Une approche pour l'ingénierie des systèmes interactifs critiques multimodaux et multi-utilisateurs : application à la prochaine génération de cockpit d'aéronefs

Martin Cronel

THÈSE

En vue de l'obtention du

DOCTORAT DE L'UNIVERSITÉ DE TOULOUSE

Délivré par : *l'Université Toulouse 3 Paul Sabatier (UT3 Paul Sabatier)*

Présentée et soutenue le *18/10/2017* **par :**

Martin CRONEL

Une approche pour l'ingénierie des systèmes interactifs critiques multimodaux et multi-utilisateurs : Application à la prochaine génération de cockpit d'aéronefs

---

JURY

Kris LUYTEN	Professeur d'Université	Rapporteur
Jean VANDERDONCKT	Professeur Ordinaire	Rapporteur
Géry CASIEZ	Professeur d'Université	Membre du Jury
Bruno DUMAS	Professeur Associé	Membre du Jury
Jean-Charles FABRE	Professeur d'Université	Membre du Jury
Célia MARTINIE	Maitre de Conférences	Membre du Jury
Philippe PALANQUE	Professeur d'Université	Directeur de Thèse

École doctorale et spécialité :
MITT : Domaine STIC : Sureté de logiciel et calcul de haute performance
Unité de Recherche :
Institut de Recherche en Informatique de Toulouse (UMR 5505)
Directeur de Thèse :
Philippe PALANQUE
Rapporteurs :
Kris LUYTEN et *Jean VANDERDONCKT*

Table des matières

Je voudrais dédicacer ce manuscrit à mes jambes, qui m'ont toujours supporté, mes bras, qui ont toujours été à mes côtés, et enfin mes doigts, sur lesquels j'ai toujours pu compter.

Remerciements

Je tiens à remercier Pr Kris Luyten et Pr Jean Vanderdonkt d'avoir accepté de rapporter sur ce mémoire.

Merci à Philippe pour la perspicacité de ses remarques sur mes travaux tout au long de la thèse. Merci à Camille et Arnaud, avec qui j'ai collaboré et qui m'ont permis de rendre ce travail moins solitaire. Merci à Didier, pour son support technique au quotidien. Merci à David, pour ses conseils qui ont éclairé certains aspects de mes travaux d'un jour nouveau.

Merci à Marco pour tout ce qu'il fait au quotidien pour que l'on puisse travailler, être payé et remboursé, manger, et j'en oublie.

Merci à Camille, Célia, David, Philippe, et surtout mon père Pierre, qui ont relu mon mémoire, me permettant de l'améliorer grandement.

Merci à toute l'équipe ICS pour la bonne ambiance au travail comme en dehors, qui empêche les journées de travail de devenir ennuyeuses.

Merci à mes amis toulousains et stéphanois ainsi qu'à ma famille, Pierre Catherine et Gaëlle pour le soutien psychologique sans lequel la rédaction aurait été encore plus difficile.

Enfin merci à Enguerran, Antoine, Pierre, Josselyn & Maëlle, Anne & James et surtout David & Léna pour l'hébergement sur Toulouse pendant ces longs mois !

Résumé de la thèse

Nos travaux contribuent au domaine de l'ingénierie des systèmes interactifs multimodaux critiques. Ils facilitent l'introduction de nouveaux périphériques (comme les tablettes multi-touch, les systèmes de reconnaissance de geste...) permettant l'interaction multimodale et multi-utilisateurs au sein des futurs cockpits. Pour le moment, les méthodes et les techniques de description des IHM (Interactions Homme Machine) existantes pour la conception des cockpits ne permettent pas de prendre en compte la complexité des techniques d'interaction multimodales. De leur côté, les méthodes de conception d'IHM grand public sont incompatibles avec les exigences de fiabilité et de certification nécessaires aux systèmes critiques.

Les travaux proposent un modèle d'architecture logicielle et matérielle MIODMIT (Multiple Input Output devices Multiple Interaction Techniques) qui vise l'intégration de périphériques permettant l'usage de multimodalité au sein de systèmes critiques. Ce modèle décrit précisément les rôles de chacun des composants ainsi que les relations qu'ils entretiennent. Il couvre l'ensemble du spectre du système interactif multimodal qui va des périphériques d'entrée et leurs pilotes, vers les techniques d'interaction et l'application interactive. Il décrit aussi le rendu allant de l'application interactive aux périphériques de sortie en passant par les techniques complexes de présentation. Au-delà de sa capacité de description, ce modèle d'architecture assure la modifiabilité de la configuration du système (ajout ou suppression de périphériques au moment du design et de l'exécution).

En outre, la modélisation des systèmes fait apparaitre qu'une partie importante du comportement est autonome c'est-à-dire qu'il évolue sans recevoir d'entrées produites par l'utilisateur. Les utilisateurs peuvent avoir du mal à comprendre et à anticiper ce genre de comportement autonome, qui peut engendrer des erreurs appelées *automation surprises*. Nous proposons une méthode d'évaluation à base de modèles des techniques d'interaction permettant d'analyser pour ensuite réduire significativement les erreurs d'utilisation liées à ces comportements inattendus et incompréhensibles.

Enfin nous avons exploité le langage formel ICO (Interactive Cooperative Objects), pour décrire de façon complète et non ambiguë chacun des composants de l'architecture. Il est exploitable au moyen d'un outil d'édition et d'interprétation appelé Petshop, qui permet de faire fonctionner l'application interactive dans son ensemble (de l'entrée à la sortie). Nous avons complété cet environnement par une plateforme que nous avons appelée ARISSIM(ARINC 653 Standard SIMulator). Elle ajoute des mécanismes de sûreté de fonctionnement aux systèmes interactifs multimodaux développés avec Petshop. Plus précisément ARISSIM permet la ségrégation spatiale et la ségrégation temporelle des processus, ce qui accroît fortement la tolérance aux fautes durant l'exécution.

Nos travaux proposent un socle aux équipes de conception pluridisciplinaires (principalement ergonomes spécialistes en IHM et développeurs) d'interaction homme-machine pour les systèmes critiques destinés aux cockpits d'aéronefs de prochaine génération.

Abstract

The work of this thesis aims at contributing to the field of the engineering of interactive critical systems. We aim at easing the introduction of new input and output devices (such as touch screens, mid-air gesture recognition ...) allowing multi-user and multimodal interactions in next generation of aircraft's cockpits. Currently, development process in the aeronautical filed is not compatible with the complexity of multimodal interaction. On the other side development process of wide spread systems cannot meet the requirements of critical systems.

We introduce a generic software and hardware architecture model called MIODMIT (Multiple Input Output devices Multiple Interaction Techniques) which aim the introduction of dynamically instantiated devices, allowing multimodal interaction in critical systems. It describes the organization of information flux with a complete and non-ambiguous way. It covers the entire spectrum of multimodal interactive systems, from input devices and their drivers, to the specification of interaction techniques and the core of the application. It also covers the rendering of every software components, dealing with fission and fusion of information. Furthermore, this architecture model ensure the system configuration modifiability (i.e. add or suppress a device in design or operation phase).

Furthermore, moralizing a system reveals that an important part of the interactive part is autonomous (i.e. not driven by the user). This kind of behavior is very difficult to understand and to foresee for the users, causing errors called automation surprises. We introduce a model-based process of evaluation of the interaction techniques which decrease significantly this kind of error.

Lastly, we exploited ICO (Interactive Cooperative Objects) formalism , to describe completely and unambiguously each of the software components of MIODMIT. This language is available in an IDE (integrated development environment) called Petshop, which can execute globally the interactive application (from input/output devices to the application core). We completed this IDE with an execution platform named ARISSIM (ARINC 653 Standard SIMulator), adding safety mechanisms. More precisely, ARRISIM allows spatial segregation of processes (memory allocution to each executing partition to ensure the confinement of potential errors) and temporal segregation (sequential use of processor). Those adding increase significantly the system reliability during execution.

Our work is a base for multidisciplinary teams (more specifically ergonoms, HMI specialist and developers) which will conceive future human machine interaction in the next generation of aircraft cockpits.

Introduction

Beaucoup considèrent comme visionnaire l'article de Marc Weiser [Weiser, 1993] abordant le devenir des systèmes informatiques. Dans ce dernier, il décrit une évolution de l'informatique, et par extension des systèmes interactifs, suivant une utilisation ubiquitaire et transparente. En effet, les systèmes électroniques qui peuplaient le quotidien des générations précédentes ont été complétés par les systèmes informatiques interactifs pervasifs. Auparavant, un appareil électronique accomplissait une tâche unique et nécessitait donc peu d'interactions au-delà du démarrage ou de l'arrêt de cette tâche. Les nouveaux systèmes sont de plus en plus connectés et configurables informatiquement. Ils offrent par ailleurs de nombreuses fonctionnalités, elles mêmes accessibles via une interaction informatique.

Au cours des dernières années, la recherche en Interaction Homme-Machine s'est fortement concentrée sur l'exploration de nouvelles techniques d'interactions. Les aspects de fiabilité des systèmes interactifs ont étés délaissés, réduisant par la même leur intégration au sein des environnements critiques.

De leur côté, les systèmes critiques intègrent des interfaces de plus en plus complexes et numériques, en raison des performances accrues des opérateurs qui les exploitent. Cependant, l'ingénierie des interactions multimodales, que l'on retrouve sur les appareils tels que les smartphones n'est, aujourd'hui, pas compatible vis-à-vis des exigences de fiabilité des systèmes critiques tels que ceux implantés dans les cockpits, et ce malgré les atouts inhérents à la multimodalité [Dumas et al., 2009].

Le domaine de recherche lié à la sûreté de fonctionnement, bien qu'utilisant depuis le début du millénaire des systèmes informatiques interactifs, peine à intégrer l'utilisabilité dans les propriétés à prendre en compte lors des phases de conception des systèmes informatiques. De nombreuses normes ont été mises en place afin d'assurer la fiabilité de nombreux systèmes. Elles concernent à la fois le matériel (hors du périmètre de nos travaux), les logiciels (par exemple la CEI 61508 [CEI, c]) et leur processus de développement (par exemple la DO-178C [RTCA, a]). Néanmoins, les problèmes d'expérience utilisateur ou d'utilisabilité, au sens de la facilité d'utilisation, ont souvent été négligés, et de fait compensés par la formation intensive des opérateurs confrontés à des systèmes toujours plus complexes. Par exemple, les interfaces WIMP sont arrivées dans les avions gros porteurs en 2005 avec le déploiement des cockpits display par AIRBUS tandis qu'elles existaient dans les systèmes grand public depuis le début des années 80 (par exemple l'Apple Lisa

doté d'une interface entièrement utilisable à la souris, présenté en janvier 1983).

Si l'on considère l'augmentation de performance induite par l'utilisation de tels systèmes interactifs, on se trouve face à un "manque à gagner" de performance, autrement dit un retard d'environ 25 ans. Une comparaison intéressante est à relever : les interfaces WIMP ont été standardisées **après** leur introduction dans les systèmes grand public avec CUA d'IBM [Berry, 1988] tandis qu'AIRBUS a standardisé les interfaces **avant** leur introduction dans les avions (norme ARINC 661 dont la première version a été développée en 2001, qui spécifie l'interface entre le sous-système de visualisation et d'interaction dans un cockpit et les systèmes avions commandés).

Le différentiel, relevé en terme d'interaction, offert par les systèmes interactifs grand public, et les systèmes interactifs critiques commence à être suffisamment important pour que naissent de potentiels problèmes de sûreté de fonctionnement. Les opérateurs, imprégnés par l'usage de leurs appareils électroniques personnels (et donc habitués aux interactions naturelles), risquent d'engendrer des erreurs de manipulation sur leurs outils de travail. L'intérêt de l'introduction d'interactions multimodales dans les systèmes interactifs critiques n'étant plus à démontrer [Oviatt, 1999], les questions relatives à leur ingénierie demeurent.

Nos travaux proposent une approche intégrée qui se situe à la croisée de ces deux domaines de recherche. Ils visent à moyen ou long terme l'introduction de techniques d'interaction modernes dans les environnements critiques, en intégrant, dès leur conception, les propriétés de fiabilité et d'utilisabilité.

La conception de systèmes informatiques incombe à plusieurs types d'intervenants. En premier lieu, nous citerons les constituants de la chaine de production : les développeurs, designers, ergonomes, analystes de besoins, spécialistes en IHM, etc. En deuxième lieu se trouvent les responsables d'infrastructure : les spécialistes de réseaux, de sécurité, etc. Enfin, apparaissent les intervenants extérieurs : les clients, utilisateurs, ou toute autre partie prenante dont l'avis, l'opinion ou les besoins sont essentiels à la conception. Nous allons cibler plus particulièrement ceux qui participent à la production du système interactif.

Afin que ces différents acteurs puissent travailler ensemble et apportent leurs expertises spécifiques, nous proposons une plateforme qui leur permettra d'apporter leur pierre à l'édifice du système informatique en conception.

Notre travail spécifique participe d'une approche issue des problématiques d'ingénierie. Le socle de nos travaux réside dans un modèle d'architecture d'un système interactif critique qui permet l'intégration de périphériques offrant à l'utilisateur des interactions multimodales dans les systèmes critiques.

Nous avons travaillé sur l'évaluation de techniques d'interaction pour supprimer les fautes génériques liées aux surprises pouvant survenir lors de l'utilisation d'un système interactif.

Enfin, nous avons réfléchi à l'outillage de notre méthode, via le développement d'une plateforme respectant les contraintes d'un système critique.

Nos travaux étant centrés sur l'ingénierie, nous avons délaissé volontairement le design

des techniques d'interaction. Nous considérons donc que la conception est réalisée en amont. Cela est le cas, par exemple, dans les environnements de développement courants tels qu'Eclipse.

La structure du manuscrit est axée sur trois parties : mise en contexte et l'état de l'art, contributions et illustrations sur des exemples.

La première partie aborde le contexte, les problématiques et les travaux relatifs existants. Elle est développée au cours de deux chapitres :

Le chapitre 1 présente le contexte industriel des travaux de ce mémoire, les systèmes critiques et les contraintes liées à la sûreté de fonctionnement, ainsi que les systèmes interactifs et les problèmes que l'on peut rencontrer lors de leur ingénierie.

Le chapitre 2 s'intéresse aux problèmes liés à l'ingénierie des systèmes interactifs critiques, qui sont au cœur de ces travaux.

La deuxième partie développe en trois chapitres les contributions des travaux de cette thèse dont l'objectif est de répondre aux différents besoins identifiés lors de l'ingénierie de systèmes interactifs critiques.

Le chapitre 3 décrit un modèle d'architecture générique qui permet de définir les bases des composants d'un système interactif critique multimodal et multi-utilisateur pouvant s'intégrer dans un processus de développement de système interactif critique.

Le chapitre 4 présente une plateforme qui rend possible l'implantation et le test d'applications interactives en milieu contraint, imitant les contraintes d'un cockpit d'aéronef.

Nous verrons au cours du chapitre 5 une méthode qui identifie, dès la conception, de potentielles surprises liées à l'utilisation du système interactif pouvant engendrer elles-mêmes des erreurs humaines.

Trois exemples illustrant les travaux forment la troisième partie.

Le chapitre 6 illustre de manière complète les aspects architecturaux et méthodologiques abordés dans les chapitres 3 et 5.

Le chapitre 7 montre un exemple de taille réelle d'exploration de techniques d'interaction multimodale et de leurs spécifications complètes pour une application existante dans les cockpits.

Le chapitre 8 présente une plateforme de développement actuellement utilisée pour prototyper des techniques d'interaction multimodale destinées aux hélicoptères du futur.

Nous conclurons en évaluant les perspectives liées à ces travaux.

Première partie

État de l'art et Positionnement de la thèse

Chapitre

1 Contexte et périmètre de la thèse

Sommaire

1.1 Introduction

Afin de détailler le contexte et les contours des travaux de ce mémoire, nous présenterons brièvement les projets industriels dans lesquels ils se sont déroulés, puis, nous définirons les systèmes critiques et les contraintes qu'ils entrainent tout au long de la conception de systèmes informatiques.

Nous définirons ensuite les systèmes interactifs, et établirons la liste des problématiques à prendre en compte lorsque l'on veut concevoir de tels systèmes.

Enfin nous nous intéresserons à la conception des systèmes interactifs destinés aux environnements critiques et verrons comment cette conception s'insère dans les deux domaines tout en ajoutant des contraintes supplémentaires.

1.2 Contexte applicatif

Les travaux de ce mémoire ont été réalisés en collaboration avec des industriels dans le cadre de financements CORAC (COnseil pour la Recherche Aéronautique Civile) de projets nationaux qui incluent des partenariats avec tous les grands acteurs aéronautiques français : AIRBUS group, AIRBUS, AIRBUS helicopters, THALES AVIONICS, DGAC, DASSAULT, ONERA, SAFRAN.

Ils se sont inscrits dans deux projets, FENICS puis IKKY.

1.2.1 FENICS : Fondement pour l'Etude de Nouvelles Interactions pour les futurs CockpitS ; WP2.7 multimodalité dans les cockpits

Le WorkPackage 2.7 de FENICS était centré sur la multimodalité dans les cockpits d'avions et a été réalisé au sein d'un partenariat entre AIRBUS et l'équipe ICS-IRIT. L'approche envisagée dans ce workpackage portait sur une vision à long terme. Son but étant de dégager des briques méthodologiques pour permettre l'insertion d'interactions multimodales dans un contexte aéronautique et notamment :

- Comment décrire de manière complète et non ambigüe une interface intégrant des interactions multimodales
- Comment intégrer les spécificités du cockpit dans un processus de développement de systèmes interactifs multimodaux

1.2.2 IKKY Intégration du KocKpit et de ses sYstèmes, WP 4.4 : MATRHIECS Multimodal activities to Rise Helicopter Interface Efficiency in cockpit systems

Le second projet est réalisé en collaboration avec AIRBUS Helicopter, BERTIN technology et l'équipe ICS-IRIT. Encore en cours au moment de la rédaction de ce manuscrit, il est centré sur la multimodalité dans les cockpits des hélicoptères du futur. L'approche

envisagée est plus abstraite, deux objectifs principaux étant définis par AIRBUS Helicopter dans ce workpackage :

- Définir une *bonne* modalité dans un contexte de cockpit d'hélicoptère. C'est à dire, comment associer un ensemble possible d'interactions pour une tâche donnée.
- Évaluer une modalité de manière à assurer une preuve de son efficacité en vue de la certification.

Le rôle de l'IRIT dans ce projet est principalement de fournir une plateforme de démonstration des concepts méthodologiques mis en œuvre tout au long du projet.

Ces deux projets ont la particularité de se situer dans le contexte des cockpits d'aéronefs pour lesquels il est important d'assurer un fonctionnement fiable, du système ET des interactions avec le système. Le domaine des systèmes critiques s'intéresse particulièrement à ces problématiques, et de nombreuses contributions ont été faites. Nous allons rapidement en présenter les bases dans la section ci après.

1.3 Les systèmes critiques

Les systèmes interactifs traités dans ce mémoire sont destinés aux environnements critiques. Nous allons les définir et voir comment mesurer la sureté de fonctionnement puis nous aborderons les problématiques qui définissent leurs conceptions.

1.3.1 Définition

La définition d'un système critique que nous utiliserons dans ce manuscrit est relativement commune, [Palanque and Bastide, 1994], [Sommerville, 2011] :

Un système est dit critique si son coût de développement est bien moindre que le coût d'une défaillance potentiellement catastrophique.

On peut en effet chiffrer aussi bien le coût d'une vie humaine suite à une catastrophe [Conley, 1976], que celui d'une erreur de développement sur un logiciel déployé dans des millions de télévisions qui entrainerait un rappel massif . Ces systèmes nécessitent donc des phases de développement et d'exploitation aussi sûres que possible afin de limiter au maximum les défaillances, et ainsi obtenir des systèmes sûrs de fonctionnement.

[Laprie et al., 1995] et [Avizienis et al., 2004] définissent les concepts de sûreté de fonctionnement (dependability) et sécurité (security). Ces deux propriétés sont essentielles pour les systèmes critiques. La sureté de fonctionnement est définie comme la propriété d'un système permettant à ses utilisateurs (des humains ou d'autres systèmes) de placer une confiance justifiée dans le service qu'il leur délivre.

La sécurité d'un système est définie comme la propriété de celui-ci à se protéger des attaques malveillantes, visant à lui nuire (autant par la création de défaillances du

système que par le vol d'informations confidentielles).

Ces deux concepts sont très liés et ont été raffinés dans un arbre présenté en figure 1.1 permettant de mettre en évidence les trois grands axes qui les constituent :

Les attributs (attributes) : définis par un ensemble de 6 propriétés inhérentes à un système sûr de fonctionnement .

Les entraves (threats) : les circonstances indésirables, leurs sources et leurs résultats, provoquant la non-sûreté de fonctionnement ou non-sécurité.

Les moyens (means) : les méthodes et techniques permettant de garantir au mieux la sûreté de fonctionnement et la sécurité.

Afin d'avoir un meilleur aperçu de ces axes, nous les détaillerons dans les sous-sections suivantes.

1.3.1.1 Les propriétés

En figure 1.1 dans le premier sous arbre, nous constatons que la sûreté de fonctionnement et la sécurité sont composées de diverses propriétés complémentaires. Ces propriétés, aussi appelées attributs, permettent, selon le système considéré, de mettre l'accent sur les aspects les plus pertinents de ces deux notions en fonction des applications auxquelles le système est destiné. On distingue ainsi :

La disponibilité (Availability) : capacité du système à être opérationnel (prêt à l'usage).

La fiabilité (Reliability) : capacité du système à fournir un service correct et continu.

La sécurité-innocuité (Safety) : capacité du système à éviter les conséquences catastrophiques pour son environnement et ses utilisateurs.

La confidentialité (Confidentiality) : capacité du système à garantir la non-divulgation d'informations confidentielles (cette propriété concerne seulement la sécurité).

L'intégrité (Integrity) : capacité du système à garantir l'absence d'altération de son information et de son fonctionnement (cette propriété concerne seulement la sécurité).

La maintenabilité (Maintainability) : capacité du système à être modifiable afin de permettre sa réparation et ses évolutions.

1.3.1.2 Les entraves

S'assurer de la présence des propriétés précédentes permet de concevoir des systèmes sûrs de fonctionnement. Cependant, ces propriétés peuvent être corrompues, entrainant des circonstances indésirables qui provoquent la non-sûreté de fonctionnement. Ces dernières ainsi que leurs sources et leurs effets, sont définies comme les entraves à la sûreté de fonctionnement et à la sécurité. On distingue trois types d'entraves :

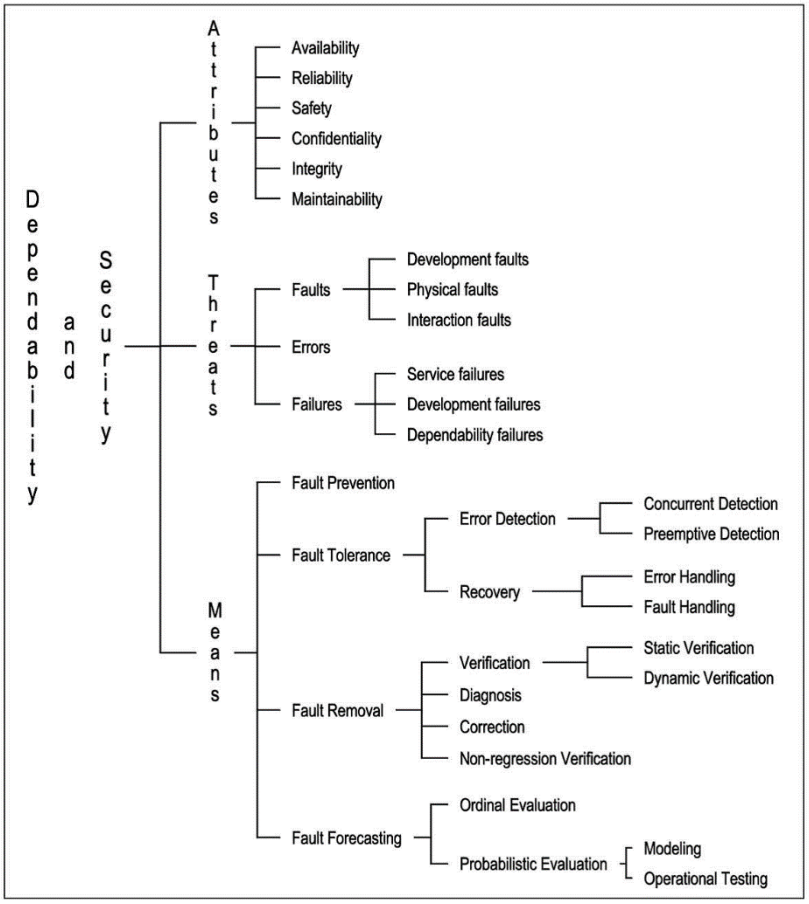


FIGURE 1.1 – Arbre des concepts de sûreté de fonctionnement et de sécurité [Avizienis et al., 2004]

les fautes (*Faults*), les erreurs (*Errors*) et les défaillances (*Failures*).

La défaillance d'un système correspond à l'occurrence d'un comportement inacceptable du système par rapport à son comportement attendu.

Une erreur est une partie de l'état du système susceptible d'entrainer une défaillance.

Les fautes, quant à elles, sont les causes (avérées ou hypothétiques) des erreurs. Suite à l'occurrence de la faute, le système passera d'un état nominal à un des états d'erreur.

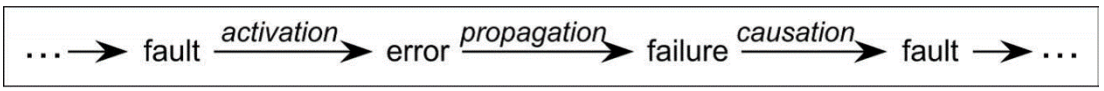


FIGURE 1.2 – Chaîne de causalité des entraves à la sûreté de fonctionnement et à la sécurité [Avizienis et al., 2004]

La figure 1.2 montre que ces trois entraves à la sûreté de fonctionnement sont liées par des relations de causalité. Ainsi, une faute est dormante tant qu'elle ne provoque pas d'erreur ; lors de son activation, elle déclenche une erreur. Par propagation, une erreur peut entrainer une défaillance (affectant alors le service du système), ou plusieurs erreurs entrainant à leur tour une défaillance. Cependant, une erreur peut également disparaître avant de provoquer la moindre défaillance. Par exemple, un service en état d'erreur qui ne sera jamais appelé, n'entrainera pas de défaillance.

On remarque également sur la figure 1.2 qu'une défaillance peut provoquer une faute. En effet, tout dépend de la frontière donnée au système : une défaillance affectant un composant A implique que le service qu'il fournit dévie de sa fonctionnalité. Ainsi, un composant B, utilisant le service du composant A, observera la défaillance du composant A comme une faute extérieure pouvant conduire à une erreur puis à une défaillance et ainsi de suite.

Les fautes pouvant affecter les systèmes informatiques peuvent être extrêmement diverses. Les travaux de [Avizienis et al., 2004] proposent une classification exhaustive permettant d'identifier toutes les classes de fautes qui peuvent affecter un même système. Cette classification se présente sous la forme d'un arbre et met en évidence les différents critères permettant d'identifier 31 classes de fautes élémentaires. Nous en présentons une version simplifiée en figure 1.1 dans la section 1.5. Nous retiendrons ainsi :

La phase de création ou d'occurrence : durant le développement ou la phase opérationnelle

La situation par rapport aux frontières du système : interne ou externe

La cause phénoménologique : due à l'homme ou à des phénomènes naturels

La dimension : affectant le matériel ou le logiciel

L'objectif : malveillant ou non malveillant

La persistance : fautes permanentes ou temporaires

1.3.1.3 Les moyens

Pour limiter l'apparition de défaillances, on agit sur leurs sources : c'est à dire les fautes, les erreurs ou d'autres défaillances. Dans ce but, plusieurs méthodes et techniques ont été identifiées. [Avizienis et al., 2004] proposent de les classifier en quatre catégories visibles sur la figure 1.1 :

La prévention des fautes (fault prevention) : permet d'éviter autant que possible l'introduction de fautes pendant le développement du système. On emploie généralement des techniques de développement rigoureuses, la formalisation, la modélisation, ...

La tolérance aux fautes (fault tolerance) : permet d'éviter la défaillance du système en présence de fautes via la détection des erreurs engendrées et leur recouvrement. La détection permet d'identifier la présence d'erreurs, leur type et leur cause. Le

recouvrement d'erreur vise à transformer l'état du système contenant une ou plusieurs erreurs en un état sans erreur de manière à ce que le service fourni par le système puisse toujours être assuré.

L'élimination de fautes (fault removal) : permet de réduire le nombre de fautes pouvant survenir à la fois pendant le développement du système (généralement en utilisant des techniques de vérification de propriétés, de preuves, model-checking, de tests, ...) et pendant la phase d'exploitation du système (par exemple en effectuant de la maintenance corrective).

La prévision des fautes (fault forecasting) : consiste à estimer le nombre, l'incidence et les conséquences probables de fautes, généralement en dressant une évaluation statistique de la fréquence et de l'impact des fautes.

1.3.2 Problématique de la mesure de la sûreté de fonctionnement

La sûreté de fonctionnement consiste à évaluer ses différents attributs et plus particulièrement ceux relevant de la fiabilité, de la disponibilité et de la maintenabilité [Laprie et al., 1995]. Ces mesures sont généralement effectuées à l'aide de définitions probabilistiques et d'estimateurs statistiques tels que le temps moyen jusqu'à la première défaillance (*MTTF Mean Time to Failure*) le temps moyen entre défaillance (*MTBF Mmean Time Between Failure*), la fiabilité initiale ou le taux de défaillance initial. Le choix des critères sur lesquels on portera un intérêt particulier pour la mesure de sûreté de fonctionnement se fait en fonction du domaine d'application des systèmes. Ainsi, pour un réseau téléphonique on privilégiera la disponibilité, pour une sonde spatiale, la fiabilité, et pour une centrale nucléaire, la sécurité-innocuité. Concernant les systèmes avioniques embarqués, le critère privilégié est la sécurité-innocuité. Il s'agit alors de garantir une probabilité d'occurrence de défaillance catastrophique inférieure à 10^{-9} par heure de vol pour chacun des systèmes critiques.

1.3.3 Problématique de la certification et des moyens de conformité

Le but de la certification est de convaincre un organisme agréé indépendant que le système développé assure un niveau de sûreté de fonctionnement suffisant. Les systèmes critiques nécessitent donc, en général, une certification, et plus particulièrement ceux dont la sécurité-innocuité est prépondérante. C'est le cas dans les domaines aéronautique, ferroviaire ou des centrales électriques. Lorsque qu'il n'y a pas d'organisme de certification, des processus de développement truffés de différentes preuves et de rationnels sur les décisions prises tout au long des phases de conception du système critique seront nécessaires (par exemple pour les systèmes informatiques dans le domaine automobile).

Les documents de certification sont le plus souvent composés d'une succession de documents relevant de preuves de sûreté de fonctionnement, plutôt que de tests.

Ce mémoire accorde peu de place aux tests pour la partie critique compte tenu des limites de son utilité pour la fiabilisation des sytèmes. La littérature grand public mentionne

souvent le test comme un moyen d'assurer la sûreté de fonctionnement. Or, les experts en sûreté de fonctionnement, par exemple Philip Koopman en 2014 lors de SAFECOMP [Koopman, 2014], mentionnent le fait que le test ne peut être utilisé comme outil principal de sûreté de fonctionnement, le test ne doit servir que pour venir compléter d'autres mécanismes et méthodes. Koopman explique que si on a testé une voiture pendant des milliers de kilomètres, ces tests ne seront pas représentatifs comparativement à la flotte de millions de véhicules vendus.

Les tests sont néanmoins des outils centraux de la conception des systèmes interactifs. Nous allons les détailler dans la section suivante.

1.4 Les systèmes interactifs

Nous définissons dans cette première section certains concepts clés des systèmes interactifs qui serviront de référence tout au long de ce mémoire.

La figure 1.3 présente un système interactif que l'on pourrait décrire comme classique. En effet, il dispose de périphériques physiques d'entrée et de sortie. Les périphériques d'entrée envoient des événements à l'interface utilisateur. Le traitement est réalisé, soit au niveau de l'interface s'il est purement destiné au rendu, soit au niveau d'un cœur applicatif ou d'un système contrôlé de l'autre coté de l'interface. Le système de rendu génère du contenu qui sera transformé en signal perceptible par l'utilisateur du système au travers des périphériques de sortie.

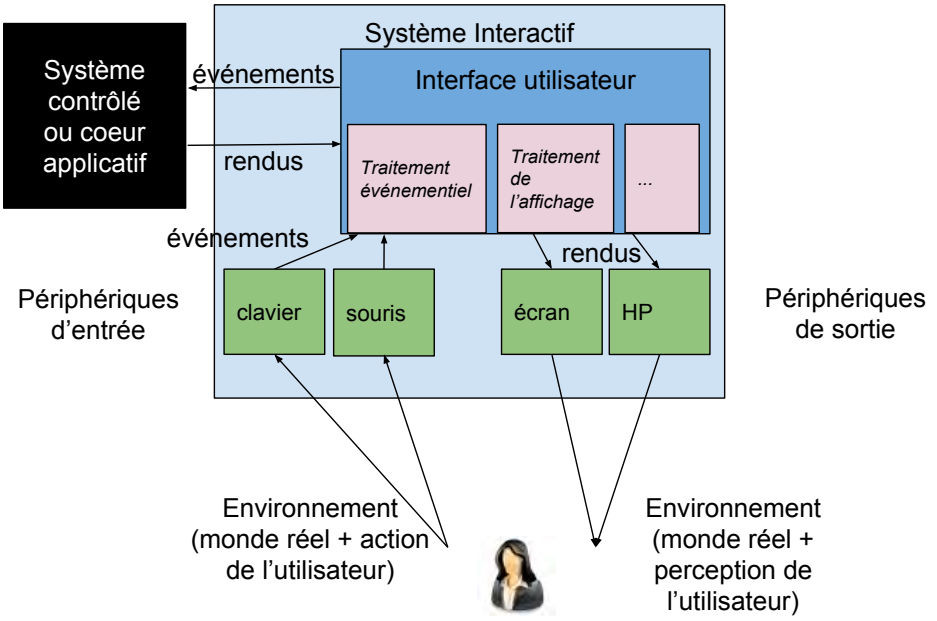


FIGURE 1.3 – Un système interactif classique

Un système interactif offre la possibilité à l'utilisateur d'interagir avec lui. Cette définition nous permet de réaliser que la plupart des appareils présents dans notre quotidien sont interactifs. Très peu de systèmes fonctionnent en parfaite autonomie, sans action humaine directe. On pourra alors faire la différence entre un système interactif centré sur l'interaction avec l'utilisateur, par exemple un smartphone dont les techniques d'interaction sont complexes afin de faciliter l'utilisation, et un système interactif dont le but premier est une mission, tel qu'un four à micro-ondes qui aurait des fonctions informatiques interactives de contrôle simples.

Un système uniquement interactif est donc dirigé principalement par les actions d'un utilisateur, qui sont imprévisibles par nature. Un système classique sera quant à lui dirigé par d'autres types d'évolutions, qu'elles soient déclenchées de manière interne, par d'autres systèmes connectés, de manière autonome ou liées au temps.

1.4.1 Problématiques des propriétés des systèmes interactifs

La conception des systèmes interactifs fait l'objet d'exigences et de besoins particuliers par rapport à celle de systèmes informatiques non interactifs. Les besoins relatifs aux systèmes interactifs sont très différents d'un système où l'interactivité n'est pas prédominante. En effet, dans ces derniers, on pourrait déterminer l'intégralité des évolutions par conception. Cela génère des problématiques et des propriétés spécifiques à assurer, qui seront pour la plupart différentes de celles des systèmes critiques.

Ces propriétés font l'objet de plusieurs travaux, standards et recommandations. Nous présentons dans les sous-sections suivantes les principales propriétés des systèmes interactifs : l'ensemble des propriétés permettant de garantir la qualité du système, l'utilisabilité du système et enfin le ressenti utilisateur, plus communément connu sous sa dénomination anglaise : *User eXperience (UX)*.

Les travaux de [Cockton and Gram, 1996] ont identifié des propriétés permettant de garantir la qualité des systèmes interactifs, déclinées notamment en deux grands types :

Les propriétés externes : liées à la qualité du système d'un point de vue de l'utilisateur

Les propriétés internes : liées à la qualité du système du point de vue de sa conception, de son ingénierie

1.4.1.1 Les propriétés externes

Les propriétés externes sont liées à l'utilisabilité du système et permettent de caractériser la qualité de l'interaction entre l'homme et la machine, en garantissant que le système est agréable à utiliser, fiable, compréhensible et qu'il permet d'accomplir les tâches nécessaires à l'utilisateur. Ces dernières se divisent en deux catégories :

La flexibilité de l'interaction qui qualifie la manière dont le système échange des informations avec l'utilisateur

La robustesse de l'interaction qui qualifie la capacité du système à permettre à l'utilisateur d'accomplir les tâches qu'il doit exécuter sans commettre d'erreurs irréversibles

Les propriétés de chaque catégories sont nombreuses, car elles sont très fortement liées à l'utilisateur et à son imprédictibilité qui rend son comportement et ses actions imprévisibles ; elles relèvent surtout du design du système interactif.

L'utilisabilité

L'utilisabilité d'un système interactif est définie par la norme ISO 9241 comme :

Le degré selon lequel un produit peut être utilisé, par des utilisateurs identifiés, pour atteindre des buts définis avec efficacité, efficience et satisfaction, dans un contexte d'utilisation spécifié [ISO, a]

Cette propriété se décompose donc en trois facteurs :

L'efficacité : capacité à atteindre le but prévu ;

L'efficience : capacité à atteindre le résultat prévu avec un effort et un temps minimal ;

La satisfaction : correspond au confort et au ressenti de l'utilisateur lors de son interaction avec le système.

L'utilisabilité est une propriété fondamentale pour les systèmes interactifs. En effet, si un système n'est pas utilisable, le système sera incapable de remplir le service désiré et ne fonctionnera pas, comme le suggère Susan Dray dans son slogan *« If the user can't use it, it doesn't work »*. Cette propriété est aujourd'hui largement répandue et mise en œuvre dans la communauté de recherche sur les interfaces homme-machine (IHM). Cette dernière a développé de nombreuses méthodes pour évaluer l'utilisabilité d'un système telles que la réalisation de tests avec des utilisateurs, d'évaluations heuristiques ou encore des méthodes basées sur les modèles [Nielsen, 1994]. Il est important de ne pas considérer l'utilisabilité d'un système seulement lorsque celui-ci est déployé mais de la considérer tout au long du développement. Cela peut être réalisé en analysant les besoins des utilisateurs lors de la conception du système et en utilisant des processus itératifs permettant de prendre en compte l'utilisabilité du système lors de différentes phases du développement.

L'utilisabilité d'un système est évaluée grâce aux trois propriétés qui la constituent :

L'efficacité peut être évaluée par une analyse des tâches utilisateurs. En effet, celle-ci permet de déterminer tous les services devant être proposés par le système. Une mise en correspondance entre les tâches utilisateurs et les fonctionnalités proposées par le système permet donc de vérifier l'efficacité de celui-ci [Martinie et al., 2015].

L'efficience peut être évaluée par des tests utilisateurs en mesurant des variables physiques telles que le temps requis pour effectuer une tâche. Elle peut également être évaluée en estimant la charge de travail de l'utilisateur en utilisant par exemple l'index NASA TLX (pour NASA Task Load Index) [Hart and Staveland, 1988].

La satisfaction est le plus souvent évaluée à l'aide de tests utilisateurs et de questionnaires tels que le questionnaire SUS (*System Usability Scale*) [Brooke et al., 1996].

Concernant les tests utilisateurs, les travaux de Nielsen [Nielsen, 1994] ont montré qu'une étude avec cinq utilisateurs pouvait détecter 80% des problèmes d'utilisabilité (voir Figure 1.4). C'est pour cela que Nielsen promeut le concept de « discount usability

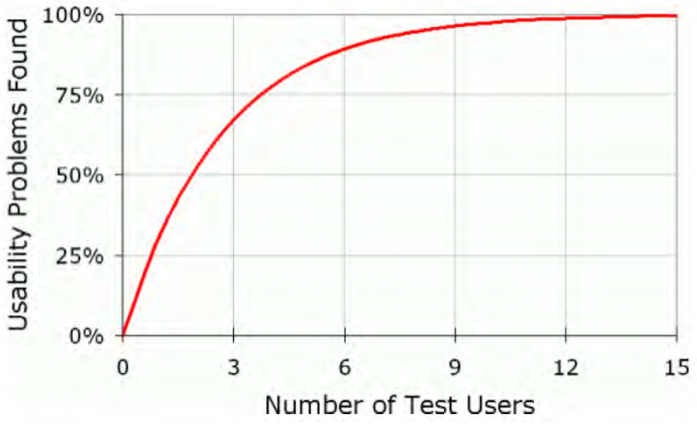


FIGURE 1.4 – Problèmes d'utilisabilités détectés en fonction du nombre de tests utilisateurs [Nielsen, 2000]

engineering » [Nielsen, 2009] en soulignant qu'une étude d'utilisabilité impliquant des tests avec cinq utilisateurs choisis selon des critère précis, permet d'assurer, à moindre coût, que le système est suffisamment utilisable.

L'User eXperience (UX)

L'expérience utilisateur, ou le ressenti utilisateur (*User eXperience*) est, à l'heure actuelle, une des propriétés des systèmes interactifs les plus étudiées par la communauté scientifique en IHM. Elle est définie par la norme ISO 9241 [ISO, a] comme « les perceptions et les réactions d'une personne qui résultent de l'utilisation effective et/ou anticipée d'un produit, système ou service ». Cependant, comme le soulignent les travaux de [Bernhaupt, 2015], cette définition se concentre sur la perception de l'utilisateur et ne permet pas de donner les moyens d'évaluer l'*user experience*. Il est important de clarifier les différents facteurs et dimensions de cette propriété afin de permettre son évaluation. Les travaux de [Bernhaupt, 2015] résument les dimensions de l'*user experience* qui ont été traitées par les communautés de recherche de l'IHM et des jeux. Cette propriété peut ainsi se diviser en quatre dimensions fondamentales :

Esthétique : comment le système est-il perçu (par exemple agréable ou esthétique)

Émotion : comment le système peut-il provoquer des émotions, ou peut-il affecter l'utilisateur

Stimulation : comment le système peut-il donner à l'utilisateur des services innovants et intéressants

Identification : comment le système peut-il permettre à l'utilisateur de s'identifier à ce dernier

Ces dimensions fondamentales sont associées à d'autres facteurs tels que la valeur et le sens, le lien social, la sûreté de fonctionnement, la sécurité et la confiance, la qualité de service, l'immersion, l'implication, l'engagement ou encore la jouabilité. Comme le mettent en évidence les travaux de [Bernhaupt, 2015], l'*user experience* doit être, de la même manière que l'utilisabilité, prise en compte tout au long du processus de développement du système, depuis sa spécification jusqu'à sa mise en service.

1.4.1.2 Les propriétés internes

Les propriétés internes sont liées à la conception, autant logicielle que matérielle. Elles sont très fortement liées aux propriétés classiques de l'ingénierie de conception. Elles permettent par exemple de garantir la capacité d'évolution du système, sa maintenabilité, ses performances et son exhaustivité fonctionnelle (capacité à fournir à l'utilisateur les fonctions dont il a besoin). Ces propriétés relèvent du développement du système. Leur garantie peut être assurée par l'utilisation d'architectures logicielles spécifiques aux systèmes interactifs.

Nous détaillerons dans la section suivante des problématiques spécifiques aux propriétés internes.

1.4.2 Les problématiques de l'ingénierie de l'interaction

Concevoir un système interactif multimodale nécessite de garder présent à l'esprit un ensemble de problématiques liées à ses différents composants. En effet, à la prise en compte les aspects génériques du génie logiciel, devront s'ajouter celles des périphériques et de leur gestion dans le système.

1.4.2.1 Préliminaire : Notion de modalités

[Nigay and Coutaz, 1996] définit une modalité comme le couple <p, r> où :

- p désigne un dispositif physique (par exemple, une souris, une caméra, un écran, un haut-parleur).
- r dénote un système représentationnel, c'est-à-dire un système conventionnel structuré de signes assurant une fonction de communication (par ex., un langage pseudo naturel, un graphe, une table).

L'expression suivante fournit une définition complète de la notion de modalité :

$$\text{modalité} ::= \langle p, r \rangle \mid \langle \text{modalité}, r \rangle$$

Le premier pas vers l'utilisation d'interaction multimodale est donc l'intégration d'un dispositif physique (c'est à dire d'un périphérique), qui fournira un signal au système ou produira un rendu utilisateur. La gestion informatique d'un signal électrique continu peut être réalisée de très nombreuses manières et pose plusieurs problèmes.

1.4.2.2 Problématique des périphériques physiques et de leurs représentations informatiques

Un des problèmes de la gestion de l'interaction est le choix du moyen utilisé pour interagir physiquement sur le système face à sa représentation informatique

Un exemple représentatif de cette problématique est le smartphone : la représentation de l'utilisateur serait le prolongement des doigts de la main (avec plusieurs récepteurs) alors que l'appareil n'est qu'un seul périphérique (la dalle tactile), qui fournit plusieurs ensembles de coordonnées au système informatique.

Il faut donc séparer la représentation de l'interacteur de celle du périphérique fournissant le signal électrique, ou tout du moins extraire une représentation informatique de l'interacteur à partir du signal fourni par le périphérique.

1.4.2.3 Problématique du domaine continu vs domaine discret

La différence fondamentale entre le monde réel et le monde informatique, est que le premier est continu, alors que le second est discret. C'est un problème générique pour tout système possédant des capteurs.

La figure 1.5 présente le flux auquel les concepteurs de systèmes sont confrontés.

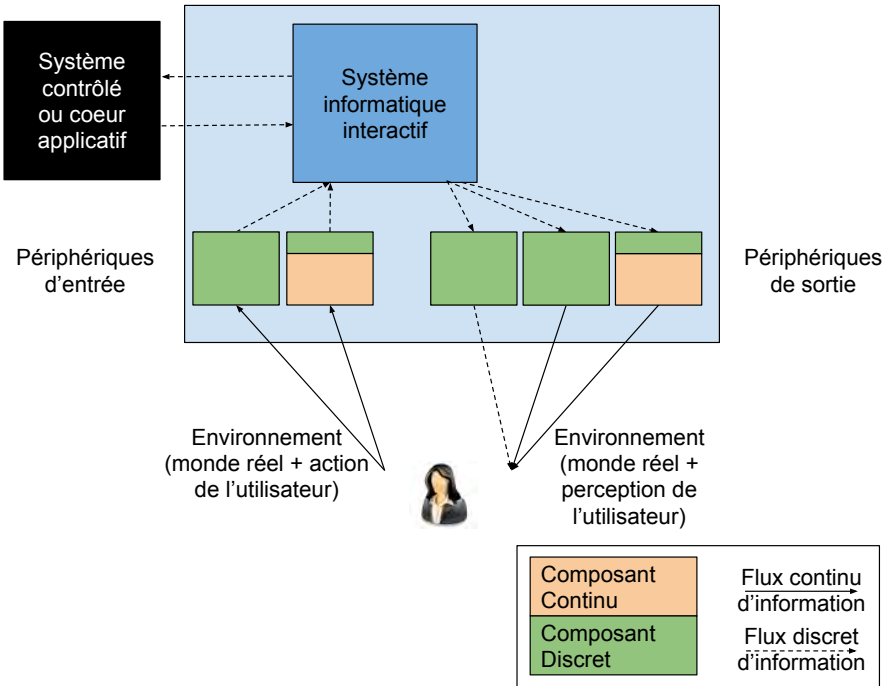


FIGURE 1.5 – Le monde continu *vs* le monde informatique discret

Du côté des entrées, en bas à gauche de la figure 1.5, les utilisateurs vont toujours réaliser des mouvements continus, qu'ils soient détectés par le système d'entrée ou pas.

Les périphériques sont alors de deux grands types :

1. Les périphériques purement discrets, par exemple un bouton ou une touche de clavier, qui possède exactement deux états : enfoncé ou relâché
2. Les périphériques qui convertissent un signal continu en un signal discret, par exemple un potentiomètre numérique qui transformera un signal électrique continu en valeur numérique exploitable par le système

En conséquence, le système informatique recevra toujours un flux discret d'informations.

De l'autre coté, le système informatique va toujours produire un flux discret d'informations, les périphériques de sortie seront alors de trois types :

1. Discret vers discret : par exemple une LED qui clignotera, même si le signal lumineux est bien continu, l'information perçue par l'utilisateur est belle et bien discrète
2. Discret vers perception continue : Par exemple un écran d'ordinateur, grâce à la persistance rétinienne, permet à l'utilisateur de percevoir une information continue, alors que le périphérique produit une information discrète, plusieurs fois par seconde.
3. Discret vers continu : dans ce cas un vrai signal continu sera reproduit à partir d'informations discrètes, par exemple le son d'un haut-parleur.

La nécessité de continuité du flux d'informations dans le système implique une prise en compte des différentes classes de périphériques dans l'élaboration de l'interaction.

1.4.2.4 Problématique de la normalisation des périphériques

Les interactions se multipliant et se complexifiant, il devient nécessaire de normaliser les flux, au sens mathématique. En d'autres termes, il faut unifier les types de flux d'entrée dans le système afin de pouvoir appliquer des méthodes de traitement similaire sur les interactions. Par exemple, on souhaitera avoir le même comportement sur un bouton, que l'on ait cliqué avec le curseur d'une souris, ou avec un doigt sur une surface tactile.

La représentation informatique d'une même interaction peut parfois poser problème en fonction du périphérique utilisé. Par exemple, deux souris n'ayant pas la même résolution temporelle ou surfacique déplaceront le curseur à des vitesses bien différentes malgré une quantité identique de mouvement sur la table. Le problème est encore plus flagrant si pour déplacer le curseur, on utilise plusieurs types de périphériques différents. Par exemple, le trackpad, et la souris, il faut alors réfléchir à la normalisation, souvent au travers des fonctions de transfert [Casiez and Roussel, 2011] des entrées afin d'obtenir le même type, et les mêmes limites, pour chacun des flux de données.

1.4.2.5 Problématiques de la combinaison de modalités

Il existe de nombreux travaux dans le domaine de la multimodalité dans la littérature. Les propriétés CARE [Coutaz et al., 1995] et TYCOON[Martin and Beroule, 1993]

font partie de la base commune de classification des modalités. Les propriétés CARE constituent un cadre formel permettant de décrire les relations entre modalités. Bien qu'originellement présentées par J coutaz et L Nigay pour caractériser les modalités en entrée, elles ont été étendues par la suite par Vernier [Vernier, 2001] pour deux modalités en sortie.

La gestion des modalités en entrée implique aussi la fusion d'événements, qu'elle soit réalisée au niveau des données, au niveau temporel ou encore de manière sémantique [Nigay and Coutaz, 1996]. Vernier résume dans son manuscrit de thèse les compositions de deux modalités selon cinq aspects, que nous pouvons retrouver en figure 1.6

Schémas de composition

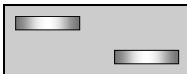
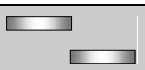
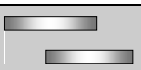
Aspects de composition \ Composition					
Temporelle	Anachronique	Séquentielle	Concomitante	Coïncidente	Parallèle / Simultanée
Spatiale	Disjointe	Adjacente	Intersectée	Imbriquée	Recouvrance
Articulatoire	Indépendance	Fissionnée	Fissionnée + Dupliquée	Partiellement Dupliquée	Dupliquée
Syntaxique	Différente	Complétion	Divergence	Extension	Jumelage
Sémantique	Concurrente	Complé-mentaire	Complémentaire + Redondante	Partiellement Redondante	Totalement Redondante

FIGURE 1.6 – Différents aspects de composition pour deux modalités., extrait de [Vernier, 2001]

Ces propriétés sont néanmoins insuffisantes pour définir un cadre d'utilisation dans les systèmes critiques. En effet, elles résultent d'une vision optimiste, et se concentrent sur les combinaisons positives et entre modalités. Par exemple, on ne pourra pas définir l'exclusion de modalités. Par exemple dans les cockpits d'AIRBUS, bien qu'il y ait un écran partagé et deux curseurs, lorsqu'ils sont tous les deux sur l'écran partagé, le curseur du copilote est désactivé pour donner la priorité à celui du pilote.

1.4.2.6 Problématiques temporelles

Utiliser plusieurs modalités implique de devoir gérer la temporalité des événements. En effet, utiliser deux modalités "en même temps" peut avoir plusieurs sens selon la précision temporelle, ou encore la durée de certains événements.

La problématique temporelle peut être dissociée en deux grands axes, le traitement des différentes combinaisons de fenêtre temporelle de manière sémantique, qui déterminera le design ainsi que le comportement haut niveau de l'interaction, et le traitement quantitatif du temps, qui fait partie intégrante de l'ingénierie du système interactif.

Problématique du temps réel quantitatif

La combinaison de modalités doit se faire dans une fenêtre temporelle définie, par exemple 300 ms. Une variation sur ces données précises peut amener un utilisateur à ne pas réussir une interaction. En général, dans le monde de la recherche, les aspects quantitatifs sont laissés en dehors des modèles. Leur traitement devra donc se pencher sur les moyens et outils nécessaires à la garantie d'exécution d'une spécification technique temporelle. Dans les systèmes critiques, on utilise le plus souvent des analyses de WCET (worst case execution time), temps pire d'exécution, pour s'assurer d'un comportement précis au pire cas lors de l'opération du système. Cela nécessite d'avoir un contrôle complet sur le système interactif, que ce soit en matière de ressources matérielles ou de ressources logicielles.

Problématique du retard (lag)

Les techniques d'interaction modernes sont plus fréquemment basées sur des interactions complexes, telles que la reconnaissance vocale, les gestuelles mid-air, les interactions face à une caméra. Le temps de reconnaissance de l'action devient un problème à gérer. Le traitement de la voix ou de l'image prend un certain temps, qui peut être non négligeable pour un humain, ou pire encore, si ce mode est couplé à une interaction instantanée comme un "*put that there*" [Bolt, 1980] avec une souris et la reconnaissance vocale. Il faudra alors prendre en compte ces différences de temps de traitement pour la gestion de l'interaction. Un autre genre de *lag* peut se produire lorsque l'interaction elle-même est longue à générer comme dans les exemples de [Bau and Mackay, 2008]. l'état du système peut alors évoluer plus rapidement que le temps de produire l'interaction.

1.4.2.7 Problématique du rendu

Problématique de l'état du système interactif

Campos et Harrison ont écrit dans [Campos and Harrison, 1997] que les entrées d'un système interactif sont basées sur des événements, et que ses sorties sont basées sur des états. Cette distinction est extrêmement importante et sa prise en compte a un impact sur les contributions présentées dans ce mémoire.

La partie entrée a été couverte dans les sections précédentes, mettant en avant en particulier les aspects continus *vs* discret. Les sorties (rendus) étant basées sur l'état du système, il est nécessaire d'avoir une représentation complète et non ambigüe de l'ensemble des états du système. Pour pouvoir effectuer ces rendus par état, il faut avoir une représentation de l'état du système complet, ou au minimum un sous-état, permettant d'effectuer un rendu unique et déterminant. On pourrait décider d'effectuer un rendu sur l'intégralité des états décrits dans la spécification. Une trop grande quantité d'informations serait malheureusement présentée à l'utilisateur, rendant leurs prises en compte incorrectes. En effet, la quantité d'états cachés d'un système interactif est souvent bien trop grande pour qu'un utilisateur puisse en interpréter les données en temps réel.

On n'affiche pas, par exemple, la quantité d'accélération courante du pointeur d'une souris sur les systèmes d'exploitation grand public. Une délégation de certaines tâches interactives est donc nécessaire ; nous présentons cet aspect dans la section 1.4.2.8

Problématique de l'espace de rendu

Contrairement aux dispositifs d'entrée qui sont le plus souvent assignés à une modalité à un moment donné, les dispositifs de rendu doivent être capables de présenter plusieurs informations aux utilisateurs. On trouve des exemples de composition intersectée tôt dans la littérature, par exemple dans [Bier et al., 1993]. L'assignation, ou la composition de rendu en sortie fait donc partie des problématiques majeures à gérer en ingénierie de l'interaction. Les relations entre modalités doivent aussi être prises en compte entre les entrées et les sorties.

Problématique de la synchronisation entrée/sortie

Les interactions purement informatiques ont une spécificité. Contrairement aux interactions physiques, elles nécessitent l'ajout de *feedback immédiat*. En effet, l'utilisateur doit pouvoir accéder au rendu de son interaction, puisque celle–ci n'est qu'une représentation indirecte de son interaction physique. Pour être efficace, ce rendu doit être synchronisé avec l'interaction physique. Plusieurs études dont [Barnard and May, 1995] ont démontré qu'il fallait régler ce rendu pour qu'il ait une fenêtre temporelle par rapport à l'interaction en entrée inférieure à 100 millisecondes afin de ne pas perturber les tâches de l'opérateur. Cela vient en complément des règles d'ergonomie sur le type de rendu à effectuer dans le cas d'un traitement long de l'interaction.

1.4.2.8 L'automatisation dans les systèmes interactifs

Wood définit dans [Woods, 1985], qu'avec la complexité grandissante des systèmes, il est nécessaire de déléguer certaines tâches de décision au système interactif, bien que cela dresse de nouvelles problématiques (supervision ou reprise de contrôle) sur un système rendu en partie voire totalement autonome.

[Parasuraman and Riley, 1997] ont décrit l'automatisation comme étant

Un dispositif ou un système qui accomplit une fonction *qui était ou pourrait être, de par sa conception, totalement ou partiellement réalisée par un opérateur humain.*

Dès lors, ils définissent les degrés selon lesquels peut varier l'automatisation, depuis le niveau le plus faible, où tout est opéré de manière manuelle, jusqu'au niveau le plus élevé, où tout est automatisé, sans intervention humaine ni notification.

Même si le tableau de la figure 1.7 peut aider à comprendre les niveaux d'automatisation, ces derniers ne peuvent être utilisés comme outils d'analyse généraux de l'automatisation. Elle devra être abordée *fonction* par *fonction*. S'il est possible d'obtenir en détail chacune des *fonctions*, on pourra décider des tâches pouvant être migrées de l'opérateur vers le système, ainsi que de l'optimisation d'une telle migration.

Low	1	The computer offers no assistance : human must take all decision and actions.
	2	The computer offers a complete set of decision/action alternatives, or
	3	Narrows the selection down to a few, or
	4	Suggests one alternative, and
	5	Executes that suggestion if the human approves, or
	6	Allows the human a restricted time to veto before automatic execution, or
	7	Executes automatically, then necessarily informs humans, and
	8	Informs the human only if asked, or
	9	Informs the human only if it, the computer, decides to.
High	10	The computer decides everything and acts autonomously, ignoring the human.

FIGURE 1.7 – Automation Level Automation Description, extrait de [Parasuraman and Riley, 1997]

Ces niveaux d'automatisation peuvent être appliqués à des macro-tâches, par exemple le maintien d'un cap et d'une altitude sur un aéronef. Ils peuvent aussi être appliqués à un niveau micro, par exemple, au sein des techniques d'interaction. Prenons le double clic : si l'intervalle de temps écoulé entre deux simple clic est supérieur à celui défini par design, la technique d'interaction rejettera automatiquement le déclenchement du double clic.

Les précédents travaux en IHM, comme en sûreté de fonctionnement, ont négligé la nécessité d'automatisation de certaines tâches dans les systèmes de contrôle et commande pour que les utilisateurs puissent rester efficaces en opération. Il faut gérer autant l'aspect système, lié à l'automatisation de tâche, que l'impact sur les utilisateurs.

1.4.3 L'aspect utilisateurs

1.4.3.1 Problème de la multiplicité des utilisateurs

Cette thèse s'intéresse au contexte particulier réunissant plusieurs utilisateurs sur un seul système. Contrairement au smartphone, utilisé par une seule personne à la fois, un cockpit d'aéronef est généralement opéré par deux pilotes. Certaines tâches ne peuvent être effectuées que par l'un d'entre eux, induisant le problème de l'identification des utilisateurs, ou de leur rôle. D'autres peuvent être réalisées par les deux pilotes. Le système doit ainsi fournir des commandes qui leur permettent d'effectuer tous deux les tâches, mais il faut alors définir une priorité quant au rôle, comme c'est le cas actuellement dans les avions. Enfin, certaines tâches sont collaboratives, la composante multimodale de l'interaction s'exprime alors dans l'utilisation multiple d'un type de modalité, plutôt que dans la multiplication de modalités différentes. La particularité de ces systèmes collaboratifs réside dans la co-localisation des utilisateurs qui interagissent avec un seul système interactif.

Nous n'aborderons pas le cas des systèmes distribués distants.

1.4.3.2 L'importance des fautes humaines

[Woods, 1985] décrit les *joint human-machine cognitive systems*, comme étant une intégration de l'humain et de la machine, afin de raisonner sur un seul ensemble lors de la conception. Pour assurer la sûreté de fonctionnement, il faut donc aussi s'intéresser aux fautes, erreurs et défaillances humaine, adressées dans un domaine de recherche à part entière.

