Projet VERBATIM

Sous-projet 2 – Lot 2

Test d'applications multimodales à l'aide de l'approche synchrone :
Fournitures :
«Vers une plate-forme de validation d'applications multimodales combinant Lutess et ICARE»

Version : 1.0

Date : décembre 2006

Frédéric Jourde, Laurence Nigay, Ioannis Parissis

Laboratoires LSR-IMAG et CLIPS-IMAG

BP 53 38041 Grenoble Cedex 9

www-lsr.imag.fr www-clips.imag.fr

LSR

Table des matières

Introduction

Le travail présenté dans ce document est en très grande partie issu du projet de Master Recherche effectué par Frédéric Jourde, dirigé par Laurence Nigay et Ioannis Parissis. Ce projet a eu comme objectif de proposer une plate-forme de validation basée sur l'outil de test de logiciels synchrones Lutess et l'environnement de développement d'applications multimodales ICARE.

Une application multimodale repose sur la disponibilité de multiples techniques d'interaction ou modalités basées sur des mécanismes sophistiqués de reconnaissance et de synthèse : parole, vision par ordinateur, génération automatique de textes et d'images réalistes, vidéo, etc. Ces modalités d'interaction peuvent être combinées par l'utilisateur ou le système (par exemple la combinaison de la parole avec un geste de désignation) en fonction de la tâche à réaliser, du contexte et des préférences utilisateur. La multimodalité permet de rendre l'interaction plus efficace, plus flexible ou plus robuste. Néanmoins la multiplicité des possibilités de la multimodalité complexifie la conception et l'implémentation des applications interactives. De même, elle rend difficile et coûteuse leur vérification et leur évaluation. Dans ce cadre, nous proposons une approche de test formel tirant partie de l'architecture produite par la conception de logiciel à base de moteur de fusion d'événements multimodaux ICARE [Bouchet 04], et d'un logiciel de vérification de système réactif synchrone : Lutess [Parissis 96].

Notre objectif n'est pas de supprimer l'évaluation expérimentale de l'application interactive par une approche de tests formels mais au contraire de l'enrichir par une approche dite prédictive sans l'intervention d'utilisateurs. En effet, en Interaction Homme-Machine (IHM), il est classique de distinguer les méthodes d'évaluation expérimentales de celles prédictives. Aussi notées respectivement méthodes empiriques et analytiques, elles se distinguent par la présence ou non des utilisateurs finaux. Les méthodes expérimentales (ou empiriques) reposent sur le recueil de données comportementales d'utilisateurs représentatifs mis en situation tandis que les méthodes prédictives (ou analytiques) ne nécessitent pas la présence de l'utilisateur final. Ces méthodes permettent, à partir d'une description du système et de l'utilisateur, d'identifier des problèmes potentiels d'utilisabilité. Les méthodes prédictives et expérimentales se complètent et peuvent se pratiquer tout au long du processus de développement. La pratique itérative "tests-corrections" définit le fondement de l'évaluation "formative" (conception itérative centrée utilisateur) : chaque évaluation fournit de nouveaux enseignements dont l'intégration conduit à une nouvelle version du produit de la conception et du logiciel.

Notre objectif est donc de proposer une méthode d'évaluation prédictive (ou analytique) formelle d'applications interactives multimodales, qui se justifie par une plus grande complexité de conception et de développement liée à la multimodalité.

Motivations

La multimodalité est un axe de recherche en pleine expansion. Nous constatons que les travaux à l'origine de nouvelles modalités d'interaction et formes de multimodalité reposent sur un ensemble de dispositifs physiques dédiés à l'interaction et centrés autour d'un même espace d'interaction : l'ordinateur de bureau. Ainsi, un microphone combiné à une souris, deux souris manipulées simultanément, une caméra couplée à une souris, définissent des formes d'interaction multimodale visant à offrir une interaction plus efficace et plus naturelle avec

l'ordinateur. Or, les formidables progrès accomplis dans la miniaturisation des microprocesseurs et dans les réseaux informatiques sans fil permettent d'envisager que la "boîte grise" du calculateur personnel soit "condamnée à disparaître" ou, du moins, à ne pas être le seul lieu d'interaction entre l'utilisateur et le monde numérique. Ceci s'inscrit dans le mouvement de l'ordinateur pervasif et évanescent. La multiplication des ordinateurs de poche et des assistants personnels et la multiplication des systèmes embarqués dans des objets d'usage courant (automobile, télévision, etc.) constituent des témoins de cet essor. Nous retenons de cet essor que l'espace d'interaction devient plus vaste. Il comprend l'environnement physique (correspondant à des paradigmes d'interaction récents comme les interfaces tangibles et la réalité augmentée) et ne se limite plus seulement à un ordinateur sur un bureau. Par exemple, la situation de mobilité de l'utilisateur impose de concevoir des modalités d'interaction et des combinaisons de modalités qui sortent du cadre classique d'un grand écran, d'un clavier et d'une souris - comme le paradigme des interfaces incarnées, ainsi que de prendre en compte des contextes d'interaction par définition variables (contexte physique et social). L'espace d'interaction sort alors définitivement de l'espace composé par le bureau et l'ordinateur. Ce constat entraîne de nouveaux paramètres à prendre en compte dans l'interaction multimodale et de nombreuses possibilités de modalités d'interaction et formes de multimodalité. La multiplicité des possibilités est grande et tout objet physique peut servir de dispositif d'interaction. Cette multiplicité peut se voir comme un facteur de souplesse, avec comme implication la complexité : la "barre à franchir" pour la conception et la réalisation logicielle a été déplacée vers le haut. La complexification de telles applications interactives rend nécessaire leur développement et leur validation rigoureux. Ces constats motivent notre étude qui vise une méthode d'évaluation prédictive formelle d'applications multimodales.

Pluridisciplinarité des travaux et démarche de travail

L'originalité de l'étude est issue de sa pluridisciplinarité (GL et IHM). Basée sur les compétences complémentaires de deux équipes de l'IMAG (VASCO et IIHM), notre démarche de travail consiste à étudier l'application de techniques de test de logiciels réactifs synchrones développées par l'équipe VASCO à la validation de systèmes interactifs multimodaux, en s'appuyant sur le constat suivant : un système interactif peut, sous certaines conditions, être considéré comme un système réactif synchrone. S'il est vrai que la rapidité de réaction des systèmes interactifs à un événement externe est moins cruciale que dans le cas des systèmes traditionnellement qualifiés de synchrones, Nous pouvons toutefois constater que leurs comportements sont constitués de cycles « action-réaction ». Ainsi, les interactions multimodales correspondant à des comportements des utilisateurs, certaines propriétés de l'application et de son contexte, peuvent s'exprimer dans un formalisme synchrone à des fins de tests. Ce constat est conforté par des travaux sur les interfaces à manipulation directe [Ausbourg 98] qui ont montré que le comportement de l'interface et ses propriétés sont exprimables en Lustre.

Partant de ce constat, notre démarche de travail s'articule en trois étapes :

- Étudier les approches de vérification/validation d'interaction multimodale prédictives ou expérimentales, formelles ou informelles ainsi que les propriétés de la multimodalité à vérifier.
- Étudier l'intérêt et la faisabilité du test formel d'interaction multimodale, au travers du couplage ICARE – Lutess. L'étude de faisabilité a consisté à tester une application multimodale existante : l'application interactive multimodale YellowPage version 1.
- Proposer une méthode semi-automatique, rapide, et efficace pour le test d'interaction multimodale. Notre solution est instrumentée et consiste en une extension de la

plateforme ICARE afin de générer automatiquement le couplage ICARE-Lutess. Notre outil est testé sur une application multimodale que nous avons conçue et développée avec notre outil ICARE étendu : la version 2 de l'application YellowPage.

- Ces travaux de réflexion ont donné lieu à plusieurs réalisations et expérimentations. La première est le couplage manuel sur la première version de YellowPage, la deucième est le développement de la deuxième version de YellowPage, et la troisième est une extension de l'éditeur de schéma ICARE.

Pour le développement de la solution proposée, nous nous sommes appuyés sur les outils disponibles dans chacune des deux équipes : Lutess de l'équipe VASCO et ICARE de l'équipe IIHM. Notre outil qui instrumente notre méthode d'évaluation prédictive formelle n'est donc pas général puisqu'il restreint les applications multimodales à celles développées à partir de l'approche à composants ICARE dédiée à la multimodalité. Notre méthode tire partie de l'architecture logicielle liée à la plateforme ICARE. De plus nous avons utilisé l'environnement LUTESS pour le test automatique de programmes au comportement synchrone qui n'ont pas nécessairement été développés au moyen de spécifications formelles. Les spécifications formelles sont uniquement nécessaires à la définition de l'environnement externe du programme à tester ainsi que de certaines de ses propriétés.

La généralisation de nos résultats (par l'utilisation de d'autres architectures logicielles ou méthodes de tests) constitue une perspective intéressante à nos travaux.

Structure du document

L'organisation du mémoire reflète notre démarche d'analyse, de l'espace des possibilités aux solutions. La première partie composée de quatre chapitres principaux décrit l'existant en considérant d'abord l'interaction monomodale puis l'interaction multimodale. Cette partie est conclue par un chapitre de synthèse. Les possibilités étant cernées, la deuxième partie du mémoire est composée de quatre chapitres dédiés à notre méthode d'évaluation prédictive et formelle instrumentée par un outil résultant du couplage ICARE-Lutess.

La **première partie** de notre mémoire dresse un bilan sur les méthodes de vérification/validation des applications interactives. Tandis que les chapitres 2 et 3 concernent les applications interactives en général, les chapitres 4 et 5 sont dédiés à l'interaction multimodale.

Le chapitre 2 est bref et rappelle les notions de qualité logicielle positionnée au sein d'un cycle de vie d'une application. L'accent est mis sur un des critères de qualité logicielle, l'utilisabilité qui est au centre des méthodes d'évaluation ergonomique et donc de notre étude.

Le chapitre 3 dresse ensuite un bilan sur les approches de vérification/validation d'applications interactives en mettant l'accent sur les approches formelles.

Le cadre général cerné, nous focalisons ensuite sur les applications multimodales. La terminologie adoptée pour la multimodalité fait l'objet du chapitre 4. Ce chapitre a aussi pour objectif de souligner la complexité de conception et de développement d'applications multimodales.

La terminologie figée, le chapitre 5 étudie les méthodes de vérification/validation formelle d'applications multimodales. Les méthodes présentées ont été développées pour la multimodalité.

Nous concluons cette partie par une synthèse de l'existant au chapitre 6 qui motive nos solutions qui font l'objet de la partie 2.

La **deuxième partie** consacrée à nos contributions s'articule en quatre chapitres.

Nous commençons la deuxième partie du mémoire par la description des deux versions de l'application multimodale YellowPage au chapitre 7. Cette application nous permet d'illustrer les outils utilisés et nos contributions des chapitres 8, 9 et 10. Tandis que la version 1 de YellowPage était déjà développée par l'équipe IIHM, la version 2 a été conçue et développée avec notre outil issu du couplage ICARE-Lutess.

Le chapitre 8 est consacré à l'approche ICARE, son modèle conceptuel et son instrumentation par la plateforme ICARE. Nous illustrons l'approche par les deux versions de YellowPage, et nous décrivons dans ce chapitre les composants ICARE conçus et développés pour notre réalisation de la version 2 de YellowPage. Dans ce chapitre, nous mettons l'accent sur l'architecture à composants ICARE sur laquelle nous basons le couplage ICARE-Lutess décrit au chapitre 10.

Le chapitre 9 expose notre méthode de vérification d'applications interactives multimodales au moyen de l'outil de vérification de systèmes réactifs synchrones, Lutess. Pour cela, nous expliquons comment exprimer des propriétés logiques et temporelles à vérifier dans un fichier d'oracle et qui correspondent à des propriétés ergonomiques de la multimodalité. Enfin nous identifions différents niveaux d'abstraction pour les tests qui sont issus de l'architecture logicielle à composants ICARE.

La méthode de vérification exposée, le chapitre 10 est consacré à son instrumentation par le couplage de la plateforme ICARE à celle Lutess. Nous présentons différentes approches de connexion avant de justifier celle retenue. Puis nous décrivons notre étude de faisabilité par une connexion manuelle expérimentée sur la version 1 de YellowPage. Les résultats de tests effectués sont présentés comme résultats positifs de cette étude de faisabilité. Nous détaillons ensuite notre solution pour automatiser le couplage ICARE-Lutess. Notre extension de la plateforme ICARE, comme outil pour notre méthode de vérification d'applications interactives multimodales, est illustrée par la conception, le développement et le test de la version 2 de YellowPage. Les résultats de tests obtenus soulignent d'une part le caractère fonctionnel de l'instrumentation automatique, et d'autre part, son adéquation pour la vérification des propriétés ergonomiques CARE de la multimodalité.

Le chapitre 10 conclut nos travaux en dressant un bilan des contributions. Nous développons ensuite les perspectives de travaux futurs, à court terme comme extensions immédiates de nos travaux et à plus long terme.

Espace Problème

Vérification/Validation d'applications interactives : de l'interaction monomodale à l'interaction multimodale

Qualité Logicielle

Génie logiciel

Depuis quelques années, la qualité est devenue un enjeu de taille pour les activités de production logicielle. De nombreuses études ont été menées afin de définir les caractéristiques de la qualité. C'est le cas du modèle **McCall** [macCall 77] illustré en **Figure 1**. Il identifie trois facteurs essentiels de la qualité logicielle : l'utilisabilité, l'efficacité, et la maintenabilité. Ces trois facteurs sont soumis à la satisfaction de huit critères. L'évaluation du respect de ces critères permet de mesurer la qualité d'une production logicielle selon ce modèle.

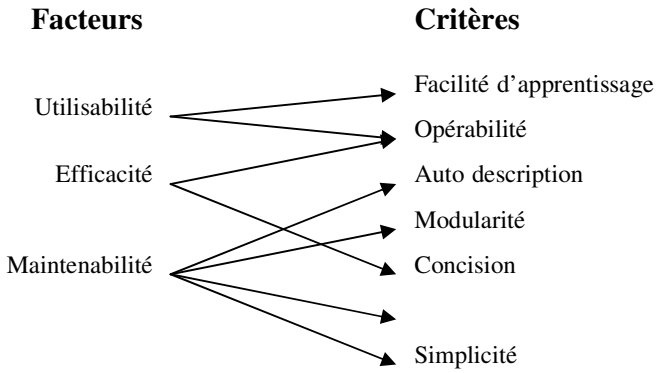

Figure 1 ***: modèle de McCall***

Afin de situer les différentes activités de la production logicielle au cours du temps, citons le modèle en V illustré à la **Figure 2**, qui fait apparaître clairement neuf étapes et lie les activités d'analyse des besoins, de spécification, de conception architecturale et détaillée, respectivement aux activités de validation des besoins, de test de validation, de test d'intégration et de test unitaire. Outre le découpage en activités, ce modèle montre la nécessité d'évaluer la qualité d'un logiciel tout au long de son développement. Le modèle en étoile va plus loin dans le positionnement de l'évaluation par rapport aux autres activités. Il place l'activité d'évaluation au cœur des activités de production logicielle que sont l'analyse des tâches, l'analyse fonctionnelle, la spécification des besoins, la conception, le prototypage et le codage.

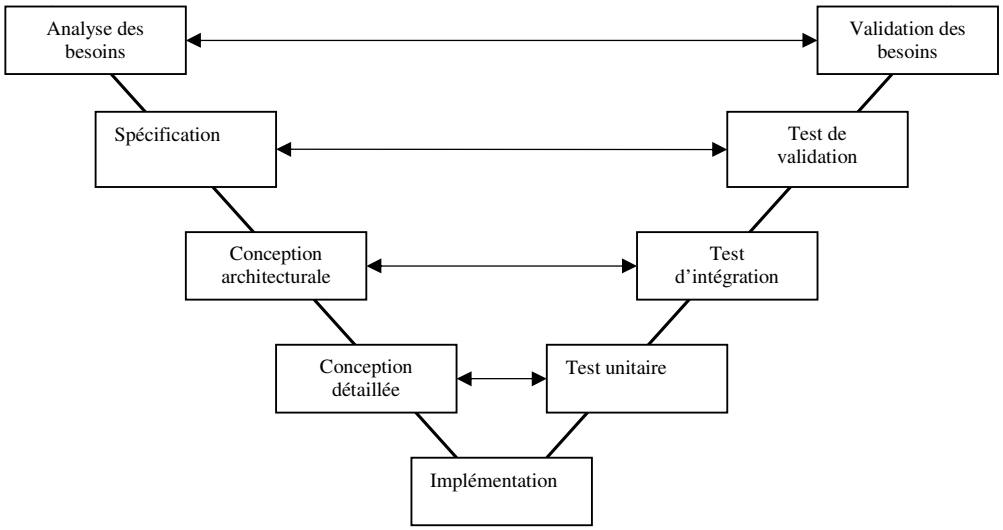

Figure 2 ***: modèle de développement en V***

Utilisabilité

L'évaluation de la qualité logicielle peut s'effectuer au moyen d'outils adaptés. Ainsi, différentes approches ont été proposées, couvrant chacune une part plus ou moins large de cette activité. L'activité d'évaluation nécessite des métriques pour mesurer la qualité du logiciel. Pour ce qui constitue notre préoccupation, c'est-à-dire les logiciels interactifs (avec un utilisateur humain), plusieurs ensembles de recommandations ergonomiques ont été proposés. Les recommandations ergonomiques sont des règles spécifiques, élaborées par des ergonomes, afin de guider la conception d'une interface utilisable. Ces recommandations peuvent s'appliquer à des éléments de l'interface ou à des classes d'interfaces. Il est à noter que les recommandations ergonomiques se trouvent souvent présentées sous des dénominations différentes : recommandations, critères, règles, propriétés, etc. Certaines sont sensées intervenir en amont du processus de conception, tandis que d'autres apparaissent plutôt en phase d'évaluation, en proposant des métriques permettant d'évaluer l'utilisabilité d'un système donné. Mais une grande partie de ces recommandations se retrouvent dans les deux catégories. Devant la multitude de guides, de règles et de recommandations, nous avons choisi d'en présenter ici une partie représentative, émanant d'auteurs dont les travaux sont reconnus par la communauté internationale spécialisée dans ce domaine : Shneiderman, Nielsen, Bastien & Scapin, Coutaz.

Shneiderman [Schneiderman 98] a énoncé un ensemble de huit règles de base applicables à la conception d'une grande partie des systèmes interactifs. Citons par exemple la production de feed-back (retour d'action) : Schneiderman indique que chaque action de l'utilisateur sur le système doit entraîner un feed-back qui montre à cet utilisateur que son action a été prise en compte, tout comme le résultat de cette action. Il indique également dans une autre règle que l'utilisateur doit avoir la sensation de contrôler le système et non l'inverse ; c'est-à-dire que l'utilisateur initie des actions, et doit avoir l'impression que le système réagit à ses actions, et non le contraire.

Nielsen a défini un ensemble des dix heuristiques [Nielsen 94]. Parmi celles-ci, l'heuristique **liberté et contrôle de la part de l'utilisateur** concerne le droit à l'erreur de l'utilisateur et la possibilité de les corriger. En effet, il arrive fréquemment que l'utilisateur sélectionne involontairement une fonction du système. Il faut lui fournir dans ce cas une "porte de sortie" lui permettant de quitter rapidement un état indésirable du système sans avoir à parcourir

toute une série de dialogues. Dans ce but, le système doit être capable de proposer les fonctionnalités de "refaire" (redo) et "défaire" (undo). L'heuristique nommée **visibilité de l'état du système** indique que le système doit en permanence tenir l'utilisateur informé de ce qui est en train de se produire. Ceci peut être obtenu grâce à l'utilisation d'un retour (feed-back) approprié dans un délai de temps raisonnable.

Les critères ergonomiques, selon Scapin et Bastien [Scapin 90], ont pour vocation d'être utilisables par des non-spécialistes de l'utilisabilité, bien qu'à l'origine ils étaient destinés à des fins d'évaluation par les ergonomes. Ils sont basés sur une analyse de l'interface, activité plus rapide et moins dispendieuse que les tests d'utilisabilité. De plus, ils sont suffisamment explicites pour permettre des mesures précises et suffisamment standardisées pour donner des résultats reproductibles. Ils contribuent ainsi à éviter les pièges de la subjectivité et des goûts personnels en donnant un cadre de travail neutre et efficace. Le critère de **guidage** regroupe l'ensemble des moyens mis en œuvre pour conseiller, orienter, informer et conduire l'utilisateur dans son interaction avec la machine. Parmi ces moyens, on peut citer l'incitation, le groupement et la distinction entre items (par localisation ou par format), le feedback immédiat, ou encore la lisibilité. Le **contrôle explicite** concerne la prise en compte par le système des actions explicites des utilisateurs et le contrôle qu'ont les utilisateurs sur le traitement de leurs actions.

Les recommandations ergonomiques ont pour objectif d'améliorer l'utilisabilité du système interactif. Selon Coutaz, l'utilisabilité d'un système interactif est déterminée par deux notions principales : la souplesse et la robustesse de l'interaction [Abowd 92]. Chacune de ces deux notions est définie par un ensemble de propriétés permettant de mesurer le degré d'utilisabilité du système. La **souplesse** de l'interaction représente le degré des possibilités offertes aussi bien à l'utilisateur qu'au système. Elle regroupe plusieurs propriétés, dont l'**atteignabilité**, qui désigne la capacité du système à offrir à l'utilisateur la possibilité de naviguer dans l'ensemble des états observables du système. Formellement, un état q est atteignable à partir d'un état p s'il existe une suite de commandes {ci} qui permettent de passer de l'état p à l'état q. La longueur de la trajectoire d'interaction nécessaire pour passer de l'état p à l'état q offre une métrique permettant de mesurer l'atteignabilité. Les **propriétés CARE**, quant à elles, permettent de caractériser la multimodalité offerte par un système multimodal. Nous renvoyons le lecteur au paragraphe 0 où les propriétés CARE sont décrites en détail. La **robustesse** de l'interaction a pour objectif de prévenir les erreurs et d'augmenter les chances de succès de l'utilisateur. Elle définit un ensemble de propriétés, parmi lesquelles l'**observabilité**, qui représente la capacité du système à rendre son état pertinent perceptible pour l'utilisateur. Par conséquent, elle représente également la capacité pour l'utilisateur d'évaluer l'état du système. La **prévisibilité** implique chez l'utilisateur la capacité de prévoir, pour un état donné, l'effet d'une action. Cette prévisibilité peut être obtenue grâce à la cohérence (conformité aux règles/usages). Il convient cependant de prendre en compte le fait que les règles/usages de l'utilisateur ne sont pas nécessairement celles du concepteur. La cohérence peut en général elle même s'obtenir par conformité à des normes d'IHM et par conformité à l'expérience de l'utilisateur dans le monde réel (analogie, métaphore).

Les recommandations ergonomiques sont des indicateurs de la qualité d'un logiciel. Malheureusement, ces propriétés sont heuristiques et prédictives à l'heure de l'automatisation grandissante du travail. En effet, la revue est le moyen adéquat pour vérifier si les recommandations ont été suivies : cela consiste à mobiliser plusieurs personnes pour passer le logiciel au crible, et ce pour chaque recommandation. Il s'agit d'un travail coûteux, sur les plans financier et humain. Le résultat de l'utilisation d'une telle méthode n'est pas une garantie ni d'utilité, ni d'utilisabilité, ni de succès commercial pour un logiciel. Ainsi, la réduction des coûts de production des logiciels, ainsi que la diminution des risques, sont des

facteurs qui expliquent en partie l'intérêt que suscitent des méthodes formelles de validation, automatiques ou semi-automatiques.

