_{t \in \mathbb{N}^*, v^i \in X_i \times \{vrai, faux\}, x^{i+1} \neq x^i \Rightarrow s^{i+1} = vrai}$

La *relation d'équivalence* $\equiv$ entre deux historiques de $\mathcal{H}(S)$ est définie comme suit :

Soient h et $h' \in \mathcal{H}(S)$.
$h \equiv h'$ ssi
$h = h'$ ou si h et h' sont de la forme :
$h = \left\langle \langle x, faux \rangle, \ldots, \langle x, faux \rangle, v^k, \ldots, v^t \right\rangle$
$h' = \left\langle \langle x', faux \rangle, \ldots, \langle x', faux \rangle, v^k, \ldots, v^t \right\rangle$

C.2.3 Signaux valués multiples

Par commodité pour la suite, nous étendrons les définitions précédentes aux signaux valués multiples du type $S = \langle S_1, \ldots, S_n \rangle$:

Les *valeurs* de S sont de la forme $v = \langle v_1, \ldots, v_n \rangle$. L'ensemble des *valeurs possibles* de S est $\mathcal{V}(S) = \left\{ \langle v_1, \ldots, v_n \rangle \right\}_{v_i \in X_i \times \{vrai, faux\}}$

Les *historiques* $h^t(S)$ de S sont de la forme :

$h^t(S) = \langle h^t(v_1), \ldots, h^t(v_n) \rangle = \left\langle \langle v_1^1, \ldots, v_1^t \rangle, \ldots, \langle v_n^1, \ldots, v_n^t \rangle \right\rangle$

L'ensemble des *historiques possibles* $\mathcal{H}(S)$ de S est :

$\mathcal{H}(S) = \left\{ \left\langle \langle v_1^1, \ldots, v_1^n \rangle, \ldots, \langle v_n^1, \ldots, v_n^t \rangle \right\rangle \right\}_{t \in \mathbb{N}^*, v_i^j \in X_i \times \{vrai, faux\}, x_i^{j+1} \neq x_i^j \Rightarrow s_i^{j+1} = vrai}$

La *relation d'équivalence* $\equiv$ entre deux historiques de $\mathcal{H}(S)$ est définie comme suit :

Soient h et $h' \in \mathcal{H}(S)$, avec $h = \langle h_1, \ldots, h_n \rangle$ et $h' = \langle h'_1, \ldots, h'_n \rangle$.
$h \equiv h'$ ssi $\forall i \in [1, n]$, $h_i \equiv h'_i$

C.2.4 Processeurs

Un *processeur* est une fonction qui à chaque historique d'un ensemble de signaux valués nommés *signaux d'entrée* associe des valeurs à d'autres signaux valués, nommés *signaux de sortie*.

Soit $P_{E,S}$ un processeur défini sur les signaux d'entrée $E = \langle e_1, \ldots, e_m \rangle$ et les signaux de sortie $S = \langle s_1, \ldots, s_n \rangle$. La fonction $P_{E,S}$ est définie comme suit :

$$P_{E,S} : \left| \begin{array}{l} \mathcal{H}(E) \rightarrow \mathcal{V}(S) \\ h^t(E) \mapsto v(S) \end{array} \right.$$

Cette fonction vérifie en outre la condition d'absence de valeur énoncée précédemment, c'est-à-dire qu'elle « ne peut pas lire » les variables qui n'ont pas encore reçu de signal. Cette condition s'énonce ainsi :

$\forall h$ et $h' \in \mathcal{H}(E)$, $h \equiv h' \Rightarrow P_{E,S}(h) = P_{E,S}(h')$

Un processeur $P_{E,S}$ est dit *passif* si en plus, il ne peut pas générer de signal (et à fortiori de valeur) sans en recevoir, ce qui se traduit par la condition suivante :

$\forall h \in \mathcal{H}(E)$
tel que $h = \left\langle \left\langle v_1^1, \ldots, v_1^{t-1}, \langle x_1^t, faux \rangle \right\rangle, \ldots, \left\langle v_n^1, \ldots, v_n^{t-1}, \langle x_n^t, faux \rangle \right\rangle \right\rangle$
l'image de h est de la forme $P_{E,S}(h) = \left\langle \langle y_1, faux \rangle, \ldots, \langle y_m, faux \rangle \right\rangle$

Dans le cas contraire, le processeur est dit *actif*.

C.2.5 Fonction d'exécution d'un dispositif

Comme nous l'avons vu en guise d'introduction, le comportement en exécution d'un dispositif est décrit par un processeur qui opère sur les signaux d'entrée et de sortie associés à ses slots d'entrée et de sortie (ces signaux valués sont créés lors du lancement de la configuration, nous le verrons dans la section suivante).

Le comportement en exécution dépend uniquement des valeurs prises par les paramètres du dispositif, ainsi que de ses types connectés s'il est mutable. À chaque *paramétrage* (→ section B.2.3

page 213) d'un dispositif correspond par conséquent un processeur. La correspondance paramétrage/processeur est décrite par la *fonction d'exécution* ε du dispositif, qui à chacun de ses paramétrages possibles associe un processeur P :

$$\varepsilon : \Pi(d) \mapsto P$$

C.3 Lancement et exécution d'une configuration

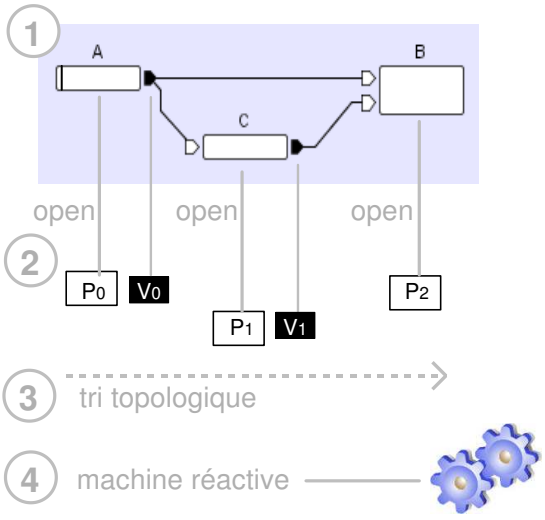


FIG. C.3 – Les quatre phases du processus de lancement d'une configuration.

Le *lancement* d'une configuration d'entrée consiste en un ensemble d'opérations destinées à préparer cette configuration d'entrée à l'exécution. Ces mécanismes aboutissent à la création d'une *machine réactive*, qui sera la version exécutable de la configuration d'entrée. Les différentes étapes de ce processus, illustrées sur la figure C.3 pour une configuration-exemple, sont les suivantes :

1. Aplatissement de l'arbre des configurations
2. Création des valeurs et ouverture des dispositifs
3. Tri topologique
4. Création de la machine réactive

La première phase consiste à aplatir l'arbre des configurations afin de placer tous les dispositifs au même niveau. L'algorithme, appelé décomposition totale, a déjà été décrit en annexe A (algorithme A.1). La seconde phase consiste à créer les structures de base qui serviront à l'exécution : les Valeurs (signaux valués) et les Processeurs. Un tri topologique est ensuite effectué sur les processeurs, à partir du graphe de dépendances entre dispositifs (→ section A.3.5 page 204). La dernière phase consiste à construire la machine réactive et remplir sa structure.

Nous ne décrirons pas le mécanisme d'aplatissement (déjà décrit en Annexe A) et le tri topologique (déjà connu [LACOMME *et al.* 03]). En revanche nous détaillerons la phase consistant à créer les «

structures de base », après avoir décrit ces structures. Puis nous décrirons les machines réactives et la façon dont elles sont créées.

C.3.1 Codage des signaux d'entrée et des processeurs

Dans la section précédente, nous avons introduit et décrit les processeurs à l'aide d'outils mathématiques simples. Dans cette partie, nous emploierons des représentations alternatives sous forme de structures de données.

Un signal valué est codé par une structure nommée `Valeur`, qui encapsule une *valeur de variable* et un *temps*. Le temps est un entier précisant à quel moment le signal valué a reçu un signal pour la dernière fois. Cette représentation est équivalente à celle des signaux valués vus précédemment : un signal est présent ($s = vrai$) si et seulement si le temps de la Valeur est égal au temps courant de la machine réactive (i.e. au temps global).

La structure `Valeur` est décrite ci-dessous :

Valeur	
`parent`	**Machine**
`valIndex`	entier
`valeur`	variable
`temps`	entier
`signal`	vrai/faux

TAB. C.1 – Structure d'un signal valué.

- `parent :` La machine réactive à laquelle appartient cette Valeur.
- `valIndex :` Spécifie l'indice de cette Valeur dans la machine réactive.
- `valeur :` La *valeur de variable* du signal valué, stockée dans une variable au sens langage de programmation. La représentation concrète de cette variable dépend du type du slot associé, ainsi que du langage de programmation choisi pour l'implémentation. Nous ne nous en préoccupons pas ici.
- `temps :` Le *temps* associé à la Valeur, spécifiant à quel moment elle a reçu un signal pour la dernière fois.
- ► `signal :` Attribut booléen dérivé spécifiant si un signal est présent. Cet attribut vaut `vrai` ssi le temps de la valeur est égal au temps de la machine réactive parente.

Un processeur est codé par une structure `Processeur`. Cette structure comporte des références à des Valeurs d'entrée et de sortie, ainsi qu'une décomposition de $P_{E,S}$ en deux séries de fonctions nommées *calculValeur* et *calculSignal*. Ces fonctions sont des projections de la fonction $P_{E,S}$ sur les valeurs et les signaux de son ensemble d'arrivée :

Soit $P_{E,S} : h^t(E) \mapsto v(S) = \langle \langle x_1, s_1 \rangle, \ldots, \langle x_n, s_n \rangle \rangle$.
$calculValeur[i] : h^t(E) \mapsto x_i$
$calculSignal[i] : h^t(E) \mapsto s_i$

- `entrées :` Cette table contient les Valeurs représentant les signaux d'entrée du processeur.
- `sorties :` Cette table contient les Valeurs représentant les signaux de sortie du processeur.

Processeur	
`entrées`	**Valeur** [ ]
`sorties`	**Valeur** [ ]
calculValeur	*fonction* [ ]
calculSignal	*fonction* [ ]

TAB. C.2 – Structure d'un processeur.