Classification générique des erreurs humaines

Normann [Norman, 1983] définit trois grands types d'erreurs humaines :

Laps : les lapsus, ou erreurs par omission

Mistakes : les fautes à proprement parler, qui relèvent du problème sémantique

Slips : les ratés, qui sont liés le plus souvent à une imprécision lors de l'interaction. Ils peuvent être gérés au niveau du design de l'interaction

Ces trois types d'erreurs sont génériques, mais ne peuvent pas être gérées au niveau de la méthode d'ingénierie. Ces problèmes peuvent être pris en compte au cas par cas, lors de la conception, ou plutôt lors du design de l'interaction du système, par exemple à l'aide de méthodes basées sur la modélisation des tâches comme HAMSTERS [Fahssi et al., 2014].

Le point de vue de ce manuscrit étant orienté système, nous traiterons les erreurs humaines comme étant des causes probables d'erreurs et de défaillances pour le système. Nous parlerons donc de **fautes humaines** plutôt que d'erreurs humaines.

Les fautes humaines liées à l'interaction avec un système autonome

Un autre type de faute, spécifique à l'utilisation des systèmes interactifs, n'est généralement pas différencié de ces trois grands types d'erreurs : les fautes liées aux surprises causées par les comportements autonomes du système, appelées *automations surprise*s dans [Palmer, 1995]. Ces surprises dégradent la performance de l'utilisateur alors que l'automatisation est censée l'améliorer. Ce contexte va nous intéresser particulièrement car les fautes liées au surprises peuvent potentiellement se gérer de manière générique, augmentant de fait la fiabilité du système.

Si la conception de l'interaction n'est pas faite de manière formelle, il peut apparaître des comportements autonomes non souhaités. Si la spécification n'est pas complète, voire ambiguë, les codeurs pourront être amenés à faire des choix de comportement qui ne seront pas rendus à l'utilisateur. Ces comportements autonomes au niveau micro peuvent entrainer des surprises pour l'utilisateur si le comportement effectif du système ne correspond pas à son modèle mental. Ces surprises liées aux comportements autonomes peuvent entrainer des pannes de l'interaction, voire plus grave, si les fonctionnalités touchées sont d'un niveau élevé de criticité .

Il est donc nécessaire d'intégrer aux processus de développement une étape de validation des comportements autonomes dans les techniques d'interaction, sachant que l'automatisation des macro-tâches est normalement gérée en amont, dès la conception du système interactif critique.

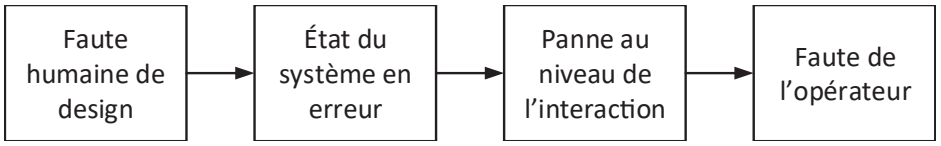


FIGURE 1.8 – Processus menant à une faute humaine à cause d'une surprise liée au comportement autonome

1.5 Les problématiques de l'ingénierie logicielle des systèmes interactifs critiques

1.5.1 Les systèmes interactifs et la sûreté de fonctionnement

Les systèmes interactifs critiques actuels ont, jusqu'à présent et pour des raisons de sûreté de fonctionnement, mis l'accent sur l'utilisabilité plutôt que sur l'expérience utilisateur. La possibilité pour l'opérateur de réaliser les tâches de manière efficace est considérée comme essentielle par rapport à l'expérience qu'il en tire, ce qui est relativement contraire à la politique des grands constructeurs informatiques actuels. De plus, les systèmes interactifs critiques possèdent des jeux de périphériques d'entrée/sortie relativement restreints. Aujourd'hui, un cockpit d'avion Airbus possède deux ensembles clavier/souris appelés KCCU (Keyboard and Cursor Control Unit). Les interactions présentes dans les cockpits sont directement issues du paradigme d'interaction WIMP, et donc de la troisième grande ère des interactions telles que décrites par Van Dam dans [Van Dam, 1997]. La conception de systèmes multimodaux post-WIMP telle que définie par Van Dam , étant beaucoup plus complexe, il faut de nouvelles méthodes et processus adaptés, pour permettre leur introduction dans les systèmes critiques [Palanque and Bastide, 1994]. Les interactions naturelles, modernes et esthétiques de notre quotidien n'existent pas dans les systèmes interactifs des cockpits d'avions.

La conception de systèmes interactifs critiques, comparée à celle dédiée aux systèmes grand public, nécessite des précautions particulières. Le temps et les durées d'exploitation élevés de ces systèmes nécessitent la prise en compte, dès la conception, de l'intégration ou de la suppression d'un périphérique, que cela se produise durant l'opération suite à une panne, ou longtemps après la conception initiale, par exemple pour des évolutions de technologie.

En effet, La durée d'exploitation d'un modèle d'avion variant de 30 à 80 ans entre sa conception et le retrait du service du dernier modèle, ses technologies sont donc amenées à évoluer.

Le temps d'exploitation est lui aussi bien plus important que celui des systèmes grand public. Par exemple, le temps de démarrage des systèmes informatiques d'un AIRBUS A380 est d'environ 8h lorsque toutes les procédures de sécurité sont actives. Ces systèmes ne sont donc que très rarement arrêtés, et doivent supporter la disparition d'un périphérique pendant leur opération.

D'un point de vue logiciel, l'interaction avec le système doit être décrite de manière complète et non ambiguë [Palanque, 1992]. L'ingénierie des systèmes critiques ayant très rapidement nécessité la production d'architectures logicielles et matérielles ainsi que l'utilisation de langages formels et de mécanismes de sûreté de fonctionnement, ces contraintes se sont naturellement retrouvées au cœur de la conception des systèmes interactifs destinés aux milieux critiques. Les approches zéro défaut, utilisées dans des processus de développement adaptés aux systèmes critiques sont donc aujourd'hui au cœur des processus utilisés pour ce type de système. La complexité de ces systèmes amène les concepteurs à décrire de manière indépendante les techniques d'interaction pour pouvoir vérifier leurs comportements face à une spécification.

Le traitement de l'aspect critique est tributaire de celui des fautes, erreurs et défaillances. Nous devons tout d'abord nous pencher sur les formes qu'elles peuvent revêtir dans les systèmes interactifs critiques, et aborder leur classification.

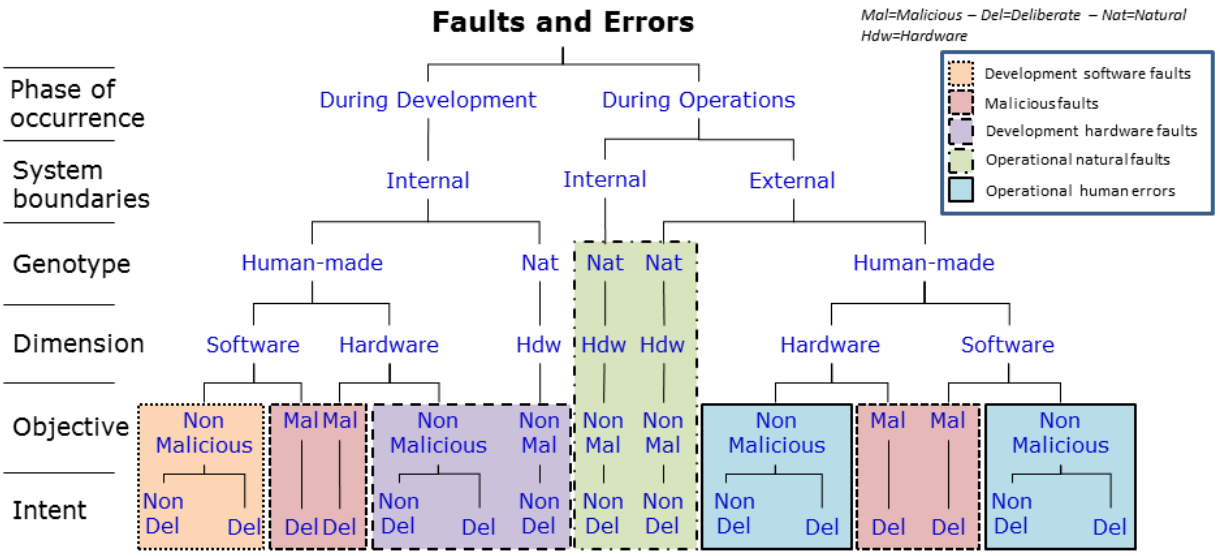


FIGURE 1.9 – Classification des fautes des systèmes informatiques, extrait de [Fayollas, 2015]

la figure 1.9 présente une version simplifiée de la classification des fautes pouvant affecter les systèmes informatiques proposés par les travaux de Avizienis, Laprie, et al. 2004 [Avizienis et al., 2004]. Cette figure met en évidence cinq regroupements de classes de fautes que nous avons identifiés. Ces regroupements peuvent être considérés comme des classes de fautes de haut niveau :

Fautes logicielles de développement : fautes introduites involontairement durant le développement du système. Par exemple, les erreurs de codage ou les erreurs de conception.

Fautes malveillantes : fautes introduites délibérément pour provoquer la défaillance du

système. Elles relèvent de la sécurité du système. Par exemple, la prise de contrôle extérieur ou un déni de service inopiné, un crash du système.

Fautes matérielles de développement : fautes ayant une cause naturelle ou humaine et impactant le matériel durant sa conception. Par exemple, un court-circuit à l'intérieur d'un processeur ou un matériel affecté lors de son développement par l'utilisation d'une eau avec une concentration trop forte en uranium, provoquant ainsi des erreurs lors de l'utilisation de celui-ci comme ce fut le cas en 1978 pour une usine d'IBM [Ziegler et al., 1996].

Fautes naturelles en opération : fautes causées par un phénomène naturel. Elles affectent le matériel (et par conséquent le logiciel) et surviennent pendant le fonctionnement du système. Par exemple, la modification de la mémoire suite à un rayonnement électromagnétique ou l'effacement d'un disque dur dans une zone magnétisée.

Fautes humaines en opération : fautes qui résultent d'une action humaine pendant le fonctionnement du système. Elles peuvent être matérielles et logicielles, délibérées ou non : par exemple, l'oubli d'une étape dans une procédure ou l'appui sur un mauvais bouton.

À partir de ces cinq grandes classes de fautes, on peut travailler à la réduction ou la suppression des effets de certaines d'entre elles :

- Les fautes logicielles de développement peuvent être réduites, voire supprimées en utilisant une approche zéro défaut, basée sur les méthodes formelles, méthodes suivies dans ce manuscrit.
- Les fautes malveillantes lors du développement ou au cours de l'opération sont en dehors du périmètre de ce mémoire. Elles relèvent de la sécurité informatique, qu'elle soit du ressort de l'organisation ou du développement de solutions spécifiques.
- Les travaux de ce mémoire ne prennent pas en compte les fautes liées au développement du matériel, elles ne sont pas du ressort de l'informatique.
- Les fautes naturelles sont gérées à plusieurs endroits : matériellement tout d'abord, via des circuits de vérification, logiciellement ensuite, via la prévention de fautes, par l'utilisation de mécanismes de sûreté du logiciel. On peut, en utilisant certaines de ces méthodes, appliquer la sûreté de fonctionnement aux systèmes interactifs critiques, comme dans les travaux de [Fayollas, 2015]
- Les fautes humaines en opérations sont, la plupart du temps, non traitées dans l'ingénierie des systèmes interactifs. Elles relèvent le plus souvent d'analyse d'accidents à postériori. Nous proposerons néanmoins une méthode permettant de supprimer celles déclenchées par les surprises liées à l'utilisation des systèmes interactifs.

Nous nous appuierons sur la figure 1.9 tout au long de ce manuscrit pour situer les problématiques traitées par rapport aux fautes et erreurs prises en compte dans les contributions.

1.5.2 Conflits entre propriétés

Des difficultés apparaissent lorsque l'on essaye d'assurer les bonnes propriétés d'un système interactif critique du fait de l'entrée en conflit de certaines des propriétés présentées dans ce chapitre (propriétés des systèmes interactifs, des systèmes critiques, et bien entendu, de leur combinaison).

Les systèmes interactifs critiques relèvent à la fois des propriétés des systèmes interactifs et de celles des systèmes critiques. Parmi celles que nous avons exposées précédemment, certaines peuvent être plus importantes que d'autres selon le domaine d'application des systèmes concernés. Ainsi, comparons le domaine des systèmes interactifs critiques à l'activité entrepreneuriale comme les domaines des télévisions interactives ou des jeux vidéos. L'*UX (User eXperience)* aura une importance primordiale dans l'activité entrepreneuriale, alors qu'elle sera reléguée à bas niveau pour les systèmes interactifs critiques où les vies humaines et l'utilisabilité sont prioritaires.

Dans tous les cas, la sûreté de fonctionnement reste primordiale. Nous nous intéressons ici aux systèmes interactifs présents dans les cockpits d'aéronefs et par extension aux systèmes interactifs critiques par rapport à la vie humaine, nous retiendrons donc les propriétés de sûreté de fonctionnement et d'utilisabilité comme étant les plus importantes à garantir.

Ces deux propriétés sont orthogonales et peuvent faire l'objet d'un conflit. Ceci a été montré dans les travaux de [Martinie et al., 2010]. Ces derniers proposent une notation pour aider les concepteurs de systèmes interactifs critiques dans leur choix de conception. Ils mettent en évidence que certains choix de conceptions permettant de garantir la sûreté de fonctionnement du système peuvent affecter l'utilisabilité du système. De la même manière, rendre un système plus utilisable, en implantant par exemple des techniques d'interaction avancées, peut affecter sa sûreté de fonctionnement en introduisant des fonctionnalités supplémentaires ayant pour conséquence d'augmenter la complexité du logiciel. Ces deux propriétés sont généralement traitées de manière séparée lors de la conception et du développement du système sans étudier leur impact mutuel. Il est fondamental les prendre en compte en les traitant de manière intégrée et systématique afin de pouvoir concevoir et développer des systèmes interactifs critiques utilisables et sûrs de fonctionnement. Certains travaux proposent des solutions pour traiter ces deux propriétés communément et de manière systématique. On retrouve par exemple le processus de développement présenté dans les travaux de [Martinie et al., 2012] ainsi que ceux de [Tankeu Choitat, 2011] présentant une approche pour le développement de systèmes interactifs intégrant à la fois les aspects sûreté de fonctionnement et utilisabilité.

Les travaux de ce mémoire chercheront donc à fournir des moyens pour identifier, et ordonner les propriétés nécessaires à la conception d'un système interactif sûr de fonctionnement.

1.6 Synthèse

Cet état de l'art nous permet de dégager deux axes de problématique principaux centrés sur la conception de systèmes interactifs critiques et de leurs interactions. Les concepteurs de tels systèmes ont besoin de processus et de méthodes qui permettront l'intégration d'interactions avancées dans les environnements critiques. Ces méthodes et outils doivent permettre dans un premier temps de supporter l'identification des propriétés pertinentes parmi celles des systèmes critiques et des systèmes interactifs cités dans ce chapitre, et dans un second temps de résoudre les conflits potentiels en ordonnant les propriétés.
De plus, l'évaluation des interactions en vue de minimiser les surprises liées à l'utilisation du système doit être intégrée à ces méthodes.

Les travaux de ce mémoire se situent à la croisée des domaines de la sûreté de fonctionnement et de l'ingénierie des systèmes interactifs. Plus particulièrement ils s'inscrivent dans une démarche de processus de développement sûr de fonctionnement. Ils viennent compléter les travaux de l'équipe ICS à l'IRIT, en spécifiant des briques méthodologiques qui s'inscrivent dans les processus de développement centrés utilisateurs destinés aux systèmes critiques tels que celui décrit par [Ladry, 2010], qui a notamment détaillé un processus de développement permettant la réalisation de systèmes interactifs multimodaux critiques.

Dans cette optique, les travaux de ce manuscrit s'inscrivent dans une démarche parallèle à celle effectuée sur la sûreté de fonctionnement appliquée aux systèmes interactifs critiques de Camille Fayollas [Fayollas, 2015]. Ils se concentrent sur les systèmes mono modaux et la fiabilisation de systèmes interactifs.

Les travaux de ce mémoire sont portés sur les méthodes de conception des systèmes multimodaux et multi-utilisateurs.

Nous allons détailler les moyens de conception des systèmes interactifs critiques dans le chapitre suivant.

Chapitre

2 L'ingénierie des systèmes interactifs critiques

Sommaire

2.1 Introduction

Les concepteurs de systèmes interactifs, critiques ou non, doivent garder présent à l'esprit l'ensemble des problématiques d'ingénierie que nous avons présentées au cours du chapitre précédent.

Certains outils et méthodes peuvent servir de guide tout au long de leur démarche. Il existe notamment des processus de développement, répondant le plus souvent à un seul type de problématique (par exemple la prise en compte de l'utilisabilité), ou conçus pour un domaine spécifique (par exemple l'aéronautique).

La première section de ce chapitre présente un tour d'horizon de ces processus.

La conception informatique nécessite le choix d'un langage de développement utilisé dans différentes phases du processus de développement. Le contexte de systèmes interactifs critiques dans lequel les travaux de ce mémoire se situent nous oblige à nous inscrire dans un cadre sûr de fonctionnement. Les besoins de notation formelle pour les systèmes interactifs critiques ont été décrits dans [Palanque and Bastide, 1994], et les normes de l'aéronautique imposent désormais leur utilisation.

La seconde partie de ce chapitre présente une notation formelle, adaptée aux contraintes des systèmes interactifs et des systèmes critiques.

Enfin, tout logiciel complexe est organisé selon une architecture permettant de décomposer le logiciel en un ensemble d'entités communicantes. Au delà de cet aspect de décomposition, certaines utilisations permettent d'assurer la présence de propriétés telles que la disponibilité ou la modifiabilité.

La troisième partie de ce chapitre présente les architectures des systèmes interactifs et des systèmes critiques.

2.2 Les Processus de développement

L'ingénierie des systèmes a nécessité la mise en place de processus d'approches et de recommandations pour le développement, permettant de systématiser le passage d'un problème vers sa solution. Ils permettent de rester dans un cadre conceptuel et ainsi de gérer la complexité, d'aider à la gestion de projet, et de faciliter la communication entre les différents intervenants. Plusieurs niveaux de processus et d'approches peuvent être présentés. Ils sont centrés sur l'organisation du travail, sur l'organisation des besoins, ou sur la gestion d'étapes plus fines pour résoudre des problèmes techniques particuliers.

Nous verrons au cours de cette section, trois grands types de processus de développement :

Les processus macro qui, en génie logiciel, permettent de gérer l'organisation du travail et des besoins

Les processus de développement dédiés aux systèmes interactifs qui se concentrent sur la relation entre les concepteurs d'un système et les utilisateurs du système pour maximiser, entre autre, l'utilisabilité

Les processus de développement dédiés aux systèmes critiques qui mettent l'accent sur les preuves et le rationnel nécessaires à l'obtention d'une certification

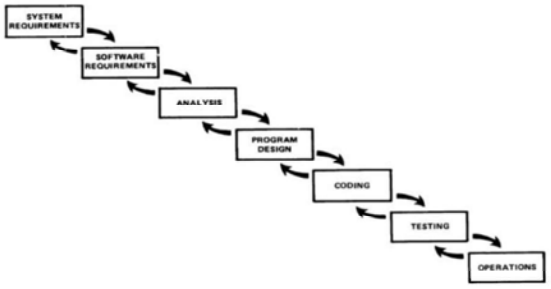
2.2.1 Les processus de développement en Génie Logiciel

Chaque projet génère des problématiques spécifiques, qu'elles relèvent de ressources humaines, de budget, de contraintes temporelles, etc. Cette diversité ne permettra à aucun processus de développement de résoudre intégralement des problématiques liées au développement de projets. Ceci explique que le domaine du génie logiciel a été très prolixe en matière de production de processus de développement. La figure 2.1 présente un échantillon de processus courants dans la littérature.

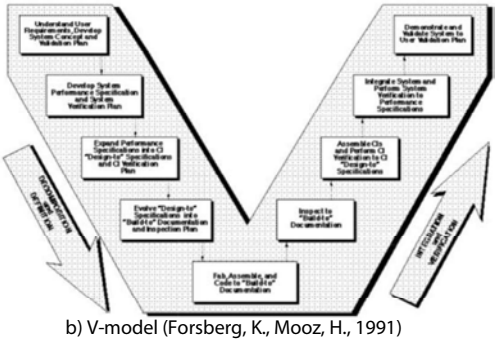
Le processus en cascade [Royce, 1987] (*a* de la figure 2.1) fait partie des plus anciens processus de développement informatique. Il liste les étapes allant de la spécification informelle du système jusqu'à son exploitation. C'est une séquence d'activités dont l'enchaînement ordonné permet à chaque activité de profiter des réflexions et des artefacts produits par la précédente. Largement utilisées dans les années 70 et 80, cette famille de processus est très critiquées [Gladden, 1982], [McCracken and Jackson, 1982] notamment parce qu'elle est basée sur l'hypothèse (largement remise en cause aujourd'hui) que les besoins sont stables et qu'ils sont définis une fois pour toutes au début du processus. Le principal problème de cette famille de processus de développement, si elle est appliquée aux systèmes interactifs, réside dans l'identification des problèmes d'utilisabilité. En effet, avec ce genre de processus, on risque de découvrir des problèmes que les utilisateurs rencontrent seulement au moment des phases d'opérations du système. On risque alors de devoir remonter l'intégralité de la cascade pour pouvoir résoudre, par exemple, des problèmes d'utilisabilité.

Le cycle de développement en V [Forsberg and Mooz, 1991] a été présenté initialement en 1983 (*b* de la figure 2.1). Sa particularité réside dans le fait que chaque étape du cycle descendant produit un artefact qui est mis en correspondance avec une étape d'évaluation du cycle montant. Les étapes de la phase descendante couvrent le raffinement des besoins jusqu'à l'implémentation de l'application. Le raffinement part des besoins les plus abstraits pour s'acheminer vers les plus concrets. Le bas du cycle correspond au codage de l'application qui fait suite à la conception. Ainsi, les étapes de la phase ascendante correspondent aux étapes d'évaluation de la phase descendante. La flexibilité et la renommée du cycle en V ont fait de ce processus un des plus utilisés au sein des entreprises. Les documents produits pour permettre l'évaluation de chaque étape servent à la fois à la rationalisation et à la gestion du projet. Cela permet de garder une trace des choix de conception et des exigences du système tout au long du cycle de développement.

Le modèle spirale [Boehm, 1986](*c* de la figure 2.1) correspond à une exécution en quatre étapes, dont le contenu évolue au fil des opérations, mais dont chacun des artefacts

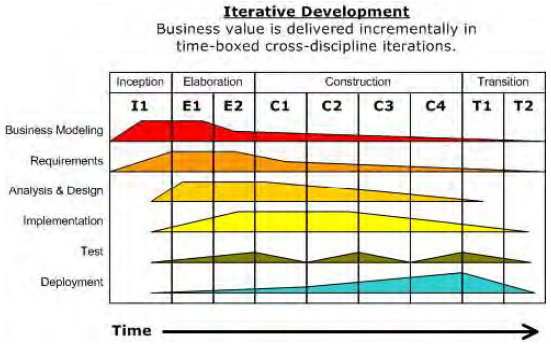


a) Cycle de vie en cascade (Royce, W. W., 1970)

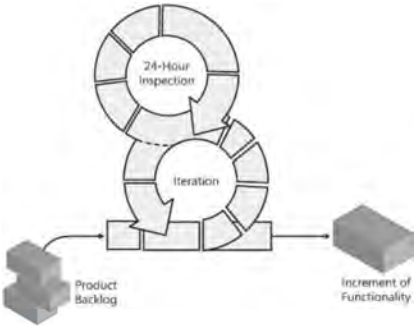


b) V-model (Forsberg, K., Mooz, H., 1991)

c) Spiral model of the software process (Boehm, B. 1986)



d) Rational Unified Process – RUP (IBM, 2003)



e) SCRUM skeleton (Shwaber & Beedle, 2002)

FIGURE 2.1 – Un échantillon des processus de développement courant en Génie Logiciel

conserve la même nature :

- Définition des objectifs, des alternatives possibles et des contraintes du projet
- Évaluation des alternatives en réponse aux besoins tout en tenant compte des contraintes : cette phase comporte un travail de prototypage, dont le résultat deviendra plus opérationnel au fur et à mesure des itérations
- Conception d'une solution apparaissant comme la meilleure des alternatives étudiées dans la phase précédente : cette phase englobe les étapes de conception et de vérification déjà observées dans les cycles de développement en cascade et en

V (spécification, conception préliminaire puis conception détaillée, tests unitaires puis tests d'intégration et déploiement)
- Planification de la phase suivante et notamment l'affectation des tâches à réaliser au sein de l'équipe de conception

Ce type de développement permet d'évaluer plusieurs alternatives répondant aux contraintes du projet. Lorsque la réflexion sur les alternatives a atteint un point suffisant, l'équipe peut alors prendre une décision et enchaîner sur un cycle de développement plus standard, tel que le cascade ou le V. Bien qu'ils prennent en compte l'utilité du système développé, ces types de développement sont longs et coûteux.

Le quatrième processus de développement présenté est le RUP pour Rational Unified Process [Kruchten, 2004](en *d* de la figure 2.1). Il permet d'assurer le développement d'un système correspondant aux besoins des utilisateurs. Il est itératif et incrémental, car découpé en phases de courte durée. Chaque phase produit une version exécutable de l'application qui sert de base à l'incrémentation suivante. Ce processus est basé sur la modélisation UML (*Unified Modeling Language*). Son architecture doit être modélisée graphiquement. L'axe horizontal correspond au temps de déploiement du cycle de vie qui comprend le lancement, l'élaboration, la construction, et la transition. L'axe vertical représente l'ensemble des disciplines nécessaires à la réalisation du projet, qu'elles relèvent de l'ordre technique ou organisationnel.

Le cinquième processus fait partie d'une des méthodes AGILE officialisées en 2001 par le Manifeste Agile [Beck et al., 2001] qui vise à inciter à la satisfaction du client. Elles impliquent le client ou parfois l'utilisateur final dans le processus de développement. Elles reposent sur quatre valeurs principales :

L'équipe : Personnes et interaction plutôt que processus et outils

L'application : Logiciel fonctionnel plutôt que documentation complète

La collaboration : Collaboration avec le client plutôt que négociation de contrat

L'acceptation du changement : Réagir au changement plutôt que suivre un plan

L'approche Scrum présentée en *e* sur la figure 2.1, suit ces principes.

L'ensemble des ces processus ne permet pas d'intégrer les problématiques spécifiques des systèmes interactifs détaillées à la section 1.4, et notamment les problématique liée à l'utilisabilité, l'efficacité ou la satisfaction de l'utilisateur. Ainsi, d'autres processus ont été créés dans cette optique.

2.2.2 Les processus de développement dédiés aux systèmes interactifs

Les approches et processus dédiés aux systèmes interactifs ajoutent des contraintes liées à la prise en compte des humains et plus particulièrement des utilisateurs ou opérateurs des systèmes, très différents des clients mentionnés par les processus AGILE. La conséquence la plus fréquente est l'ajout d'étapes telles que :

- Analyse des besoins de l'utilisateur, de son comportement et de ses activités pour concevoir un système en adéquation.
- Prototypage du système afin de le confronter à l'utilisateur avant d'engager des frais de développement, sans être auparavant sûr que le produit sera accepté dans sa globalité par l'utilisateur.
- Évaluation informelle des prototypes et du système en utilisation afin de s'assurer de l'adéquation entre l'utilisateur et le système.

Ces trois étapes forment la base de la conception centrée utilisateur.

[Dix, 2009] décrit les techniques courantes de conception centrée utilisateur. On pourra citer, de manière non exhaustive, des processus qui utilisent ces techniques :

Le processus en étoile : [Hartson and Hix, 1989] permet de passer d'une étape à une autre sans ordre particulier

Le processus de développement en couche : [Curtis and Hefley, 1994] qui définit précisément les relations entre conception classique et conception de l'interface

Le processus de développement en cercle : [Collins and Collins, 1995] accorde une grande part aux tâches utilisateur, donc plus généralement aux facteurs humains durant la conception

Le processus itératif cyclique : [Rauterberg, 1992] met l'accent sur l'utilisabilité tout au long du processus sans s'attarder sur la fiabilité.

Le design de système centré utilisateur *The usability design process*, présenté en figure 2.2 est aussi dédié à l'utilisabilité du système, tout au long du cycle de vie du projet, et non pas seulement pendant les phases de développement [Göransson et al., 2004]

Des recommandations existent pour les types d'étapes à intégrer dans un processus destiné aux systèmes interactifs, par exemple dans le standard ISO TR 18529 [ISO, 1997] ou encore dans l'ISO 13407[ISO, b](obsolète), qui a été intégrée depuis à la dernière révision de la norme ISO 9241 [ISO, a]. Elles sont bien moins nombreuses et exhaustives que les recommandations existantes dans le domaine de la sûreté de fonctionnement.

Bien que ces processus de développement permettent l'augmentation de l'utilisabilité ou fournissent des aides au développement de systèmes interactifs, aucun d'entre eux ne décrit la prise en compte des spécificités des systèmes critiques détaillés en section 1.5.

2.2.3 Les processus de développement dédiés aux systèmes critiques

Le développement des systèmes critiques requiert des étapes supplémentaires afin d'assurer la sûreté de fonctionnement par rapport aux systèmes classiques. Des étapes de vérification, de validation, de preuves ou de certification sont nécessaires. La variété des systèmes critiques a engendré plusieurs normes de standardisation :

La norme CEI-61508[CEI, c], est une norme générique traitant de sûreté de fonctionnement. Elle a été déclinée en plusieurs autres normes : l'ISO 26262 [ISO, c]

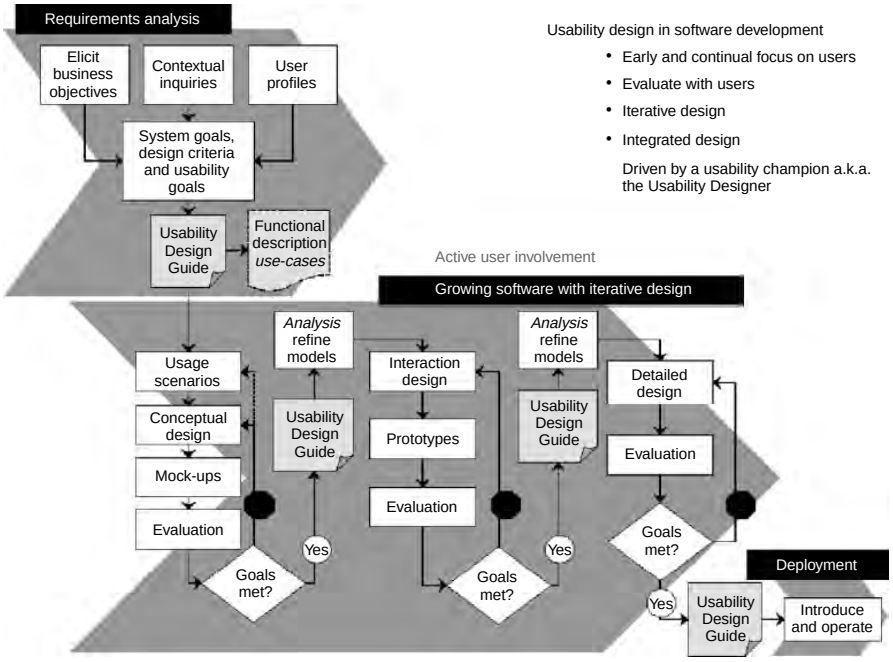


FIGURE 2.2 – The usability design process [Göransson et al., 2004]

s'adresse à l'automobile, la norme CEI 61511 est centrée sur les procédés industriels, la norme CEI 61513 [CEI, a] vise le secteur du nucléaire, la norme CEI 62061 [CEI, b] fait référence à la sécurité des machines et est utilisée dans le domaine ferroviaire.

En aéronautique, les processus de développement logiciel sont imposés par des normes de plus haut niveau permettant d'obtenir le certificat de navigabilité définie par la CS-25 [EASA,]. Les dérivées de la CEI-61508 ne sont donc pas utilisées. C'est le processus de développement de la norme DO 178C[RTCA, a] qui sert de référence. Ce standard est régulièrement mis à jour, le C étant la troisième révision. Au vu du contexte applicatif de ce mémoire, nous présentons le processus de cette dernière.

Quatre phases le composent :

- La phase d'analyse des exigences (« SW Requirements process ») qui produit les exigences logicielles de haut niveau (HLR)
- La phase de design (« SW Design process ») qui produit les exigences logicielles de bas niveau (LLR) ainsi que l'architecture du logiciel à partir des exigences de haut niveau(HLR)
- La phase de codage (« SW Coding process ») qui produit le code source et le code de l'application
- La phase d'intégration (Integration Process) qui produit le code exécutable et lie le logiciel au système intégré.

Les exigences logicielles de haut niveau (HLR) sont produites directement par

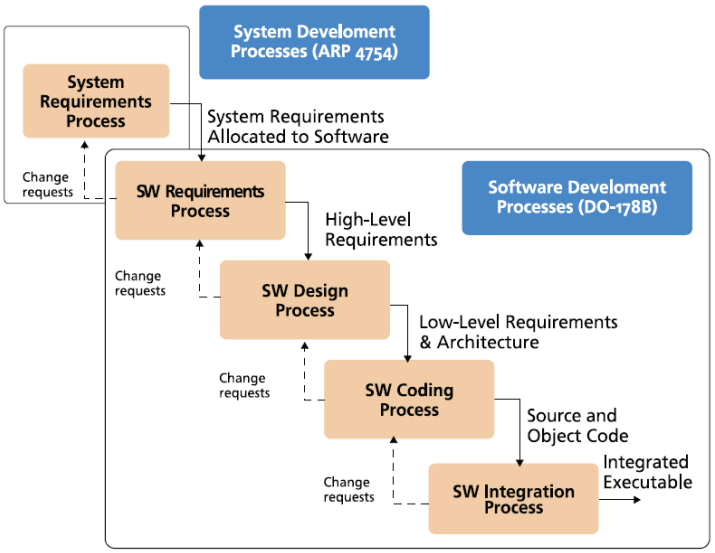


FIGURE 2.3 – Processus de développement DO-178

l'analyse des exigences du système et l'architecture du système. Elles comportent des spécifications d'exigences fonctionnelles et opérationnelles, les contraintes de mémoire, interfaces matérielles et logicielles, les détections de panne et les exigences de sécurité. Les HLR sont ensuite développées au cours du processus de conception de logiciels, produisant ainsi l'architecture logicielle et les exigences de bas niveau (LLR). Il s'agit notamment des descriptions des entrées / sorties, les données et les flux de contrôle, la limitation des ressources, la planification et des mécanismes de communication, ainsi que des composants logiciels. Grâce au processus de codage, les exigences bas-niveau sont mises en œuvre sous forme de code source. Le code source est compilé et lié par le processus d'intégration en un code exécutable lié à l'environnement cible. À toutes les étapes, la traçabilité est requise : entre les exigences du système et HLR, HLR et LLR, entre LLR et le code, et aussi entre les essais et les exigences. De plus, le standard DO-178 fournit des lignes directrices pour les processus de vérification du logiciel, de gestion de la configuration du logiciel et d'assurance qualité du logiciel. La phase d'architecture ne mentionne pas les exigences spécifiques d'un système multimodale.

Les responsables de la conception et du développement d'un système critique doivent démontrer que le système est suffisamment sûr pour que les autorités de certification émettent l'autorisation de mise en circulation du système. [Storey, 1996] indique trois aspects techniques importants pour accéder aux phases de certification :

- La démonstration que les risques majeurs ont été identifiés et pris en compte.
- La preuve que le système est conforme aux standards en vigueur.
- L'argumentation montrant que le système est sûr et qu'il le restera au cours de son cycle de vie.

Les étapes-clés des approches de développement des systèmes critiques fournissant un support pour les phases de certification sont : la vérification, la validation et le test.

- La vérification sert à montrer que les ressources produites par une phase du cycle de développement sont conformes aux exigences soumises en entrée de cette phase du cycle.
- La validation sert à montrer que les exigences émises au début du cycle de vie du cycle de développement sont appropriées et en cohérence avec les besoins du client ou de l'utilisateur.
- Le test est une manière de vérifier et de valider tout ou partie du système (les revues de spécifications et de code sont faites de manière différente). Dans le cas d'un système interactif, il est impossible de couvrir toutes les possibilités de combinaisons d'interactions avec le test. Ainsi, la modélisation formelle est une activité de plus en plus utilisée et faisant l'objet de nombreuses recherches [Miller et al., 2006].

Depuis la révision DO178C en 2012, plusieurs suppléments ont été ajoutés à la norme :

Le supplément 330 [RTCA, b] dicte les règles pour *qualifier les outils de développement*, et donc, non seulement le code déployé à la fin, mais les moyens utilisés pour le générer durant la conception.

Le supplément 331 [RTCA, c] consolide l'utilisation de la *modélisation* pour améliorer la qualité du développement et de la vérification.

Le supplément 332 [RTCA, d] définit les conditions dans lesquelles le paradigme de *programmation orienté objet* peut être utilisé.

Le supplément 333 [RTCA, e] impose l'utilisation des *méthodes formelles* pour compléter le test.

L'ajout de ces 4 suppléments entraine donc de nouvelles contraintes qui étaient jusque là définies, au mieux, comme simples recommandations. L'obtention des certifications nécessaires aux autorisations de vols doit satisfaire les exigences des nouvelles normes. En restant dans un environnement de développement outillé permettent l'utilisation du paradigme de programmation par objet, on doit utiliser à la fois des méthodes formelles et la modélisation.

2.3 Notations formelles pour la spécification des systèmes interactifs : focus sur ICO

2.3.1 Les besoins en notation pour la description de systèmes interactifs multimodaux

La conception et le développement des systèmes interactifs destinés aux environnements critiques implique une modélisation concise, complète et non ambigüe de leur comportement par le biais d'une notation formelle. Il est important de noter que cette modélisation devra, dans un second temps, être complétée par des approches de vérification

et de validation. Le besoin de notation formelle pour la modélisation du comportement des systèmes interactifs a été exprimé et étudié dans de nombreux travaux tels que ceux de [Palanque and Bastide, 1994], [Johnson et al., 1995], [Thimbleby, 2007] ou [Dix, 1995]. Ils ont permis de mettre en évidence l'intérêt de l'utilisation de notations formelles pour la description du comportement des systèmes en mettant en avant un certain nombre d'avantages non spécifiques aux aspects interactifs du système :

- Décrire le comportement des systèmes de manière complète et non ambigüe.
- Pratiquer des activités d'analyse et de vérification sur les modèles et donc sur le comportement du système.
- Améliorer la communication entre les différents acteurs du cycle de développement.

Une notation pour la description comportementale des systèmes interactifs doit avoir un pouvoir d'expression permettant de décrire les aspects spécifiques aux systèmes interactifs que nous listons ci-dessous :

- La description des objets et de leur valeur
- La description des états
- La représentation des événements
- La représentation des aspects temporels
- La représentation des comportements concurrents et de l'instanciation dynamique
- La description de la présentation et des activations qui font suite à une action de l'utilisateur

En plus de ces besoins de pouvoir d'expression, nous nous intéressons à une notation formelle permettant de décrire tous les composants logiciels de notre architecture. Celle-ci doit également être outillée afin de permettre la description de systèmes complexes tels que les applications interactives dans les cockpits d'avions. [Cuenca et al., 2014] compare les différents paradigmes, tels que événements-réponse, diagramme d'état, diagramme centré sur un processus pour l'IHM et [Hamon-Keromen, 2014] fait un inventaire des besoins liés aux interactions Post WIMP dans ce domaine. Il résume dans un tableau présenté en figure 2.4, l'expressivité des notations existantes au regard des besoins exprimés.

Les besoins en terme de notation de nos travaux sont identiques à ceux exprimés dans [Hamon-Keromen, 2014]. Nous en tirerons la même conclusion quant la sélection d'une notation formelle : les exigences nous amènent à choisir la notation ICO.

2.3.2 ICO

Les ICO (pour Interactive Cooperative Objects) [Palanque, 1992] sont une notation s'appuyant sur l'utilisation des réseaux de Petri à objets. C'est une notation formelle supportée par un outil d'édition de modèles et d'interprétation appelé Petshop. Ce mémoire n'apporte pas de contribution à la notation ni à l'outil Petshop, ICO étant développé dans l'équipe de recherche ICS-IRIT. Nous présentons ici la notation et l'outil pour permettre au lecteur d'avoir accès aux modèles qui sont présentés dans les chapitres suivants. La présentation du langage et de l'outil associé est extraite de la thèse de Camille Fayollas [Fayollas, 2015], plus précisément de la partie 5.2 de son manuscrit :

FIGURE 2.4 – Expressivité des Notations

La notation ICO s'appuie sur les concepts de la programmation orientée-objet pour décrire les aspects structurels des systèmes interactifs et sur les concepts des réseaux de Petri haut-niveau Genrich [Genrich, 1991] pour décrire les aspects comportementaux. Une première version de cette notation a été définie par les travaux de [Palanque, 1992] et s'appuie sur l'utilisation des réseaux de Petri à objets comme présentés dans les travaux de [Bastide and Palanque, 1990].

Nous présentons dans ce document la dernière version de la notation telle qu'elle a été définie par les travaux de [Hamon-Keromen, 2014].

Pour mieux comprendre la notation ICO, nous présentons tout d'abord les réseaux de Petri sur lesquels elle s'appuie.

Nous présentons ensuite la notation CO (pour Cooperative Objects ou Objets Coopératifs) dont la notation ICO est une extension.

Enfin, nous présentons la notation ICO. Seuls les éléments essentiels à la compréhension de la notation et à son utilisation dans le cadre de ce mémoire seront présentés. Une présentation plus complète de ces trois notations (réseaux de Petri, CO et ICO) peut-être trouvée dans les travaux de Hamon [Hamon-Keromen, 2014].

Il est important de noter que les différentes illustrations présentées dans ce chapitre correspondent à celles fournies par l'outil PetShop qui permet notamment d'éditer les modèles ICO.

2.3.2.1 Les réseaux de Petri et les réseaux de Petri haut niveau

Les réseaux de Petri (Petri 1962) sont une technique de description formelle décrivant le comportement dynamique des systèmes à événements discrets. Concrètement, ils forment un langage graphique s'appuyant sur un modèle mathématique qui permet de modéliser explicitement les notions d'état et de changement d'état en prenant en compte un nombre infini d'états.

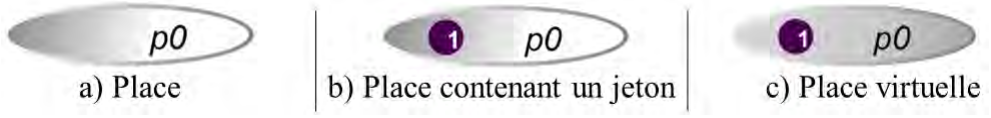


FIGURE 2.5 – Représentation des places dans les réseaux de Petri

Un réseau de Petri est un graphe comprenant deux types de nœuds (les places et les transitions) reliés par des arcs orientés. Les places sont représentées par des ellipses (voir Figure 2.5) et permettent de modéliser les différents états du système. Elles peuvent contenir des jetons qui sont représentés par des cercles violets marqués d'un chiffre représentant le nombre de jetons présents dans la place (voir Figure 2.5). Ceux-ci représentent les variables d'état du système. L'état d'un réseau de Petri à un instant donné (et donc du

système associé) est défini par son marquage, c'est-à-dire par la distribution et la valeur des jetons dans ses places.

FIGURE 2.6 – Représentations des transitions franchissables et non franchissables dans les réseaux de Petri

L'évolution de l'état d'un réseau de Petri est conditionnée par le franchissement des transitions. Les transitions sont représentées par des rectangles (voir Figure 2.6) et permettent de caractériser l'évolution des jetons en fonction du marquage du réseau et du parcours défini par les arcs.

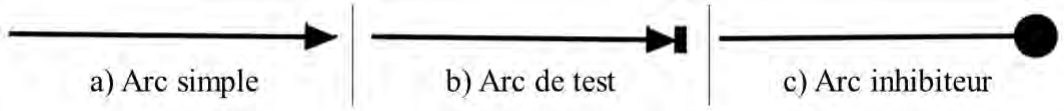


FIGURE 2.7 – Représentation des différents types d'arcs dans les réseaux de Petri

Les arcs sont des arcs orientés (voir Figure 2.7) qui permettent de relier une place à une transition et inversement. Ils permettent de définir le parcours des jetons en conditionnant la franchissabilité des transitions. Ainsi, une transition est franchissable si et seulement si le nombre de jetons dans ses places d'entrées (places reliées par un arc entrant à la transition) est supérieur ou égal au poids des arcs simples (voir Figure 2.7) reliant respectivement chacune des ces places à cette transition. Une transition franchissable est représentée par un rectangle violet, alors qu'une transition non franchissable est représentée par un rectangle gris (voir Figure 2.7). Le franchissement d'une transition ne met en jeu que les jetons contenus dans les places d'entrée et de sortie. Si une transition est franchissable, elle est immédiatement franchie. Le franchissement d'une transition consomme un jeton présent dans la place d'entrée (le retirant ainsi de la place d'origine) et dépose un jeton dans la place en sortie. Ce comportement est illustré en Figure 2.8. Le nombre de jetons consommés dans la place d'entrée et déposés dans la place de sortie dépend du poids des arcs associés.

FIGURE 2.8 – État d'un réseau de Petri avant et après un franchissement de transition

Les places peuvent être reliées aux transitions par des arcs spéciaux, ceux-ci modifient alors les conditions de franchissabilité des transitions. Nous retiendrons ainsi les arcs

de test et les arcs inhibiteurs (voir Figure 2.7). Les arcs de test rendent une transition franchissable si et seulement si la place en entrée contient au moins un jeton. Celui-ci ne sera pas consommé par le franchissement de la transition. Ce comportement est illustré en Figure 2.9.

FIGURE 2.9 – État d'un réseau de Petri avant et après le franchissement de transition conditionnée par un arc de test

Les arcs inhibiteurs rendent une transition franchissable si et seulement si la place en entrée ne contient pas de jeton. Ce comportement est illustré en Figure 2.10.

FIGURE 2.10 – État d'un réseau de Petri avant et après le franchissement de transition conditionnée par un arc inhibiteur

Les réseaux de Petri temporisés [Ghezzi et al., 1989]) permettent de modéliser le temps de manière quantitative au travers de l'utilisation de transitions temporisées. Une transition temporisée est franchie seulement après un certain temps après être devenue franchissable. Ce temps est défini en millisecondes et présenté entre crochets. Ce comportement est illustré en Figure 2.5 où l'on peut voir que la transition t0_ n'est franchie qu'après 200ms.

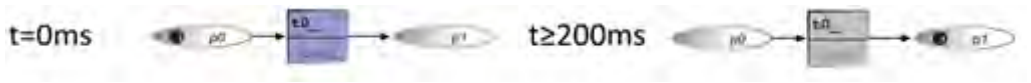


FIGURE 2.11 – État d'un réseau de Petri avant (a) et après (b) le franchissement d'une transition temporisée

Par construction, plusieurs transitions peuvent être en conflit en ayant les mêmes places et conditions d'entrées comme illustrées en Figure 2.12. Dans ce cas, le réseau sera indéterministe : le choix de la transition franchie sera fait de manière aléatoire.

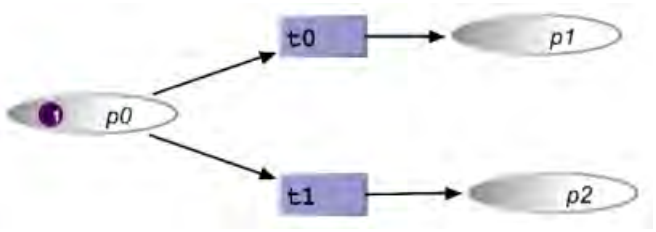


FIGURE 2.12 – Exemple de comportement indéterministe dans les réseaux de Petri

Les réseaux de Petri à objet (ou OPN pour Object Petri Nets), définis par les travaux de (Jensen 1987), introduisent les caractéristiques suivantes :

- Les jetons peuvent être des références à des objets ;
- Les arcs sont étiquetés par des noms de variables permettant de décrire le flot des objets ;
- Les transitions peuvent contenir des actions qui sont effectuées lors du franchissement ;
- La franchissabilité d'une transition peut être conditionnée (en plus de la présence ou l'absence de jetons) par une précondition portant sur la valeur de ces jetons ;
- La franchissabilité d'une transition peut également être conditionnée par la propriété d'unification.

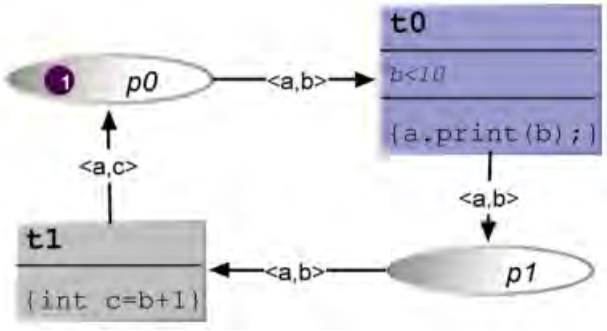


FIGURE 2.13 – Exemple de réseau de Petri à objets

La Figure 2.13 présente un exemple de réseau de Petri à objets. Les jetons circulant dans le réseau sont des couples (a,b) (respectivement (a,c)) constitués d'un objet a ayant une méthode print(int) et d'un entier b (respectivement c). La transition t0 possède une précondition sur la valeur de l'objet b, elle est franchissable si et seulement si la valeur de l'entier b contenu dans le jeton de la place p0 est strictement inférieure à 10 (c'est le cas sur la figure) ; cette transition possède également une action a.print(b) qui exécute la méthode print de l'objet a avec comme paramètre l'entier b. La transition t1 ne possède pas de précondition, mais possède une action : int c = b +1. Supposons que dans l'état initial le jeton de la place p0 contienne un objet x du bon type et d'un entier 0, autrement

dit, un couple <x,0>. Lors du franchissement de la transition t0, l'action print(0) est effectuée sur l'objet x et le jeton <x,0> est transmis à la place p1. La transition t1 est alors franchissable. Lors de son franchissement, un jeton est transmis à la place p0, contenant les valeurs d'objets modifiés suite à l'action effectuée lors du franchissement de t1, autrement dit <x,1>. La transition t0 est alors de nouveau franchissable, tant que la valeur de b est inférieure à 10 (c'est-à-dire jusqu'à ce que la transition t1 ait été franchie dix fois).

FIGURE 2.14 – Exemple de franchissement dans un réseau de Petri à objets avant (a) et après (b) unification

La Figure 2.14 illustre la propriété d'unification. Lorsque les arcs d'entrée d'une transition sont étiquetés avec un nom de variable commun, la transition est franchissable si et seulement si les places d'entrée contiennent chacune un jeton ayant une valeur identique pour cette variable étiquetée. C'est ce comportement qui est reporté en Figure 2.14 où la valeur de la variable a est égale pour le jeton contenu dans la place p0 et pour celui contenu dans la place p1 ; si les deux jetons n'avaient pas contenu une valeur de variable a égale, la transition t0 n'aurait pas été franchissable.

2.3.2.2 Le formalisme ICO

Le formalisme des Objets Coopératifs est constitué d'un ensemble de classes (les CO-classes) qui définissent des objets coopératifs (les instances). Une CO-classe est constituée d'une interface Java décrivant les services qu'elle propose et d'un réseau de Petri à objets décrivant son comportement ; celui-ci est appelé ObCS (pour Structure de Contrôle de l'Objet ou Object Control Structure). La communication entre les objets coopératifs est rendue possible par deux moyens : l'utilisation d'un protocole de communication de type client-serveur (en utilisant des appels des services définis par l'interface Java) ; ou l'utilisation d'une communication par événements.

2.3.2.3 Les services dans les objets coopératifs

L'interface logicielle Java de la CO-classe permet de définir les services offerts par la CO-classe. L'exécution de ces services est synchrone. Pour chacun de ces services, trois places sont générées dans l'ObCS (le réseau de Petri) correspondant à la CO-classe : une place d'entrée (SIP), une place de sortie (SOP) et une place d'exception (SEP). L'appel du service consiste à déposer un jeton contenant les paramètres de l'appel dans la place d'entrée du service (SIP). Rendre le service revient à déposer un jeton contenant le résultat

de la requête dans la place de sortie du service (SOP). Enfin, la notification d'une erreur d'exécution du service revient à déposer un jeton décrivant l'erreur de la requête dans la place exception (SEP). La Figure 2.15 présente un exemple de CO-classe proposant une méthode add(int a,int b) réalisant l'addition de deux entiers a et b. Lors d'un appel de cette méthode, un jeton est déposé dans la place SIP_add. La transition add est alors franchie et réalise l'action c = a + b. Un jeton contenant la valeur de c est alors placé dans la place SOP_add, rendant ainsi le service.

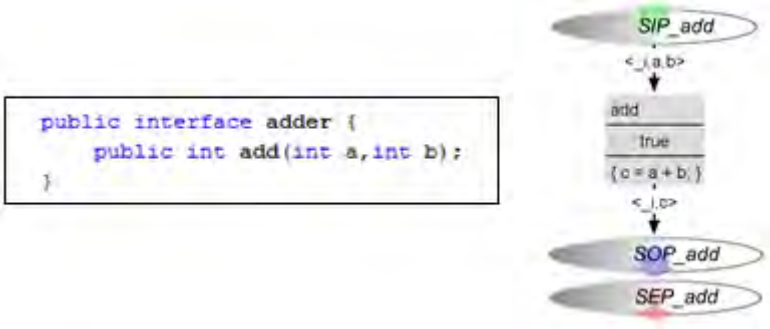


FIGURE 2.15 – Exemple de CO-classe comprenant son interface Java et son ObCS

2.3.2.4 Les événements dans les objets coopératifs

En plus de la notion de services permettant la communication synchrone et unicast (un à un) entre objets coopératifs, la notion d'évènements a été intégrée dans les Objets Coopératifs afin de permettre la communication asynchrone et multicast (vers plusieurs autres modèles). Cette communication est rendue possible grâce à des abonnements, une syntaxe pour le postage d'événements et un type de transitions permettant la réception d'un événement. La syntaxe d'abonnement à un événement ainsi que son envoi sont décrits en Figure 2.16.

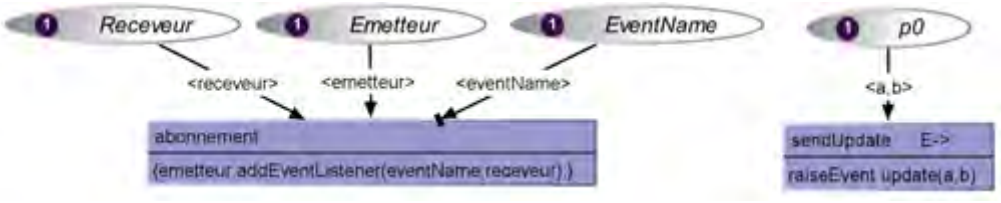


FIGURE 2.16 – Abonnement à un événement (a) et envoi d'un événement (b)

La Figure 2.17 présente un exemple de transition réceptrice d'événement. Cette transition a été créée pour recevoir les événements update provenant de l'objet émetteur. Elle reçoit un ensemble de paramètres provenant de cet événement appelé eventParams

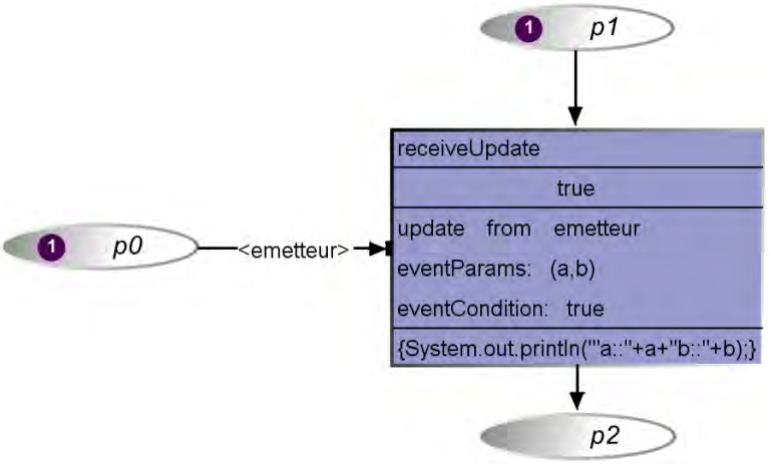


FIGURE 2.17 – Exemple de transition réceptrice d'événements

(a et b dans cet exemple). Elle est conditionnée par deux préconditions : la première est du même type que les préconditions des transitions classiques (elle est réglée à true dans cet exemple) ; la seconde, appelée eventCondition, peut dépendre des paramètres transmis par l'événement (elle est également réglée à true dans cet exemple). Enfin, cette transition peut effectuer une action à la manière des transitions classiques (System.out.println("a : :"+a+" b : :"+b) ; dans cet exemple).

Nom	**Événement**
PN_TokenAdded	Un jeton est entré dans la place PN
PN_TokenRemoved	Un jeton est sorti de la place PN
PN_MarkingReset	Le marquage de la place PN a été réinitialisé
TN_TransitionCompleted	La transition TN a été franchie
EN_Enabled	L'event handler de l'événement EN a été activé
EN_Disabled	L'event handler de l'événement EN a été désactivé

FIGURE 2.18 – Événements correspondants aux changements d'états d'un objet coopératif. PN : Place Name, TN : Transition Name, EN : Event Name

La notation, associée à l'outil PetShop et son interpréteur permet également l'abonnement à des événements relatifs aux changements d'états d'un objet coopératif. Ces événements sont décrits dans le Tableau sur la figure 2.18.

2.3.2.5 Les Objets Coopératifs Interactifs (ICO)

Les Objets Coopératifs Interactifs (ICO) sont une extension des CO (Objets Coopératifs). Cette extension permet de décrire les aspects interactifs des systèmes modélisés :

- Le comportement des différents composants du système est décrit grâce à un ensemble de CO-Classes.
- Le lien avec l'apparence graphique du système interactif est décrit grâce à la partie présentation.

Le lien entre le comportement du système et l'évolution de son rendu graphique se fait grâce à deux types de fonctions : les fonctions de rendu et les fonctions d'activation. Elles permettent de maintenir la cohérence entre l'état de la CO-Classe et son apparence. Dans un réseau de Petri, les places sont les variables d'état du système (l'état du système étant modélisé par une distribution de jetons dans les différentes places : le marquage) et les transitions sont les opérateurs de changement d'état. Les arcs, quant à eux, représentent les conditions prérequises aux changements d'état et leurs effets sur les états du système. La fonction de rendu met en relation les places du comportement de l'application avec l'information affichée au moyen de trois types de méta événements associés à trois cas d'évolution de l'état du système :

- L'entrée d'un jeton dans une place (événement *PlaceName_ TokenAdded*).
- La sortie d'un jeton d'une place (événement *PlaceName_ TokenRemoved*).
- La réinitialisation du marquage d'une place (événement *PlaceName_ MarkingReset*).

La fonction d'activation a un double emploi. Elle permet premièrement de décrire le lien entre la disponibilité d'un service de l'application et la possibilité d'accomplir une action sur un objet de l'interface graphique. Deuxièmement, elle permet de décrire le lien entre les actions enregistrées sur l'interface graphique et le comportement de l'application. Elle utilise pour cela, deux types de méta événements associés à deux cas d'évolution de l'état du système :

- La disponibilité nouvelle d'un service utilisateur (événement *EventName_ Enabled*).
- L'indisponibilité nouvelle d'un service utilisateur (événement *EventName_ Disabled*).

Les méta événements peuvent être utilisés comme des événements classiques pour déclencher des évolutions du système. La partie rendu des exemples de ce mémoire étant principalement basée sur ces mécanismes, nous les illustrerons en détail dans la troisième partie de ce mémoire, notamment dans le chapitre 6.

2.3.2.6 Synthèse sur les ICO

La présentation de cette notation confirme la possibilité de modéliser tous les aspects essentiels à la description du comportement des systèmes interactifs que nous avons listé :

- La description des objets et de leur valeur est supportée par le fondement de la notation ICO sur les objets de Petri à objets.
- La description des états est supportée par le fondement de la notation ICO sur les réseaux de Petri : un état correspond au marquage dans le réseau.
- La représentation des événements est supportée par les aspects spécifiques de la notation ICO et l'envoi et la réception d'événements.

- La représentation des aspects temporels est supportée par les aspects spécifiques de la notation ICO et les transitions temporisées.
- La représentation des comportements concurrents et de l'instanciation dynamique est supportée par le fondement de la notation ICO sur les réseaux de Petri et la diversité des systèmes qu'ils permettent de modéliser.

Enfin, du fait de son fondement sur les réseaux de Petri, la notation ICO rend possibles différents moyens d'analyse formelle des modèles ICO. Ces moyens permettent de supporter les activités de vérification et de validation du logiciel.

La notation formelle seule ne suffit pas à produire un logiciel fiable, et, pour supporter un logiciel complexe, il faut mettre en place une architecture, garante de certaines propriétés.

2.4 Les architectures logicielles

Réaliser une architecture logicielle peut avoir plusieurs sens, et plusieurs buts. Len Bass définie dans son livre « software architecture in practice » [Bass, 2007] :

Une architecture logicielle d'un système est l'ensemble des structures nécessaires pour raisonner sur ce système, ce qui comprend les composants logiciels, les relations entre ceux-ci, et leurs propriétés.

Cette définition ne met pas en œuvre de temps ou de date dans un projet, où il faudrait définir l'architecture, ni même la flexibilité qu'elle pourrait subir dans le temps. Par exemple, sur le processus AGILE, l'architecture est amenée à évoluer à chaque itération.