Vérification/Validation d'Applications Interactives

Ce chapitre est consacré à l'activité de vérification/validation d'applications interactives classiques. Nous présentons d'abord les méthodes de vérification/validation existantes, à partir de la taxonomie proposée par Ivory [Ivory 01] qui reprend en partie celle de Balbo [Balbo 95]. Ensuite, nous détaillons les méthodes formelles de vérification/validation automatiques et semi-automatiques, en les classant par rapport à la taxonomie d'Ivory.

Méthodes de vérification/validation

Il existe un grand nombre de méthodes permettant de valider/vérifier l'utilisabilité d'applications interactives. Dans [Ivory 01], l'auteur considère que ces méthodes sont constituées d'une suite d'activités, dont les plus communes sont : la capture, l'analyse, et la critique. L'auteur détaille ces trois activités (ces définitions proviennent d' [Ivory 01]):

- La capture : il s'agit de collecter des données d'utilisabilité, telles que le temps, les erreurs, les violations de recommandations et de notations subjectives.
- L'analyse : il s'agit d'interpréter les données collectées, afin d'identifier des problèmes d'utilisabilité dans l'interface.
- La critique : il s'agit ici de suggérer des solutions ou des améliorations, qui minimisent ou éradiquent les problèmes rencontrés.

Afin de classer ces méthodes, [Balbo 95] propose une taxonomie qui les distingue selon le niveau d'automatisation dans les trois activités. L'auteur propose quatre catégories :

- non – automatique : ce sont les méthodes qui requièrent des compétences de spécialistes.
- capture automatique : Il s'agit de méthodes proposant des instrumentations logicielles capables d'effectuer des enregistrements d'informations.
- analyse automatique : ce sont les méthodes qui proposent des outils capables d'identifier automatiquement des problèmes d'utilisabilité.
- critique automatique : Il s'agit de méthodes capables non seulement d'analyser les problèmes d'utilisabilité, mais de surcroît capables de proposer des améliorations.

Dans [Ivory 01], l'auteur identifie le manque de catégories dans la taxonomie de Balbo, et ainsi, la difficulté de classer les différentes méthodes. Elle propose une nouvelle classification des méthodes d'évaluation selon une taxonomie possédant quatre dimensions, illustrées à la Figure 3:

- La classe de la méthode : axe qui décrit le type d'évaluation.
- Le type de la méthode : axe qui décrit comment est réalisée la méthode.
- Le niveau d'automatisation : axe qui décrit les activités d'évaluations automatisées.
- Le niveau d'effort : axe qui décrit le type des efforts nécessaires pour utiliser la méthode.

Pour chacune de ces quatre dimensions, l'auteur propose plusieurs niveaux, afin de classer correctement les méthodes d'évaluation d'utilisabilité. Je reprends ici en détails ces

différentes catégories, afin de pouvoir positionner les méthodes présentées dans l'espace problème, ainsi que notre propre méthode d'évaluation. L'auteur classe les méthodes d'évaluation d'utilisabilité selon cinq classes :

- Le test : un évaluateur observe un utilisateur interagir avec l'interface, afin de déterminer les problèmes d'utilisabilité.
- L'inspection : un évaluateur utilise un ensemble de critères et d'heuristiques, comme celle présentée en 0, pour identifier les problèmes potentiels d'utilisabilité.
- L'interrogatoire/ l'entretien : l'utilisateur émet des retours sur l'interface par le biais d'entretiens et de surveillances.
- L'analyse de modèles : un évaluateur utilise des utilisateurs et des modèles de l'interface pour générer des prédictions d'utilisabilité.
- La simulation : un évaluateur utilise des utilisateurs et des modèles d'interface pour imiter le comportement interactif des utilisateurs avec une interface, et obtient ainsi des observations sur l'interaction.

L'auteur laisse le champ libre en ce qui concerne le type de méthode. En effet, il est possible de créer de nouvelle manière d'effectuer les classes de méthodes. Dans la classification de travaux que l'auteur propose, nous retrouvons des types comme : l'analyse de fichiers de log, la mesure de performance, ou encore la situation de test où l'utilisateur est seul face à l'interface, isolé du testeur.

Le niveau d'automatisation reprend la taxonomie de Balbo pour définir les activités d'évaluation d'utilisabilité qui sont automatisées. Ce qui donne quatre niveaux : aucune automatisation, automatisation de la capture de données d'utilisabilité, automatisation de l'analyse de ces données, et automatisation de la critique, c'est-à-dire par la suggestion d'améliorations.

Le niveau d'effort indique la quantité de travail que les humains doivent produire pour utiliser la méthode. L'auteur identifie quatre niveaux pertinents d'effort à produire:

- L'effort minimal : c'est-à-dire que la méthode ne requiert pas l'utilisation d'interface, ou de modèle.
- Le développement de modèle : l'évaluateur doit développer un modèle de l'interface, et/ou de l'utilisateur, afin d'utiliser la méthode.
- L'utilisation informelle : il s'agit de laisser à un utilisateur le soin d'interagir librement avec l'interface.
- L'utilisation formelle : cela consiste à faire effectuer à l'utilisateur des tâches précises sur l'interface.

L'auteur propose une classification de nombreuses méthodes grâce à sa taxonomie. Nous ne revenons pas sur cette classification. Nous laissons au lecteur le soin de compléter ses connaissances si besoin avec [Ivory 01].

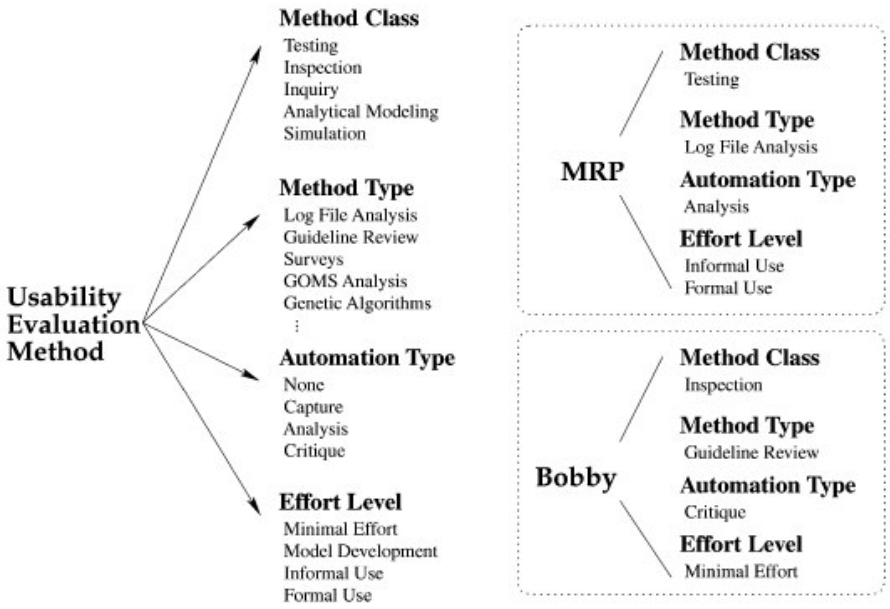

Figure 3 : ***synthèse de la taxonomie pour la classification des méthodes d'évaluation de l'utilisabilité [Ivory 01]***

Méthodes formelles de vérification

Dans ce paragraphe nous présentons un ensemble représentatif de méthodes formelles d'évaluation d'Interface Homme-Machine et nous les positionnons selon la taxonomie de [Ivory 01] présentée précédemment. Nous présentons quatre approches : la première est basée sur l'utilisation de la méthode formelle B d'une part pour valider la décomposition du modèle des tâches d'une application interactive [Aït-Ameur 03] et d'autre part pour développer une application de la spécification jusqu'à l'implémentation avec la garantie de propriété [Jambon 02]. La deuxième consiste en l'utilisation des IOGs (Interaction Object Graphs) pour décrire une application interactive et pour vérifier des propriétés dynamiques sur cette description [Carr 97]. La troisième prône l'utilisation des VEG (Visual Events Grammars) pour concevoir une application interactive et la vérifier au moyen d'un outil de model-checking [Berstel 05]. Enfin, nous présentons une approche formelle utilisant des descriptions LUSTRE et le model-checking [Ausbourg 98].

Méthode formelle B

Dans [Aït-Ameur 03], les auteurs montrent qu'il est possible d'utiliser la méthode B pour décrire et valider le modèle des tâches de l'utilisateur, au niveau des phases de spécification et de conception. L'approche consiste à décrire une machine abstraite ou un modèle B pour représenter le contrôleur de dialogue de l'application interactive. Cette description est ensuite raffinée plusieurs fois afin de décrire les tâches et sous tâches jusqu'au niveau des événements tels que l'appui sur une touche du clavier. Notons qu'une tâche décomposable en sous tâches et/ou en événements est dite abstraite.

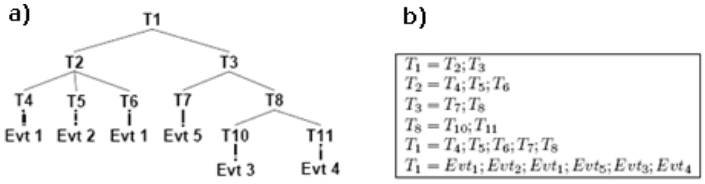

Figure 4 ***: modèle des tâches et décomposition***

La **Figure 4** a) propose le raffinement de la tâche abstraite T1 en T2 suivie de T3, avec l'opérateur séquence ";". La séquence est le seul opérateur permis. Les tâches T2 et T3 sont elles-même raffinées. La correction de la décomposition est assurée par les obligations de preuve effectuées lors du raffinement qui doit préserver les propriétés de la tâche initiale T1. La **Figure 4** b), montre la décomposition en événements atomiques (non décomposables) à partir du contrôleur de dialogue avec l'opérateur de composition. Ainsi, lorsque dans le raffinement nous atteignons les niveaux élémentaires des événements, le processus de validation est complet. Cette approche de validation présente deux intérêts. En premier lieu, la séquence des événements montre qu'il existe une séquence d'éléments simples pour implémenter une tâche abstraite, ce qui permet la validation de la tâche. En second lieu, si un ou plusieurs éléments simples sont manquants, les obligations de preuve relatives aux événements élémentaires ne peuvent pas être prouvées ce qui montre que la conception doit être complétée. Il s'agit donc d'une validation de la conception et de la décomposition architecturale.

Nous présentons la classification de la méthode d'évaluation proposée par les auteurs, par rapport à la taxonomie d'Ivory [Ivory 01] dans le tableau de la **Figure 5**. Nous classons cette méthode dans la classe analyse de modèle, car elle consiste, dans un premier temps, à produire un modèle et à le raffiner. Dans un deuxième temps, il s'agit d'effectuer les obligations de preuve pour le raffinement, ce qui constitue le type de la méthode. Ces preuves étant faites automatiquement, cette méthode a pour niveau d'automatisation l'analyse de modèles et elle requiert simplement la production de ces modèles.

Catégorie	Classement
Classe de la méthode	*Analyse de modèle*
Type de la méthode	*Obligation de preuve du raffinement du modèle B*
Niveau d'automatisation	*Analyse automatique*
Niveau d'effort	*Production de modèle*

Figure 5 ***: classement de la méthode [Aït-Ameur 03] par rapport à la taxonomie d'Ivory [Ivory 01]***

Dans [Jambon 02], l'auteur propose une méthode de développement de systèmes critiques interactifs qui garantit la sûreté, et dans une certaine mesure, l'utilisabilité du produit logiciel obtenu. Cette méthode, qui s'appuie sur des outils, propose de spécifier les besoins, puis de décrire l'application selon la méthode B et une architecture CAV. A partir d'un certain niveau de raffinement, il propose d'utiliser la syntaxe BØ. Il est possible d'exprimer certains invariants qui garantissent la sûreté de l'application. L'expression de pré conditions permet de garantir dynamiquement certaines propriétés d'utilisabilité, comme l'insistance. Le

raffinement successif effectué pour décrire complètement l'application, ainsi que les invariants permettent de garantir, en réalisant les obligations de preuve, que l'application est sûre, et que les propriétés d'utilisabilité décrites sont vérifiées. Notons qu'un outil permet de compiler les descriptions B produites vers du code C. Il s'agit donc ici d'exploiter la méthode B, son principe de raffinement et d'obligation de preuve pour montrer la sûreté, et quelques propriétés d'utilisabilité. Dans la Figure 6, nous classons cette utilisation de la méthode B conformément à la taxonomie de [Ivory 01]. Dans la classification, la différence entre cette utilisation de la méthode B et celle faite par [Aït-Ameur 03], réside dans le type de la méthode, c'est-à-dire la manière d'effectuer l'évaluation : par obligation de preuve sur les invariants, les pré conditions et les raffinements de la description B. Nous pouvons dire que la méthode B couvre les activités suivantes du modèle de développement en V : la spécification, la conception architecturale et détaillée, l'implémentation par génération et le test unitaire d'intégration et de validation. Ces trois dernières activités sont couvertes grâce au mécanisme d'obligation de preuve lié au raffinement du modèle, ainsi qu'aux pré conditions et aux invariants.

Catégorie	Classement
Classe de la méthode	*Analyse de modèle*
Type de la méthode	*Obligation de preuve sur des invariants, pré condition, et raffinements.*
Niveau d'automatisation	*Analyse automatique*
Niveau d'effort	*Production de modèle*

Figure 6 : ***classement de la méthode [Jambon 02] par rapport à la taxonomie d'Ivory [Ivory 01]***

IOGs, Interaction Object Graphs

IOGs (Interaction Object Graphs) est une notation graphique de spécification exécutable [Carr 97], extension des StateCharts et des diagrammes de transition. IOGs peut servir à spécifier un système interactif jusqu'aux détails du fonctionnement des widgets (objets graphiques). Pour cela, elle propose de décrire les widgets, puis les relations entre les widgets, et, enfin, le dialogue complet de l'Interface Homme-Machine, par la description d'états et de transitions. Les descriptions sont interprétables, et il est possible de prouver certaines propriétés. Les auteurs proposent, par exemple, de montrer qu'un système est complet s'il donne une réponse à toutes les actions utilisateurs. Cette méthode, dont la description selon la classification d'Ivory est donnée en **Figure 7**, est intéressante si l'on souhaite prototyper un modèle d'interaction et l'exécuter.

Catégorie	Classement
Classe de la méthode	*Analyse de modèle*
Type de la méthode	*Preuve de propriétés d'utilisabilité, et test par l'interprétation des modèles.*
Niveau d'automatisation	*aucune*
Niveau d'effort	*Production de modèle*

Figure 7 : ***classement de la méthode IOGs [Carr 97] par rapport à la taxonomie d'Ivory [Ivory 01]***

VEG, Visual Event Grammar

Dans [Berstel 05], les auteurs proposent d'utiliser des VEG (Visual Event Grammar), pour spécifier, concevoir, et implémenter formellement le comportement des composants d'une interface graphique. VEG étend les grammaires BNF traditionnelles pour permettre la description de modèles de dialogue. Les spécifications VEG sont indépendantes des rendus visuels, et peuvent être facilement intégrées pour s'adapter à différentes boîtes à outils graphiques. La méthode est accompagnée d'outils pour concevoir, compiler, et effectuer des vérifications formelles sur les descriptions produites. La conception passe par la rédaction d'un modèle de dialogue d'interface graphique sous la forme d'un automate, utilisant comme vocabulaire terminal des actions utilisateurs de haut niveau. L'évaluation est réalisée en produisant d'abord un modèle abstrait des descriptions VEG selon une série de règles proposées par les auteurs, puis en effectuant des vérifications de propriétés logiques et temporelles par model – checking avec l'outil SPIN. Le model - checking est une technique puissante pour vérifier de manière automatique qu'une machine d'états finis vérifie des propriétés données, préalablement décrites dans un langage de logique temporelle (ici Promela). Cette méthode permet de montrer l'atteignabilité d'un état, l'absence d'interblocage, ou encore la correction d'invariants. Elle permet également, à condition d'avoir pris connaissance du comportement de l'application interactive modélisé, de valider le modèle d'interaction. La méthode complète couvre également la branche descendante du modèle de développement en V (spécification, conception, implémentation). Nous classons cette méthode d'évaluation dans la classe analyse de modèle, et de type vérification par model-checking. Cette méthode nécessite la description manuelle du dialogue selon VEG, et les vérifications par model-checking sont semi automatiques. La Figure 8 reprend ce classement sous la forme d'un tableau.

Catégorie	Classement
Classe de la méthode	*Analyse de modèle*
Type de la méthode	*Vérification de propriété logique et temporelle par model-checking*
Niveau d'automatisation	*Analyse automatique*
Niveau d'effort	*Production de modèle*

Figure 8 : ***classement de la méthode VEG [Berstel 05] par rapport à la taxonomie d'Ivory [Ivory 01]***

Modélisation LUSTRE

[Ausbourg 98] présente un environnement de développement et de vérification de propriétés basé sur le langage LUSTRE. La démarche proposée est la suivante : à partir d'un générateur d'interface UIM/X, une description LUSTRE du comportement interactif de l'application est produite automatiquement. Cette description peut ensuite être vérifiée par model-checking. Pour réaliser la production du modèle de l'application en LUSTRE, l'auteur utilise une bibliothèque de widgets de base décrits en LUSTRE, et un générateur de modèles, qui prend en entrée le code C, et une description UIL de l'interface. Le résultat de la génération est un ensemble de nœuds LUSTRE. Ceux-ci peuvent être soumis à des vérifications par model-checking. Ceci permet de vérifier des propriétés d'atteignabilité et de sûreté. Les propriétés LUSTRE à vérifier sont décrites graphiquement par le testeur au sein d'un éditeur. Cette

approche graphique rend accessible la méthode de validation. La partie évaluation présente deux aspects : d'une part, le model-checking permet de montrer certaines propriétés d'utilisabilité sur les modèles LUSTRE de l'application interactive, ce qui permet de valider la décomposition architecturale, ainsi que les éléments de l'architecture. D'autre part, lorsque le model–checker montre la véracité d'une propriété, il produit des descriptions de jeux d'essais. Ces jeux d'essais peuvent être appliqués à l'implémentation finale de l'application interactive afin de montrer la correspondance entre celle-ci et son modèle (c'est-à-dire la correspondance entre la description et l'implémentation finale d'une application interactive). Nous classons cette méthode (Figure 9) dans la catégorie analyse de modèle, puisque son objectif premier est la vérification par model- checking.

Catégorie	Classement
Classe de la méthode	*Analyse de modèle*
Type de la méthode	*Vérification de propriété logique et temporelle par model-checking et production de jeu d'essais pour la validation de la correspondance entre modèle et implémentation finale.*
Niveau d'automatisation	*Analyse automatique*
Niveau d'effort	*Effort minimal*

Figure 9 : ***classement de la méthode LUSTRE [Ausbourg 98] par rapport à la taxonomie d'Ivory [Ivory 01]***

Un des inconvénients des méthodes présentées (B [Aït Ameur 03, Jambon 02], IOGs [Carr 97], VEG [Berstel 05], et d'Ausbourg [Ausbourg 98]) est qu'elles ne sont pas largement répandues dans la communauté IHM, et ne tiennent pas compte de la production réelle d'application interactive. L'approche VEG parait toutefois intéressante dans le sens où elle est susceptible d'être adaptée à différents langages de programmation objet. L'approche LUSTRE d'Ausbourg [Ausbourg 98] est complètement intégrée et est accessible à des non spécialistes. Néanmoins, toutes les applications interactives ne sont pas décrites en UIM/X. Ces approches constituent un fondement qui montre à quel point l'enjeu de la validation d'Interface Homme - Machine est important. Nous montrons dans le paragraphe suivant que si ces approches ne permettent pas d'évaluer complètement des applications interactives, elles sont encore plus démunies face à la complexité croissante qu'impose la multimodalité à l'activité d'évaluation d'application interactive.

Interaction Multimodale

Dans ce chapitre, nous montrons que l'introduction de la multimodalité dans les applications interactives rend leur développement et leur validation plus complexe. Cette complexification est due, d'une part à l'augmentation du nombre de modalités liées aux dispositifs physiques, et d'autre part à l'utilisation du temps pour la combinaison de plusieurs modalités. Nous définissons d'abord les notions de modalité et de multimodalité en termes de combinaisons naturelles et temporelles de modalités. Enfin, nous montrons quelques exemples de combinaisons de modalités proposés dans la littérature.

Modalité d'interaction et multimodalité

Modalité

Nous adoptons la définition orientée vers la technologie de [Nigay 95] : une modalité d'interaction est définie par un couple <d, L> où d désigne un dispositif physique et L un langage d'interaction.

- Un *dispositif physique* est un élément du système qui acquiert des informations (dispositif d'entrée) ou fournit des informations à l'utilisateur (dispositif de sortie). Les dispositifs physiques communs d'entrée sont le clavier, la souris, le microphone, l'appareil GPS (Guidage par satellite), ou encore le magnétomètre. Parmi les dispositifs de sortie, nous pouvons citer l'écran, les haut-parleurs ou encore le casque semi transparent.
- Un *langage d'interaction* définit un ensemble d'expressions bien formées et significatives (par exemple, un assemblage conventionnel de symboles). En entrée, la génération d'un symbole, ou d'un ensemble de symboles, résulte d'actions sur les dispositifs physiques d'entrée. Des exemples de langages d'interaction incluent le langage pseudo-naturel, la manipulation directe ou encore la localisation.

Une modalité d'entrée comme la parole peut être décrite par le couple <microphone, langage pseudo-naturel>, où le langage pseudo-naturel est défini par une grammaire spécifique. De façon similaire, la modalité de localisation d'un utilisateur peut être décrite par le couple <GPS, localisation en données GPS>. Cette définition d'une modalité caractérise les échanges entre le système et l'utilisateur en mettant en relation deux niveaux d'abstraction : le niveau physique (dispositif) et le niveau logique (langage d'interaction). Basés sur cette définition de la modalité, nous distinguons les modalités actives et passives.

Modalité active

Une modalité d'entrée est qualifiée d'active quand l'utilisateur doit réaliser une action explicite avec un dispositif en vue de spécifier une commande au système comme la parole <microphone, langage pseudo-naturel>. De même dans les interfaces manipulables, la modalité <ordinateur de poche, langage gestuel> constitue une modalité active, l'utilisateur inclinant par exemple l'ordinateur de poche pour faire défiler une liste. Une modalité de sortie est dite active lorsque l'utilisateur doit effectuer une action explicite pour percevoir les informations véhiculées par la modalité, telle que regarder l'écran d'un ordinateur de poche.

Modalité passive
Une modalité est caractérisée de passive quand le dispositif qui la constitue ne requiert pas l'attention et d'action explicite de l'utilisateur, telle que la capture par un GPS de la localisation d'un utilisateur, la modalité passive correspondante étant décrite par le couple <GPS, localisation en données GPS>. De nombreuses interfaces (interfaces sensibles au contexte et "*perceptual user interfaces*") exploitent des modalités d'entrée passives afin de rendre l'interaction plus robuste et plus efficace. De même, de nombreux systèmes de réalité augmentée sur supports mobiles reposent sur des modalités de sortie passives exploitant un casque semi-transparent (selon le principe de l'attention minimale, qui est considéré comme crucial en situation de mobilité).

Multimodalité

La multimodalité reflète le caractère de multiplicité des modalités pour un système interactif. Un système est multimodal s'il dispose d'au moins deux modalités pour un sens donné, entrée ou sortie. Ainsi, le système est qualifié de multimodal en entrée (en sortie) si au moins deux modalités d'entrée (de sortie) sont disponibles.

Usage combiné de modalités

Des études ont caractérisé la multimodalité, en terme d'usage combiné de plusieurs modalités. Les propriétés CARE [Coutaz 95, Nigay 97] définissent les quatre manières d'utiliser une modalité : la complémentarité, l'assignation, la redondance et l'équivalence. Nous revenons sur les définitions proposées pour ces usages.