- *calculValeur* **:** Les projections de la fonction *P* sur les valeurs de variable de chaque signal de sortie. Cette table comporte autant de fonctions que d'éléments dans la table `valeursSortie`.
- *calculSignal* **:** Les projections de la fonction *P* sur les signaux de chaque signal de sortie. Cette table comporte autant de fonctions que d'éléments dans la table `valeursSortie`.

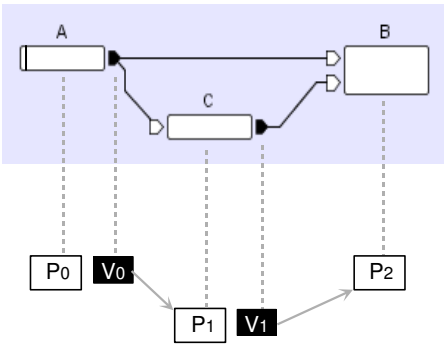
C.3.2 Création des valeurs et ouverture des dispositifs

La seconde étape phase du lancement d'une configuration d'entrée consiste à créer les structures contenant les valeurs des slots.

Une structure `Valeur` est créée *pour tout slot de sortie* de la configuration d'entrée. Les slots d'entrée ne comportent pas de Valeur (voir la figure C.3 page 227), mais référencent celle de leur slot connecté. Il n'y aura donc pas à proprement parler de propagation de valeur entre deux slots, puisque ceux-ci partageront la même valeur.

Une fois les valeurs créées, les dispositifs sont *ouverts* : lorsqu'un dispositif est ouvert, celui-ci retourne une structure Processeur créée à partir de sa *fonction d'exécution*.

C.3.3 Construction de la machine réactive



ProcInfo	**0**	**1**	**2**
Processeur	P_0	P_1	P_2
actif	V	F	F
déclenché	V	F	F

ValInfo	**0**	**1**
Valeur	V_0	V_1
variable	-	-
temps	0	0
depFirst	0	2
depLast	1	2

DepInfo	**0**	**1**	**2**
procIndex	2	1	2

temps

FIG. C.4 – Exemple de machine réactive créée à partir d'une configuration exemple.

La machine réactive gère tous les aspects de l'exécution des configurations d'entrée, et en particulier le *déclenchement* des processeurs et la propagation des valeurs. Un exemple de structure de machine réactive est illustré sur la figure C.4 page précédente, qui reprend la configuration-exemple de la figure C.3 page 227. Cette structure est la suivante :

Machine	
`ProcInfo`	(**Processeur**, vrai/faux, vrai/faux) [ ]
`ValInfo`	(**Valeur**, entier, entier) [ ]
`DepInfo`	vrai/faux [ ]
`temps`	entier

TAB. C.3 – Structure d'une machine réactive.

- `ProcInfo :` Cette table contient une liste topologiquement ordonnée des processeurs avec leurs conditions de déclenchement. Le booléen `actif` spécifie si le processeur doit être continuellement déclenché. La valeur de cet attribut dépend de la nature de la fonction $P_{E,S}$ du processeur (→ section C.2.4 page 226). Le second booléen, `déclenché` est interne à l'algorithme d'exécution. Initialement égal à `actif`, il servira à spécifier pour l'itération courante si les valeurs du processeur doivent être mises à jour.
- `ValInfo :` Cette table contient la liste des Valeurs ainsi que leurs dépendances (illustrées sur la figure C.4 page précédente par des flèches pleines). Les dépendances d'une Valeur sont définies par les attributs `depFirst` et `depLast`, qui référencent par leur index un ensemble d'éléments contigus dans la table `DepInfo`.
- `DepInfo :` Cette table contient la liste des dépendances référencées par `OutInfo`, et est remplie en même temps qu'elle. Lorsqu'un slot de sortie est traité par l'ajout d'une Valeur dans `OutInfo`, une dépendance est créé pour chaque slot d'entrée qui lui est directement connecté. Une dépendance est uniquement décrite par l'attribut `ProcIndex`, qui référence le processeur du dispositif parent du slot d'entrée, en pointant vers son index dans la table `ProcInfo`.
- `temps :` Cet attribut est interne à l'algorithme d'exécution au même titre que l'attribut `déclenché` de la première table. Initialement à zéro, ce compteur entier spécifiera le temps courant de la machine réactive.

C.4 Algorithme d'exécution

L'exécution est partagée entre les Processeurs qui mettent à jour leurs Valeurs de sortie, les Valeurs qui notifient la machine réactive, et enfin la machine réactive qui synchronise l'ensemble.

C.4.1 Mise à jour d'un processeur

La mise à jour d'un Processeur consiste à modifier ses valeurs de sortie, et à notifier la machine réactive parente afin que cette mise à jour soit propagée. L'ensemble de ce mécanisme est décrit par l'algorithme C.1 page suivante : la procédure *miseAJour* définie dans la structure `Processeur` applique les fonctions *calculValeur* et *calculSignal* à ses Valeurs de sortie. Elle délègue les assignations finales et les notifications à la procédure *miseAJour* définie dans la structure `Valeur`. Lorsqu'un signal est présent, cette dernière appelle la procédure *envoiSignal* de la machine réactive, décrite dans la section suivante.

```
proc Processeur.miseAJour()
   for i=0 to sorties.length do
      sorties[i].miseAJour(calculValeur[i](entrées), calculSignal[i](entrées))
   end for
end proc

proc Valeur.miseAJour(nouvelleValeur, signal)
   valeur ← nouvelleValeur
   if signal = vrai then
      machine.envoiSignal(valIndex)
   end if
end proc
```

Algorithme C.1: Mise à jour d'un processeur.

C.4.2 La boucle d'exécution

L'exécution d'une machine réactive consiste en une série d'itérations appelées *ticks*. Pendant un tick, tous les signaux sont propagés dans la machine réactive. La procédure *tick* (algorithme C.2) décrit une itération : la structure `ProcInfo` est parcourue dans l'ordre, et chaque processeur est mis à jour s'il est *déclenché*. La mise à jour d'un dispositif peut amener le déclenchement d'autres processeurs situés après lui dans la table.

```
proc Machine.tick()
   temps++
   for i=0 to ProcInfo.length do
      pinfo ← ProcInfo[i]
      if pinfo.déclenché then
         pinfo.Processeur.miseAJour()
      end if
      if not pinfo.actif then
         pinfo.déclenché ← faux
      end if
   end for
end proc

proc Machine.envoiSignal(int valIndex)
   dinfo ← DepInfo[valIndex]
   dinfo.Valeur.temps ← temps
   for i=dinfo.depFirst to dinfo.depLast do
      ProcInfo[DepInfo[i].procIndex].déclenché ← vrai
   end for
end proc
```

Algorithme C.2: Itération d'une machine réactive.

Cet algorithme de propagation assure qu'à la fin d'un tick, tous les changements ont correctement été propagés, et que la mise à jour d'un dispositif n'est *jamais effectuée plus d'une fois* : dans la configuration-exemple de la figure C.4 page 229, quand *B* reçoit une valeur du dispositif actif *A*, il attendra la mise à jour de *C* avant de se mettre à jour.

L'exécution d'une configuration d'entrée consiste en des appels successifs de la procédure *tick*. Notre modèle ne décrit pas comment ces appels sont effectués, en particulier s'ils une pause les sépare (pour laisser la main à d'autres processus ou garder une fréquence constante), ou encore s'ils sont appelés uniquement lorsque des valeurs sont présentes en entrée (interruptions matérielles).

C.5 L'environnement

Le modèle d'exécution que nous avons décrit est particulièrement bien adapté à la description de traitements de données *déterministes*. Cependant, un système interactif comporte souvent une part de non-déterminisme, en particulier du non-déterminisme temporel résultant de processus asynchrones, qui « prennent du temps ». En outre, il est évident que notre système nécessite d'être alimenté en informations, en particulier celles provenant des dispositifs d'entrée, et doit également « agir » à l'extérieur du système, soit sur l'application contrôlée, soit pour générer un retour utilisateur (feedback). Tous ces mécanismes ne peuvent être pris en compte sans inclure dans le modèle la notion d'*environnement*, système externe à la machine réactive.

Nous évoquerons dans cette partie comment nous pouvons, dans notre modèle et dans une éventuelle implémentation, prendre en compte cette communication, puis nous aborderons un autre problème lié à l'environnement, celui de l'hypothèse réactive.

C.5.1 Communication avec l'environnement : non-déterminisme et effets de bord

Lorsqu'un dispositif ne possède pas d'entrées ou de sorties *implicites*, son comportement lors de l'exécution est entièrement déterminé par un processeur opérant uniquement sur les signaux d'entrée et de sortie liés aux slots du dispositif. Dans le cas contraire, le dispositif communique avec l'*environnement*, et ce processeur ne peut décrire intégralement son comportement.

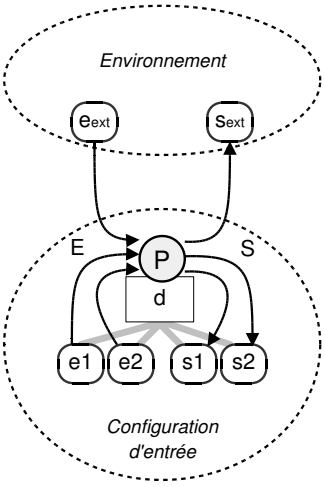


FIG. C.5 – Communication entre un processeur et un système extérieur à la configuration, appelé environnement.

Nous pouvons cependant généraliser notre modèle déterministe en supposant que l'environnement

est accessible à travers deux signaux valués (dont les types peuvent bien sûr être complexes) : le signal valué e_{ext} et le signal valué s_{ext}. Le dispositif interagit alors avec l'environnement en lisant e_{ext} (s'il comporte des entrées implicites) et en écrivant dans s_{ext} (s'il comporte des sorties implicites). Le comportement d'un tel dispositif peut alors être décrit avec un processeur $P_{E,S}$ défini sur les signaux valués relatifs aux slots auxquels on ajoute les signaux valués de l'environnement (figure C.5 page ci-contre) :

$$E = E_{slots} \cup \{e_{ext}\}$$
$$S = S_{slots} \cup \{s_{ext}\}$$

Notre modèle d'exécution décrit le comportement des configurations d'entrée, mais non le comportement de l'environnement (qui, par définition, ne fait pas partie de notre système), ni les mécanismes de communication avec cet environnement. En particulier, les variables associées à e_{ext} et s_{ext} ne sont pas explicites et ni leur type ni leur valeur ne sont connues : elles permettent uniquement d'évoquer, dans notre modèle, la présence de non-déterminisme (pour le premier) et d'effets de bord (pour le second).