Tous les concepteurs de systèmes complexes réalisent donc des architectures à un moment ou à un autre. Il n'existe cependant pas de standards pour réaliser des modèles d'architectures génériques.

Nous allons faire un tour d'horizon des architectures dans le domaine des IHM et des systèmes critiques, en débutant par une brève présentation d'une notation pour les architectures.

2.4.1 Préliminaires

2.4.1.1 AADL, un langage de description d'architecture logicielle

Le périmètre des travaux n'inclue pas les langages de description d'architectures. Néanmoins, dans la mesure où nous allons proposer un modèle d'architecture générique de système interactif critique, nous allons présenter AADL, une notation largement utilisée dans l'aéronautique.

AADL [Feiler et al., 2006] pour *Architecture Analysis and Design Language* est un langage de description d'architectures pour les systèmes embarqués. Il a été conçu pour permettre de spécifier les tâches et les communications dans les architectures destinées aux

systèmes critiques sécurisés, embarqués, temps réel, tolérant aux fautes, ou d'utilisation intense.

AADL permet de modéliser des architectures partitionnées, et en particulier celles qui suivent le standard ARINC 653 qui régit les systèmes d'exploitation embarqués dans l'aéronautique, que nous présentons en section 2.4.4.2.

Ce langage de description étant spécifique aux systèmes embarqués, peu de travaux l'utilisent, et aucun dans le domaine des systèmes interactifs. Il est destiné avant tout à produire des architectures instanciables. Il n'existe par ailleurs aucun formalisme ou consensus pour l'élaboration des modèles d'architectures génériques applicables par la suite à un système spécifique.

Il existe trois représentations d'AADL, la textuelle, l'XML et la graphique. Nous présentons en figure 2.19 les composants graphiques AADL que nous utiliserons dans les chapitres suivants.

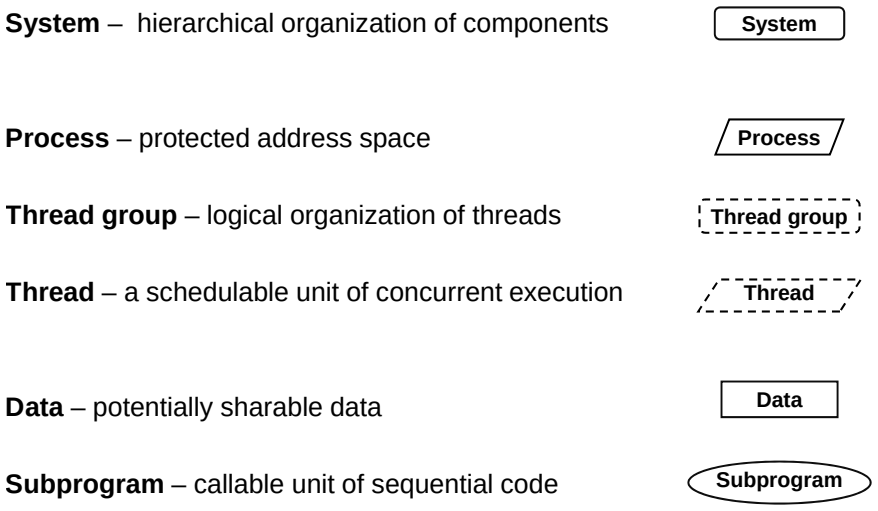


FIGURE 2.19 – Application Software Components, extrait de [Feiler et al., 2006]

2.4.1.2 Principes d'une architecture autotestable

Bien que la définition et la validation de nouveaux mécanismes de sûreté de fonctionnement ne fassent pas partie du périmètre de cette thèse, nous allons expliquer succinctement quelques principes qui ajoutent certaines contraintes à l'ingénierie des systèmes interactifs critiques.

Afin de tolérer des fautes , il faut tout d'abords pouvoir les détecter. Nous allons présenter dans cette section une architecture de détection de fautes. Nous ne nous intéresserons pas aux mécanismes de recouvrement qui sont en dehors du périmètre de ce mémoire.

Un composant autotestable [Laprie et al., 1995], est capable de vérifier son propre fonctionnement et de notifier une erreur si celui-ci n'est pas correct. Pour cela, un sous-composant est chargé de vérifier le comportement et l'exécution du sous-composant fonctionnel (le sous-composant que l'on souhaite rendre tolérant aux fautes). Le mécanisme de tolérance aux fautes autotestable est également appelé COM-MON, notamment dans l'industrie avionique [Traverse et al., 2004]. Cette dénomination provient de la redondance du sous-composant fonctionnel (le sous-composant COM pour commande) par le sous-composant qui vérifie son exécution (le sous-composant MON pour moniteur).

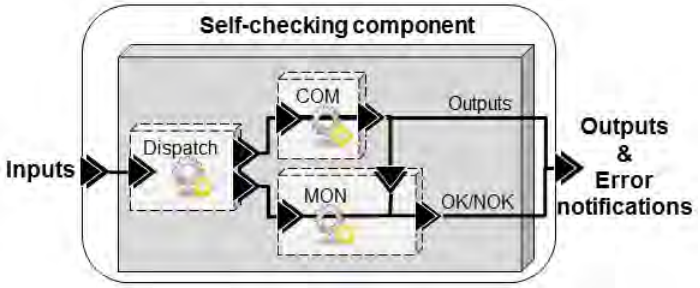


FIGURE 2.20 – Architecture logicielle d'un composant autotestable (Fayollas 2015)

La Figure 2.20 présente l'architecture logicielle correspondant à un composant autotestable. Elle nous permet de mettre en évidence les trois sous-composants qui le constituent :

Le composant COM (COMmande) est le composant fonctionnel, celui que l'on souhaite rendre tolérant aux fautes.

Le composant MON (MONitor) est le composant en charge de la vérification du comportement du composant COM.

Le composant Dispatch est le composant en charge de délivrer les entrées du composant autotestable aux composants COM et MON.

L'instanciation de cette architecture soulève deux problèmes de réalisation : celle du composant Dispatch et celle du composant MON. Nous identifions trois manières différentes de réaliser ce dernier :

Redondance : le composant MON est composé d'une copie du composant COM et d'un composant comparateur en charge de vérifier la similitude des résultats du COM et de sa copie.

Diversification : le composant MON est composé d'une variante du COM et d'un composant comparateur en charge de vérifier la similitude des résultats du COM et de sa variante.

Contrôleur d'assertion : le composant MON est un contrôleur d'assertions. Une assertion est une expression logique décrivant une partie du comportement attendu du composant COM. Un contrôleur d'assertion est un test booléen permettant de

vérifier la concordance entre les assertions et les résultats du composant COM. Le résultat du test prend la valeur vraie si la vraisemblance est validée. Dans le cas contraire, il prend la valeur fausse et un signal d'erreur est déclenché.

Il est important de rappeler qu'un composant autotestable ne permet que la détection des erreurs et ne permet pas leur recouvrement. Il peut-être effectué de plusieurs manières. Par exemple, un composant spécifique peut être développé pour traiter les erreurs détectées. Ou encore, on peut utiliser de la redondance de composants autotestables, ce qui correspond à une architecture n-autotestable [Yeh, 1996]. On confie alors le rôle applicatif à un composant identique tandis que le composant défaillant devient silencieux pour éviter les interférences. En effet, un composant défaillant pourrait continuer de fournir un flux d'informations au système. Il est primordial que les composants autotestables défaillants soient silencieux lors de l'occurrence d'une faute.

2.4.2 Les architectures dédiées aux systèmes interactifs

2.4.2.1 Seeheim

Le modèle d'architecture Seeheim [Pfaff and ten Hagen, 1985] est le premier modèle d'architecture qui structure les applications interactives en trois composants logiques correspondants aux trois composantes décrites ci-dessus. La Figure 2.21 présente ce modèle et permet de mettre en évidence ces trois composants :

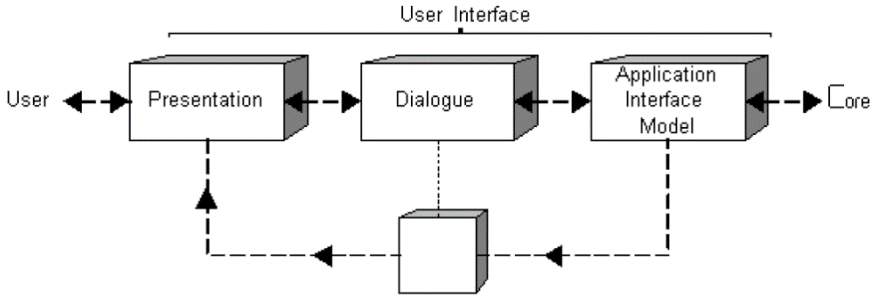


FIGURE 2.21 – Modèle d'architecture Seeheim (Pfaff 1985)

La présentation : correspond à la composante lexicale. Elle permet de gérer l'interaction avec l'utilisateur en interprétant les actions de celui-ci et en générant les sorties qu'il peut percevoir.

Le dialogue : correspond à la composante syntaxique. Il permet de gérer les échanges entre l'utilisateur et le système en maintenant une représentation graphique de l'état du système et des actions rendues possibles à l'utilisateur.

L'interface au noyau fonctionne : correspond à la composante sémantique. Elle permet de convertir les actions de l'utilisateur en appels de fonctions sur le noyau fonctionnel ainsi qu'une présentation de l'état du noyau fonctionnel à l'utilisateur.

En plus de ces trois composants, ce modèle introduit un composant supplémentaire, plus abstrait, permettant de représenter le retour sémantique rapide qui est rendu à l'utilisateur. Celui-ci est plus communément connu sous sa dénomination anglaise, *feedback*, et représente les modifications immédiates du rendu graphique telles que la mise à jour du manipulateur de souris (du curseur graphique).

2.4.2.2 ARCH/Slinky

Le modèle architectural ARCH et son méta-modèle Slinky [Kazman and Bass, 1994] ont été développés en 1992 par un groupe de travail désireux de combler les lacunes des modèles architecturaux existants pour les systèmes interactifs. Ils étendent le modèle architectural Slinky. La Figure 2.22 représente le modèle architectural ARCH qui décompose les systèmes interactifs en cinq composants que nous retrouvons sur la figure de gauche à droite :

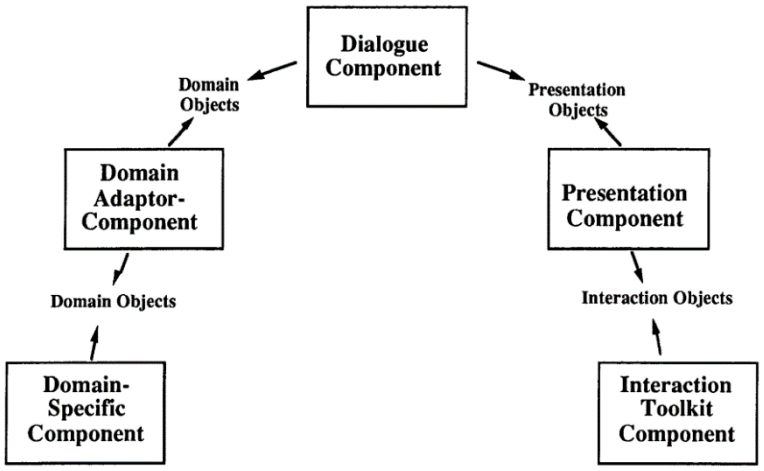


FIGURE 2.22 – Le modèle architectural ARCH (Bass, et al. 1992)

Le noyau fonctionnel : encapsule les fonctions non interactives de l'application. Il contrôle et manipule les objets et données du domaine de l'application sans se soucier de la manière dont l'information sera rendue à l'utilisateur.

L'adaptateur du noyau fonctionnel : traduit les données provenant du noyau fonctionnel en données compréhensibles par le contrôleur de dialogue. Il traduit également les données provenant du contrôleur de dialogue en données compréhensibles par le noyau fonctionnel.

Le contrôleur de dialogue : assure le séquencement des tâches. Il décrit, en fonction de l'état du système, l'ensemble des tâches autorisées ainsi que l'effet de l'exécution de celles-ci.

Le composant d'interaction logique (ou présentation) : traduit les informations fournies par le composant d'interaction physique en informations indépendantes de l'interface (indépendantes du type d'objets physiques utilisés) et les transmet au contrôleur de dialogue. De la même manière, il traduit les informations du contrôleur de dialogue en informations spécifiques aux objets physiques utilisés.

Le composant d'interaction physique (ou boîte à outils, toolkit) : permet de gérer l'interaction au plus bas niveau (le niveau lexical). Il transmet les entrées de l'utilisateur sur les objets de l'interaction (par exemple les widgets) au composant d'interaction logique et transforme les données provenant du composant d'interaction logique (par exemple des modifications de l'état d'un widget) en informations graphiques visualisables et perceptibles par l'utilisateur.

Le modèle architectural ARCH introduit également trois types d'objets qui permettent de décrire la nature des données qui transitent entre chaque composant :

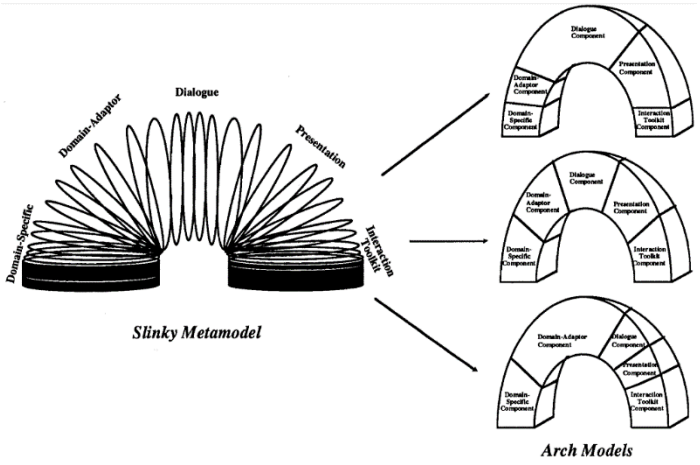


FIGURE 2.23 – Le méta-modèle Slinky associé au modèle architectural ARCH [Kazman and Bass, 1994]

- Les objets du domaine décrivent les données provenant directement ou indirectement du noyau fonctionnel.
- Les objets de présentation décrivent de manière abstraite les événements provoqués par l'utilisateur sur les composants physiques de l'interaction ainsi que les données qui sont présentées à l'utilisateur.
- Les objets d'interaction sont des instances propres à un composant d'interaction physique. Ils implémentent des techniques d'interaction et de visualisation qui leur sont spécifiques.

Le méta-modèle Slinky est présenté en Figure 2.23 . Il a été conçu au-dessus du modèle architectural ARCH afin de représenter le poids des différents composants ; il montre les variations en fonction des choix fixés lors de la spécification du système interactif et des efforts portés sur un ou plusieurs composants par rapport aux autres.

2.4.2.3 PAC et PAC-Amodeus

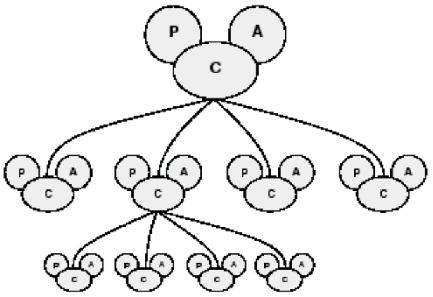


FIGURE 2.24 – Modèle architectural PAC (J. Coutaz 1987)

Le modèle architectural PAC [Coutaz, 1987] permet de modéliser les systèmes interactifs comme une hiérarchie d'agents PAC. Cette hiérarchie, présentée en Figure 2.24 est composée de trois facettes différentes pour chaque agent PAC :

P pour Présentation : cette facette décrit les entrées et sorties de l'agent perçues par l'utilisateur.

A pour Abstraction : cette facette décrit les données et les méthodes de l'agent.

C pour Contrôleur : cette facette assure la cohérence entre l'abstraction et la présentation et permet la communication entre les agents.

Une approche récursive permet de modéliser l'architecture complète d'une application interactive par une hiérarchie d'agents PAC. Cette hiérarchie est composée de plusieurs niveaux d'abstraction se rapprochant des couches définies par le modèle architectural Seeheim.

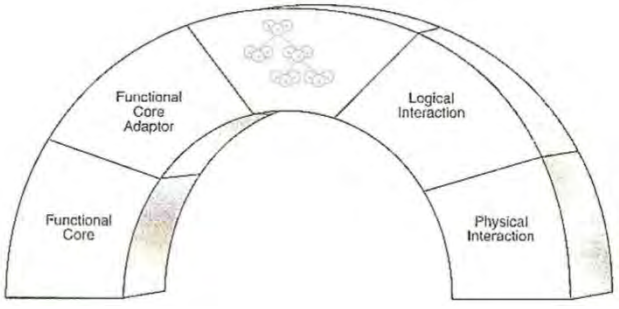


FIGURE 2.25 – Modèle architectural PAC-Amodeus (Nigay et Coutaz 1993)

Le modèle architectural PAC-Amodeus [Nigay and Coutaz, 1993], présenté en Figure 2.25 est un modèle hybride associant le point de vue linguistique du modèle architectural

ARCH au modèle architectural PAC et sa philosophie régit par des agents.

Ces différentes architectures sont très génériques mais ne suffisent pas pour la prise en compte des problématiques des systèmes interactifs multimodaux.

2.4.3 Les architectures dédiées aux systèmes interactifs multimodaux

La conception de systèmes multimodaux est complexe, et malgré des travaux datant de nombreuses années, avec par exemple [Bolt, 1980], cela reste une tâche difficile. Des travaux ont émergé pour faciliter l'abstraction des composants par exemple ICARE [Bouchet et al., 2004]. Ils décrivent un modèle conceptuel de la multimodalité qui organise dans un canevas unificateur les modalités et leurs formes de combinaisons. Basés sur ce modèle, ils définissent une approche générique à composants logiciels, notée ICARE, facilitant et accélérant la conception, le développement et le maintien des interfaces multimodales. L'outil ICARE permet de rendre opérationnel l'approche par composants. Un éditeur graphique est fourni, simplifiant la phase d'assemblage des composants et générant automatiquement le code correspondant à l'interaction multimodale. Les modèles d'architectures génériques dédiés aux systèmes interactifs multimodaux restent néanmoins rares.

2.4.3.1 MMI

Multimodal Architecture and Interfaces est un standard ouvert en développement depuis 2005 [MMI, 2005]. Il est encore aujourd'hui considéré comme un brouillon. Édité par le World Wide Web Contortium., son but est de standardiser les interfaces pour faciliter l'intégration et la gestion des interactions multimodales dans un environnement informatique. C'est une architecture orientée événement, par opposition aux architectures orientées service. Elle est basée sur le patron de conception MVC (Modèle Vue Controler [Goldberg and Robson, 1983]). Cette architecture reste très abstraites et est présentée sous forme de recommandations sur les interfaces plutôt qu'une solution d'implantation intégrée à une méthode. Les trois principaux modules de l'architecture sont :

Le contrôleur d'interaction est chargé de tous les échanges de messages entre les composants. Il s'agit d'un bus de communication, d'un gestionnaire d'événements qui permet la communication entre les différents composants de modalités du système et le système lui-même. Chaque application possède un contrôleur qui est au cœur de l'interaction. À la manière du gestionnaire de fenêtres, il gère la communication entre les modules, les comportements spécifiques, assure la cohérence entre les entrées et sorties et fournit un état général de l'application.

Les composants de modalités sont responsables de la gestion des diverses entrées et sorties par exemple la reconnaissance vocale, la reconnaissance de gestes, ou en sortie, la vidéo, la synthèse vocale. Un composant de modalités pourra être unimodal ou multimodal.

Le composant de données gère les données publiques de l'application qui peuvent être partagées par plusieurs composants de modalités ou par les autres modules. Ils peuvent donc être partagés par chaque module dans le cas d'une gestion privée du stockage de données.

MMI est plutôt orientée vers les technologies web, même si elle peut être adaptée à des systèmes interactifs plus classiques, son principal défaut par rapport à notre contexte est son caractère abstrait. Elle est portée sur les interfaces plutôt qu'un véritable outil de conception de systèmes critiques. Elle vise la standardisation plutôt que la fiabilisation des systèmes.

2.4.3.2 Smartkom

Smarkom [Wahlster, 2006], visible en figure 2.26, est une technologie issue d'un projet de recherche allemand qui vise à à faciliter l'interaction basée sur l'interaction entre l'homme et son environnement en coordonnant de multiples modalités. Le projet a abouti à trois scénarios se concentrant sur les modalités suivantes :

1. En mobilité
2. Pour l'interaction à l'intérieur de la maison
3. En public

Le projet a tout d'abord été pensé comme un assistant virtuel, par lequel transite l'intégralité de l'interaction. Le caractère multimodal de l'interaction a poussé les membres du projet à développer une architecture plus ou moins spécifique, permettant l'intégration à l'assistant virtuel de nouvelles modalités . C'est une des premières réalisations issues de la recherche en terme d'architecture de taille industrielle pour des systèmes multimodaux, que ce soit en entrée ou en sortie. La philosophie de flexibilité est inintéressante dans le cadre de nos travaux mais le manque de rationnel et de formalisme élimine ce modèle pour l'utilisation dans un milieu critique.

2.4.3.3 MUDRA

Dans des travaux un peu plus récents, des chercheurs belges de l'université de Namur et de Bruxelles ont développé un framework appelé MUDRA [Hoste et al., 2011]. Leur but principal est de favoriser le prototypage rapide d'applications multimodales. À l'aide d'une infrastructure développée par des spécialistes, un designer d'interaction pourra tester des interactions multimodales grâce à un framework intégré. MUDRA propose la fusion multimodale [Lalanne et al., 2009] à différents niveaux : au niveau des données, au niveau des fonctionnalités et au niveau sémantique, le plus haut niveau, comme indiqué dans [Hoste et al., 2011]. Ce framework gère les probabilités, pour permettre la décision en fonction d'un paramètre extérieur comme le facteur de confiance d'une reconnaissance vocale. Il permet aussi l'utilisation de temps quantifié, via l'utilisation de fenêtres courantes de validité des événements.

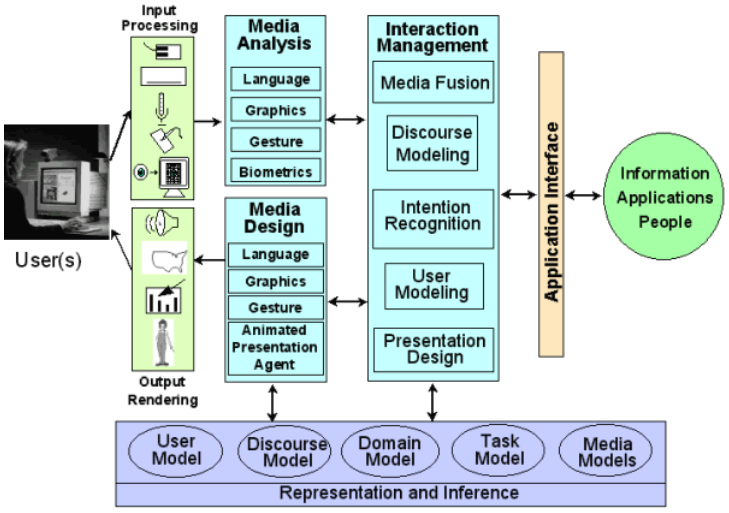


FIGURE 2.26 – The General Architecture of SmartKom [smartkom,]

MUDRA possède une propriété générique intéressante : l'exclusion de modalité, appelé *negation of event* dans [Hoste et al., 2011]. Cette propriété, absente des propriétés CARE, permet de spécifier qu'une modalité est désactivée lorsqu'une autre modalité est en cours d'utilisation. Il n'y a pas eu de définition formelle de cette propriété dans la littérature, et c'est une propriété technique dans MUDRA, plus qu'un formalisme.

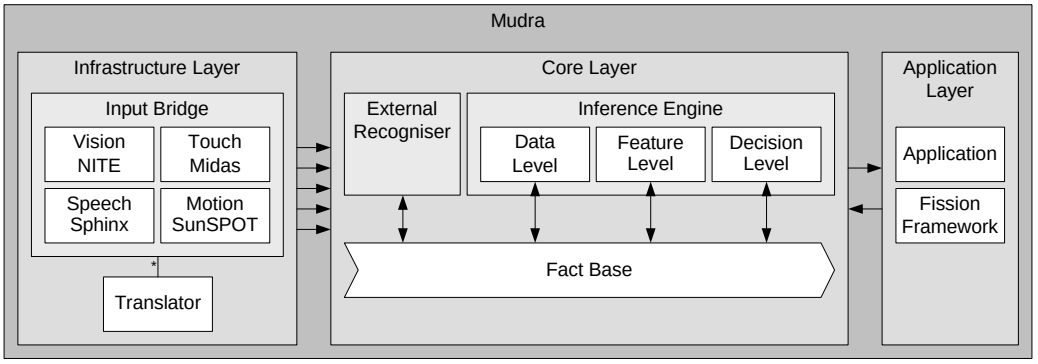


FIGURE 2.27 – L'architecture de Mudra

L'absence de formalisme permettant de décrire de manière complète et non ambigüe les relations entre modalités ainsi que son caractère figé sont les deux principaux problèmes de MUDRA. Un développeur voulant intégrer une nouvelle modalité non incluse dans le framework ne pourra pas le faire facilement.

MUDRA permet, via l'ajout de contraintes sur les systèmes d'entrée, de gérer une partie des problèmes liés aux utilisateurs multiples. Le réglage fin de la gestion des rôles restera en dehors des limites de ce framework. De plus, il est pensé pour du prototypage

rapide et non comme un outil intégré à un processus de développement visant à certification. MUDRA est basé sur un langage non formel à base de règles, qui pose problème sur la résolution des conflits et qui rend rédhibitoire l'utilisation dans un système critique.

2.4.3.4 Autres Framwork

Il existe par ailleurs d'autres environnements de spécification d'interfaces multimodales comme SKEMMI [Lawson et al., 2009] ou OPENINTERFACE [**?**] dans lesquels les composants sont confondus, quelles que soient leur fonctions et ne permette donc pas l'architecture de composant par famille de fonction.

2.4.4 ARINC 653, et son architecture dédiée aux systèmes critiques

Les concepteurs de systèmes critiques doivent assurer, dès le développent, que le système conçu sera sûr de fonctionnement. Tous les domaines critiques possèdent des contraintes différentes, ce qui a amené à la rédaction de standards propres à chaque domaine, particulièrement pour l'architecture et le fonctionnement des systèmes opératoires temps réel, par exemple AUTOSAR [Schmerler and Rimkus, 2013] pour l'automobile, ou encore ARINC 653 [Prisaznuk, 2014] pour l'avionique. Néanmoins certains principes de base de sûreté de fonctionnement se retrouvent dans l'immense majorité des standards. Bien qu'adaptés à leurs contraintes particulières respectives, ces standards reposent sur des principes de base communs.

2.4.4.1 Principes de base

Les deux principes de base pour le bon fonctionnement des systèmes critiques sont le confinement des processus dans la mémoire ainsi que la séparation de leur exécution sur processeur. Illustrons ces principes de base avec la norme ARINC 653, qui définit la ségrégation spatiale et temporelle des systèmes opératoires temps réel dans les milieux avioniques.

2.4.4.2 ARINC 653, standard de partitionnement temporel et spatial de ressources informatiques

La ségrégation spatiale

Pour pouvoir fonctionner, un logiciel a besoin d'espace mémoire dans la RAM, mais aussi dans le stockage persistant. Ainsi, si l'on veut faire fonctionner plusieurs logiciels en s'assurant qu'aucun ne risque de perturber le fonctionnement des autres, la première solution est de multiplier les machines afin que chaque processus soit seul sur une machine.

La multiplication des machines n'étant viable sur le plan technique, économique, et encore moins vis à vis du poids embarqué, il faut donc trouver une solution pour partager les ressources matérielles. Le risque du partage de ressources d'une machine est que l'erreur d'une fonction influe sur les autres fonctions présentes. Ainsi une apparemment minuscule

erreur sur une entrée d'altitude pourrait influer sur l'affichage du *PFD (primary flight display)*. C'est totalement impensable dans un aéronef, car cet écran sert à afficher le minimum vital pour faire voler un avion, et donc possède le niveau de criticité le plus élevé. La seconde solution est de partager les ressources matérielles en s'assurant que les processus ne puissent pas influer les uns sur les autres, même en cas de défaillance de l'un d'entre eux.

La ségrégation spatiale des processus consiste donc à affecter à chacun des processus un espace mémoire distinct, qui ne puisse pas "baver" sur celui des autres processus au sein de la machine. Elle est généralement réalisée de manière à ce que les processus ne sachent pas qu'ils ne sont pas seuls. La table mémoire commence donc à 0 pour tous les processus et la ségrégation impose un confinement de chacun des processus, par un espace mémoire défini et non extensible.

La ségrégation temporelle

Pour s'assurer que les logiciels n'entrent pas en concurrence lors de l'utilisation de la ressource matérielle de calcul, le processeur, les différents processus sont ségrégés temporellement, c'est-à-dire qu'ils fonctionnent l'un après l'autre sur le processeur. La figure 2.28 présente le fonctionnement d'une ségrégation de 3 processus sur un processeur.

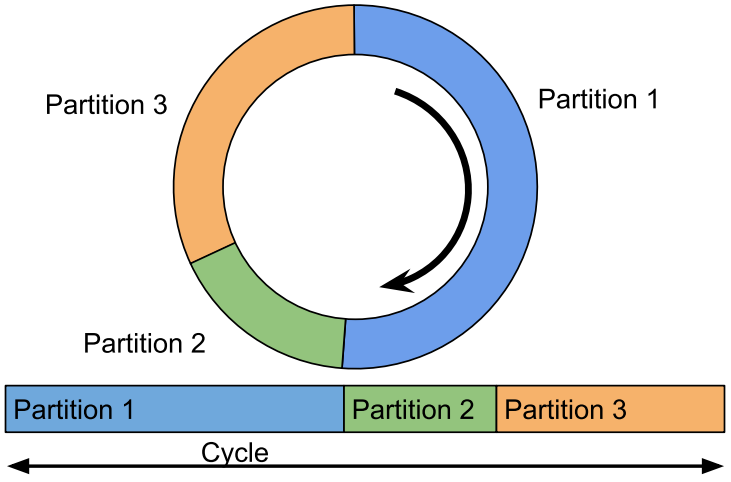


FIGURE 2.28 – La ségrégation temporelle de processus

De cette manière, les processus peuvent jouir de l'intégralité de la puissance matérielle lorsqu'ils sont en exécution. Un seul processus actif à la fois, les autres étant en veille. deux approches sont généralement utilisées pour ce genre de ségrégation :

- Les processus gèrent leur temps et se mettent en pause à la fin de leur exécution. la gestion du temps est flexible et engendre le risque qu'un processus monopolise la ressource
- Un scheduler gère le temps de manière externe et met les processus en veille ou en action. La gestion peut alors être **souple** si on met en pause les processus, ou **dure**,

si les processus sont tués lorsqu'ils dépassent le temps imparti.

Une fois les processus ségrégés de manière temporelle et spatiale, on réduit les risques d'influence d'une faute sur les autres processus. Cependant, comme ils n'ont plus d'espace mémoire ni de temps d'exécution commun, ils ne peuvent plus communiquer comme on le ferait dans un programme intégré.

2.4.4.3 La communication inter-processus

Pour pouvoir résoudre ce problème, ARINC 653 définit deux types de communication entre les partitions :

La communication en Queuing : est une communication de type FIFO *first in first out* où tous les messages sont transmis et reçus dans l'ordre. La lecture d'un message est destructrice dans ce type de communication : une fois lu, le message est détruit.

La communication en Sampling : est une communication de type tableau noir, où seul le dernier message écrit peut être lu. La lecture d'un message n'est pas destructrice : lorsqu'un processus lit le message, il y a toujours une valeur disponible, qui est la plus récente. Pour comprendre ce type de communication moins commune, on peut prendre un exemple de communication entre deux personnes au moyen d'un tableau noir dans une salle de classe.

- Une personne A vient sporadiquement et met à jour une donnée à la craie sur le tableau noir. Seul A possède le droit d'effacer le tableau et elle l'effacera à chaque mise à jour.
- Une personne B peut venir à n'importe quel moment, il y aura toujours une valeur écrite sur le tableau, B n'ayant pas le droit d'effacer le tableau. Si B vient plus souvent que A, elle verra parfais deux fois la même valeur. Si B vient moins souvent que A, elle ne verra que la valeur la plus récente.

L'écriture est destructrice, ce qui permet d'assurer que seule la valeur la plus récente d'une variable est conservée.

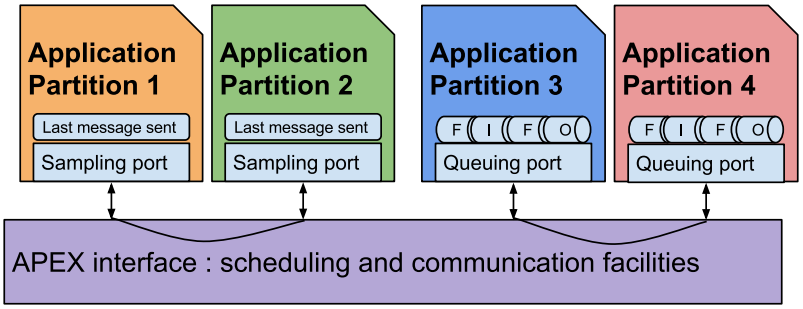


FIGURE 2.29 – La Communication inter Partition

Avec ces deux types de ségrégations, couplées à la communication, on dispose d'un ensemble de contraintes minimum, commune à la plupart des systèmes critiques.

2.4.4.4 Les Systèmes d'exploitation et les simulateurs d'ARINC 653

Pour pouvoir mettre en œuvre des mécanismes de sûreté de fonctionnement nécessaires dans les systèmes critiques, il faut disposer d'un environnement de test possédant des attributs de ségrégation spatiale et temporelle. De par le contexte applicatif de ce mémoire, nous nous intéressons aux systèmes opératoires qui suivent le standard ARINC 653 qui définit, en autre, des propriétés de ségrégation spatiale et temporelle des logiciels.

Plusieurs systèmes d'exploitation respectant ce standard sont disponibles. Nous pouvons ainsi citer, pour les systèmes d'exploitation commerciaux, VxWorks de Wind River, LynxOS de Lynx Software Technologies, INTEGRITY-178 RTOS de Green Hills Software et PikeOS de Sysgo ; l'unique solution open source que nous avons pu trouver est le système d'exploitation POK , qui a permis de supporter plusieurs travaux de recherche. Il permet également de développer le code des partitions à partir d'une spécification AADL. Les systèmes commerciaux sont extrêmement onéreux et ne sont pas accessibles aux simples travaux de recherche. Il serait également très coûteux en temps de développement de s'adapter à un système d'exploitation, même lorsque l'on considère POK qui pourrait être accessible. Il faudrait en effet coder les différents composants du système interactif tolérant aux fautes en langage C et l'adapter aux contraintes de ces systèmes d'exploitation. Ces différents systèmes ne supportent également pas le langage Java, ce qui nous empêcherait d'utiliser la modélisation ICO.

Le but des travaux étant de proposer une méthode, il n'est pas judicieux dans notre cas d'utiliser un vrai système d'exploitation respectant le standard ARINC 653. Nous nous sommes donc intéressés aux simulateurs existants, car ceux-ci permettent de moins fortes contraintes de développement et sont donc beaucoup plus adaptés à des objectifs de test et de preuve de concept. Ce genre de système a beaucoup été étudié par la recherche, mais très peu de simulateurs de ce genre sont accessibles. Nous pouvons ainsi citer le simulateur AMOBA [Pascoal et al., 2008], intégré au simulateur SIMA qui permet de simuler des plateformes avioniques modulaires intégrées [Schoofs et al., 2009]. Ce simulateur est accessible de manière commerciale. Il existe également CCM [Dubey et al., 2010], un modèle composant du standard ARINC 653, auquel est associée une suite d'outils appelée ACMTOOLSUITE [Dubey et al., 2011] ; mais encore une fois, celui-ci ne permet pas l'utilisation du langage Java pour le développement du code des partitions.

2.4.5 Les propriétés des architectures

Toutes les architectures présentées jusqu'à présent permettent d'identifier certaines propriétés. Len bass, dans "*Sofware architecture in practice*" [Bass, 2007], a listé les propriétés que doit assurer une architecture. Certaines de ces propriétés recoupent déjà celles des systèmes interactifs et systèmes critiques que nous avons présentées au chapitre précédent. Les propriétés listées par Len Bass sont toutes pertinentes dans le cadre de ce mémoire :

La disponibilité décrit la capacité du système à être prêt à être utilisé, par exemple

en offrant des possibilités de redondance, ou de gestion de la charge. C'est une des propriétés déjà présentes pour les systèmes critiques.

L'interopérabilité décrit la capacité de l'architecture à aider la communication avec d'autres systèmes, par exemple en offrant des interfaces standardisées, compatibles avec le domaine d'opération.

La modifiabilité ; une architecture doit assurer la modifiabilité du système à moindre coût, en assurant que les composants ont des fonctions définies, et notamment que la modification de l'un impacte au minimum le reste du système. Cette propriété est extrêmement importante au vu des durées de vie des systèmes visés dans ce mémoire. Cette propriété est courante dans les systèmes interactifs.

La performance est une propriété attendue par les exploitants du système. L'architecture doit permettre de garantir la performance attendue quelles que soient les conditions d'exploitation du systèmes (charge). Par exemple dans le cas d'une architecture dont les composants sont en série, la performance du système complet est limitée à celle du composant le moins performant.

La security englobe la sûreté de fonctionnement, la fiabilité, la sécurité, etc. Il est intéressant de noter que la définition de Len Bass se trouve en conflit avec la classification présentée au chapitre précédent (où elle est découpée en deux propriétés, *dependability* et *security*). Cette propriété, ou plutôt ces propriétés non fonctionnelles, sont centrales dans les systèmes critiques. L'architecture doit en effet permettre d'assurer ces propriétés par des composants ou des mécanismes adaptés.

L'utilisabilité dans le cadre d'une architecture, est la propriété permettant d'augmenter l'utilisabilité du système conçu. Par exemple, si on veut implanter une fonction *Undo*, il faut que l'architecture soit résilente à son ajout. Cette propriété est d'après [Bass and John, 2003], assurée grâce à l'utilisation de *paterns d'architecture* lors de l'implantation des composants.

La capacité à être testée ; une architecture doit permettre d'être testée tout ou partie. Il est particulièrement important que les composants d'une architecture puissent être testés unitairement à cause de la complexité des systèmes complets. Il est en général impossible de développer un système en un seul tenant sans faire de tests unitaires.

Comme pour les propriétés des systèmes interactifs vues dans le chapitre 1, certaines d'entre elles peuvent entrer en conflit. Il faut alors pouvoir ordonner ces propriétés et résoudre d'éventuels conflits. Len Bass explique qu'une "bonne" architecture n'existe pas, ou tout du moins qu'elle ne sera pas forcément applicable à un autre projet. Une bonne architecture répond à un besoin spécifique et l'importance que l'on accorde à chaque propriété sera très différente en fonction du domaine. Un des prérequis à la réalisation d'une architecture est donc l'identification des propriétés mises en jeu dans le domaine d'application.

Les travaux que nous avons listés sur les architectures, dans le domaine des systèmes interactifs comme dans celui des systèmes critiques, ne combinent pas l'ensemble des aspects nécessaires à la conception d'un système interactif dans un environnement critique.

2.5 Synthèse

Les concepteurs de systèmes interactifs destinés aux environnements critiques ont un réel besoin de méthodes et d'outils qui leur permettront d'intégrer des interactions avancées dans les futurs cockpits. Ils disposent :

- Des processus de développement dans lesquels nous nous inscrirons en ajoutant une méthode d'évaluation des techniques d'interaction pour supprimer les fautes humaines déclenchées par des surprises lors de l'utilisation du système
- Des langages formels dont nous retiendrons ICO qui permet la spécification complète et non ambigüe des systèmes interactifs critiques
- Des environnement de tests que nous viendrons compléter par une plateforme permettant de tester de nouvelles interactions spécifiées formellement, avec les contraintes des environnements critiques
- Des éléments structurants tels que les architectures logicielles, mais aucune n'est adaptée à la conception de systèmes interactifs critiques et nous en proposerons un modèle compatible avec ces exigences

Nous allons maintenant présenter les contributions.

Deuxième partie

Contribution

Chapitre

3 MIODMIT, un modèle d'architecture générique pour les systèmes interactifs critiques multimodaux

Sommaire

3.1 Introduction

L'intérêt d'implantation de la multimodalité dans les systèmes interactifs critiques n'est plus à démontrer [Oviatt, 1999]. L'ingénierie de ces systèmes impose néanmoins des besoins non résolus, comme nous avons pu le voir dans le chapitre 2.

La conception d'un système interactif multimodal critique repose avant tout sur une architecture intégrant de manière appropriée le hardware et le software. Faire reposer un cycle de développement sur une architecture permet d'apporter certaines propriétés au système interactif que nous avons détaillées à la section 2.4.5.

La principale propriété fonctionnelle recherchée pour une architecture destinée à la conception de systèmes interactifs critiques est d'assurer la modifiabilité du système. On veut pouvoir le faire évoluer aisément, que ce soit pendant les phases de conception, ou plus tard, au cours de l'opération (par exemple si des périphériques doivent être remplacés).

Concernant les propriétés non fonctionnelles, l'architecture doit permettre au système conçu d'être performant, interopérable avec des systèmes existants, d'utiliser des mécanismes de sûreté de fonctionnement pour augmenter la fiabilité, et d'implanter des mécanismes d'utilisabilité.

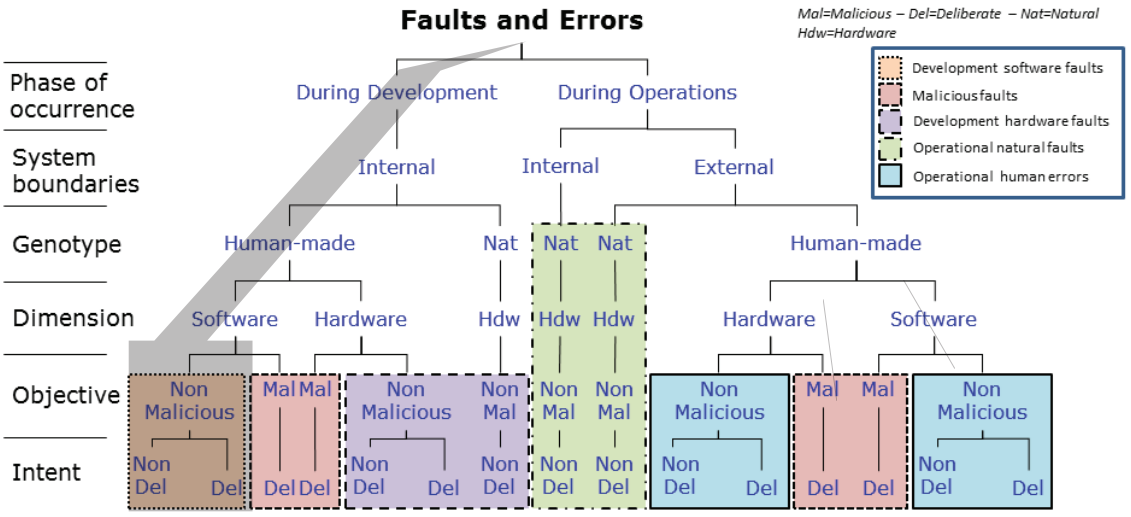


FIGURE 3.1 – Erreurs lors du développement, classification des fautes des systèmes informatiques

La conception d'un modèle d'architecture générique pour les systèmes critiques interactifs permet de traiter une partie des fautes introduites lors du développement, grisées sur la figure 3.1 que nous avions déjà présentée en section 1.5. Grâce au modèle d'architecture, on pourra notamment modifier aisément les prototypes afin d'y inclure de nouvelles interactions sans toucher à l'intégralité du système, limitant ainsi les fautes lors de la modification d'un composant.

3.1.1 Organisation du chapitre

Dans ce chapitre, nous verrons l'organisation d'un flux d'informations et de données d'un système interactif critique au sein d'une architecture générique, composée d'un ensemble de sous-systèmes et de composants détaillés dont les fonctionnalités sont identifiées. Cette architecture décrit pour chacun des composants : les fonctions qu'il implante, les données qu'il utilise, et celles qu'il produit. En phase de conception, cette architecture guide le concepteur pour l'identification des composants dans le but, par exemple, d'intégrer un composant supplémentaire dans une application. Dans le cadre d'une évolution, elle permettra de limiter l'impact en termes de développement, grâce à l'identification et la localisation de chaque fonctionnalité. De plus son utilisation permettra l'édition, la fourniture, d'un *design rational* lors de la phase de certification.

Le rationnel des fonctionnalités des composants étant majoritairement relatif à des problématiques d'implantation, il sera traité dans la troisième partie de ce manuscrit, sous forme d'exemples.

Le chapitre commence par une présentation d'un exemple simple qui illustrera les concepts basiques tout au long du chapitre, que nous approfondirons dans les chapitres d'exemples illustratifs 6 & 7. Puis nous aborderons une vue générale du modèle d'architecture MIODMIT (pour Multiple Input Output devices Multiple Interaction Techniques) dans la section 3.2 avant d'entrer dans les détails du rôle de chaque composant et sous-système dans la section 3.3. Nous détaillerons ensuite l'utilisation de l'architecture et les livrables préalables nécessaires. Enfin nous verrons les propriétés assurées par l'architecture.

3.1.2 Présentation de l'exemple filé : les quatre saisons

L'application des Quatre Saisons, présentée sur la figure 3.2 permet de contrôler la saison active sur le label au centre de la fenêtre. Un seul bouton est actif à la fois.

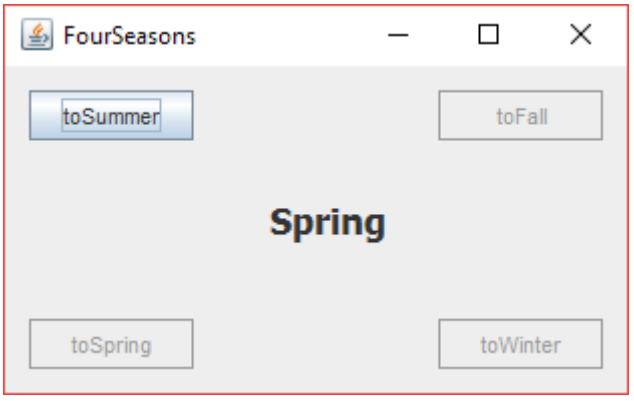


FIGURE 3.2 – Les Quatre Saisons

Le fonctionnement est donc le suivant :

- En été, seul le bouton «Vers Automne» est actif
- En automne, seul le bouton «Vers Hivers» est actif
- En hivers, seul le bouton «Vers Printemps» est actif
- Au printemps, seul le bouton «Vers Été» est actif

La figure 3.3 présente le modèle du noyau fonctionnel de l'application, au sens de l'architecture ARCH. Cette application ne mettra en œuvre qu'une petite partie de l'ar-

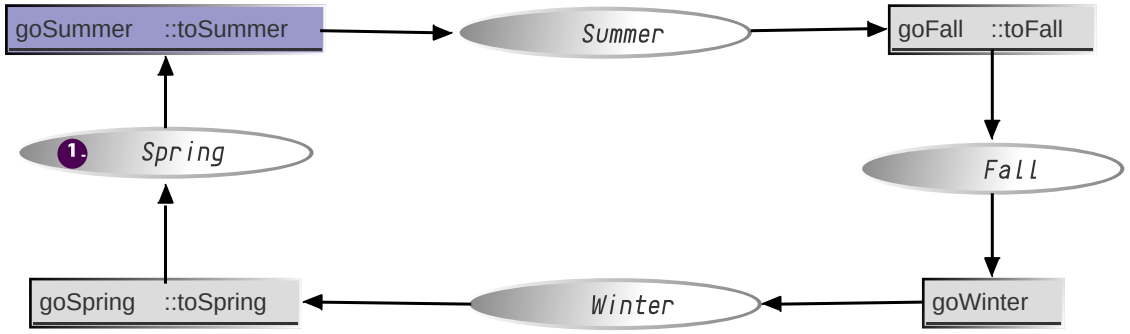


FIGURE 3.3 – Les Quatre Saisons, modèle en réseau de Petri de l'application

chitecture, mais elle permet d'en comprendre les grandes lignes avant de se plonger dans les détails sur des systèmes complexes.

3.2 Présentation générale de l'architecture

La figure 3.4 présente une vue générale du modèle d'architecture MIODMIT [1]. La représentation s'inspire librement du formalisme AADL [Feiler et al., 2006]. AADL est destiné à produire des architectures instanciables et testables. L'architecture MIODMIT devant être raffinée, il nous a semblé approprié d'utiliser une représentation se rapprochant d'une de celles utilisées dans les systèmes critiques.

Tout au long du chapitre, L'architecture sera exemplifiée sur l'application des 4 saisons dans un encadré gris.

3.2.1 Ensemble des systèmes d'entrée

Les rectangles gris représentent l'ensemble des systèmes très bas niveau permettant l'inclusion d'un périphérique dans un système. On rajoutera un de ces ensembles pour chaque nouveau type de périphérique. Ils constituent le minimum nécessaire pour pouvoir

1. Multiple Input Output devices Multiple Interaction Techniques

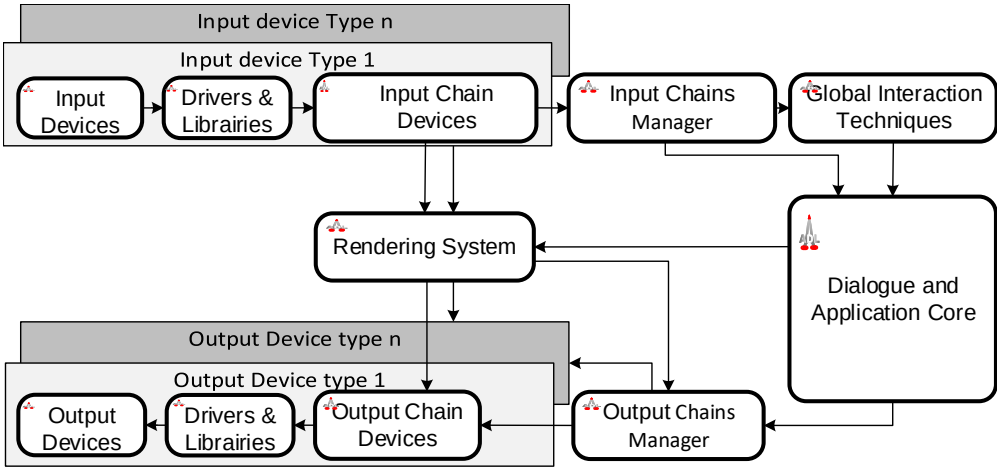


FIGURE 3.4 – Présentation générale des systèmes dans le modèle d'architecture MIODMIT

faire des interactions simples avec un périphérique donné. La suite de cette section présente les différentes fonctionnalités des systèmes décrits sur la figure 3.4.

Dans notre application, l'ensemble des systèmes d'entrée contient les composants permettant de faire fonctionner la souris et son curseur associé. Ces fonctionnalités sont intégrées au système d'exploitation dans le cas d'application ne disposant que de périphériques d'entrée standards.

Regardons maintenant en détail les composants de cette chaîne.

3.2.1.1 Périphériques d'entrée

Pour pouvoir utiliser un nouveau périphérique dans un système interactif, il faut bien évidemment commencer par avoir un périphérique physique. Ce premier système est appelé *Input Devices* sur la figure 3.4.

Il peut être actif, c'est-à-dire qu'il peut envoyer de manière autonome des événements, ou passif, et dans ce cas il faudra lire les valeurs dans un registre depuis un autre système.

Le périphérique est relié au système interactif soit de manière physique, via un port ou une connectique, soit de manière virtuelle, via une connexion logicielle, par exemple une connexion sans fil de type Bluetooth. S'il est placé sur un réseau quelconque, par exemple un scanner réseau, il sera aussi relié de manière logicielle. Quelle que soit sa connectivité, un pilote, plus communément appelé *driver*, devra faire le lien entre les événements et données très bas niveau du périphérique et le système d'exploitation.

Les périphériques de notre application sont la souris, et de manière générale, tout autre périphérique capable de contrôler le curseur du système d'exploitation, comme un trackpad sur un ordinateur portable.

3.2.1.2 Pilotes et Librairies

Le deuxième système (*Drivers and libraires* sur la figure 3.4) sert de porte d'entrée aux événements pour le système interactif, son rôle sera :

- D'assurer la détection du périphérique
- De récupérer de manière active ou passive les événements du périphérique
- S'il y a lieu, de permettre le réglage des paramètres du périphérique, par exemple la fréquence d'échantillonnage permettant la synchronisation entre le périphérique et le système d'exploitation.
- De fournir un ensemble de fonctions permettant d'utiliser les données et les événements au niveau du système d'exploitation et donc dans un langage de programmation de plus ou moins haut niveau.

Ce système est généralement très dépendant du système d'exploitation du fait de la gestion au plus proche du matériel, il est même en général partiellement ou totalement intégré au système pour les périphériques standard.

Pour les quatre saisons, les périphériques(souris et trackpad) sont courants. Leurs pilotes et librairies sont intégrés à tous les systèmes d'exploitation. Cependant, certains constructeurs proposant des souris plus élaborées fournissent aussi des pilotes réalisés par leurs soins, pour offrir plus de contrôle sur le comportement des périphériques ; par exemple la résolution et la fréquence de rafraichissement.

3.2.1.3 Chaîne de périphériques d'entrée

Le système nommé *Input chain Devices* sur la figure 3.4 permet de transformer les événements et données brutes venant des pilotes en objet utilisables pour faire de l'interaction. Par exemple, pour une souris, ce système transformera, entre autres, les informations de quantités de déplacement (dx, dy) en un couple de coordonnées (x,y) destiné à définir la position du curseur. Encore une fois ici, pour des périphériques standards, ce système fera le plus souvent partie intégrante du système d'exploitation. Les applications fonctionnant en plein écran, par exemple les jeux vidéo, proposent parfois des composants propres, qui viennent soit en complément soit en remplacement de celui de l'OS, pour contrôler plus finement l'interaction.

Dans notre application, cet ensemble de composants correspond au curseur, sa fonction de transfert associé [Casiez and Roussel, 2011] ainsi qu'à la gestion des reconnections des périphériques sur le curseur.

3.2.2 Gestionnaire de Chaînes de périphériques d'entrée

Si on veut permettre la reconfiguration dynamique d'interaction, par exemple, pour pouvoir tolérer la perte ou l'ajout d'un type de périphérique, il faudra inclure un gestionnaire de chaîne de périphériques d'entrée, nommé *Input Chains Manager* sur la figure 3.4. Si la configuration est unique ou gérée uniquement à l'initialisation, ce système peut être facultatif. Il sert principalement à basculer à chaud entre plusieurs configurations du système impliquant un ensemble de modalités différentes. Un exemple de re-configuration d'interaction est disponible sur l'environnement WINDOWS sur les tablettes surface ;

lorsque l'on débranche le clavier physique, l'OS propose de reconfigurer l'interface afin de l'adapter à l'unique modalité restante.

Il est impossible à l'heure actuelle de s'assurer de la viabilité d'un système critique dans le cas où l'ensemble des configurations n'est pas connu à priori. Ceci est dû au fait que le nombre d'états dans lesquels pourrait se trouver le système est infini. Le modèle d'architecture MIODMIT [2] n'est donc pas conçu pour être adaptable à chaud à des périphériques non prévus lors de la conception. Le couplage faible entre les composants assure néanmoins une grande modifiabilité en cas d'évolution du système ainsi que la possibilité de permuter entre des configurations pré-établies.

Dans l'application des quatre saisons, seul le curseur permet d'interagir sur l'application. On est en présence d'un seul type de périphérique.

3.2.3 Techniques d'interaction globales

Les techniques d'interaction peuvent être classées en deux grands types :

les techniques d'interaction locales liées à une zone sensible spécifique ou à un widget

les techniques d'interaction globales n'ont pas de cible particulière.

Les techniques d'interaction, quel que soit leur type et afin d'assurer une abstraction suffisante, ont pour but de déclencher des événements haut niveau qui pourront, à leur tour, déclencher des commandes. La particularité des techniques d'interactions globales est de ne pas avoir de zone sensible particulière associée, c'est-à-dire qu'elles pourront se déclencher sans avoir un widget particulier activé.

Par exemple, sur les systèmes d'exploitation Android, la commande vocale «OK Google» est disponible, quel que soit l'état du système tandis que «dis-moi quel temps il fera demain» ne rendra un résultat que lorsque le widget de reconnaissance vocale de Google est actif. La première n'est donc pas liée à une *zone sensible* particulière et sera catégorisée dans les techniques d'interactions globales, tandis que la seconde, liée à un état d'activation d'un widget et donc, par extension, à une zone sensible, sera catégorisée dans les techniques d'interactions locales. On peut citer un exemple sur les systèmes plus standards : l'événement *double-clic*. Il est généralement produit par le Windows manager, sans avoir besoin d'activer un widget. L'événement *double-clic* est donc produit par une technique d'interaction globale.

Dans notre application, on va utiliser les événements du système ainsi que les librairies Java qui permettent d'avoir directement les techniques d'interaction sur les boutons (clic, bouton pressé, bouton relâché, etc.)

3.2.4 Dialogue et noyau fonctionnel de l'application

Le système *Dialogue and Application Core* représente le cœur de l'application que l'on peut rapprocher du dialogue et du cœur d'application de ARCH. Ses composants seront

2. Multiple Input Output devices Multiple Interaction Techniques

le plus souvent codés dans un langage de programmation haut niveau et liés, au travers des APIs et d'événements, aux systèmes de gestion des entrées et des sorties.

La figure 3.3 représente le modèle de ce composant, correspondant à l'intégralité du comportement du noyaux fonctionnel de cette application simple.

3.2.5 Système de rendu

Le modèle d'architecture MIODMIT prend pour hypothèse que les entrées sont basées sur des événements et les sorties sur des états comme indiqués par Campos et Harrison dans [Campos and Harrison, 1997]. Le système de rendu va observer l'état des différents systèmes et réaliser un rendu adapté à la configuration de sortie, par exemple un rendu graphique couplé à de la synthèse vocale. Le système de rendu est aussi bien responsable du rendu de l'application, que du feedback immédiat dont les périphériques d'entrée et les techniques d'interaction globales pourraient nécessiter.

Dans les quatre saisons, on utilisera les librairies swing de java, qui centralisent les fonctions du système de rendu.

3.2.6 Gestionnaire de chaînes de périphériques de sortie

De la même manière que le gestionnaire de chaînes de périphériques d'entrée, son homologue en sortie sert à gérer le basculement dynamique entre les configurations de sortie afin de tolérer la perte d'un périphérique et le changement d'une configuration à chaud. Par exemple l'application WAZE permet de supprimer le guidage sonore lorsque l'écran du téléphone est allumé, mais les réactive automatiquement si l'écran est éteint.

L'écran est utilisé seul dans notre application, et nous n'aurons pas ce composant.

3.2.7 Ensemble de systèmes de sortie

L'ensemble des trois systèmes restant *Output Chain Devices*, *Driver and Librairies* et *Output Devices* fait miroir à leurs homologues en entrée et servent respectivement à gérer le branchement à chaud d'un type de périphériques de sortie, à fournir les moyens nécessaires au système pour utiliser le périphérique, et enfin à transformer physiquement l'information pour l'utilisateur.

Ces fonctions seront gérées par le système d'exploitation directement, dans le cas des quatre saisons.

3.3 Présentation détaillée du modèle d'architecture

Tous les systèmes vus sur la figure 3.4 sont composites, c'est-à-dire qu'ils sont composés de plusieurs sous-systèmes ou processus, qui peuvent à leur tour être composites, lors de l'implantation. Examinons la figure 3.5, qui présente une vue détaillée des composants.

3.3.1 Ensemble des systèmes d'entrées

L'ensemble des systèmes d'entrée est représenté par un rectangle gris. Ces systèmes sont généralement totalement intégrés aux systèmes d'exploitation, soit de manière native, soit via l'ajout de composants logiciels fournis par les constructeurs. Ils comprennent :

- Les périphériques d'entrée
- Les pilotes et librairies associées
- La chaîne de périphériques d'entrée

3.3.1.1 Périphériques d'entrée

Par définition, un système interactif multimodal est caractérisé par plusieurs modalités. Généralement cela se traduit par l'adjonction au système informatique de plusieurs périphériques d'entrée, une exception notable réside dans les systèmes uniquement multitouch qui sont multimodaux même s'ils ne possèdent qu'un seul périphérique d'entrée, l'écran tactile. Un périphérique informatique n'est ni plus ni moins qu'un capteur qui fournit des événements et/ou des données à un système informatique.

La configuration de ce système peut être statique si tous les périphériques sont définis avant le démarrage du système, ou dynamique si on veut que le système puisse accepter les branchements dits «à chaud».

La récupération des événements se fera au travers des pilotes.