- Complémentarité : l'usage de deux modalités distinctes est complémentaire pour un but donné, s'il est nécessaire d'utiliser les deux pour l'expression de ce but.
- Assignation : une modalité est dite assignée à un but si elle seule est disponible pour exprimer ce but.
- Redondance : l'usage de deux modalités est redondant pour un but donné lorsque qu'elles sont utilisables en *même temps* pour la réalisation de ce but. Chacune des modalités exprime le même but.
- Équivalence : l'usage de deux modalités est équivalent pour un but donné s'il est possible d'utiliser indifféremment l'une ou l'autre pour la réalisation de cette tâche. Une seule des modalités est utilisable à la fois.

Notons que la redondance implique l'équivalence des modalités.

Aspects temporels de l'usage combiné de modalités

Il a été observé que les propriétés CARE pour la multimodalité ne tiennent pas suffisamment compte de la notion de temps dans l'usage combiné de plusieurs modalités [Vernier 00]. Ainsi, l'auteur propose un espace de conception pour l'interaction multimodale en sortie. Dans cette proposition, on peut distinguer la *sélection* d'une ou plusieurs modalité(s) de manière statique, soit par le concepteur, soit par l'utilisateur via un paramétrage, et de manière dynamique par le système, en fonction de l'offre des dispositifs en sortie. D'autre part, on distingue la *composition* de ces modalités, et plus précisément la *composition temporelle*. Les auteurs proposent un ensemble de termes permettant de nommer les différentes compositions -temporelle, spatiale, syntaxique, et sémantique. Ces termes sont rassemblés dans le tableau de la Figure 10. Ils sont classés à l'horizontale, selon l'agencement des modalités, décrit par un schéma (disjoint et espacé, disjoint et adjacent, recouvrement partiel, recouvrement total, et simultané), et à la verticale selon la caractérisation de cet agencement par rapport aux notions temporelles, spatiale, syntaxique et sémantique. Par

exemple, la première case du tableau en haut à gauche définit comme *anachronique* une composition disjointe et espacée dans le temps. De même, une composition disjointe et adjacente de modalités dans le temps est dite en *séquence*. Cet espace de conception montre que la notion de temporalité, ainsi que celles de spatialité, de syntaxe, et de sémantique de l'utilisation combinée de plusieurs modalités, ne sont pas immédiates. Il faut donc définir rigoureusement ce que l'on souhaite réaliser comme combinaison lors de la production d'une application multimodale interactive. De la même manière, cette avalanche de cas possibles de combinaison de modalités influe sur la difficulté de vérifier une application multimodale, quelque soit son stade d'élaboration, depuis la spécification jusqu'à l'implémentation dans le modèle en V. Il est par exemple difficile de vérifier que la spécification des différentes compositions utilisées de l'application interactive multimodale est respectée.

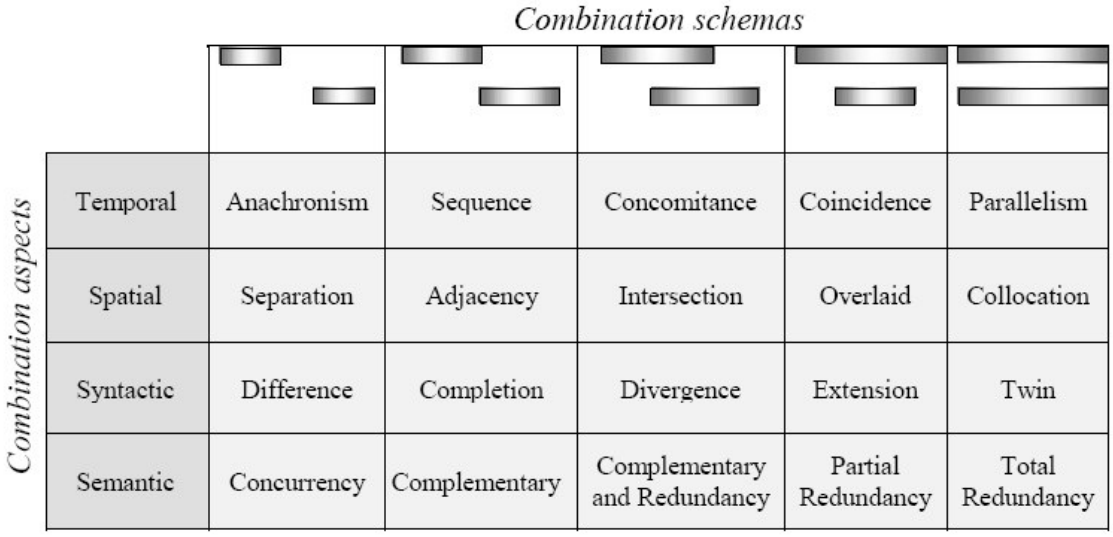

Figure 10 ***: tableau des schémas de combinaison de modalités en fonction de la notion de temps, d'espace, de syntaxe, et de sémantique.***

Méthodes Formelles pour la vérification d'Applications Multimodales

Nous présentons dans ce paragraphe quelques méthodes formelles proposées pour l'évaluation des applications interactives multimodales. Il faut préciser ici que la méthode B [Aït Ameur 03, Jambon 02], la méthode IOGs [Carr 97], la méthode VEG [Berstel 05] présentées en 0, peuvent être adaptées pour l'évaluation d'applications multimodales, bien qu'elles n'aient pas été spécifiquement produites à cet effet. Nous choisissons de présenter dans cette partie uniquement des méthodes d'évaluation dont le but affiché par leurs auteurs est l'évaluation d'applications interactives multimodales. Dans ce cadre, nous présentons l'approche de description et de vérification utilisant des machines à états finis, proposée par Bourget [Bourget 02, 03, 04]. Nous montrons ensuite comment le formalisme ICO [Palanque 03, Schyn 03], fondé sur l'utilisation de réseaux de Pétri de haut niveau permet l'évaluation d'applications interactives multimodales.

Machine d'états finis

Dans [Bourget 02,03,04], l'auteur présente deux logiciels, IMBuilder et IMEngine pour la spécification d'applications multimodales, et leur vérification. IMBuilder permet de modéliser le comportement en entrée d'une application interactive multimodale. Ces modélisations sont décrites à partir de machines à états finis (FSM pour Finite State Machine). Ces machines à états finis permettent de décrire le comportement dynamique d'une application à partir de quatre éléments de base : des états, des événements, des transitions et des actions. Ces quatre éléments sont représentés à la Figure 11. Cette figure contient un état source et un état cible, tous deux représentés par une ellipse. Une transition est modélisée entre les deux états par une flèche. Celle-ci est déclenchée par un événement, représenté par un soleil du coté de l'état source, et engendre une action représentée par un cercle noir, du coté de l'état cible.

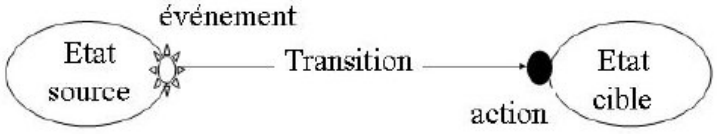

Figure 11 ***: exemple d'utilisation du formalisme FSM (finite state machine) [Bourguet 02]***

L'IMEngine est une plate-forme multimodale, permettant d'exécuter les descriptions de machines à états. L'auteur propose ainsi de tester informellement les modèles d'interaction sur la plateforme IMEngine. Le classement de cette méthode selon la taxonomie d'Ivory est illustré dans le tableau de la Figure 12. Nous associons à cette méthode la classe analyse de modèle, puisqu'il s'agit d'effectuer des tests sur des modèles. Aucune automatisation n'est précisée par l'auteur. L'effort de test porte sur deux niveaux : tout d'abord, la production de

modèles de l'application interactive multimodale, et ensuite le test informel des modèles produits.

Catégorie	Classement
Classe de la méthode	*Analyse de modèle*
Type de la méthode	*Test via une plateforme d'interprétation des modèles*
Niveau d'automatisation	*Aucun*
Niveau d'effort	*Production de modèle, Test informel*

Figure 12 : ***classement de la méthode FSM [Bourget 02,03,04] selon la taxonomie d'Ivory[Ivory 01]***

ICO (Interactive Co-operative Objects)

[Palanque 03] propose une méthode de description formelle dédiée aux systèmes interactifs multimodaux. Il s'agit du formalisme ICO (Interactive Co-operative Objects) étendu à la multimodalité. ICO repose sur l'utilisation de réseaux de Pétri de haut niveau. En ICO, un objet est une entité caractérisée par quatre composants : un objet coopératif (CO) avec des services utilisateur, une partie présentation, et deux fonctions (activation et rendu). L'article propose comme extension utile pour la multimodalité [Schyn 03] le temps, un mécanisme de communication par production et consommation d'événements, un mécanisme de structuration basé sur l'utilisation de transducteurs et un mécanisme de rendu afin de décrire le comportement des médias en sortie. Les auteurs montrent qu'il est possible de décrire un moteur de fusion d'événements multimodaux en entrée à l'aide de ce formalisme. Il présente ainsi la description du moteur de fusion d'une application interactif multimodal avec ICO, utilisant la voix et le geste.

La Figure 13 illustre le formalisme ICO. Les transitions sont représentées sous forme de rectangles, et les ellipses représentent les places. Les places peuvent contenir des jetons. Un état est modélisé par une distribution de jetons donnée entre les places. L'extension pour la multimodalité propose les transitions synchronisées sur des événements. Pour qu'une transition synchronisée sur un événement e_1 soit franchie, il faut avant tout qu'elle soit franchissable et que l'événement e_1 arrive. Une transition est franchissable si toutes ses places d'entrée possèdent au moins un jeton, comme c'est le cas pour la place p1 de la Figure 13. Si l'événement e_1 a lieu alors qu'aucune des transitions synchronisées associées n'est franchissable, il ne se passe rien (l'événement est implicitement consommé). Lorsqu'une transition est synchronisée sur un événement, on ajoute un éclair entrant accompagné du nom (et du détail, s'il y a lieu) de l'événement reçu. Lors de son franchissement, une transition peut aussi, en plus de son fonctionnement normal, émettre un événement e_2. Ce mécanisme est décrit dans la Figure 13 ci-dessous. Lorsqu'une transition émet un événement, on ajoute un éclair sortant accompagné du nom (et du détail, s'il y a lieu) de l'événement émis.

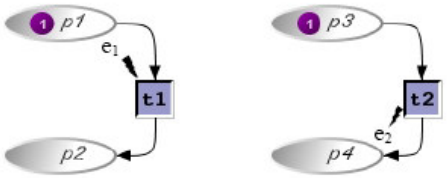

Figure 13 ***: formalisme ICO et extension pour la multimodalité***

L'évaluation du mécanisme de fusion produit, est réalisée par la preuve de propriétés sur le modèle ICO au sein de l'éditeur PetShop. Il est possible de vérifier si le réseaux de Pétri est *borné* (il ne peut ni consommer ni produire de ressources) ou s'il est en *vie* (il est toujours possible de trouver une séquence d'envoi, qui passe par toutes les transitions). Selon la taxonomie d'Ivory, nous plaçons cette méthode dans la classe analyse de modèles par la preuve. La production des modèles est nécessaire, afin d'être analysée au sein de l'outil PetShop.

Catégorie	Classement
Classe de la méthode	*Analyse de modèles*
Type de la méthode	*Preuve de vivacité*
Niveau d'automatisation	*Analyse de modèles*
Niveau d'effort	*Production de modèle*

Figure 14 ***: classement de la méthode ICO [Palanque 03, Schyn 03] selon la taxonomie d'Ivory[Ivory 01]***

Lutess et le test d'application interactives multimodales

Dans l'ensemble des méthodes formelles d'évaluation présentées, la production de modèles est nécessaire. Une autre approche d'évaluation est le test formel d'application interactive multimodale avec une approche synchrone, proposé dans [Madani 05]. Cette approche utilise un outil de vérification de système réactif synchrone, Lutess, pour le test d'applications interactives multimodales. Elle est fondée sur l'hypothèse qu'une application interactive multimodale peut, dans certaines conditions, se comporter comme un système réactif synchrone. L'application interactive multimodale est instrumentée dans le but de créer un certain nombre de points d'entrée et de sortie. Lutess simule des actions utilisateur au travers de la production d'un environnement. Un certain nombre de propriétés logiques et temporelles peuvent être exprimées sous la forme de nœud LUSTRE et vérifiées lors du test. Cette méthode permet notamment de vérifier des propriétés ergonomiques pour la multimodalité, telles que les propriétés CARE [Coutaz 95, Nigay 97]. Nous donnons plus de détails sur cette méthode d'évaluation au sein de notre espace solution au 0En se référant à la taxonomie d'Ivory, nous classons cette méthode dans la catégorie simulation, puisqu'il s'agit de simuler des actions utilisateur afin de vérifier des propriétés logiques et temporelles sur les entrées – sorties de l'application à tester. L'analyse est automatique, et ne nécessite aucun effort. En effet, l'instrumentation peut être comprise dans la production de l'application.

Catégorie	Classement
Classe de la méthode	*Simulation*
Type de la méthode	*Simulation d'action utilisateur et vérification de propriété logique et temporelle*
Niveau d'automatisation	*Analyse automatique*
Niveau d'effort	*Production de modèle*

Figure 15 ***: classement de la méthode Lutess [Madani 05] selon la taxonomie d'Ivory[Ivory 01]***

ICARE, Interaction CARE

[Bouchet 04] propose un formalisme permettant d'exprimer un moteur de fusion d'événements multimodaux : ICARE. Ce formalisme repose sur l'utilisation des propriétés CARE pour la multimodalité [Coutaz 95, Nigay 97], et de la notion de modalité comme couple <dispositif, langage d'interaction> [Nigay 95]. Cette approche est proposée avec un éditeur qui permet d'assembler et de manipuler graphiquement des composants ICARE sous la forme de boîtes et de relier graphiquement ces composants entre eux par des flèches. Le formalisme ICARE modélise la notion d'Assignation, d'Équivalence, de Redondance, et de Complémentarité, ainsi que la notion de modalité, à partir de ces boîtes et ces flèches. L'éditeur génère automatiquement du code exécutable JAVA à partir des schémas produits selon ce formalisme. Précisons que cette approche à composants permet uniquement de définir la partie interactive en entrée d'un système multimodal. Les auteurs illustrent l'utilisation de leur approche sur une application multimodale hautement interactive : le simulateur de l'avion militaire Rafale. Notons que l'implémentation de ce modèle à composants repose d'une part sur les JAVA Beans, pour les composants, et sur le modèle à événements JAVA pour la communication entre les différents composants. Cette approche à composants ICARE, bien qu'intéressante pour le prototypage et l'implémentation d'application interactive, ne dispose pas à l'heure actuelle de méthode d'évaluation automatique ou semi automatique des assemblages de composants ICARE, ni du résultat produit en terme de moteur de fusion. Nous classons cette méthode selon la taxonomie d'Ivory comme une méthode qui permet l'analyse des schémas en cours de production, mais également le test et l'inspection du produit fini, puisqu'il s'agit de développer la partie interactive de l'application.

Catégorie	Classement
Classe de la méthode	*Analyse de modèle, test, inspection*
Type de la méthode	*Analyse par l'expérience, le test et l'inspection du produit peuvent s'effectuer selon tout types de méthodes non formelles d'évaluation*
Niveau d'automatisation	*Aucun*
Niveau d'effort	*Test formel et informel*

Figure 16 ***: classement de la méthode ICARE [Madani 05] selon la taxonomie d'Ivory[Ivory 01]***

Synthèse

Dans cette partie, nous avons défini les propriétés ergonomiques comme critères de qualité pour l'utilisabilité d'un système interactif. Comme l'originalité de notre approche réside dans la vérification formelle de propriétés ergonomiques, il nous a semblé important de bien situer ces propriétés par rapport aux critères de qualité du Génie Logiciel qui sont classiquement considérés pour la vérification formelle.

Nous avons ensuite étudié les méthodes d'évaluation ergonomique afin de situer notre travail par rapport aux méthodes existantes. Pour cela, nous nous sommes basés sur la taxonomie d'Ivory [Ivory 01] qui regroupe et étend plusieurs taxonomies existantes. Cette taxonomie nous a servi de canevas intégrateur pour présenter les méthodes d'évaluation existantes qu'elles soient formelles ou non et dédiées à la multimodalité ou générales à l'interaction. Parmi les approches formelles, nous soulignons le peu de méthodes dédiées à la multimodalité, qui pourtant, par sa complexité accrue, justifierait l'utilisation de telles approches.

La plupart des méthodes formelles présentées ne visent pas la vérification d'une application interactive mais celle de modèles lors de la conception ou alors indirectement par la production de jeux d'essais. Lorsque la méthode permet de faire de la vérification/validation sur un modèle exécutable, elle nécessite de spécifier l'application complète dans un formalisme. Le coût peut alors être prohibitif et souvent le passage à l'échelle est difficile.

Ces deux constats nous semblent être à l'origine du peu d'intérêt dans la communauté IHM pour les approches formelles. En adoptant une approche pragmatique, notre objectif est la vérification formelle de l'utilisabilité d'applications interactives multimodales, avec un effort minimal [Ivory 01]. Pour cela nous optons pour une méthode de vérification par le test qui n'implique pas que l'application interactive elle-même soit spécifiée dans un formalisme particulier. Dans [Madani 05], une première étude de faisabilité a montré qu'il était possible de vérifier une application multimodale avec un outil de vérification de systèmes réactifs synchrones, Lutess, basé sur le langage Lustre. Cependant le coût de connexion de l'outil Lutess à l'application sous test est très important puisque les entrées/sorties de Lutess sont des booléens. Notre objectif est alors de fournir une méthode complète de vérification formelle basée sur Lutess en simplifiant la connexion entre Lutess et l'application multimodale sous test. Pour cela des hypothèses doivent être faites sur l'application sous test sans pour autant impliquer une spécification formelle de l'application. Notre solution consiste à considérer que l'application est développée par un assemblage de composants ICARE. Cette hypothèse sur l'architecture logicielle de l'application nous permet d'envisager une connexion semi-automatique avec Lutess : en effet nous verrons dans la partie 2 qu'il est possible d'exploiter la structure du code induite par l'utilisation d'ICARE pour simplifier la production de l'instrumentation nécessaire aux tests avec Lutess. De plus, ICARE étant une plateforme de développement rapide d'interfaces multimodales pour favoriser une conception itérative centrée utilisateur, le couplage avec Lutess nous permet d'obtenir une plateforme complète et intégrée de développement et de tests formels d'applications multimodales.

Pour conclure cette partie, la Figure 17 situe notre méthode de vérification et son instrumentation basée sur le couplage ICARE-Lutess au sein de la taxonomie d'Ivory. Notre solution fait l'objet de la partie suivante composée de quatre chapitres.

Catégorie	Classement
Classe de la méthode	*Simulation*
Type de la méthode	*La simulation d'action utilisateur et la vérification de propriété logique et temporelle s'effectue avec Lutess*
Niveau d'automatisation	*Analyse automatique*
Niveau d'effort	*Effort minimal*

Figure 17 ***: classement de notre méthode de vérification basée sur le couplage ICARE – Lutess au sein de la taxonomie d'Ivory [Ivory 01]***

Espace Solution

Méthode d'évaluation prédictive formelle basée sur le couplage ICARE-Lutess

Application Interactive Multimodale YellowPage

Nous commençons la deuxième partie du mémoire par la description des deux versions de l'application multimodale YellowPage. Cette application nous permet d'illustrer les outils utilisés et nos contributions des chapitres suivants.

YellowPage : présentation générale

Nous illustrons nos travaux en considérant deux versions d'une application interactive multimodale YellowPage. La première version développée par l'équipe IIHM nous a été fournie dans le cadre du projet VERBATIM. En effet, YellowPage est une des études de cas communes à tous les partenaires de VERBATIM. Nous avons conçu et développé la deuxième version.

YellowPage est une version très simplifiée d'une application comme PagesJaunes disponible à www.pagesjaunes.fr. En effet, l'application YellowPage permet de localiser un lieu sur une carte, à partir du nom et de l'adresse d'une personne. Pour cela, l'utilisateur réalise plusieurs tâches successives en rapport avec les lieux d'interaction de l'interface graphique présentée à la Figure 18 :

1) Spécifier le nom d'une personne.
2) Spécifier l'adresse de cette personne
3) Lancer une recherche.
4) Naviguer dans la carte (zoom, déplacement, recentrage, etc.).

YellowPage propose plusieurs modalités d'interaction en entrée, de l'utilisateur vers le système. Les trois dispositifs physiques disponibles sont : une souris, un clavier et un microphone. Ainsi, les tâches (1, 2, 3, et 4) peuvent être réalisées par l'utilisation de plusieurs modalités. La tâche de recherche peut par exemple être initiée soit avec la souris, en sélectionnant le bouton "Search", soit par l'enfoncement de la touche "enter" du clavier, ou encore par la commande vocale "search".

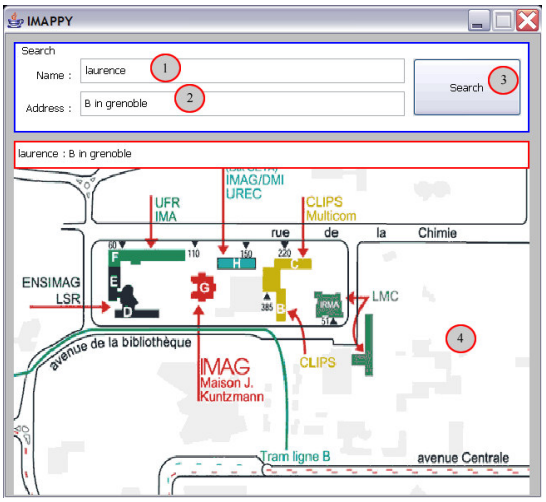

Figure 18 : ***capture d'écran du système interactif multimodal YellowPage version 1***

Au sein de ce prototype, pour les champs de texte "Name" et "Address", le nombre de noms reconnus est limité à trois (noms parmi "Cooper", "Laurence" et "Smith"), et le nombre d'adresses à deux, chacune décrite de trois manières (adresses parmi "Buidling < B ou C > in Grenoble", "< B ou C > in Grenoble" et "< B ou C> Grenoble").

YellowPage, deux versions, deux finalités

Nous considérons deux versions de YellowPage afin de répondre aux deux étapes de notre étude. Les deux versions ont été développées avec un moteur de fusion d'événements multimodaux réalisé en composants ICARE. La différence essentielle entre les deux versions réside dans l'implémentation. Dans la version 1, l'assemblage des composants est réalisé manuellement, par l'écriture d'une classe. Dans la version 2, que nous avons développée, l'assemblage des composants ICARE a été spécifiée graphiquement et le code généré automatiquement grâce à la plateforme ICARE. Aussi au sein du logiciel de la version 1, avec assemblage manuel, l'instanciation des composants ICARE et la communication entre ces composants sont explicites. A l'opposé dans notre version 2, le code généré par la plateforme ICARE est sérialisé : les composants ICARE et la communication entre eux ne sont plus explicites. La modification du code est alors impossible. Or elle est incontournable car le test implique une instrumentation de l'application sous test.

Lors de la conception/développement de la version 2 de YellowPage avec la plateforme ICARE étendue, nous avons réutilisé certains composants ICARE de la version 1 que nous avons ajoutés dans la plateforme pour qu'ils soient manipulables au sein de l'éditeur ICARE. Nous avons aussi conçu une nouvelle interface présentée à la **Figure 19** avec de nouvelles fonctionnalités. Par rapport à la version 1 de la Figure 18, de zones d'interaction sont ajoutées: la barre de zoom (1), ainsi qu'un panneau de déplacement (2), et un panneau de recherche amélioré (3).