Dans une implémentation concrète, la communication avec l'environnement (librairies et parties du code externes à la machine réactive) sera implémentée librement dans la fonction de mise à jour, mais devront cependant être explicitement déclarés par la présence d'entrées et de sorties implicites. La présence d'entrées implicites implique souvent un dispositif *actif*, qui a besoin de lire l'environnement même en l'absence de signaux en entrée : il y a donc un sens à ce qu'un dispositif à entrées implicites soit actif par défaut. La présence de sorties implicites n'a quant à lui pas d'influence sur les mécanismes d'exécution, mais peut apporter (tout comme les entrées implicites) une information utile sur la sémantique du dispositif dans une éventuelle représentation graphique.

C.5.2 Ouverture non-déterministe des dispositifs

Nous avons décrit la phase d'ouverture des dispositifs comme un mécanisme permettant de créer des processeurs de manière déterministe. Cependant, les dispositifs comportant des entrées ou des sorties implicites ont souvent besoin d'allouer au sein de l'environnement des ressources pour préparer leur exécution. Ces mécanismes d'initialisation peuvent, de manière non-déterministe, *réussir* ou *échouer* (dans le cas des dispositifs sans entrées ou sorties implicites, nous supposerons qu'elle réussit toujours).

Lorsque l'ouverture d'un dispositif d a réussi, celui-ci retourne, comme décrit précédemment, un Processeur construit à partir de sa fonction d'exécution $P = \varepsilon(\Pi(d))$. Dans le cas contraire, il retourne le *Processeur nul* décrit par la fonction $P_\varnothing$:

$$P_\varnothing : h^t(E) \mapsto \varnothing$$

C.5.3 L'hypothèse réactive dans ICoM

Dans une implémentation concrète, il est évident que le traitement de l'information (effectué par l'ensemble des processeurs) prend un certain temps, ainsi que la communication entre ces processeurs (pris en charge par la machine réactive). En pratique, cette dernière est souvent négligeable par rapport aux traitements de données. La durée d'un tick dans une machine réactive est par conséquent au moins égale à la somme des durées des traitements effectués par les processeurs. En outre, le temps

de réponse d'une machine réactive augmente linéairement avec la taille d'une configuration d'entrée.

Un modèle réactif ne reste valide que tant que l'hypothèse du synchronisme parfait est vérifiée, c'est-à-dire si la machine réactive réagit *plus vite que l'environnement*. Dans notre modèle, le signal valué e_{ext} est synchronisé sur l'horloge de la machine réactive. En supposant que e_{ext} « lit » des signaux que l'environnement lui envoie selon son horloge propre, l'hypothèse réactive est vérifiée si durant chaque tick, au plus un signal est émis par l'environnement.

e_{ext} décrit en général l'information provenant d'un dispositif d'entrée concret. En supposant, pour simplifier, que ce dispositif concret émet des informations à fréquence constante, la machine réactive doit réagir au moins à la même fréquence[1]. Pour une configuration d'entrée comportant plusieurs de ces dispositifs, la fréquence de réaction de la machine réactive doit être au moins égale à celle du dispositif émettant à la plus haute fréquence.

[1] Il n'est cependant pas nécessaire qu'une machine réactive réagisse à fréquence constante.

Bibliographie

[ACCOT *et al.* 98] Johnny ACCOT, Stéphane CHATTY, Yannick JESTIN, et Stéphane SIRE. « Conception des interfaces : et si nous analysions enfin la tâche du programmeur ? ». Dans *Prise de position pour les dixièmes journées francophones sur l'Interaction Homme Machine (IHM'98)*, septembre 2–4 1998.

[ACCOT *et al.* 97] Johnny ACCOT, Stéphane CHATTY, Sébastien MAURY, et Philippe PALANQUE. « Formal transducers : Models of devices and building bricks for the design of highly interactive systems ». Dans M. D. HARRISON et J. C. TORRES, éditeurs, *Design, Specification and Verification of Interactive Systems '97*, Eurographics, pages 143–159, Wien, 1997. Springer-Verlag. Proceedings of the Eurographics Workshop in Granada, Spain, June 4 – 6, 1997.

[ALIAS 01] « Maya real-time author (RTA) ». Alias Systems (Alias|Wavefront), décembre 2001. `http://www.alias.com`.

[APPERT *et al.* 03] Caroline APPERT, Michel BEAUDOUIN-LAFON, et Wendy MACKAY. « Context matters : Evaluating interaction techniques with the CIS model ». Rapport Technique 1372, Laboratoire de Recherche en Informatique (LRI), Université de Paris-Sud, 2003.

[APPLE 02] « Introduction to the aqua human interface ». Apple Computer, Inc., 2002. `http://developer.apple.com`.

[APPLE 03] « Interface builder ». Apple Computer, Inc., 2003. `http://developer.apple.com/tools/interfacebuilder/`.

[BALAKRISHNAN *et al.* 99] Ravin BALAKRISHNAN, George FITZMAURICE, Gordon KURTENBACH, et William BUXTON. « Digital tape drawing ». Dans *Proceedings of the 12th Annual ACM Symposium on User Interface Software and Technology*, pages 161–170, N.Y., novembre 7–10 1999. ACM Press.

[BARLOW *et al.* 90] Horace BARLOW, Colin BLAKEMORE, et Miranda WESTON-SMITH. *Images and Understanding : Thoughts about images : Ideas about understanding*. Cambridge University Press, 1990. ISBN 0-521-36944-4.

[BASTIDE *et al.* 02] Rémi BASTIDE, David NAVARRE, et Philippe PALANQUE. « A model-based tool for interactive prototyping of highly interactive applications ». Dans *CHI '02 extended abstracts on Human factors in computer systems*, pages 516–517. ACM Press, 2002.

[BAUDEL 95] Thomas BAUDEL. *Morphologie de l'Interaction Humain-Ordinateur : Étude de Modèles d'Interaction Gestuelle*. Thèse de doctorat, Université de Paris-Sud, U.F.R. scientifique d'Orsay, dec 1995.

[BAUDISCH *et al.* 03] P. BAUDISCH, E. CUTRELL, D. ROBBINS, M. CZERWINSKI, P. TANDLER, B. BEDERSON, et A. ZIERLINGER. « Drag-and-pop and drag-and-pick : Techniques for accessing remote screen content on touch- and pen-operated systems ». Dans *Proceedings of the 9th IFIP TC13 International Conference on Human-Computer Interaction (INTERACT'03)*, pages 57–64, septembre 2003.

[BEAUDOUIN-LAFON 97] Michel BEAUDOUIN-LAFON. « Interaction instrumentale : de la manipulation directe à la réalité augmentée ». Dans *Actes des Neuvièmes Journées sur l'Interaction Homme-Machine, IHM'97*, sep 1997.

[BEAUDOUIN-LAFON 00] Michel BEAUDOUIN-LAFON. « Instrumental interaction : an interaction model for designing post-WIMP user interfaces ». Dans Thea TURNER, Gerd SZWILLUS, Mary CZERWINSKI, et Paternò FABIO, éditeurs, *Proceedings of the 2000 Conference on Human Factors in Computing Systems (CHI-00)*, pages 446–453, N. Y., avril 1–6 2000. ACM Press.

[BEAUDOUIN-LAFON *et al.* 90] Michel BEAUDOUIN-LAFON, Yves BERTEAUD, et Stéphane CHATTY. « Creating direct manipulation applications with xtv ». Dans *Proc. European X Window System Conference*, novembre 1990.

[BEAUDOUIN-LAFON & LASSEN 00] Michel BEAUDOUIN-LAFON et Henry Michael LASSEN. « The architecture and implementation of CPN2000, a post-WIMP graphical application ». Dans *Proceedings of the 13th Annual Symposium on User Interface Software and Technology (UIST-00)*, pages 181–190, N.Y., novembre 5–8 2000. ACM Press.

[BEDERSON 03] Ben BEDERSON. « The piccolo 1.0 toolkit ». Human Computer Interaction Lab (HCIL), University of Maryland, avril 2003. `http://www.cs.umd.edu/hcil/piccolo/`.

[BEDERSON *et al.* 00] Benjamin B. BEDERSON, Jon MEYER, et Lance GOOD. « Jazz : an extensible zoomable user interface graphics toolkit in java ». Dans *Proceedings of the 13th Annual Symposium on User Interface Software and Technology (UIST-00)*, pages 171–180, N.Y., novembre 5–8 2000. ACM Press.

[BERRY 89] Gérard BERRY. « Real time programming : special purpose or general purpose languages ». Rapport interne RR-1065, Inria, Institut National de Recherche en Informatique et en Automatique, 1989.

[BERRY 99] Gérard BERRY. « The Esterel v5 language primer ». Rapport Technique, avril 1999. `http://www-sop.inria.fr/meije/esterel/doc/main-papers.html`.

[BERRY 00] Gérard BERRY. *The Foundations of Esterel*. MIT Press, 2000.

[BERRY *et al.* 87] Gérard BERRY, P. COURONNÉ, et G. GONTHIER. « Programmation synchrone des systèmes réactifs : le langage ESTEREL ». *Technique et Science Informatique*, 6(4), 1987.

[BIER *et al.* 93] E. BIER, M. STONE, K. PIER, W. BUXTON, et T. DEROSE. « Toolglass and magic lenses : The see-through interface ». *Proceedings of SIGGRAPH'93*, pages 73–80, 1993.

[BIER & FREEMAN 91] E. A. BIER et S. FREEMAN. « MMM : A user interface architecture for shared editors on a single screen ». Dans *Proceedings of the Fourth Annual Symposium on User Interface Software and Technology (UIST '91)*, pages 79–86, South Carolina, USA, novembre 11-13 1991. ACM Press.

[BIER & STONE 86] E. A. BIER et M. C. STONE. « Snap-dragging ». *Computer Graphics (Proc. ACM SIGGRAPH '86)*, 20(4) :233–240, 1986.

[BLANCH 02] Renaud BLANCH. « Programmer l'interaction avec des machines états hiérarchiques ». Dans *Proceedings of the 14th French-speaking conference on Human-computer interaction (Conférence Francophone sur l'Interaction Homme-Machine)*, pages 129–136. ACM Press, 2002.