3.3.1.2 Pilotes et Librairies

Les pilotes, *Driver* sur la figure 3.5, sont la porte d'entrée vers le système informatique. Ils doivent permettre :

- Le branchement à chaud du périphérique, c'est à dire sa connexion après l'allumage du système
- Le réglage des paramètres du périphérique, y compris la fréquence d'échantillonnage
- L'accès aux événements et données du périphérique

Généralement les pilotes sont fournis par le constructeur du périphérique, car ils nécessitent une grande connaissance du matériel afin d'assurer l'intégration complète des fonctionnalités et l'utilisabilité au sein d'un système d'exploitation. Certains pilotes sont directement intégrés aux systèmes d'exploitation afin d'assurer un fonctionnement de base des périphériques standards, tels que les souris et claviers, et ce, quelle que soit la configuration matérielle. Ils doivent être minimaux, car ils impactent directement le système

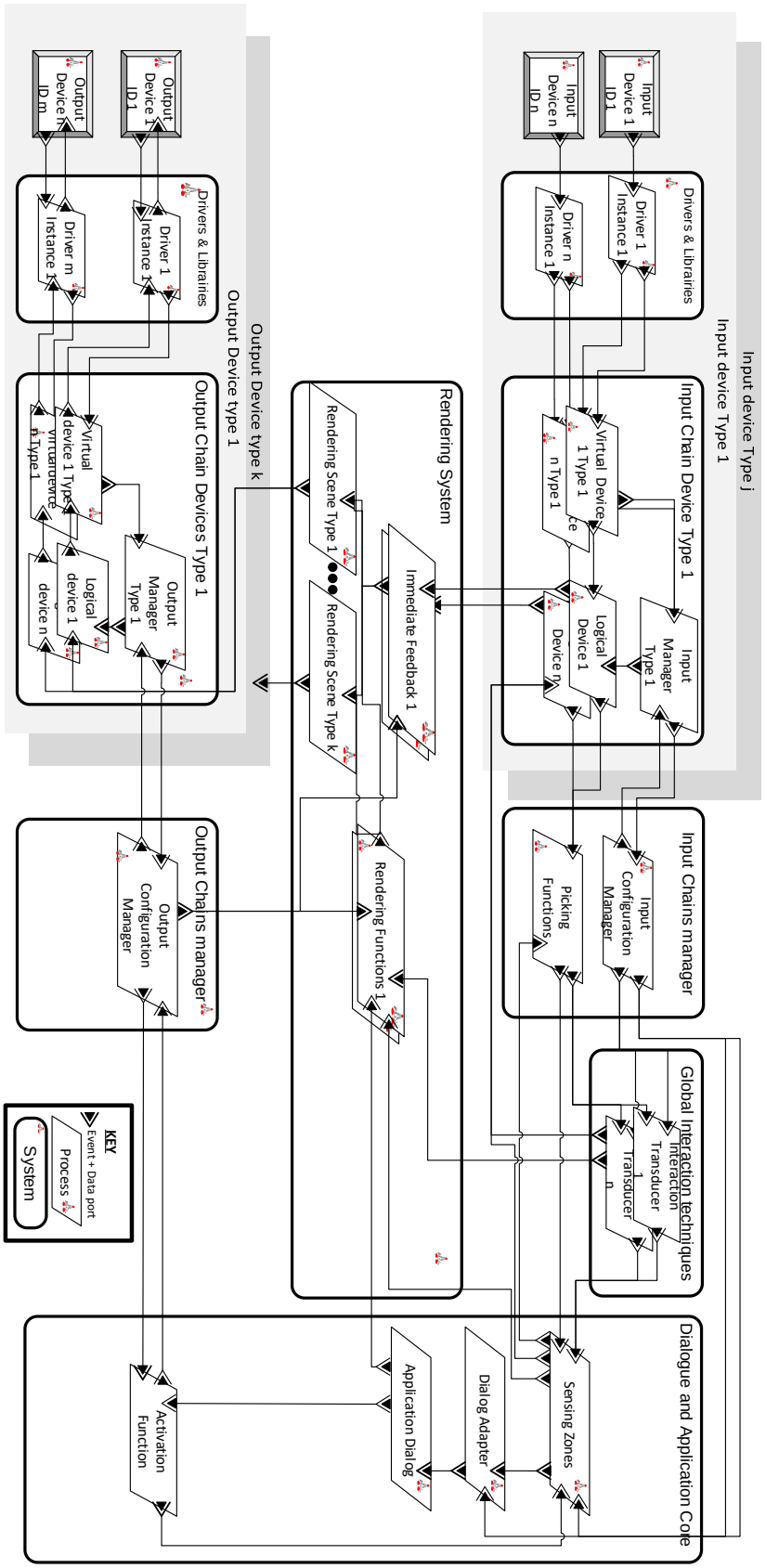


FIGURE 3.5 – le modèle d'architecture MIODMIT

de base compte tenu de la nécessité d'accès aux fonctions matérielles de l'OS, et peuvent introduire des fautes.

La mise à disposition des fonctions de haut niveau se fera par l'intermédiaire des librairies, non représentées sur la figure, car elles ne constituent pas un processus autonome. Là encore, elles sont normalement éditées par les fabricants de périphériques à cause de la nécessité de connaitre dans les détails le matériel ainsi que le fonctionnement du driver. La librairie dépend du langage de haut niveau visé, plusieurs librairies pourront être éditées pour un même périphérique afin de permettre son utilisation dans différents langages informatiques. Elle est exécutée dans l'espace utilisateur et risque donc d'impacter beaucoup moins le reste du système.

La configuration interne du système *Pilotes et Librairie* peut varier assez lourdement d'un système à un autre, et ce, pour plusieurs périphériques donnés. Si plusieurs claviers sont branchés sur un système,par exemple un clavier externe sur un ordinateur portable, les deux claviers seront gérés, dans les OS grands public, par la même instance de driver. Il peut arriver qu'on ait à lancer deux instances de driver pour pouvoir gérer deux périphériques, par exemple avec la Kinnect, il faudra lancer deux fois le driver avec des paramètres différents sous Linux pour permettre la gestion simultanée des deux Kinnects.

Un périphérique peut produire différents types d'information que le pilote devra traiter et rendre utilisables pour les composants de plus haut niveau. Sur les systèmes grand-public, le principal composant qui se sert de l'information provenant des pilotes est le système d'exploitation lui-même. Puisque le rôle du pilote est de faire l'interface entre le matériel et le logiciel, ces deux aspects se retrouvent dans les spécifications. Le pilote devra ainsi prendre en compte du côté matériel :

- Le type de lien physique avec le matériel informatique : port et bus physique ou logiciel
- La disponibilité du périphérique sur ce lien

et du côté logiciel :

- L'accès direct aux ressources matérielles
- La mise à disposition des primitives uniquement

Les fonctionnalités de plus haut niveau seront alors exécutées dans l'espace utilisateur. Dans [Accot et al., 1997], Chatty and al. définissent quelques propriétés qu'un pilote doit respecter. Notamment, il ne doit ni produire plus d'événements que le périphérique ne produit : pilote bavard (*chatty*), ni moins d'événements que le périphérique ne produit : pilote timide (*shy*). Le pilote doit donc, au delà de permettre le réglage du périphérique, produire une correspondance directe des événements jusqu'aux systèmes d'exploitation.

3.3.1.3 Chaîne de périphériques d'entrée

La chaîne de périphériques d'entrée gère le comportement très bas niveau lié à un type de périphérique, et contient trois types de processus différents que nous détaillerons dans les sous-sections suivantes :

- Les périphériques virtuels

- Les périphériques logiques
- Le gestionnaire de périphériques

Les périphériques virtuels, *virtual devices* sur la figure 3.5, ont pour but d'obtenir une représentation informatique de l'interacteur physique fidèle au monde réel. Une souris virtuelle possédera plusieurs attributs booléens correspondant à l'état des boutons physiques, mais aussi des quantités de mouvement instantané sur les deux axes de déplacement. Dans le cas d'une interaction gestuelle, dites «mid-air», on devra avoir un squelette de main ou un modèle de main le plus précis possible.

Ces périphériques virtuels peuvent être à leur tour composites ; dans le cas d'un écran tactile, on pourra par exemple différencier le fournisseur de doigts virtuels, la matrice tactile, et les interacteurs qui nous intéresseront, c'est-à-dire les doigts.

Pour atteindre le maximum de modifiabilité et afin d'avoir un impact limité des évolutions logicielles, il faut envisager l'évolution du matériel. On peut par exemple doter le doigt d'attributs tels que l'angle d'inclinaison, la surface en contact ou la pression exercée sur cette surface. [Hamon-Keromen, 2014].

Actuellement, les périphériques virtuels sont soit fournis dans les paquets logiciels du constructeur, soit n'existent pas, limitant ainsi les possibilités de modification et d'ajout d'interactions liées à un périphérique.

Dans le cas de l'application des quatre saisons, le périphérique virtuel de la souris est le modèle de la souris, composé de plusieurs variables :

- Une quantité de mouvement horizontale *dx*
- Une quantité de mouvement verticale *dy*
- Une quantité de rotation de la molette *dscroll*
- Un état booléen pour le bouton gauche *buttonLeftPressed*
- Un état booléen pour le bouton droit *buttonRightPressed*
- Un état booléen pour le bouton de la molette *buttonMiddlePressed*

Même si dans notre application, on utilisera seulement les quantités de mouvement et l'état du bouton gauche, ces autres informations sont présentes au sein du système d'exploitation pour pouvoir gérer la grande majorité des interactions avec une souris générique.

Les périphériques logiques, *logical devices*, restent, dans ce manuscrit, identiques à la notion de Buxton [Buxton, 2009]. Il convient de raisonner pour l'interaction en termes de fournisseur d'événement générique. On préférera donc une interaction ayant en entrée un périphérique de pointage plutôt que la spécification d'une souris. Par exemple, le curseur pourra être contrôlé par une souris, mais aussi un pavé tactile, ou encore, par une action de pointage extérieure telle qu'une interaction gestuelle.

Le périphérique logique est celui qui aura le plus grand impact sur la représentation du feedback immédiat de l'interaction. C'est sur celui-ci que l'utilisateur va ajuster son interaction.

Le principal intérêt de différencier le périphérique logique du périphérique virtuel est la possibilité d'utiliser un seul périphérique virtuel pour différents types d'interaction. On pourra alors associer des fonctions de transfert [Casiez and Roussel, 2011] complètement différentes, et ainsi obtenir des modalités distinctes à partir d'un seul périphérique. Par exemple avec un seul périphérique de reconnaissance de geste, tel que le leap motion, on aura une main informatisée, notre périphérique virtuel que l'on pourra associer à :

Un curseur représentation classique du périphérique logique de pointage en deux dimensions, pour faire de l'interaction dans un espace de conception en 2 dimensions

Un autre objet interactif plus proche d'une main complète pour naviguer dans un espace de conception en 3 dimensions.

On peut aussi associer, à type de périphérique logique, plusieurs périphériques virtuels. Le curseur de Windows, par exemple, peut-être contrôlé par la souris, le trackpad, ou encore un smartstylet.

Dans notre application, le modèle du curseur est notre périphérique logique. Il est intégré au système et possède les attributs de position absolue dans l'écran. Il n'existe pas de représentation pour avoir l'équivalence de l'état des boutons de la souris. Dans les OS standards, on utilise directement l'état de la souris plutôt qu'un report de ces informations dans un périphérique logique.

Le gestionnaire de périphériques, *input manager* sur la figure 3.5, sert à gérer la dynamicité des périphériques virtuels et logiques. Il est responsable de l'instanciation des périphériques virtuels en cas de branchement à chaud.

Même si le périphérique virtuel est statique il peut arriver que le périphérique logique doive être instancié dynamiquement. Reprenons l'exemple de l'écran tactile. Le «fournisseur de doigt» est statique, le doigt virtuel est dynamique, puisqu'il n'existe aucun moyen de suivre le doigt lorsqu'il n'est pas en contact avec l'écran, et le curseur associé au doigt sera lui aussi dynamiquement instancié.

Dans les quatre saisons, le curseur est toujours présent à l'écran, et on n'aura donc pas besoin de ce composant.

3.3.2 Gestionnaire de Chaînes de périphériques d'entrée

Le gestionnaire de chaînes de périphériques d'entrée a deux rôles :

- Gérer la reconfiguration du flux d'événements et de données en cas de changement du nombre de périphériques physiques en entrée.
- Aiguiller le flux d'information et identifier les composants aval concernés.

3.3.2.1 Gestionnaire de configurations d'entrée

Ce composant doit gérer le chargement et le passage d'une configuration d'entrée à une autre. Il est optionnel dans les systèmes ayant une configuration statique ne gérant pas de défaillance, ou dans les systèmes qui gèrent la configuration uniquement au démarrage. Il sert à refaire les liens entre les différentes chaînes d'entrée et *les techniques d'interaction globales* ainsi que les *zones sensibles*. Une implantation d'un tel composant peut être trouvée dans [Hamon et al., 2014b] .

N'utilisant que la souris dans l'application des quatre saisons, il n'y aura pas de gestionnaire de configuration d'entrée.

3.3.2.2 Fonctions de picking

Les *fonctions de picking* doivent délivrer l'information depuis les chaînes d'entrée aux *zones sensibles* et *techniques d'interactions globales* activées. Le rôle est généralement, là encore, assuré par les systèmes d'exploitation pour les périphériques standards tels que le curseur ou les entrées clavier. Ces fonctions peuvent être omniscientes, c'est à dire possédant une connaissance hiérarchique de toutes les *zones sensibles* et *techniques d'interaction globales*, ou récursives, en déléguant une partie de l'identification des zones activables à d'autres composants.

Dans le cas d'une application se servant des librairies java et donc par extension des librairies du système, les fonctions de picking sont gérées directement dans le Windows manager du système. Il n'y aura donc pas de composant spécifique pour les quatre saisons.

3.3.3 Techniques d'interaction globales

Les techniques d'interaction globales ont la particularité de ne pas avoir de *zones sensibles* associées ; elles peuvent donc être déclenchées sans condition particulière sur le designateur. Elles ont donc un spectre d'activation plus large que les widgets WIMP ou post-WIMP.

Elles servent principalement à traiter des événements simples, en les combinant, soit en nombre soit dans le temps, pour créer une commande haut niveau plus complexe. Elles sont souvent nommées transducteur dans la littérature [Accot et al., 1997],[Casiez and Roussel, 2011] Par exemple le composant qui prendra les événements émanant du curseur pour les transformer en événement *Clic*, *Double Clic* ou encore *Drag*, rentre dans la catégorie *technique d'interaction globale*s, car ces événements haut niveau sont générés indépendamment de l'activation des zones sensibles.

Là encore pour les quatre saisons, les techniques d'interaction utilisées sont celles fournies par le système d'exploitation et les librairies java swing.

3.3.4 Dialogue et noyau fonctionnel de l'application

Ce système, qui constitue le cœur de métier de l'application contient plusieurs processus que l'on pourra rapprocher des briques bas niveau des architectures ARCH [Kazman and Bass, 1994] et Seeheim [Pfaff and ten Hagen, 1985].

3.3.4.1 Zones sensibles

Les zones sensibles *Sensing Zones* sur la figure 3.5 englobent les composants interactifs standards de type widget WIMP, mais aussi composants post-WIMP standardisés tels que les boutons tactiles. Ces zones sensibles sont délimitées dans un espace, qui peut être graphique, mais aussi tactile ou encore sonore. Par exemple une zone sensible sonore pourra être délimitée en volume sonore d'entrée, un timbre de voix ou encore en hauteur de son tandis qu'une zone sensible graphique sera activée lorsqu'un périphérique de pointage entrera dans son champ d'activation.

Dans le cas des quatre saisons, les zones sensibles sont les quatre boutons de l'application. Le reste de la fenêtre ne réagit pas à la présence du curseur ni aux techniques d'interaction de la souris.

3.3.4.2 Fonctions d'adaptation

Les fonctions d'adaptation de dialogue *Functional Adapter* sur la 3.5 servent à transformer les événements haut niveau en entrée pour les rendre utilisables dans le cœur de l'application. Il correspond à la brique *Dialog Adapter* de l'architecture ARCH [Kazman and Bass, 1994]. Nous aurons besoin de ce cette brique pour rendre les événements indépendants de leur sémantique et ainsi faciliter le déclenchement d'action ou plus précisément le changement d'état par plusieurs moyens différents.

Lorsque l'on utilise des librairies et des interactions standards, les événements sont généralement tous traités, ce qui est le cas dans l'application des quatre saisons. Les seules interactions nécessaires sont le *mouseDown* pour engager un bouton, et *mouseUp* pour relâcher le bouton. L'événement produit lors de cette succession d'événement, l'*actionPerfomed* est quant à lui directement géré dans java et la librairie swing.

3.3.4.3 Dialogue et noyau fonctionnel de l'application

Ce composant constitue le cœur applicatif. Il sera normalement très composite, avec de nombreux appels vers d'autres systèmes. Il inclut notamment les parties classiques de l'architecture ARCH qui concerne le bas niveau. Nous ne détaillerons pas ce composant dans le manuscrit sachant qu'il n'est pas directement lié à l'ingénierie de l'interaction.

La figure 3.3 présente le modèle de ce composant pour l'application des quatre saisons.

3.3.4.4 Fonctions d'activation

Les fonctions relaient l'état du système pour permettre l'activation et la désactivation des composants interactifs. Par exemple le sous-menu «enregistrer» est grisé lorsqu' un document sous Microsoft Word ne contient pas de modifications courantes. Dès que l'utilisateur entame une modification, ce sous-menu se réactivera grâce aux fonctions d'activation.

les fonctions d'activation de l'application des quatre saisons, au nombre de quatre, permettent de n'avoir qu'un seul bouton activé à la fois et de désactiver les trois autres

3.3.5 Système de rendu

Une des particularités de cette architecture est la dé-corrélation du système de rendu du reste des composants. En partant du principe énoncé dans [Campos and Harrison, 1997], les entrées sont basées sur des événements tandis que les sorties sont basées sur des états. Le rendu doit donc observer l'état des différents composants nécessitant une sortie et composer en fonction de l'état de ces derniers.

Le système de rendu, dans notre exemple, est composé de la librairie java swing, associé aux fonctions du système d'exploitation permettant l'affichage.

3.3.5.1 Fonctions de rendu

Les fonctions de rendu servent à créer l'information qui sera présentée à ou aux utilisateurs. Elles peuvent être reliées à une ou plusieurs scènes de rendu. Elles ne doivent pas avoir d'intelligence propre, pas d'autonomie particulière. Leur but est de rendre un état du système. Une fonction de rendu associée à l'état d'un bouton aura plusieurs méthodes de rendues, associées aux différents états de ce bouton par exemple une liste non exhaustive :

Disabled : le bouton est grisé, car l'état du bouton ne permet pas l'interaction

Enabled : le bouton est dans un état d'interaction possible

Highlighted : le bouton est emphasé, pour souligner une présélection ou un survol par le curseur.

Engaged : le bouton est enfoncé suite à une action provenant de la chaîne de périphériques d'entrée.

Les fonctions de rendu constituent donc la spécification de l'affichage par rapport à un état observable défini.

Dans l'application des quatre saisons, les fonctions de rendu prennent l'état du système pour pouvoir afficher la saison courante sur un label placé au centre.

3.3.5.2 Feedback immédiat

Les fonctions de feedback immédiat sont des fonctions de rendues spécifiques qui s'appliquent aux périphériques logiques et aux techniques d'interaction globales. Ces deux types de composants n'ont pas de zone spécifique associée dans les différentes scènes de rendues et doivent produire un affichage de circonstance. Elles sont différenciées des autres fonctions de rendu, car la boucle interactive ne traverse pas tous les composants de l'architecture, mais seulement ceux nécessaires à l'interaction, en boucle courte. Cela permet de les placer éventuellement dans un processus indépendant qui pourra être prioritaire par rapport à d'autres fonctions non interactives.

Dans notre exemple, le feedback immédiat se résume à l'affichage du curseur ainsi qu'au rendu des zones sensibles, les boutons, qui est directement géré dans la librairie swing.

3.3.5.3 Scène de rendu

Les scènes de rendu sont chargées de la composition finale de l'information avant la transformation finale en signal physique des informations à l'utilisateur. Elles vont regrouper les différentes informations venant des fonctions de rendu. Il y aura une scène de rendue par modalité de sortie. Les scènes de rendu peuvent être composites. Par exemple la scène de rendu graphique pourra avoir une sous scène par fenêtre, une pour le bureau et une scène de composition finale. La scène de rendu sonore pourra avoir plusieurs sous-scènes de rendu si on veut faire de la composition sur l'oreille droite et gauche par exemple.

Dans le cas des quatre saisons, ce rôle est joué par le Windows manager qui s'occupe de la composition finale graphique.

3.3.6 Gestionnaire de chaînes de périphériques de sortie

Contrairement au gestionnaire de chaînes de périphériques d'entrée, le gestionnaire de chaînes de périphériques de sortie ne gère qu'une seule tâche, la gestion de la reconfiguration des flux d'événements et de données en cas de changement sur les périphériques physiques en sortie. Il devra donc stocker plusieurs configurations de lien entre les scènes de rendu et les différentes chaînes de périphériques de sortie.

Dans le cas des 4 saisons ce composant n'est pas nécessaire sachant que nous utilisons exclusivement la modalité visuelle.

3.3.7 Ensemble des systèmes de sortie

L'ensemble des systèmes de sortie est représenté par un rectangle gris sur le bas de la figure 3.5. Ces systèmes sont généralement totalement intégrés aux systèmes d'exploitation, soit de manière native, soit via l'ajout de composants logiciels fournis par les constructeurs. Ils comprennent :

- la chaîne de périphériques de sortie
- les pilotes et librairies associés aux périphériques
- Les périphériques de sortie

Ils font miroir aux systèmes d'entrée.

Dans le cas de notre application, tous les composants de cet ensemble sont directement gérés au sein du système d'exploitation, et ne seront pas détaillés dans les sous point suivants.

3.3.7.1 Chaîne de périphériques de sortie

L'ensemble de la chaîne de périphériques de sortie est similaire à celui de la chaîne d'entrée.

- Le composant *Virtual Device* sert de représentation informatique la plus fidèle au périphérique de sortie
- Le composant *Logical Device* est l'abstraction d'un type de périphérique, cela permettra par exemple de ne pas avoir à se soucier si l'affichage se fait sur une matrice de LED, sur un mur d'écran ou simplement sur un écran LCD classique. On aura ici un simple périphérique d'affichage.
- Le composant *Output Manager* gère la composante dynamique des périphériques d'un type donné.

3.3.7.2 Pilotes et Librairie

Comme pour leurs homologues en entrée, les périphériques de sortie vont servir de passerelle entre le monde numérique et le monde physique, qui lui est analogique par nature.

3.3.7.3 Périphériques de sortie

Les périphériques de sortie vont servir à la transformation finale, du signal informatique, en signal physique que l'utilisateur pourra interpréter avec ses sens.

Dans le cas où un périphérique fait à la fois de l'entrée et la sortie, il faudra prendre soin de différencier les deux chaînes dans le modèle d'architecture MIODMIT afin de faciliter la modifiabilité. La séparation entre l'entrée et la sortie d'un périphérique d'entrée sortie possède un autre avantage non négligeable : en cas de panne d'une des deux fonctionnalités, on pourra continuer à utiliser le système avec l'autre. Par exemple, un écran tactile pourra éventuellement être utilisé comme trackpad contrôlant un curseur sur un autre écran si l'affichage ne fonctionne plus.

3.4 Contextualiser MIODMIT : un processus intégré

L'architecture MIODMIT est un outil servant à guider le développement d'un système interactif critique. Nous proposons dans cette section un mini processus "mode d'emploi" pour pouvoir utiliser le modèle d'architecture MIODMIT de manière intégrée à un processus global.

3.4.1 Prérequis

MIODMIT sert à l'ingénierie du système et son utilisation se situe après les phases de prototypages. Afin de définir une architecture respectant les principes énoncés dans ce manuscrit, nous ferons face aux prérequis suivants :

- Une liste des périphériques d'entrée et de sortie du système
- Une description informelle des modalités et de leur technique d'interaction, obtenues par les phases de prototypage.

Les deux items précédents sont nécessaires pour utiliser MIODMIT mais pas suffisants pour réaliser une architecture de système interactif viable. Il peut et doit y avoir d'autres livrables lors du prototypage. De manière non exhaustive nous pouvons citer, les cas d'utilisation, les scénarios de test et autres tests d'utilisabilité. Ces livrables complémentaires aideront naturellement la réalisation d'un système fiable et utilisable.

3.4.2 Raffinage de l'architecture

Une fois les deux livrables résultants de la phase de prototypage fournis, on peut commencer à raffiner MIODMIT pour en faire une architecture instanciable.

Un des intérêts de MIODMIT réside dans sa flexibilité et la réutilisabilité de ses composants. L'adaptation de la configuration aux périphériques devra donc identifier quels sont les composants qui sont déjà codés, soit par le constructeur, soit dans d'autres projets disponibles. Le but est donc d'éviter de repartir de zéro à chaque nouveau projet, mais plutôt d'utiliser les briques précédentes, ou celles fournies par d'autres codeurs ayant réalisé des projets avec une base de modalités similaires, tout du moins ayant utilisé les mêmes périphériques.

Les principaux changements qui vont transformer le modèle d'architecture MIODMIT en une architecture adaptée au projet se situent sur la partie directement liée aux périphériques.

Premièrement, on aura une chaîne de composants par type de périphérique. Le contenu des ensembles *Drivers & Librairies* ainsi que *Input chain Device type x* peuvent grandement varier en fonction des périphériques. Du fait de l'absence de standards dans la réalisation de ces composants, pour pouvoir arriver à ses fins sur le développement du système interactif, il faudra adapter la configuration interne des composants.

On pourra avoir plusieurs périphériques gérés par un seul et même driver comme lorsque plusieurs claviers sont branchés à un ordinateur. Il faudra par contre lancer deux

instances du driver avec des paramètres différents si on veut brancher deux Kinnect sur un ordinateur.

La configuration interne de l'ensemble *Input chain manager* peut elle aussi varier en fonction des périphériques gérés à ce niveau. Là où, pour une souris, on aura un composant de chaque type (un périphérique virtuel, un périphérique logique et un gestionnaire d'entrée), pour faire de l'interaction tactile, il y aura toujours un seul périphérique virtuel, correspondant à la dalle tactile, mais les périphériques logiques seront aussi nombreux que le nombre de doigts reconnus, et donc possiblement dynamiquement instanciés.

La configuration interne du composant *Global interaction techniques* peut elle aussi être plus complexe que sur la figure 3.5. en effet, selon la complexité des techniques d'interaction générale, elles peuvent être amenées à être composites, ou à dépendre les unes des autres, et ne pas être seules sur le flux d'information. Par exemple un moyen d'avoir du *feedforward* comme dans [Vermeulen et al., 2013] sera de regrouper plusieurs techniques d'interaction au sein d'un système de décision, et de faire le rendu en extrayant les états des différents modèles.

Les autres ensembles de composants sont suffisamment génériques pour rester tel quel en termes de configuration interne avant l'implantation effective. Il peut néanmoins arriver de ne pas avoir besoin de certain d'entre eux, comme nous avons pu le voir plus tôt.

3.4.3 Vérification et validation

Une fois définie, on peut vérifier si l'architecture raffinée correspond aux besoins utilisateurs en termes de modalités et d'agencements. Si les besoins utilisateurs et l'architecture correspondent, on peut alors passer au processus d'implantation. L'intérêt d'utiliser un standard tel qu'AADL est la possibilité de générer automatique certaines parties du code, ainsi que de vérifier des propriétés notamment en termes de cohérence sur les flux.

3.4.4 Implantation

Enfin, une fois l'architecture finale terminée, il reste à modéliser les composants eux-mêmes et éventuellement les coder, si les modèles ne sont pas interprétables. Comme cette étape reste un processus itératif, si les tests des fonctionnalités ne correspondent pas aux besoins utilisateurs, on recommencera certaine des étapes jusqu'à la validation du projet. Si les composants ont été codés en respectant un formalisme permettant l'analyse de modèle, on peut intégrer, aux phases de test, des vérifications sur la viabilité des techniques d'interaction en termes d'utilisabilité, ou de possible incohérence entre les états d'interaction et leur rendu, ou d'autres surprises liées aux comportements autonomes.

3.4.5 Résultat sur l'exemple

La figure 3.6 présente MIODMIT raffinée pour l'application des quatre saisons.

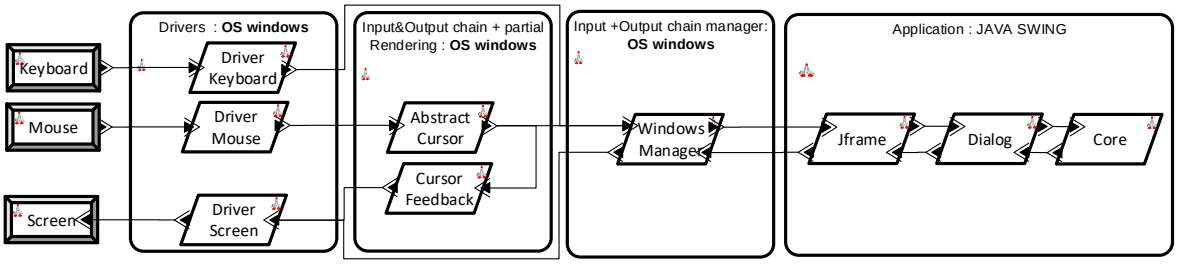


FIGURE 3.6 – Instance de MIODMIT adaptée à l'application des quatre saisons

Sur l'exemple des quatre saisons, la plupart des composants sont déjà intégrés au système d'exploitation et ne nécessiteront pas de recodage. Cet exemple est très simple en termes de composants. Le raffinement de l'architecture sera plus complexe sur un système complet. Les chaînes d'entrée et de sortie sont gérées directement par le système d'exploitation. On utilise le curseur système et le Windows manager pour pouvoir transmettre les événements provenant de l'interaction vers notre application. Du fait de l'utilisation de la librairie Java, le recodage des techniques d'interaction ne sera pas nécessaire. Le seul élément spécifique est l'application, et plus particulièrement le dialogue et le noyau fonctionnel.

On peut voir ici, que les chaînes d'entrée et de sortie sont fusionnées, car on n'a aucun contrôle au niveau de l'OS.

3.5 Propriétés assurées par MIODMIT

Grâce à MIODMIT et le couplage faible entre les différents composants, on peut ajouter ou supprimer des périphériques durant les phases de conception, ou le faire à postériori. La disponibilité du déclenchement des commandes du système est augmentée grâce à la possibilité de multiplier les modalités pour réaliser une tâche. Le couplage faible entre les composants et le fait que les entrées soient basées sur des événements et les sorties sur les états permet de répartir(au sens des systèmes répartis qui serait équivalents) les différents sous-systèmes. Il est donc tout à fait possible d'avoir une partie du système interactif sur une machine, disposant par exemple d'un écran tactile qui sera à porté de main des opérateurs, tandis que le reste sera localisé sur une ou plusieurs machines distantes ne faisant que de l'affichage. L'architecture permet de découpler plusieurs couches habituellement intégrées, et de fait peu modifiable. Cela permet d'augmenter l'interopérabilité du système complet, grâce à la possibilité d'adapter des composants venant de plusieurs fournisseurs.

Une fois l'architecture instanciée, si le formalisme AADL est respecté, de nouvelles propriétés viennent s'ajouter, comme la possibilité d'être testées avant même d'implanter l'intégralité des composants, on peut ainsi vérifier la cohérence entre composants.

La performance n'a pas été mise en avant ni testée dans le travail de ce mémoire. En effet, le travail s'étant concentré sur la spécification de système interactif critique, nous n'avons pas pu tester de manière exhaustive plusieurs architectures, et plusieurs implantations différentes pour démontrer un éventuel accroissement de performance. Cette propriété est néanmoins peu importante pour un système interactif critique étant donné que le temps de réaction d'un humain se situe aux alentours des 100ms. Vu la rapidité des systèmes interactifs actuels, la performance n'est un problème dans ce contexte. Pour avoir un ordre de grandeur, dans un système critique actuel, les boucles d'interaction des systèmes interactifs d'avion moderne sont de 33ms. Tant que le système est utilisable et respecte ces ordres de grandeur, il n'y a pas de problème de performance propre à l'architecture, mais plutôt sur celle des composants, ce qui est hors du périmètre de la thèse.

3.6 Conclusion

Ce chapitre a présenté MIODMIT, un modèle d'architecture permettant de concevoir des systèmes interactifs multimodaux compatible avec les mécanismes de sûreté de fonctionnement, qui pourront être utilisés dans des domaines critiques sous réserve de suivre un processus de développement normé tel que la norme DO178C (présentée en section 2.2.3) et ses suppléments pour l'aéronautique. MIODMIT permet de spécifier les composants d'un système interactif en les découpant en composants cohérents faiblement couplés. Ces composants permettent de définir un cadre décrivant les entrées et les sorties de chaque sous-système. L'architecture organise le flux d'information interne, en entrée et en sortie de manière à avoir une vision globale d'un système, quelle que soit sa complexité. MIODMIT vise à produire des systèmes interactifs fiables. Toutefois, la sécurité et la sûreté de fonctionnement ne peuvent malheureusement pas se définir simplement au niveau de l'architecture, nous verrons dans le chapitre 4 comment on peut atteindre certains objectifs de sûreté de fonctionnement dans la spécification interne des composants de l'architecture.

Le modèle d'architecture générique MIODMIT permet d'obtenir un système visant une certification. La certification d'un système complexe a pour but de convaincre les autorités de certification que le système est sûr de fonctionnement en prouvant que le processus, que chaque décision dans ce processus, et que chaque outil exploité dans les différentes phase du processus ont mené à un système plus sûr de fonctionnement. Le rationnel de chaque décision permet d'avoir une spécification complète, cohérente et non ambiguë du système interactif. Couplée à une spécification formelle, l'utilisation de MIODMIT permet de justifier les choix de conception qui concernent la délégation de fonction à tel ou tel

composant et surtout de définir un cadre pour le flux d'information en entrée comme en sortie d'un système interactif. La certification ne s'arrêtera pas à l'architecture et elle prendra bien entendu compte de l'implantation des composants.

Chapitre

4 Un environnement de développement et une plateforme d'exécution pour le prototypage de systèmes interactifs critiques

Sommaire

Introduction

L'ingénierie des systèmes interactifs critiques ne se résume pas aux méthodes de conception. Pour obtenir des systèmes fonctionnels, il faut aussi s'intéresser à leur exécution. Le chapitre précédent propose une architecture s'inscrivant dans une approche de conception zéro défaut. Cela ne suffit pas à adresser les fautes que nous avons évoquées dans la section 1.3. Nous devons aussi adresser les fautes commises lors de la conception du code et celles déclenchées en opération. Nous allons nous intéresser à la mise en œuvre.

Afin de mettre en œuvre la méthode supportée par l'architecture MIODMIT, nous avons besoin d'un environnement de développement des système interactifs critiques ainsi que d'une plateforme d'exécution répondant aux contraintes dressées par le domaine d'application : l'aéronautique.

Comme nous l'avons vu dans l'état de l'art, le domaine aéronautique impose et définit, via l'utilisation de la norme DO-178C [RTCA, a] et de ses suppléments DO 330 [RTCA, b],DO 331 [RTCA, c],DO 332 [RTCA, d] et DO 333 [RTCA, e], l'utilisation respective de méthodes formelles, d'outils certifiables, de langages à objets et enfin de modélisation. Compte tenu de l'absence d'alternative complète, l'état de l'art des simulateurs ne permettant pas de combiner ces contraintes, nous avons choisi d'utiliser le formalisme ICO, couplé à l'outil de modélisation et d'interprétation Petshop. Ce choix permet de s'inscrire dans une démarche de conception zéro défaut permettant de réduire les fautes lors du développement.

L'utilisation du formalisme ICO et de Petshop ne suffit pourtant pas à assurer la conception d'un système sûr de fonctionnement. Pour ce faire, il faut adjoindre des mécanismes propres à la sûreté de fonctionnement, par exemple de la redondance couplée à des systèmes de votes, ou encore un système de commande/moniteur (COM/MON) (présenté en section 2.4.1.2). Ces mécanismes se situent en dehors du périmètre de la thèse et font partie d'un domaine complet de recherche, mais ils dressent des contraintes supplémentaires qu'il faudra passer en revue.

L'aspect critique du système interactif apporte des contraintes telles que la nécessité de ségréguer temporellement et spatialement les processus (voir section 2.4.4.2). Cela évitera qu'une faute dans un processus n'impacte les autres processus de la pile mémoire, ou qu'un processus ne monopolise les ressources de calcul, entrainant alors des fautes en chaine.

Pour prouver les bonnes propriétés de l'approche, nous avons besoin d'une plateforme d'exécution qui permette à la fois l'utilisation d'un langage formel, la modélisation du système interactif, et la simulation d'un environnement réel aéronautique possédant les contraintes supplémentaires liées aux ségrégations spatiale et temporelle, nécessaires pour les logiciels embarqués. Cette contribution permet à la fois de traiter certaines des fautes commises lors du développement ainsi que des fautes naturelles apparaissant lors de l'opération, en suivant la classification de la figure 4.1.

Les plateformes de prototypage de systèmes interactifs multimodaux dans l'état de l'art ne permettent pas à la fois l'utilisation d'un langage formel, de modélisation, et l'ajout de contraintes liées à l'opération en milieu contraint. Nous ne traiterons pas de

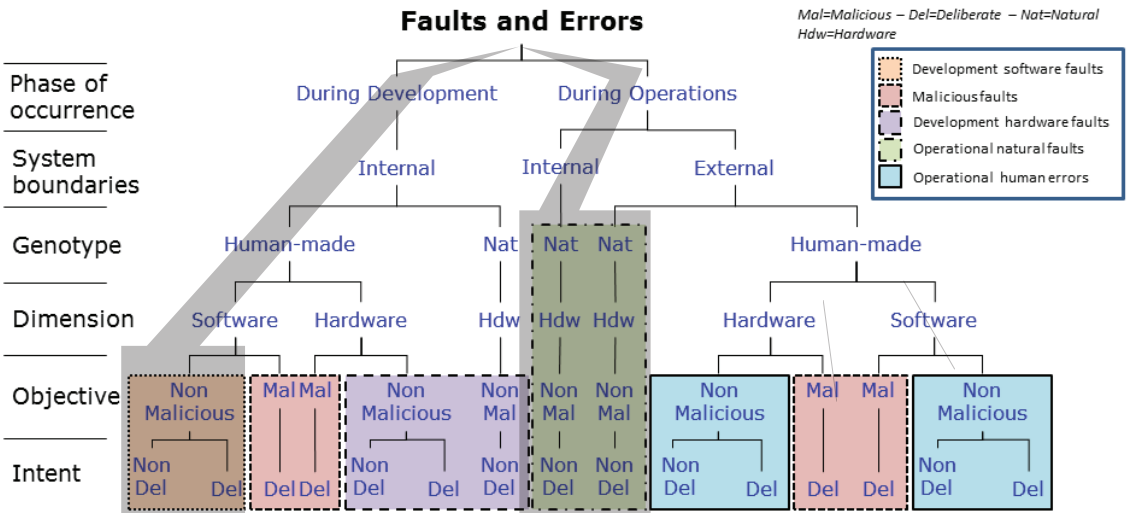


FIGURE 4.1 – Classification des fautes des systèmes informatiques

l'utilisation d'outils certifiables compte tenu du temps et des ressources nécessaires pour faire ce genre d'exercice qui ne peuvent se situer, de fait, qu'à la portée des industriels.

Nous avons donc décidé d'adjoindre à un environnement de modélisation et d'interprétation en temps réel d'un langage formel existant (Petshop), une sous-couche de simulation de système d'exploitation temps réel.

Trois parties formeront ce chapitre : premièrement une présentation succincte de l'environnement de modélisation et d'interprétation Petshop, ensuite un simulateur de système d'exploitation aéronautique temps réel, et enfin un exemple d'application de système interactif fonctionnant sur ce système.

4.1 Un environnement de développent de systèmes interactifs critiques : Petshop

4.1.1 Les besoins d'un outil pour supporter la modélisation avec la notation ICO

Notre méthode de conception des systèmes interactifs critiques s'appuie sur la modélisation des composants logiciels du système interactif à l'aide de la technique de description formelle ICO que l'on peut retrouver dans [Navarre et al., 2009] et [Hamon-Keromen, 2014]. Ainsi, le comportement des différents composants du système interactif est décrit par des modèles ICO. Un modèle ICO est constitué d'un Objet Coopératif (CO) qui a été associé une fonction de rendu et une fonction d'activation. Un Objet Coopératif est une instance d'une CO-classe qui implémente une interface Java et dont le comportement est décrit par un réseau de Petri haut-niveau appelé ObCS

(pour Structure de Contrôle de l'Objet, Object Control Structure). Pour rendre notre méthode et la notation ICO utilisables, il est nécessaire de proposer des outils supportant la modélisation de systèmes interactifs à l'aide de cette technique de description formelle [Navarre et al., 2001]. En effet, les systèmes interactifs sont complexes et nécessitent la modélisation de nombreux composants. De plus, les modèles ICO de tels composants présentent un grand nombre d'informations, il est donc difficile de modéliser sans un environnement logiciel permettant l'exécution. Cet environnement de conception a pour but de modéliser et spécifier des systèmes interactifs. Ainsi, il nécessite un certain nombre de fonctionnalités :

- L'édition de modèles ICO (et donc de leur interface en Java et leur ObCS), mais également des fonctions de rendu et d'activation et du rendu graphique final du système interactif.
- L'analyse statique des modèles ICO afin de renforcer la modélisation formelle .
- Le passage à l'échelle pour les modèles compliqués permettant de spécifier des études de cas réalistes telles que celles que nous présenterons dans les chapitres 6, 7 et 8.
- La gestion plusieurs modèles ainsi que leurs connexions.
- La simulation des modèles pour permettre l'évaluation des systèmes spécifiés et ainsi vérifier que le comportement spécifié correspond au comportement attendu.
- L'enregistrement de tous les événements se produisant pendant l'utilisation du système (actions utilisateurs, franchissement de transitions dans les modèles ICO, les jetons entrants ou sortants des places...) afin de générer des fichiers de journalisation (log) qui permettront par exemple l'analyse du système d'un point de vue efficacité [Palanque et al., 2011].
- Il doit également être suffisamment utilisable comme nous le montrent les travaux de [Barboni et al., 2003].

4.1.2 Présentation de l'outil

Nous décrivons dans cette section l'environnement logiciel PetShop qui permet la mise en œuvre de notre méthode. Il correspond à tous les critères énoncés dans la section précédente. L'outil PetShop pour Petri Net workShop [Barboni et al., 2010] est un environnement d'édition, de vérification et d'exécution des modèles ICO. Nous reprenons ici la présentation de l'outil PetShop dans sa dernière version qui a été réalisée dans les travaux de [Hamon-Keromen, 2014]. Cet outil est développé par l'équipe de recherche en systèmes interactifs critiques (ICS) de l'IRIT (Institut de Recherche en Informatique de Toulouse). La Figure 4.2 présente une copie d'écran de PetShop. L'agencement de l'espace de travail est personnalisable en fonction de l'activité du développeur. Cet agencement présente les vues suivantes :

1. Un navigateur de projets qui contient les modèles ICO, les interfaces Java de leurs CO-classes ainsi que les classes Java pour le rendu graphique du système.
2. Une fenêtre pour l'édition de modèles ICO, qui permet également leur visualisation

et leur analyse.

3. Une palette d'édition pour les modèles ICO, qui permet de glisser/déposer les différents éléments de la notation dans le modèle.
4. Un éditeur de code Java qui permet d'éditer les interfaces de CO-classes ainsi que les classes de rendu graphique du système.
5. Un débogueur.

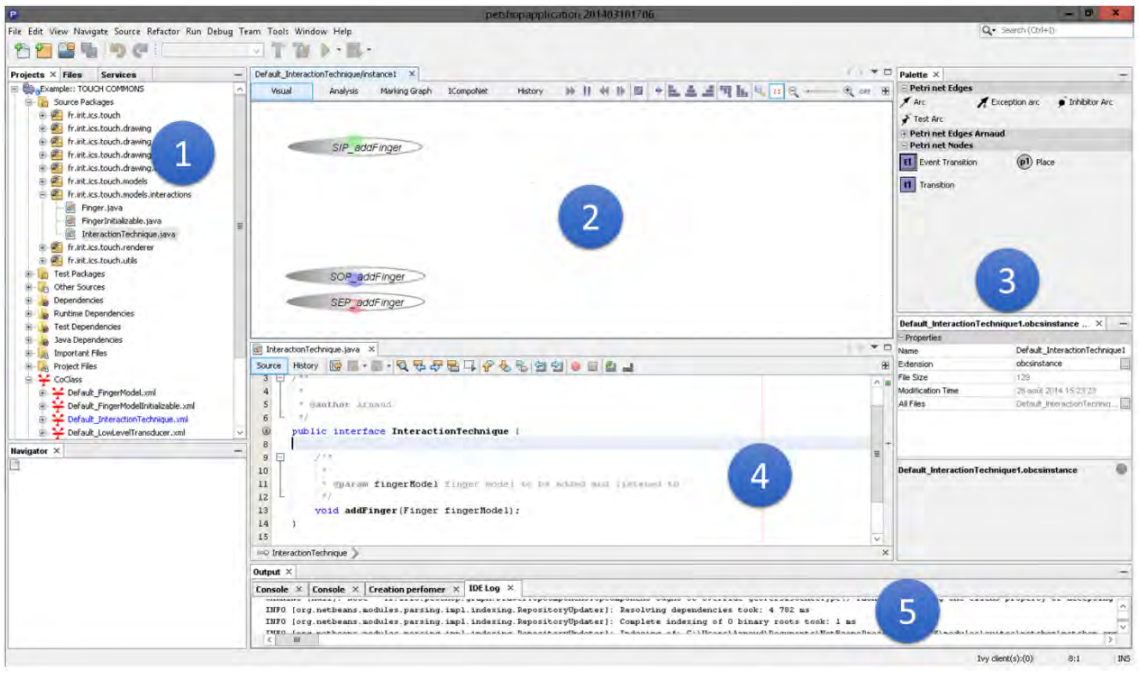


FIGURE 4.2 – Copie d'écran de l'environnement de PetShop [Hamon-Keromen, 2014]

4.1.3 Principe de fonctionnement

L'outil PetShop est développé avec le langage Java, ce qui rend possible son déploiement sur les systèmes d'exploitation les plus répandus tels que Windows, Linux ou MacOs.

Ses principes de fonctionnement sont représentés sur la Figure 4.3. Ainsi, la dernière version de PetShop est une application NetBeans. PetShop s'appuie sur certaines fonctionnalités de cet IDE pour intégrer un éditeur et un compilateur de code Java ainsi qu'un gestionnaire de projets et de versions qui facilitent grandement le travail en équipe ainsi que l'édition des classes Java associées aux modèles ICO. L'éditeur de modèles ICO utilise le résultat de la compilation des interfaces Java des CO-classes pour créer la structure des ICO. Les modèles produits sont sauvegardés dans des fichiers ObCS. L'interprète permet de simuler les modèles ICO tout en les liant avec les classes Java qui décrivent à la fois le rendu graphique du système interactif et les liens entre les modèles ICO et les

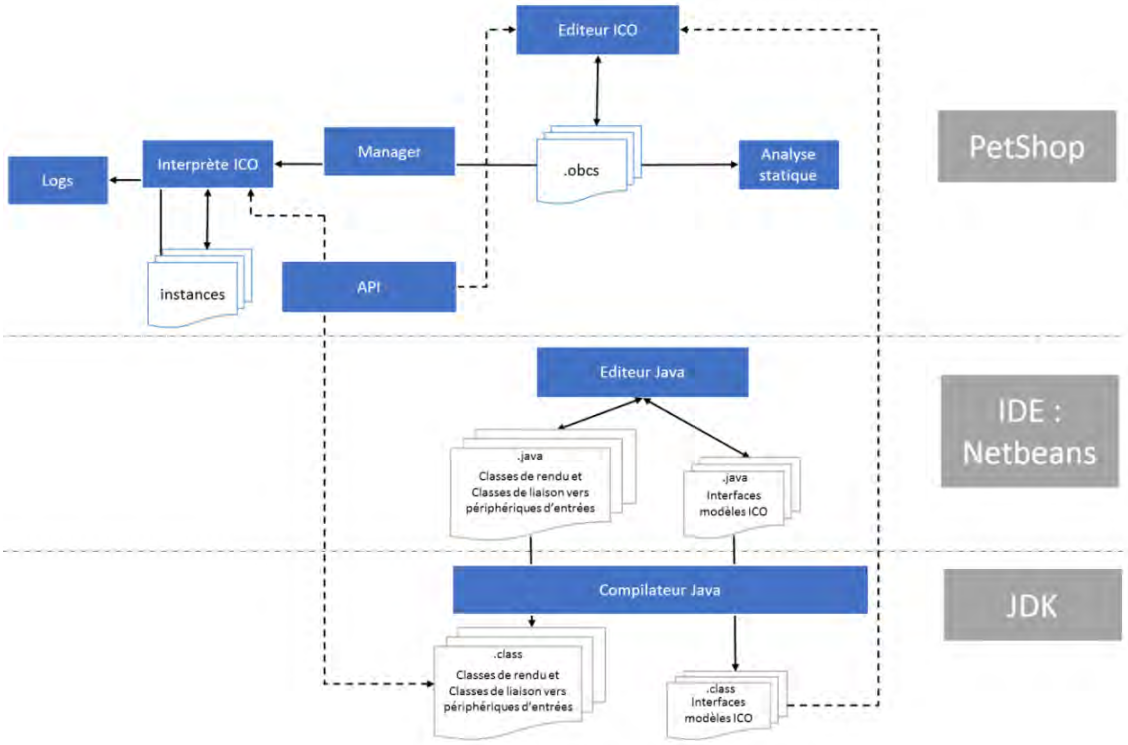


FIGURE 4.3 – Schéma illustrant les principes de fonctionnement de PetShop [Hamon-Keromen, 2014]

périphériques d'entrée et de sortie. Il assure également la communication entre les différents modèles ICO. Enfin l'interprète assure l'enregistrement de tous les événements des différents modèles ICO pour les sauvegarder dans les fichiers de journalisation (log).

Nous développons dans les sous-sections suivantes les différentes fonctionnalités de l'outil PetShop.

4.1.4 Édition, interprétation et débogage des modèles ICO

Afin de permettre le prototypage du système interactif, PetShop ne propose pas de séparation entre les activités d'édition et d'exécution des ICO [Navarre, 2001]. Ainsi, les modèles ICO sont en permanence édités, exécutés et analysés. On peut donc, par exemple, rajouter des places ou des transitions durant l'exécution des modèles ICO afin de modifier en « temps réel » le comportement du système interactif. L'outil permet de modéliser tous les éléments des modèles ICO en parallèle et au sein d'un même projet. Ainsi, il permet l'édition des CO (leur interface en Java et leur ObCS, autrement dit leur comportement en réseaux de Petri haut-niveau). Il permet également l'édition des fonctions de rendu et d'activation ainsi que le rendu graphique final du système interactif et des liens avec les périphériques d'entrée et de sortie.

Enfin, PetShop fournit, pour chaque instance de modèle, un moyen de contrôler son exécution au travers d'un contrôleur d'instances. Par défaut, l'exécution se réalise en mode automatique. Cependant, l'utilisateur peut choisir de suspendre temporairement cette exécution. Il pourra alors exécuter l'interprétation du modèle ICO en mode manuel et évaluer pas à pas le franchissement des transitions en choisissant les substitutions valides qui seront utilisées.

4.1.5 L'analyse des modèles ICO

L'outil PetShop propose également un outil d'analyse mathématique statique des modèles ICO. La Figure 4.4 présente une copie d'écran de l'analyse d'un modèle ICO. On peut ainsi retrouver :

1. La matrice d'incidence du réseau.
2. Les invariants de place.
3. Les invariants de transition.
4. Les places puits.
5. Les places sources.

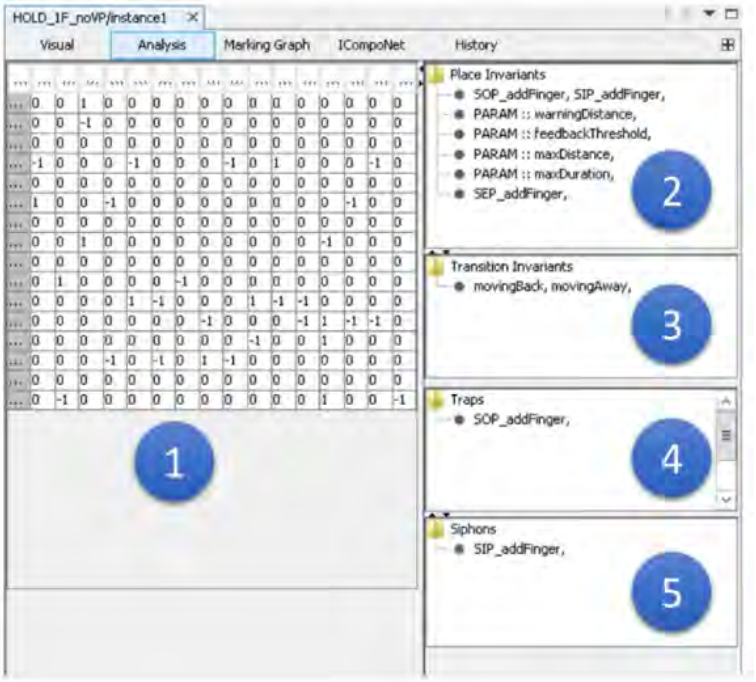


FIGURE 4.4 – Copie d'écran du résultat du module d'analyse des modèles ICO dans PetShop [Hamon-Keromen, 2014]

Cette analyse est disponible dès la création du modèle ICO. Elle ne nécessite pas de compilation et est mise à jour à chaque modification du modèle (et ce, même lors de l'exécution de celui-ci).

Outre les fonctionnalités d'analyse mathématique statique, Petshop offre la possibilité d'enregistrer toutes les évolutions des réseaux de Petri dans des log externes. Cela comprend les mouvements de jetons, mais aussi tous les meta événements tels que le changement de franchissabilité des transitions.

En utilisant une conversion et une vérification des XML, on peut analyser finement le résultat d'un test utilisateur [Palanque et al., 2011].

4.1.6 Synthèse sur Petshop

ICO et l'outil associé Petshop permettent de décrire des systèmes de manière complète et non ambigüe. Grâce à la notation formelle employée et la possibilité d'exécuter les modèles, ICO et Petshop sont particulièrement adaptés à la spécification de systèmes interactifs critiques.

Cependant, les contraintes de ségrégation spatiale et temporelle soulevées dans la section 2.4.4.2 ne sont pas mises en place dans cet outil.

4.2 Une plateforme d'exécution permettant la ségrégation spatiale et temporelle : ARISSIM

Nous avons développé un simulateur, ARISSIM (ARINC 653 Standard SIMulator) nous permettant de reproduire certaines des contraintes liées à ce standard. En effet, un des principes de base de la sûreté de fonctionnement consiste à s'assurer que les fautes d'un processus n'impactent pas les autres processus du système. Pour cela, il est nécessaire de ségréguer spatialement les processus (séparer leur espace mémoire) et temporellement (s'assurer que les processus ne fonctionnent pas en même temps et donc qu'ils disposent du maximum de ressources de calcul lors de leur exécution).

4.2.1 Principes de fonctionnement d'ARRISIM

ARISSIM est un simulateur de système d'exploitation respectant les principes de **communication** et de **partitionnement spatial** et **partitionnement temporel** du standard ARINC 653, présenté dans le paragraphe 2.4.4.2. Ce simulateur a été développé au sein du groupe TSF du LAAS-CNRS afin de pourvoir à des besoins de recherche et d'enseignement. La Figure 4.5 illustre le principe de fonctionnement du simulateur ARISSIM. CE dernier fonctionne sur les systèmes Unix, grâce aux mécanismes de contrôle des processus qu'ils offrent. Le paramétrage du simulateur est défini par deux fichiers de configuration. Le premier fichier permet de définir les différentes partitions qui doivent être exécutées par le simulateur ainsi que la période d'exécution qui leur est affectée. Le second fichier permet de définir les différents canaux de communication entre les partitions.

Chaque partition correspond à un processus Unix multitâche.

Le partitionnement spatial c'est-à-dire l'isolation entre les zones de mémoire affectées à chaque partition, s'appuie sur la notion de processus et les protections mémoire associées du système d'exploitation Unix. Le mécanisme *fork* permet la création d'un nouveau processus complètement indépendant et garantit l'isolation de la mémoire. Ainsi, les partitions ne partagent aucun espace mémoire dans leur pile respective.

Le partitionnement temporel est fixe durant l'exécution des partitions (il est fixé par le premier fichier de configuration). Il s'appuie sur l'utilisation des signaux Unix (SIGSTOP et SIGCONT) qui permettent l'ordonnancement des processus Unix. Ces deux signaux sont envoyés aux différentes partitions et permettent de stopper ou d'activer leur exécution. Les différentes partitions sont ainsi exécutées dans un ordre cyclique et ne partagent aucun temps d'exécution commun.

La communication inter-partition s'appuie sur l'utilisation des sockets Unix. La communication par *queuing* est assurée par une communication point à point s'appuyant sur la communication par sockets qui permet ainsi de garantir une communication par file

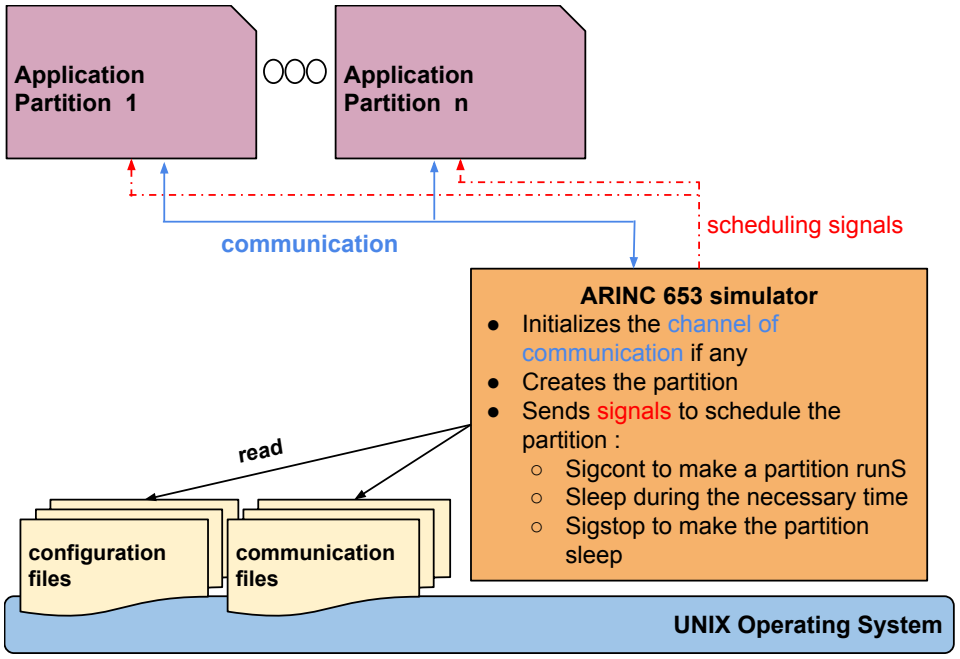


FIGURE 4.5 – Principe de fonctionnement du simulateur ARISSIM [Cronel,]

d'attente sans aucune perte de message. La communication par *sampling* est assurée également grâce à l'utilisation des sockets, mais est réalisée de manière à ce que seul le dernier message soit lisible et que la lecture ne supprime pas celui-ci. L'utilisation des sockets permet également une communication entre différentes machines sur lesquelles serait exécuté un simulateur ARISSIM en utilisant le réseau internet. Pour leur permettre d'utiliser les canaux de communications, le simulateur fournit aux partitions une librairie de fonctions permettant l'envoi et la réception de messages pour les deux modes (queuing et sampling).

Le code source du simulateur est disponible en libre accès depuis octobre 2014 [1]. Le simulateur a également été utilisé pour plusieurs projets étudiants en partenariat avec Airbus Defence and Space, le LAAS-CNRS et l'INP-ENSEEIHT où il a servi pour le développement d'une maquette de système embarqué satellite agile de prise de vue (tout le satellite pivote pour les prise de vue, pas seulement le module photo) sur une carte Raspberry Pi (Beaussart, et al. 2014), pour l'analyse de sûreté de fonctionnement de cette maquette (Bedoin, et al. 2015), ainsi que pour un projet récurrent à l'ISAE, où le simulateur a été embarqué dans un drone Parrot afin de reproduire un environnement contraint dans un drone civil.

1. https://github.com/ARISSIM

4.2.2 Limites du simulateur

Bien que le standard ARINC 653 impose de tuer les partitions ayant dépassé le temps imparti, nous avons développé une ségrégation temporelle souple où les processus sont simplement mis en pause. Le rationnel de cette décision vient du fait que nous nous intéressons aux processus interactifs qui n'ont pas forcément de temps mort ou de fin de boucle prévisible, du fait de l'interprétation des ICO par la JVM.

Le standard ARINC 653 impose l'utilisation de méthodes pour la communication interne au processus. Nous n'avons pas implanté ces méthodes à cause des contraintes fortes qu'elles imposent sur les processus développés.

4.3 Mise en œuvre de l'architecture autotestable sur la plateforme d'exécution

L'intérêt majeur de posséder un simulateur similaire au OS respectant la norme ARINC 653 est de pouvoir mettre en œuvre des techniques de sûreté de fonctionnement nécessitant du partitionnement qu'on retrouvera dans les avions. Un des premiers principes de la tolérance aux fautes consiste en la détection de ces fautes. En effet, si on ne détecte pas l'apparition d'une faute, on n'aura aucun moyen de mettre en œuvre des mécanismes de recouvrement de fautes.

Nous allons donc mettre en oeuvre une architecture COM/MON, dont les principes ont été présentés en section 2.4.1.2.