1. Nous avons entièrement développé la barre de zoom (1) : celle-ci fonctionne selon le principe des barres de défilement. Pour spécifier un facteur de zoom, l'utilisateur peut cliquer sur la loupe et maintenir le bouton de la souris appuyé tout en déplaçant le curseur. L'utilisateur peut aussi déplacer la loupe en cliquant directement sur un facteur de zoom.
2. Le nouveau panneau de déplacement (2) que nous avons conçu et développé contient sept boutons de déplacement. Ces derniers sont faciles à utiliser puisque chacun contient un libellé et une icône facilitant leur identification par l'utilisateur. De plus ce panneau indique aussi à l'utilisateur les mots-clefs reconnus pour le déplacement ou la rotation par commandes vocales. Le panneau joue donc un double rôle. Par rapport à la version 1, les commandes vocales pour la rotation ont été ajoutées.
3. Le panneau de recherche (3) a été programmé aussi afin de changer son apparence en particulier pour le bouton de recherche qui, là encore, contient un libellé et une icône pour une meilleure identification et une prévisibilité acrue (deux propriétés ergonomiques).

De plus, nous avons ajouté des nouvelles fonctionnalités pour que les panneaux soient rétractables, afin de laisser de la place à la visualisation de la carte, objet central de la tâche de l'utilisateur. Chaque panneau présenté ci-dessus peut être fermé ou étendu, soit par manipulation directe en cliquant sur la zone en bleu du bord du panneau, soit par commandes vocales. Les commandes vocales ajoutées sont : "search/move/zoom hide/show" (<nom du panneau> cacher/montrer). Lorsque l'utilisateur rétracte un panneau, une animation fait alors

disparaître le panneau vers le bord de la fenêtre, ne laissant visible que la carte et la zone bleue du panneau rétracté. Ces possibilités de manipulation des panneaux par la souris ou la parole ne sont pas explicitées à l'utilisateur et réservées aux utilisateurs experts. Enfin, nous avons ajouté des animations lors des déplacements et rotations de la carte, afin de minimiser les risques de désorientation de l'utilisateur. Le développement de cette version de YellowPage avec la plateforme ICARE a duré trois semaines.

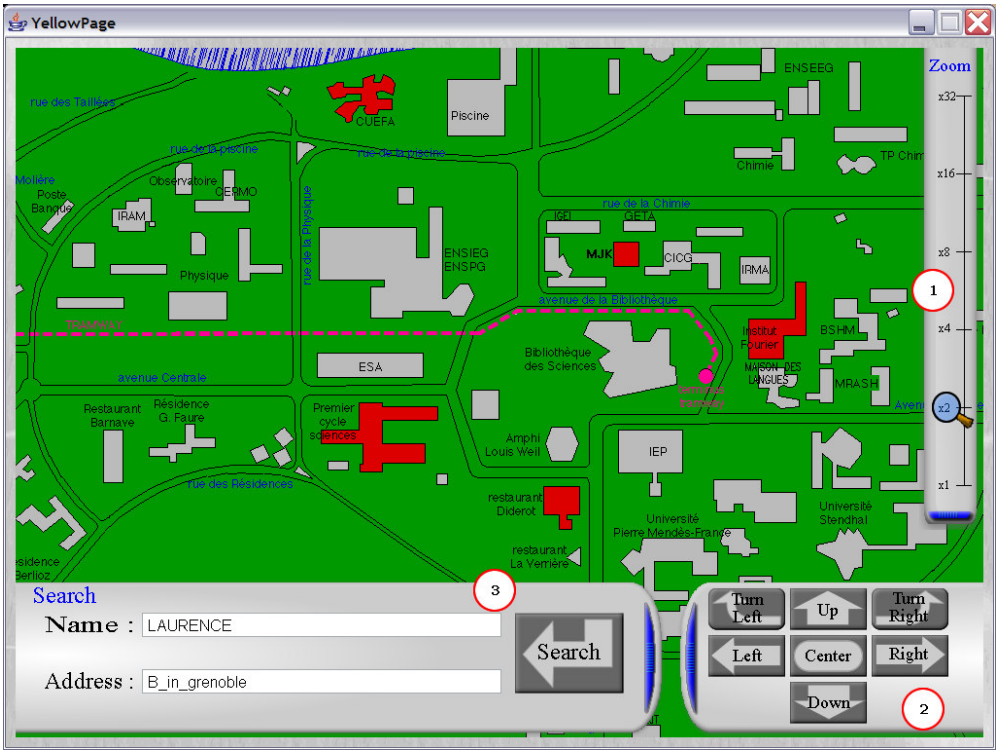

Figure 19 : ***capture d'écran du système interactif multimodal YellowPage version 2***

Dans le chapitre suivant, nous détaillons la conception logicielle des deux versions de YellowPage en présentant les assemblages des composants ICARE correspondant, comme illustration de l'architecture Ã composants ICARE sur lequel nous nous basons pour connecter l'application sous test et Lutess.

Plateforme ICARE

Notre méthode de vérification formelle d'applications multimodales que nous détaillons au chapitre suivant (chapitre 9) repose sur le fait que l'application sous test est développée par un assemblage de composants ICARE. En effet, cette contrainte sur l'architecture logicielle de l'application sous test nous permet d'envisager une connexion semi-automatique avec la plateforme de tests Lutess que nous exposons au chapitre 10. Aussi dans ce chapitre, nous présentons l'approche ICARE, son architecture logicielle en composants et sa plateforme de conception/développement que nous illustrons avec les deux versions de l'application YellowPage.

Nous présentons d'abord le modèle à composants ICARE. Puis, nous exposons les combinaisons d'entrées proposées par YellowPage dans sa version 1 (assemblage manuel), accompagnées des schémas ICARE les implémentant. Nous présentons ensuite la plateforme ICARE [Bouchet 04], pour Interaction-CARE (Complémentarité, Assignation, Redondance, Équivalence). Elle permet aux concepteurs de manipuler graphiquement et d'assembler des composants logiciels ICARE afin de spécifier l'interaction multimodale pour une tâche donnée d'une application interactive multimodale. De cette spécification, le code de l'assemblage est automatiquement généré. Nous illustrons l'usage de la plateforme ICARE avec l'application YellowPage version 2.

Présentation d'ICARE (modèle à composants)

Le modèle conceptuel à composants ICARE comprend deux types de composants : les composants élémentaires qui permettent de définir des modalités d'interaction et les composants de composition (ou combinaison) qui permettent de spécifier l'usage combiné de modalités. Les composants de composition sont indépendants des modalités à composer.

Basés sur la définition d'une modalité d'interaction comme étant l'association d'un dispositif physique d avec un langage d'interaction L, <d, L> [Nigay 97], deux types de composants ICARE élémentaires sont identifiés : les composants de type Dispositif et les composants de type Langage d'Interaction. Une modalité d'interaction modélisée par deux composants élémentaires est présentée à la **Figure 20**.

- Un composant Dispositif représente une couche supplémentaire du pilote d'un dispositif physique. Il s'agit du niveau physique d'une modalité. Par exemple, le composant Dispositif Souris abstrait les données (mouvements effectués ou pression sur les boutons) fournies par le pilote de la souris.
- Un composant Langage d'Interaction correspond au niveau logique d'une modalité d'interaction. Par exemple, un composant Langage d'Interaction peut abstraire les données d'un composant Dispositif Souris en une commande correspondant à la sélection dans un menu d'options.

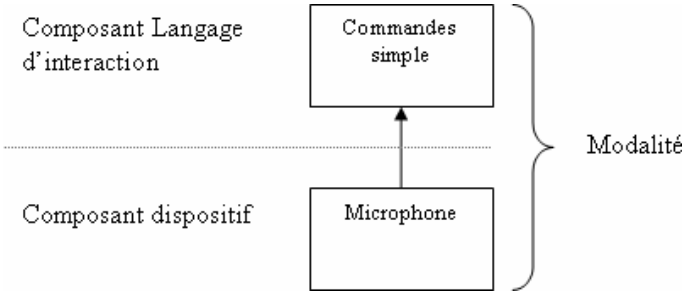

Figure 20 : ***une modalité constituée d'un composant dispositif et d'un composant langage d'interaction***

Les composants de composition sont définis en se basant sur les propriétés CARE (Complémentarité, Assignation, Redondance, Équivalence) [Coutaz 95, Nigay 97]

- L'équivalence et l'assignation caractérisent la portée des choix en matière de modalités d'interaction. Deux modalités sont équivalentes pour une tâche élémentaire ou commande si l'utilisateur peut utiliser l'une ou l'autre des modalités pour réaliser la tâche. Au contraire lorsqu'une modalité est assignée à une tâche, l'utilisateur n'a pas le choix de la modalités.
- La redondance et la complémentarité qualifient la combinaison de modalités. Deux modalités sont utilisées de manière redondante si elles sont équivalentes et véhiculent la même information dans une fenêtre temporelle donnée. C'est le cas lorsque l'utilisateur énonce la commande vocale « search » tout en cliquant sur le bouton « Search » avec la souris.
  Deux modalités sont utilisées de façon complémentaire dans une fenêtre temporelle donnée si chacune véhicule une information utile pour obtenir une commande complète. Par exemple lorsque l'utilisateur énonce la commande vocale « Zoom in here » tout en sélectionnant une position sur la carte, deux modalités distinctes sont utilisées pour spécifier la commande de zoom sur un point.

Basés sur ces propriétés CARE, quatre composants ICARE de composition sont définis et permettent de combiner les données de 2 à n composants : un composant de composition pour la Complémentarité, un pour la Redondance, un pour l'Équivalence et un pour la Redondance/Équivalence. Ce dernier caractérise l'usage de modalités équivalentes qui peut être soit redondant (Redondance) soit exclusif (Équivalence). L'Assignation de CARE n'est pas explicite dans la spécification ICARE car elle est représentée par un lien unique entre deux composants. En effet, un composant A lié à un composant B implique que A est assigné à B.

La communication entre les composants ICARE est implémentée sous la forme d'événements ICARE (ICAREEvents). Cette communication est statique à l'exécution et est représentée dans un schéma ICARE par des flèches comme le montre la **Figure 20**. Une flèche dénote un langage entre les composants émetteur et receveur. Les ICAREEvents circulant entre deux composants portent un des mots de ce langage. Les composants de type Dispositif émettent des ICAREEvents de bas niveau d'abstraction. Ceux-ci sont traduits au sein des composants de type Langage d'Interaction en événements de plus haut niveau d'abstraction, pour être ensuite fusionnés par les composants de composition afin d'obtenir des tâches ou commandes complètes traitées par le reste du système interactif.

Pour cerner le rôle des composants ICARE au sein d'une application interactive, nous les positionnons par rapport à l'architecture logicielle complète d'une application interactive. Pour cela, nous considérons le modèle d'architecture de référence : ARCH. A la **Figure 21**, nous observons à gauche (a), une application interactive multimodale décrite selon le modèle d'architecture ARCH. Celui-ci comprend cinq parties : le noyau fonctionnel qui intègre les fonctionnalités propres au domaine de l'application, l'adaptateur du noyau fonctionnel qui a un rôle d'interface logicielle, le contrôleur de dialogue qui gère l'enchaînement des tâches, l'interaction logique et physique qui définissent l'interaction entre l'utilisateur et le système. Les composants ICARE se positionnent au niveau des parties interaction logique et physique, comme cela est représenté à la **Figure 21** (b). Pour l'interaction multimodale en entrée, la communication des composants ICARE vers le reste de l'application s'effectue donc vers le contrôleur de dialogue et est réalisée au travers de composants "tâche abstraite" : ces derniers reçoivent des événements ICARE et les transmettent sous la forme de commandes au contrôleur de dialogue de l'application. Il s'agit donc d'une simple indirection.

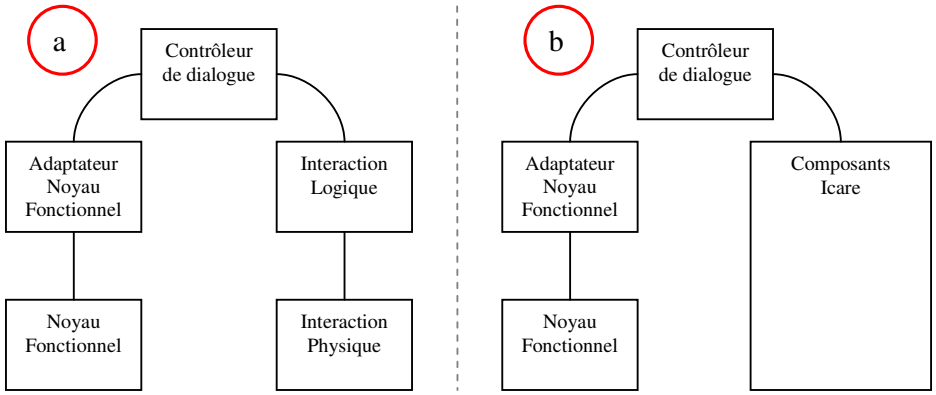

Figure 21 ***: (a) Le modèle d'architecture ARCH. (b) Les composants ICARE dans ARCH.***

Une implémentation choisie pour les composants ICARE est basée sur les javaBeans. Trois classes JAVA spécifiques à ICARE sont définies. Ces classes sont utilisées et étendues pour définir un composant ICARE : une classe événement ICARE "ICAREEvents", une classe dispositif "ICAREDevice", et une classe langage d'interaction "ICAREInteractionLanguage". Les deux dernières sont étendues respectivement dans l'implémentation de la classe mère de chaque composant dispositif et langage d'interaction.

Les composants ICARE ont une structure générique. Ils possèdent une interface "<nom composant>Listener", une classe "<nom composant>Support", ainsi qu'une classe "<nom composant>Info". Cette dernière sert à spécifier les informations pour la manipulation graphique des composants dans l'éditeur ICARE, telles que l'image qui doit être utilisée pour représenter le composant. L'interface "<nom composant>Listener" et la classe"<nom composant>Support" permettent la communication entre composants. La première est une classe écouteur d'événements qui peut être implémentée afin d'obtenir un accès aux événements en sortie du composant. La deuxième gère les abonnements à ce composant et l'émission des événements par notification à ses abonnés. Ainsi, lorsque nous souhaitons connecter deux composants entre eux, l'un émetteur, et l'autre récepteur, il convient :

- créer une classe anonyme A qui implémente l'interface de l'écouteur du composant émetteur,

- étendre la méthode de notification d'événements de cette classe anonyme A pour qu'elle appelle le composant récepteur.
- abonner la classe anonyme au composant émetteur, par un appel à la méthode d'abonnement de la classe "emmetteurSupport".

Une fois ces opérations effectuées, si le composant émetteur produit un événement, alors l'instance de sa classe "emmetteurSupport", émet l'événement à l'ensemble de ses écouteurs, dont la classe anonyme A. Celle-ci reçoit la notification et retransmet l'événement au composant récepteur.

Nous illustrons l'approche ICARE avec la première version de YellowPage développée par un assemblage manuel de composants ICARE.

Schéma de YellowPage v1

Les schémas ICARE présentés dans cette section sont ceux conçus pour le développement de l'application YellowPage v1. Nous verrons au chapitre 10 comment nous avons exploité ces schémas ICARE pour la vérification formelle de propriétés ergonomiques. Dans cette section, nous présentons d'abord les schémas ICARE de YellowPage v1, puis nous abordons leurs implémentations.

Conception

Un schéma ICARE est défini par tâche (ou commande) ou par classe de tâches que le contrôleur de dialogue gère. Nous considérons d'abord les commandes simples sans paramètre : le schéma ICARE correspondant est décrit à la **Figure 22**. Celui-ci comprend trois composants de type Dispositif : "Mouse", "Keyboard" et "Microphone" associés à 3 composants de type Langage d'Interaction pour constituer 3 modalités. La souris est assignée à la commande rechercher, aussi dans le schéma ICARE le composant « Mouse » est assigné au composant Langage "Search". Les composants Langage "Simple Commands 1 et 2" traduisent les entrées provenant du clavier et du microphone en commandes parmi "zoom in", "zoom out", "up", "down", "right", "left", "center", et "search". Les trois modalités sont fusionnées de manière redondante et équivalente grâce au composant de composition ICARE Redondance/Equivalence. Le composant Redondance/Equivalence définit une fenêtre temporelle T dans laquelle sera traitée la redondance d'information : par exemple si l'utilisateur énonce la commande vocale « search » et successivement clique sur le bouton «Search» avec la souris dans un intervalle de temps inférieur à T, alors les deux commandes sont fusionnées en une seule commande de recherche.

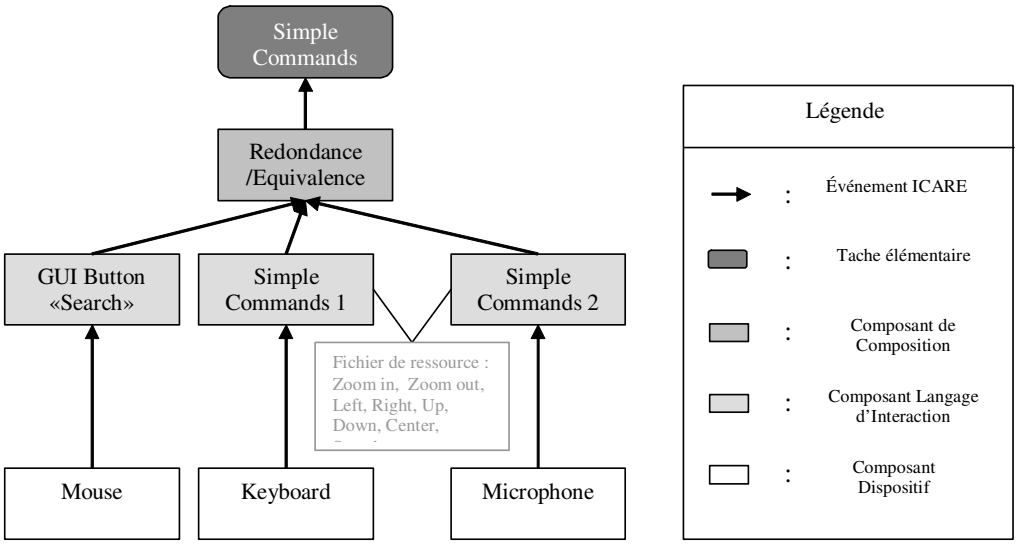

Figure 22 ***: schéma ICARE pour les commandes simple sans paramètre***

Outre les commandes simples, l'utilisateur peut utiliser une ou deux modalités pour les tâches complémentaires de sélection d'un champ et de remplissage de ce champ. La sélection du champ peut s'effectuer par la souris en cliquant sur le champ ou oralement par le mot "name" ou "address". Le remplissage peut s'effectuer au clavier ou à l'oral (les valeurs autorisées étant pour le nom : "laurence", "smith", "cooper", et pour l'adresse : "B_in_grenoble", "building_B_in_grenoble", "B_grenoble", "C_in_grenoble","building_C_in_grenoble", et "C_grenoble"). La **Figure 23** présente le schéma ICARE des tâches de spécification d'un nom et d'une adresse d'une personne. L'utilisateur peut spécifier le nom ou l'adresse d'une personne de manière uniquement vocale par les phases : "name <nom d'une personne>" ou "address <adresse d'une personne>", ce qui est représenté par les deux entrées du contrôleur de dialogue : "Specify Name" et "Specify Address". L'assemblage pour "Specify Name or Address", utilisant le composant "complementarity 2" assure la construction d'une commande complète à partir d'une action de sélection d'un champ de texte (microphone : « name » ou « address » ou sélection par clic souris) et l'insertion d'une phase dans le champ de texte sélectionné (oralement ou au clavier). Le composant « complémentarity 2 » fusion de manière complémentaire ces entrées et produit des sorties de la forme "<Champ : X, Valeur : Y>" . Notons que le composant de composition Complémentarité fusionne les informations de sélection et de remplissage de manière ordonnée (port de gauche, puis port de droite).

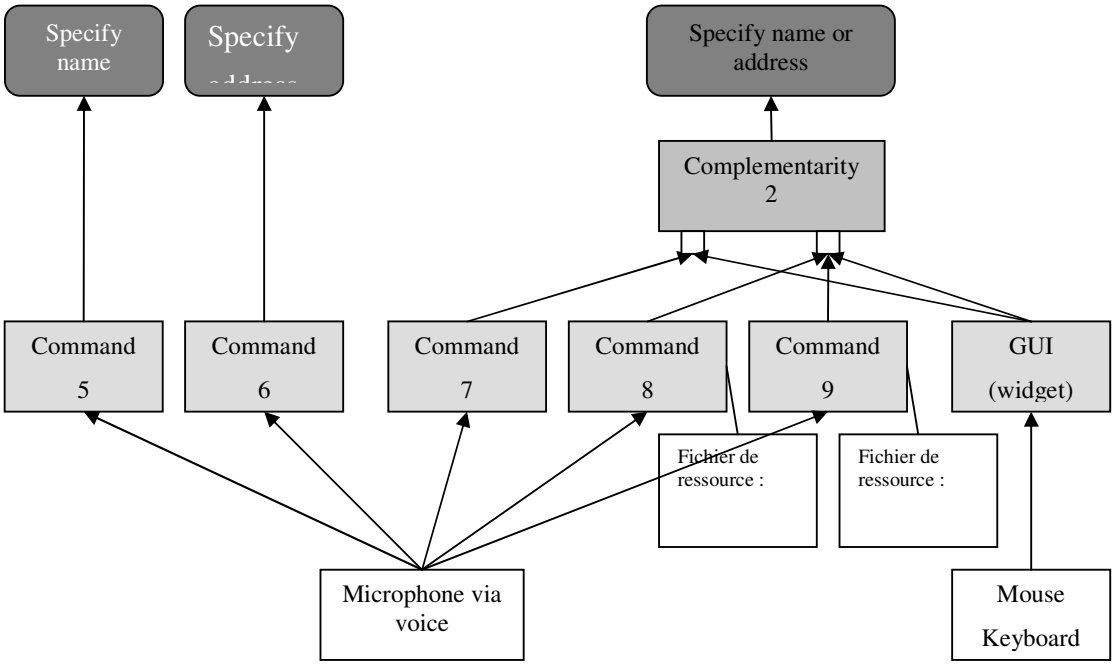

Figure 23 : ***schéma ICARE pour les commandes Spécifier : <nom> ou <Adresse>***

Enfin YellowPage v1 inclut des tâches de zoom avant/arrière et de centrage, sur un point donné de la carte, dont une partie du schéma ICARE est présentée à la **Figure 24**. Pour effectuer un zoom, l'utilisateur peut spécifier "zoom in/out here" au clavier ou au microphone, et spécifier un point avec la souris. Par exemple l'utilisateur prononce "zoom in here" et pointe avec la souris aux coordonnées (x,y) de la carte, les deux actions étant effectuées à des dates comprises dans la fenêtre temporelle T. Le composant "Complementarity 1" fusionne alors ces deux entrées pour obtenir la commande complète "zoom in point(x,y)". La tâche de centrage sur un point représentée par le schéma ICARe de droite à la **Figure 24**, est réalisée en cliquant avec la souris sur un point de la carte.

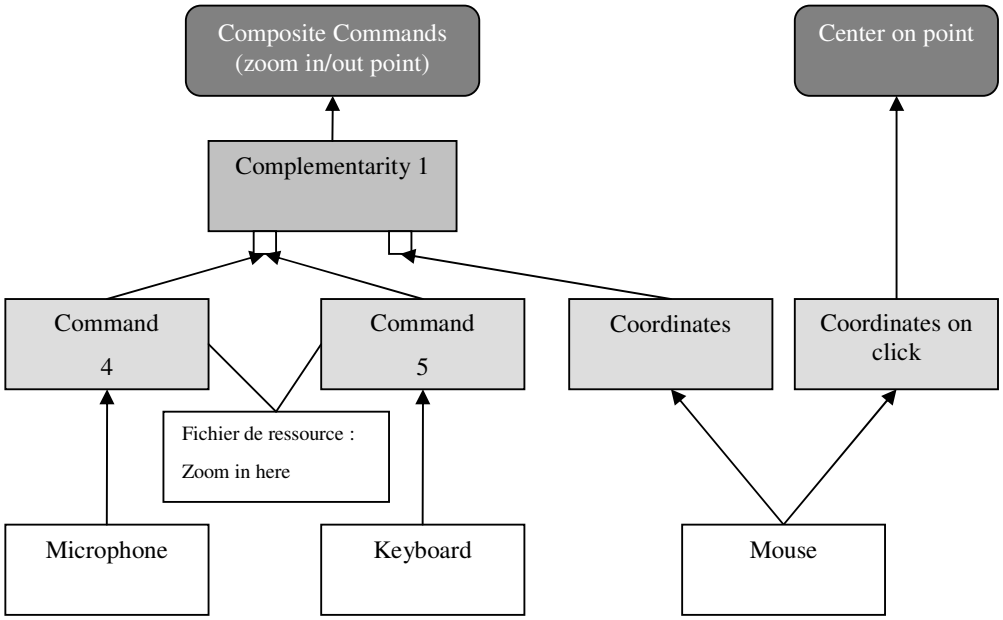

Figure 24 : schéma ICARE pour les commandes zoom avant/arrière sur <point> de YellowPage.