[BOLT 80] Richard A. BOLT. « "put-that-there" : Voice and gesture at the graphics interface ». *Computer Graphics*, 14(3) :262–270, juillet 1980.

[BORNING 79] Alan Hamilton BORNING. *Thinglab - A Constraint-Oriented Simulation Laboratory*. PhD thesis, Stanford University, juillet 1979. Également disponible comme rapports de recherche STAN-CS-79-746 (Stanford Computer Science Department) et SSL-79-3 (XEROX Palo Alto Research Center).

[BOURIT 00] Cyril BOURIT. « Chamois : un logiciel de constructions géométriques ». août 2000. http://membres.lycos.fr/bourit/.

[BUXTON 83] William BUXTON. « Lexical and pragmatic considerations of input structures ». *Computer Graphics*, 17(1) :31–37, janvier 1983.

[BUXTON 86A] William BUXTON. « Chunking and phrasing and the design of human-computer dialogues ». Dans H. J. KUGLER, éditeur, *Information Processing '86, Proceedings of the IFIP 10th Worm Computer Congress*, pages 475–480. North Holland Publishers, 1986.

[BUXTON 86B] William BUXTON. « There is more to interaction than meets the eye : Some issues in manual input ». Dans D. A. NORMAN et S. W. DRAPER, éditeurs, *User Centred System Design : New Perspectives on Human-computer Interaction*, pages 319–337. Lawrence Erlbaum Associates, Hillsdale, NJ., 1986.

[BUXTON & MYERS 86] William BUXTON et Brad A. MYERS. « A study in two-handed input ». *Human Factors in Computing Systems*, pages 321–326, 1986.

[CADOZ 94] Claude CADOZ. « Le geste, canal de communication homme/machine : la communication instrumentale ». *Technique et Science de l'Information*, 13(1) :31–61, 1994.

[CAPPONI & LABORDE 91] B. CAPPONI et C. LABORDE. « Cabri-géomètre, un environnement pour l'apprentissage de la géomètrie élémentaire ». Dans *Actes de la VIème école d'été de didactique des mathématiques*, pages 220–22, 1991.

[CARD *et al.* 90] Stuart K. CARD, Jock D. MACKINLAY, et George G. ROBERTSON. « The design space of input devices ». Dans *Proceedings of ACM CHI'90 Conference on Human Factors in Computing Systems*, Multi-Media, pages 117–124, 1990.

[CARD *et al.* 91] Stuart K. CARD, Jock D. MACKINLAY, et George G. ROBERTSON. « A morphological analysis of the design space of input devices ». *ACM Trans. on Inf. Sys.*, 9(2) :99, 1991.

[CARR 91] R. M. CARR. « The point of the pen (PenPoint operating system) ». *Byte Magazine*, 16(2) :211–214, 216, 219–221, février 1991.

[CARSON 97] George S. CARSON. « Standards pipeline :the Open GL specification ». *Computer Graphics*, 31(2) :17–18, mai 1997.

[CHAPMAN 03] Davis CHAPMAN. *Visual C++ 6*. Le Programmeur. CampusPress, mars 2003. ISBN 2744015571.

[CHATTY 94] Stephane CHATTY. « Extending a graphical toolkit for two-handed interaction ». Dans *Proceedings of the ACM Symposium on User Interface Software and Technology*, Two Hands and Three Dimensions, pages 195–204, 1994.

[CMI 02] « Le projet GINA (Géométrie Interactive et NAturelle) ». École des Mines de Nantes, équipe CMI, 2002. http://www.emn.fr/x-info/gina/.

[COHEN *et al.* 89] P. R. COHEN, M. DALRYMPLE, D. B. MORAN, F. C. PEREIRA, et J. W. SULLIVAN. « Synergistic use of direct manipulation and natural language ». Dans *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 227–233. ACM Press, 1989.

[COUTAZ 87] Joelle COUTAZ. « PAC, an object-oriented model for dialog design ». Dans *Proceedings of IFIP INTERACT'87 : Human-Computer Interaction*, 2. Design and Evaluation Methods : 2.5 Dialogue Design and Evaluation, pages 431–436, 1987.

[CYCORE 03] « Cult3D Designer ». Cycore, 2003. http://www.cult3d.com/Cult3D/.

[DIX & RUNCIMAN 85] Alan DIX et Colin RUNCIMAN. « Abstract models of interactive systems ». Dans *Proceedings of the HCI'85 Conference on People and Computers : Designing the Interface*, The Design Process : Models and Notation for Interaction, pages 13–22, 1985.

[DOHERTY *et al.* 01] Eamon DOHERTY, Gilbert COCKTON, Chris BLOOR, et Dennis BENIGNO. « Improving the performance of the cyberlink mental interface with "yes / no program" ». Dans *Proceedings of ACM CHI 2001 Conference on Human Factors in Computing Systems*, Motion and Emotion, pages 69–76, 2001.

[DRAGICEVIC 98] Pierre DRAGICEVIC. « Concrétiser les dispositifs d'entrée dans les outils de développement », Prise de position. Dans *Actes des dixièmes journées francophones sur l'Interaction Homme Machine (IHM'98)*, pages 133–139, septembre 2–4 1998.

[DRAGICEVIC 01] Pierre DRAGICEVIC. « Une architecture en cascade pour des systèmes interactifs multi-dispositifs », Rencontres Doctorales, papier court. Dans A. BLANDFORD, J. VANDERDONKT, et P. GRAY, éditeurs, *People and Computers XV Interaction without Frontiers : Joint proceedings of IHM 2001 and HCI 2001*, volume 2, pages 191–192. Springer Verlag, 2001.

[DRAGICEVIC 02] Pierre DRAGICEVIC. « Page Web du projet ICON ». École des Mines de Nantes, équipe CMI, 2002. `http://www.emn.fr/x-info/icon/`.

[DRAGICEVIC & FEKETE 99] Pierre DRAGICEVIC et Jean-Daniel FEKETE. « Étude d'une boîte à outils multi-dispositifs ». Dans *Actes de la 11ième conférence francophone d'Interaction Homme-Machine (IHM99)*, pages 55–62. Cepadues, novembre 1999.

[DRAGICEVIC & FEKETE 00] Pierre DRAGICEVIC et Jean-Daniel FEKETE. « Input device selection and interaction configuration with ICON ». Rapport Technique 00/5/INFO, École des Mines de Nantes, département Informatique, novembre 2000.

[DRAGICEVIC & FEKETE 01] Pierre DRAGICEVIC et Jean-Daniel FEKETE. « Input device selection and interaction configuration with ICON ». Dans A. BLANDFORD, J. VANDERDONKT, et P. GRAY, éditeurs, *People and Computers XV Interaction without Frontiers : Joint proceedings of IHM 2001 and HCI 2001*, pages 543–558. Springer Verlag, 2001.

[DRAGICEVIC & FEKETE 02] Pierre DRAGICEVIC et Jean-Daniel FEKETE. « ICON : Input device selection and interaction configuration », Démonstration, papier court. Dans *UIST '02 Companion. The 15th Annual ACM Symposium on User Interface Software and Technology*, octobre 27–30 2002.

[DRAGICEVIC & FEKETE 03] Pierre DRAGICEVIC et Jean-Daniel FEKETE. « ICON : Towards high input adaptability of interactive applications ». Rapport Technique (en cours de publication), École des Mines de Nantes, département Informatique, décembre 2003.

[DRAGICEVIC & FEKETE 04] Pierre DRAGICEVIC et Jean-Daniel FEKETE. « The input configurator toolkit : Towards high input adaptability in interactive applications ». Dans SOUMIS À *AVI'2004 Advanced Visual Interfaces*. ACM Press, mai 25–28 2004.

[DRAGICEVIC & HUOT 02] Pierre DRAGICEVIC et Stéphane HUOT. « SpiraClock : a continuous and non-intrusive display for upcoming events ». Dans *CHI '02 extended abstracts on Human factors in computer systems*, pages 604–605. ACM Press, 2002.

[DUKE & HARRISON 93] D. J. DUKE et M. D. HARRISON. « Abstract interaction objects ». Dans R. J. HUBBOLD et R. JUAN, éditeurs, *Eurographics '93*, pages 25–36, Oxford, UK, 1993. Eurographics, Blackwell Publishers.

[DUNN 00] Jeff DUNN. « Developing accessible JFC applications ». Sun Microsystems' Accessibility Team, juin 2000. `http://www.sun.com/access/developers/developing-accessible-apps/`.

[DUNN & HERZOG 77] Robert M. DUNN et Bertram HERZOG. « Status report of the Graphics Standards Planning Committee of ACM/SIGGRAPH ». *Computer Graphics*, 11(3) :I–19 + II–117, Fall 1977.

[ECKERT *et al.* 79] R. ECKERT, G. ENDERLE, K. KANSY, et F.J. PRESTER. « GKS'79 - proposal of a standard for a graphical kernel system ». Dans *Proceedings of Eurographics'79*, 1979.

[ECKSTEIN & LOY 02] Robert ECKSTEIN et Marc LOY. *Java Swing*. O'Reilly & Associates, Inc., 981 Chestnut Street, Newton, MA 02164, USA, 2e édition, 2002. ISBN 0-596-00408-7.

[ELLIOTT *et al.* 94] Conal ELLIOTT, Greg SCHECHTER, Ricky YEUNG, et Salim ABI-EZZI. « TBAG : A high level framework for interactive, animated 3D graphics applications ». Dans Andrew GLASSNER, éditeur, *Proceedings of SIGGRAPH '94 (Orlando, Florida, July 24–29, 1994)*, Computer Graphics Proceedings, Annual Conference Series, pages 421–434. ACM SIGGRAPH, ACM Press, juillet 1994. ISBN 0-89791-667-0.

[ELO 02] « Keys to a successful kiosk application ». Elo TouchSystems, Inc., 2002. `http://www.elotouch.com`.

[ESTEBAN 97] Olivier ESTEBAN. *Programmation visuelle pour la construction d'interfaces homme-machine hautement interactives*. Thèse de doctorat, Laboratoire Interface Homme Systèmes (LIHS), avril 1997.

[ESTEBAN *et al.* 95] O. ESTEBAN, S. CHATTY, et P. PALANQUE. « Whizz'ed : a visual environment for building highly interactive software ». Dans *Proceedings of IFIP INTERACT'95 : Human-Computer Interaction*, pages 121–126, 1995.