Afin de s'assurer que les deux composants COM et MON sont indépendants et non victimes de fautes au même moment, ou tout du moins, qu'une faute de l'un n'impacte l'autre, il faut qu'ils soient ségrégués. Pour illustrer les principes sous-jacents, nous allons reprendre l'exemple des 4 saisons.

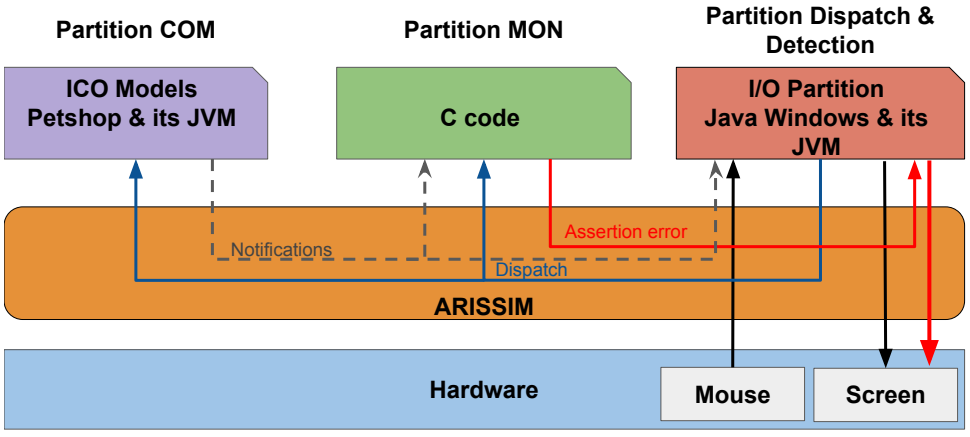


FIGURE 4.6 – Architecture logicielle et matérielle de la maquette de preuve de concept

La figure 4.6 présente la configuration logicielle de l'exemple des 4 saisons avec une architecture autotestable mis en oeuvre sur la plateforme d'exécution ARISSIM. Trois partitions sont configurées :

La partition COMmande (en violet) correspond au composant fonctionnel de notre application, le coeur applicatif des 4 saisons, modélisé avec le formalisme ICO et exécuté dans l'environnement Petshop

La partition MONiteur (en vert) reçoit en entrée les notifications de changement d'état du modèle ICO de la partition COM ainsi que les événements d'interaction réalisés sur la fenêtre Java et provenant de la partition Dispatch & Detection. Le MON effectue la vérification de chacune des propriétés de l'application, et détecte ainsi l'apparition de fautes par l'identification d'assertions non remplies

La partition Input/Ouput (en rouge) fait le dispatch et la détection. Il contient la fenêtre Java, présenté en figure 4.7, servant à l'interaction quatre saisons. Le dispatch transmet les événements de cette fenêtre aux partitions COM et MON pour les événements d'entrée et reçoit les notifications de la partition COM pour le rendu de la fenêtre. Il récupère les erreurs détectées par la partition MON, et les affiche sur l'écran (flèches rouges).

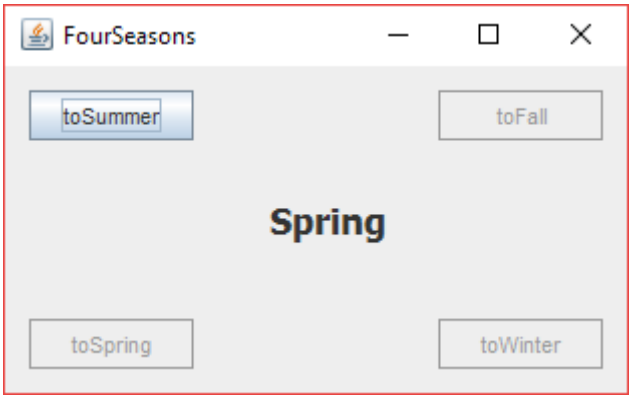


FIGURE 4.7 – Les Quatre Saisons

Les partitions sont mises exécutée tour à tour afin que l'intégralité de la ressource processeur leur soit allouée lorsqu'elles fonctionnent. La durée affectée à chaque partition est la suivante :

Commande : 10 ms

Moniteur : 6 ms

Entrées/Sorties : 4 ms

Ces durées sont expérimentales car il est actuellement difficile d'évaluer le WCET (Worst Case Execution Time) du fait de l'interprétation des réseaux de Petri. Ces valeurs ont étés choisies pour obtenir une manipulation fluide, les entrées sorties étant rafraichies toutes

les 20 ms. Pour rappel, dans les cockpits actuels, le rafraichissement se fait toutes les 33 ms sur les écrans centraux, et toutes les 100 ms sur les écrans latéraux. Si un événement est produit alors que la partition I/O n'est pas active, il n'est pas perdu, mais il sera simplement traité lors de l'activation de la partition.

Les sections suivantes détaillent les partitions COMmande et MONiteur

4.3.1 Partition COMmande

4.3.1.1 Comportement du COM

Le comportement de la partition COMmande est identique à celui de l'application présentée dans le chapitre précédent : une coeur applicatif permettant de passer d'une saison à l'autre à l'aide de boutons. Le modèle ICO correspondant est présenté en figure 4.8. Le franchissement des transitions est déclenché à la réception des événements produits par les boutons de la fenêtre Java (*toSummer, toFall, toSpring, toWinter*). Ces événements sont encapsulés dans des objets de communication Queuing et arrivent ainsi dans le bon ordre.

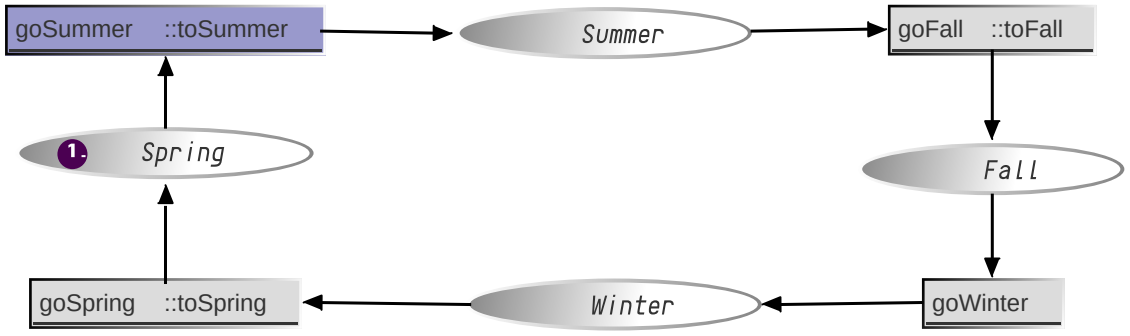


FIGURE 4.8 – Les Quatre Saisons, modèle en réseau de Petri de l'application

4.3.1.2 Principe de la notification à l'application

Une fonction de concrétisation du rendu observe les changements d'état du modèle (par exemple TokenEntered_Summer) pour déclencher le changement de rendu sur la fenêtre dans la partition I/O. Cette fonction permet aussi d'observer les méta-événements, c'est à dire, pas seulement les mouvements de jeton, mais la disponibilité d'un service. Elle permet donc de déclencher l'activation d'un bouton (e.g. le bouton *toFall*) lorsque son service associé est disponible (par exemple lorsque la transition *goFall* est franchissable).

Ces informations sont ensuite encapsulées dans un objet de communication Queuing et envoyées à la partition I/O.

4.3.1.3 Principe de la notification au MON

Plusieurs philosophies pourraient être mises en œuvre pour la communication entre le COM et le MON. plus le COM partage d'informations, plus le MON est à même de vérifier des propriétés complexes sur le COM. L'interprète Petshop utilise une librairie de communication spécifique à ARISIM pour effectuer la notification. Le déclenchement de la notification repose sur le même principe que le rendu qui sera présenté dans la section 6.4.4. C'est donc l'environnement Petshop qui communique les évolutions des modèles ICO qu'il exécute (e.g. lors des changements d'état).

4.3.2 Partition MONiteur

4.3.2.1 Comportement du MONiteur

Nous avons choisis des propriétés très simples. Le MONiteur de cette application vérifie 5 propriétés, exclusivement liées au changement d'état du dialogue de l'application :

1. Une seule saison est active en même temps
2. Après l'été vient l'automne
3. Après l'automne vient l'hiver
4. Après l'hiver vient le printemps
5. Après le printemps vient l'été

Ces propriétés étant toutes liées aux états de du modèle ICO décrivant le dialogue de l'application, la seule communication nécessaire du COM vers le MON consiste à notifier le MON des changements d'état (en queuing) et de l'état courant du COM (en sampling). Le MON va vérifier que les changements d'état ne violent pas les propriétés qu'il contient. Dans le cas où une propriété est violée, le MON notifie la partition Dispatch & Detection qui présentera l'information de violation de propriété sur l'écran.

Nous pourrions aussi vérifier que l'exécution du COM (le modèle ICO) correspond bien aux entrées venant de l'interface Java. Il faut alors recoder d'une autre manière le modèle comportemental (afin d'assurer la diversification et ainsi réduire les points communs de défaillance), le faire évoluer en suivant les entrées provenant de la partition dispatch, et comparer les comportements, ou tout du moins les états courants du modèle du COM et du comportement associé dans le MON.

4.3.2.2 Implantation

Les propriétés citées nécessitent le stockage d'un état courant du COM dans le MON. Les changements d'état nécessaires à la vérification d'assertions sont seulement les *tokenEntered* ainsi que le marquage courant du modèle.

Le marquage courant du modèle est transmis sous la forme d'un tuple de 4 entiers (int summer,int fall,int winter,int spring), correspondant au nombre de jetons dans chaque place. La réception de *tokenEntered_Summer* entraine l'exécution du code présenté sur la figure 4.9.

```
if( currentState!=SPRING){
                assertionViolation(SummerDidntCameAfterSpring);
                }
else{
                currentState = SUMMER;
                assertionState4season();
                }
```

FIGURE 4.9 – Assertion :Après le printemps vient l'été

On vérifiera, à chaque changement d'état, que le tuple ne contient qu'un et un seul "1" (pour l'unicité d'activation des saisons) par l'exécution du code présenté sur la figure 4.10

```
void assertionState4season(){
        if(summer>=0 &&fall>=0 &&winter>=0 &&spring>=0){
                        int sigma = summer + fall + winter + spring;
                        if(sigma!=1){
                                        assertionViolation(moreThanOneToken);}
                        }
                        else{
                                assertionViolation(ImpossibleTokenValue);
                        }
        }
```

FIGURE 4.10 – Assertion : Une seule saison est active en même temps

4.4 Bénéfices

Nous avons présenté ici une mise en oeuvre très simplifiée du concept d'application interactive autotestable dans un but pédagogique pour expliquer les concepts et leur mise en œuvre. Cette mise en œuvre peut être beaucoup plus complexe et adaptable à des architectures matérielles et logicielles spécifiques. Nous avons montré dans [Fayollas et al., 2016] comment appliquer ces concepts sur l'architecture des systèmes des cockpits interactifs compatibles avec le standard ARINC 661. Dans cet article, les assertions du MONiteur sont écrites en C. Elles portent sur les composants ARINC 661 et sur une application interactive. Le COM et le MON sont distribués sur plusieurs machines. On peut alors s'assurer que la détection d'erreur ne sera pas défaillante dans le cas où la machine supportant la

partition COMmande serait victime d'une défaillance impactant tout son système. L'implantation d'une architecture COM/MON distribuée est possible car ARISSIM permet la communication entre plusieurs machines.

4.5 Conclusion

Nous venons de présenter un environnement de développement permettant la mise en œuvre de l'architecture dans un milieu reproduisant les contraintes d'exécution des aéronefs.

L'utilisation de Petshop permet la modélisation formelle, l'analyse statique des modèles et offre la possibilité de s'inscrire dans une démarche zéro défaut. En couplant Petshop à un simulateur tel que la plateforme d'exécution ARISSIM, nous pouvons ajouter la ségrégation spatiale et temporelle, ainsi que des mécanismes de détection. Cela nous permettra, à terme, d'ajouter des mécanismes de recouvrement, non présents dans ces travaux. Avec cet environnement de développement couplé à la plateforme d'exécution, nous disposons d'un outil permettant de prototyper et de tester des systèmes interactifs critiques destinés à l'aéronautique. Dans la suite de ce mémoire, nous allons nous consacrer à la présentation de MIODMIT et à la description du comportement de ses composants au moyen du formalisme ICO. Les aspects de tolérance aux fautes liés à l'architecture ARISIM ne seront qu'évoqués, car leur mise en œuvre est en tout point identique à la présentation ci dessus.

Chapitre

5 Ingénierie des techniques d'interaction avec comportements autonomes

Sommaire

5.1 Introduction

Ce chapitre propose une méthode prenant en compte un type spécifique de fautes humaines (appelées habituellement erreur humaine en IHM, nous utiliserons le mot faute, car ces erreurs humaines peuvent être la source d'une faute ou erreur du système, selon le diagramme de causalité de la figure 1.8) : celles liées à la mauvaise compréhension par l'utilisateur de comportements autonomes du système interactif (*automation surprises* dans la littérature anglaise). Les fautes humaines peuvent conduire le système dans un état non souhaité par l'utilisateur, et peuvent potentiellement engendrer des catastrophes. Ce chapitre traite des fautes humaines en opération, représentées grisées dans la classification de Laprie [Laprie et al., 1995] sur la figure 5.1.

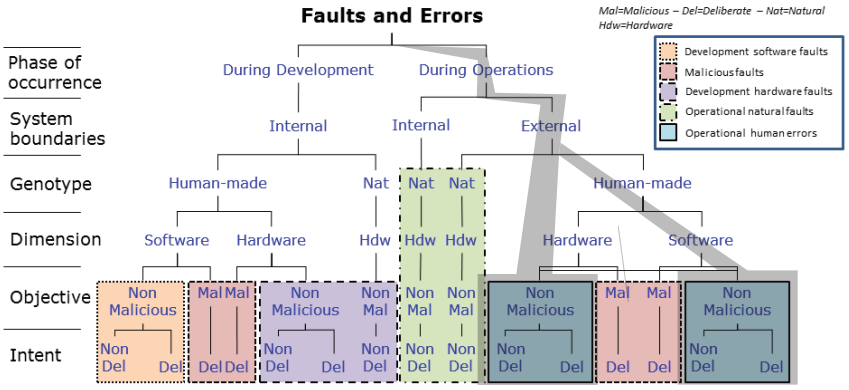


FIGURE 5.1 – Fautes humaines en opération, dans la classification des fautes des systèmes informatiques

Le développement de systèmes critiques interactifs et celui des systèmes grand public sont très différents. Les problèmes d'utilisabilité de l'interface, identifiés au cours des cycles de bêta test des systèmes grand public (proposés à un public ciblé), ne peuvent pas faire l'objet du même protocole concernant les systèmes critiques. Dans les systèmes grand public, les techniques d'interaction initiales peuvent être modifiées ou adaptées en fonction des retours des utilisateurs ou en fonction des données recueillies automatiquement quand ils utilisent le système. Pour les systèmes critiques, les techniques d'interaction doivent être étudiées à un niveau très fin. Nous proposons qu'elles bénéficient d'une phase dédiée dans les processus de conception de système interactif critique.

Il arrive parfois que les développeurs introduisent des incohérences entre le comportement réel du système et le rendu affiché à l'utilisateur. Ces incohérences sont une des sources majeures déjà identifiées (par expérience) des surprises liées aux comportements autonomes.

Une solution serait d'afficher un rendu pour l'intégralité des états et des variables du système, le soin étant laissé à l'opérateur de prendre les décisions basées sur l'état du système. Cependant, le mélange entre systèmes et interactions étant très complexe, une

trop grande quantité d'informations serait présentée aux opérateurs, rendant impossible la prise en compte. L'utilisabilité serait alors affectée. Par exemple, aucun des systèmes d'exploitation grand public n'affiche l'accélération courante appliquée au curseur de la souris. Cette accélération n'est pourtant pas un gain constant, mais une courbe, dépendante de leur éditeur respectif. Certains états doivent être donc masqués.

De plus, certaines tâches interactives (répétitives, sans intérêt décisionnel, etc.) peuvent être déléguées aux systèmes interactifs. Il faut alors trouver le bon équilibre entre les tâches à réaliser par le système, et celles relevant de l'opérateur. Cet équilibre doit être trouvé lors de la conception du système interactif pour placer le curseur sur chaque tâche en fonction de l'automatisation voulue. L'automatisation de certaines tâches est nécessaire et difficile à concevoir, à spécifier et à implanter.

L'approche décrite dans ce chapitre propose d'analyser, décrire et identifier les potentielles surprises liées aux *automation surprises*, qui pourrait conduire à une erreur humaine. Elle permet également d'identifier la localisation et le mode de modification des techniques d'interaction ou leur rendu, afin de réduire ces *automation surprises* et la dégradation de performance induite.

Nous démontrerons qu'en nous appuyant sur une classification des techniques d'interaction, nous pouvons supporter la conception et l'évaluation des techniques d'interaction grâce à l'utilisation d'une approche basée sur la modélisation. Nous pourrons identifier les problèmes sur l'intégralité des composants du modèle d'architecture MIODMIT, même s'il est plus facile de travailler sur les composants spécifiquement dédiés à l'interaction (par exemple le composant des *techniques d'interaction globales* sur lequel portera l'exemple filé de ce chapitre).

5.2 Introduction à l'exemple filé

Cette méthode a été élaborée en vue d'une utilisation dans les environnements critiques. Pour simplifier l'explication concernant l'utilisation de systèmes interactifs, nous prendrons pour exemple un système d'exploitation grand public qui comporte des similitudes avec les systèmes avioniques : le contrôle et le rendu du curseur sous Microsoft Windows 8 & 10. Le curseur du système, bien qu'utilisé quotidiennement par des millions de personnes, possède un comportement pouvant engendrer des *automation surprises*.

Pour simplifier l'écriture et la représentation des figures de cette section, nous avons, sur la figure 5.2, présenté l'ensemble des événements produits par le périphérique virtuel de la souris (voir section 3.3.1.3), qui sont transmis par le curseur. Ces éléments seront exploités dans les modèles des figures suivantes.

Nous prendrons en particulier le cas d'utilisation de l'ouverture d'un dossier, réalisé par un double clic sur l'icône de dossier. Ce cas d'utilisation nous permettra d'analyse la technique d'interaction globale du clic et double clic, ainsi que le rendu effectué par le système en suivant ces événement.

Physical Events (action on input devices)	Events Produced (by the mouse driver)
d : Button Down ; u : Button Up ; m : Mouse Move t : Time Out ; (system event)	C : Single Click ; DC : Double Click ; M : Move

FIGURE 5.2 – Events appearing in the automata of Figure 5.4 and Figure 5.6

Nous décrirons l'application du processus sur l'exemple dans un encadré gris.

5.3 Un processus pour prendre en compte les automatisations transparentes et éviter les surprises

Nous proposons dans cette section un processus, décrit sur la figure 5.3, permettant l'identification systématique des *automation surprises*. Il repose sur la description du comportement des techniques d'interaction, leurs analyses concernant le nombre d'états, d'événements produits et la correspondance de ces états et leurs rendus(la présentation effectuée à l'utilisateur selon les modalités de sortie exploitées par le système).

La figure 5.3 décrit les cinq grandes étapes de ce processus.

Les quatre premières étapes décrivent l'analyse et la modélisation, tandis que la dernière fait intervenir les aspects de design de l'interaction pour pouvoir éviter les *automation surprises*. Les trois premières étapes reposent sur le principe de la théorie de l'action de [Norman, 1988] et notamment sur la comparaison entre le modèle mental de l'utilisateur (*user's model*) et le modèle du concepteur (*design model*). Chacune de ces étapes est présentée dans les sections suivantes et les principes sont illustrés au travers de l'exemple filé, portant sur les interactions simples, mais représentatives des problèmes soulevés dans ce chapitre ainsi que de leurs solutions.

5.3.1 Première étape : Analyse de la technique d'interaction

L'analyse de la technique d'interaction du point de vue de l'utilisateur va permettre d'appréhender son utilisation. Le modèle mental de l'utilisateur (*user's model* dans [Norman, 1988]) (sa représentation mentale du comportement et de l'utilisation de la technique d'interaction) doit être décrit formellement de manière complète et non ambigüe, de sorte que, plus tard, il puisse subir la comparaison avec le modèle comportemental (*design model* [Norman, 1988]).

Le but premier de cette phase est la compréhension du le détail la technique d'interaction vue par l'utilisateur.

Un but secondaire réside dans la compréhension de la correspondance entre les attentes de l'utilisateur, en matière de comportement, et les techniques d'interaction potentielle-

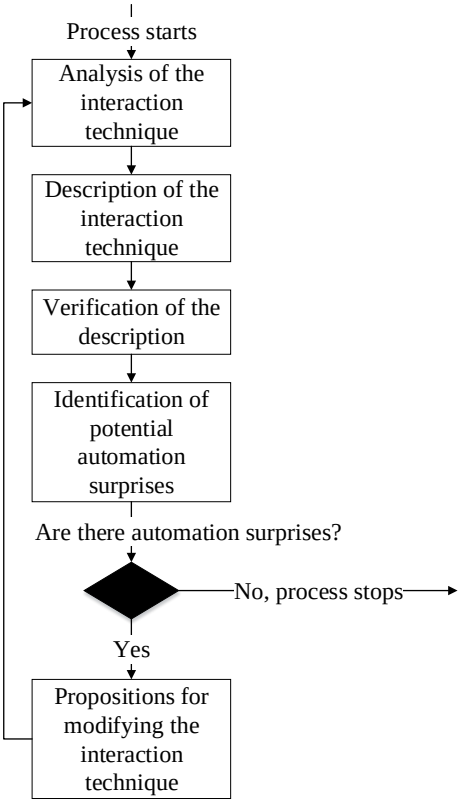


FIGURE 5.3 – Un processus de prise en compte des *automation surprises*

ment supportées. Concernant les techniques d'interaction telles que celles basées sur la gestuelle et le tactile, l'analyse peut inclure des entretiens ou des observations, pour analyser le comportement naturel des utilisateurs (quels gestes exécuteraient-ils, de quelles commandes vocales souhaiteraient-ils disposer) pour ensuite comparer les attentes en matière d'interaction à celles que le système implante en réalité.

Un autre but secondaire vise à comprendre pourquoi certains aspects des techniques d'interaction ne sont pas utilisés (par exemple, beaucoup d'utilisateurs de Microsoft Word ignorent qu'un triple clic sélectionne l'intégralité d'un paragraphe). L'identification de l'absence d'usage peut-être supportée par une analyse des logs, si le système offre la possibilité de loguer les interactions effectuées.

Dans notre exemple de la souris du système d'exploitation, cette première étape consiste à décrire en réseau de Petri le *user's model* de l'ouverture via un double clic sur un dossier, qui est une des interactions les plus rependues. Le modèle présenté figure 5.4 décrit en réseau de Petri le *user's model* le plus commun du fonctionnement de la souris dans les systèmes d'exploitation lorsqu'on demande à un utilisateur moyen comment fonctionne la souris lors de l'ouverture d'un dossier conformément à

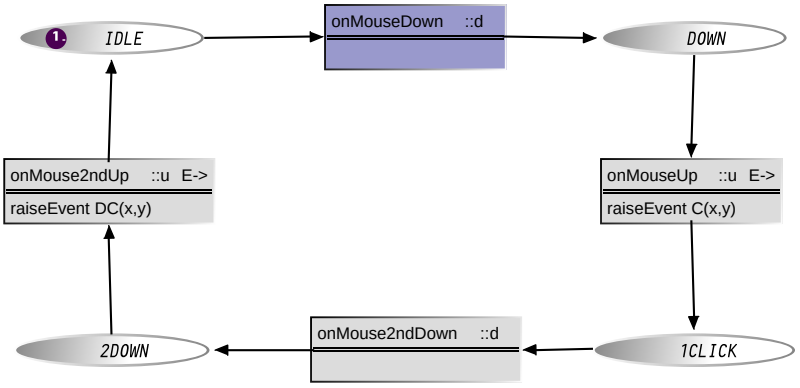


FIGURE 5.4 – La représentation mentale commune du modèle comportemental du curseur de Windows

son expérience. Dans ce modèle, l'événement DC *(Double Click)* est produit lors de la production de la séquence d'action utilisateur [d , u, d, u], soit la succession de deux pressions & relâchements du bouton gauche de la souris. Depuis l'état initial *(IDLE)*, la première pression (d) emmène vers l'état pressé *(DOWN)*, lorsqu'on relâche le bouton (u) on va vers l'état *(1 CLICK)* et l'événement *Click* est généré. Puis la deuxième pression (d) emmène vers *(2 DOWN)*. Enfin lorsque l'on relâche le bouton pour la deuxième fois (u) on revient à l'état repos et l'événement *Double Click* est généré.

Pour obtenir le rendu complet et non ambigüe de chacun des quatre états de la figure 5.4, il est important de présenter à l'utilisateur un rendu différent pour chacun de ces quatre états.

La figure 5.5 présente les quatre états correspondants au modèle mental de la figure 5.6 soit :

- l'état *(IDLE)*
- l'état *(DOWN)*
- l'état *(1 CLICK)*
- l'état *(2 DOWN)*

On peut alors remarquer que les états *(DOWN)* et *(1 CLICK)* ont des présentations identiques, ce qui est une source identifiée de confusion pour l'utilisateur [Pirker and Bernhaupt, 2011]. En effet, le seul moyen de différentier les deux états d'interaction est la mémorisation de la séquence d'actions ayant mené à l'état courant.

La confusion entre deux états ayant le même rendu est d'ailleurs une des sources des problèmes ayant conduit à l'accident Rio-Paris [Conversy et al., 2014].

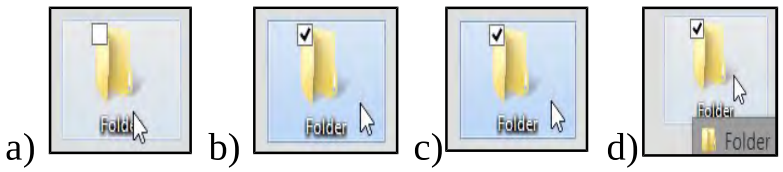


FIGURE 5.5 – Le rendu de l'interaction double clic sur un dossier dans Windows

5.3.2 Deuxième étape : Description de la technique d'interaction

En se basant sur les résultats de la phase d'analyse, les techniques d'interaction doivent être complètement décrites de manière complète et non ambigüe. Cette description peut-être, pour le moment, un texte, ou une vidéo, ou encore un prototype interactif. Dans les domaines où le coût d'une surprise d'interaction causée par une technique d'interaction est élevé (e.g. dans les jeux vidéo) nous recommandons d'utiliser des méthodes formelles.

Le réseau de Petri (figure 5.6) représente une partie du comportement de la souris (son périphérique logique partiel, voir 3.3.1.3) dans Windows 8 & 10. Les quatre états sont les mêmes que ceux décrits dans le *user's model* de la figure 5.4, mais les changements d'états, d'actions et d'événements produits sont plus nombreux.

- L'événement C *(Click)* n'est plus produit sur le premier *u*, mais directement sur le premier *d*.
- La flèche épaisse partant de *(1CLICK)* pour aller vers *(IDLE)* représente un comportement autonome qui survient si, après avoir fait un *clic* dans une fenêtre de temps donnée, l'utilisateur ne produit pas de second *clic*.
- La flèche fine partant de *(1CLICK)* pour aller vers *(IDLE)* correspond au comportement de l'utilisateur lorsqu'il effectue un clic et déplace ensuite la souris
- La flèche grise n'est pas connectée à l'un de ses bouts, elle représente le début de l'interaction *Drag*.

Ce réseau de comportement réel du *double clic* de Windows 8&10 met en exergue une évolution temporelle en plus de l'évolution liée aux pressions sur le bouton. Elle est doublée d'un comportement autonome, représenté par **la flèche épaisse**, qui est généralement absent du modèle mental de l'utilisateur lambda, puisqu'il ne perçoit que les rendus de la figure 5.5. Cela peut entrainer des *automations surprises* lors des tâches interactives, qui, bien que se situant à un niveau très concret, peuvent avoir un réel impact sur la performance des opérateurs, ces micro-tâches étant souvent répétées pour réaliser les macro-tâches. L'automatisation de cet exemple correspond au niveau 10 de Sherridan, soit dans le tableau 1.7, *"The computer decides everything and acts autonomously, ignoring the human"*

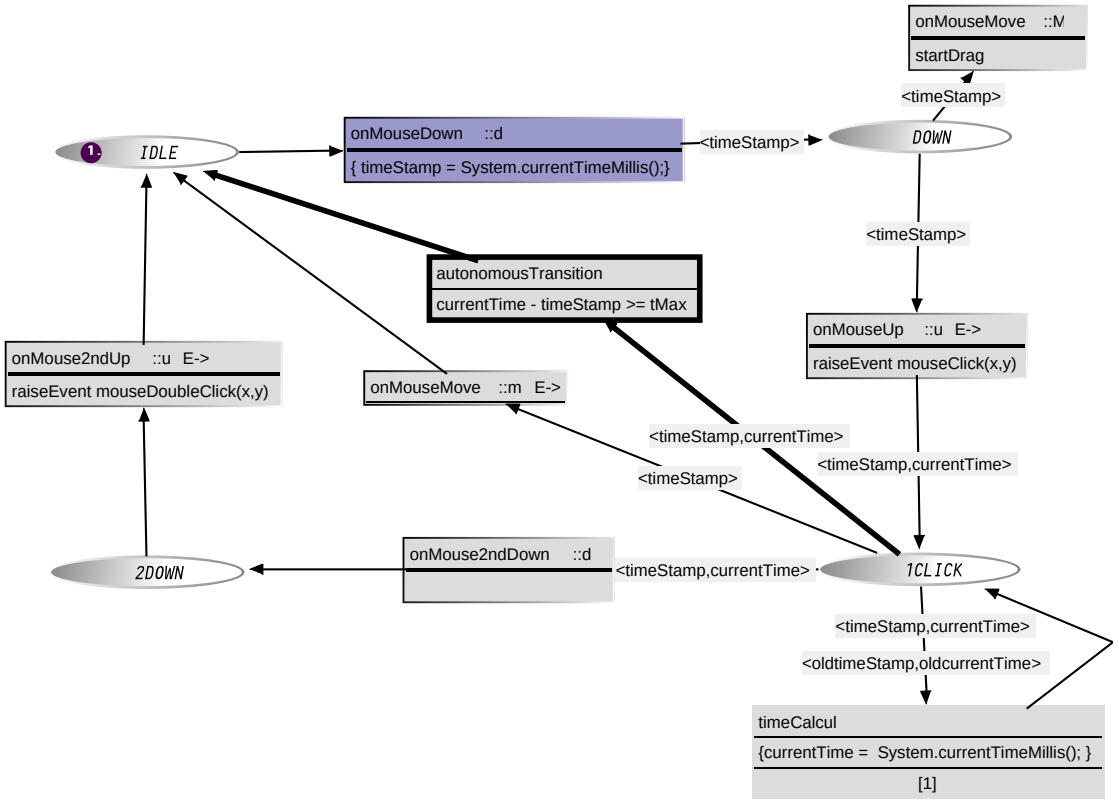


FIGURE 5.6 – Le modèle comportemental du curseur dans Windows

5.3.3 Troisième étape : Vérification de la conformité du design model avec le user's model

L'objectif de cette étape est de vérifier que la description des techniques d'interaction correspond aux observations faites dans la phase d'analyse. En d'autre termes, il s'agit d'analyser la correspondance (ou le conflit éventuel) entre le *user's model* et le *design model* (la description des techniques d'interaction). Cette correspondance peut se situer à un niveau lexical et syntaxique tandis que le niveau sémantique concerne le système sous-jacent.

Un conflit au niveau lexical correspond à une différence entre les actions disponibles dans le *user's model* et le *design model* et peut se traduire de deux manières :

- Une ou plusieurs actions dans le *user's model* ne sont pas disponibles sur le système.
- Une ou plusieurs actions dans le *design model* ne sont pas dans le *user's model*. Par exemple, un utilisateur ne pense pas possible l'utilisation du triple clic sur la souris alors que cette fonction est effectivement disponible et réalisable sur Microsoft Word.

Un conflit au niveau syntaxique correspondra à des enchaînements non valides. Par exemple produire deux événements « bouton pressé » d'affilés sur une souris : il faut en effet relâcher le bouton la souris avant de pouvoir à nouveau produire l'événement « bouton pressé ».

Un conflit au niveau sémantique existera dans le cas où un utilisateur pense qu'un événement produira une action alors que le système ne la réalisera pas. Par exemple, si l'utilisateur cherche à modifier le nom d'un fichier ouvert par ailleurs dans un autre programme.

Nous pouvons alors comparer les deux modèles et en déduire qu'il y a incompatibilité entre le modèle mental et le transducteur implanté dans Windows. Ces incompatibilités ont été formalisées dans [Combéfis et al., 2011]. Lorsque l'utilisateur veut produire le double clic, la séquence de changement d'état est alors un peu plus complexe que son modèle mental :

- Le temps impacte l'interaction ; si l'utilisateur est trop lent pour effectuer la séquence [d,u,d,u] le double clic ne sera pas produit.
- Le mouvement de la souris impacte l'interaction ; si l'utilisateur bouge la souris lors de la séquence, le double clic ne sera pas produit.
- Le double clic est produit directement après le second "d" ; si l'utilisateur veut interrompre le double clic alors qu'il a déjà pressé le bouton pour la deuxième fois, il ne pourra pas le faire sous Windows puisque l'événement aura déjà été produit.

Le rendu relatif à la production du double clic n'est donc pas suffisamment détaillé par rapport à l'ensemble des ses changements d'état.

5.3.4 Quatrième étape : Identification des potentielles surprises d'automatisation

L'identification des *automation surprises* est basée sur l'analyse détaillée du comportement effectif de la technique d'interaction, mais aussi sur l'analyse des divergences entre le *user's model* et le *design model*. Voici une liste d'heuristiques (éprouvées par l'expérience) permettant l'identification des surprises liées aux comportements autonomes [Accot et al., 1996] [Accot et al., 1997].

1. Les événements produits automatiquement par une technique d'interaction sans action explicite de l'utilisateur
2. Les événements consommés sans évolution de comportement résultante (par exemple lors de la réalisation d'un double clic, lorsque l'on relâche le bouton de la souris pour la première fois, l'événement « bouton relâché » ne déclenche aucune action)
3. Les événements produits plus tôt ou plus tard qu'attendu par l'utilisateur

4. Les états qui n'ont pas de rendu graphique (la technique d'interaction produit un changement d'état, mais aucun rendu associé n'est produit)
5. Les ensembles d'états qui ont un rendu identique
6. Les changements d'état autonomes déclenchés par des évolutions temporelles s'il n'y a pas de rendu associé

Dans l'exemple du double clic, on peut trouver :

- un problème de type 3 le double clic est produit sur le second « bouton pressé » tandis que la majeure partie des utilisateurs s'attendent à ce qu'il soit produit plus tard (lorsque le second « bouton relâché » est reçu)
- un problème de type 5, il n'y a pas de différence de rendu entre l'état DOWN et l'état 2 DOWN, bien que ces deux états ne permettent pas les mêmes interactions et ne déclenchent pas les mêmes actions
- un problème de type 6, la transition timée n'est pas représentée et n'a pas de rendu graphique associé, elle combine alors plusieurs des problèmes listés ci-dessus dans un seul modèle

5.3.5 Cinquième étape : Redesign pour éviter les surprises d'automatisation

En fonction de l'identification des surprises liées à un comportement autonome, il faudra effectuer les modifications dans cette cinquième étape, aussi bien au niveau de la chaîne d'entrée qu'au niveau des rendus graphiques (ou les deux). Cette étape nécessite le concours des équipes de design, mais aussi des ingénieurs.

Premièrement, il est important d'évaluer et de décider si la surprise identifiée risque de poser un problème d'utilisabilité.

Deuxièmement, il faut décider de la correction : faut-il changer le rendu de la technique d'interaction ou son comportement ? Et lorsqu'il est décidé qu'un changement est nécessaire, concevoir ce changement et l'implanter.

Il peut exister des automatisations transparentes prévues par design pour aider les utilisateurs ou pour accroitre leurs performances. C'est généralement le cas pour les fonctions de transfert [Casiez and Roussel, 2011], par exemple, celle de la souris qui augmente automatiquement la quantité de mouvement en fonction de l'accélération, rendant plus rapide la sélection des cibles distantes sur l'écran. D'autres surprises nécessiteront au contraire un redesign. Une stratégie possible (que nous avons adoptée sur l'exemple illustratif), consiste à ajouter ou à augmenter le feedback visuel pour représenter tous les changements d'état. Pour d'autres formes de techniques d'interaction, il est possible qu'un redesign de l'intégralité de la technique d'interaction, en ce qui concerne les commandes disponibles pour l'utilisateur, soit nécessaires (par exemple augmenter ou réduire l'ensemble des gestes et des commandes vocales disponibles). Nous ne préconisons pas de méthode pour régler ces problèmes dans cette thèse, mais nous proposons une méthode systématique pour identifier et prendre en compte ces problèmes.

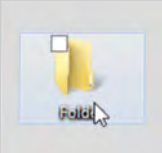
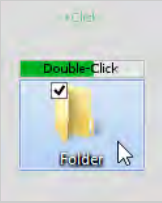
Graphical Rendering of State	Description
	IDLE state, the mouse has just entered the icon.
	DOWN state after a “d” event from the IDLE state; A label “+Click” is displayed, indicating that a click event has been fired; A timer bar is displayed at the top of the icon, indicating the remaining time to perform a double click. This timer bar decreases automatically.
	1 CLICK state after a “u” event; The ”+Click” label is disappearing (an animation is moving it towards the top of the screen); The time bar is decreasing.
	2 DOWN state after a “d” event in 1 CLICK state; A label “+Double-Click” is displayed; indicating that a double-click event has been fired; The time bar is disappearing; The double click is succeeded and a window named “Folder” has appeared.

FIGURE 5.7 – Exemple de rendus possibles pour le transducteur de Windows 8, qui rendraient l'automatisation plus transparente pour l'utilisateur

Dans l'exemple du double clic pour l'ouverture d'un dossier, le redisign proposé consiste en l'ajout de feedback sur les dossiers. Cela n'a pas été évalué avec des utilisateurs et a pour simple vocation l'illustration des concepts.

Les deux figures (5.7 & 5.8)proposent une solution à l'incompatibilité entre le *user's model* et le *design model*. La première (figure 5.7) présente un exemple de rendu pour chaque état du comportement du double clic.

- La première ligne décrit le rendu de l'état *(IDLE)*, lorsque le curseur de la

souris entre dans la zone de l'icône. L'icône est alors mise en surbrillance pour indiquer à l'utilisateur que la souris à pénétré la zone.

- La seconde ligne décrit le comportement de l'état *(DOWN)*, lorsque l'utilisateur a pressé le bouton de la souris. On fait apparaitre au-dessus de l'icône un label « +Click » ainsi qu'une jauge dynamique « double clic » qui permet d'indiquer à l'utilisateur le temps restant pour pouvoir effectuer le *double clic*.
- La troisième ligne décrit le comportement de l'état *(1CLICK)*, qui survient lorsque le timer poste son événement. Le label « +Click » s'estompe peu à peu et le temps restant pour effectuer le *double click* est représenté par une jauge diminuant.
- La dernière ligne décrit le comportement lorsqu'on a effectué la deuxième pression sur la souris dans le temps imparti du *double click*. la jauge se fige, et le label « +double clic » apparait.

Ce design permet alors de réduire les surprises liées aux comportements autonomes en réduisant le niveau d'automatisation à 7 sur l'échelle de Parasuraman de la figure 1.7 (s'exécute de manière autonome et en informe l'utilisateur).

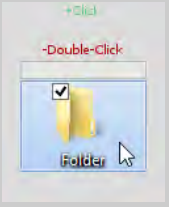
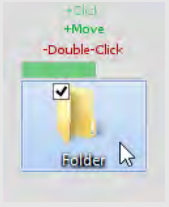
Graphical Rendering State	Details
	IDLE State after a "t" event when the timer has reached its limit, in the 1 CLICK state; The time bar is empty and is disappearing; A label "-Double-Click" is displayed, indicating that this event has not been fired.
	IDLE state after "m" event in the 1 CLICK state; A label "+Move" is displayed; indicating that a move event has been fired; A label "-Double-Click" is displayed, indicating that this event has not been fired;

FIGURE 5.8 – Exemple de rendus possibles pour le transducteur de Windows 8, qui rendraient l'automatisation plus transparente pour l'utilisateur dans le cas où le double clic n'est pas produit

La figure 5.8 représente les rendus possibles dans le cas où *double-click* pourrait être annulé durant sa détection ou à l'exécution.

- La première ligne présente le rendu de l'état repos, lorsque le timer a déclenché son événement, qui correspond à la flèche en gras sur la figure. Le label « - Double Click » indique à l'utilisateur que l'événement n'a pas été produit et donc, que le *double-click* est annulé.
- La deuxième ligne présente le rendu de l'état de repos lorsque l'utilisateur a déplacé la souris, et donc que l'événement m a été reçu. Dans ce cas, l'événement *double-click* est aussi annulé, comme l'indique encore une fois le label « -double clic ».

Pour tous ces comportements autonomes, le niveau d'automatisation avait été réduit au niveau 7 de l'échelle de Parasuraman [Parasuraman and Riley, 1997].

5.4 Illustration des potentielles surprises dans les cockpits

Nous avons présenté des problèmes génériques sur un exemple simple. Lorsqu'on passe dans un domaine d'application particulier, ces problèmes génériques se déclinent sur le domaine d'application. En reprenant le contexte applicatif de cette thèse, le domaine des cockpits interactifs constitue un cas où l'on relève plusieurs de ces déclinaisons :

Le *Control and display System* (CDS) constitue le système interactif des cockpits d'avions modernes. Il offre aux personnels navigants des fonctionnalités opérationnelles variées, allant de l'affichage des paramètres de vol, jusqu'à la possibilité d'interagir graphiquement sur les écrans en utilisant un ensemble de périphériques d'entrée appelé KCCU, *Keyboard and Cursor Control Unit*, entouré d'un rectangle rouge sur la figure 5.9.

Le pilote et le copilote disposent chacun d'un KCCU avec lesquels ils peuvent interagir avec les *Display Unit*, les écrans interactifs du cockpit au moyen de leurs claviers et *trackball*. Les objets interactifs et leur protocole de communication sont standardisés dans la norme ARINC 661.

Bien que la configuration matérielle des cockpits actuels permette l'implémentation de techniques d'interaction multi-souris telles que présentées dans [Accot et al., 1996], les designers de cockpit ont délibérément supprimé la possibilité d'utiliser deux souris en synergie. Cette suppression pourrait engendrer des *automation surprises* selon notre définition. Toutefois, il est important de comprendre que les opérateurs sont formés et que ces interactions auront donc étés apprises et qu'elles sont le résultat de compromis entre la sécurité innocuité et l'utilisabilité.

Trois déclinaisons d'interactions pouvant engendrer des surprises ont été identifiées :

- Lorsque le pointeur du pilote est sur un écran partagé, celui du copilote est désactivé. Si les deux pilotes ne communiquent pas et que le copilote ne voit pas le pointeur du pilote, il peut être surpris par l'absence de réponse de son propre pointeur.
- Les pointeurs n'ont pas une position stable dans un avion. Les coordonnées du

FIGURE 5.9 – Les deux KCCU d'un Cockpit d'A380, crédit : [wikimedia commons,]

curseur sont ramenées à une position de repos s'il n'est pas utilisé pendant 1 min. Si un pilote regarde le curseur à 59 s d'inactivité et reprend la main à 1min passée, il peut être surpris.

- Il n'y a pas de rendu du clic sur les curseurs. Si les pilotes ne communiquent pas, il peut y avoir des surprises quant aux actions effectivement réalisées par l'autre pilote.

Gardons présents à l'esprit que certains systèmes interactifs sont déployés au cœur d'environnements critiques. Les détails techniques d'implantation des interactions peuvent avoir un impact important sur la sûreté de fonctionnement de ces dispositifs, et à fortiori s'il y a des surprises lors de leur utilisation. Notamment les fautes et erreurs humaines en opération selon la classification de 5.1.

5.5 Conclusion

Nous avons décrit une méthode permettant l'analyse des techniques d'interaction à un niveau très concret, alors que les designers de systèmes interactifs se concentrent souvent sur les niveaux plus abstraits. Cette problématique du niveau micro (très concret) est généralement traitée par le système d'exploitation, c'est pourquoi les designers ont tendance à l'oublier, ou en tout cas à ne pas le prendre en compte. Dans les systèmes critiques, tout doit être recodé. Il est en effet impossible d'utiliser les composants logiciels grand public. Tous les composants utilisés doivent être certifiés pour le système.

Dans l'approche présentée dans ce chapitre, grâce à l'analyse explicite des techniques d'interaction, supportée par une notation formelle, nous permettons :

- l'identification systématique de toutes les *automation surprises* potentielles
- Le support d'une réduction des surprises liées au comportement autonome dans les techniques d'interaction
- L'identification des modifications nécessaires pour permettre la réduction de la dégradation des performances dues à des surprises

Cette approche a été illustrée par l'exemple du double clic dans le système d'exploitation de Microsoft.

En ajoutant cette approche à un processus de développement de systèmes interactifs critiques, on peut donc identifier la source de potentiels risques sur les automatisations des techniques d'interaction et ainsi réduire les risques liés à ces surprises.

Troisième partie

Étude de cas et validation sur une plate-forme opérationnelle

Introduction des études de cas et de la validation sur la plateforme

Nous allons illustrer les principes des chapitres précédents en nous appuyant sur trois chapitres.

Le premier s'inspire de l'exemple filé des quatre saisons et est une représentation complète de l'utilisation de l'architecture. Il nous permettra de compléter l'explication de certains aspects de MIODMIT, notamment le rationnel de découpage des composants bas niveau.

Le second permettra de prouver l'approche et sa résistance à la complexité dans une application de taille réelle qui explore de nouvelles techniques d'interaction pour les futurs cockpits : un radar météo possédant des techniques d'interaction complexe.

Le dernier chapitre présentera une plateforme permettant de faire la liaison entre l'approche théorique d'ingénierie présentée dans cette thèse, et de potentiels concepteurs d'interactions pour les cockpits de prochaine génération.

Chapitre

6 Un exemple générique et complet : les 4 saisons multi-événements

Sommaire

6.1 Introduction

Nous allons détailler les composants du modèle d'architecture MIODMIT présenté au chapitre 3, instanciés dans une application apparemment simple. Nous présenterons principalement l'aspect fonctionnel des composants. L'utilisation du formalisme ICO permet de s'inscrire dans une démarche de conception basée sur des modèles, et d'accéder à des outils d'analyses, détaillés en partie à la section 4.1.5. Les modèles et parties de modèle liées à l'initialisation ne seront pas présentés dans le chapitre.

Les modalités disponibles sont un peu différentes de celles de l'exemple filé des chapitres précédents. Nous nous inspirons de son cœur applicatif, tout en ajoutant des fonctionnalités.

6.2 Présentation informelle de l'application

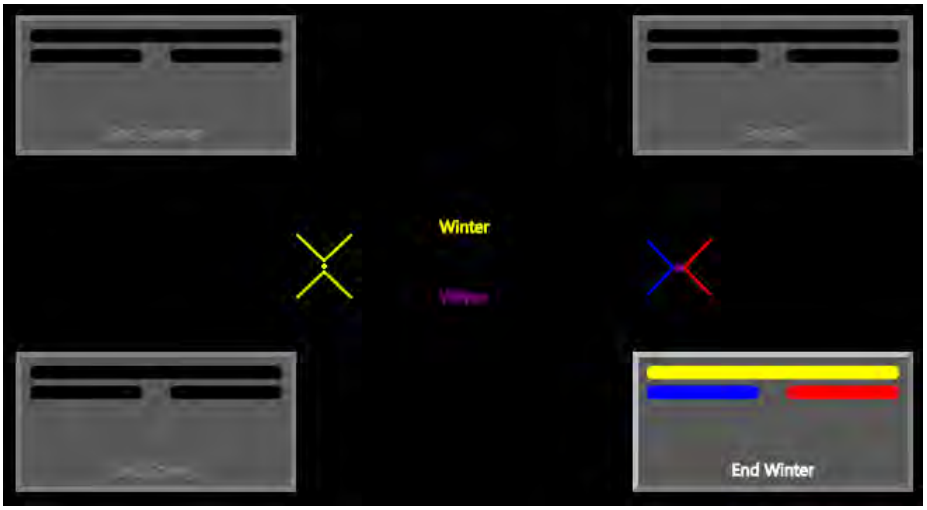


FIGURE 6.1 – L'interface graphique de l'application des quatre saisons multi-événements lorsque les saisons sont en hiver

Cette application, comme l'exemple filé des chapitres précédents, fait évoluer les saisons. Cependant, nous allons rajouter les *saisons ressenties*, faisant écho au vieil adage *"il n'y a plus de saisons"*. L'évolution des *vraies saisons*, au sens scientifique du terme, ne peut s'effectuer qu'en respectant l'ordre [été->automne->hiver->printemps] alors que les *saisons ressenties* évoluent plus librement. Néanmoins, elles font face à des contraintes :

- La *saison ressentie* ne peut avoir plus d'une saison de différence avec la *vraie saison* (en positif comme en négatif)
- La *saison ressentie* peut évoluer dans le bon sens comme dans le mauvais
- La *saison ressentie* ne peut pas sauter de saison lors de son évolution dans un sens comme dans l'autre

Les boutons servent à terminer une saison. En fonction du service utilisé, l'action réalisée par le bouton sera différente. La disponibilité des services permettant à l'utilisateur de faire évoluer les saisons est présentée dans le tableau de la figure 6.2.

Type saison	**Rendu associé sur les boutons**
Vraie saison	Oblong jaune
Saison ressentie sens positif	Oblong bleu
Saison ressentie sens négatif	Oblong rouge

FIGURE 6.2 – Relations entre service utilisateur disponible et rendu sur les boutons

Les moyens d'interaction seront :

En entrée :

- Une souris à un bouton
- Une souris à deux boutons

En sortie :

- Un écran
- Des hautparleurs

Cet ensemble de périphériques nous permettra d'illustrer une très grande partie des composants du modèle d'architecture MIODMIT. On notera tout de même trois principaux composants de 3.5, non représentés ou différents dans ce chapitre :

1. Il n'y aura qu'un seul driver pour les deux souris. Nous expliquions dans la section 3.3.1.2 que cet ensemble pouvait varier fortement en fonction du périphérique utilisé. Dans notre cas, un seul driver bas niveau est chargé de récupérer les événements des deux souris. Ce driver est chargé de faire le tri entre les différentes sources et de les retransmettre de manière indépendante au système suivant.
2. Il n'y aura pas d'*Input Configuration Manager* dans le système *Input Chain Manager*. Ne possédant qu'un set de périphériques, nous n'offrons pas la possibilité de reconfigurer l'interaction en entrée.
3. Cette application ne possède pas de technique d'interaction globale : les seules interactions se font au travers de manipulations type WIMP, en cliquant sur des boutons. Les événements de bas niveau suffiront à déclencher l'évolution des autres composants.

La souris à 1 bouton (associée au curseur jaune) permettra de faire évoluer les *vraies saisons* , tandis que la souris à 2 boutons (associée au curseur bleu & rouge) permettra de faire évoluer les *saisons ressenties*. Se présentent alors :

- Le curseur jaune pour le service jaune (évolution positive des vraies saisons), déclenché par l'unique bouton fonctionnel de la souris 1.
- Le curseur bleu&rouge pour le service bleu (évolution positive des saisons ressenties), déclenché par le bouton gauche de la souris 2

- Le curseur bleu&rouge pour le service rouge (évolution négative des saisons ressenties), déclenché par le bouton droit de la souris 2

Les saisons seront affichées sur un label au centre de la fenêtre, en jaune pour les *vraies saisons*, et en violet pour les *saisons ressenties*. La figure 6.1 présente la fenêtre de cette application. La synthèse vocale viendra compléter le rendu des *vraies saisons*.

Intéressons-nous à la conception, via MIODMIT, de cette application. Elle a une visée purement illustrative. Nous n'avons pas d'utilisateurs ni de tests à effectuer. Nous allons décrire l'instanciation de l'architecture ainsi que la modélisation des composants.

6.3 Instanciation du modèle d'architecture MIODMIT sur l'exemple

Trois types de périphériques sont utilisés : les deux souris en entrée, l'écran et les hautparleurs en sortie. On retrouvera sur la figure 6.3 trois cadres gris, représentant les chaînes de traitement de ces périphériques.

Un des intérêts de MIODMIT réside dans la réutilisabilité de ses composants. Certains de ceux présentés ci-après ont plus de fonctionnalités que nécessaire dans cette application du fait de leur utilisation dans d'autres projets. Pour illustrer l'architecture, nous voulons décrire formellement une grande partie des composants. Nous utiliserons certains services du système d'exploitation dans la gestion des périphériques de sortie. Nous n'utiliserons pas les composants liés aux souris du système d'exploitation compte tenu de l'impossibilité de gérer deux curseurs de manière fine.

La figure 6.3 présente l'architecture raffinée de notre application. Les systèmes bleu clair sur l'architecture n'ont pas été modélisés ; on utilise directement ceux du système d'exploitation.

Une fois l'architecture raffinée, nous pouvons nous intéresser à la modélisation des composants. L'utilisation du formalisme ICO nous permet d'obtenir des sous-états utilisables pour le rendu. Cela rend possible la décorrélation du comportement du système et de son rendu associé.

6.4 Modélisation des Composants

Nous présenterons les modèles des composants suivant la spécification ICO. Par convention, lorsqu'un rendu est effectué en utilisant le marquage (dynamique ou statique) d'une place, nous la nommons intégralement en majuscule ($PLACE$). Les places sans rendu commencent par une majuscule ($Place$), les transitions ainsi que les événements commencent par une minuscule ($transitionTruc$, $evenmtMachin$).

Le rendu associé aux places en majuscule sera décrit dans un encadré gris.

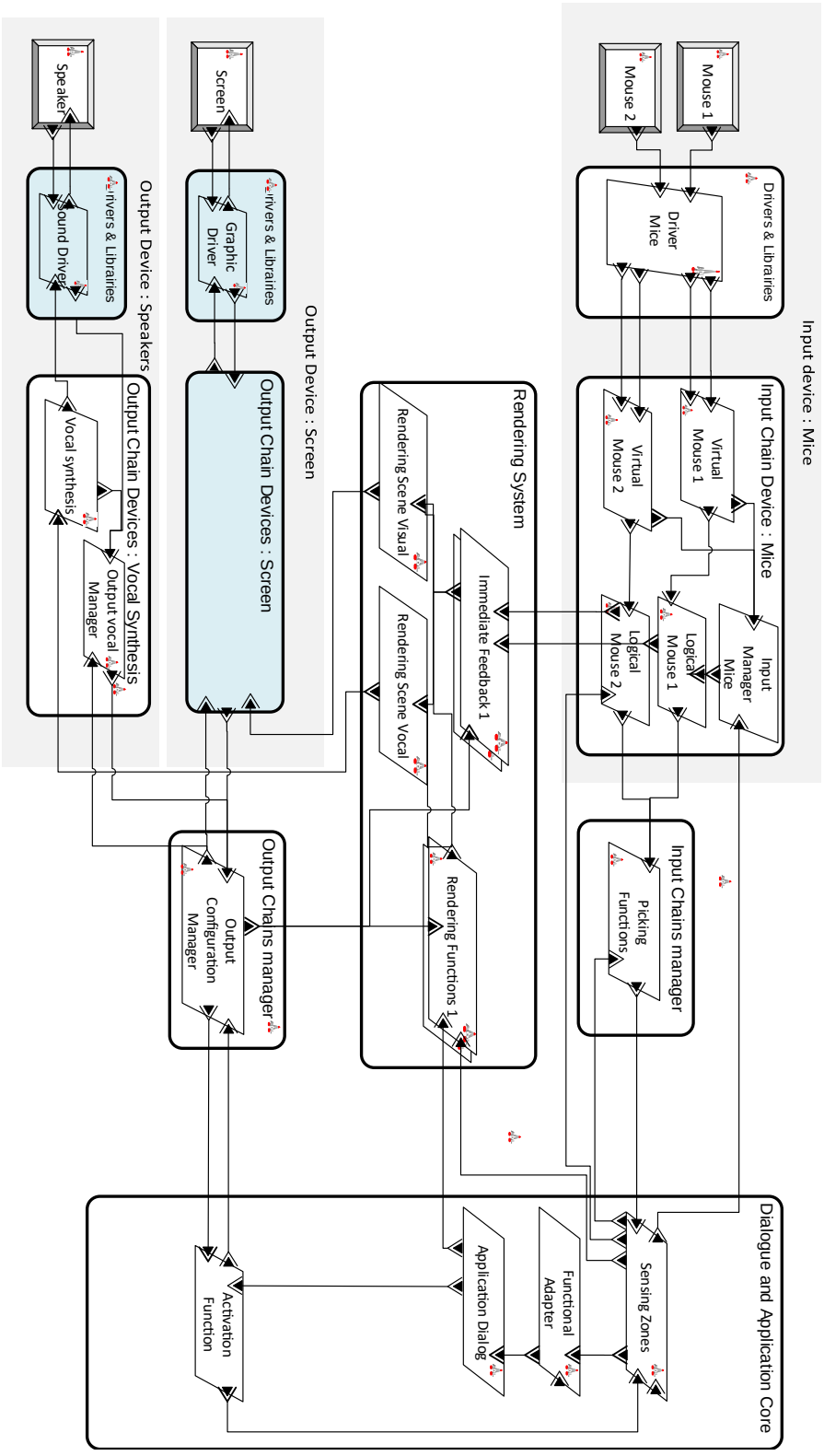


FIGURE 6.3 – Architecture de l'application des Quatre Saisons multi événements

L'application nécessite, en sus des modèles présentés au cours de ce chapitre, deux modèles d'initialisation :

Le modèle principal qui crée :

- La fenêtre (qui initialise automatiquement le moteur de rendu JavaFX)
- Le modèle du container principal
- La fonction de picking du container principal
- Les gestionnaires de chaîne de périphérique

Le container principal qui crée les zones sensibles avec leur *layout* fixé par constante.

Par souci de complétude, ils sont disponibles en annexe à la section 9.2. Ces deux modèles sont liés à la manière d'implanter l'architecture de l'application dans l'environnement de développement Petshop. Une implantation réalisée dans un environnement de développement différent mettrait en œuvre les fonctions de ces modèles de manière différente et idosyncratique à l'environnement. À l'opposé, la mise en œuvre des autres modèles dans un autre environnement devra être similaire en tous points aux comportements décrits dans ce chapitre.

6.4.1 Chaîne d'entrée

La chaîne d'entrée permet l'utilisation de deux souris supplémentaires branchées sur le système. Nous présentons dans cette section les composants relatifs à la partie de l'architecture présentée en figure 6.4

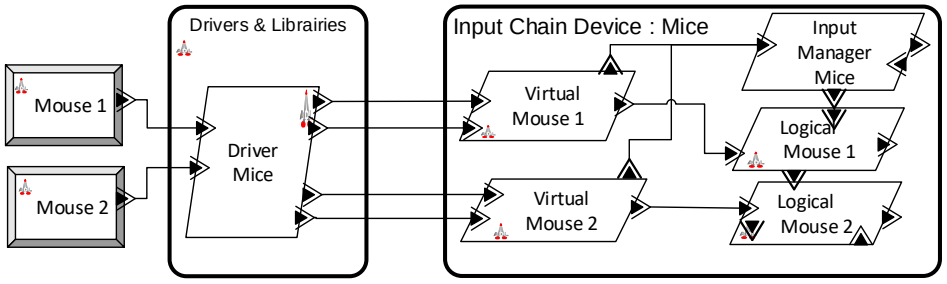


FIGURE 6.4 – Architecture de l'application des Quatre Saisons multi événements relative à la chaîne d'entrée

6.4.1.1 Périphérique d'entrée : deux souris

Nous avons utilisé deux souris Logitech modèle B100, possédant 3 boutons et une molette (boutons droit et gauche et une molette cliquable).

6.4.1.2 Pilote et librairie des souris

Les systèmes d'exploitation grand public offrent tous une gestion basique des périphériques de pointage. Les souris utilisées ne nécessitent normalement pas d'installation supplémentaire de driver particulier pour pouvoir être utilisées.

Sous Windows, afin de faciliter l'identification des souris sur les ports USB par rapport aux autres périphériques de pointage, nous avons installé le driver Logitech setPoint, qui permet de transformer le nom générique des souris "souris HID" en "HID-compliant Optical Mouse". setPoint ne nous sert qu'à identifier les souris, nous n'utilisons pas les autres fonctions du logiciel associé.

Sous Linux, les souris sont automatiquement identifiées en "Logitech USB Optical Mouse".

Le comportement par défaut des OS est d'associer tous les périphériques de pointage branchés, à un unique curseur système. Pour obtenir deux pointeurs dans nos applications, il faut donc passer outre le système. Il existe une possibilité de gestion de multiples curseurs sous Linux via le package *xinput*. Néanmoins, nous cherchons à modéliser la majorité des composants pour les décorréler de la plateforme. Nous avons donc intégré cette partie en Java.

Concernant la partie pilote, nous avons utilisé une librairie Java existante : Jinput. Elle permet de récupérer les événements directement produits sur les bus USB, sans les transformations du système d'exploitation. En développant un Pilote en Java utilisant cette librairie, nous pouvons récupérer les événements, et les trier pour les associer à un périphérique virtuel. L'instanciation se fait automatiquement par les modèles ICO au travers d'appels de services. La boucle de rafraichissement des valeurs du périphérique est directement intégrée à ce composant logiciel.