Implémentation

Les trois schémas ICARE présentés précédemment constituent la partie interactive en entrée de la première version de YellowPage. Cette version réalisée sans la plateforme ICARE, a été développée manuellement par l'écriture d'une classe JAVA qui crée l'instance des composants ICARE et leurs communications. A titre d'exemple, nous présentons à la **Figure 25**, la réalisation logicielle de la modalité <dispositif : clavier, langage d'interaction : commande simple> selon 5 étapes. La première étape ([a]) consiste à déclarer les deux composants, dispositif et langage d'interaction, les deux étapes suivantes ([b] et [c]) créant une instance de chacun de ces composants. L'étape ([d]) consiste alors à charger le fichier de ressources du composant langage d'interaction. Ce fichier décrit la correspondance entre des touches du clavier et des commandes. Par exemple le couple <1C_PRESSED -> CENTER> signifie que si le composant reçoit un événement "touche espace enfoncé", alors il émet un événement "Centrer la carte". La connexion entre les deux composants constitue la dernière étape ([e]). Celle-ci est réalisée par l'instanciation d'un écouteur du dispositif clavier. La méthode "NewKeyboardEvent" (nouvel événement clavier) de l'interface "KeyboardListener" (écouteur de clavier) est étendue pour appeler la méthode "setData" du composant langage d'interaction avec en paramètre l'événement reçu par l'écouteur.

```
// déclaration des composants [a]
keyboardDirectInput.KeyboardDirectInput kdi;
SimpleCommand.SimpleCommand scl;

// instanciation du dispositif : clavier [b]
kdi = new keyboardDirectInput.KeyboardDirectInput();

//instanciation du langage d'interaction : commande simple [c]
scl = new SimpleCommand.SimpleCommand();
    //chargement du fichier de ressources des commandes [d]
    scl.setCommandsFile("simplecommands.ica");

// creation du lien entre les deux composants [e]
kdi.addKeyboardInputListener(new keyboardDirectInput.KeyboardInputListener() {
    public void NewKeyboardEvent(ICARE.ICAREEvent data_values) {
     scl.setData(data_values);
    }
 });
```

Figure 25 : ***déclaration d'une modalité à partir du composant dispositif "clavier" et langage d'interaction "commande simple"***

Le principe de réalisation, présentée en cinq étapes à la **Figure 25**, est applicable pour les autres composants "dispositif" et "langage d'interaction". Les composants de "composition" que sont "Redondance/Équivalence", "Complémentarité", et "Redondance" possèdent deux entrées. Cela implique à l'étape [e], la création d'un lien supplémentaire avec un autre composant ICARE en entrée.

Dans cette section, nous avons présenté la réalisation manuelle d'une application multimodale, YellowPage v1, pour illustrer le modèle conceptuel à composants ICARE et son implémentation avec la technologie javaBeans. Nous considérons maintenant la plateforme ICARE qui produit automatiquement le code d'une application multimodale à partir de schémas ICARE.

Logiciel "ICARE tools"

La plateforme ICARE [Bouchet 04] permet aux concepteurs de manipuler graphiquement et d'assembler des composants logiciels ICARE afin de spécifier l'interaction multimodale en entrée pour une tâche donnée d'une application interactive. Le code de l'assemblage est automatiquement généré à partir de la spécification construite.

Une copie d'écran de la plateforme ICARE est présentée à la **Figure 26**. Nous présentons les fonctionnalités de la plateforme par rapport aux zones d'interaction numérotées sur cette copie d'écran. La plateforme permet à l'utilisateur de créer/charger des projets (1), dans lesquels il peut créer/modifier des schémas ICARE (2). L'utilisateur dispose aussi d'une palette d'outils (3). Cette dernière présente les composants disponibles pour l'édition de schémas ICARE. Ceux-ci sont organisés selon les trois types de composant ICARE : composant de "dispositif" (4), composants de "langage d'interaction" (5), et composants de "composition" (6). La palette contient deux autres outils : le composant "tâche abstraite" (7) et la liaison entre composants (8).

Placer un composant dans un schéma s'effectue dans un premier temps, en sélectionnant le composant dans la palette, et dans un deuxième temps, en cliquant sur la surface d'édition d'un schéma ICARE (9). La liaison entre les composants s'effectue en trois étapes : tout d'abord, sélectionner le composant émetteur par exemple "mouseDirectInput" (9a), ensuite, presser le bouton "link" (9b) de la palette, et enfin, sélectionner le composant récepteur par exemple le composant "MouseCoordinates" (9c). Cette suite d'actions produit un lien unidirectionnel entre le composant émetteur et le composant récepteur, représenté par un

flèche noire (9d). L'utilisateur peut modifier les propriétés des composants ICARE présent dans le schéma. Pour cela, il sélectionne un composant, par exemple le composant de "tâche abstraite" "EffaceCa" (10a). Cela provoque l'affichage d'une palette de propriétés modifiables (10b). L'utilisateur peut alors modifier les valeurs des propriétés du composant sélectionné. Pour le composant "tâche abstraite", seul le nom peut être modifié (10c). Une fois que le schéma est terminé, l'utilisateur peut sauvegarder le projet. Cela provoque la génération du code correspondant au schéma décrit.

L'assemblage de composant proposé dans la **Figure 26** est une commande "Effacer ça". L'utilisateur final peut énoncer la commande vocale "Effacer ça" tout en pointant avec la souris sur un objet numérique de l'application. Le composant complémentarité fusionne les deux informations pour produire une commande "Effacer l'objet aux coordonnées <X,Y>".

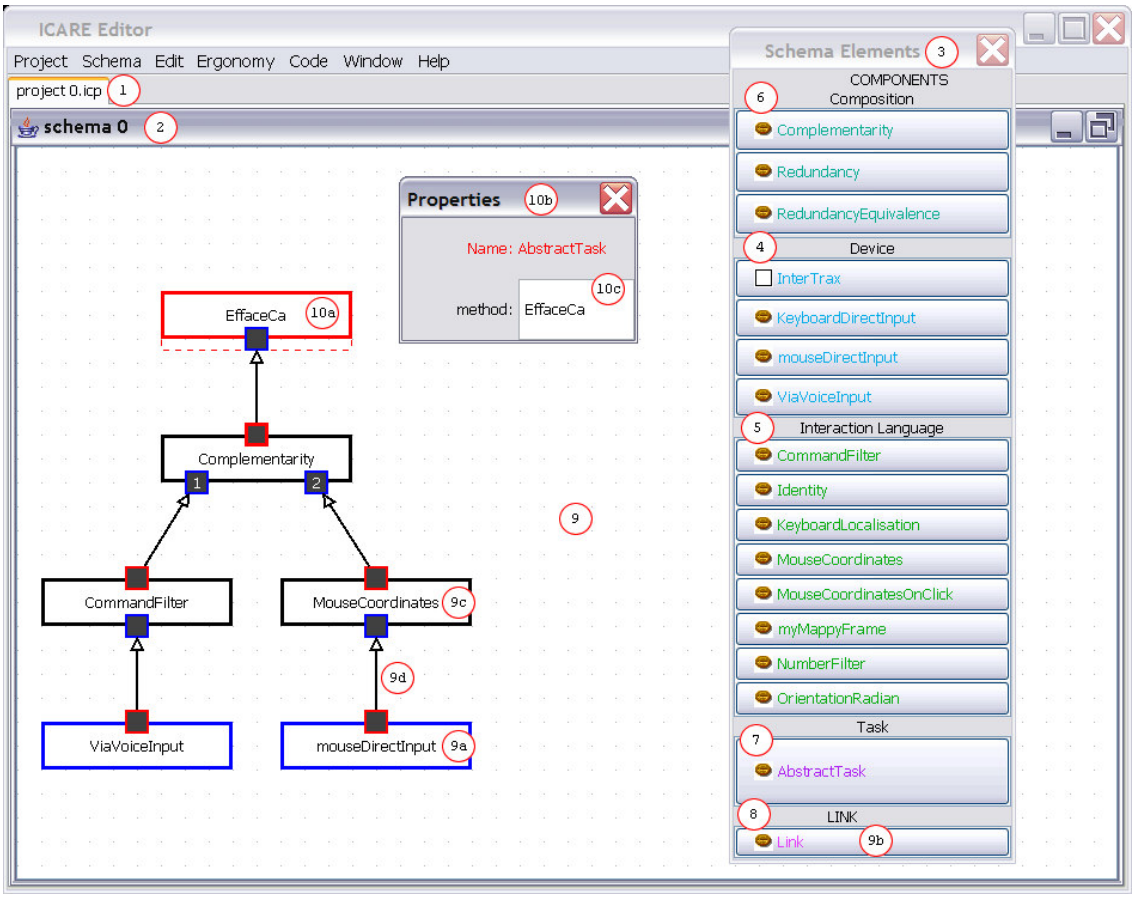

Figure 26 ***: capture d'écran de l'éditeur graphique de schéma ICARE.***

La structure du schéma ICARE est gérée au sein d'une classe JAVA nommée "ICAREContainer". La structure du schéma comprend l'ensemble des composants ICARE qui y sont représentés, ainsi que l'ensemble des liaisons entre ces composants. Pour cela, la classe "ICAREContainer" contient deux structures de données (composants et liaisons) sous la forme de "Hashtable" JAVA (classe implémentant une table de hachage). La classe "ICAREContainer" regroupe un ensemble de méthodes pour ajouter ou enlever un composant ou une liaison à ces tables de hachage. Les composants stockés dans la table de hachage sont des instances de la classe du composant qu'ils implémentent. Il s'agit d'objets (au sens du

paradigme de la programmation objet). Ainsi, la modification d'une propriété d'un composant dans l'éditeur, modifie la propriété de cet objet. Les clefs des instances de composants dans la table de hachage sont des objets "Integer" JAVA (classe implémentant les entiers), dont la valeur entière correspond au numéro ordonné de création des composants dans l'éditeur. Les instances des classes écouteurs d'événements ICARE, qui implémentent les liaisons dynamiques entre composants, sont stockées au sein de la deuxième table de hachage. Les clefs d'accès de cette table sont des "String" JAVA (classe implémentant une chaîne de caractères), composés du numéro du composant émetteur, et du numéro du composant récepteur séparé par un "-". La méthode de gestion des composants de la classe "ICAREContainer" consiste donc de manipuler ces tables de hachage au fur et à mesure de l'assemblage au sein de l'éditeur.

Cette structure facilite l'utilisation du mécanisme de sérialisation proposé par JAVA pour le chargement et la sauvegarde d'un schéma ICARE dans l'éditeur. La sérialisation consiste à sauvegarder séparément le code d'une classe correspondant à une instance de cette classe et les valeurs des attributs de cette classe pour une instance donnée. Dans le cas de la sauvegarde d'un schéma ICARE, la sérialisation est produite en trois étapes. La première étape est l'arrêt de l'ensemble des composants du schéma ICARE par la méthode "stop()". La deuxième étape est la sauvegarde des attributs de chaque objet du schéma (composants et écouteurs). La troisième étape est l'écriture du contenu des deux tables de hachage et des autres attributs de l'instance de la classe "ICAREContainer". Ces trois étapes produisent un fichier nommé "<nom_du_schéma>.icp". Notons que le fichier produit n'est pas lisible, puisqu'il est constitué pour moitié du contenu des attributs sous forme sérialisée dans une syntaxe propre à JAVA, et pour moitié du contenu des tables de hachage, dans une syntaxe également propre à JAVA. L'opération inverse (souvent appelée *dé-sérialisation*) crée les instances des classes à partir du code sérialisé et du code des classes. Le chargement d'un schéma ICARE utilise ce principe. En effet, il consiste dans un premier temps à charger l'ensemble des composants sérialisés à partir du code des classes de ces composants, et du code de ses attributs sérialisés. Dans un deuxième temps, l'ensemble des objets écouteurs est chargé, à partir du code de ces écouteurs (classe contenue dans chacun des JAVABeans représentant les composants ICARE), et du code de leurs attributs sérialisés. Dans un troisième temps, les composants JAVABeans sont redémarrés avec la méthode "run()". Notons que l'utilisation du mécanisme de sérialisation de JAVA implique que l'ensemble des composants ICARE et des classes les constituant implémentent l'interface "serializable". La classe "ICAREContainer" implémente également cette interface.

Le mécanisme de sauvegarde produit, à une adresse choisie par l'utilisateur de la plateforme, un ensemble de fichiers, répartis selon l'arborescence de la **Figure 27**. A la racine, dans le dossier du projet, nous retrouvons le fichier de sérialisation ".icp", ainsi qu'un dossier "nom_du_projet_code" contenant le code ICARE. Ce dossier contient lui-même plusieurs éléments : l'ensemble des composants ICARE présents dans le schéma, sous la forme d'archive ".jar". Il contient deux dossiers "FC" et "IcareInput" qui incluent des classes générées : celles-ci servent d'une part à démarrer le moteur de fusion ICARE, et d'autre part à lancer l'interface de celui-ci avec le reste de l'application interactive multimodale. Le dossier "nom_du_projet_code" contient également deux fichiers exécutables : "generate_interface.bat" pour compiler l'ensemble des classes nécessaires pour le fonctionnement du moteur de fusion ICARE et de son interfaçage avec l'application multimodale, et "run.bat" pour lancer le moteur de fusion.

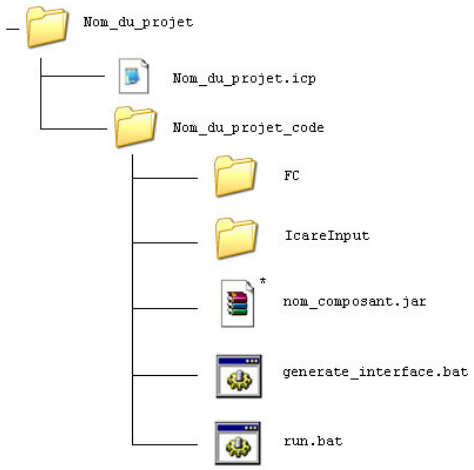

Figure 27 ***: hiérarchie des fichiers produits par la sauvegarde d'un schéma ICARE***

L'exécution d'une application possédant un moteur de fusion ICARE implique le démarrage de ce moteur de fusion. Celui-ci est implémenté dans une classe "ICARE.java" généré par l'éditeur, et placé dans le dossier "IcareInput" de la **Figure 27**. Le mécanisme de chargement du schéma ICARE par la classe "ICARE.java" est identique au chargement dans l'éditeur présenté plus haut. La connexion de la partie ICARE au reste d'une application interactive multimodale est à réaliser manuellement au sein d'un fichier en partie généré nommé "IcareInterface.java" présent dans le dossier "FC" de la **Figure 27**. Ce dernier contient un ensemble de méthodes qui sont appelées par les écouteurs des composants "tâche abstraite" du schéma ICARE. Elles possèdent le nom attribué par l'utilisateur dans l'éditeur ICARE, par exemple le composant "tâche abstraite" "EffacerCa" présent à la **Figure 26**. L'interface consiste à implémenter une indirection vers les méthodes souhaitées du reste de la partie interactive de l'application multimodale. Le développeur de schéma ICARE doit fournir des fichiers de configuration pour certains des composants ICARE comme le composant "commandFilter". Celui-ci effectue la traduction d'un sous-ensemble des événements reçus en entrée en commandes de plus haut niveau d'abstraction au sein du moteur de fusion ICARE. Le nom de ces fichiers est précisé au sein de l'éditeur par l'utilisateur, et le nom de leur extension est ".ica".

Après avoir expliqué le fonctionnement de la plateforme ICARE, nous l'illustrons en considérant la conception et le développement de la deuxième version de YellowPage que nous avons conçue et développée.

Schéma de YellowPage v2

Le schéma de YellowPage v2, présenté dans cette section, est moins clair que ceux de YellowPage v1. La différence provient du fait que les schémas de la version 1 sont des descriptions manuelles, explicitement scindés et clarifiés. Notre objectif dans cette section est de montrer le schéma ICARE d'une application interactive multimodale, comme il peut être produit avec la plateforme.

Nous parcourons ici le schéma ICARE de la **Figure 28** de gauche à droite (depuis le repère (a) jusqu'au repère (j)). Nous retrouvons la spécification d'un nom et d'une adresse de manière uniquement vocale par les assemblages de composants en (a). Le dispositif (b) correspond à un microphone fournissant des mots ou des expressions reconnus. Nous retrouvons également un assemblage en composition (c) pour la sélection/spécification d'un

nom/adresse avec deux modalités distinctes. Le composant (d) n'est pas un composant ICARE classique : en effet il contient une partie de la présentation graphique de l'application multimodale YellowPage (donc l'interface en sortie). L'objectif est ici de ne pas réécrire une boîte à outils mais bien de l'inclure dans un schéma ICARE afin de ne pas forcer le développeur à apprendre une nouvelle boîte à outils. Ainsi le composant (d) contient des commandes liées à l'interface graphique en sortie que nous avons conçue et présentée au chapitre 7 :

- les commandes de zoom grâce à la barre de zoom (1),
- les commandes de navigation proposées par le panneau de mouvement (2),
- les commandes de sélection à la souris d'un champ de texte(3),
- les commandes de remplissage d'un champ de texte au clavier(3),
- les commandes pour cacher/montrer les différents panneaux (zone bleu des panneaux de zoom (1), de navigation (2), et de recherche (3)).

La tâche abstraite "specifyZoom" de la **Figure 28** (e) définit le point de contact avec le contrôleur de dialogue de l'application permettant d'émettre des commandes de zoom avec un facteur : par exemple "zoom x4". L'assemblage en "Redondance/Equivalence" (f) correspond à la construction des commandes simples, telles que "zoom in", "search" ou encore "stop YellowPage". Le dispositif (g) correspond à l'appui d'une touche du clavier. L'assemblage (h) représente la possibilité d'émettre des commandes complexes comme "zoom in point(x, y)". Le dispositif nommé "MouseDragInput" (i) correspond à l'usage de la souris pour le déplacement haut/bas/gauche/droite et le clic. Ainsi l'assemblage (j) permet de produire des commandes "center on point(x,y)", simplement en cliquant sur un zone de la carte.

Nous constatons que le schéma obtenu n'est pas très différent de ceux proposés pour la première version. Cela est due, d'une part à la grande souplesse de la plateforme ICARE pour modifier une application et d'autre part, à l'ajout de commandes pouvant être insérées dans le schéma équivalent à celui de la version 1. Les fichiers de configuration des composants sont néanmoins plus volumineux. Notons que la modification de l'interface graphique n'entraîne pas ou peu de modifications au sein du schéma ICARE d'une application puisque dédié à l'interaction en entrée. La nouvelle interface que nous avons conçue est implémentée dans le composant "myMappyFrame" (d).

Conclusion

Dans ce chapitre, nous avons détaillé l'approche ICARE et sa plateforme. Nous avons illustré cette approche en considérant les deux versions de l'application multimodale YellowPage : la version 1 étant développée par un assemblage manuel et la version 2 développée avec la plateforme (code généré). Dans ce chapitre, l'accent a été mis sur l'architecture logicielle en composants ICARE et la structure du code généré par la plateforme. En effet, notre méthode d'évaluation formelle basée sur Lutess s'appuie sur la structure ICARE du code correspondant à l'interaction multimodale de l'application sous test. Notre méthode d'évaluation formelle fait l'objet du chapitre suivant.

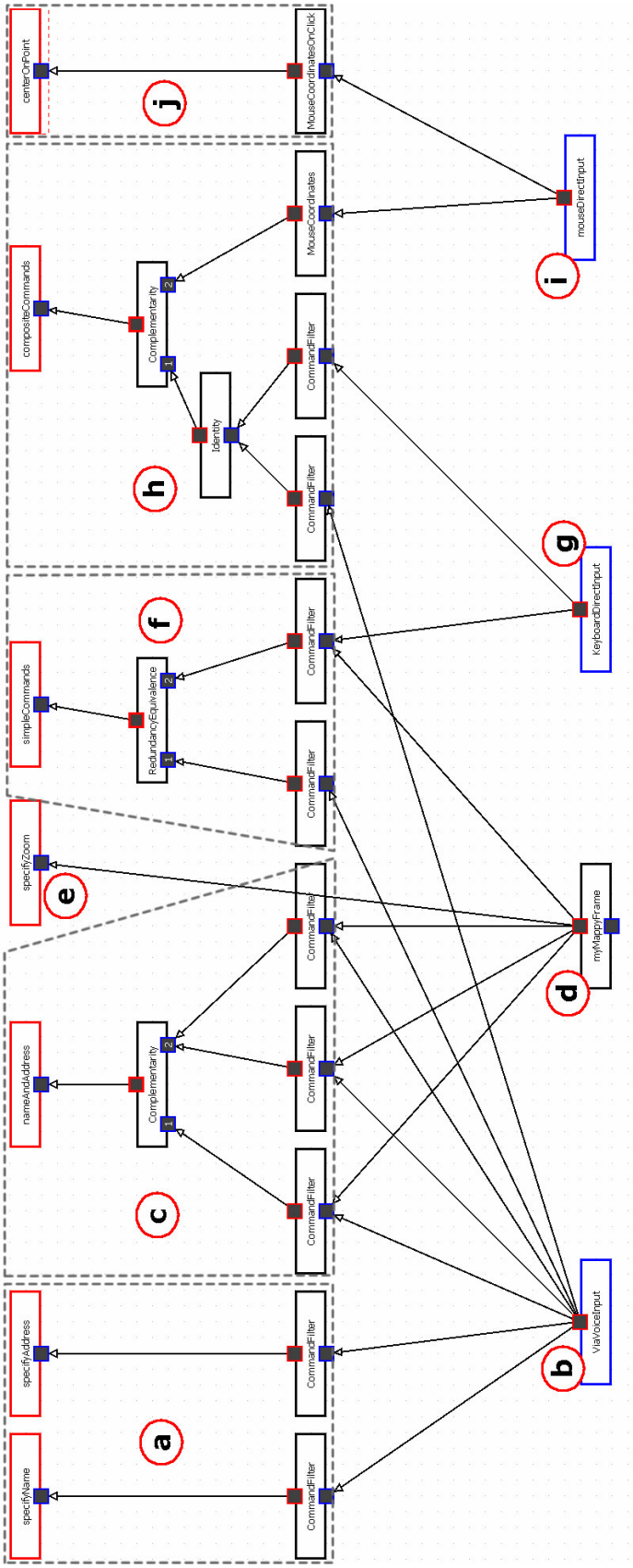

Figure 28 ***: schéma ICARE complet de YellowPage version 2 construit avec l'éditeur ICARE***

Vers une Méthode d'Évaluation Prédictive Formelle

Nous rappelons que nous avons opté pour une méthode d'évaluation basée sur Lutess (chapitre 5) une plateforme à l'origine dédiée pour le test de systèmes réactifs synchrones et que nous faisons l'hypothèse que l'application sous test est développée selon une architecture ICARE. Cette contrainte sur l'application sous test nous permet d'envisager une connexion semi-automatique entre l'application sous test et Lutess. En effet l'application sous test doit disposer d'entrées et de sorties auxquelles Lutess se connectent. Lutess utilise une description d'environnement pour générer des jeux d'essai au cours du temps, et une description d'oracle pour vérifier une propriété logique et temporelle sur les entrées – sorties de l'application sous test [Madani 05].