[FEKETE & BEAUDOUIN-LAFON 96] Jean-Daniel FEKETE et Michel BEAUDOUIN-LAFON. « Using the multi-layer model for building interactive graphical applications ». Dans *Proceedings of the ACM Symposium on User Interface Software and Technology*, Papers : Tools, pages 109–118, 1996.

[FEKETE & DRAGICEVIC 00] Jean-Daniel FEKETE et Pierre DRAGICEVIC. « Une architecture multi-dispositifs ». Dans *Actes du Colloque sur les Interfaces Multimodales*, mai 9–10 2000.

[FEKETE *et al.* 98] Jean-Daniel FEKETE, Martin RICHARD, et Pierre DRAGICEVIC. « Specification and verification of interactors : A tour of esterel ». Dans Chris R. ROAST, éditeur, *Formal Aspects of Human Computer Interaction FAHCI'98*, pages 103–118. British Computer Society, septembre 1998.

[FEKETE 96A] Jean-Daniel FEKETE. « Les trois services du noyau sémantique indispensables à l'IHM ». Dans *Actes Huitièmes Journées sur l'Ingénierie des Interfaces Homme-Machine, IHM'96*, pages 45–50, septembre 1996.

[FEKETE 96B] Jean-Daniel FEKETE. *Un modèle multicouche pour la construction d'applications graphiques interactives*. Thèse de doctorat, Université de Paris-Sud, Orsay (France), janvier 1996.

[FIGUEROA *et al.* 02] Pablo FIGUEROA, Mark GREEN, et H. James HOOVER. « InTml : a description language for VR applications ». Dans *Proceeding of the seventh international conference on 3D Web technology*, pages 53–58. ACM Press, 2002.

[FINGERWORKS 03] « iGesture multitouch Pad ». FingerWorks, 2003. Brevet US. 6323846. `http://www.fingerworks.com/`.

[FITZMAURICE & BUXTON 97] George FITZMAURICE et William BUXTON. « An empirical evaluation of graspable user interfaces : Towards specialized, space-multiplexed input ». Dans *Proceedings of ACM CHI 97 Conference on Human Factors in Computing Systems*, volume 1 de *PAPERS : Handy User Interfaces*, pages 43–50, 1997.

[FITZMAURICE *et al.* 95] George W. FITZMAURICE, Hiroshi ISHII, et William BUXTON. « Bricks : Laying the foundations for graspable user interfaces ». Dans Irvin R. KATZ, Robert MACK, Linn MARKS, Mary Beth ROSSON, et Jakob NIELSEN, éditeurs, *Proceedings of the Conference on Human Factors in Computing Systems (CHI'95)*, pages 442–449, New York, NY, USA, mai 1995. ACM Press.

[FOLEY *et al.* 90] J. FOLEY, A. VAN DAM, S. FEINER, et J. HUGHES. *Computer graphics Principles and Practice*. Addison-Wesley, 2e édition, 1990.

[FOLEY *et al.* 84] James D. FOLEY, Victor L. WALLACE, et Peggy CHAN. « The human factors of computer graphics interaction techniques ». *IEEE Computer Graphics and Applications*, 4(11) :13–48, novembre 1984.

[FOWLER 99] A. FOWLER. « A Swing architecture overview : The inside story on JFC component design ». Sun Microsystems, 1999. `http://java.sun.com/products/jfc/tsc/articles/architecture/`.

[FRANTZ 00] Gérard FRANTZ. *Visual Basic 6 : Le guide du programmeur*. La référence. Osman Eyrolles Multimédia, novembre 2000. ISBN 2746402343.

[GILL 02] Lisa GILL. « The secret of logitech's success ». oct 2002. E-Commerce Times. `http://www.ecommercetimes.com/perl/story/19664.html`.

[GOLDBERG & ROBSON 81] A. GOLDBERG et D. ROBSON. « The Smalltalk-80 system ». *Byte Magazine*, 6(8) :36–48, août 1981.

[GREENBERG & BOYLE 02] Saul GREENBERG et Michael BOYLE. « Customizable physical interfaces for interacting with conventional applications ». Dans *Proceedings of the 15th annual ACM symposium on User interface software and technology (UIST-02)*, pages 31–40, New York, octobre 27–30 2002. ACM Press.

[GREENBERG & FITCHETT 01] Saul GREENBERG et Chester FITCHETT. « Phidgets : Easy development of physical interfaces through physical widgets ». Dans *Proceedings of the 14th Annual Symposium on User Interface Software and Technology (UIST-01)*, pages 209–218, New York, novembre 11–14 2001. ACM Press.

[GSA 91] GSA. *Managing end user computing for users with disabilities*. General Services Administration, 1991. Washington, DC.

[GUIARD 87] Yves GUIARD. « Asymmetric division of labor in human skilled bimanual action : The kinematic chain as a model. ». *The Journal of Motor Behavior*, 19(4) :486–517, 1987.

[HALBWACHS *et al.* 91] N. HALBWACHS, P. CASPI, P. RAYMOND, et D. PILAUD. « The synchronous data flow programming language LUSTRE ». Dans *Proceedings of the IEEE*, volume 79, septembre 1991.

[HAPTEK 03] « PeoplePutty : dynamic 3-D character building tool ». Haptek, Inc., 2003. `http://www.haptek.com/`.

[HAREL 87] David HAREL. « Statecharts : A visual formalism for complex systems ». *Science of Computer Programming*, 8(3) :231–274, juin 1987.

[HEWITT 84] W. T. HEWITT. « PHIGS — Programmer's Hierarchical Interactive Graphics System ». *Computer Graphics Forum*, 3(4) :299–300, décembre 1984.

[HINCKLEY *et al.* 98] Ken HINCKLEY, Mary CZERWINSKI, et Mike SINCLAIR. « Interaction and modeling techniques for desktop two-handed input ». Dans *Proceedings of the 11th Annual Symposium on User Interface Software and Technologie (UIST-98)*, pages 49–58, New York, novembre 1–4 1998. ACM Press.

[HINCKLEY *et al.* 03] Ken HINCKLEY, R.J.K. JACOB, et C. WARE. *Computer Science and Engineering Handbook Second Edition (in press)*, Chapitre Input/Output Devices and Interaction Techniques. CRC Press, 2e édition, 2003.

[HINCKLEY *et al.* 94A] Ken HINCKLEY, R. PAUSCH, J. C. GOBLE, et N. F. KASSELL. « Passive real-world interface props for neurosurgical visualization ». *Proceedings of CHI '94*, pages 452–458, 1994.

[HINCKLEY *et al.* 94B] Ken HINCKLEY, Randy PAUSCH, John C. GOBLE, et Neal F. KASSELL. « A survey of design issues in spatial input ». Dans *ACM UIST'94 Symp. on User Interface Software & Technology*, pages 213–222. 1994.

[HONEYWELL 99] Steve HONEYWELL. *Quake III Arena : Prima's Official Strategy Guide*. Prima Lifestyles, dec 1999. ISBN 0761525882.

[HONG & LANDAY 00] Jason I. HONG et James A. LANDAY. « SATIN : a toolkit for informal ink-based applications ». Dans *Proceedings of the 13th Annual Symposium on User Interface Software and Technology (UIST-00)*, pages 63–72, N.Y., novembre 5–8 2000. ACM Press.

[HOURCADE & BEDERSON 99] Juan Pablo HOURCADE et Benjamin B. BEDERSON. « Architecture and implementation of a java package for multiple input devices (MID) ». Rapport interne CS-TR-4018, University of Maryland, College Park, mai 1999.

[HOURIZI & JOHNSON 01] R. HOURIZI et P. JOHNSON. « Beyond mode error : Supporting strategic knowledge structures to enhance cockpit safety ». Dans A. BLANDFORD, J. VANDERDONKT, et P. GRAY, éditeurs, *People and Computers XV Interaction without Frontiers : Joint proceedings of IHM 2001 and HCI 2001*, 2001.

[HUDSON 90] Scott E. HUDSON. « Adaptive semantic snapping - a technique for semantic feedback at the lexical level ». Dans *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 65–70. ACM Press, 1990.

[HUDSON *et al.* 97] Scott E. HUDSON, Roy RODENSTEIN, et Ian SMITH. « Debugging lenses : A new class of transparent tools for user interface debugging ». Dans *Proceedings of the ACM Symposium on User Interface Software and Technology*, Making Things Visible, pages 179–187, 1997.

[HUDSON & SMITH 96A] Scott E. HUDSON et Ian SMITH. « SubArctic UI Toolkit user's manual st. Paul release (beta version 0.8e) ». septembre 1996. `http://www.cc.gatech.edu/gvu/ui/sub_arctic/`. GA 30332-0280.

[HUDSON & SMITH 96B] Scott E. HUDSON et Ian SMITH. « Ultra-lightweight constraints ». Dans *Proceedings of the 9th annual ACM symposium on User interface software and technology*, pages 147–155. ACM Press, 1996.

[HUDSON & STASKO 93] Scott E. HUDSON et John T. STASKO. « Animation support in a user interface toolkit : flexible, robust, and reusable abstractions ». Dans *Proceedings of the 6th annual ACM symposium on User interface software and technology*, pages 57–67. ACM Press, 1993.

[HUOT 03] Stéphane HUOT. « Page Web du projet MagLite ». École des Mines de Nantes, équipe CMI, 2003. `http://www.emn.fr/x-info/gina/maglite`.

[HUOT *et al.* 03] Stéphane HUOT, Cédric DUMAS, et Gérard HÉGRON. « Toward creative 3D modeling : an architects' sketches study ». Dans *Proceedings of the 9th IFIP TC13 International Conference on Human-Computer Interaction (INTERACT'03)*, septembre 2003.

[HUTCHINS *et al.* 86] E. L. HUTCHINS, J. D. HOLLAN, et D. A. NORMAN. « Direct manipulation interfaces ». Dans NORMAN, D.A. et S. W. DRAPER, éditeurs, *User Centered System Design : New Perspectives on Human-Computer Interaction*, pages 87–124. Lawrence Erlbaum Associates, Hillsdale, NJ and London, 1986.