Sous Linux il est possible de "décrocher" les souris du curseur système, toujours grâce à *xinput*. Une fois l'application initialisée, le pointeur système est contrôlé par les périphériques classiques de l'ordinateur (souris, touch pad etc.). Les deux pointeurs supplémentaires fonctionnent uniquement dans la fenêtre JavaFx, et sont contrôlés par les souris génériques Logitech.

En résumé, la partie pilote, codée en java, nous permet d'extraire les événements du bus USB pour les rendre utilisables en JAVA. Nous n'avons pas fait de repackaging de notre implantation, il n'y a donc pas de librairie associée à la solution de cette application.

Nous ne faisons pas de rendu sur ces modèles, l'intégralité des actions étant transférée sur les périphériques logiques et virtuels.

6.4.1.3 Chaîne de périphérique liée aux souris

Les souris virtuelles sont une représentation informatique des souris physiques. Dans notre cas les souris utilisées sont identiques. Le périphérique virtuel associé modélise les états et les transitions en conséquence. Il est présenté sur la figure 6.5. Ce modèle rend

compte de l'absence de corrélation physique entre les différents composants de la souris. Par exemple, il n'y a pas de lien entre le capteur de rotation de la molette et le bouton de la molette.

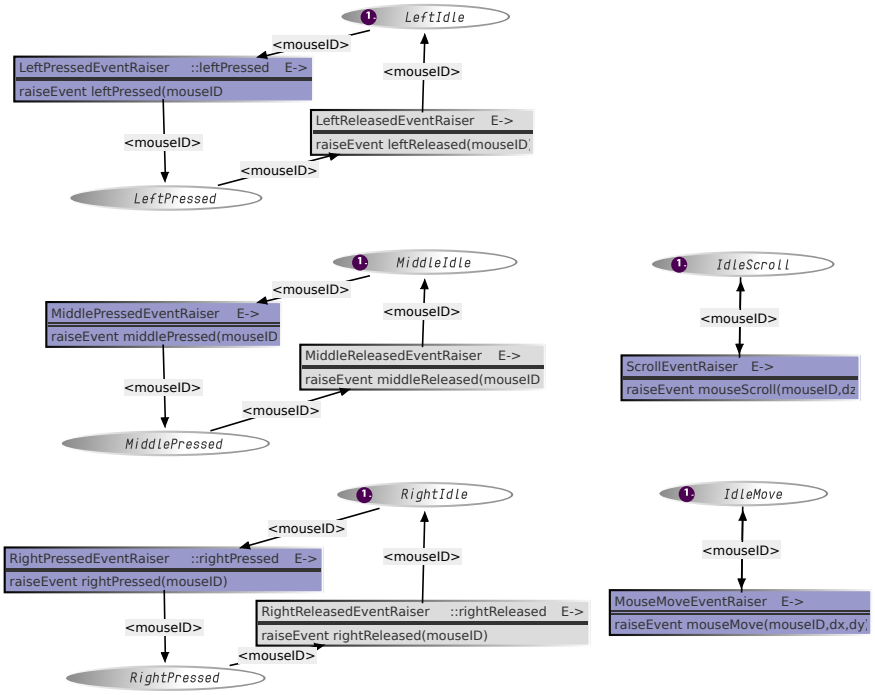


FIGURE 6.5 – Modèle de la souris virtuelle associée à la souris Logitech à 2 boutons et une molette cliquable

Nous ne faisons pas de rendu sur ce modèle, la charge de représentation informatique étant déléguée à la souris logique. Ce type de modèle pourrait néanmoins engendrer un feedback en vue d'introduire, par exemple, un retour haptique sur une surface tactile.

Les souris logiques modélisent l'objet informatique associé au périphérique. La représentation commune utilisée est le curseur. Bien que nous ayons deux curseurs différents, seul le rendu diffère, la modélisation des deux curseurs est identique et est présentée en figure 6.6.

La partie haute du modèle gère les événements liés aux clics. Pour des raisons de décorrélation sémantique avec le périphérique physique, nous avons nommé les états [Engagé/*Engaged*] et [Relâche/*Idle*] plutôt que Cliqué et Relâché, la notion de clic n'ayant pas de sens pour un curseur informatique.

La partie basse du modèle de la figure 6.6 gère la transformation des quantités de mouvement en coordonnées absolues. C'est donc dans ce modèle que se trouve la fonction de transfert (ici un gain constant de 1). Ce modèle fixe des limites de taille de fenêtre non

modifiable, notre fenêtre étant non redimensionnable. Pour ajouter cette caractéristique, il faut adjoindre un service permettant de changer les limites de tailles de fenêtre contenues dans les places *LimitX* et *LimitY* et l'appeler lorsque la fenêtre est redimensionnée.

Nous n'utilisons qu'un seul bouton de la souris 1. Nous aurions pu créer un modèle de curseur à un bouton, mais, pour des questions de duplication des modèles, il est plus simple d'ignorer les événements produits que de créer un deuxième modèle.

La séparation du périphérique virtuel et du périphérique logique permet tout particulièrement d'utiliser plusieurs représentations informatiques pour un même périphérique physique. Par exemple, dans une autre application, nous avons utilisé simplement les boutons des souris, sans associer de curseur.

Le feedback immédiat associé au curseur se sert du marquage de la place *CURSOR_POSITION* pour effectuer les mises à jour de la position du curseur dans la fenêtre. Dès qu'un jeton (contenant les coordonnées absolues mises à jour) arrive dans la place, un nouveau rendu met à jour l'affichage du curseur.

Le curseur associé à la première souris est jaune tandis que le curseur associé à la seconde souris est bleu et rouge, avec un point violet au milieu. Le design des pseudocroix est tel que si les curseurs sont au même endroit, il n'y a pas superposition de l'affichage.

FIGURE 6.6 – Modèle du curseur

Le gestionnaire de souris s'occupe de l'initialisation des modèles, des abonnements ainsi que de l'affectation des événements aux zones sensibles.

Il est notamment responsable de la boucle de mise à jour des affectations des événements, que l'on peut voir en vert sur la figure 6.7. C'est la boucle de rafraichissement des interactions liées aux curseurs. Elle participe à l'utilisabilité, car si elle est trop longue, il risque d'y avoir des événements envoyés aux mauvais composants. Si elle est trop rapide, elle risque de monopoliser les ressources matérielles.

La partie bleue du modèle gère l'abonnement et le désabonnement des widgets aux événements en fonction de la présence du curseur dans leur zone. Pour cette raison, il doit garder en mémoire le dernier widget *pické*.

Nous n'effectuons pas de rendu dans ce modèle. Les mécaniques de ce modèle servent à l'utilisabilité, mais pas directement à l'interaction. Effectuer un rendu impacterait la lisibilité.

6.4.2 Gestionnaire de chaîne d'entrée

Il n'y a pas plusieurs configurations possibles pour l'interaction dans cette application, nous trouverons dans ce système seulement les fonctions de *picking*. Elles font habituellement partie intégrante du *windows manager*, mais nous les avons modélisées en raison de l'impossibilité d'effectuer un réglage fin ainsi que pour intégrer la multimodalité.

Le *picking* est l'action qui consiste à déterminer sur quel objet de l'interface se trouve un point en particulier. Ce bloc doit donc faire le lien entre les périphériques logiques et les objets de la présentation que sont les *widgets*. Ce bloc est aussi relié à la partie dialogue et lui notifie les *widgets* qui sont *pickables*, c'est-à-dire à la fois visibles et interactifs.

Nous avons fait le choix de fonctions de picking récursives qui parcourent une arborescence composée des zones sensibles pour leur affecter les événements. Cela permet d'éviter une connaissance globale des zones sensibles et de raisonner en nœud d'éléments interactifs, en suivant la philosophie des *widgets* JavaFX [1] pour les zones sensibles. Les attributs d'une zone sensible parente, telles les activations/désactivations, seront ainsi transmis aux zones filles sans appel manuel de fonction .

Nous avons réutilisé les fonctions de *picking* des travaux d'Arnaud Hamon [Hamon-Keromen, 2014]. La description des fonctions de *picking* de son manuscrit est reproduite en annexe de ce mémoire en section 9.1.

Comme pour le modèle précédent, nous n'effectuons pas de rendu. Les mécaniques de ce modèle servent là encore à l'utilisabilité, mais pas directement à l'interaction. Les *widgets* pickés peuvent néanmoins changer d'état lorsque les fonctions de picking leur signifient qu'ils sont pickés. Ils peuvent ainsi effectuer leur propre rendu.

1. http://www.oracle.com/technetwork/java/javase/overview/javafx-overview-2158620.html, accès en juin 2017

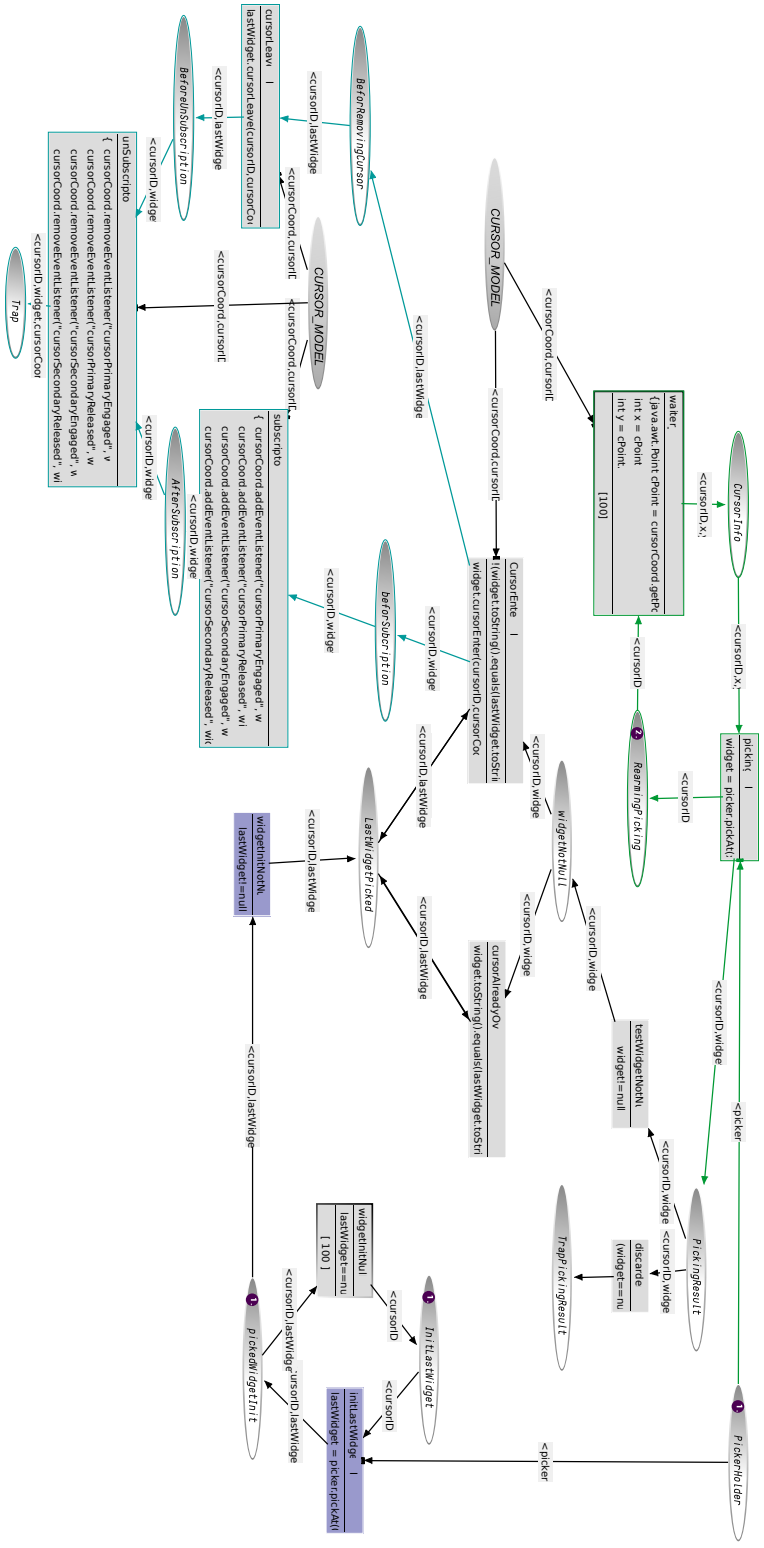


FIGURE 6.7 – Modèle partiel du gestionnaire de souris : boucle de rafraichissement en vert, abonnement dynamique en bleu

6.4.3 Dialogue et cœur applicatif

Nous présentons dans cette section les composants relatifs à la partie de l'architecture présentée en figure 6.8

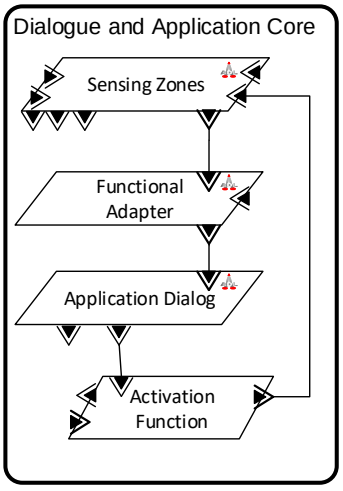


FIGURE 6.8 – Architecture de l'application des Quatre Saisons multi événements relative au dialogue et coeur de l'application

6.4.3.1 Les zones sensibles

Les boutons de cette application sont les seules zones sensibles disponibles. L'intégralité de l'interaction se fait via les boutons ; il n'y a pas de technique d'interaction globale comme le *clic* ou le *double-clic* dans notre application. Nous présentons trois modèles partiels du comportement des boutons.

Le premier, en figure 6.9, décrit la partie relative à la gestion de l'activation des trois services proposée à l'utilisateur. Les services actifs sont représentés par des jetons dans la place *AVAILABLE_EVENTS_ID* tandis que les services indisponibles sont dans la place *UNAVAILABLE_EVENTS_ID*. L'activation ou la désactivation d'un service se fait via l'appel de méthode *setEventEnable(bool enabled, int eventNumber)*. Cette activation est déclenchée par les fonctions d'activation (présentées en section 6.4.3.4).

Cette partie est spécifique à ces boutons qui produisent des événements différents en fonction des interacteurs les activant. Le rendu est associé à la disponibilité de ces services :

Vraie saison pouvant évoluer dans le sens positif est représentée par un oblong jaune lorsque le marquage de la place *AVAILABLE_EVENTS_ID* correspond

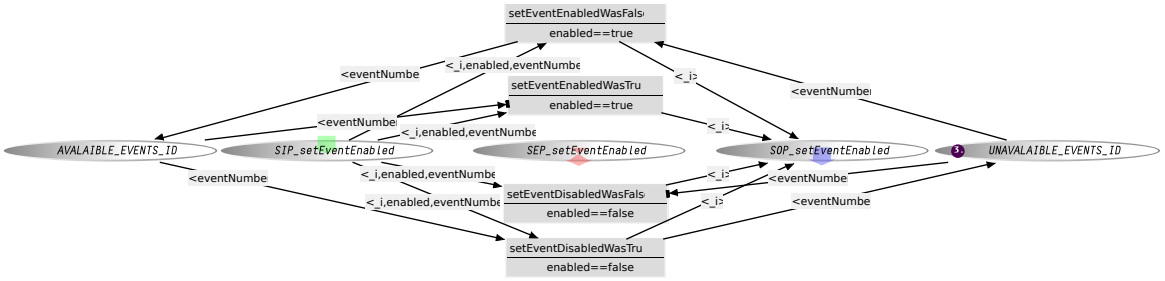


FIGURE 6.9 – Modèle partiel du bouton relatif à la gestion des événements disponibles du cœur de l'application des Quatre Saisons multi événements

à la présence d'un jeton possédant l'attribut *eventType=1*

Saison ressentie pouvant évoluer dans le sens positif est représentée par un oblong rouge lorsque le marquage de la place *AVAILABLE_EVENTS_ID* correspond à la présence d'un jeton possédant l'attribut *eventType=2*

Saison ressentie pouvant évoluer dans le sens négatif est représentée par un oblong bleu lorsque le marquage de la place *AVAILABLE_EVENTS_ID* correspond à la présence d'un jeton possédant l'attribut *eventType=3*

La figure 6.10 décrit la gestion de l'armement du bouton. Lorsqu'un curseur passe au-dessus du bouton, il s'arme et un rendu s'effectue pour signaler à l'utilisateur la possibilité d'utiliser le ou les services disponibles.

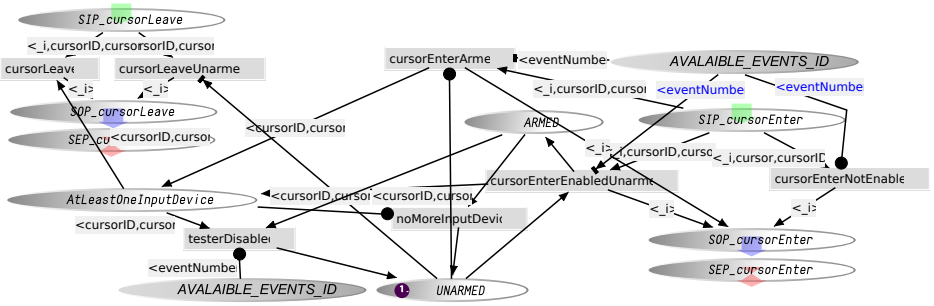


FIGURE 6.10 – Modèle partiel du bouton relatif à la gestion de l'armement du bouton (présence d'un ou plusieurs curseurs au-dessus du bouton)

Le rendu relatif à l'armement correspond à une coloration de la bordure du bouton. Il est déclenché lorsque la place *ARMED* contient un jeton.

Le dernier modèle partiel, présenté en figure 6.11, décrit le comportement du bouton lorsqu'on engage le curseur placé au-dessus. Ce modèle présente des caractéristiques similaires à celui du curseur puisqu'il reproduit certaines de ces transitions afin d'effectuer

l'interaction. On trouve les transitions par événement relatives à l'engagement primaire et secondaire des deux curseurs. Trois places servent à décrire l'état courant :

IDLE est la place qui nous sert à identifier l'état du bouton au repos

PRIMARY_PRESSED est relatif à l'engagement du bouton via le premier service du curseur

SECONDARY_PRESSED est relatif à l'engament du bouton via le deuxième service du curseur

La place *LABEL* sert à stocker le nom à afficher sur le label. On trouve une fois encore la place virtuelle *AVAILABLE_Event_ID* qui permet de conditionner l'activation des transitions par événement.

La présence de jeton dans *PRIMARY_PRESSED* ou *PRIMARY_PRESSED* déclenche un rendu qui "enfonce" le bouton selon un principe de skeuomorphisme pour les ombres. Le jeton contenu dans la place *LABEL* déclenche le rendu du nom à afficher sur le bouton.

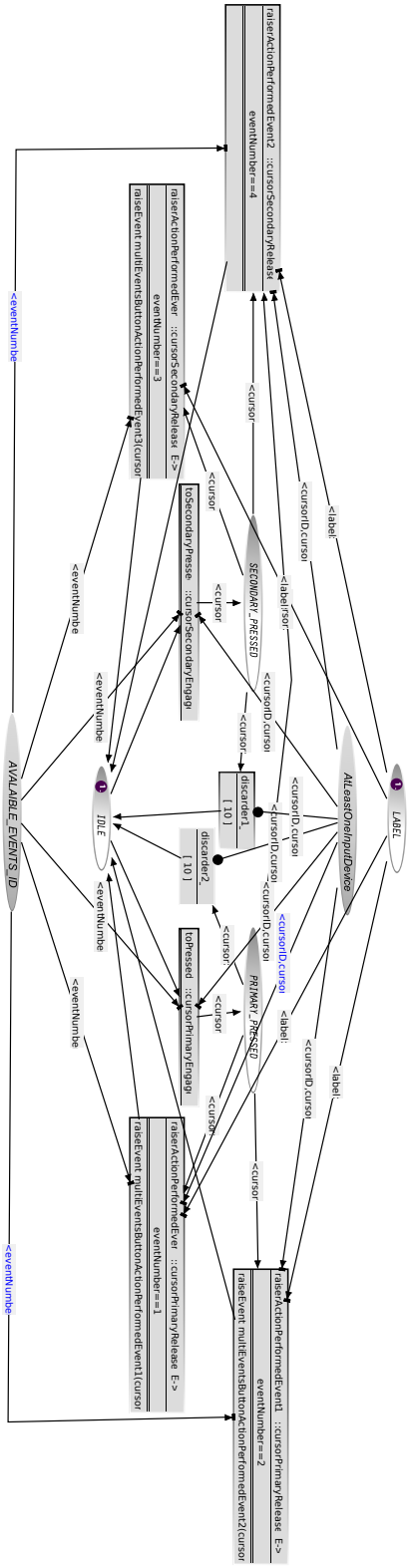


FIGURE 6.11 – Modèle partiel du bouton relatif à la gestion de l'engagement des boutons (enfoncé ou non)

6.4.3.2 L'adaptateur de dialogue

L'adaptateur de dialogue permet de décorréler les zones sensibles du coeur de l'application. La figure 6.12 décrit les transitions qui permettent de transformer en événements génériques le couple événement /source ayant une signification sémantique liée à la technique d'interaction. Avec de tels modèles, on peut imaginer déclencher les mêmes

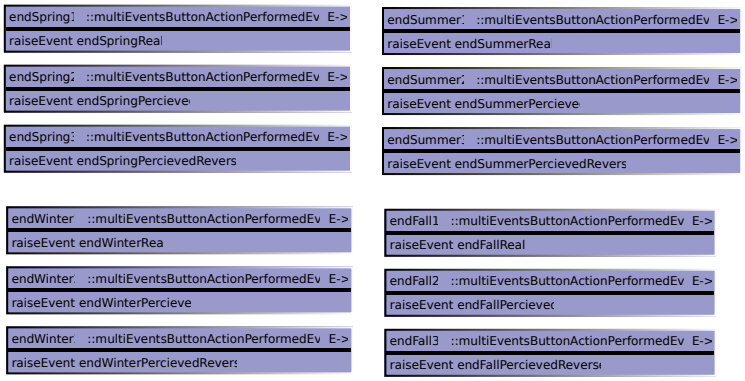


FIGURE 6.12 – Modèle de l'adaptateur de dialogue

événements de sortie avec des interactions tout à fait différentes en entrée. Cela permet de diversifier les types d'entrées sans impacter le modèle comportemental.

Ce modèle ne possède pas d'état à proprement parler, il n'existe pas de rendu lui étant associé.

6.4.3.3 Le cœur de l'application

Le coeur de l'application, ou dialogue est présenté en figure 6.13. Le centre de ce modèle, en bleu, correspond à l'évolution des vraies saisons, l'extérieur, en vert, correspond à l'évolution des saisons ressenties. La saison ressentie peut évoluer de trois manières, correspondant à trois transitions présentes entre chaque place du calque vert :

- L'interaction avec clic gauche de la souris déclenche l'évolution dans le sens naturel (*endSeasonPercieved*)
- L'interaction avec clic droit de la souris déclenche l'évolution dans le sens inverse (*endSeasonPercievedReverse*). Deux cas se présentent :
  1. La saison ressentie est en avance : on conditionne le franchissement avec la vraie saison courante en écart d'une unité
  2. Les saisons sont identiques : on conditionne le franchissement avec la vraie saison courant identique

  Ces deux conditionnements permettent de traiter les cas d'évolutions négatives des saisons sans avoir plus d'une saison de retard.

- Pour ne pas avoir plus d'une saison de retard avec la saison naturelle lors de son évolution, la saison ressentie évolue automatiquement dans le même sens (via la transition *catchUpSeason*). Cette contrainte du retard max d'une saison est modélisée en rouge.

Le rendu est effectué sur une grande majorité des places. Les quatre places intérieures déclenchent le rendu sur le label correspondant aux *vraies saisons* ainsi que la synthèse vocale.

Les places extérieures déclenchent le rendu relatif au label des *saisons ressenties*.

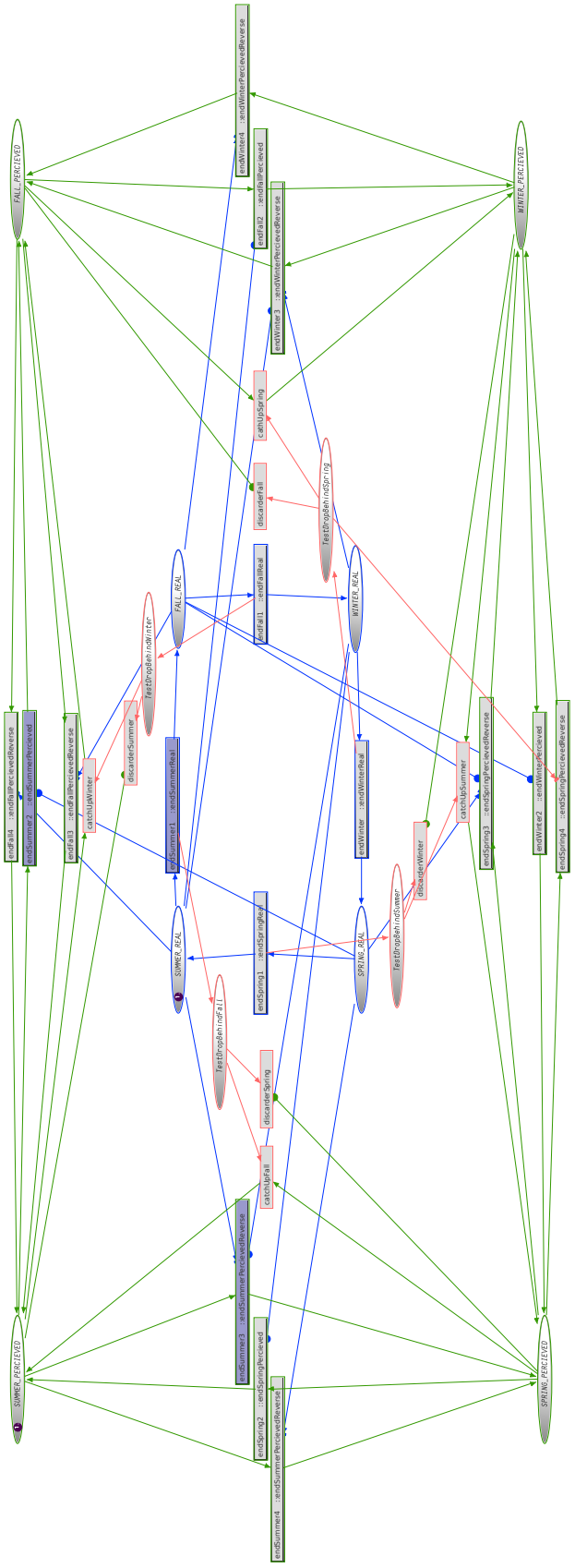


FIGURE 6.13 – Modèle complet du cœur de l'application des Quatre Saisons multi événements

6.4.3.4 Les fonctions d'activation

Les fonctions d'activation (présentées sur la figure 6.14) permettent de gérer l'activation et la désactivation des zones sensibles conditionnées par la disponibilité des services qu'elles déclenchent. Ces fonctions sont à la fois liées au dialogue et aux zones sensibles.

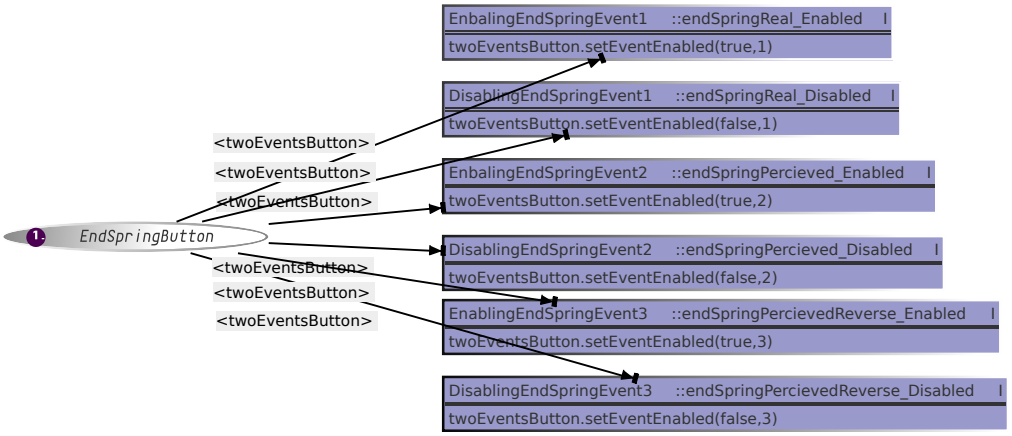


FIGURE 6.14 – Modèle de la fonction d'activation lié au bouton *EndSpring*

Le rendu est délégué aux zones sensibles (les boutons) via une fonction d'activation. Par exemple sur la figure 6.14, l'événement *endSpringReal_ Enable* est reçu lorsqu'au moins une transition déclenchée par l'événement *endSpringReal* sur le modèle de l'application est franchissable. La transition appelle la fonction de la zone sensible *setEnable*, avec les paramètres *(true,1)* le 1 correspondant au type d'événement faisant évoluer la vraie saison.

Le rendu d'activation est alors effectué en observant le changement d'état dans la zone sensible.

6.4.4 Système de rendu et gestionnaire de chaîne de sortie

Grâce au mécanisme de méta-événements présenté en section 2.3.2.5, nous pouvons déclencher des actions relatives à l'état du système interactif. On utilise, en plus du mouvement des jetons dans les places, la franchissabilité des transitions pour gérer le comportement des modèles.

Le principe repose sur deux objets qui formeront le rendu de présentation :

Les fonctions de concrétisation de rendu sur la plateforme effectuent la correspondance entre les changements d'état des différents modèles et les fonctions de rendu de présentation de la toolkit visible en figure 6.15. Par exemple, la première ligne indique que lorsque qu'un jeton est ajouté dans la place *LABEL*, on appellera la fonction de rendu de présentation *setLabel*. Ces fonctions ont pour argument le

marquage des places ; on a accès à une chaîne de caractères présente dans le jeton de la place *LABEL* donnant le nom à afficher sur le bouton.

Les fonctions de rendu de présentation gèrent le rendu effectif (graphique et vocal pour notre exemple). Elles font partie intégrante des librairies de la toolkit, et donc ne sont pas présentes de manière autonome dans l'architecture.

```
public ICMI2015MultiEventsButtonRenderer(CapabilityManager theObCS, IWidget
    presentationPart) {
 super(theObCS, presentationPart);
 addTokenAddedRenderingAdapter("LABEL", "setLabel");
 addMarkingResetRenderingAdapter("LABEL", "initLabel");
 addTokenAddedRenderingAdapter("AVALAIBLE_EVENTS_ID", "setEventEnabled"
     );
 addTokenRemovedRenderingAdapter("AVALAIBLE_EVENTS_ID", "
     setEventDisabled");
 addTokenAddedRenderingAdapter("ARMED", "setArmed");
 addTokenRemovedRenderingAdapter("ARMED", "setUnArmed");
 addTokenAddedRenderingAdapter("IDLE", "setIdle");
 addTokenAddedRenderingAdapter("PRIMARY_PRESSED", "setPressed");
 addTokenAddedRenderingAdapter("IDLE", "setIdle");
 addTokenAddedRenderingAdapter("SECONDARY_PRESSED", "setPressed"); }
```

FIGURE 6.15 – Fonction de concrétisation de rendu du bouton

6.4.5 Chaîne de sortie liée à l'écran

Nous n'avons pas modélisé la gestion de la modalité visuelle dans cette application, nous utilisons les fonctions du système, appelées par la toolkit javaFX. L'intégralité des rendus visuels effectués passe donc par les librairies javaFX qui utilisent à leur tour les fonctions système. C'est la raison pour laquelle ce sous-système est présenté en bleu clair sur la figure 6.3.

6.4.6 Chaîne de sortie liée à la synthèse vocale

La synthèse vocale des systèmes d'exploitation n'étant pas facilement intégrable directement en Java, nous avons fait le choix d'en intégrer une autre. L'environnement de développement Petshop est basé sur des modules Maven. Nous avons donc intégré un moteur de synthèse vocale libre appelé maryTTS [2]. L'utilisation d'un module Maven nécessite peu de travail d'implémentation. En effet, il nous suffit de développer le gestionnaire de

2. Mary Text To Speech est disponible sur https://github.com/marytts/marytts(accès 06/2017)

synthèse vocale pour pouvoir profiter pleinement des fonctions de synthèse vocale dans Petshop.

Le gestionnaire se charge de faire l'instanciation du moteur de synthèse vocale et de fournir les liens pour la gestion des abonnements aux événements déclenchant le rendu de présentation vocal. Il suffit alors de fournir une chaîne de caractères au moteur pour qu'il rende une synthèse vocale du texte entré.

6.5 Analyse de l'interaction et des comportements autonomes

Cette application étant purement illustrative et n'ayant pas d'utilisateur mis à part ses concepteurs, nous ne pouvons pas appliquer l'intégralité du processus que nous avons présenté au chapitre 5. Les techniques d'interaction sont déjà décrites de manières formelles. Nous pouvons appliquer le processus à partir de la quatrième étape : *l'identification des potentielles surprises*, présentée en section 5.3.4.

En étudiant l'application, nous avons remarqué que le curseur n'avait pas de rendu associé à son changement d'état [*Engaged/Idle*], cela pouvait porter à confusion dans un contexte où deux utilisateurs peuvent interagir sur la même interface. En effet, si un opérateur est en train de cliquer sur un bouton, cela empêche l'autre utilisateur d'interagir sur ce même bouton. Afin d'éviter que le deuxième opérateur pense que l'interface est frisée, nous avons rajouté un rendu relatif à ces états. Ces rendus sont présentés en figure 6.16.

- Un cercle jaune est affiché à gauche du curseur jaune lorsque le bouton de la souris 1 est enfoncé
- Un cercle bleu est affiché à gauche du curseur bleu&rouge lorsque le bouton 1 de la souris 2 est enfoncé
- Un cercle rouge est affiché à droite du curseur bleu&rouge lorsque le bouton 2 de la souris 2 est enfoncé

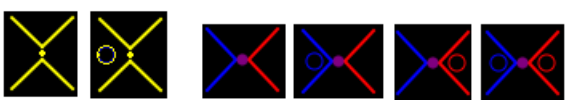


FIGURE 6.16 – Modification du rendu des curseurs avec l'ajout de rendus sur les places *ENGAGED* et *IDLE*

Avec ces ajouts de rendu, cela permet aux deux utilisateurs de savoir quelle est l'action effectuée par l'autre utilisateur sur sa propre souris.

6.6 Conclusion

Ce chapitre nous a permis d'illustrer l'instanciation du modèle d'architecture MIOD-MIT sur un exemple relativement simple, mais représentatif de la problématique de la multimodalité en entrée et en sortie.

Au-delà de la généricité des composants apparaissant dans l'architecture, les modèles ICO des composants décrits peuvent être réutilisés dans d'autres applications en suivant la méthode présentée au chapitre 3.

Ceci a aussi démontré que la notation ICO possède le pouvoir d'expression nécessaire à la description d'interactions multimodales complexes. Pour faciliter la lecture, nous n'avons pas montré comment les modèles peuvent être exploités dans le but d'analyser leur comportement pour en déduire des propriétés attendues (p. ex. l'application est ré-initialisable, les modèles ne consomment pas de ressources..).

Chapitre

7 Une étude de cas de taille réelle : Le radar météo d'un avion de ligne

Sommaire

7.1 Introduction

Afin de prouver la validité de l'approche et sa résistance à mise à l'échelle sur une étude de cas de taille réelle, nous présentons une application ayant un cœur applicatif réaliste destinée aux cockpits d'avion, et qui possède un ensemble de modalités exploratoires. Cette application est une extension des travaux réalisés dans [Hamon-Keromen, 2014], qui ajoute de nouvelles modalités. L'explication du comportement (coeur de l'application) est extraite de son manuscrit. Cette étude de cas n'a pas pour objectif de justifier de l'utilisabilité ni du design d'une telle application. L'ambition de notre travail ne consiste pas en la refonte du design d'application, mais propose la mise en avant de moyens permettant de spécifier les comportements de tels designs. Le cas particulier des animations n'est pas non plus abordé dans cette étude de cas. En effet, [Mirlacher et al., 2012] a démontré la capacité à décrire les animations sur une interface. Le design de l'application proposée n'est donc pas représentatif d'une application multimodale critique pour un cockpit. Néanmoins, celui que nous proposons dans ce chapitre prend en compte les problèmes liés à la spécification des comportements complexes des widgets multimodaux.

7.2 Présentation de l'application avant extension

Nous présentons dans cette section un extrait de [Hamon-Keromen, 2014], détaillant l'application destinée à illustrer la méthodologie de définition d'interaction tactile pour les systèmes critiques.

7.2.1 Présentation des systèmes avions existants

L'étude de cas se base sur des systèmes avions existants. Les sous-sections suivantes les décrivent tels qu'ils existent aujourd'hui dans un cockpit d'Airbus A350.

7.2.1.1 Le Navigation Display

Le ND (*Navigation Display*) est l'application cockpit qui permet de visualiser différentes informations liées au vol telles que sa trajectoire, les aéroports environnants ainsi que les avions en vol. De plus, cette application permet d'afficher les informations provenant du radar météorologique de l'avion. La Figure 7.1 illustre l'interface de cette application ainsi que son possible positionnement dans le cockpit (dans le cockpit de l'A350, il est possible de reconfigurer les applications pour les positionner sur des écrans différents). Pour cette étude de cas, nous nous focalisons sur la partie latérale du ND. En effet, le VD *(Vertical Display)* qui affiche le profil de vol vertical n'est pas pris en compte, car nous n'avons pas besoin de cette partie pour atteindre les objectifs fixés.

Enfin, dans le but d'animer l'interface, nous avons relié le *Navigation Display* à un modèle de radar qui simule la présence d'avions à proximité. Les informations envoyées

FIGURE 7.1 – Écrans d'un cockpit d'A350, extrait de [Hamon-Keromen, 2014]

par ce modèle de radar ne sont pas représentatives de données envoyées au cockpit, mais leur but est uniquement de montrer une situation qui évolue au cours du vol.

7.2.1.2 Le FCU et le FMS

Le FCU (pour *Flight Control Unit*) est un panneau de commande composé de plusieurs contrôles physiques. Il est situé à l'avant du cockpit, au-dessus des écrans et est divisé en deux types de panneaux :

- Deux panneaux de commande de type EFIS (*Electronic Flight Information system*) qui permettent de configurer les écrans de pilotage et de navigation (PFD et ND) et de régler l'altitude barométrique. On distingue, un panneau pour le capitaine (CAPT EFIS control Panel) et un pour le copilote (F/O EFIS Control Panel).
- Un panneau de commande de type AFS (*Auto Flight System*) qui permet le paramétrage du pilote automatique.

Pour les besoins de notre étude, nous reprenons les fonctions suivantes du FCU :

- Le mode d'affichage du ND ainsi que son *range* : Nous avons choisi de représenter deux visualisations de la carte : un mode ARC et un mode ROSE. Ces deux modes définissent une vue particulière de la carte sur le *Navigation Display*. Dans les deux cas, le *range* est la distance maximale affichée.
- Le choix du cap de l'avion : La sélection de ce cap se fera directement sur la carte, mais sa validation sera réalisée sur un *pushButton* particulier sur l'IHM.

7.2.1.3 Le radar météorologique

Le radar météorologique, installé dans la pointe avant des avions de ligne, permet de détecter la présence de gouttes d'eau. Grâce aux informations recueillies, les systèmes de

l'avion sont capables de générer une image de la météo devant l'avion, suivant le faisceau de l'antenne radar. Ils ne possèdent qu'une seule antenne. Dans cette étude de cas, nous avons fourni un moyen de contrôler directement l'orientation de cette antenne. Bien que cette fonctionnalité ne soit pas représentative d'une utilisation normale de ce type de radar, nous l'avons introduit afin d'illustrer les problèmes liés à la commande simultanée par plusieurs utilisateurs d'une ressource unique, dans notre cas, l'antenne elle-même.

7.2.2 Fonctionnement de l'application pour l'étude de cas

Nous détaillons l'ensemble de fonctions que propose notre étude de cas ainsi que les techniques d'interaction qui les supportent. L'IHM finale est présentée Figure 7.2. L'application est composée de trois parties détaillées dans les sous-sections suivantes :

- visualisation de la carte (en bas)
- contrôle du radar météorologique (en haut à droite)
- gestion de la trajectoire de l'avion (en haut à gauche)

Enfin, la simulation permet de faire fonctionner simultanément deux écrans. Ils sont décrits avec des modèles identiques et correspondent à l'écran du commandant de bord ainsi que celui du copilote.

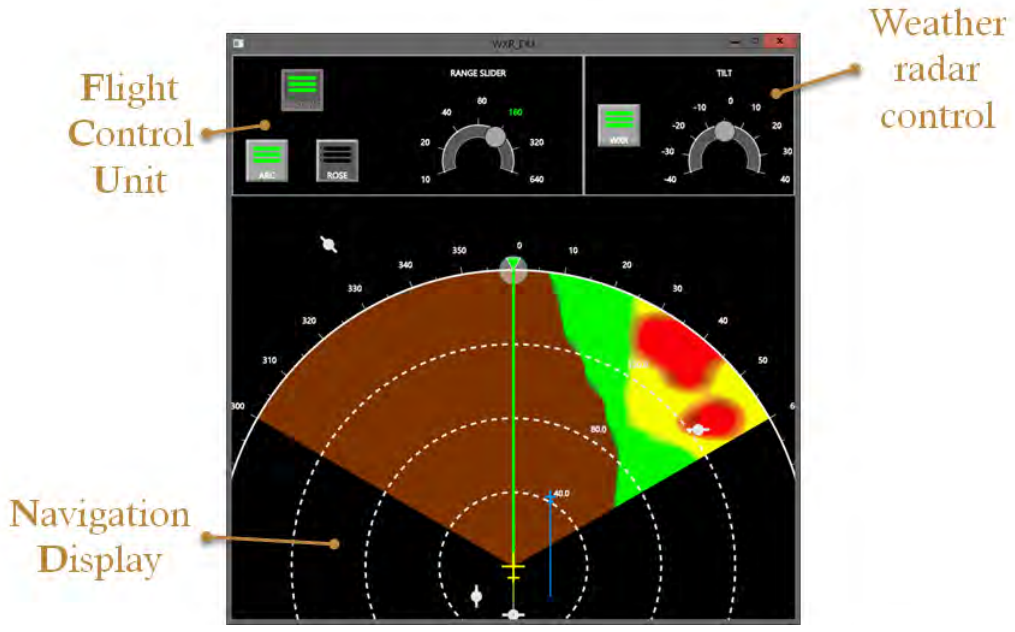


FIGURE 7.2 – IHM de l'étude de cas du ND tactile, extrait de [Hamon-Keromen, 2014]

7.2.2.1 Partie Flight Control Unit

La partie en haut à gauche de l'interface (Figure 7.2), ou FCU tactile, est composée de trois *pushButtons* et un *rangeSlider* customisé. Ils permettent de réaliser les fonctions suivantes :

- La validation du cap présélectionné. Cette commande est réalisée par un appui sur le *pushButton* « HEADING » (celui du haut).
- Le changement de mode d'affichage de la carte. Cette commande permet de passer de la visualisation type ARC (Figure 7.3-A) à la visualisation type ROSE (Figure 7.3 -B) et inversement. Les deux *pushButtons* « ARC » et « ROSE » sont les interacteurs pour ce changement de mode. Le mode ARC permet d'afficher principalement l'espace devant l'avion tandis que le mode ROSE propose une vue centrée sur la position de l'avion.
- Le changement du niveau de zoom de la carte. Le range, ou distance maximale affichée sur la carte est modifiable grâce au *slider* « RANGE SLIDER ». Il permet de sélectionner un niveau de zoom correspondant à l'un des ranges disponibles (10, 20, 40, 80, 160, 320 et 640). Ces valeurs correspondent à celles des *ranges* dans les cockpits Airbus.

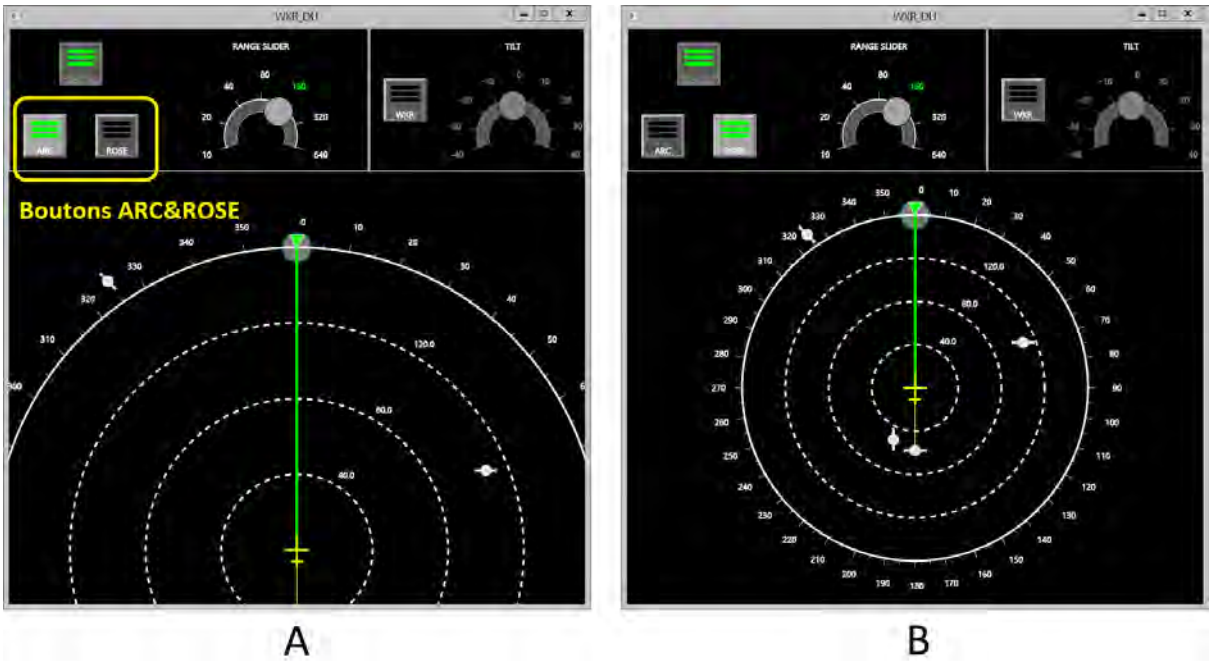


FIGURE 7.3 – Modes d'affichages ARC (A) et ROSE (B), extrait de [Hamon-Keromen, 2014]

Les changements du *range* et du mode d'affichage n'impactent que l'écran sur lequel se fait l'interaction, tout comme dans un cockpit actuel. Chaque pilote peut ainsi régler l'affichage de la carte selon les besoins de sa mission.

7.2.2.2 Partie Weather Radar Control

La partie en haut à droite de l'interface présentée dans la Figure 7.2 permet de contrôler certains paramètres du radar météorologique. Le *pushButton* « WXR » active ou désactive le radar et affiche le cas échéant les informations qu'il envoie. Enfin, le *slider* « TILT » permet de modifier l'orientation de l'antenne du radar. Sur la capture d'écran de la Figure 7.2, ce *slider* est *enabled* (actif), car le radar est allumé. Il n'est en effet possible de modifier l'orientation de l'antenne que si le radar est allumé. Inversement, les captures d'écran de la Figure 7.3 montrent le *slider* désactivé.

7.2.2.3 Partie Navigation Display

La partie centrale basse de l'interface présentée dans la Figure 7.2 affiche la carte de navigation de l'avion. Elle permet de visualiser les aéroports ainsi que les avions à proximité . Pour complémenter la fonction de changement de range via le FCU tactile, la carte est interactive : une interaction de zoom à deux doigts est modélisée. Cette carte permet aussi l'affichage des informations météorologiques quand le radar est actif. De plus, une interaction de type « Flick » à deux doigts permet d'activer (de droite à gauche) ou de désactiver (de gauche à droite) ce radar. Ensuite une interaction à trois doigts sur cet affichage permet d'en modifier l'opacité. Cette interaction est similaire au « Z-translate » [Kin et al., 2011]. Enfin, le cercle bleu transparent est un interacteur qui permet de présélectionner un cap pour la navigation. Sa manipulation se fait grâce à un glissé-déposé.

7.3 Présentation des modifications en vue d'ajouts de modalités

Pour assurer la validité de notre approche, nous avons ajouté un ensemble de modalités à cette application. L'objectif étant de compléter l'utilisation du tactile et d'illustrer l'utilisation de multiple périphériques sur la plateforme.

7.3.1 Configuration matérielle

L'ensemble des périphériques présents pour notre étude et le suivant :

- En entrée
  - Une dalle tactile
  - Deux souris
  - Un Leap Motion (interaction gestuelle dans l'espace)
- En sortie
  - Un écran
  - Des haut-parleurs (synthèse vocale)

7.3.2 Interactions disponibles

7.3.2.1 Interactions tactiles

L'écran tactile met à disposition trois interactions :

- Le *touch simple tap* pour engager les boutons
- Le *touch drag* pour manipuler la rose
- Le *pinch* à deux doigts pour changer l'orientation
- Une interaction à trois doigts pour changer l'opacité d'affichage des nuages

7.3.2.2 Interaction de pointage

Dans la configuration normale, les souris ne permettent pas de déclencher toutes les interactions. Nous avons fait ce choix afin de démontrer les possibilités de reconfiguration de la plateforme. Elles permettent néanmoins d'interagir avec tous les boutons.

7.3.2.3 Interaction gestuelles mid-air

Nous proposons une interaction de type mid air au moyen du Leap motion qui permettra aussi la modification de l'opacité. Cette interaction repose sur l'ouverture de la main. Plus la main est ouverte, plus l'opacité est faible.

7.3.2.4 Présentation en sortie

Côté sortie, le Weather Radar est bien entendu affiché sur un écran. Il peut déclencher une alarme en synthèse vocale lorsqu'il y a des rafales (alarme *Windshear*) risquant une perte de la portance sur un côté de l'avion.

7.3.2.5 Reconfiguration d'interaction

Les fonctionnalités de reconfiguration seront illustrées par la possibilité de transformer les entrées en provenance des souris en entrées tactiles. Cela nous permettra de maintenir les fonctionnalités accessibles uniquement par interaction tactile dans le cas de la perte des fonctions de la dalle tactile.

Le fonctionnement des périphériques d'entrée a été illustré sur une vidéo [1]

1. https://youtu.be/up6-h55yvZY. accès en juillet 2017

7.4 Instanciation de l'architecture

La figure 7.4 présente l'architecture complète de cette application. Nous ne présenterons pas les chaînes de composants liées aux souris, à la synthèse vocale et aux écrans, étant en tous points identiques à celles présentées au chapitre précédent (chapitre 6). Les composants de la chaîne de périphériques liée à la dalle tactile sont extraits des travaux de [Hamon-Keromen, 2014], et ont été légèrement adaptés pour abstraire certains aspects limitant la modifiabilité de l'application.

Les composants MIODMIT que nous présentons dans ce chapitre viennent compléter ceux de l'architecture du chapitre précédent (figure 6.3). Nous décrivons dans la figure 7.4 les chaînes de gestion liées au Leap Motion et au tactile, ainsi que le composant de gestion de configuration d'entrée. Les modèles de comportement de l'application du Weather Radar ne seront pas décrits, car ils n'apportent pas de précision supplémentaire quant à l'utilisation de MIODMIT. L'ensemble des modèles représente une cinquantaine de modèles fonctionnant en parallèle. Le système *Dialogue and Application Core* représente à lui seul une moitié de ces modèles.

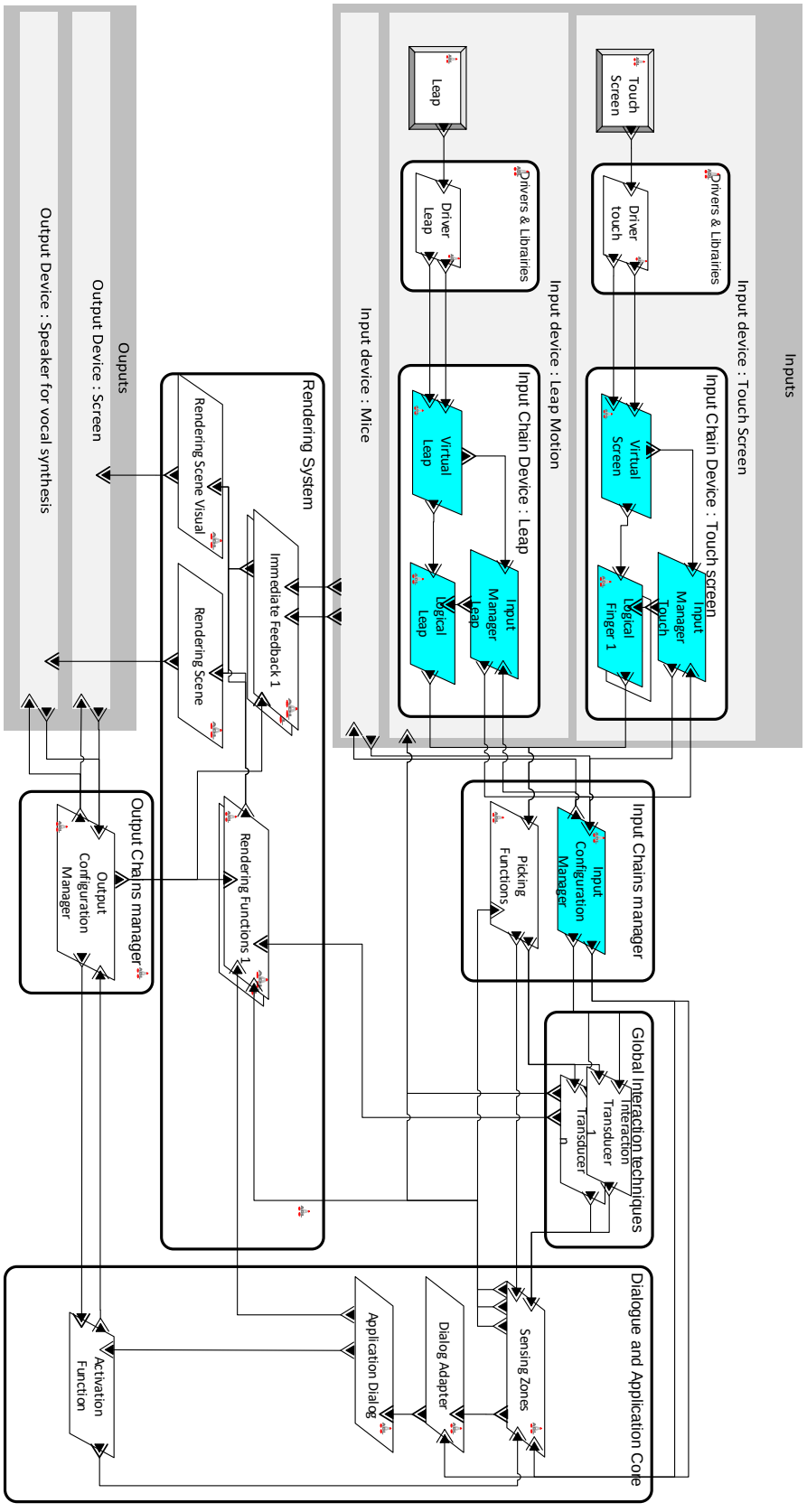


FIGURE 7.4 – Architecture du radar météo

7.4.1 Chaîne d'entrée liée au leap motion

La figure 7.5 décrit les composants nécessaires à l'inclusion de la chaîne de gestion du Leap Motion. La librairie fournie par le constructeur est, de part quantité de fonctions disponibles, plus complète que pour des périphériques plus simples tels que les souris voire même plus récent comme l'eye tracking. Nous aurions pu extraire simplement les événements en provenance de la librairie. Nous avons fait le choix de retranscrire une partie de la chaîne pour mettre à la disposition du reste de l'application tous les types d'événements disponibles dans la librairie. La librairie Leap Motion fournit un grand nombre de fonctionnalités via une API fournie [2]. La base de la reconnaissance passe par l'analyse de *Frame*. La librairie Leap permet d'avoir accès à un ensemble de données reconnaissance telle que la position des mains, les vecteurs associés à chaque doigt, la position du bras, mais aussi des informations plus haut niveau tel que des techniques d'interaction *CircleGesture*, *KeyTapGesture*, *ScreenTapGesture*, and *SwipeGesture*.

Nous n'utilisons que les données permettant d'extraire le pourcentage d'ouverture de la main qui sert de base à notre interaction.

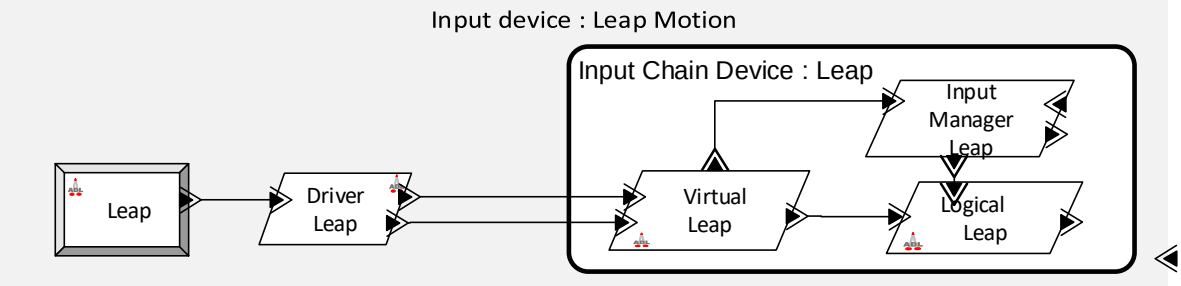


FIGURE 7.5 – Chaîne d'entrée liée au Leap Motion

Nous expliquions dans le chapitre 3 que les composants pouvaient être composites. Le Leap Motion est un exemple de la multiplication d'un type de composants en plusieurs modèles. En effet, le *logical device* associé est séparé en *Limb*, *Arm*, *Hand* et *Fingers*, tous récupérables via l'API. Nous n'utilisons que le Limb, qui permet d'extraire les données relatives à la main et son ouverture dans cette application.

2. https://developer.leapmotion.com/documentation/v2/java/devguide/Leap_Overview.html accès en juillet 2017

7.4.1.1 Frame Manager

Du fait du nombre de fonctions disponibles dans l'API, le modèle permettant d'extraire les données est conséquent. Nous le présentons en figure 7.6.

Le haut (en gris)du modèle permet de limiter la récupération des données aux *frames* contenant un membre

Le milieu du modèle(en rose) permet l'instanciation dynamique des membres. En effet, contrairement aux périphériques de pointage qui sont normalement tout le temps branchés sur le système, les membres ne seront pas toujours en vue de la caméra infrarouge du Leap Motion, de la même manière que les doigts ne sont pas toujours en contact avec la dalle tactile. Il faut donc prévoir, comme pour la reconnaissance des doigts pour les dalles tactiles, une instanciation dynamique des modèles correspondants.

Le bas du modèle à gauche (en bleu) ne sert qu'à l'initialisation.

Le bas du modèle à droite sert à la gestion du picking.

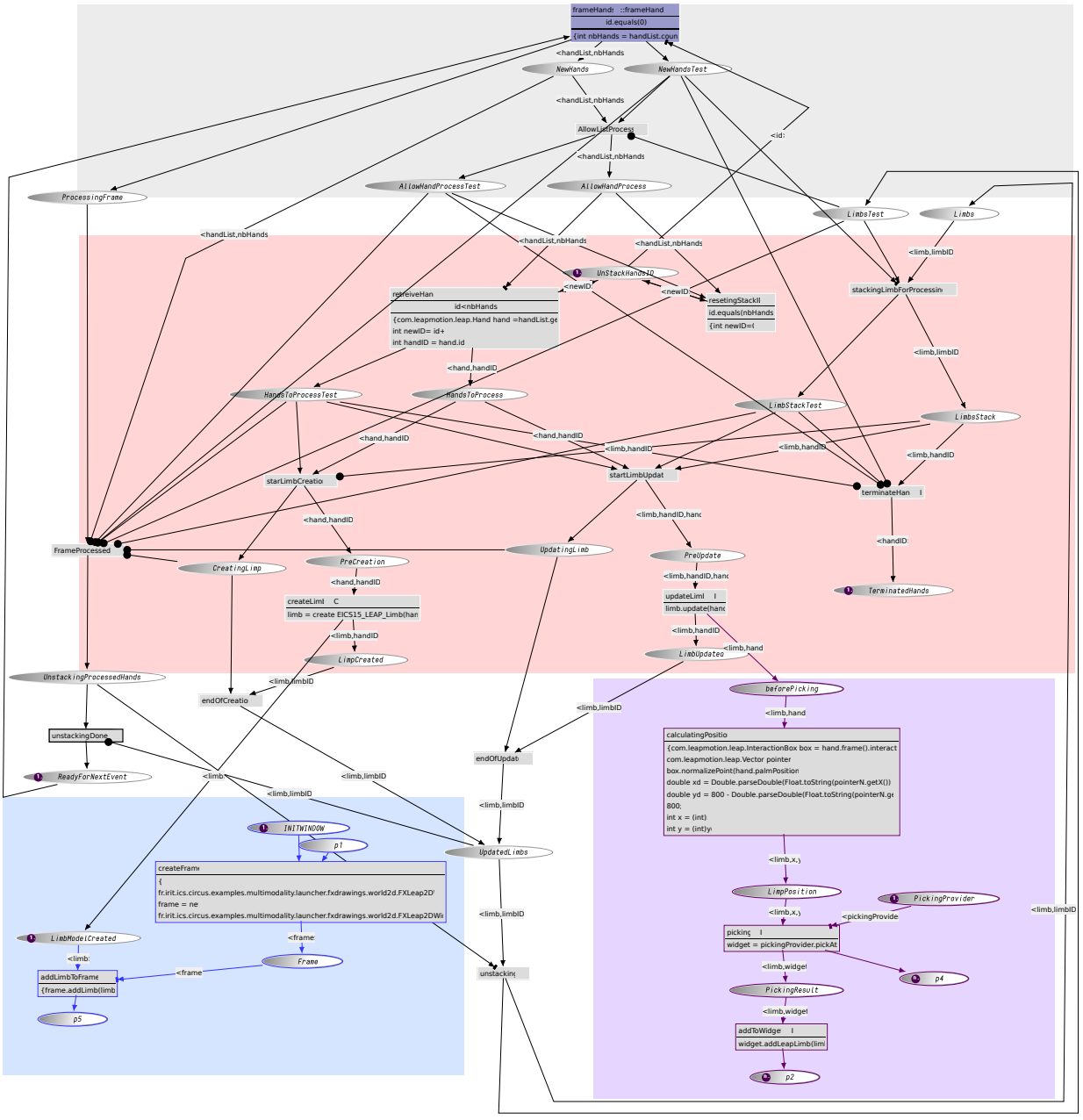


FIGURE 7.6 – Le modèle ICO du gestionnaire de Frame du Leap motion. En bleu les fonctions d'initialisation, en violet les appels aux foncions de picking

7.4.1.2 Membre extrait du Leap Motion

La figure 7.7 décrit le périphérique logique associé au membre. C'est celui qui contient toutes les informations nécessaires à la mise à jour des autres sous-éléments du membre reconnu dans la librairie. Après l'extraction de la Frame, ce modèle est appelé et effectue la fission des informations de la *Frame*.