Ce chapitre explique notre méthode d'évaluation formelle d'application multimodale en plusieurs étapes. Les aspects techniques et en particulier la connexion entre les composants ICARE et Lutess sont ensuite détaillés dans le chapitre 10 suivant.

Pour expliquer nitre méthode, nous rappelons d'abord le principe général d'une vérification par Lutess. Dans un deuxième temps, nous exposons la construction d'environnements pour guider la génération des jeux d'essai pour l'interaction multimodale puis la construction d'oracles pour le test. Enfin, nous proposons une liste non exhaustive des propriétés ergonomiques que nous souhaitons vérifier sur une application interactive multimodale.

Principe de vérification

Notre approche de vérification repose sur l'hypothèse qu'une application interactive multimodale peut se comporter comme un système réactif synchrone, c'est-à-dire, que son comportement au cours du temps peut être représenté sous la forme d'une suite de couples <entrée, sortie>. Une application interactive est sensée réagir aux sollicitations de l'utilisateur dans un temps donné. Si on considère que le temps de réaction de l'application est nul, alors on peut considérer que l'application fonctionne sous la forme d'action – réaction. Fort de cette hypothèse, nous utilisons l'outil de vérification de systèmes réactifs synchrones Lutess pour le test d'application interactive multimodale. Lutess construit automatiquement un générateur de données de test pour le logiciel sous test. Il permet donc de simuler des entrées de l'application interactive multimodale. Lutess est également capable de récupérer les sorties du logiciel sous test. Ces sorties peuvent être prises en compte lors de la génération des vecteurs d'entrées, et peuvent donc être utilisées pour l'expression de la propriété de l'oracle.

Lutess impose un modèle de fonctionnement pour le test. Celui-ci correspond à une suite de cycles action – réaction. Un cycle se déroule comme suit :

1. Lutess génère un vecteur d'entrées.
2. Lutess l'envoie au logiciel sous test puis attend.
3. Le logiciel sous test lit le vecteur d'entrées.
4. Le logiciel sous test réagit en envoyant un vecteur de sorties à Lutess.
5. Lutess lit les sorties du logiciel sous test.
6. Lutess vérifie la propriété décrite dans l'oracle en prenant en compte les entrées-sorties de ce cycle.

A la **Figure 29**, nous situons ces étapes du cycle au sein de l'architecture globale qui regroupe Lutess et le logiciel sous test. Précisons que la génération d'un vecteur d'entrées est guidé par la description d'un fichier dit "d'environnement", et que la propriété logique et temporelle testée est décrite dans un fichier dit "d'oracle". Lutess impose pour sa communication vers

l'extérieur, de passer par la lecture – écriture de vecteurs de booléens sur la sortie standard (booléens représentés par des entiers de valeur 0 pour "faux" et de valeur 1 pour "vrai"). La **Figure 29** illustre cette communication par les étapes 2, 3, 4, et 5.

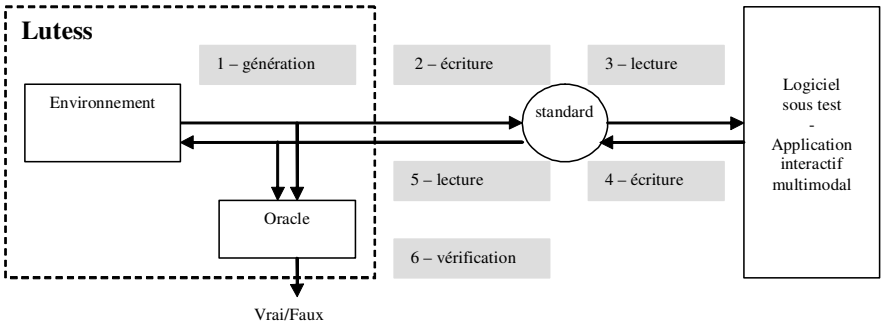

Figure 29 : ***architecture globale et étapes des cycles de vérification d'un logiciel interactif multimodal avec Lutess.***

Environnement

La spécification d'un fichier d'environnement permet d'une part, de décrire les entrées – sorties du logiciel sous test, et d'autre part de guider Lutess dans la génération des vecteurs de test. Lutess possède trois modes de génération :

- La génération aléatoire : un vecteur de booléen est tiré au hasard pour chaque cycle du test.
- La génération guidée par les probabilités : l'environnement définit un ensemble d'affectation de probabilités conditionnelles pour que chaque booléen soit vrai.
- La génération par scénario : l'environnement définit un ensemble de scénarios basés sur l'écriture de relation causale.

Chaque mode permet d'exprimer des contraintes particulières, qui vont guider Lutess dans la génération des vecteurs de test. Lutess reconnaît trois types de contraintes, exprimées par la :

- Spécification d'invariants : il s'agit de propriétés logiques et temporelles sur les entrées-sorties du logiciel sous test. Lutess garantit alors que l'ensemble des jeux d'essai qui sont soumis vont satisfaire ces invariants. Par exemple, si E1 et E2 sont deux entrées du logiciel sous test, il est possible d'exprimer la propriété : "E2 suit toujours E1" par l'expression Lustre "not E2 or pre E1".
- Spécification de probabilités conditionnelles : il s'agit d'affecter la probabilité P qu'une entrée E soit vraie, si une condition C est vraie, par l'expression "proba ((E, P, C))". Lutess vérifie à chaque cycle la validité de C, puis génère la valeur de E en fonction du résultat.
- Spécification de contraintes par rédaction de scénarios : il s'agit d'écrire des scénarios sous la forme d'une suite de <condition, action>. La génération de tests va être orientée vers la reproduction des séquences satisfaisant le scénario.

La spécification d'invariants est utilisable quel que soit le mode de génération choisi pour le test, alors que la spécification de probabilités conditionnelles et la spécification de scénarios sont liées respectivement à l'usage des modes de guidage par les probabilités conditionnelles et par les scénarios.

Lors de la rédaction de fichiers d'environnements, nous souhaitons tout d'abord, que Lutess génère des test réalistes. Par exemple, par des contraintes générales, nous pouvons rendre impossible la génération simultanée de plusieurs entrées vocales. Ensuite, nous souhaitons laisser une liberté contrôlée à Lutess pour la génération des jeux d'essai. Ainsi, nous choisissons d'exprimer les contraintes sur les entrées grâce au mécanisme utilisant les probabilités. Cette combinaison de contraintes générales inconditionnelles, de contraintes conditionnelles et non conditionnelles avec probabilité, permet de créer un guidage par scénario non déterministe.

Oracle et temporalité

Lutess vérifie la validité d'une propriété logique et temporelle exprimée en pseudo LUSTRE dans un fichier d'oracle. Cette propriété s'exprime sur les entrées – sortie du logiciel sous test. LUSTRE étant une logique temporelle du passé, il est impossible d'exprimer des références au futur. Notons que les références au passé s'expriment par le préfixe : "pre". L'usage du préfixe, pour une variable booléenne donnée A, permet d'avoir accès à la valeur de A au dernier passage dans le nœud LUSTRE, correspondant au dernier cycle de test. L'usage de trois préfixes permet d'avoir accès à une valeur de la variable, trois cycles auparavant.

La notion de temporalité fait référence au temps discret que représentent les cycles de test (c'est-à-dire des couples <entrée, sortie> de YellowPage). L'implication "si sortie alors entrée" traduit le fait qu'une sortie n'a jamais lieu sans l'entrée au préalable. Cette implication s'écrit par "not (sortie) or entrée". Or, nous souhaitons vérifier une application interactive multimodale utilisant un moteur à fusion ICARE, et ce dernier peut tenir compte du temps qui passe pour la gestion de la fusion des événements. Par exemple, considérons un composant ICARE qui émet une réponse à un événement ICARE dans un délai de 300 ms. Nous souhaitons exprimer que tous événements ICARE en entrée de ce composant entraînent l'émission d'un événement ICARE en sortie dans les 300 ms. La propriété que l'on souhaite exprimer dépend du temps réel du test tandis que Lutess autorise seulement l'expression de propriétés en fonction du temps discret d'un cycle de test. Il est ainsi nécessaire d'exprimer une correspondance entre le temps réel du test et le temps discret d'un cycle de test, afin de pouvoir écrire nos propriétés temps réel en fonction du temps discret.

La correspondance entre le temps réel et le temps discret n'est pas immédiate. Une proposition consiste à dire que si un test compte N cycles, alors le test dure N fois le durée d'un cycle. En d'autres termes, cela signifie que la durée d'un cycle est constante. Nous montrons ici en quoi cette proposition est erronée en l'état, et quelle solution nous définissons pour le test d'application interactive multimodale.

Si les cycles ont une durée constante, alors la durée totale du test est égale à N fois cette durée pour une vérification de N cycles. Ce qui est une condition suffisante pour exprimer des propriétés sur le temps réel à partir du temps discret n'est a priori pas valide, puisque le temps d'exécution d'applications dépend de facteur non maîtrisable : temps de génération d'un jeu d'essai, temps de réponse du logiciel sous test, activité des autres programmes en mémoire, activités du système d'exploitation, etc. De plus, nous montrons que la durée du test est différente pour chaque exécution du même jeu de test non contraint par l'environnement et sans oracle. Pour cela, nous avons instrumenté l'architecture générale de la Figure 29, en insérant une instruction enregistrant l'horloge du processeur au début de chaque cycle. Nous avons limité le nombre de cycle à 10000 sous Lutess. Ainsi, la différence entre les deux temps obtenus donne une valeur assez précise du temps total d'une vérification de 10000 cycles. Nous avons répété cette expérience plusieurs fois avec des germes de générations aléatoires différents et plusieurs fois avec des germes identiques. Le résultat de cette expérimentation

présenté dans le tableau de la **Figure 30** indique que le temps total d'une vérification de 10000 cycles varie de près de 40 % avec deux germes différents et même avec le même germe. Cela confirme que la durée d'un cycle n'est pas constante entre deux vérifications puisque qu'un cycle moyen peut mesurer de 0,47 ms à 0,68 ms en fonction du moment d'exécution du test.

	Germe aléatoire choisit			
	742		5668	
Durée (ms) / numéro	Test	cycle	Test	cycle
1	5559	0,56	5894	0,59
2	6436	0,64	6106	0,61
3	5978	0,60	5619	0,56
4	4797	0,48	4997	0,50
5	5868	0,59	6596	0,66
6	5951	0,60	5054	0,51
7	4951	0,50	5724	0,57
8	4822	0,48	6573	0,66
9	4734	0,47	5279	0,53
10	6836	0,68	5042	0,50

Figure 30 ***: tableau de durée d'un test et d'un cycle en fonction du germe aléatoire choisit.***

Cette expérimentation nous apprend néanmoins que la durée moyenne d'un cycle de test est d'environ 0,56 ms. Remarquons que ce temps est très faible, et, qu'en ajoutant une temporisation assez grande, nous pouvons la négliger. Nous ajoutons une attente de 1000 ms à chaque cycle. Cette attente est suffisamment grande pour considérer qu'un cycle de 1000,56 ms dure 1000 ms. Cette augmentation de la durée du cycle a pour effet de la rendre constante entre deux cycles d'une vérification, et également entre deux cycles de deux vérifications différentes. En effet, 1000,45 est du même ordre de grandeur que 1000,67. Il est alors possible de dire qu'un test dure N*1000 ms quel que soit le test. C'est une condition suffisante pour exprimer des propriétés sur le temps réel à partir du temps discret. Par exemple, considérons le composant de composition "Redondance/Équivalence" de la **Figure 22**. Il assure que deux entrées équivalentes sémantiquement et appartenant à une fenêtre temporelle de 20000 ms sont considérées redondantes et une seule sortie est alors produite. Pour cela, le composant émet une sortie à l'arrivée d'une première entrée et ignore les entrées suivantes si elles appartiennent à la fenêtre temporelle. Cette description du composant implique qu'une seule sortie peut être émise toutes les 20000 ms. Nous pouvons exprimer cette propriété en fonction du temps discret : si on a une sortie au cycle t, alors dans les 20 cycles précédents, il n'y a pas d'autre sortie. Cela s'écrit : not(sortie) or ZeroDansFenetreTemporelle(20, sortie) avec ZeroDansFenetreTemporelle(N, E), un nœud LUSTRE qui renvoie vrai si E est faux pendant les N derniers cycles.

Connexion entre l'application sous test et Lutess

Nous présentons ici un ensemble d'intervalles de connexion pour une application interactive multimodale développée avec des composants ICARE. La **Figure 31** présente la structure

d'une telle application, décrite selon le modèle d'architecture ARCH et intégrant un moteur de fusion ICARE en entrée. Nous retrouvons, le noyau fonctionnel, l'adaptateur de noyau fonctionnel et le contrôleur de dialogue dans la partie supérieure du schéma. Dans la partie inférieure, nous avons distingué les entrées et sorties de l'application interactive multimodale, pour faire apparaître le moteur de fusion ICARE en entrée. A la **Figure 31**, nous proposons six intervalles entrée – sortie pour la connexion de Lutess à une application interactive multimodale. Nous faisons ici abstraction de la manière dont nous connectons Lutess à l'application interactive multimodale. Les différents intervalles proposées sont l'occasion de détailler les propriétés ergonomiques que nous souhaitons vérifier, et celles que nous pouvons vérifier. Les intervalles sont :

(A) : Il s'agit de simuler en entrée un ou plusieurs composants ICARE de type dispositif, et d'étudier les sorties du moteur de fusion ICARE à destination du contrôleur de dialogue. Cet intervalle de connexion au système interactif permet de vérifier des assemblages de composants ICARE, c'est-à-dire conceptuellement l'interaction multimodale en entrée de l'application interactive.

(B) : Il s'agit ici de se connecter en entrée – sortie du contrôleur de dialogue. Cet intervalle de connexion permet de vérifier le contrôleur de dialogue.

(C) : La connexion s'effectue en entrée du contrôleur de dialogue, et en sortie au niveau de la partie interaction logique. Cela permet de vérifier l'ensemble : contrôleur de dialogue + noyau fonctionnel + adaptateur de noyau fonctionnel. En effet, nous pouvons simuler des actions utilisateurs à haut niveau d'abstraction, et observer les réponses de l'application.

(D) : La connexion en entrée – sortie s'effectue au niveau de l'adaptateur de noyau fonctionnel. Elle permet de tester le noyau fonctionnel.

(E) : Nous simulons ici des composants de type dispositif du moteur de fusion ICARE, et nous observons les sorties du contrôleur de dialogue. Cet intervalle de connexion permet de vérifier l'assemblage des composants constituant la partie interactive d'une application interactive multimodale.

(F) : Il s'agit de simuler les composants ICARE de type dispositif, et d'observer l'interaction logique en sortie. Cet intervalle permet de vérifier le comportement global de l'application.

En sortie de l'application interactive multimodale sous test, nous distinguons deux approches parmi les intervalles de connexion de la **Figure 31**. La première consiste à connecter Lutess au niveau du moteur de fusion ICARE, ce qui est le cas pour l'intervalle (A). La deuxième approche propose une connexion directe au code du logiciel sous test. C'est le cas pour les intervalles (B), (C), (D), (E), et (F). De la même manière, en entrée de l'application, nous distinguons deux propositions. La première consiste à simuler des actions utilisateurs de bas niveau : (A), (E) et (F), ou de haut niveau : (B) et (C), au travers du moteur de fusion ICARE. La deuxième consiste à simuler des entrées quelconques, à d'autres niveaux de l'application (D). Pour notre étude, nous éliminons d'ores et déjà cet intervalle (D) de connexion, car il ne correspond pas à nos objectifs de test : c'est-à-dire, simuler des entrées utilisateurs pour observer la réponse de l'application. Il ne permet d'ailleurs pas non plus de vérifier l'interaction multimodale.

Dans le temps limité de notre étude, nous avons décidé de focaliser sur les propriétés CARE dédiées à la multimodalité. Nous avons ainsi appliqué l'intervalle de connexion (A). Afin de mieux guider la génération des jeux d'essai de Lutess, nous avons prévu des retours d'information, comme la connaissance de l'affichage de la carte dans l'application YellowPage. Ces connaissances correspondent aux intervalles de connexion (B) et (E). Nous avons aussi conçu un test complet de l'application selon l'intervalle de connexion (F). Celui-

ci utilise un scénario qui peut simuler l'ensemble des entrées de l'application YellowPage, et constater l'ensemble des sorties pertinentes : carte affichée, nom et adresse spécifiés. Malheureusement, le nombre de variables booléennes explose littéralement pour un tel test, et il devient très difficile de spécifier correctement un fichier d'environnement. Ce genre de test à grande échelle atteint rapidement une autre limite : une seule propriété peut être vérifiée à la fois lors d'un test. Ainsi, pour un test de grande taille, il devient fastidieux de spécifier une propriété complexe, tenant compte de toutes les entrées dans l'oracle. Or dans le cas de test de grande taille, se limiter à des propriétés simples a peu d'intérêt puisque peu d'entrées et sorties sont alors concernées. A la suite de ces essais plutôt infructueux sur la méthodologie de test, nous avons conclu qu'il convient d'effectuer de nombreux tests de petites parties de l'application, sous peine de se noyer rapidement devant le nombre énorme de variables.

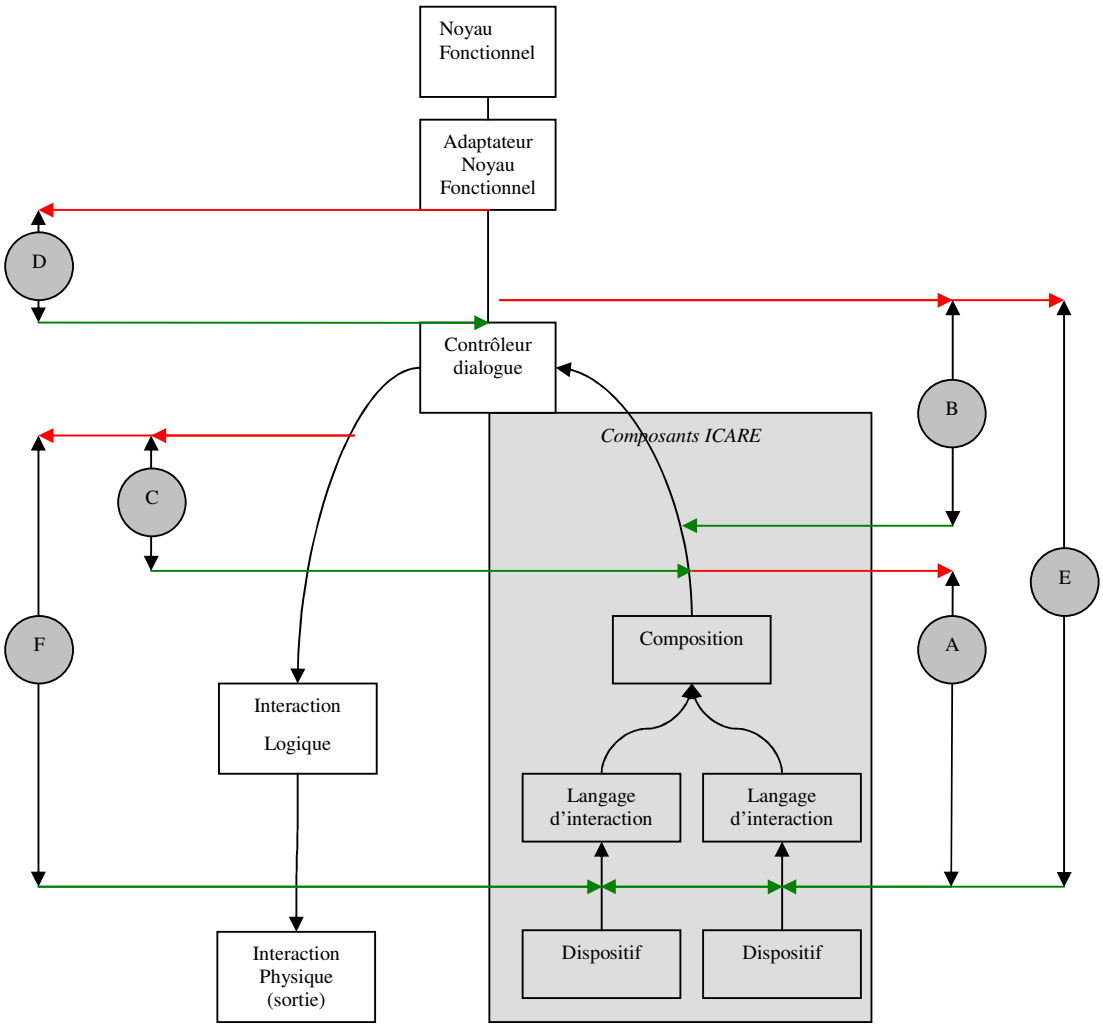

Figure 31 ***: Six intervalles entrée – sortie pour la connexion de Lutess à un système interactif multimodal construit selon le modèle ARCH et intégrant un moteur de fusion ICARE.***

Nous avons commencé à étudier quelles sont les propriétés ergonomiques qu'il serait possible de tester. Nous avons considéré uniquement les propriétés ergonomiques présentées dans [Coutaz 95]. D'autres listes et règles ergonomiques doievent être étudiées. Ceci constitue une perspective à nos travaux. Nous listons ici les propriétés que nous avons identifiées comme

candidates au test. Pour chacune d'elle, nous rappelons sa définition et expliquons comment la vérifier. Pour concrétiser le discours, nous considérons l'application YellowPage.

Atteignabilité : c'est la capacité du système à permettre à l'utilisateur de naviguer dans l'ensemble des états observables du système. Un état q est atteignable à partir d'un état p, s'il existe une suite de commande {ci} qui font passer de l'état p à l'état q.

Nous pouvons modéliser les états atteignables de l'application YellowPage. Les états peuvent être représentés par un quintuple de booléens défini comme suit :

État = (La carte est affichée, Le champ nom est sélectionné, Le champ adresse est sélectionné, Un nom est spécifié, Une adresse est spécifiée).

Les actions utilisateurs de haut niveau peuvent être à l'origine de changements d'état. Nous définissons que les transitions correspondent aux actions utilisateur qui ne sont pas des tâches de navigation et que les actions de navigation ne modifient pas l'état de l'application Nous pouvons alors représenter le comportement de YellowPage par un graphe d'états et de transitions, comme illustré en partie à la **Figure 32**. Les états sont représentés par des vecteurs de booléens dans l'ordre donné précédemment.

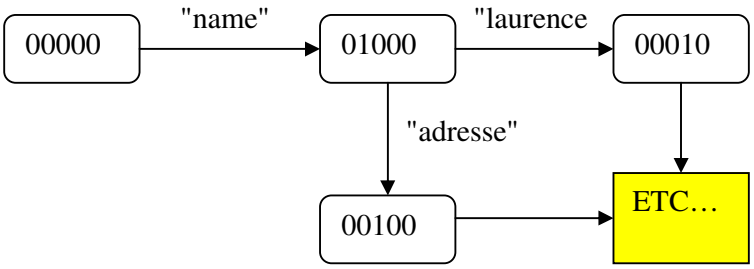

Figure 32 ***: graphe de transition représentant une partie du comportement de l'application YellowPage***

A partir d'une modélisation de YellowPage sous cette forme, il est possible d'exprimer la propriété d'atteignabilité, par exemple sous la forme d'un compteur de passage dans chaque état.

Observabilité : c'est la capacité du système à rendre perceptible l'état pertinent du système afin que l'utilisateur puisse évaluer l'état actuel du système.

Un aspect de l'observabilité est l'inspectabilité, qui représente la capacité pour l'utilisateur d'explorer l'état interne du système au moyen de commandes articulatoires. En prenant les mêmes états que pour l'atteignabilité, nous pouvons vérifier que les commandes de navigation n'engendrent pas de changement d'état. Et ceci pour tous les états.