[INGALLS *et al.* 88] Dan INGALLS, Scott WALLACE, Yu-Ying CHOW, Frank LUDOLPH, et Ken DOYLE. « Fabrik : A visual programming environment ». Dans Norman MEYROWITZ, éditeur, *OOPSLA'88 : Object-Oriented Programming Systems, Languages and Applications : Conference Proceedings*, pages 176–190, 1988.

[ISHII & ULLMER 97] Hiroshi ISHII et Brygg ULLMER. « Tangible bits : Towards seamless interfaces between people, bits and atoms ». Dans *Proceedings of ACM CHI 97 Conference on Human Factors in Computing Systems*, volume 1 de *PAPERS : Beyond the Desktop*, pages 234–241, 1997.

[ISO 85] *The Graphical Kernel System (GKS) : ISO 7942*. International Organization for Standardization, Geneva, Switzerland, 1985.

[ISO 87] « LOTOS - language of temporal ordering specification ». Rapport Technique ISO DP 8807, 1987.

[JACOB 96] Robert J. K. JACOB. « Human-computer interaction : Input devices ». *ACM Computing Surveys*, 28(1) :177–179, mars 1996.

[JACOB *et al.* 99] Robert J. K. JACOB, Leonidas DELIGIANNIDIS, et Stephen MORRISON. « A software model and specification language for non-WIMP user interfaces ». *ACM Transactions on Computer-Human Interaction*, 6(1) :1–46, mars 1999.

[JACOB *et al.* 94] Robert J. K. JACOB, Linda E. SIBERT, Daniel C. MCFARLANE, et M. Preston MULLEN JR.. « Integrality and separability of input devices ». *ACM Transactions on Computer-Human Interaction*, 1(1) :3–26, mars 1994.

[JENSEN 95] Kurt JENSEN. *Coloured Petri Nets : Basic Concepts, Analysis Methods and Practical Use, vol. 2*. EATCS Monographs on Theoretical Computer Science. Springer Verlag, 1995. ISBN 3-540-58276-2.

[JUSB 03] « jusb : Java USB ». Projet open source, 2003. `http://jusb.sourceforge.net/`.

[KANSY 85] K. KANSY. « 3D extension to GKS ». *Computers and Graphics*, 9(3) :267–273, 1985.

[KNEP *et al.* 95] Brian KNEP, Craig HAYES, Rick SAYRE, et Tom WILLIAMS. « Dinosaur input device ». Dans Irvin R. KATZ, Robert MACK, Linn MARKS, Mary Beth ROSSON, et Jakob NIELSEN, éditeurs, *Proceedings of the Conference on Human Factors in Computing Systems (CHI'95)*, pages 304–309, New York, NY, USA, mai 1995. ACM Press.

[KRASNER & POPE 88] G. E. KRASNER et S. T. POPE. « A cookbook for using the model-view-controller user interface paradigm in Smalltalk-80 ». *Journal of Object Oriented Programming*, 1(3) :26–49, août/septembre 1988.

[KURTENBACH & BUXTON 94] Gordon KURTENBACH et William BUXTON. « User learning and performance with marking menus ». Dans Beth ADELSON, Susan DUMAIS, et Judith OLSON, éditeurs, *Proceedings of the Conference on Human Factors in Computing Systems*, pages 258–264, New York, NY, USA, avril 1994. ACM Press.

[KURTENBACH *et al.* 97] Gordon KURTENBACH, George FITZMAURICE, Thomas BAUDEL, et Bill BUXTON. « The design of a GUI paradigm based on tablets, two-hands, and transparency ». Dans *Proceedings of ACM CHI 97 Conference on Human Factors in Computing Systems*, volume 1 de *PAPERS : Handy User Interfaces*, pages 35–42, 1997.

[LACOMME *et al.* 03] Philippe LACOMME, Christian PRINS, et Marc SEVAUX. *Algorithmes de graphes*. Eyrolles, 2e édition, octobre 2003. ISBN 2212113854.

[LANDAY & MYERS 95] James A. LANDAY et Brad A. MYERS. « Interactive sketching for the early stages of user interface design ». Dans Irvin R. KATZ, Robert MACK, Linn MARKS, Mary Beth ROSSON, et Jakob

NIELSEN, éditeurs, *Proceedings of the Conference on Human Factors in Computing Systems (CHI'95)*, pages 43–50, New York, NY, USA, mai 1995. ACM Press.

[LINTON *et al.* 89] Mark A. LINTON, John M. VLISSIDES, et Paul R. CALDER. « Composing user interfaces with InterViews ». *IEEE Computer*, 22(2), Febuary 1989.

[LIPSCOMB & PIQUE 93] James S. LIPSCOMB et Michael E. PIQUE. « Analog input device physical characteristics ». *ACM SIGCHI Bulletin*, 25(3) :40–45, 1993.

[MACKAY 96] Wendy E. MACKAY. « Réalité augmentée : le meilleur des deux mondes ». *La Recherche, numéro spécial 'L'ordinateur au doigt et à l'œil'*, 284, mars 1996.

[MACKAY *et al.* 98] Wendy E. MACKAY, Anne-Laure FAYARD, Laurent FROBERT, et Lionel MEDINI. « Reinventing the familiar : Exploring an augmented reality design space for air traffic control ». Dans *Proceedings of ACM CHI 98 Conference on Human Factors in Computing Systems*, volume 1 de *Computer Augmented Environments*, pages 558–565, 1998.

[MACKENZIE & ZHANG 97] I. Scott MACKENZIE et Shawn ZHANG. « The immediate usability of Graffiti ». Dans *Graphics Interface '97*, pages 129–137, mai 1997.

[MALONEY *et al.* 89] John H. MALONEY, Alan BORNING, et Bjorn N. FREEMAN-BENSON. « Constraint technology for user-interface construction in ThingLab II ». Dans Norman MEYROWITZ, éditeur, *OOPSLA'89 Conference Proceedings : Object-Oriented Programming : Systems, Languages, and Applications*, pages 381–388. ACM Press, 1989.

[MANKOFF *et al.* 00] Jennifer MANKOFF, Scott E. HUDSON, et Gregory D. ABOWD. « Providing integrated toolkit-level support for ambiguity in recognition-based interfaces ». Dans Thea TURNER, Gerd SZWILLUS, Mary CZERWINSKI, et Paternò FABIO, éditeurs, *Proceedings of the 2000 Conference on Human Factors in Computing Systems (CHI-00)*, pages 368–375, N. Y., avril 1–6 2000. ACM Press.

[MANN 96] Steve MANN. « 'smart clothing' : Wearable multimedia computing and 'personal imaging' to restore the technological balance between people and their environments ». Dans *Proceedings of the Fourth ACM Multimedia Conference (MULTIMEDIA'96)*, pages 163–174, New York, NY, USA, novembre 1996. ACM Press.

[MARSAN *et al.* 95] M. Ajmone MARSAN, G. BALBO, G. CONTE, S. DONATELLI, et G. FRANCESCHINIS. *Modelling with Generalized Stochastic Petri Nets*. Wiley Series in Parallel Computing. John Wiley and Sons, 1995. ISBN 471 93059 8.

[MAXIMILIEN *et al.* 01] Michael MAXIMILIEN, Boyd DIMMOCK, Dan STREETMAN, Brian WEISCHEDEL, Paul KLISSNER, Sunil DUSANKAR, Ron KLEINMAN, Harry MCKINLAY, WINCOR-NIXDORF, Peter DUELLINGS, Roger LINDSJÖ, Steve TURNER, Paul GAY, et Boris DAINSON. « Java API for USB (javax.usb), JSR-80 specification v0.9.0 ». avril 2001. `http://javax-usb.org/`.

[MCCORMACK & ASENTE 88] Joel MCCORMACK et Paul ASENTE. « Using the X toolkit or how to write a widget ». Dans *Proceedings of the USENIX Summer Conference*, pages 1–14, Berkeley, CA, USA, juin 1988. USENIX Association.

[MCDANIEL & MYERS 98] Richard G. MCDANIEL et Brad A. MYERS. « Building applications using only demonstration ». Dans *Proceedings of the 3rd international conference on Intelligent user interfaces*, pages 109–116, 1998.

[MEALY 55] George H. MEALY. « A method for synthesizing sequential circuits ». *Bell System Technical Journal*, 34(5) :1045–1079, 1955.

[MERTZ *et al.* 00] C. MERTZ, J.L. VINOT, et D. ETIENNE. « Entre interaction directe et reconnaissance d'écriture : les gestes écologiques ». Dans *ERGO-IHM 2000*, pages 170–177, Biarritz, France, octobre 2000. CRT ILS & ESTIA.

[MICROÏDS 02] « Syberia ». Microïds, 2002. `http://www.microids.com`, `http://www.syberia.info/`.

[MICROSOFT 03A] « DirectInput C/C++ reference ». Microsoft Corporation, 2003. `http://msdn.microsoft.com/`.

[MICROSOFT 03B] « Transcriber ». Microsoft Corporation, 2003. `http://www.microsoft.com/windowsmobile/resources/downloads/pocketpc/transcriber.mspx`.

[MILGRAM & KISHINO 94] P. MILGRAM et F. KISHINO. « A taxonomy of mixed reality visual displays ». *IEICE Transactions on Information Systems*, E77-D(12), dec 1994.

[MODUGNO 93] Francesmary MODUGNO. « PURSUIT : Programming in the user interface ». Dans *Proceedings of ACM INTERCHI'93 Conference on Human Factors in Computing Systems – Adjunct Proceedings*, Doctoral Consortium, page 217, 1993.

[MOORE 56] E. F. MOORE. « Gedanken-experiments on sequential machines ». Rapport Technique, 1956.

[MOZILLA 03] « The optimoz project ». The Mozilla Organization, 2003. `http://optimoz.mozdev.org/`.

[MURAKAMI *et al.* 95] Tamostsu MURAKAMI, Kazuhiko HAYASHI, Kazuhiro OIKAWA, et Naomasa NAKAJIMA. « DO-IT : Deformable objects as input tools ». Dans *Proceedings of ACM CHI'95 Conference on Human Factors in Computing Systems*, volume 2 de *Interactive Experience*, pages 87–88, 1995.

[MYERS 90] Brad A. MYERS. « A new model for handling input ». *ACM Transactions on Information Systems*, 8(3) :289–320, juillet 1990.

[MYERS 92] Brad A. MYERS. « Demonstrational interfaces : A step beyond direct manipulation ». *Computer*, 25(8) :61–73, août 1992.