Nous extrairons directement de ce modèle la quantité d'ouverture de la paume dans la transition *updateHand* qui nous servira à mettre à jour le modèle de la main.

Ce modèle de main ne contient qu'une place contenant les informations de position et d'ouverture de la main qui servent à la fois à changer l'opacité du radar, et dont la valeur servira pour le feedback immédiat du diamètre du "curseur" du périphérique logique.

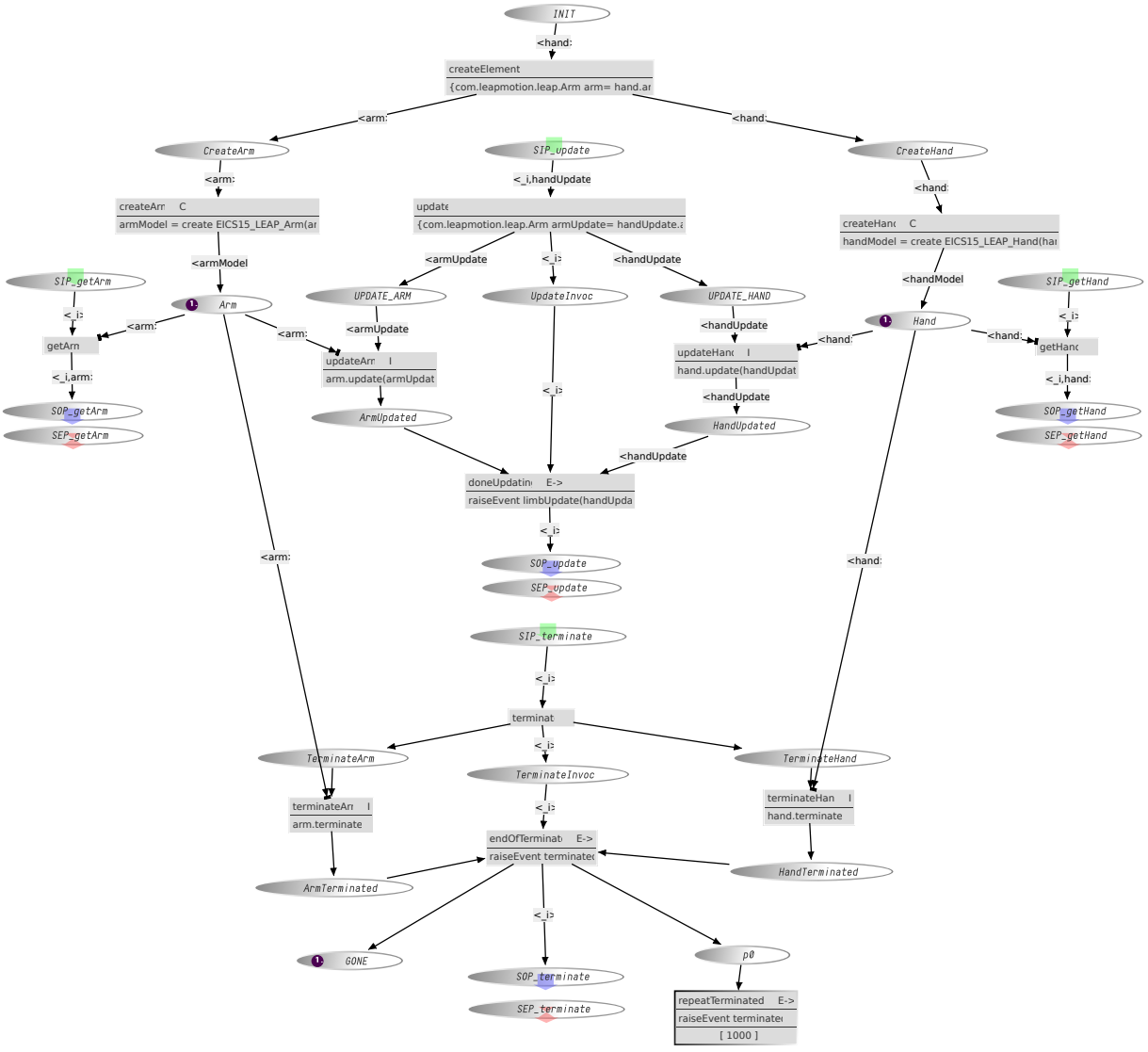


FIGURE 7.7 – Le modèle ICO du membre du Leap motion

7.4.2 Chaîne d'entrée liée au tactile

Nous présentons les composants nécessaires à l'inclusion de la chaîne de gestion de la dalle tactile (Figure 7.5).

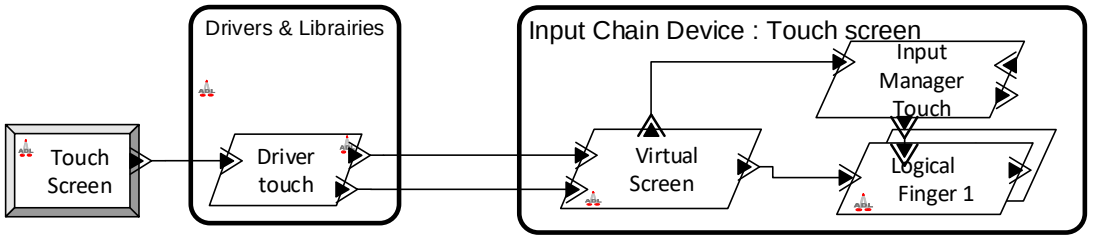


FIGURE 7.8 – Chaîne d'entrée liée au tactile

7.4.2.1 Périphérique virtuel de l'écran

La figure 7.9 présente le périphérique virtuel lié à l'écran. Il est le fournisseur des événements qui constitueront les doigts. Les interactions multi-touch sont très liées aux technologies qui permettent de détecter les points de contact. Suivant les technologies employées, il est possible de détecter une pression exercée sur la surface ou encore l'angle que réalise le doigt (le stylet,. . .) avec cette surface. Ces informations, qui complètent les informations de position, doivent être décrites, et donc supportées, par les notations utilisées. Actuellement, les périphériques tactiles se limitent à l'envoi de coordonnées sous la forme d'une paire (x,y) pour chaque point de contact. Néanmoins, des applications peuvent nécessiter de connaître la zone de contact dans son intégralité afin de proposer de nouvelles techniques d'interaction comme, par exemple, dans les travaux de [Bonnet et al., 2013]. Dans tous ces cas, l'architecture et les modèles qui la composent doivent permettre de s'adapter à ces scénarii.

Sur le modèle de la figure 7.9, les trois différentes branches utilisent les événements provenant du driver de l'écran pour les transformer en événements utilisables dans le reste de l'application. Ces trois branches mènent à la création de trois événements : *rawToucheventf_Down* (branche violette) pour le contact entre le doigt et la dalle tactile, *rawToucheventf_Up* (branche bleue) pour la fin de ce contact,*rawToucheventf_Update* (branche rouge) pour la mise à jour des coordonnées de contact.

Le haut du modèle correspond à la fonction ad hoc nous permettant d'activer ou désactiver les fonctions tactiles afin de simuler leur perte.

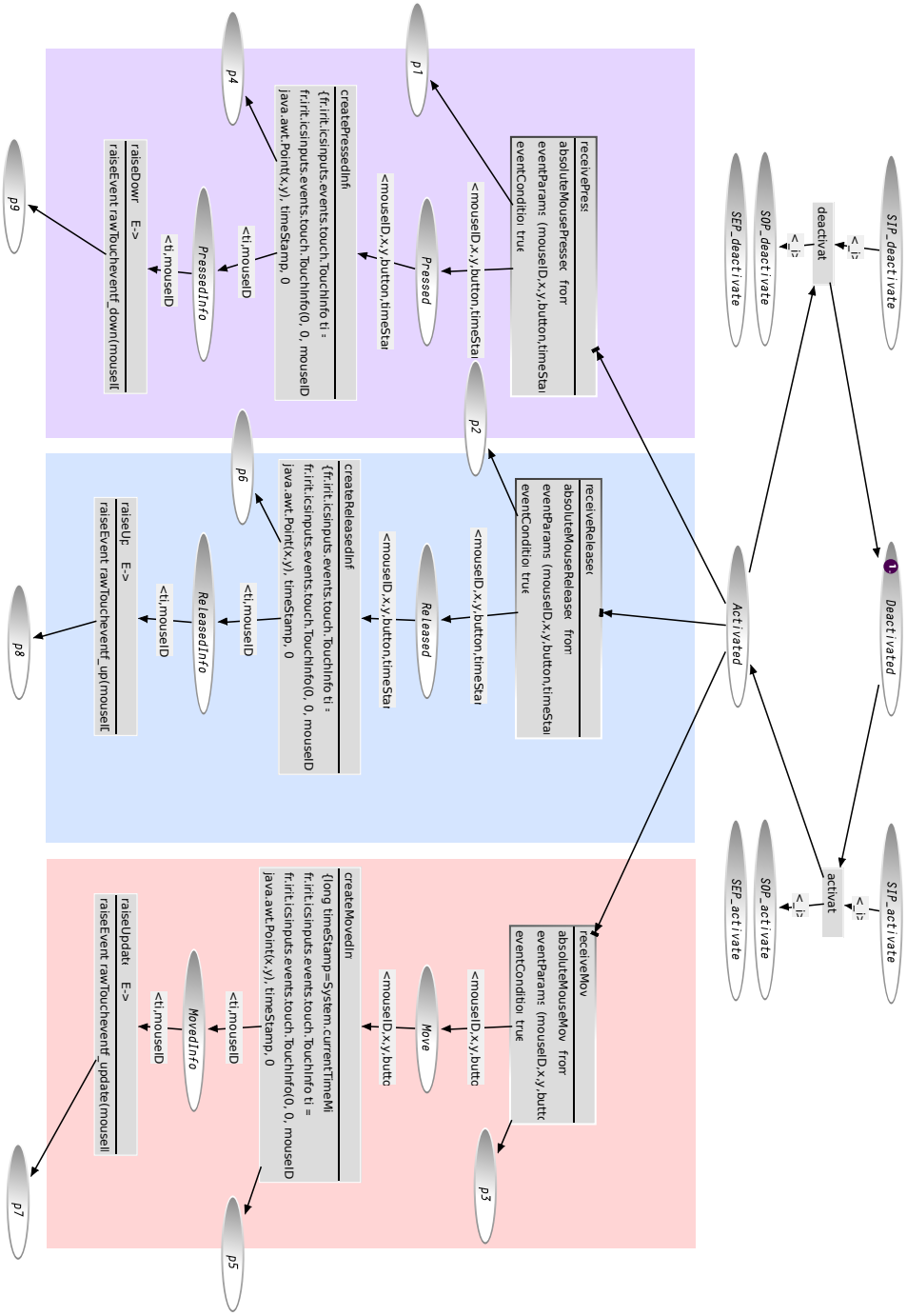


FIGURE 7.9 – Le modèle ICO de l'écran virtuel

7.4.2.2 Périphérique logique du doigt

La figure 7.10 décrit le périphérique logique associé au doigt.

La partie gauche du modèle (en rouge) contient les places et transitions relatives à la mise à jour lorsque le doigt bouge sur la surface tactile.

La partie haute (en violet) du modèle est relative à l'activation ou non du *feedback*.

La partie centrale est relative à la gestion de la fin du contact avec la surface tactile.

Enfin la partie à droite (en jaune) gère la mise à jour de l'accélération et de la direction du mouvement, toutes deux nécessaires pour la reconnaissance des techniques d'interaction gestuelles.

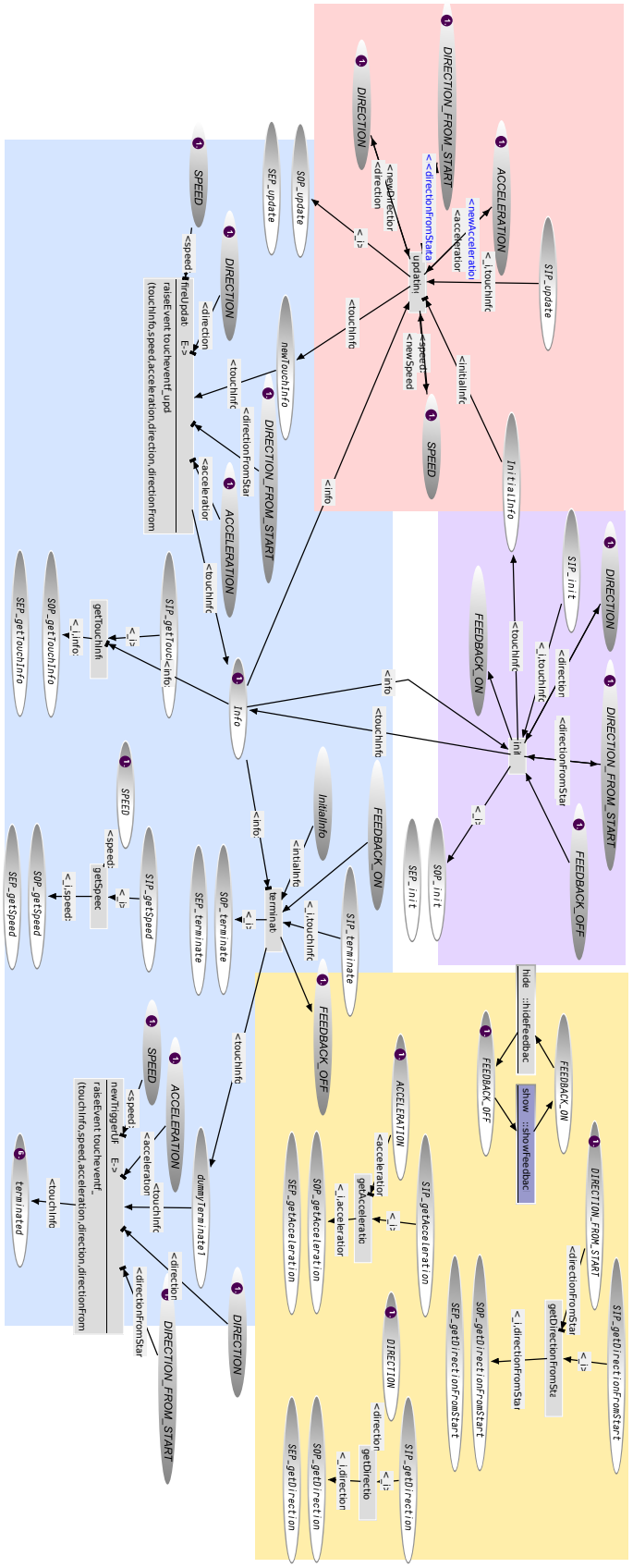


FIGURE 7.10 – Le modèle ICO du doigt

7.4.3 Gestionnaire de configuration d'entrée

Nous avons décrit deux configurations de périphérique d'entrée dans cette application. La première, pour le fonctionnement normal avec les souris et le tactile en parallèle, le second type de configuration dans le cas ou les fonctions tactiles seraient perdues. Pour cela, nous avons modélisé un *InputConfigurationManager*, permettant de gérer les deux configurations d'entrées liées à une fenêtre de contrôle *InputDeviceManager* permettant de modifier les paramètres de simulation. Cette fenêtre est présentée en figure 7.11.

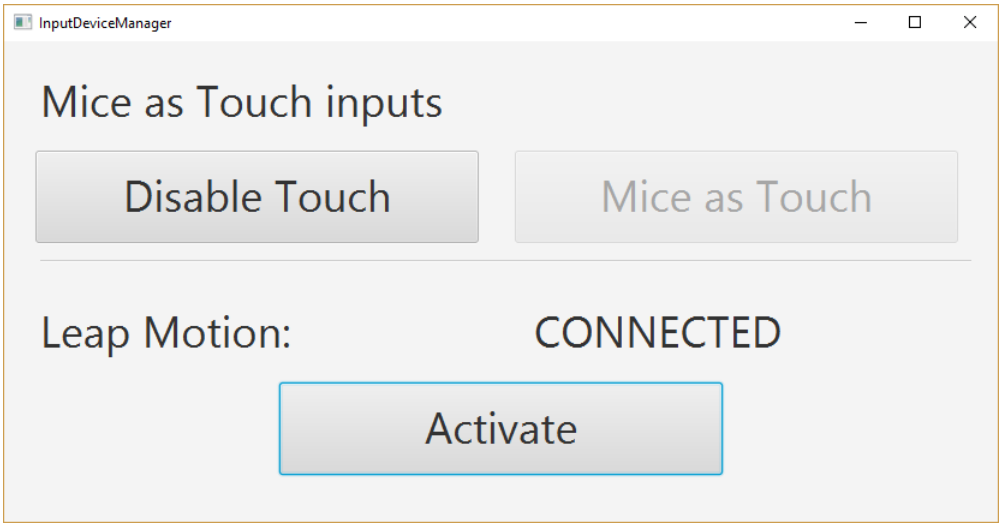


FIGURE 7.11 – IHM de l'*InputDeviceManager*

Le modèle de l'*InputConfigurationManager* est ad hoc et permet la transformation des événements du curseur en événement touch ainsi que la gestion dynamique du Leap Motion. Il est présenté en deux parties.

La figure 7.12 présente la partie relative à la gestion dynamique du Leap Motion.

Grâce à ce modèle, il est possible de brancher et débrancher le périphérique durant la simulation en prenant en compte les événements provenant du driver (cadre violet) Il est aussi possible d'activer ou non l'interaction mid air (cadre jaune lié au bouton de la fenêtre 7.11) pour éviter de changer l'opacité lorsque l'on effectue des mouvements n'ayant aucun rapport avec l'interaction (alternant d'interaction).

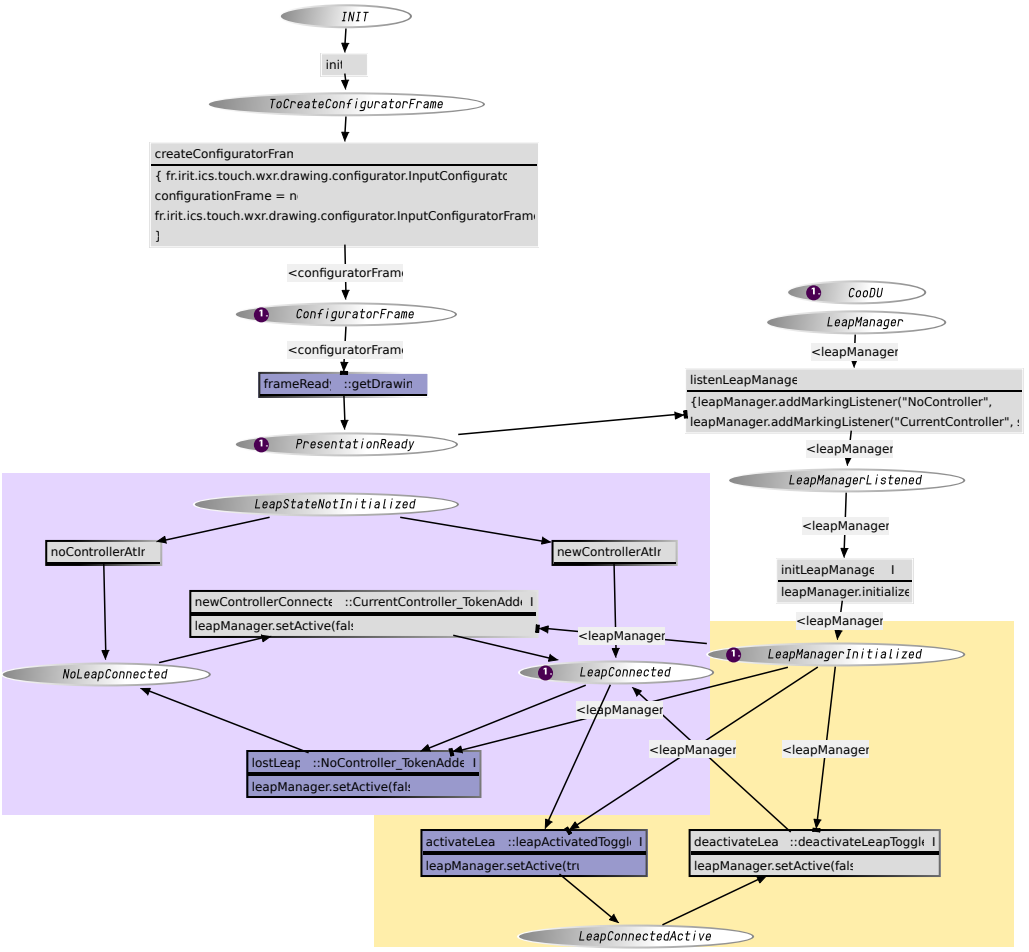


FIGURE 7.12 – Partie du modèle InputConfigurationManager liée à la gestion dynamique du Leap Motion

La figure 7.13 présente la gestion de la perte des fonctions tactiles. Cette partie de modèle décrit les séquences d'action et d'événements permettant de transformer les événements du curseur en événements tactiles. La partie haute du modèle décrit la réception des événements de l'interface de contrôle pour désactiver le tactile (par un appel de service). De la même manière, la partie basse du modèle sert à appeler le gestionnaire de souris, pour lui demander (par appel de service encore une fois) de transmettre les événements du curseur (par exemple CursorMove) à un modèle d'écran de secours *backupVirtualScreen* qui transforme les événements curseur en événements tactiles. De cette manière, la chaîne d'interaction liée au tactile est toujours accessible grâce aux événements souris.

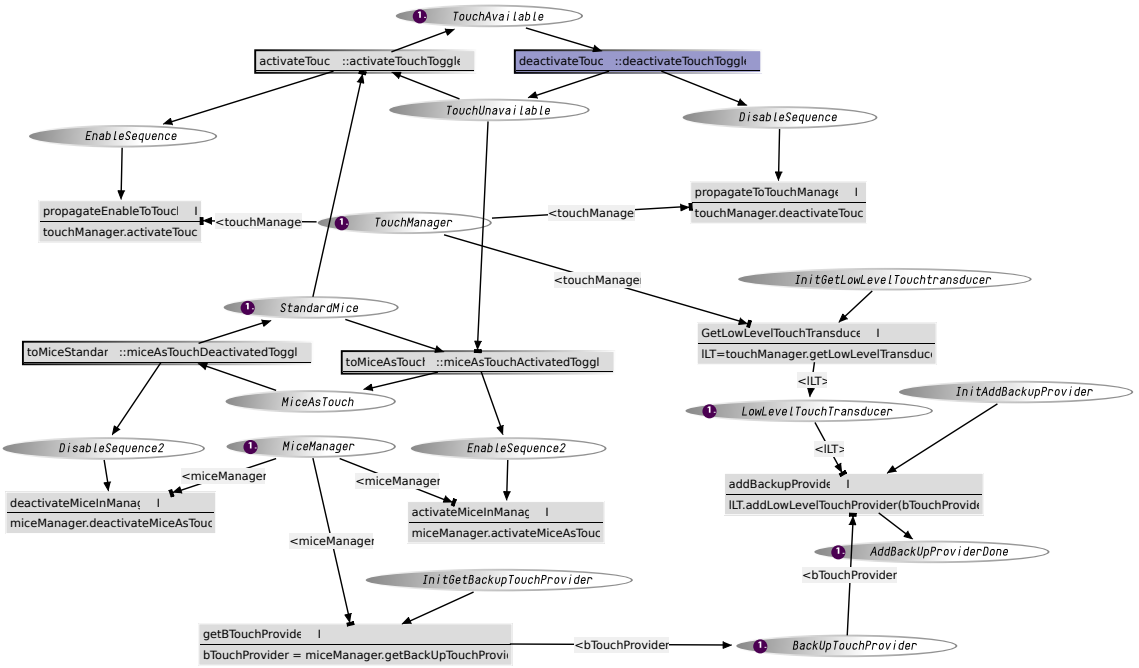


FIGURE 7.13 – Partie du modèle InputConfigurationManager liée à la gestion de configuration qui transforme des événements curseur en événements tactiles

7.5 Application et chaîne de sortie

Les modèles du cœur de l'application n'apportent pas de précision sur le comportement de MIODMIT. Leurs évolutions sont déclenchées par des événements génériques qui proviennent de l'adaptateur de dialogue. La multiplicité des modalités implique une adaptation des zones sensibles pour qu'elles puissent gérer les interactions en provenance de plusieurs types de périphériques, que ce soit pour le comportement ou pour le rendu. Dans le cas de notre application, les modèles de zones sensibles relatifs au tactile n'ont pas été affectés grâce à la transformation d'événements de l'*InputConfigurationManager*.

7.6 Conclusion

Cette étude de cas nous a permis d'illustrer comment, grâce au respect du modèle d'architecture MIODMIT, nous pouvions ajouter des modalités avec un impact minimum sur une application existante qui suit préalablement le modèle et les principes énoncés dans les chapitres 3 et 5.

La plateforme permet l'intégration de périphériques partiellement modélisés comme le Leap Motion afin de faciliter l'intégration de travaux externes pendant les phases de prototypage.

Le respect de MIODMIT couplé à la modélisation ICO dans Petshop permet de développer des configurations alternatives simples en transformant des événements dans le composant *Input Configuration Manager*.

Enfin, nous avons prouvé que MIODMIT peut être utilisée dans une application de taille réelle grâce à la définition des fonctionnalités de chaque composant.

Chapitre

8 Une plateforme opérationnelle pour les systèmes multimodaux critiques : Application aux cockpits d'hélicoptères du futur

Sommaire

8.1 Introduction

La thèse est centrée sur l'ingénierie des systèmes interactifs multimodaux, laissant de côté les aspects de conception et validation des techniques d'interaction et des applications interactives. Nous avons jusqu'ici considéré que la conception était réalisée en amont et était fournie en entrée (sans remise en cause) des processus présentés dans les chapitres 3 & 5. Toutefois, ce chapitre montre que les contributions présentées dans cette thèse peuvent servir au support des activités des équipes de conception d'IHM intégrant des acteurs pluridisciplinaires (ergonomes, spécialistes en facteurs humains, ingénieurs, etc.). Plus particulièrement nous allons montrer comment MIODMIT, Petshop (et ARISSIM) peuvent apporter leur contribution à différents endroits des phases de prototypage de systèmes interactifs critiques.

Ce chapitre présente une plateforme opérationnelle liée à un simulateur de vol qui permet l'exécution d'applications mettant en œuvre des prototypes de techniques d'interaction dans un cadre compatible avec les contraintes des environnements critiques.

Le contexte de développement de cette plateforme opérationnelle se situe dans le projet IKKY, présenté dans la section 1.2.2, et destiné à explorer les modalités des futurs cockpits d'hélicoptère. Les modalités définies par AIRBUS Helicopter se résumaient à l'introduction d'écrans tactiles, complétés par un rotacteur haptique à deux niveaux (décrit dans la section 8.3.3.1), ainsi que la synthèse vocale en sortie, en plus des traditionnels écrans. La vision très en amont de ce projet ne nécessitait pas l'ajout des contraintes liées aux systèmes critiques et nous n'avons donc pas inclus ARISSIM dans le prototype présenté dans la section 8.3.3. ARISSIM peut néanmoins être ajouté à cette plateforme selon les modalités présentées au chapitre 4.

8.2 Objectifs et besoins pour la plateforme opérationnelle

Les spécifications relatives à la plateforme opérationnelle dans le cadre de ce projet sont les suivantes :

- L'interface homme-machine testée et le comportement de l'application doivent suivre une approche modulaire pour pouvoir effectuer des modifications le plus facilement possible.
- Lors des tests utilisateurs, toutes les évolutions du système doivent être enregistrées, qu'elles soient dirigées par les utilisateurs ou déclenchées en interne par le système (évolutions autonomes). Cela permettra d'étudier ensuite les interactions de l'utilisateur avec le système, et éventuellement de prévoir un rejeu des interactions. Cette fonctionnalité consommant beaucoup de ressources, la plateforme doit permettre de répartir les applications sur différentes machines (au sens des systèmes répartis équivalents) ce qui permettra de répartir la charge entre le contrôleur central et les tablettes tout en maintenant une performance adéquate.
- Le nombre de tablettes et de simulateurs qui fonctionnent en parallèle n'est pas défini. La plateforme doit être modulaire et centraliser le cœur du système interactif

conçu au sein d'une machine dédiée.

- Le simulateur de vol doit être isolé sur une machine afin de minimiser les actions du responsable du test. L'impact sur une simulation complète lors de modifications du comportement des prototypes durant le test sera ainsi réduit

Les composants logiciels de la plateforme opérationnelle doivent respecter l'architecture MIODMIT, rendant cette plateforme facilement modifiable pour inclure ou supprimer des périphériques.

8.3 Présentation informelle de la plateforme opérationnelle

8.3.1 Architecture générique de la plateforme opérationnelle

Nous présentons en figure 8.1 l'architecture globale de la plateforme opérationnelle. Deux humains sont identifiés : le pilote (l'utilisateur testant le système) et le développeur (le responsable du test).

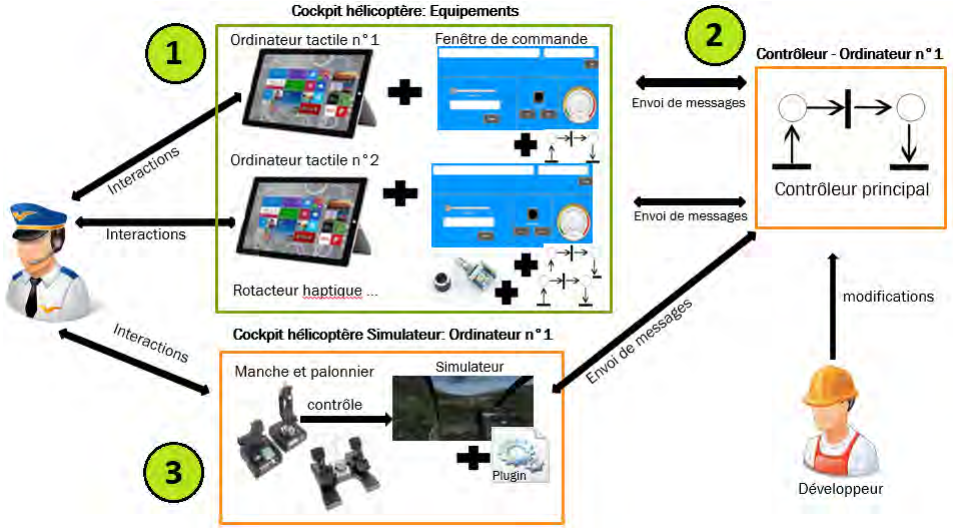


FIGURE 8.1 – Architecture physique globale du système

Le système est séparé en trois types de machines (numérotée sur la figure 8.1) :

Les machines dédiées aux interactions testées (1 en haut à gauche de la figure)telles que les tablettes tactiles contenant le programme responsable de l'affichage et du comportement des widgets composant l'interface graphique. On branchera sur ces machines tous les interacteurs nécessaires. Les périphériques suivants sont disponibles simplement en entrée/sortie sur cette plateforme, et ce, grâce à la réutilisation de composants de MIODMIT précédemment modélisés et décrit dans les chapitres 6 & 7 :

- Les dalles tactiles

- Deux souris
- La reconnaissance de geste du Leap Motion, via les techniques d'interaction reconnues dans le SDK
- La synthèse vocale

Le clavier pourra être utilisé au travers de la librairie JavaFX et n'a pas été modélisé. L'ensemble de périphériques de jeux (manche palonnier et manette de gaz) est branché directement sur le simulateur de vol et n'a pas été modélisé non plus, ces travaux ne visant pas à remplacer les commandes de vol.

La machine dédiée à la simulation du système : *Le contrôleur central* (2 en haut à droite de la figure) décrit par exemple le comportement lors de la réception d'un message depuis le simulateur de vol ou lors d'un changement d'état des interfaces testées sur les machines dédiées.

La machine dédiée au simulateur de vol (3 en bas à gauche de la figure), associée à un *plugin* permettant la réception et l'envoi de messages ainsi que la modification / récupération de paramètres du simulateur.

8.3.2 Le simulateur de vol X-Plane

Pour permettre le test des applications dans un contexte réel (les applications visées n'étant pas relatives aux commandes de vol direct), nous devons fournir la possibilité de les tester dans un environnement de simulation de vol. Les applications visées servent à la réalisation de tâches annexes au pilotage. L'utilisation d'un simulateur de vol permet donc d'immerger les applications testées dans un contexte réaliste.

8.3.2.1 Présentation du simulateur de vol choisi

X-Plane est un logiciel de simulation de vol développé par Laminar Research. Il se différencie des autres simulateurs par l'utilisation d'un modèle de vol basé sur la géométrie de l'aéronef utilisé. La simulation modélise en temps réel les forces qui s'exercent sur les différentes parties de l'avion afin de déterminer la portance et la trainée, pour ensuite calculer les accélérations, la vitesse et la position de l'appareil. Cette propriété fait que X-Plane peut être utilisé par des professionnels pour concevoir et tester des prototypes d'aéronefs.

D'autre part, X-Plane fournit un SDK en langage C permettant de créer et d'intégrer ses propres plugins dans le simulateur. En vue de cette étude, cette fonctionnalité nous a particulièrement intéressés, car elle va nous permettre d'intégrer des applications externes en lien avec le simulateur de vol. En effet, l'ensemble des données (dataref) de Xplane accessible est très important et couvre aussi bien les paramètres de vol (commandes outils de mesure, etc.) comme les paramètres de simulation de vol (météo, altitude position, etc.).

8.3.2.2 Services offerts par le plugin développé

En utilisant le SDK, nous avons développé un plugin se lançant automatiquement au démarrage d'Xplane qui nous permet d'accéder aux dataref, ceci afin de pouvoir greffer une application interactive au cockpit d'aéronef existant. Les services nécessaires devront couvrir le rafraichissement périodique d'une donnée dans le simulateur, la modification d'une donnée, et l'accès en lecture sporadique à une donnée.

Le plugin développé offre donc ces trois services :

La demande de modification d'une dataref :

Le contrôleur envoie un message du type «type@dataref@Value».

Le plugin renvoie un message du type «notify@type@dataref@value ».

La demande d'information sur une dataref :

Le contrôleur envoie un message du type «ask@dataref@-1».

Le plugin renvoie un message du type «response@type@dataref@value ».

La demande de "monitor" d'une dataref :

Le contrôleur envoie un message du type subscribe@type@dataref@value .

XPlane va mettre sur "écoute" la dataref et envoyer un message du type notify@type@dataref@value à chaque fois que la dataref change de valeur, selon la période spécifiée par *value* .

Grâce à ces trois services, on peut interagir sur les paramètres de vol, ou directement sur les paramètres relatifs à l'interaction. Cela nous permet de venir greffer d'autres applications interactives, externes au simulateur. Elles pourront avoir accès aux fonctionnalités du cockpit grâce au plugin. les dataref étant nombreuses et relatives tant aux données d'interaction du cockpit (par exemple la fréquence de la radio) qu'aux données de simulation (par exemple l'altitude de l'aéronef). Il sera donc possible de contrôler depuis une autre interface ces paramètres.

8.3.3 Prototype d'illustration des fonctionnalités pour le projet IKKY

Pour illustrer les possibilités de la plateforme opérationnelle et plus particulièrement celles du plugin, nous avons développé une application utilisant ses fonctionnalités principales.

8.3.3.1 Ajout des roues encodeuses à la plateforme opérationnelle

Les rotacteurs haptiques sont des roues encodeuses (des potentiomètres numériques sans butée) dont le crantage (le nombre de crans lorsqu'on tourne la roue) et la résistance (la résistance à la rotation) sont réglables. Un rotacteur haptique à deux niveaux est un dispositif présentant deux étages de rotacteurs haptiques partageant le même axe. Le rotacteur du premier niveau servira à effectuer des réglages grossiers, tandis que le rotacteur du second niveau servira à effectuer des réglages fins (par exemple, pour la cible

d'altitude du pilote automatique, le niveau grossier changera les centaines de pieds tandis que le niveau fin réglera pied par pied).

Ces dispositifs n'étant pas disponibles à des fins de recherche (exclusifs actuellement aux grosses séries ou aux constructeurs automobiles) nous avons remplacé les rotacteurs haptiques à deux niveaux, par deux roues encodeuses simples.

L'intérêt de la gestion informatique de l'entrée réside en la possibilité d'insertion d'une fonction de transfert, et permet donc d'ajuster la sensibilité de rotation en fonction de l'accélération ou de la vitesse de rotation.

Pour pouvoir ajouter les roues encodeuses, nous nous sommes servis d'une interface Phidget [1] ainsi que de potentiomètres infinis. Phidget fournit une libraire permettant l'accès à des composants électroniques passifs, via l'interface de conversion analogique numérique. Outre le code permettant l'instanciation des objets Java de l'interface (les pilotes selon la figure 3.5), il a fallu réaliser un modèle de périphérique virtuel, que nous présentons partiellement en figure 8.2. L'interface ayant 8 emplacements, nous avons utilisé les deux premiers emplacements et doublé ce modèle (les 2 roues encodeuses sont identiques).

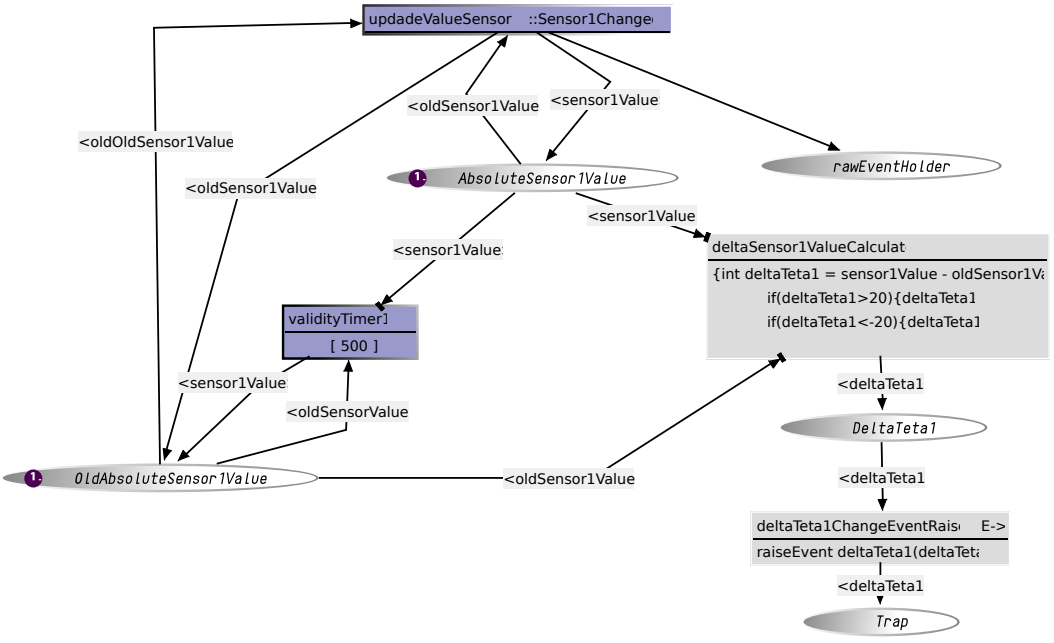


FIGURE 8.2 – Périphérique virtuel du phidget de roue encodeuse

Le périphérique logique des roues encodeuses consiste seulement en une fonction de transfert propre à chaque zone sensible, il n'y a pas de représentation informatique prévue. Le projet étant encore en cours au moment de la rédaction de ce manuscrit, nous n'avons pas encore de spécification de ces fonctions de transfert. Pour les besoins de la

1. phidget disponibles sur https://www.phidgets.com/ accès en juin 2017

démonstration, elles consistent en un gain constant de 100 et de 1, permettant de changer les centaines et les unités de l'altitude de l'aéronef dans la simulation.

8.3.3.2 Interface et interactions du prototype

La figure 8.3 présente la fenêtre JavaFX de notre prototype d'illustration de fonctionnalités.

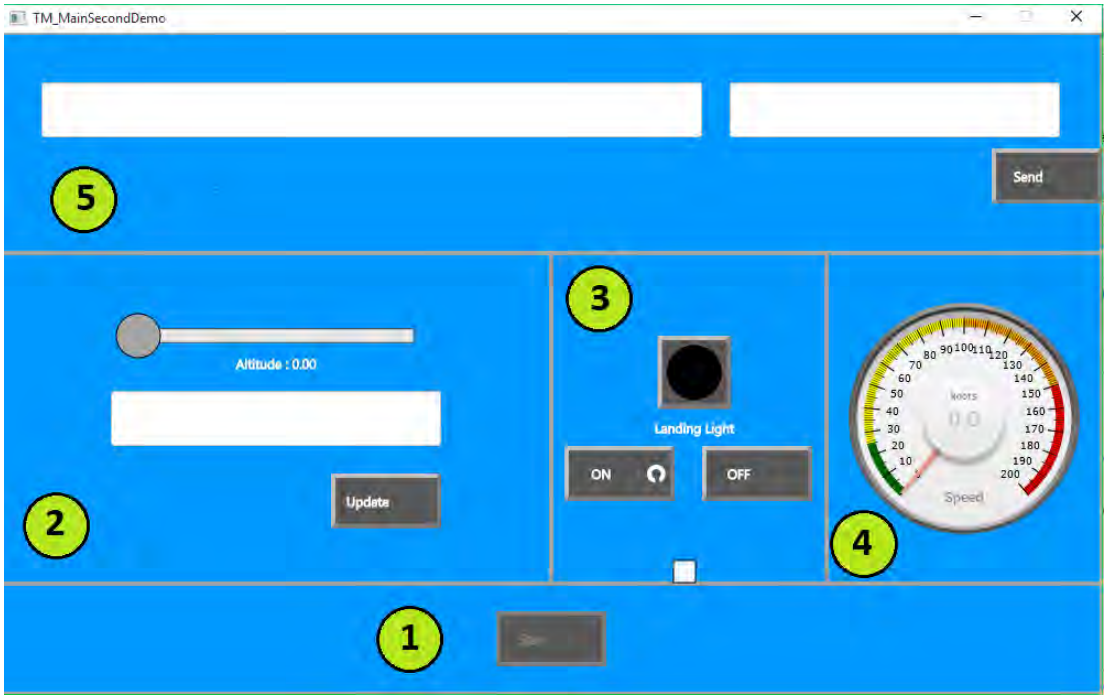


FIGURE 8.3 – Application illustrative des fonctionnalités de la plateforme opérationnelle

Le groupe de gestion de l'application (numéroté 1 en bas) contient le bouton « Start ». Au lancement de l'application, tout est désactivé, le bouton « Start » devient actif dès que la tablette se connecte au contrôleur central. Lorsqu'on appuie sur « Start », le contrôleur crée les modèles ICO qui gèrent le comportement des différents groupes de widgets. Ils vont envoyer des messages à la tablette pour activer les widgets et désactiver le bouton « Start », et s'abonner aux paramètres de X-Plane nécessaires.

Le groupe de gestion de l'altitude et de la radio (numéroté 2 à gauche) contient un *Slider*, un *Textfield* et un bouton *Update*. Le *Slider* permet de changer instantanément l'altitude de l'aéronef sur X-Plane. Lorsqu'on active le bouton *Update*, après environ 500 ms le *TextField* affiche la fréquence de la radio de l'aéronef. Les roues encodeuses permettent aussi de changer la fréquence de la radio, que l'on verra s'afficher directement dans le cockpit.

Le groupe de gestion de la lumière (numéroté 3 au centre) : Il contient une lampe et deux boutons « ON » et « OFF ». La lampe s'allume et les boutons s'activent en fonction de la lumière d'atterrissage de l'aéronef présent sur X-Plane. Les boutons

permettent d'allumer ou d'éteindre cette lumière. On peut voir un oculus tourner à côté des boutons lorsqu'on a appuyé dessus, bien que la valeur de la lampe n'ait pas encore changé.

Le groupe d'affichage de la vitesse (numéroté 4 à droite) : Il contient l'indicateur de vitesse, qui affiche en temps réel la vitesse de l'aéronef.

Le groupe de gestion des paramètres de X-Plane (numéroté 5 en haut) : Il contient un bouton *Send* et deux *TextField* :

Le premier va contenir le nom d'un paramètre de X-Plane (exemple : sim/flight-model/position/local_x), et le deuxième la valeur.

Lorsqu'on entre le nom d'un paramètre et qu'on appuie sur *Send*, le deuxième *TextField* va afficher la valeur du paramètre.

Lorsqu'on entre le nom et la valeur, le paramètre sera modifié sur X-Plane avec la valeur entrée, la valeur dans le deuxième TextField est remplacée par « Sent ! ». Si plusieurs tablettes sont connectées, seule celle sur laquelle le bouton a été appuyé reçoit les modifications du deuxième TextField.

Ce prototype permet d'illustrer les fonctionnalités offertes par la plateforme opérationnelle et notamment :

- La possibilité d'utiliser sur la plateforme des capteurs non informatiques tels que les potentiomètres
- L'abonnement à des paramètres internes à la simulation dans X-Plane
- La demande et la mise à jour de la valeur de paramètres internes à la simulation dans X-Plane
- La modification de la valeur de paramètres internes à la simulation dans X-Plane
- La possibilité d'envoyer des informations à une machine responsable de l'interaction ou à toutes les machines (tablettes tactiles sur l'architecture de la figure 8.1)

8.4 Support au prototypage

Ce chapitre n'a pour objectif de démontrer ni l'utilisation de l'architecture ni son instanciation sur une étude de cas, cela ayant été réalisé dans les deux chapitres précédents (6 & 7). Le but est de montrer qu'une fois l'architecture instanciée et que chacun des modèles est été créé, l'environnement de développement Petshop couplé avec ARISSIM constitue un outil de prototypage de systèmes interactifs critiques (techniques d'interaction et applications).

Les trois sections suivantes vont montrer respectivement comment ajouter des périphériques d'entrée, comment modifier une technique d'interaction et comment modifier le dialogue de l'application interactive.

8.4.1 Ajout de périphérique d'entrée ou de sortie

L'inclusion d'un nouveau périphérique et de ses interactions implique l'ajout de tous les composants de la chaîne de gestion d'entrée. Prenons un exemple d'ajout du multi souris à la plateforme opérationnelle. En plus des périphériques, il faut ajouter le ou les pilotes, les périphériques virtuels, les périphériques logiques ainsi que le gestionnaire de chaîne d'entrée. Les modèles de tels composants ont déjà été présentés au chapitre 6 et sont inclus dans la chaîne d'entrée présentée en figure 8.4.

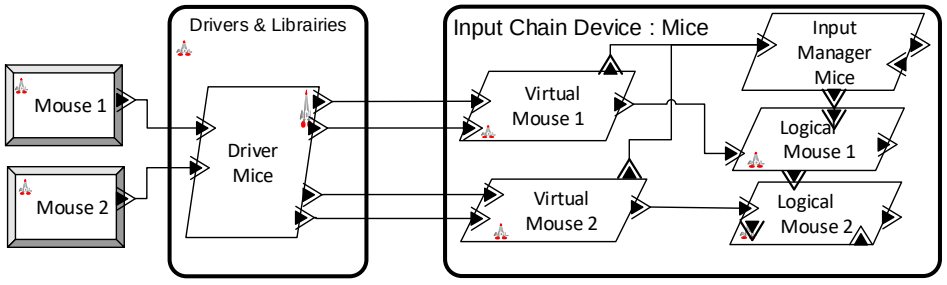


FIGURE 8.4 – Chaîne d'entrée relative à l'ajout de deux souris

Si on veut inclure des techniques d'interaction du type clic, double clic et leurs équivalents en synergie entre les deux curseurs, il faut rajouter un modèle de technique d'interaction global tel que celui présenté en figure 8.5.

Afin d'inclure un nouveau périphérique ou une nouvelle modalité, deux possibilités s'offrent au développeur :

- Soit on modifie les zones sensibles ou l'application pour qu'elles réagissent aux nouveaux événements, créant ainsi un nouvel ensemble d'interaction et d'évolution de l'application grâce au nouveau périphérique
- Soit on adapte les événements du nouveau périphérique pour déclencher l'évolution des zones sensibles ou de l'application existante. Cette solution permet de déclencher l'évolution de l'application avec de nouvelles techniques d'interaction sans modifier les comportements existants (par exemple, sur les systèmes d'exploitation actuels, un *tap* avec un doigt sur une dalle tactile sera interprété comme un *clic* d'un curseur).

Dans le cas d'un périphérique complètement nouveau, il faudra commencer par modéliser et développer les différents composants de sa chaîne de gestion, allant du pilote jusqu'aux techniques d'interaction.

Grâce au respect du paradigme des systèmes interactifs qui décrit que les entrées sont basées sur les événements et que les sorties sont basées sur les états, il est aussi possible d'inclure des modalités qui ne sont pas entièrement modélisées dans l'environnement Petshop en incluant simplement un composant de type *technique d'interaction globale*,

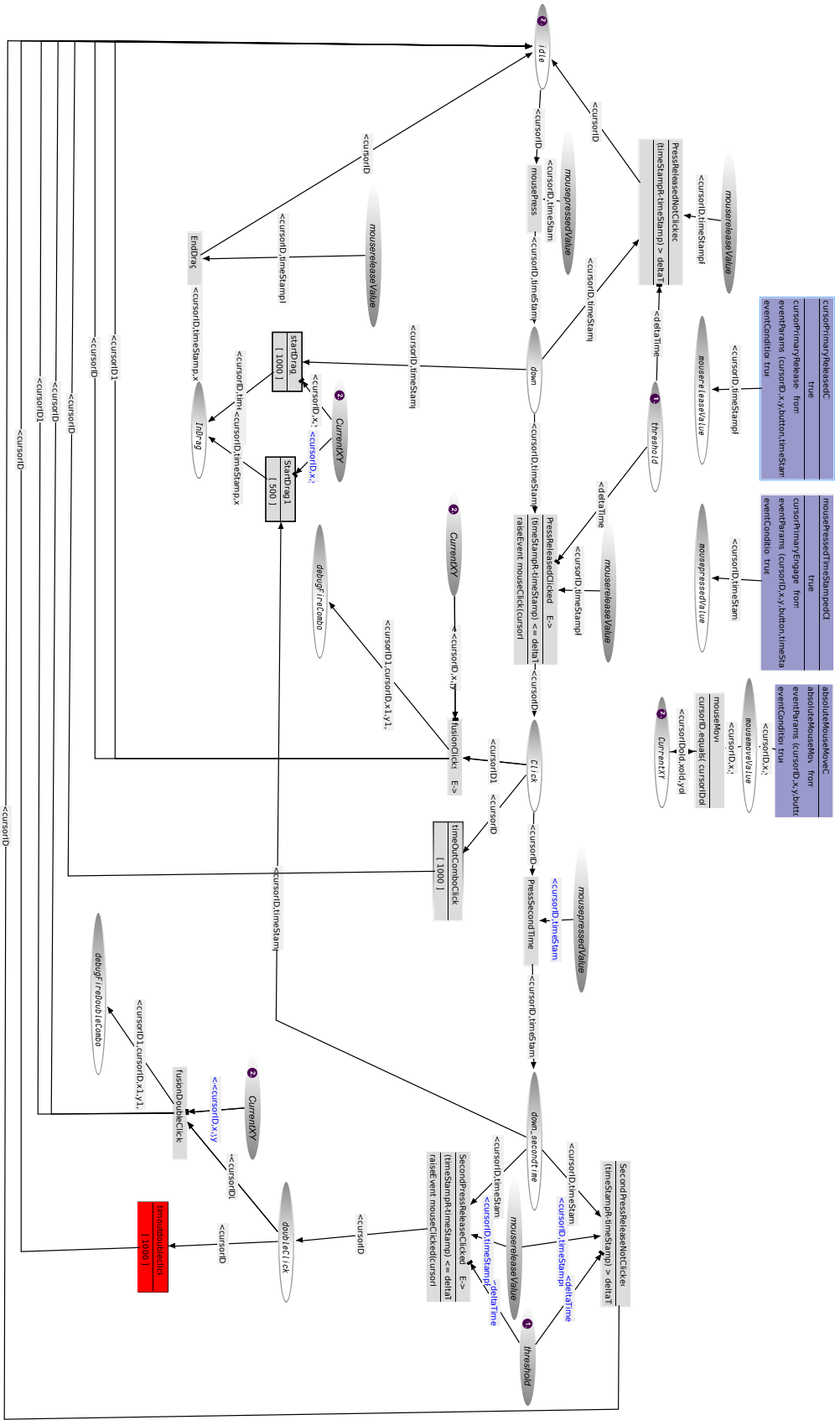


FIGURE 8.5 – Modèle de technique d'interaction multi souris, incluant la synergie des deux curseurs *ComboClic* et *ComboDoubleClic*

transformant un événement extérieur en événement Petshop. C'est ce que nous avons fait avec le Leap Motion (présenté au chapitre 7). Le SDK permet de décrire à la fois le périphérique virtuel (le squelette de la main), mais fournit aussi des techniques d'interaction. Nous avons donc décrit un modèle permettant d'utiliser les interactions du Leap Motion telles que le cercle dans l'espace ou encore utilisant la quantité d'ouverture de la main, et ce, sans avoir réalisé la modélisation ICO de la reconnaissance de geste.

8.4.2 Modification d'une technique d'interaction

La modification d'une technique d'interaction se fera au travers de la modification des différents modèles liés à l'interaction. Le développeur fera face à plusieurs possibilités :

- La modification porte sur l'interacteur. On devra venir modifier généralement les fonctions de transfert dans le périphérique logique. Par exemple dans le cadre de la souris, prendre en compte l'accélération ou changer le gain associé au curseur.
- La modification porte sur une technique d'interaction globale. On devra modifier le modèle correspondant à cette interaction. Par exemple, on peut modifier la fenêtre de validité de prise en compte du double clic combiné dans le modèle ICO de la figure 8.5 en ajustant la valeur temporelle de la transition *timeOutDoubleClic* (en rouge, en bas à droite de la figure 8.5) (fixé à 1000ms sur le modèle).
- La modification porte sur les zones sensibles. On devra alors ajuster le comportement des zones sensibles en fonction des événements déclenchés par les événements en provenance des périphériques logiques. Ces modifications peuvent être lourdes à mettre en place, car elles peuvent impacter plusieurs modèles. Par exemple, ajouter la gestion d'interaction multi souris pour une zone sensible peut impacter les modèles des zones sensibles, leur rendu et éventuellement les modèles des périphériques logiques et leur rendu (notamment s'ils doivent s'adapter à ces nouvelles zones). Par exemple, ajouter la gestion de deux curseurs sur un bouton pourra impacter, en plus du comportement du bouton, le rendu du bouton (e.g. ajouter un highlight pour chacun des curseurs et un combiné, idem pour l'appui sur le bouton). Cela pourra aussi impacter les curseurs (ajout d'un état relatif au lien, par exemple le curseur change de comportement au-dessus d'une zone sensible, auquel il faudra ajouter un rendu) .
- La modification porte sur le comportement de l'application. Si la répartition de fonctionnalité a été respectée en suivant les recommandations de MIODMIT, l'évolution de l'application n'est pas impactée par la technique d'interaction directement, mais seulement par l'événement déclenchant l'évolution. Nous ne devrions donc pas avoir à modifier l'application dans le cadre d'une modification de technique d'interaction. De la même manière, si le comportement de l'application change, il n'y aura pas forcément de modification des techniques d'interaction en entrée.
- La modification porte sur le rendu. Il faudra tout d'abord identifier quelles sont les places, ou plus précisément quel est le marquage des réseaux de Petri qui correspond

aux nouveaux états que l'on veut rendre. Il faudra ensuite vérifier les fonctions de concrétisation du rendu (présentées en section 6.4.4) et modifier le rendu effectif dans les fonctions de rendu de présentation.

L'analyse menant aux modifications est supportée par les fonctionnalités de log de Petshop (présentées à la section 4.1.5). [Palanque et al., 2011] décrit le processus permettant d'évaluer des techniques d'interaction et l'intérêt d'utiliser la modélisation ICO pour supporter la conception. La figure 4.4 décrit informellement le processus itératif et l'architecture permettant cette analyse.

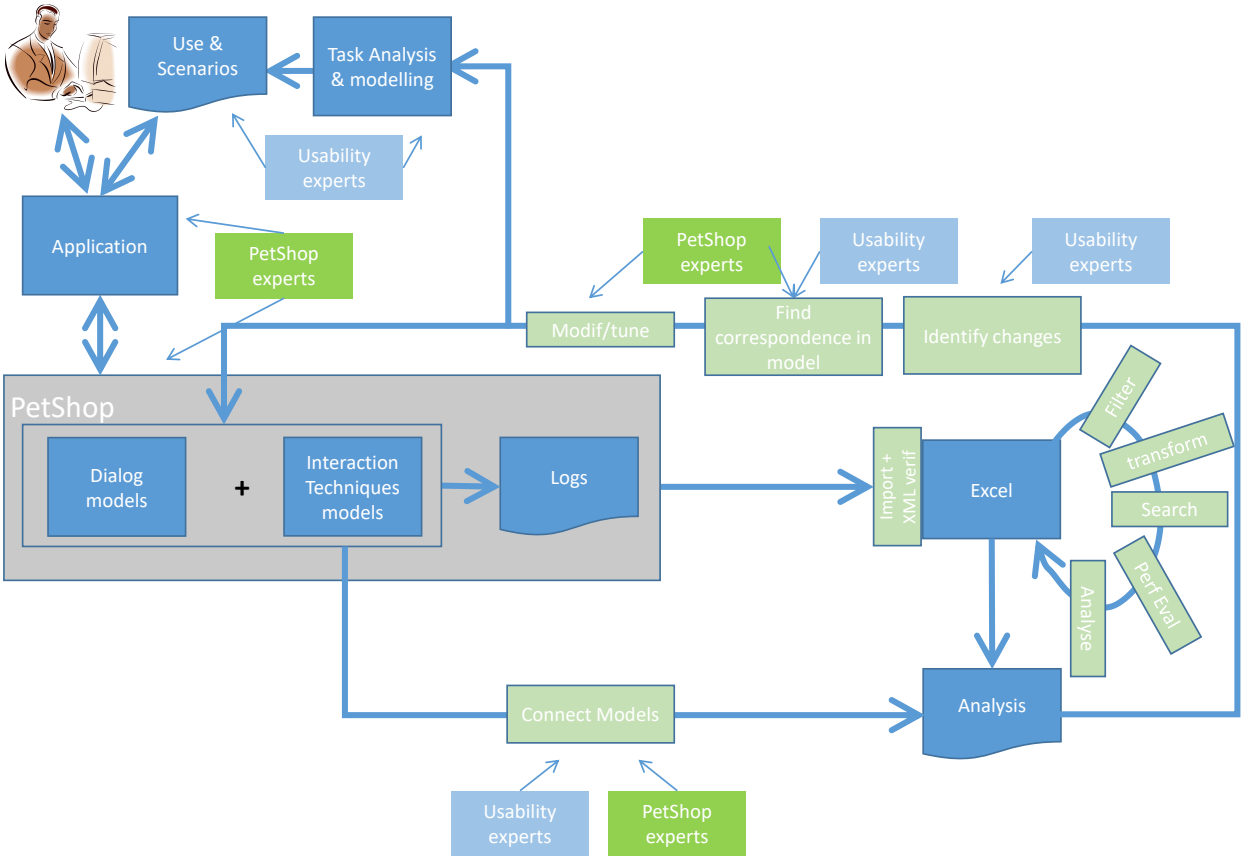


FIGURE 8.6 – Processus d'analyse de technique d'interaction grâce au log, extrait de la présentation de [Palanque et al., 2011]

Dans leurs travaux, ils décrivent une technique d'interaction similaire à celle présentée en figure 8.5, et analysent grâce aux log (présentés en figure 8.7) la réussite de l'interaction clic combiné (*comboClic*) en fonction de l'intervalle de temps.

	class	type	name	action	time	data1	data2
605	mouse_transducer	transition	mousePress_t1	fire	8000		1*{evt:{mId=>1}}
609	mouse_transducer	transition	mouseRelease_t1	fire	9900		1*{evt:{mId=>1}}
613	mouse_transducer	transition	timerExpired	fire	10500		1*{evt:{mId=>1}}
617	mouse_transducer	transition	click	fire	10700		1*{evt:{mId=>1}}
650	mouse_transducer	transition	click	fire	10950		1*{evt:{mId=>2}}
752	mouse_transducer	transition	mousePress_t1	fire	12000		1*{evt:{mId=>1}}
758	mouse_transducer	transition	mousePress_t1	fire	12010		1*{evt:{mId=>2}}
761	mouse_transducer	transition	mouseRelease_t1	fire	12020		1*{evt:{mId=>2}}
774	mouse_transducer	transition	timerExpired	fire	12220		1*{evt:{mId=>2}}
781	mouse_transducer	transition	click	fire	12420		1*{evt:{mId=>2}}
794	mouse_transducer	transition	mouseRelease_t1	fire	13010		1*{evt:{mId=>1}}
1425	CombinedClick_Delete_File	transition	combinedClick	fire	20300		1*{evt:{x1=>40,x2=>400,y1=>40,y2=>400},presentationFrame=>javax....}
1426	CombinedClick_Delete_File	place	testIcon	tokenAdded	20300	1	Icon1,Trash
1427	CombinedClick_Delete_File	place	Frame	tokenRemoved	20300		
1428	CombinedClick_Delete_File	transition	fileIconAndTrashSelected	fire	20300		1*{Icon2,Trash}
1435	CombinedClick_Delete_File	transition	combinedClick	fire	22100		1*{evt:{x1=>40,x2=>400,y1=>80,y2=>400},presentationFrame=>javax....}
1436	CombinedClick_Delete_File	place	testIcon	tokenAdded	22100	1	Icon2,Trash
1437	CombinedClick_Delete_File	place	Frame	tokenRemoved	22100		
1438	CombinedClick_Delete_File	transition	fileIconAndTrashSelected	fire	22100		1*{Icon1,Trash}
1699	CombinedClick_Delete_File	transition	combinedClick	fire	23450		1*{evt:{x1=>290,x2=>400,y1=>40,y2=>400},presentationFrame=>javax....}
1700	CombinedClick_Delete_File	place	testIcon	tokenAdded	23450	1	null,Trash
1701	CombinedClick_Delete_File	place	Frame	tokenRemoved	23450		
1702	CombinedClick_Delete_File	transition	fileIconAndTrashNotSelecte	fire	23450		1*{null,Trash}
2897	CombinedClick_Delete_File	transition	combinedClick	fire	40340		1*{evt:{x1=>120,x2=>400,y1=>120,y2=>400},presentationFrame=>javax....}
2898	CombinedClick_Delete_File	place	testIcon	tokenAdded	40340	1	Icon6,Trash
2899	CombinedClick_Delete_File	place	Frame	tokenRemoved	40340		
2900	CombinedClick_Delete_File	transition	fileIconAndTrashSelected	fire	40340		1*{Icon6,Trash}

FIGURE 8.7 – Log d'un test utilisateur pour analyser la réussite de l'interaction clic combiné, extrait de [Palanque et al., 2011]

8.4.3 Modification du dialogue

Grâce à l'approche modulaire de développement du dialogue, des entrées et des sorties, la modification du comportement de l'application ne doit impacter qu'au minimum les entrées. Il pourra néanmoins impacter les modèles relatifs au rendu de l'application. De telles modifications ont été présentées dans les travaux de [Palanque et al., 2009], [Navarre et al., 2009] et [Bastide et al., 2002].