YellowPage est plutôt simple et il est par conséquent difficile d'imaginer un état non observable. Nous pouvons néanmoins vérifier que les commandes ont bien les effets attendus sur l'état de l'application, c'est-à-dire vérifier des triplets : <état de départ, transition, état d'arrivée>. Par exemple l'action utilisateur "search" implique que pour un état où un nom et une adresse sont déjà spécifiés, un état avec une carte affichée.

Honnêteté : c'est la capacité du système à rendre observable l'état du système sous une forme conforme à cet état et qui engendre une interprétation correcte de la part de l'utilisateur.

Nous pouvons dans un premier temps vérifier la conformité de l'état des variables internes de YellowPage avec l'état publié de l'interaction. Les variables internes de YellowPage au niveau du noyau fonctionnel sont : le nom et l'adresse d'une personne, et le fait qu'une carte soit affichée ou non. Nous retrouvons des variables leur correspondant aux niveaux des objets graphiques de l'interface de YellowPage. La correspondance entre la valeur des variables du noyau fonctionnel et la valeur des attributs graphiques publiés sur l'interface graphique peut donc être vérifiée à chaque cycle.

Propriété CARE (Complémentarité, Assignation, Redondance, Équivalence) : les propriétés CARE sont détaillées dans le chapitre 4 consacré à l'interaction multimodale.

Il est possible de simuler des actions utilisateur de bas niveau, comme des clic souris en un point donnée, ou la prononciation d'un phrase. La vérification des propriétés CARE correspond à l'analyse de la fusion effectuée par les composants ICARE. Si chaque composant pris séparément a été validé, il est pertinent de vérifier les assemblages de ces composants pour savoir s'ils produisent bien l'effet prévu lors de la production du schéma. Dans ce mémoire, nous produisons les tests de ces propriétés CARE sur plusieurs assemblages de composants, dans les paragraphes **0** pour la redondance et l'équivalence, **0**, et **0** pour la complémentarité.

Nous avons proposé quelques propriétés ergonomiques qu'il est possible de vérifier avec notre méthode. Nous avons expliqué la manière de conduire le test en considérant notre application interactive multimodale YellowPage. La vérification d'une propriété donnée conduit directement au choix d'un intervalle de connexion : l'intervalle (A) pour les propriétés CARE, l'intervalle (B) pour l'atteignabilité, ou encore l'intervalle (F) pour l'honnêteté. Le choix de ces intervalles définit des points d'entrée et de sortie spécifiques pour l'application interactive multimodale. La vérification d'une propriété donnée impacte directement les points d'entrée et de sortie de l'application, mais également, ce que nous observons en ces points : la valeur d'une variable (par exemple, le nom "laurence" est-il spécifié dans le champ de texte "nom" ?) ou l'observation globale de cette variable (par exemple, un nom quelconque est-il spécifié dans le champ de texte "nom" ?).

Couplage ICARE-Lutess

Tandis que le chapitre précédent expose les étapes de notre méthode d'évaluation formelle de propriétés ergonomiques, ce chapitre est consacré à la connexion entre l'application sous test structurée en composants ICARE et Lutess qu'il est nécessaire de réaliser pour effectuer des tests : le couplage ICARE-Lutess.

Nous énonçons d'abord la problématique concernant la connexion d'une application interactive à Lutess, puis nous expliquons notre proposition de couplage ICARE – Lutess. Nous présentons ensuite des résultats de test produits grâce à l'implémentation manuelle de notre couplage sur la première version de YellowPage. Suite à cette étude de faisabilité (approche manuelle), nous expliquons l'outil de génération automatique d'instrumentation que nous avons développé. Enfin, nous montrons l'exploitation de cet outil sur la deuxième version de YellowPage.

Nécessité de la connexion à Lutess

Le chapitre précédent explique les étapes de notre méthode d'évaluation qui implique que l'application sous test soit connectée à Lutess. Plusieurs intervalles de connexion entre l'application sous test et Lutess sont décrits et leur choix dépend de la propriété ergonomique à vérifier. Nous détaillons dans ce chapitre comment réaliser cette connexion.

Nous rappelons que Lutess communique vers l'extérieur par la lecture – écriture de booléen sur l'entrée – sortie standard. La connexion de l'application sous test à Lutess implique donc le développement de deux mécanismes du côté de l'application sous test.

- Un mécanisme de communication qui implémente la lecture – écriture de vecteurs de booléens sur l'entrée – sortie standard du côté de l'application sous test.
- Un mécanisme de traduction de la sémantique en entrée et en sortie qui permette à Lutess et à l'application sous test de se comprendre. Ce mécanisme dépend des entrées – sorties choisies pour être simulées et observées sur l'application sous test. Ces entrées – sorties dépendent elles-mêmes des propriétés à vérifier.

Le premier mécanisme peut être générique, puisqu'il n'est pas impacté directement par l'application sous test, mais dépend exclusivement de l'architecture de communication de Lutess. Le second mécanisme nécessite une solution adaptée aux entrées – sorties que nous souhaitons simulées – observées sur l'application sous test. Ce point entraîne une discussion sur le choix des entrées – sorties de l'application sous test.

Approche de connexion YellowPage - Lutess

Dans cette section, nous présentons notre étude préliminaire pour la connexion de Lutess à une application interactive multimodale. Afin de pouvoir instrumenter celle-ci de façon manuelle, pour proposer une connexion acceptable pour effectuer des tests, nous avons exploré les possibilités sur la première version de l'application interactive multimodale YellowPage. Rappelons que celle-ci est développée sans l'éditeur de schéma ICARE, et que la structure du schéma est explicitement implémentée dans une Classe JAVA.

Différentes possibilités de connexion

Au vue de la structure de la classe JAVA implémentant le schéma ICARE en entrée, présenté au chapitre 8, plusieurs propositions sont possibles pour réaliser le mécanisme de connexion

entre l'application interactive multimodale YellowPage et Lutess. Nous rappelons ici succinctement notre objectif, puis trois solutions envisagées et mises à l'écart.

Notre objectif pour la vérification est de simuler des entrées de l'utilisateur, et de vérifier si le comportement de l'application interactive multimodale est adéquate au regard de certaines propriétés ergonomiques. Dans ce sens, il est intéressant de pouvoir simuler des actions utilisateur de bas niveau, tel un clic souris. D'autre part, la simulation d'événements de plus haut niveau, par exemple, la commande "zoom in point(x,y)", spécifiée par une commande vocale accompagnée d'un geste de désignation peut également être pertinente.

Une première solution consiste à considérer l'application interactive multimodale comme un tout, c'est-à-dire ne pas prendre en compte le fait qu'une partie de l'application soit développée avec ICARE. Ainsi, l'utilisateur a le choix de simuler et d'observer de manière fine n'importe quelle variable des différentes classes de l'application. Cette solution, si elle permet de couvrir tous les points de l'application interactive, ne permet pas de guider le choix de la position des points d'entrée – sortie de l'application. En effet, le grand nombre de variables rend difficile un choix pertinent des entrées - sorties. D'autre part, cette solution est très fortement liée à l'implémentation des composants ICARE et du reste l'application interactive. Le changement d'un composant ICARE par une version supérieure peut remettre en question l'instrumentation. Cette solution est à rejeter dans l'absolu, mais, elle sera malgré tout utilisée en partie en pratique, car elle est un moyen efficace pour obtenir une variable donnée, dans les tréfonds des classes composant l'application.

Une deuxième solution peut être envisagée pour la connexion de l'application interactive à Lutess. Il s'agit cette fois d'ajouter une interface vers l'extérieur, aux différents composants qui composent l'application. L'implémentation de cette solution peut prendre plusieurs formes : la première est de concevoir une interface JAVA, proposant les entêtes de méthodes pour le mécanisme de traduction et le mécanisme de lecture – écriture sur l'entrée – sortie standard, que l'ensemble des composants ICARE doivent implémentés. L'implémentation de cette interface au sein des composants du mécanisme de lecture – écriture, ne soulève pas à priori de problème, sauf dans l'ordonnancement des différents composants, cette question pouvant être traitée par ailleurs. Néanmoins, la partie traduction est problématique car les composants ICARE n'ont pas obligatoirement la connaisance ni de la syntaxe ni de la sémantique des événements ICARE qu'ils reçoivent et transmettent. C'est le cas notamment des composants de composition, qui sont totalement génériques. Cette démarche implique également l'insertion dans les différents composants de code inutile pour leur usage normal. De plus, elle propose en réalité de reprendre l'ensemble des composants ICARE déjà produits, afin de leurs adjoindre l'implémentation de l'interface de communication.

Cette deuxième solution, qui consiste à ajouter une interface de communication vers l'extérieur aux différents composants ICARE, peut être envisagée sous l'angle de la programmation par aspect. Ce paradigme considère que les services d'une application et ses propriétés non fonctionnelles correspondent à des aspects qui doivent être découplés les uns des autres. L'application est obtenue par la composition ("assemblage") de ces différents aspects. Pour le côté technique de la programmation orientée aspects, nous pouvons retenir une approche par transformation de programme AspectJ pour les raisons suivantes : facilité d'apprentissage et de démonstration, simplicité de la définition des aspects. Avec AspectJ, un aspect est avant tout une classe. Il conserve les possibilités d'une classe Java (abstraction, héritage, interface, etc.). Tout comme une classe, il comporte un état, l'ensemble de ses attributs, et un comportement, l'ensemble des actions effectuées quand ses méthodes sont appelées. Pour définir les propriétés d'intersection des aspects, AspectJ étend le langage Java avec de nouveau mots clés. Cette extension nécessite donc l'utilisation d'un compilateur

spécifique chargé d'effectuer la composition des aspects et de restituer un byte-code interprétable par une machine virtuelle Java standard : c'est la phase de tissage illustrée à la **Figure 33**.

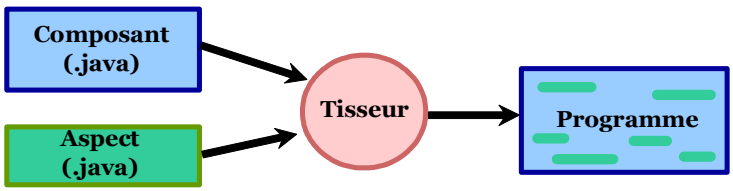

Figure 33 ***: phase de tissage avec AspectJ***

Avec AspectJ, nous pouvons par exemple ajouter des variables et des méthodes dans les classes de l'application YellowPage qui permettent de lire les entrées générées par Lutess. De même, le mécanisme d'interception de méthodes, peut permettre de renvoyer les réactions du système vers Lutess. Cette solution intéressante dans sa mise en œuvre n'est néanmoins pas dépourvue de défaut. En effet, il n'est pas plus facile de décider de la pertinence d'une variable à simuler que dans la première solution de connexion proposée. Elle ajoute en plus la possibilité d'observer les appels des méthodes des différentes classes. Cet ajout qui peut être souhaitable pour correctement tester l'application interactive engendre une indécision encore plus grande sur ce que doivent être les points d'entrée et de sortie de l'application pour un test.

Une troisième solution moins ambitieuse peut être envisagée. Elle est dépendante de l'éditeur graphique de schéma ICARE. Il s'agit de tirer partie de l'interfaçage entre ICARE et le reste de l'application interactive multimodale. En effet, une classe "ICARE.java" est produite lors de la génération du code du schéma ICARE comme cela est décrit dans le chapitre 8. La classe "ICARE.java" contient plusieurs méthodes qui réalisent une indirection depuis la partie ICARE, vers la partie non – ICARE de l'application interactive multimodale. Cette solution consiste à déconnecter la partie ICARE à l'exécution, à l'exclusion de cette classe. Celle-ci aurait été préalablement instrumentée afin de permettre l'interface avec Lutess en entrée. Cette solution présente peu d'intérêt utilisée seule. En effet, elle permet uniquement de créer l'interface en entrée du logiciel sous test vers Lutess, et seulement à haut niveau d'abstraction pour les entrées. Son apport peut être néanmoins plus intéressant à condition de l'utiliser conjointement avec la première solution de connexion proposée. Le défaut qui justifie son écartement provient de l'implémentation réelle des composants ICARE. En effet, le modèle ICARE rend possible l'encapsulation d'éléments de l'interface graphique. C'est le cas pour la deuxième version de YellowPage (chapitre 8). Cette méthode est donc inapplicable dans notre cas, puisqu'elle nécessite de déconnecter une grande partie de l'interface utilisateur.

Notre solution de connexion

Les méthodes de connexion proposées précédemment sont soit incomplètes comme la première et la troisième, soit au contraire trop complète et complexe comme l'utilisation d'AspectJ. Nous proposons une solution plus large dans son utilisation que la première ou la troisième proposition, et plus générique et précise que la solution avec AspectJ. Notre proposition tire partie de la structuration produite par l'utilisation d'un moteur de fusion d'événement multimodaux ICARE. En effet, la communication entre composants s'effectue

au moyen d'événements ICAREEvents qui sont échangés au sein d'un schéma ICARE depuis les composants de type Dispositif (niveau inférieur d'un schéma ICARE) jusqu'aux tâches ou commandes (niveau supérieur d'un schéma ICARE). Chaque ICAREEvent porte une sémantique propre correspondant à un niveau d'abstraction dans la fonction qui abstrait les actions de l'utilisateur en tâches ou commandes traitées par l'application interactive. Pour simuler une entrée de l'utilisateur, nous créons une instance d'un ICAREEvent, puis nous l'envoyons à un composant ICARE du schéma. Nous pouvons ainsi simuler le comportement d'un composant C, en émettant des ICAREEvents à destination des composants cibles de C. De manière symétrique, en sortie, nous observons les ICAREEvents qui circulent entre les composants du schéma ICARE. Cette approche limite de fait le nombre possible de points d'entrée simulables et de points de sortie observables de l'application interactive multimodale. En effet, le nombre de points en entrée et en sortie est restreint au nombre de liens entre les composants dans le schéma ICARE (représentés par des flèches unidirectionnelles).

Nous implémentons cette approche sous la forme suivante : nous écrivons une classe "ICARELutess.java". Cette classe contient une méthode modélisant l'adaptation à la communication de Lutess. Les étapes 1 et 4 de l'adaptateur, décrites ci-dessous correspondent aux phases 3 et 4 du déroulement d'un cycle de test illustré à la **Figure 29**.

1. Lecture des vecteurs de booléens sur l'entrée standard.
2. Appel des méthodes de traduction.
3. Attente de 1000 ms.
4. Écriture de la traduction booléenne des ICAREEvents observés sur la sortie standard.

Les étapes 2 et 4 comprennent les étapes de traduction de la sémantique des événements qui permettent à Lutess et à YellowPage de se comprendre. L'attente de 1000 ms, comme nous l'avons précisée à la section **0** permet de stabiliser la durée d'un cycle de simulation de manière à permettre l'expression de propriétés temporelles sur le temps réel à partir du temps discret.

La structure de la connexion en entrée et en sortie est représentée à la **Figure 34**. Cette figure reprend une partie du schéma ICARE des commandes simples de YellowPage version 1 illustré à la **Figure 22**. Toutes les méthodes de la **Figure 34** sont contenues dans la classe "ICARE.java". En entrée, nous écrivons un ensemble de méthodes de traduction, une par sortie de composant dans le schéma ICARE, de la forme de celle illustrée en (a). Cette méthode prend en paramètre un tableau de booléens. Elle crée une instance événement ICARE, traduit le tableau de booléens en une sémantique pour cet événement, puis émet l'événement au composant cible (le composant de langage d'interaction "SimpleCommand2") du composant simulé (le composant de dispositif "Microphone"). A l'exécution, le composant "SimpleCommand2" peut recevoir des événements simulés, comme s'il provenait du composant "Microphone". L'implémentation de la sortie est réalisée en 3 étapes. Tout d'abord, nous ajoutons à l'écouteur d'événements provenant de "Microphone", un appel à une méthode de traduction en sortie (b), avant d'émettre l'événement ICARE au composant cible (b). Ensuite, la méthode appelée réalise la traduction de la sémantique de l'événement sous la forme d'un tableau de booléen (c). Celui-ci est stocké comme un attribut de la classe "ICARE.java" (d). Enfin, l'adaptateur appelle les méthodes qui récupèrent la valeur de ces attributs booléens stockés (e). Ce sont ces tableaux de booléens qui sont écrits sur la sortie standard pour être lus par Lutess.

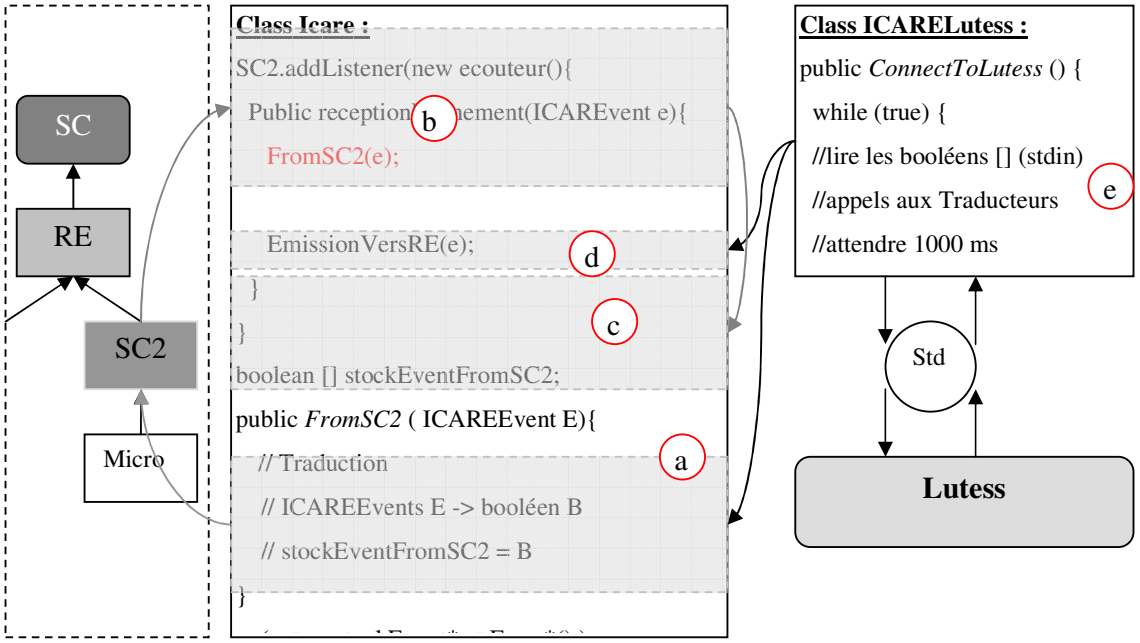

Figure 34 ***: représentation de la partie traduction de l'instrumentation en entrée et en sortie***

Notre approche permet d'effectuer des tests en simulant des entrées de l'utilisateur de l'application interactive multimodale à différents niveaux d'abstraction. Il est donc possible de simuler des actions physiques de l'utilisateur (Dispositif) et des actions de plus haut niveau d'abstraction (Langage d'Interaction et Composition). Évaluons notre approche par rapport aux propositions précédentes. D'une part, cette approche identifie clairement les points de connexion en entrée et en sortie, puisqu'il s'agit de simuler / observer les événements ICARE pouvant circuler dans un schéma ICARE. En cela, elle est plus utilisable que les deux premières solutions proposées. D'autre part, elle permet de simuler l'ensemble des composants ICARE présents dans le schéma, y compris les composants "tâches abstraites". Elle recouvre donc la troisième proposition. Notons que cette méthode est systématique : les méthodes nécessaires pour l'instrumentation sont toutes construites sur le modèle présenté. Enfin, cette approche de connexion permet de vérifier des propriétés comme CARE, puisque d'un côté, nous simulons des entrées de l'utilisateur, et de l'autre, nous observons les commandes combinées par le moteur de fusion. Cette méthode de connexion n'est cependant pas dépourvue de défaut : en effet, elle ne permet pas en l'état de placer des points d'entrée et de sortie au-delà des limites de la partie ICARE. Cela ne représente pas un inconvénient majeur en entrée, puisque nous souhaitons simuler des entrées multimodales d'utilisateur. En revanche, en sortie, il peut être intéressant d'observer des états internes du noyau fonctionnel, du contrôleur de dialogue, et de l'interface en sortie de l'application. Nous proposons dans ce cas d'exploiter la première proposition, c'est-à-dire de décider au cas par cas, les points supplémentaires de sortie qui sont nécessaires pour un test. Dans certains de nos tests effectués, nous avons observé par exemple l'état d'affichage de la carte de YellowPage. Cette utilisation conjointe d'une part de notre proposition systématique de connexion, et d'autre part de la sélection au cas par cas de variables précises à observer permet d'envisager la vérification de propriétés ergonomiques comme l'observabilité ou l'honnêteté [Abowd 92] selon la manière présentée au paragraphe **0**.

Réalisation Manuelle sur YellowPage v1

Nous avons appliqué cette méthode de connexion en entrée et en sortie à la première version de YellowPage. Nous avons suivi la manière d'implémenter la connexion proposée au paragraphe précédent. La **Figure 35** montre l'implémentation de l'instrumentation proposée au sein de la classe "Icare.java" pour l'application interactive multimodale YellowPage v1. Elle présente le code concernant la connexion pour le cas de "commandes vocales simples", spécifiées selon la modalité définie par le couple de composants ICARE "microphone" et "Simple Command 2". La **Figure 35** présente en noir le code original de l'application YellowPage, et en bleu le code de l'instrumentation. Nous observons en (a), l'appel de la méthode de traduction (g) en sortie de YellowPage. Cet appel engendre le remplissage du tableau de booléens correspondant (e). Ce vecteur peut être récupéré par l'adaptateur au moyen de la méthode (f). Pour les entrées, la méthode (b) opère la traduction vers une sémantique d'événements ICARE (c) des vecteurs de booléens produits par Lutess, et envoie l'événement ICARE produit au composant cible (d).