[MYERS 93] Brad A. MYERS. « Why are human-computer interfaces difficult to design and Implement ? ». Rapport Technique CMU-CS-93-183, Carnegie Mellon University, School of Computer Science, juillet 1993. `ftp://reports.adm.cs.cmu.edu/usr/anon/1993/CMU-CS-93-183.ps`.

[MYERS 96] Brad A. MYERS. « The Amulet v2.0 reference manual ». Carnegie Mellon University Computer Science Department, février 1996. `http://www.cs.cmu.edu/~amulet`.

[MYERS 98A] Brad A. MYERS. « A brief history of human-computer interaction technology ». *interactions*, 5(2) :44–54, 1998.

[MYERS 98B] Brad A. MYERS. « Scripting graphical applications by demonstration ». Dans *Proceedings of ACM CHI 98 Conference on Human Factors in Computing Systems*, volume 1 de *Software Behind the Scenes*, pages 534–541, 1998.

[MYERS & KOSBIE 96] Brad A. MYERS et David S. KOSBIE. « Reusable hierarchical command objects ». Dans Michael J. TAUBER, Victoria BELLOTTI, Robin JEFFRIES, Jock D. MACKINLAY, et Jakob NIELSEN, éditeurs, *Proceedings of the Conference on Human Factors in Computing Systems : Commun Ground*, pages 260–267, New York, avril 13–18 1996. ACM Press.

[MYERS *et al.* 93] Brad A. MYERS, Richard G. MCDANIEL, et David S. KOSBIE. « Marquise : Creating complete user interfaces by demonstration ». Dans *Proceedings of ACM INTERCHI'93 Conference on Human Factors in Computing Systems*, Demonstration Based Systems, pages 293–300, 1993.

[MYERS *et al.* 97] Brad A. MYERS, Richard G. MCDANIEL, Robert C. MILLER, Alan S. FERRENCY, Andrew FAULRING, Bruce D. KYLE, Andrew MICKISH, Alex KLIMOVITSKI, et Patrick DOANE. « The Amulet environment : New models for effective user interface software development ». *IEEE Transactions on Software Engineering*, 23(6) :347–365, juin 1997.

[MYNATT *et al.* 99] Elizabeth D. MYNATT, W. Keith EDWARDS, Anthony LAMARCA, et Takeo IGARASHI. « Flatland : New dimensions in office whiteboards ». Dans Marian G. WILLIAMS, Mark W. ALTOM, Kate EHRLICH, et William NEWMAN, éditeurs, *Proceedings of the Conference on Human Factors in Computing Systems (CHI-99)*, pages 346–353, New York, mai 15–20 1999. ACM Press.

[NG *et al.* 98] Elizabeth NG, Tim BELL, et Andy COCKBURN. « Improvements to a pen-based musical input system ». Dans *OzCHI'98 : The Australian Conference on Computer-Human Interaction*, pages 239–252. IEEE Press, nov 1998.

[NIGAY & COUTAZ 93] Laurence NIGAY et Joelle COUTAZ. « A design space for multimodal systems : Concurrent processing and data fusion ». Dans *Proceedings of ACM INTERCHI'93 Conference on Human Factors in Computing Systems*, Voices and Faces, pages 172–178, 1993.

[NIGAY & COUTAZ 95] Laurence NIGAY et Joelle COUTAZ. « A generic platform for addressing the multimodal challenge ». Dans *Proceedings of ACM CHI'95 Conference on Human Factors in Computing Systems*, volume 1 de *Papers : Multimodal Interfaces*, pages 98–105, 1995.

[NORMAN 81] D. A. NORMAN. « Categorization of action slips ». *Psychological Review*, 1(88) :1–15, 1981.

[NORMAN 02] Donald A. NORMAN. *The Design of Everyday Things*. Basic Books, sep 2002. ISBN 0465067107.

[NORMAN & DRAPER 86] D. A. NORMAN et St.W. DRAPER. *User Centered System design*. Lawrence Earlbaum Assoc., Hillsdale, 1986. ISBN 0-89859-781.

[OPERA 03] « Mouse gestures in opera ». Opera Software ASA, 2003. `http://www.opera.com/features/mouse/`.

[ORR & ABOWD 00] Robert J. ORR et Gregory D. ABOWD. « The smart floor : a mechanism for natural user identification and tracking ». Dans *CHI '00 extended abstracts on Human factors in computer systems*, pages 275–276. ACM Press, 2000.

[PALANQUE & BASTIDE 93] Philippe PALANQUE et Rémi BASTIDE. « Interactive Cooperative Objects : an Object-Oriented Formalism Based on Petri Nets for User Interface Design ». Dans *IEEE / System Man and Cybernetics 93*, pages 274–285. Elsevier Science Publisher, octobre 1993.

[PALANQUE & BASTIDE 97] Philippe PALANQUE et Rémi BASTIDE. « Synergistic modelling of tasks, users and systems using formal specification techniques ». *Interacting with Computers*, 9(2) :129–153, 1997.

[PALAY *et al.* 88] Andrew J. PALAY, Wilfred J. HANSEN, Mark SHERMAN, Maria G. WADLOW, Thomas P. NEUENDORFFER, Zalman STERN, Miles BADER, et Thom PETERS. « The Andrew Toolkit — an overview ». Dans USENIX ASSOCIATION, éditeur, *EUUG Conference Proceedings, Spring, 1988. London, England*, pages 311– ??, Buntingford, Herts, UK, Spring 1988. EUUG.

[PARADIGM 03] « Vega prime ». Multigen-Paradigm, Inc., 2003. `http://www.paradigmsim.com`.

[PARAGON 03] « Penreader ». Paragon Software, 2003. `http://www.penreader.com`.

[PARTRIDGE *et al.* 02] Kurt PARTRIDGE, Saurav CHATTERJEE, Vibha SAZAWAL, Gaetano BORRIELLO, et Roy WANT. « TiltType : accelerometer-supported text entry for very small devices ». Dans *Proceedings of the 15th annual ACM symposium on User interface software and technology (UIST-02)*, pages 201–204, New York, octobre 27–30 2002. ACM Press.

[PATERNÒ & FACONTI 92] F. PATERNÒ et G. FACONTI. « On the use of LOTOS to describe graphical interaction ». Dans *Proceedings of the HCI'92 Conference on People and Computers VII*, Graphics – Design and Techniques, pages 155–173, 1992.

[PATRICK & SACHS 94] Mark PATRICK et George SACHS. « X11 input extension library specification ». X Consortium Standard, X11R6, avril 1994.

[PEARSON & WEISER 86] Glenn PEARSON et Mark WEISER. « Of moles and men : The design of foot controls for workstations ». Dans *Proceedings of ACM CHI'86 Conference on Human Factors in Computing Systems*, Haptic Techniques, pages 333–339, 1986.

[PERLIN 98] Ken PERLIN. « Quikwriting : Continuous stylus-based text entry ». Dans *Proceedings of the ACM Symposium on User Interface Software and Technology*, Fast Pen Input, pages 215–216, 1998.

[PETRI 62] C. A. PETRI. « Fundamentals of a theory of asynchronous information flow ». *Proc. IFIP Congress, N-H*, 1962.

[PFAFF 85] G. E. PFAFF, éditeur. *User Interface Management Systems : Proceedings of the Seeheim Workshop*, Berlin, 1985. Springer-Verlag. proceedings of the Workshop on User Interface Management Systems, held in Seeheim, FRG, November 1-3, 1983.

[PIEPER & KOBSA 99] Michael PIEPER et Alfred KOBSA. « Talking to the ceiling : an interface for bed-ridden manually impaired users ». Dans *CHI '99 extended abstracts on Human factors in computer systems*, pages 9–10. ACM Press, 1999.

[PIPER *et al.* 02] Ben PIPER, Carlo RATTI, et Hiroshi ISHII. « Illuminating clay : a 3-D tangible interface for landscape analysis ». Dans Loren TERVEEN, Dennis WIXON, Elizabeth COMSTOCK, et Angela SASSE, éditeurs, *Proceedings of the CHI 2002 Conference on Human Factors in Computing Systems (CHI-02)*, pages 355–362, New York, avril 20–25 2002. ACM Press.

[POLLER & GARTER 84] M.F. POLLER et S. K. GARTER. « The effects of modes on text editing by experienced editor users ». *Human Factors*, 26(4) :449–462, 1984.

[POUPYREV *et al.* 96] Ivan POUPYREV, Mark BILLINGHURST, Suzanne WEGHORST, et Tadao ICHIKAWA. « The go-go interaction technique : Non-linear mapping for direct manipulation in VR ». Dans *Proceedings of the ACM Symposium on User Interface Software and Technology*, Papers : Virtual Reality (TechNote), pages 79–80, 1996.

[POYNER 96] Rick POYNER. « Wintab interface specification 1.1 : 16- and 32-bit API reference ». LCS/Telegraphics, mai 1996. `http://www.pointing.com`.

[PREECE *et al.* 94] Jenny PREECE, Yvonne ROGERS, Helen SHARP, David BENYON, Simon HOLLAND, et Tom CAREY. *Human-Computer Interaction*. Addison-Wesley Publishing, Reading, Mass., 1994. ISBN 0-201-62769-8. OCLC 35598754.

[RAISAMO & RÄIHÄ 96] Roope RAISAMO et Kari-Jouko RÄIHÄ. « A new direct manipulation technique for aligning objects in drawing programs ». Dans *Proceedings of the Ninth Annual Symposium on User Interface Software and Technogy*, pages 157–164, New York, novembre 6–8 1996. ACM Press.

[REITMAYR & SCHMALSTIEG 01] Gerhard REITMAYR et Dieter SCHMALSTIEG. « An open software architecture for virtual reality interaction ». Dans Chris SHAW et Wenping WANG, éditeurs, *Proceedings of the ACM Symposium on Virtual Reality Software and Technology (VRST-01)*, pages 47–54, New York, novembre 15–17 2001. ACM Press.

[REKIMOTO 96] Jun REKIMOTO. « Tilting operations for small screen interfaces ». Dans *Proceedings of the Ninth Annual Symposium on User Interface Software and Technogy*, pages 167–168, New York, novembre 6–8 1996. ACM Press.