8.5 Conclusion

Cette plateforme opérationnelle offre un environnement de simulation de cockpits et permet de greffer des prototypes d'applications interactives mettant en œuvre des techniques d'interaction multimodales complexes.

Les périphériques déjà modélisés (tous n'ont pas été présentés dans les exemples) et utilisables rapidement sur cette plateforme sont :

- En entrée
  - — Des souris

  - Des dalles tactiles
  - Un Leap Motion
  - Des roues encodeuses
  - Un eye tracker Tobii
  - Une kinnect
- En sortie
  - Un ou plusieurs écrans
  - Une synthèse vocale

En outre, on peut utiliser le clavier via les librairies traditionnelles Java. La plateforme opérationnelle est évolutive, grâce à sa structure respectant le modèle d'architecture MIODMIT. On pourra donc ajouter ou supprimer des périphériques à la convenance des concepteurs.

Cette plateforme opérationnelle propose un outil destiné aux équipes pluridisciplinaires responsables de la conception des interactions et des interfaces homme-machine pour les cockpits d'aéronef de prochaine génération. Grâce à la plateforme opérationnelle, les ergonomes, les spécialistes d'IHM et les développeurs, disposeront d'un outil leur permettant d'exécuter et de tester avec des utilisateurs, les prototypes d'applications interactives spécifiées auparavant de manière complète et non ambiguë sur l'environnement de développement Petshop. Leur processus itératif sera supporté par les fonctionnalités d'analyse statique mathématiques de modèle et par la possibilité d'analyser les log des tests utilisateur.

Conclusion et Perspectives

Ce travail de thèse avait pour objectif de contribuer au domaine de l'ingénierie des systèmes interactifs multimodaux critiques. En ce qui concerne les aspects multimodalité, nous nous sommes intéressés aux nouveaux dispositifs d'entrée (comme les tablettes mutli-touch, les systèmes de reconnaissance de geste dans un espace en 3 dimensions (p. ex. Leap ou Kinect)) mais aussi la combinaison de multiples systèmes d'entrée plus standards comme les souris. Concernant les aspects criticité, nous avons pour cible des systèmes de commande et contrôle de système pour lesquels les coûts d'une défaillance sont bien au-delà des coûts de développement. Ces coûts peuvent être dus à des pertes de services ou même à des pertes humaines.

Ces travaux se sont basés sur une étude détaillée de la littérature dans les domaines de recherche pertinents et couvrent en particulier l'Interaction Homme-Machine (pour les aspects utilisabilité et interactions multimodales), le génie du logiciel interactif (pour les aspects architectures et processus de développement) et enfin le génie des systèmes critiques (pour les aspects normes de développement, tolérance aux fautes et sureté de fonctionnement).

Plus précisément, les contributions présentées dans ce mémoire sont proposées comme des compléments et des extensions aux méthodes, processus, techniques et outils existants et ont pour but d'augmenter la sureté de fonctionnement, la modifiabilité et, dans une moindre mesure l'utilisabilité des systèmes interactifs multimodaux critiques.

Ces contributions peuvent être regroupées autour de trois axes :

Le modèle d'architecture logicielle et matérielle MIODMIT [2] qui vise l'intégration de périphériques instanciables dynamiquement et permettant l'usage de multimodalité complexe au sein de systèmes critiques. Il permet d'organiser et de décrire de façon précise et non ambiguë le flux d'informations. Ce modèle décrit aussi précisément les rôles de chacun des composants ainsi que les relations qu'ils entretiennent. Il couvre l'ensemble du spectre du système interactif multimodal allant des périphériques d'entrée et leurs pilotes, vers les techniques d'interaction et l'application interactive. Il décrit aussi le rendu allant de l'application interactive aux périphériques de sortie en passant par les techniques complexes de présentation permettant la fusion et la fission d'informations. Au-delà de sa capacité de description, ce modèle d'architecture assure la modifiabilité de la configuration du système (ajout ou suppression de périphérique au moment du design et de l'exécution).

2. Multiple Input Output devices Multiple Interaction Techniques

Nous avons exploité le langage formel ICO (Interactive Cooperative Objects), pour décrire de façon complète et non ambiguë chacun des composants de l'architecture. Ce langage étant exploitable au moyen d'un outil d'édition et d'interprétation appelé Petshop, qui permet de faire fonctionner l'application interactive dans son ensemble (des périphériques d'entrée aux périphériques de sortie). Nous avons complété cet environnement de développement et d'exécution par une plateforme d'exécution que nous avons appelée ARISSIM [3] de rajouter des mécanismes de sureté de fonctionnement aux systèmes interactifs multimodaux développés avec Petshop. Plus précisément ARISSIM permet la ségrégation spatiale (attribution d'espace mémoire d'exécution à des partitions dédiées pour éviter les contaminations aux autres processus du système interactif en cas de faute) et la ségrégation temporelle des processus (utilisation séquentialisée du processeur par les différentes partitions) ce qui accroît fortement la tolérance aux fautes durant l'exécution.

La modélisation à base d'ICO des techniques d'interaction fait apparaître qu'une partie importante du comportement est autonome c'est-à-dire qu'il évolue sans recevoir d'entrées produites par l'utilisateur. Ce genre de comportement autonome est difficile à comprendre et à anticiper pour les utilisateurs, entrainants des erreurs appelées *automation surprises*. Nous avons proposé une méthode d'évaluation à base de modèles des techniques d'interaction permettant d'analyser (pour ensuite réduire significativement) les erreurs d'utilisation liées à ces comportements inattendus et incompréhensibles.

Ces contributions ont été illustrées et validées au travers de trois chapitres.

Le premier nous a permis de décrire le comportement des composants de MIODMIT et d'illustrer les fonctionnalités et le rationnel du découpage en composants.

Le second a démontré la faisabilité de l'approche y compris dans le cas de systèmes interactifs offrant des techniques d'interaction en entrée et en sortie complexes.

Enfin nous avons présenté une plateforme opérationnelle de prototypage de techniques d'interaction dans un contexte avionique intégrant un simulateur de vol et de multiples périphériques d'entrée (tablettes, multi-souris , Leap Motion qui pratique la reconnaissance de gestes, etc.) et de périphériques de sortie (affichages sur écrans multiples et synthèse vocale). Cette plateforme opérationnelle est évolutive (permet l'ajout de nouveaux périphériques) et entièrement structurée autour de MIODMIT, démontrant l'exploitabilité des contributions dans un contexte de développement de systèmes interactifs.

En résumé, ces contributions nous permettent de prendre en compte un ensemble significatif des fautes décrites dans la classification de la figure 8.8. En effet la plateforme de développement Petshop couplée au langage formel ICO, le modèle d'architecture MIODMIT, la plateforme d'exécution ARISSIM le tout dans le respect de la norme DO-178C de développement de systèmes avioniques (d'autres processus sont aussi compatibles comme celui de [CEI, c]) permettent de couvrir un ensemble de faute :

Les fautes générées durant le développement (logicielles en orange et matérielles en

3. ARINC 653 Standard SIMulator

violet sur la figure 8.8)sont réduites par l'utilisation d'un formalisme et d'une approche de conception zéro défaut. L'identification de fautes est facilitée par l'environnement de développement Petshop couplé à la plateforme d'exécution ARISSIM.

Les fautes en opération sont couvertes de plusieurs manières :

Les fautes naturelles (en vert sur la figure 8.8)sont couvertes par la possibilité d'inclure des mécanismes de recouvrement dans le système

Les fautes et erreurs humaines (en bleu sur la figure 8.8)sont couvertes dans notre approche par la systématisation de l'identification des potentielles surprises liées à l'utilisation des systèmes interactifs.

Les fautes malicieuses (en rouge sur la figure 8.8) sont en dehors du périmètre de cette thèse. Ce sujet extrêmement important est d'actualité avec la volonté d'intégrer les bénéfices du monde connecté aux systèmes critiques. Ce sujet est toutefois complexe dans la mesure où les propriétés de sécurité entrent souvent en conflit avec les autres propriétés telles que la sureté de fonctionnement ou l'utilisabilité [Sasse et al., 2001].

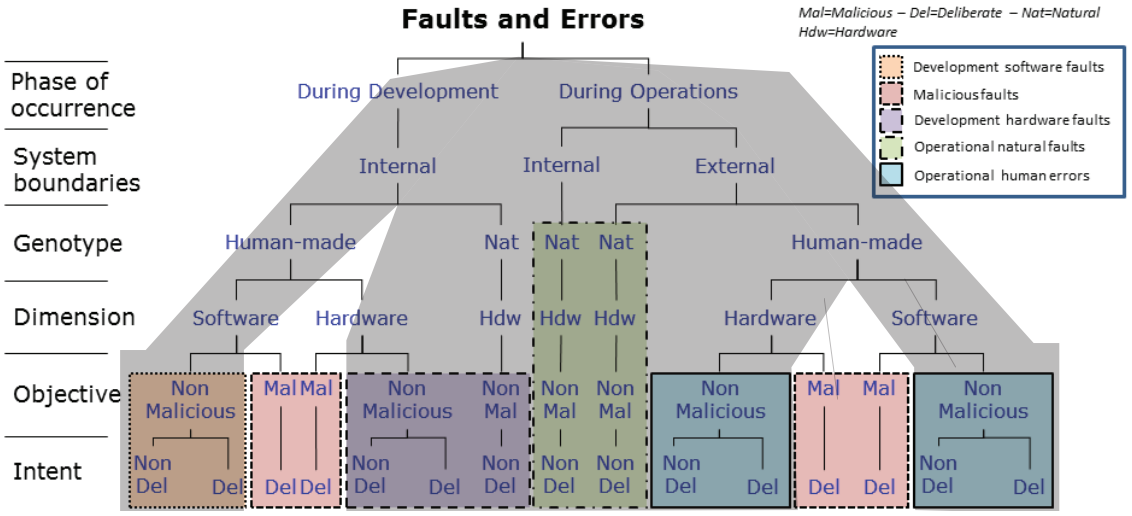


FIGURE 8.8 – Couverture des travaux de thèse par rapport à la classification des fautes des systèmes informatiques de [Laprie et al., 1995]

Cette non-prise en compte de la sécurité (dans le sens sécurité immunité) dans le périmètre de la thèse démontre l'étendue du domaine de recherche dans lequel se situent ces travaux. La prise en compte de cette propriété nécessiterait l'intégration de connaissances d'un domaine, certes connexe, mais fortement éloigné du point de vue scientifique et d'ingénierie. Sans aller aussi loin, nous avons identifié un ensemble de perspectives de recherche qui pourraient être menées dans le but de compléter les

contributions présentées précédemment. Parmi l'ensemble des perspectives possibles, nous en avons sélectionné trois qui vont être détaillées en commençant par la plus simple à implémenter (et la plus concrète) pour finir par la plus complexe.

Une première perspective serait de fournir une meilleure intégration de la plateforme d'exécution ARISSIM avec l'environnement de développement Petshop afin de contrôler plus finement les processus du système interactifs lors de l'exécution. Pour le moment, un processus d'ARISSIM contenant des modèles ICO contient forcément une instance de Petshop. Cela pourrait être décorrélé dans la simulation. C'est-à-dire décrire finement l'ordonnancement des processus liés aux modèles dans l'interpréteur Java. Nous pourrions ainsi évaluer la robustesse des processus en injectant des fautes de manière fine et surtout évaluer la fiabilité des mécanismes de sureté de fonctionnement proposés. Ceci est extrêmement important, car sans des campagnes d'injection de fautes précises et détaillées il est très difficile de connaitre le niveau de protection offert par ces mécanismes. Des approches telles que celles proposées par [Arlat, 1990].

Une deuxième perspective d'évolution réside dans l'intégration d'un éditeur de composants MIODMIT dans l'environnement Petshop. L'objectif est de *packager* les composants de l'architecture modélisés de manière formelle pour pouvoir les réutiliser plus facilement en les arrangeant via un utilitaire graphique basé sur AADL. Les principales difficultés résident dans l'édition des composants et de leurs connexions. Il faut aussi atteindre une gestion aisée des aspects fonctionnels (les fonctions mises en œuvre par les composants) et des aspects non fonctionnels (leur initialisation, leur instanciation ...). Dans ce contexte particulier, l'intégration simplifiée des mécanismes de sureté de fonctionnement (comme les composants auto-testables) permettrait une intégration facilitée et, de fait, un déploiement plus large de cette technologie.

Enfin, d'un point de vue méthodologique, il serait fondamental d'intégrer les contributions présentées dans cette thèse avec les processus de conception et de développement actuellement utilisés pour concevoir et évaluer des systèmes interactifs. Ces processus (voir par exemple [ISO, b]) mettent en avant une conception centrée utilisateur impliquant les utilisateurs (ainsi que les designers, les développeurs, les ergonomes ...) dans la conception et l'évaluation de ces systèmes. Ces processus pluridisciplinaires sont antinomiques aux processus séquentiels mis en avant par les normes de développement de systèmes critiques qui mettent en œuvre une pluridisciplinarité différence (ingénieurs en fiabilité, en sureté de fonctionnement, développement logiciel, assurance qualité ...). Dans ces systèmes la formation des opérateurs/utilisateurs est une production cruciale permettant d'assurer l'utilisation complète, correcte et performante du système. Ces trois éléments ne font pas partie des approches centrées utilisateurs qui considèrent qu'un bon design évite la formation des utilisateurs. La route sera encore longue avant d'intégrer ces approches pour concevoir, développer et valider des systèmes interactifs critiques. En effet, la vision très différente de l'exploitation d'un système par ses utilisateurs, la différence fondamentale

des processus de développement et l'entrée en conflit de nombreuses propriétés (p. ex. fiabilité d'un côté, utilisabilité et ressenti utilisateur de l'autre), restent des obstacles majeurs à leur unification dans un processus de développement universel pour les systèmes interactifs critiques.

Liste des publications

Articles Publiés

Les travaux de thèse ont été publiés dans un ensemble d'article de conférences :

1. A software-implemented fault-tolerance approach for control and display systems in avionics [Fayollas et al., 2014a]
2. Multi-Touch Interactions for Control and Display in Interactive Cockpits : Issues and a Proposal [Hamon et al., 2014a]
3. Transparent Automation for Assessing and Designing Better Interactions Between Operators and Partly-Autonomous Interactive Systems [Bernhaupt et al., 2015]
4. A Three-fold Approach Towards Increased Assurance Levels for Interactive Systems : A Flight Control Unit Case Study [Fayollas et al., 2016]
5. Dependable multi-touch interactions in safety critical industrial contexts : Application to aeronautics [Hamon et al., 2015]
6. Résilience des systèmes interactifs : contribution par une architecture tolérante aux fautes [Fayollas et al., 2014b]
7. Formal Modelling of Dynamic Instantiation of Input Devices and Interaction Techniques : Application to Multi-touch Interactions [Hamon et al., 2014b]

Articles en cours de soumission

MIODMIT, Select, Connect and Tune on Demand : A Generic Software Architecture for Handling Complexity and Flexibility of Interactive Critical Systems, soumission prévue à EICS 2017

Table des figures

Bibliographie

[Accot et al., 1997] Accot, J., Chatty, S., Maury, S., and Palanque, P. (1997). Formal transducers : models of devices and building bricks for the design of highly interactive systems. In *Design, Specification and Verification of Interactive Systems' 97*, pages 143–159. Springer.

[Accot et al., 1996] Accot, J., Chatty, S., and Palanque, P. (1996). A formal description of low level interaction and its application to multimodal interactive systems. In *Design, Specification and Verification of Interactive Systems' 96*, pages 92–104. Springer.

[Arlat, 1990] Arlat, J. (1990). *Validation de la sûreté de fonctionnement par injection de fautes : méthode, mise en oeuvre, application*. PhD thesis, Toulouse, INPT.

[Avizienis et al., 2004] Avizienis, A., Laprie, J.-C., Randell, B., and Landwehr, C. (2004). Basic concepts and taxonomy of dependable and secure computing. *IEEE Transactions on Dependable and Secure Computing*, 1(1) :11–33.

[Barboni et al., 2003] Barboni, E., Bastide, R., Lacaze, X., Navarre, D., and Palanque, P. (2003). Petri net centered versus user centered petri nets tools. In *10th Workshop Algorithms and Tools for Petri Nets, AWPN*.

[Barboni et al., 2010] Barboni, E., Ladry, J.-F., Navarre, D., Palanque, P., and Winckler, M. (2010). Beyond modelling : an integrated environment supporting co-execution of tasks and systems models. In *Proceedings of the 2nd ACM SIGCHI symposium on Engineering interactive computing systems*, pages 165–174. ACM.

[Barnard and May, 1995] Barnard, P. and May, J. (1995). Interactions with advanced graphical interfaces and the deployment of latent human knowledge. In *Interactive Systems : Design, Specification, and Verification*, pages 15–49. Springer.

[Bass, 2007] Bass, L. (2007). *Software architecture in practice*. Pearson Education India.

[Bass and John, 2003] Bass, L. and John, B. E. (2003). Linking usability to software architecture patterns through general scenarios. *Journal of Systems and Software*, 66(3) :187–197.

[Bastide et al., 2002] Bastide, R., Navarre, D., and Palanque, P. (2002). A model-based tool for interactive prototyping of highly interactive applications. In *CHI'02 extended abstracts on Human factors in Computing Systems*, pages 516–517. ACM.

[Bastide and Palanque, 1990] Bastide, R. and Palanque, P. A. (1990). Petri net objects for the design, validation and prototyping of user-driven interfaces. In *Interact*, volume 90, pages 625–631.

[Bau and Mackay, 2008] Bau, O. and Mackay, W. E. (2008). Octopocus : a dynamic guide for learning gesture-based command sets. In *Proceedings of the 21st annual ACM symposium on User interface software and technology*, pages 37–46. ACM.

[Beck et al., 2001] Beck, K., Beedle, M., Van Bennekum, A., Cockburn, A., Cunningham, W., Fowler, M., Grenning, J., Highsmith, J., Hunt, A., Jeffries, R., et al. (2001). The agile manifesto.

[Bernhaupt, 2015] Bernhaupt, R. (2015). User experience evaluation methods in the games development life cycle. In *Game User Experience Evaluation*, pages 1–8. Springer.

[Bernhaupt et al., 2015] Bernhaupt, R., Cronel, M., Manciet, F., Martinie, C., and Palanque, P. (2015). Transparent automation for assessing and designing better interactions between operators and partly-autonomous interactive systems. In *Proceedings of the 5th International Conference on Application and Theory of Automation in Command and Control Systems*, ATACCS '15, pages 129–139, New York, NY, USA. ACM.

[Berry, 1988] Berry, R. E. (1988). Common user access ; a consistent and usable human-computer interface for the saa environments. *IBM Systems Journal*, 27(3) :281–300.

[Bier et al., 1993] Bier, E. A., Stone, M. C., Pier, K., Buxton, W., and DeRose, T. D. (1993). Toolglass and magic lenses : the see-through interface. In *Proceedings of the 20th annual conference on Computer graphics and interactive techniques*, pages 73–80. ACM.

[Boehm, 1986] Boehm, B. (1986). A spiral model of software development and enhancement. *ACM SIGSOFT Software Engineering Notes*, 11(4) :14–24.

[Bolt, 1980] Bolt, R. A. (1980). *"Put-that-there" : Voice and gesture at the graphics interface*, volume 14. ACM.

[Bonnet et al., 2013] Bonnet, D., Appert, C., and Beaudouin-Lafon, M. (2013). Extending the vocabulary of touch events with thumbrock. In *Proceedings of Graphics Interface 2013*, pages 221–228. Canadian Information Processing Society.

[Bouchet et al., 2004] Bouchet, J., Nigay, L., and Ganille, T. (2004). Icare software components for rapidly developing multimodal interfaces. In *Proceedings of the 6th international conference on Multimodal interfaces*, pages 251–258. ACM.

[Brooke et al., 1996] Brooke, J. et al. (1996). Sus-a quick and dirty usability scale. *Usability evaluation in industry*, 189(194) :4–7.

[Buxton, 2009] Buxton, B. (2009). *Developing a Taxonomy of Input*.

[Campos and Harrison, 1997] Campos, J. C. and Harrison, M. D. (1997). Formally verifying interactive systems : A review. In *Design, Specification and Verification of Interactive Systems' 97*, pages 109–124. Springer.

[Casiez and Roussel, 2011] Casiez, G. and Roussel, N. (2011). No more bricolage ! : Methods and tools to characterize, replicate and compare pointing transfer functions. In *Proceedings of the 24th Annual ACM Symposium on User Interface Software and Technology*, UIST '11, pages 603–614, New York, NY, USA. ACM.

[CEI, a] CEI, C. l. i. Centrales nucléaires de puissance - instrumentation et contrôle-commande importants pour la sûreté - exigences générales pour les systèmes. https://www.boutique.afnor.org/norme/cei-615132011/centrales-nucleaires-de-puissance-instrumentation-et-controle-commande-importants-particle/751659/xs123221.

[CEI, b] CEI, C. l. i. Safety of machinery : Functional safety of electrical, electronic and programmable electronic control systems. https://www.boutique.afnor.org/norme/cei-62061-ac12005/corrigendum-1-a-la-norme-cei-62061-de-janvier-2005/article/709716/xs114697.

[CEI, c] CEI, C. l. i. Sécurité fonctionnelle des systèmes électriques/électroniques/électroniques programmables relatifs à la sécurité. http://www.suretedefonctionnement.fr/p/cei-61508.html.

[Cockton and Gram, 1996] Cockton, G. and Gram, C. (1996). *Design principles for interactive software*. Springer Science & Business Media.

[Collins and Collins, 1995] Collins, D. and Collins, D. (1995). *Designing object-oriented user interfaces*. Benjamin Cummings Redwood City.

[Combéfis et al., 2011] Combéfis, S., Giannakopoulou, D., Pecheur, C., and Feary, M. (2011). Learning system abstractions for human operators. In *Proceedings of the International Workshop on Machine Learning Technologies in Software Engineering*, pages 3–10. ACM.

[Conley, 1976] Conley, B. C. (1976). The value of human life in the demand for safety. *The American Economic Review*, pages 45–55.

[Conversy et al., 2014] Conversy, S., Chatty, S., Gaspard-Boulinc, H., and Vinot, J.-L. (2014). L'accident du vol af447 rio-paris, un cas d'étude pour la recherche en ihm. In *IHM'14, 26e conférence francophone sur l'Interaction Homme-Machine*, pages 60–69. ACM.

[Coutaz, 1987] Coutaz, J. (1987). Pac : An object oriented model for implementing user interfaces. *ACM SIGCHI Bulletin*, 19(2) :37–41.

[Coutaz et al., 1995] Coutaz, J., Nigay, L., Salber, D., Blandford, A., May, J., and Young, R. M. (1995). Four easy pieces for assessing the usability of multimodal interaction : the care properties. In *Human—Computer Interaction*, pages 115–120. Springer.

[Cronel,] Cronel, M. ARISSIM documentation. https://github.com/ARISSIM/ARISS/tree/master/ReadMe. accès en janvier 2017.

[Cuenca et al., 2014] Cuenca, F., Van den Bergh, J., Luyten, K., and Coninx, K. (2014). A domain-specific textual language for rapid prototyping of multimodal interactive systems. In *Proceedings of the 2014 ACM SIGCHI Symposium on Engineering Interactive Computing Systems*, EICS '14, pages 97–106, New York, NY, USA. ACM.

[Curtis and Hefley, 1994] Curtis, B. and Hefley, B. (1994). A wimp no more : the maturing of user interface engineering. *interactions*, 1(1) :22–34.

[Dix, 2009] Dix, A. (2009). *Human-computer interaction*. Springer.

[Dix, 1995] Dix, A. J. (1995). Formal methods : an introduction to and overview of the use of formal methods within hci. *Perspectives on HCI : Diverse Approaches*, pages 9–43.

[Dubey et al., 2010] Dubey, A., Karsai, G., Kereskenyi, R., and Mahadevan, N. (2010). A real-time component framework : experience with ccm and arinc-653. In *Object/Component/Service-Oriented Real-Time Distributed Computing (ISORC), 2010 13th IEEE International Symposium on*, pages 143–150. IEEE.

[Dubey et al., 2011] Dubey, A., Karsai, G., and Mahadevan, N. (2011). A component model for hard real-time systems : Ccm with arinc-653. *Software : Practice and Experience*, 41(12) :1517–1550.

[Dumas et al., 2009] Dumas, B., Lalanne, D., and Oviatt, S. (2009). Multimodal interfaces : A survey of principles, models and frameworks. *Human machine interaction*, pages 3–26.

[EASA,] EASA. Cs-25 large aeroplanes. https://www.easa.europa.eu/document-library/certification-specifications/cs-25-0.

[Fahssi et al., 2014] Fahssi, R., Martinie, C., and Palanque, P. (2014). Hamsters : un environnement d'édition et de simulation de modèles de tâches. In *IHM'14, 26e conférence francophone sur l'Interaction Homme-Machine*, pages 5–6.

[Fayollas, 2015] Fayollas, C. (2015). Architecture logicielle générique et approche à base de modèles pour la sûreté de fonctionnement des systèmes interactifs critiques.

[Fayollas et al., 2014a] Fayollas, C., Fabre, J., Palanque, P., Cronel, M., Navarre, D., and Deleris, Y. (2014a). A software-implemented fault-tolerance approach for control and display systems in avionics. In *Proceedings of IEEE Pacific Rim International Symposium on Dependable Computing, PRDC*.

[Fayollas et al., 2016] Fayollas, C., Fabre, J.-C., Palanque, P., Cronel, M., Navarre, D., and Deleris, Y. (2016). A three-fold approach towards increased assurance levels for interactive systems : A flight control unit case study. In *Proceedings of the International Conference on Human-Computer Interaction in Aerospace*, HCI-Aero '16, pages 2 :1–2 :9, New York, NY, USA. ACM.

[Fayollas et al., 2014b] Fayollas, C., Palanque, P., Fabre, J.-C., Navarre, D., Barboni, E., Cronel, M., and Deleris, Y. (2014b). Resilience of interactive systems : Contribution by a fault-tolerant architecture | Résilience des systèmes interactifs : Contribution par une architecture tolérante aux fautes. In *IHM 2014 - Actes de la 26ieme Conference Francophone sur l'Interaction Homme-Machine*.

[Feiler et al., 2006] Feiler, P. H., Gluch, D. P., and Hudak, J. J. (2006). The architecture analysis & design language (aadl) : An introduction. Technical report, DTIC Document.

[Forsberg and Mooz, 1991] Forsberg, K. and Mooz, H. (1991). The relationship of system engineering to the project cycle. In *INCOSE International Symposium*, volume 1, pages 57–65. Wiley Online Library.

[Genrich, 1991] Genrich, H. J. (1991). Predicate/transition nets. In *High-level Petri Nets*, pages 3–43. Springer.

[Ghezzi et al., 1989] Ghezzi, C., Mandrioli, D., Morasca, S., and Pezze, M. (1989). *A general way to put time in Petri nets*, volume 14. ACM.

[Gladden, 1982] Gladden, G. (1982). Stop the life-cycle, i want to get off. *ACM SIGSOFT Software Engineering Notes*, 7(2) :35–39.

[Goldberg and Robson, 1983] Goldberg, A. and Robson, D. (1983). *Smalltalk-80 : the language and its implementation*. Addison-Wesley Longman Publishing Co., Inc.

[Göransson et al., 2004] Göransson, B., Gulliksen, J., and Boivie, I. (2004). The usability design process–integrating user-centered systems design in the software development process research section.

[Hamon et al., 2014a] Hamon, A., Palanque, P., André, R., Barboni, E., Cronel, M., and Navarre, D. (2014a). Multi-touch interactions for control and display in interactive cockpits : Issues and a proposal. In *Proceedings of the International Conference on Human-Computer Interaction in Aerospace*, HCI-Aero '14, pages 7 :1–7 :10, New York, NY, USA. ACM.

[Hamon et al., 2015] Hamon, A., Palanque, P., and Cronel, M. (2015). Dependable multi-touch interactions in safety critical industrial contexts : Application to aeronautics. In *2015 IEEE 13th International Conference on Industrial Informatics (INDIN)*, pages 980–987.

[Hamon et al., 2014b] Hamon, A., Palanque, P., Cronel, M., André, R., Barboni, E., and Navarre, D. (2014b). Formal modelling of dynamic instantiation of input devices and interaction techniques : application to multi-touch interactions. In *Proceedings of the 2014 ACM SIGCHI symposium on Engineering interactive computing systems*, pages 173–178. ACM.

[Hamon-Keromen, 2014] Hamon-Keromen, A. (2014). *Définition d'un langage et d'une méthode pour la description et la spécification d'IHM post-WIMP pour les cockpits interactifs*. PhD thesis, Université de Toulouse, Université Toulouse III-Paul Sabatier.

[Hart and Staveland, 1988] Hart, S. G. and Staveland, L. E. (1988). Development of nasa-tlx (task load index) : Results of empirical and theoretical research. *Advances in psychology*, 52 :139–183.

[Hartson and Hix, 1989] Hartson, H. R. and Hix, D. (1989). Toward empirically derived methodologies and tools for human-computer interface development. *International Journal of Man-Machine Studies*, 31(4) :477–494.

[Hoste et al., 2011] Hoste, L., Dumas, B., and Signer, B. (2011). Mudra : a unified multimodal interaction framework. In *Proceedings of the 13th international conference on multimodal interfaces*, pages 97–104. ACM.

[ISO, a] ISO, O. i. d. n. Ergonomie de l'interaction homme-système. https://www.iso.org/fr/search/x/query/9241.

[ISO, b] ISO, O. i. d. n. Human-centred design processes for interactive systems. https://www.iso.org/standard/21197.html.

[ISO, c] ISO, O. i. d. n. Road vehicles – functional safety – part 1 : Vocabulary. https://www.iso.org/standard/43464.html.

[ISO, 1997] ISO, T. (1997). 18529 (2000). *Human-centred lifecycle process descriptions*.

[Johnson et al., 1995] Johnson, C., McCarthy, J., and Wright, P. (1995). Using a formal language to support natural language in accident reports. *Ergonomics*, 38(6) :1264–1282.

[Kazman and Bass, 1994] Kazman, R. and Bass, L. (1994). Toward deriving software architectures from quality attributes. Technical report, DTIC Document.

[Kin et al., 2011] Kin, K., Miller, T., Bollensdorff, B., DeRose, T., Hartmann, B., and Agrawala, M. (2011). Eden : a professional multitouch tool for constructing virtual organic environments. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 1343–1352. ACM.

[Koopman, 2014] Koopman, P. (2014). Software quality, dependability and safety in embedded systems (invited talk). https://users.ece.cmu.edu/~koopman/pubs/koopman14_safecomp_keynote.pdf.

[Kruchten, 2004] Kruchten, P. (2004). *The rational unified process : an introduction*. Addison-Wesley Professional.

[Ladry, 2010] Ladry, J.-F. (2010). *Une notion et un processus outillé pour le développement de systèmes interactifs multimodaux critiques*. PhD thesis, Université de Toulouse, Université Toulouse III-Paul Sabatier.

[Lalanne et al., 2009] Lalanne, D., Nigay, L., Robinson, P., Vanderdonckt, J., Ladry, J.-F., et al. (2009). Fusion engines for multimodal input : a survey. In *Proceedings of the 2009 international conference on Multimodal interfaces*, pages 153–160. ACM.

[Laprie et al., 1995] Laprie, J.-C., Arlat, J., Blanquart, J., Costes, A., Crouzet, Y., Deswarte, Y., Fabre, J., Guillermain, H., Kaâniche, M., Kanoun, K., et al. (1995). Guide de la sûreté de fonctionnement. *Toulouse : Cépaduès*.

[Lawson et al., 2009] Lawson, J.-Y. L., Al-Akkad, A.-A., Vanderdonckt, J., and Macq, B. (2009). An open source workbench for prototyping multimodal interactions based on off-the-shelf heterogeneous components. In *Proceedings of the 1st ACM SIGCHI Symposium on Engineering Interactive Computing Systems*, EICS '09, pages 245–254, New York, NY, USA. ACM.

[Martin and Beroule, 1993] Martin, J.-C. and Beroule, D. (1993). Types et buts de coopérations entre modalités. *Cinquièmes Journées sur l'Ingénierie des Interfaces Homme-Machine Lyon, France*, pages 17–22.

[Martinie et al., 2015] Martinie, C., Navarre, D., Palanque, P., and Fayollas, C. (2015). A generic tool-supported framework for coupling task models and interactive applications. In *Proceedings of the 7th ACM SIGCHI Symposium on Engineering Interactive Computing Systems*, pages 244–253. ACM.

[Martinie et al., 2012] Martinie, C., Palanque, P., Navarre, D., and Barboni, E. (2012). A development process for usable large scale interactive critical systems : application to satellite ground segments. In *International Conference on Human-Centred Software Engineering*, pages 72–93. Springer.

[Martinie et al., 2010] Martinie, C., Palanque, P., Winckler, M., and Conversy, S. (2010). Dreamer : a design rationale environment for argumentation, modeling and engineering requirements. In *Proceedings of the 28th ACM International Conference on Design of Communication*, pages 73–80. ACM.

[McCracken and Jackson, 1982] McCracken, D. D. and Jackson, M. A. (1982). Life cycle concept considered harmful. *ACM SIGSOFT Software Engineering Notes*, 7(2) :29–32.

[Miller et al., 2006] Miller, S. P., Tribble, A. C., Whalen, M. W., and Heimdahl, M. P. (2006). Proving the shalls. *International Journal on Software Tools for Technology Transfer (STTT)*, 8(4) :303–319.

[Mirlacher et al., 2012] Mirlacher, T., Palanque, P., and Bernhaupt, R. (2012). Engineering animations in user interfaces. In *Proceedings of the 4th ACM SIGCHI symposium on Engineering interactive computing systems*, pages 111–120. ACM.

[MMI, 2005] MMI (2005). Mmi-arch on wikipedia. https://fr.wikipedia.org/wiki/Multimodal_Architecture_and_Interfaces.

[Navarre, 2001] Navarre, D. (2001). *Une technique de description formelle et un environnement pour une modélisation et une exploitation synergiques des tâches et du système*. PhD thesis, Ph. D. dissertation, Université Toulouse I.

[Navarre et al., 2009] Navarre, D., Palanque, P., Ladry, J.-F., and Barboni, E. (2009). Icos : A model-based user interface description technique dedicated to interactive systems addressing usability, reliability and scalability. *ACM Transactions on Computer-Human Interaction (TOCHI)*, 16(4) :18.

[Navarre et al., 2001] Navarre, D., Palanque, P., Paternò, F., Santoro, C., and Bastide, R. (2001). A tool suite for integrating task and system models through scenarios. In *International Workshop on Design, Specification, and Verification of Interactive Systems*, pages 88–113. Springer.

[Nielsen, 1994] Nielsen, J. (1994). Usability inspection methods. In *Conference companion on Human factors in computing systems*, pages 413–414. ACM.

[Nielsen, 2000] Nielsen, J. (2000). Why you only need to test with 5 users. https://www.nngroup.com/articles/why-you-only-need-to-test-with-5-users/.

[Nielsen, 2009] Nielsen, J. (2009). Discount usability : 20 years. *Jakob Nielsen's Alertbox Available at http ://www. useit. com/alertbox/discount-usability. html [Accessed 23 January 2012]*.

[Nigay and Coutaz, 1993] Nigay, L. and Coutaz, J. (1993). A design space for multimodal systems : concurrent processing and data fusion. In *Proceedings of the INTERACT'93 and CHI'93 conference on Human factors in computing systems*, pages 172–178. ACM.

[Nigay and Coutaz, 1996] Nigay, L. and Coutaz, J. (1996). Espaces conceptuels pour l'interaction multimédia et multimodale. *Technique et science informatiques*, 15(9) :1195–1225.

[Norman, 1983] Norman, D. A. (1983). Design rules based on analyses of human error. *Communications of the ACM*, 26(4) :254–258.

[Norman, 1988] Norman, D. A. (1988). The design of everyday things, the action theory.

[Oviatt, 1999] Oviatt, S. (1999). Ten myths of multimodal interaction. *Communications of the ACM*, 42(11) :74–81.

[Palanque, 1992] Palanque, P. (1992). *Modélisation par Objets Coopératifs Interactifs d'interfaces homme-machine dirigées par l'utilisateur*. PhD thesis, Université Paul Sabatier Toulouse.

[Palanque et al., 2011] Palanque, P., Barboni, E., Martinie, C., Navarre, D., and Winckler, M. (2011). A model-based approach for supporting engineering usability evaluation of interaction techniques. In *Proceedings of the 3rd ACM SIGCHI symposium on Engineering interactive computing systems*, pages 21–30. ACM.

[Palanque and Bastide, 1994] Palanque, P. and Bastide, R. (1994). A formalism for reliable user interfaces.

[Palanque et al., 2009] Palanque, P., Ladry, J.-F., Navarre, D., and Barboni, E. (2009). High-fidelity prototyping of interactive systems can be formal too. *Human-Computer Interaction. New Trends*, pages 667–676.

[Palmer, 1995] Palmer, E. (1995). Oops, it didn't arm'- a case study of two automation surprises. In *International Symposium on Aviation Psychology, 8 th, Columbus, OH*, pages 227–232.

[Parasuraman and Riley, 1997] Parasuraman, R. and Riley, V. (1997). Humans and automation : Use, misuse, disuse, abuse. *Human Factors*, 39(2) :230–253.

[Pascoal et al., 2008] Pascoal, E., Rufino, J., Schoofs, T., and Windsor, J. (2008). Amobaarinc 653 simulator for modular based space applications. *emergency*, 10 :2.

[Pfaff and ten Hagen, 1985] Pfaff, G. and ten Hagen, P. (1985). User interface management systems : proceedings of the workshop on user interface management systems, held in seeheim frg.

[Pirker and Bernhaupt, 2011] Pirker, M. M. and Bernhaupt, R. (2011). Measuring user experience in the living room : results from an ethnographically oriented field study indicating major evaluation factors. In *Proceedings of the 9th European Conference on Interactive TV and Video*, pages 79–82. ACM.

[Prisaznuk, 2014] Prisaznuk, P. J. (2014). Arinc specification 653, avionics application software standard interface. In *Digital Avionics Handbook, Third Edition*, pages 625–632. CRC Press.

[Rauterberg, 1992] Rauterberg, M. (1992). An iterative-cyclic software process model. In *Software Engineering and Knowledge Engineering, 1992. Proceedings., Fourth International Conference on*, pages 600–607. IEEE.

[Royce, 1987] Royce, W. W. (1987). Managing the development of large software systems : concepts and techniques. In *Proceedings of the 9th international conference on Software Engineering*, pages 328–338. IEEE Computer Society Press.

[RTCA, a] RTCA, I. Do-178c, software considerations in airborne systems and equipment certification. https://my.rtca.org/NC__Product?id=a1B36000001IcmqEAC.

[RTCA, b] RTCA, I. Do-330 - software tool qualification considerations. https://my.rtca.org/NC__Product?id=a1B36000001IcfkEAC.

[RTCA, c] RTCA, I. Do-331 - model based development and verification supplement to do-178c and do-278a. https://my.rtca.org/NC__Product?id=a1B36000001IcfiEAC.

[RTCA, d] RTCA, I. Do-332 - object oriented technology and related techniques supplement to do-178c and do-278a. https://my.rtca.org/NC__Product?id=a1B36000001IcfgEAC.

[RTCA, e] RTCA, I. Do-333 - formal methods supplement to do-178c and do-278a. https://my.rtca.org/NC__Product?id=a1B36000001IcfeEAC.

[Sasse et al., 2001] Sasse, M. A., Brostoff, S., and Weirich, D. (2001). Transforming the 'weakest link'—a human/computer interaction approach to usable and effective security. *BT technology journal*, 19(3) :122–131.

[Schmerler and Rimkus, 2013] Schmerler, S. and Rimkus, R. (2013). Autosar—shaping the future of a global standard. *ATZelektronik worldwide*, 8(1) :42–45.

[Schoofs et al., 2009] Schoofs, T., Santos, S., Tatibana, C., and Anjos, J. (2009). An integrated modular avionics development environment. In *Digital Avionics Systems Conference, 2009. DASC'09. IEEE/AIAA 28th*, pages 1–A. IEEE.

[smartkom,] smartkom. General architecture of smartkom. http://www.smartkom.org/start_en.html.

[Sommerville, 2011] Sommerville, I. (2011). *Software engineering*. Pearson.

[Storey, 1996] Storey, N. R. (1996). *Safety critical computer systems*. Addison-Wesley Longman Publishing Co., Inc.

[Tankeu Choitat, 2011] Tankeu Choitat, A. (2011). *Approches outillées pour le développement des systèmes interactifs intégrant les aspects sûreté de fonctionnement et utilisabilité*. PhD thesis, Université de Toulouse, Université Toulouse III-Paul Sabatier.

[Thimbleby, 2007] Thimbleby, H. (2007). User-centered methods are insufficient for safety critical systems. In *Symposium of the Austrian HCI and Usability Engineering Group*, pages 1–20. Springer.

[Traverse et al., 2004] Traverse, P., Lacaze, I., and Souyris, J. (2004). Airbus fly-by-wire : A total approach to dependability. In *Building the Information Society*, pages 191–212. Springer.

[Van Dam, 1997] Van Dam, A. (1997). Post-wimp user interfaces. *Communications of the ACM*, 40(2) :63–67.

[Vermeulen et al., 2013] Vermeulen, J., Luyten, K., van den Hoven, E., and Coninx, K. (2013). Crossing the bridge over norman's gulf of execution : revealing feedforward's true identity. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 1931–1940. ACM.

[Vernier, 2001] Vernier, F. (2001). *La multimodalité en sortie et son application à la visualisation de grandes quantités d'information*. PhD thesis.

[Wahlster, 2006] Wahlster, W. (2006). *SmartKom : foundations of multimodal dialogue systems*, volume 12. Springer.

[Weiser, 1993] Weiser, M. (1993). Some computer science issues in ubiquitous computing. *Communications of the ACM*, 36(7) :75–84.

[wikimedia commons,] wikimedia commons. File :a-380 cockpit.jpg from wikimedia commons, the free media repository. https://commons.wikimedia.org/wiki/File:A-380_Cockpit.jpg. accès en janvier 2017.

[Woods, 1985] Woods, D. D. (1985). Cognitive technologies : The design of joint human-machine cognitive systems. *AI magazine*, 6(4) :86.

[Yeh, 1996] Yeh, Y. C. (1996). Triple-triple redundant 777 primary flight computer. In *Aerospace Applications Conference, 1996. Proceedings., 1996 IEEE*, volume 1, pages 293–307. IEEE.

[Ziegler et al., 1996] Ziegler, J. F., Curtis, H. W., Muhlfeld, H. P., Montrose, C. J., Chin, B., Nicewicz, M., Russell, C., Wang, W. Y., Freeman, L. B., Hosier, P., et al. (1996). Ibm experiments in soft fails in computer electronics (1978–1994). *IBM journal of research and development*, 40(1) :3–18.

Chapitre

9 Annexes

9.1 Description des fonctions de picking, extrait de [Hamon-Keromen, 2014]

Le modèle de picking propose deux services : *addWidget* et *pickAt*. Le service *addWidget* permet d'ajouter des widgets au modèle du picking. Lors de l'invocation de ce service, un jeton contenant une référence vers le widget à ajouter. La transition *t1* est alors franchissable. Son franchissement retire le jeton de la place *nbWidget* et celui de la place *SIP_addWidget* ; dépose un jeton dans les places *WidgetPool* et *nbwidget*. Le jeton de la place contient un entier qui est incrémenté à chaque invocation du service. Il permet d'ordonner les widgets sur le plan vertical. Le jeton de la place *WidgetPool* contient une référence vers le nouveau widget ainsi que son ordre vertical. Dans notre modèle, le dernier arrivé est en dessous des autres. La fonction de picking est réalisée lorsque le service *pickAt* est invoqué sur un modèle de picking.

Lors de l'invocation du sercice *pickAt*, un jeton contenant la position absolue du point est déposé dans la place *SIP_pickAt*. La transition *startPickAt* est alors franchissable. Son franchissement dépose un jeton dans les places *trigger*, *picking*, *Unstacking* et *pickingInProgress*. La transition *down_1* est alors franchissable. Son franchissement retire le jeton de la place *trigger*, instancie un entier z à 0et dépose cet entier dans la place Z et la position à tester dans la place *newPoint*. Deux cas sont alors possibles :

1. Il reste au moins un widget à tester. Le jeton correspondant est retiré de la place *WidgetPool* lors du franchissement de la transition *resultTemp* qui invoque le service *contains* sur ce widget. Le résultat, et toutes les variables d'entrées sont déposés dans un jeton dans la place *tempRes0*. Le franchissement de la transition *getIsContainer* invoque le service *isContainer* et dépose le résultat, ainsi de les autres variables dans un jeton dans la place *tempResult*. Trois cas sont alors possibles :
   (a) Le widget ne contient pas le point : la transition *noHit* est franchie. Son franchissement retire le jeton de la place *tempResult* , incrémente l'index du widget à tester et dépose un jeton dans les places Z (index du prochain widget à tester), *newPoint* (la position), *stack* (le widget testé) et *stackCount* . Le prochain widget peut alors être testé.
   (b) Le widget contient le point et n'est pas un container : la transition *hit* est

franchie. Son franchissement retire le jeton de la place *tempResult* et *Unstacking* et peuple les places de sorties.

(c) Le widget contient le point et est un *container* : la transition *hitContainer* devient franchissable. Son franchissement retire le jeton de la place *tempResult* et *Unstacking* . Elle invoque récursivement le service *pickAt* sur le container et dépose le résulat dans la place *StartRecursivePicking*. Deux cas sont alors possibles :

i. Le résultat est *null* et le modèle de picking va renvoyer le container. La transition *null* est alors franchie.

ii. Le widget est non *null*. La transition *neededToInverse* est alors franchie. Elle retire le jeton de la place *StartRecursivePicking* et peuple les places de sortie.

2. Tous les widgets ont été testé. La transition *endPicking* est alors franchie. Elle retire les jetons des places *Z*,*newPoint* et *Unstacking*. Elle instancie un widget *null* et le dépose dans la place *DONE*.

Une fois le résultat du picking dans la place *DONE*, trois cas sont alors possibles :

1. Le widget trouvé est au dessus de tous les autres (selon l'axe Z). La transition *unStackPicked* est alors franchissable.
2. Il y a au moins un widget qui est tracé au dessus de celui trouvé pour le picking. La transition *unStackNoPick2* est alors franchissable pour chaque widget précédenmment testé sans succès. Leur ordre vertical est incrémenté (ils reculent d'un cran vers le fond) et sont déposés dans la place *WidgetPool*.
3. Aucun widget n'a été trouvé. La transition *unstackNoPick* devient franchissable. Elle permet de remettre tous les widgets dans la place *WidgetPool*. La place *stackCount* est alors vide, ce qui rend la transition *unStackPicked* franchissable. Le franchissement de la transition *unStackPicked* permet de replacer le widget trouvé lors du picking et de la placer au dessus des autres puis replacer le jeton correspondant dans la place *WidgetPool* ce qui rend la transition *endUnstack* franchissable.

Le franchissement de la transition end Unstack retire le jeton de la place *DONE* ce qui rend la transition *IN* franchissable. Son franchissement retire les jetons des places *pickingInProgress*, *picking* et *PickingResult* ; dépose un jeton avec le résultat dans la place *SOP_pickAT*, ce qui clot l'invocation du service

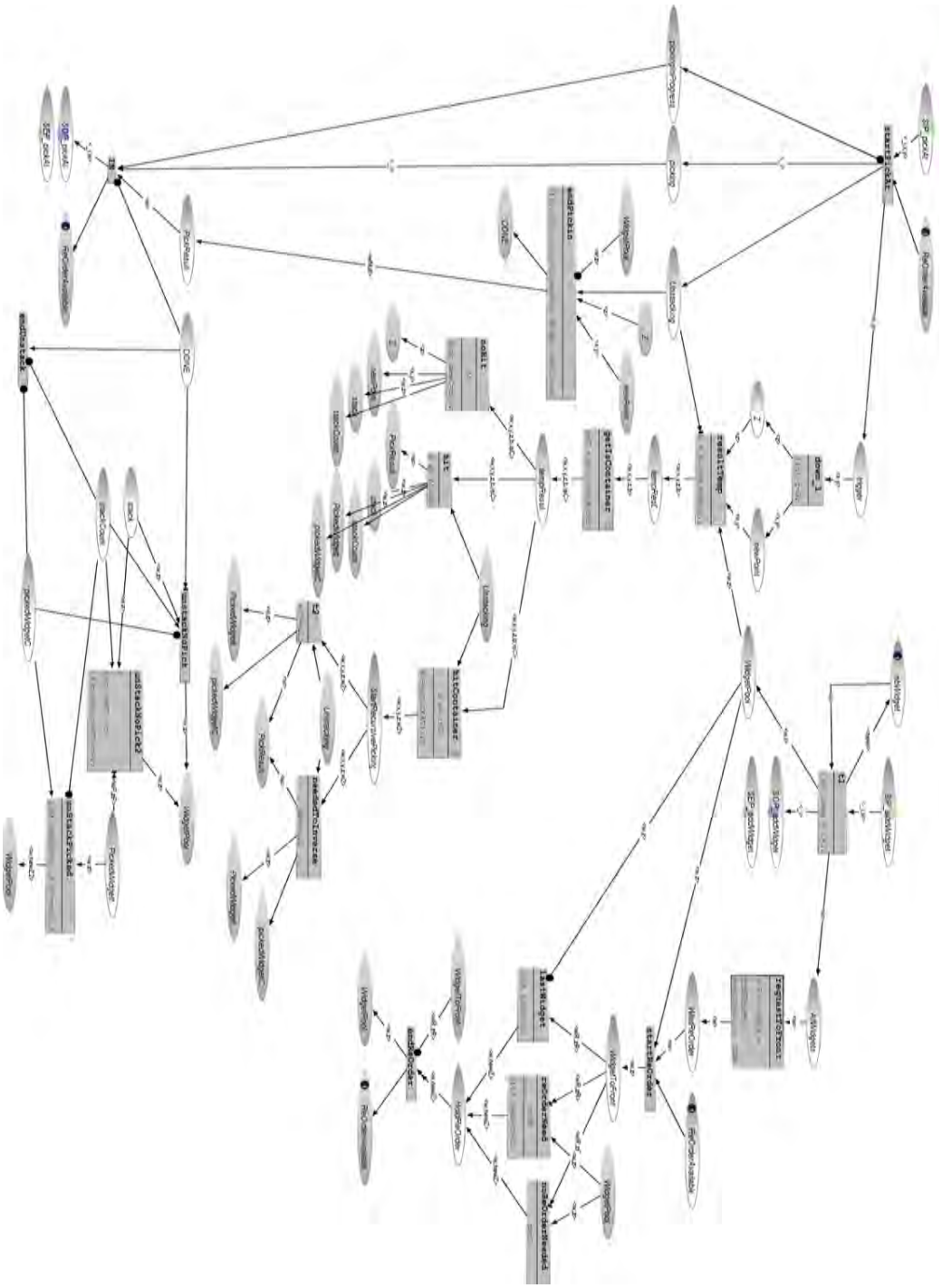


FIGURE 9.1 – Une description de la fonction de Picking

9.2 Modèles d'initialisation de l'application quatre saisons multi événements

La figure 9.2, présente le modèle permettant le lancement de l'application. Son rôle est de créer une fenêtre Javafx, de créer le container principal et sa fonction de picking, et de créer le gestionnaire de souris. Ce modèle n'est plus actif une fois l'application lancée.

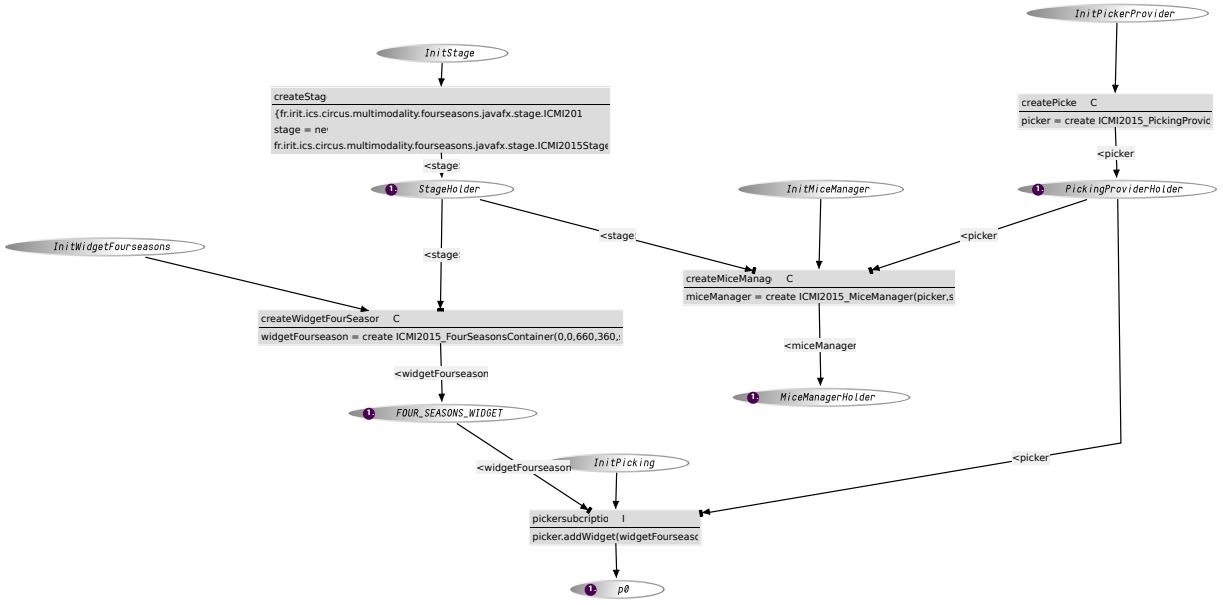


FIGURE 9.2 – Modèle de lancement de l'application

La figure 9.3 présente le *container* principal de l'application. C'est un *widget* un peu particulier puisqu'il contient tous les autres. Sa fonction de picking est la première à être appelé dans la logique récursive de notre approche. La partie haute du modèle correspond à tous les services implémentables pour un *widget* de cette application, la partie basse décrit l'initialisation des différents modèles de widgets dont les 4 boutons, ainsi que leur fonction de picking.

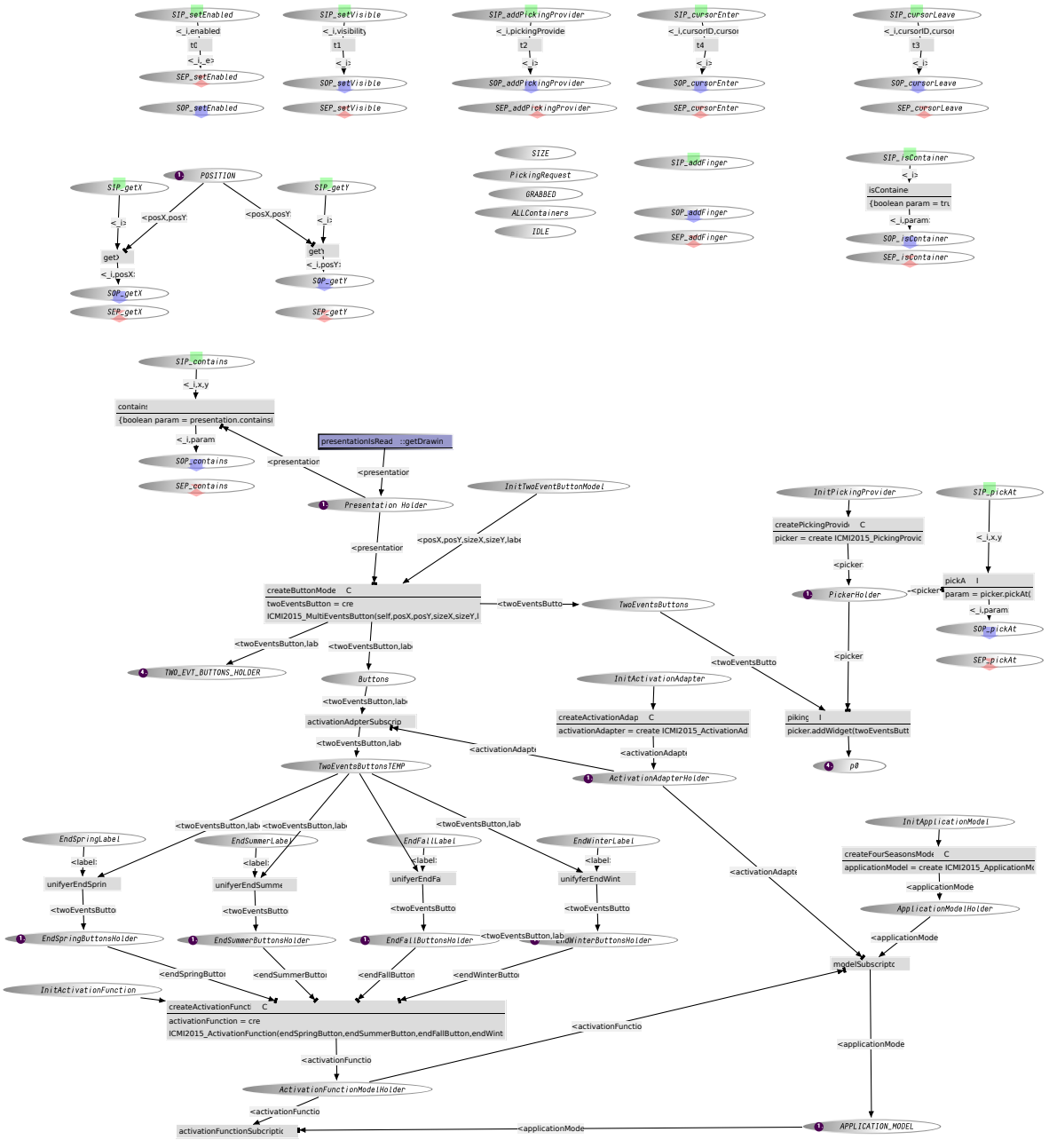


FIGURE 9.3 – Modèle du container principal de l'application