[REKIMOTO 02] Jun REKIMOTO. « SmartSkin : an infrastructure for freehand manipulation on interactive surfaces ». Dans Loren TERVEEN, Dennis WIXON, Elizabeth COMSTOCK, et Angela SASSE, éditeurs, *Proceedings of the CHI 2002 Conference on Human Factors in Computing Systems (CHI-02)*, pages 113–120, New York, avril 20–25 2002. ACM Press.

[ROOSENDAAL & WARTMANN 03] Ton ROOSENDAAL et Carsten WARTMANN. *The Official Blender Gamekit : Interactive 3-D for Artists*. No Starch Press, mars 2003. ISBN 1593270046.

[ROUSSEL 02] Nicolas ROUSSEL. « Videoworkspace : une boîte à outils pour l'exploration de nouvelles techniques de gestion de fenêtres ». Dans *Proceedings of the 14th French-speaking conference on Human-computer interaction (Conférence Francophone sur l'Interaction Homme-Machine)*, pages 271–274. ACM Press, 2002.

[SALBER *et al.* 99] Daniel SALBER, Anind K. DEY, et Gregory D. ABOWD. « The context toolkit : Aiding the development of context-enabled applications ». Dans Marian G. WILLIAMS, Mark W. ALTOM, Kate EHRLICH, et William NEWMAN, éditeurs, *Proceedings of the Conference on Human Factors in Computing Systems (CHI-99)*, pages 434–441, New York, mai 15–20 1999. ACM Press.

[SALEM & ZHAI 97] Chris SALEM et Shumin ZHAI. « An isometric tongue pointing device ». Dans *Proceedings of ACM CHI 97 Conference on Human Factors in Computing Systems*, volume 1 de *TECHNICAL NOTES : Input & Output in the Future*, pages 538–539, 1997.

[SANTORI 90] Michael SANTORI. « An instrument that isn't really ». *IEEE Spectrum*, 27(8) :36–39, août 1990.

[SCHMUCKER 86] K. SCHMUCKER. « MacApp : an application framework ». *Byte Magazine*, 11(8) :189–193, août 1986.

[SCHWERDTFEGER 00] Richard S. SCHWERDTFEGER. « IBM guidelines for writing accessible applications using 100% pure Java ». IBM Accessibility Center, août 2000. `http://www-3.ibm.com/able/guidelines/java/snsjavag.html`.

[SCHYN *et al.* 03] Amélie SCHYN, David NAVARRE, et Philippe PALANQUE. « Description formelle d'une technique d'interaction multimodale dans une application de réalité virtuelle immersive ». Dans *Actes de la 15ème conférence francophone d'Interaction Homme-Machine (IHM'2003)*, novembre 25–28 2003.

[SENSE8 03] « WorldUp and WorldToolkit : Virtual reality support software ». Sense8 Corp, 2003. `http://www.sense8.com`.

[SHNEIDERMAN 83] Ben SHNEIDERMAN. « Direct manipulation : A step beyond programming languages ». *IEEE Computer*, 16(8) :57–69, août 1983.

[SHNEIDERMAN 98] Ben SHNEIDERMAN. *Designing the User Interface*. Addison Wesley Longman, 3e édition, 1998.

[SMITH 88] David N. SMITH. « Building interfaces interactively ». Dans *Proceedings of the ACM SIGGRAPH Symposium on User Interface Software*, pages 144–151, 1988.

[SONY 03] « Eye toy ». Sony Computer Entertainment, Inc., 2003. `http://www.eyetoy.com`.

[STARNER & PENTLAND 95] Thad STARNER et Alex PENTLAND. « Visual recognition of american sign language using hidden markov models ». Dans *International Workshop on Automatic Face and Gesture Recognition*, 1995.

[STERIADIS & CONSTANTINOU 03] Constantine E. STERIADIS et Philip CONSTANTINOU. « Designing human-computer interfaces for quadriplegic people ». *ACM Transactions on Computer-Human Interaction*, 10(2) :87–118, 2003.

[STORK & HENNECKE 96] D.G. STORK et M.E. HENNECKE. « Speechreading : An overview of image processing, feature extraction, sensory integration and pattern recognition techniques ». Dans *Proceedings of the 2nd International Conference on Automatic Face and Gesture Recognition (FG 96)*, pages xvi–xxvi. IEEE, oct 1996.

[STRAUSS & CAREY 92] Paul S. STRAUSS et Rikk CAREY. « An object-oriented 3D graphics toolkit ». volume 26, pages 341–349, juillet 1992.

[SUN 98] « Java speech API specification v1.0 ». Sun Microsystems, Inc., 1998. `http://java.sun.com/products/java-media/speech/`.

[SUN 02] « The java accessibility API ». Sun Microsystems, Inc., 2002. `http://www.sun.com/access/`.

[TACTEX 03] « MWK-02 : True multi-touch pressure sensing pad ». Tactex Controls, Inc., 2003. Brevet CA. 2273113. `http://www.tactex.com/`.

[TYSON R. *et al.* 90] Henry TYSON R., Hudson SCOTT E., et Newell GARY L.. « Integrating gesture and snapping into a user interface toolkit ». Dans *Proceedings of the 3rd annual ACM SIGGRAPH symposium on User interface software and technology*, pages 112–122. ACM Press, 1990.

[UIMS 92] « A metamodel for the runtime architecture of an interactive system ». *ACM SIGCHI Bulletin*, 24(1) :32–37, 1992.

[USB/HID 01] « Universal Serial Bus (USB) device class definition for Human Interface Devices (HID). Firmware specification version 1.11 ». Rapport Technique, juin 2001. `http://www.usb.org/developers/hidpage/`.

[VAN DAM 88] Andries VAN DAM. « PHIGS+ functional description revision ». *Computer Graphics*, 22(3) :125–220, juillet 1988.

[VAN KAMPEN 00] Kurtis J. VAN KAMPEN. « The interface between humans and interactive kiosks ». Input Technologies, LLC, 2000. `http://www.visi.com/~keefner/pdfs/focus1.pdf`.

[VIRTOOLS 01] « Virtools dev ». Virtools SA, 2001. `http://www.virtools.com/`.

[WACOM 03] « Wacom pentools 3.1 ». Wacom Technology Co, dec 2003. `http://www.wacom.com/pentools/`.

[WEIMER & GANAPATHY 89] D. WEIMER et S. K. GANAPATHY. « A synthetic visual environment with hand gesturing and voice input ». Dans *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 235–240. ACM Press, 1989.

[WEISER 91] Mark WEISER. « The computer for the 21st century ». *Scientific American*, pages 95–104, septembre 1991.

[WERTH & MYERS 93] Andrew J. WERTH et Brad A. MYERS. « Tourmaline : Macrostyles by example ». Dans Stacey ASHLUND, Ken MULLET, Austin HENDERSON, Erik HOLLNAGEL, et Ted WHITE, éditeurs, *Proceedings of the Conference on Human Factors in computing systems*, pages 532–532, New York, avril 24–29 1993. ACM Press.

[WORDSPLUS 03] « EZ Keys user manual ». Words+, Inc., 2003. `http://www.words-plus.com`.

[YANG *et al.* 98] Jie YANG, Rainer STIEFELHAGEN, Uwe MEIER, et Alex WAIBEL. « Visual tracking for multimodal human computer interaction ». Dans *Proceedings of the Conference on Human Factors in Computing Systems (CHI-98) : Making the Impossible Possible*, pages 140–147, New York, avril 18–23 1998. ACM Press.

[ZANDEN & MYERS 95] Brad T. Vander ZANDEN et Brad A. MYERS. « Demonstrational and constraint-based techniques for pictorially specifying application objects and boundaries ». *ACM Transactions on Computer-Human Interaction*, 2(4) :308–365, décembre 1995.

[ZELEZNIK *et al.* 91] Robert C. ZELEZNIK, D. Brookshire CONNER, Matthias M. WLOKA, Daniel G. ALIAGA, Nathan T. HUANG, Philip M. HUBBARD, Brian KNEP, Henry KAUFMAN, John F. HUGHES, et Andries van DAM. « An object-oriented framework for the integration of interactive animation techniques ». *Computer Graphics*, 25(4) :105–112, juillet 1991.

[ZHAI 98] Shumin ZHAI. « User performance in relation to 3D input device design ». *Computer Graphics*, 32(4) :50–54, novembre 1998.

UN MODÈLE D'INTERACTION EN ENTRÉE POUR DES SYSTÈMES INTERACTIFS MULTI-DISPOSITIFS HAUTEMENT CONFIGURABLES

Pierre DRAGICEVIC
Doctorat de l'Université de NANTES

Résumé

Aujourd'hui, les applications comme les outils de développement demeurent câblés pour une utilisation exclusive et stéréotypée du clavier et de la souris, et ignorent tout autre paradigme d'interaction. Bien qu'avérés, les problèmes liés à l'adaptabilité en entrée ont encore fait l'objet de très peu d'études. Dans cette thèse, nous proposons un modèle basé sur des *configurations d'entrée*, où des dispositifs d'entrée sont librement connectés aux applications à travers des adaptateurs, et où la traditionnelle file d'événements est remplacée par un paradigme à flot de données réactif. Ce modèle a donné lieu à la boîte à outils en entrées ICON (Input Configurator), qui permet de construire des applications interactives entièrement configurables en entrée, capables d'exploiter des dispositifs et des techniques d'interaction non-standard. Son éditeur visuel permet au développeur de créer rapidement des configurations pour des entrées appauvries ou enrichies, que les utilisateurs avancés peuvent ensuite adapter à leurs besoins.

Mots-clés : Interaction Homme-Machine, Périphériques d'entrée, Techniques d'interaction, Adaptabilité, Accessibilité, Interfaces Post-WIMP.

Abstract

Today's applications and tools exclusively rely on mice and keyboards and use them a stereotyped way, and are still ignoring any other interaction paradigm. Though importance of input adaptability has been widely recognized, its issues have been very little studied. We propose a new input model based on *input configurations*. In this model, input devices are freely connected to applications through adapters, and traditional event mechanisms have been replaced by a reactive data-flow paradigm. Using this model, we developed the ICON (Input Configurator) input toolkit, which allows to build interactive applications that are fully configurable and that can make use of alternative input device and techniques. With its visual editor, developers can rapidly create by direct manipulation configurations for impoverished or enriched input. Power users can then customize those configurations to suit their specific needs.

Keywords : Human-Computer Interaction, Input devices, Interaction techniques, Adaptability, Accessibility, Post-WIMP interfaces.