La **Figure 36** montre l'implémentation de l'adaptateur proposé. Nous retrouvons dans ce code la méthode qui gère la communication vers Lutess par l'intermédiaire de l'entrée – sortie standard (a). Cette méthode est composée des 4 étapes annoncées dans notre approche de connexion : lecture du vecteur de test, appel du traducteur, attente, puis écrire des résultats. Les méthodes (b) et (c) servent respectivement à effacer le vecteur de booléens utilisé pour la lecture, et à effectuer cette lecture. Le code correspondant à ces méthodes n'est pas détaillé ici.

```
public class Icare {

 //declaration du composant de dispositif "microphone"
 viaVoiceInput.ViaVoiceInput vvi;

 //declaration du composant de langage d'interaction "simple command 2"
 SimpleCommand.SimpleCommand sc2;

 // instanciation du "microphone"
 vvi = new viaVoiceInput.ViaVoiceInput();

 //attibution du fichier de configuration du "microphone"
 vvi.setFile("myMappy.gram");

 // ajout de l'ecouteur du "microphone qui emmet à "simple command 2
 vvi.getViaVoice().addViaVoiceListener(new viaVoiceInput.ViaVoiceListener() {
    public void newViaVoiceData(ICARE.ICAREEvent data_values) {
      RemplirTabDeviceMicrophoneITSimpleCommand2C(date_values.getVector()); (a)
      sc2.setData(data_values);
    }
 });

 //instanciation du composant "simple command 2"
 sc2 = new SimpleCommand.SimpleCommand();

 // attribution du fichier de configuration de "simple command 2"
 sc2.setCommandsFile(chemincomplet+"simplecommands.ica");

 // methode creant et envoyant un évenement ICARE pour la simulation (b)
 public void HaveDeviceVoiceITSimpleCommand2 (boolean[] bool, long initial, long time) {
    ICARE.ICAREEvent event = new ICARE.ICAREEvent (this);
    Vector v = new Vector();

    if (bool[0] )  v.addElement(new String("zoom in "));
    if (bool[1] )  v.addElement(new String("zoom out "));
    if (bool[2] )  v.addElement(new String("up "));
    if (bool[3] )  v.addElement(new String("down "));
    if (bool[4] )  v.addElement(new String("left "));
    if (bool[5] )  v.addElement(new String("right "));          (c)
    if (bool[6] )  v.addElement(new String("move up "));
    if (bool[7] )  v.addElement(new String("move down "));
    if (bool[8] )  v.addElement(new String("move left "));
    if (bool[9] )  v.addElement(new String("move right "));
    if (bool[10] ) v.addElement(new String("center "));
    if (bool[11] ) v.addElement(new String("search "));

    event.setVector(v);
    event.setTime(time);
    event.setInitialTime(initial);
    event.setConfidenceFactor(1);
    event.setNbOfValues(1);

    sc2.fireNewEvent(event); [c] (d)
 }

 // boolean de stockage en sortie
 boolean[] TabDeviceMicrophoneITSimpleCommand2 = new boolean[12]; (e)

 // getter pour la récuperation depuis l'adaptateur
 public boolean[] GetTabDeviceMicrophoneITSimpleCommand2 () {
    return TabITSimpleCommand2CFRedondancyEquivalence;          (f)
 }

 // traducteur en sortie de ICARE
 private void RemplirTabDeviceMicrophoneITSimpleCommand2 (Vector v ){ (g)
   String s = (String)v.elementAt(0);
   if ( s.equals("zoom in "))  TabITSimpleCommand2CFRedondancyEquivalence[0] = true ;
   if ( s.equals("zoom out "))  TabITSimpleCommand2CFRedondancyEquivalence[1] = true ;
   if ( s.equals("up ") )  TabITSimpleCommand2CFRedondancyEquivalence[2] = true ;
   if ( s.equals("down") )  TabITSimpleCommand2CFRedondancyEquivalence[3] = true ;
   if ( s.equals("left") )  TabITSimpleCommand2CFRedondancyEquivalence[4] = true ;
   if ( s.equals("right") ) TabITSimpleCommand2CFRedondancyEquivalence[5] = true ;
   if ( s.equals("move up  ") ) TabITSimpleCommand2CFRedondancyEquivalence[6] = true ;
   if ( s.equals("move down ") ) TabITSimpleCommand2CFRedondancyEquivalence[7] = true ;
   if ( s.equals("move left ") ) TabITSimpleCommand2CFRedondancyEquivalence[8] = true ;
   if ( s.equals("move right ") )TabITSimpleCommand2CFRedondancyEquivalence[9] = true ;
   if ( s.equals("center ")) TabITSimpleCommand2CFRedondancyEquivalence[10] = true ;
   if ( s.equals("search "))  TabITSimpleCommand2CFRedondancyEquivalence[11] = true ;
 }
}
```

Figure 35 ***: partie du code de la connexion YellowPage - Lutess***

```
public class YellowPageLutess implements java.lang.Runnable{

    //vecteur de booléen pour la lecture des entrées
    boolean[] DeviceVoiceITSimpleCommand2;

  // modélise l'adapatation en Lutess et ICARE
  private void ConnexionSimpleCommand2(){        a
    long eventT,initT;

    // Boucle qui modèlise un cycle d'éxécution
    while (true) {
        // initialisation
        icare.initializeSortie();
        this.DeviceMicrophoneITSimpleCommand2 = VecteurVide(l2);

     // 1. Lecture des Entrees
        this.readTabBoolean(DeviceMicrophoneITSimpleCommand2,l2);
        // Recuperation du temps processeur pour les estapilles temporelles
        initT = System.currentTimeMillis() ;
        eventT = System.currentTimeMillis();

     // 2. Appel de la methode de traduction
        if (this.ExactementUn(DeviceMicrophoneITSimpleCommand2,l2) == true) {
           icare.HaveDeviceMicrophoneITSimpleCommand2(DeviceMicrophoneITSimpleCommand2,initT,eventT);

     // 3. Attente pour un cylce de durée constante
        try { Thread.currentThread().sleep(1000) ;
        } catch (Exception except) {}

     // 4. Ecriture des resultats
        this.writeTabBoolean(icare.GetTabDeviceMicrophoneITSimpleCommand2(),l2);
    }
  }

  //renvoi un vecteur de n booléen à faux
  private boolean[] VecteurVide ( int taille) ;    b

  // remplit un le tableau tabBool de n boolean lut sur l'entrée standard    c
  static  void readTabBoolean (boolean[] tabBool, int n) ;
}
```

Figure 36 ***: partie de code de l'adaptateur pour la connexion YellowPage – Lutess***

Quelques tests sur YellowPage v1

La méthode de test définie et la connexion de YellowPage v1 à Lutess établie grâce à notre approche de couplage ICARE – Lutess, nous proposons plusieurs tests effectués sur l'application YellowPage v1. Le couplage ICARE – Lutess manuelle sur YellowPage v1 nous a pris environ 5 heures à écrire. Il a impliqué 3 jours afin de vérifier sa correction syntaxique et sémantique. Nous montrons par les différentes expérimentations proposées, l'adéquation de notre couplage ICARE – Lutess à la vérification de propriétés ergonomiques pour la multimodalité, c'est pourquoi, nous focalisons sur le test d'assemblages de composants produisant la Redondance, l'Équivalence et la Complémentarité au sens des propriétés CARE. Cette partie montre par ailleurs comment nous adaptons ces propriétés à notre application YellowPage par leur description en LUSTRE. Nous exposons aussi concrètement la manière de décrire des fichiers d'environnements et d'oracles en LUSTRE.

Assemblage en redondance/équivalence

Nous testons ici la redondance et l'équivalence des entrées de l'application interactive multimodale YellowPage v1 pour les commandes simples sans paramètre. La **Figure 37** illustre le niveau de connexion de Lutess à ICARE par la représentation de la "boîte noire". En entrées, nous simulons les composants "dispositifs" ICARE. En sortie, nous observons les événements ICARE avant leur arrivée au contrôleur de dialogue de YellowPage.

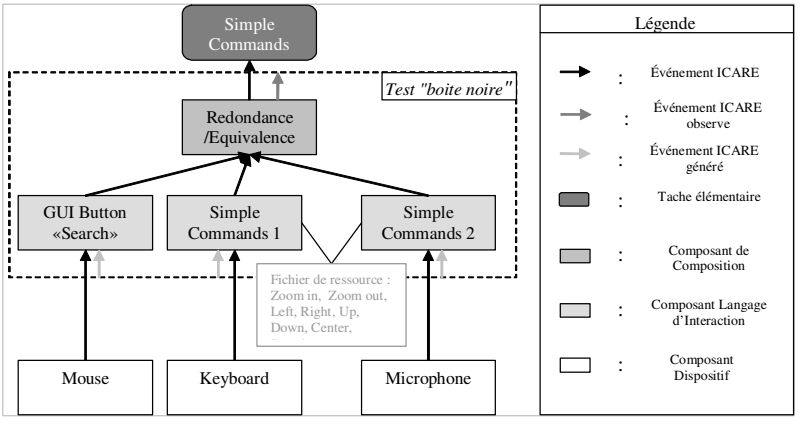

Figure 37 ***: Connexion en entrée - sortie pour le test de l'assemblage en redondance/équivalence***

Fichier d'environnement

Pour ce test, Lutess doit générer des vecteurs de test correspondant à des simulations des composant de dispositif "Mouse", "Keyboard", "Microphone". Un sous ensemble des événements que peut produire ces dispositifs nous intéresse pour ce test. Nous décrivons d'abord ces restrictions : le dispositif souris émet des l'événements : clic sur le bouton "search" de l'interface graphique. Le clavier émet des événements de type "touche <numéro> enfoncée", et le microphone émet des événements correspondant aux commandes vocales suivantes : "zoom in", "zoom out", "up", "down", "left", "right", "move up", "move down", "move left", "move right", "center", "search". Précisons que nous débutons le test dans un état où un nom et une adresse sont déjà spécifiés dans l'application YellowPage.

La **Figure 38** montre le code LUSTRE du fichier environnement que nous avons écrit. Les noms des entrées et des sorties ont été numérotés pour simplifier la lecture du fichier, cependant cela rend plus complexe sa compréhension. La correspondance est la suivante :

- Soit e1 : booléen // l'événement d'entrée clic sur le bouton "search" de l'interface graphique.
- Soit e2 : booléen[8] // les événements d'entrées touches <x> clavier avec <x> dans l'ordre : "Page Up", "Page down", "Flèche haut", "Flèche Bas", "Flèche Gauche", "Flèche Droite", "Espace", "Enter".
- Soit e3 : booléen[12] // les événements entrées vocales dans l'ordre : "zoom in", "zoom out", "up", "down", "left", "right", "move up", "move down", "move left", "move right", "center", "search".
- Soit s1 : booléen // sortie vraie si le plan est affiché.
- Soit s2 : booléen[8] // les événements en sortie du composant "Redondance/Équivalence", signifiant dans l'ordre : "Zoom in", "Zoom out", "Up", "Down", "Left", "Right", "Center", "Search".

Le code du noeud LUSTRE environnement de la **Figure 38**, possède dans son entête (a), la description des entrées – sorties de l'application interactive multimodale YellowPage. Nous imposons deux contraintes à Lutess pour la génération en mode guidé par les probabilités : la première est une contrainte générale, qui précise qu'un dispositif ne peut produire deux entrées durant un cycle de test. Cette contrainte est matérialisée par l'invariant (b). La deuxième contrainte est un scénario par probabilité qui rend plus probable la génération de certaines entrées en fonction de l'affichage de la carte (c). En effet, nous attribuons une probabilité de 0.05 d'avoir une entrée signifiant une commande de recherche (souris bouton « search », clavier <Enter>, et Microphone « Search ») tant qu'une carte n'est pas affichée, et

de 0 lorsque la carte est affichée. Nous attribuons une probabilité de 0.5 à un événement "page up" Clavier, "zoom in" à la voix, afin d'une part qu'il n'ait pas lieu trop souvent pour produire une trace résultante claire, et d'autre part car la restriction à la commande "Zoom In" suffit à produire des résultats montrant la redondance et l'équivalence . Ce fichier environnement permet d'obtenir un événement de recherche dans les premiers cycles de simulation, suivi par des événements de navigation "Zoom In" lorsque la carte est affichée.

```
node environment (s1 : bool ; s2 : bool^8) returns (e1 : bool ; e2 : bool^8 ; e3 : bool^12); a
let

environment(  AuPlusUn(1, E1) and AuPlusUn(8, E2) and AuPlusUn(12, E3) ) ; b

proba(
-- si carte pas afficher    c
  (e1_0 ,0 ,pre not s1),

  (e2_0 ,0 ,pre not s1),
  [...]
  (e2_6 ,0 ,pre not s1),
  (e2_7 ,0.05 ,pre not s1),

  (e3_0 ,0 ,pre not s1),
  [...]
  (e3_10,0 ,pre not s1),
  (e3_11,0.05 ,pre not s1),
-- si carte afficher
  (e1_0 ,0 ,pre  s1),

  (e2_0 ,0.5 ,pre  s1),
  (e2_1 ,0 ,pre  s1),
  [...]
  (e2_7 ,0 ,pre  s1),

  (e3_0 ,0.5 ,pre  s1),
  (e3_1 ,0 ,pre  s1),
  [...]
  (e3_11,0 ,pre  s1));
tel
```

Figure 38 ***: environnement pour le test de redondance équivalence***

Fichier d'oracle

Nous vérifions d'une part, l'équivalence des entrées provenant des trois dispositifs simulés, et d'autre part la redondance des entrées équivalentes. Notons que la fenêtre temporelle du composant Redondance/Équivalence est de 10000 ms. Il est possible d'exprimer le comportement attendu de la redondance de l'assemblage pour une commande donnée. Par exemple, pour "Zoom In", si l'une des deux entrées équivalentes est vraie (clavier "page down" et voix "zoom in"), alors la sortie "Zoom In" doit être vraie, sauf si elle l'a été dans les 10 derniers cycles (10 cycles = fenêtre de 10000ms). Cela se factorise en : si l'une des entrées équivalentes est vraie, alors dans les 10 derniers cycles, il doit y avoir exactement une sortie vraie correspondant à ces entrées. Nous écrivons cette propriété sous la forme d'un nœud LUSTRE "RE1" décrit en (a) de la **Figure 39**. Afin d'exprimer la redondance/équivalence, nous appliquons le nœud "RE1" à toutes les entrées équivalentes au sein du nœud LUSTRE oracle (b), en faisant un OU logique sur les entrées (c). Par exemple, "prop[0] = RE1(10, se2[0], e2[0] or e3[0]) ; " exprime l'utilisation du nœud RE1 pour tester la redondance/équivalence des entrées "zoom in" voix, et "page down" clavier.

```
node RE1(const DT :int ;sortie, entree :bool)returns(ok:bool)
let
  ok = not(entree) or ExactementUnDansFenetre(DT, sortie  ) ;
tel

node oracle (e1 : bool^1 ; e2 : bool^8 ; e3 : bool^12 ; s1 :  bool ; s2 : bool^8 ) returns( ok: bool);

  var  prop : bool^8;
let
  prop[0] = RE1(10, s2[0] ,e2[0] or e3[0] );
  prop[1] = RE1(10, s2[1] ,e2[1] or e3[1] );
  prop[2] = RE1(10, s2[2] ,e2[2] or e3[2] or e3[6] );
  prop[3] = RE1(10, s2[3] ,e2[3] or e3[3] or e3[7] );
  prop[4] = RE1(10, s2[4] ,e2[4] or e3[4] or e3[8] );
  prop[5] = RE1(10, s2[5] ,e2[5] or e3[5] or e3[9] );
  prop[6] = RE1(10, s2[6] ,e2[6] or e3[10] );
  prop[7] = RE1(10, s2[7] ,e1[0] or e2[7] or e3[11]);
  ok = LIN_AND(8 , prop);
tel
```

a, b, c

Figure 39 ***: oracle pour le test de redondance/équivalence***

Résultats des tests.

Nous avons effectué plusieurs vérifications consécutives sans détecter de viol de cette propriété. Le **Figure 40**, montre une partie des résultats obtenus sous la forme d'un tableau. Le choix de cette forme de présentation provient du fait que les fichiers résultats d'un test sont difficiles à lire et à comprendre. Nous garantissons néanmoins la conformité de ce qui est présenté dans les tableaux par rapport à ce qui est présent dans le fichier de résultat. Les cycles du test sont numérotés dans la colonne de gauche. Toutes les entrées – sorties vraies pendant un cycle sont présentées respectivement dans les colonnes "Entrées" et "Sorties". La colonne "Oracle" indique la valeur de la propriété de l'oracle à chaque cycle. Dans le tableau de la **Figure 40**, nous utilisons des préfixes pour indiquer la provenance des entrées – sorties vraies lors des cycles. Le préfixe "V-" pour la voix, le préfixe "S-" pour la souris, le préfixe "C-" pour le clavier, et "R-" pour la sortie de Redondance/Équivalence. L'affichage de la carte est précisé par code "IMG".

Cycle	Entrées	Sorties	Oracle
10	-	-	vrai
11	V-search	IMG, R-search	vrai
12	C-zoomin	IMG, R-zoomin	vrai
13	V-zoomin	IMG	vrai
14	V-zoomin, C-zoomin	IMG	vrai
15	-	IMG	vrai
16	V-search	IMG	vrai
17	C-zoomin	IMG	vrai
18	C-zoomin	IMG	vrai
19	C-zoomin	IMG	vrai
21	-	IMG	vrai
22	C-zoomin	IMG, R-zoomin	vrai
42	-	IMG	vrai
43	V-zoomin	IMG, R-zoomin	vrai

Figure 40 : ***tableau de synthèse des résultats partiels d'un test pour la redondance/équivalence***

Détaillons les résultats de la **Figure 40**. Au cycle 11, l'entrée vocale "search" provoque la commande "Search" en sortie. Cela indique que la commande vocale "search" a été envoyée au contrôleur de dialogue. La carte est affichée, ce qui implique que la commande "search" a été prise en compte par YellowPage. Au cycle 12, on observe une entrée clavier "zoom in", suivie de la réponse "zoom in" en sortie de la boîte noire. Durant les cycles 13 à 21, plusieurs commandes vocales et clavier "zoom in" sont produites, et ne provoquent aucune réponse de YellowPage. Ce comportement est normal, car pendant les 10 cycles suivant une commande "zoom in" en sortie, aucune entrée "zoom in" ne doit être prise en compte. Après ces 10 cycles qui garantissent à l'utilisateur la redondance des commandes entrées, nous observons au cycle 22 une entrée clavier "zoom in" qui déclenche la réponse "zoom in". Nous observons que les deux entrées clavier et voix sont équivalentes au cycle 43, où la sortie "zoom in" est déclenchée suite à l'entrée vocale "zoom in" alors que jusqu'à là, seules des commandes clavier "zoom in" avaient déclenché une réponse (ceci étant dû à la génération par probabilités conditionnelles).

Assemblage en complémentarité

Nous testons ici la complémentarité des modalités "clavier", "souris" et "clavier", au niveau de la boîte noire illustrée dans la **Figure 41**. Nous simulons en entrée des actions de l'utilisateur en simulant les sorties des composants dispositifs du schéma ICARE correspondant à la sélection et au remplissage des champs nom et adresse. Nous observons en sortie du système, l'entrée du contrôleur de dialogue "Specify Name or Address".

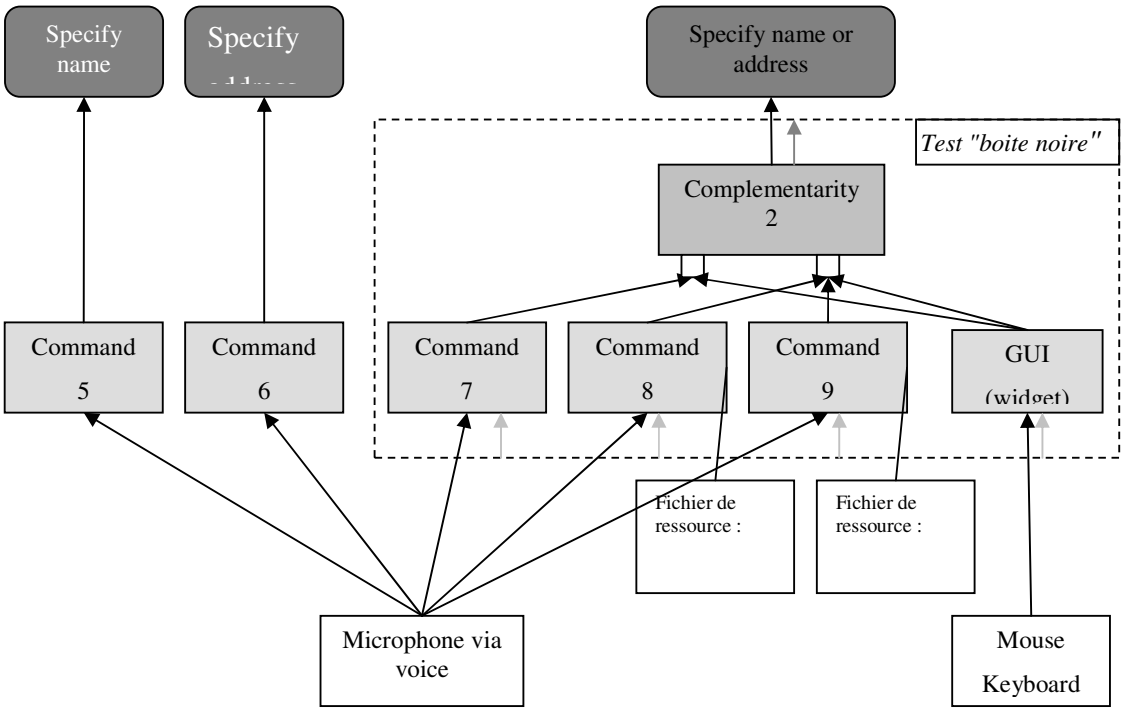

Figure 41 ***: connexion en entrée – sortie pour le test de l'assemblage en complémentarité***

Fichier d'environnement

Nous rappelons que la spécification de YellowPage permet à l'utilisateur de sélectionner un champ par un clic à la souris, et vocalement en prononçant "name" ou "address". L'utilisateur

remplit ensuite le champ sectionné, soit au moyen du clavier, soit en parole. Nous simulons les composants dispositifs "Microphone" et "Mouse – Keyboard".

Nous souhaitons ici montrer la complémentarité, ainsi, nous proposons un environnement qui génère un remplissage des champs Nom et Adresse. Nous imposons comme contrainte générale le fait qu'une seule commande par dispositif est possible chaque cycle, mettant en relief la complémentarité dans les résultats produits. Nous écrivons un scénario à l'aide des probabilités conditionnelles qui réalisent la sélection du champ Nom, puis son remplissage, la sélection du champs Adresse et son remplissage, de manière continue. Nous ne présentons pas le fichier d'environnement car il est volumineux et son contenu semblable à celui de la **Figure 38**. Précisons que le test démarre dans l'état initial de l'application, c'est-à-dire où aucun nom ni adresse n'a été spécifié.

Fichier d'oracle

Nous donnons d'abord la correspondance entre les codes utilisés et les entrées – sorties de YellowPage.

- Soit e1 : booléen [2] // commandes vocales dans l'ordre : "Name", "Address".
- Soit e2 : booléen [3] // commandes vocales dans l'ordre : "Laurence", "Cooper", "Smith".
- Soit e3 : booléen [6] // commandes vocales dans l'ordre : "B Grenoble", "B in Grenoble", "Building B in Grenoble", "C Grenoble", "C in Grenoble", "Building C in Grenoble".
- Soit e4 : booléen [2] // commandes "Mouse – Keyboard" dans l'ordre : "Name" , "Address".
- Soit e5 : booléen [5] // commandes "Mouse – Keyboard" dans l'ordre : "Laurence", "Cooper", "Smith", "B in Grenoble", "C in Grenoble".
- Soit s2 : booléen [7] // les entrées du controleur de dialogue dans l'ordre. "Name", "Address", "Laurence", "Cooper", "Smith", "B in Grenoble", "C in Grenoble".

Nous souhaitons vérifier la complémentarité des entrées du système interactif YellowPage, pour les tâches de spécification du nom et de l'adresse. Nous identifions trois propriétés exprimables en LUSTRE. La première propriété découle du fait que l'on observe la complémentarité : en effet, seules deux entrées complémentaires doivent produire une sortie. Ainsi, une sortie doit toujours être composée de la sélection d'un champ et d'une valeur pour ce champ. Nous modélisons cette propriété par le nœud LUSTRE de la **Figure 42**. Nous utilisons ce nœud avec en paramètre la sortie s2 observée sur YellowPage :

```
node propriete1( s : bool^7 ) returns( ok: bool);
  var selectionchamp, remplissagechamp : bool;
let
  selectionchamp = s[0] or s[1] ;
  remplissagechamp = s[2] or s[3] or s[4] s[5] or s[6];
  ok = (not (selectionchamp) or remplissagechamp) ;
tel
```

Figure 42 ***: nœud LUSTRE pour la première propriété de complémentarité***

La deuxième et la troisième propriétés exploitent le fait que les entrées sont ordonnées. En effet, l'ordre des entrées doit être : spécification du champ suivi du remplissage de ce champ. Ainsi l'application ne doit produire une sortie que lorsque le remplissage du champ intervient.

C'est le sens du nœud LUSTRE "propriété 2" décrit à la **Figure 43**: "si une sortie est présente alors, un champ a été rempli durant ce cycle"

```
node propriete2(e2 : bool^3 ; e3 : bool^6 ; e5 : bool^5 ; s : bool^2 ) returns( ok: bool);
  var remplissage, sortie : bool;
let
  sortie = s[0] or s[1];
  remplissage = LIN_OR(3,e2) or LIN_OR(5,e5) or LIN_OR(6,e3);
  ok = (not sortie) or  remplissage ;
tel
```

Figure 43 ***: nœud LUSTRE pour la deuxième propriété de complémentarité***

La troisième propriété exprime le fait que les événements "sélection", "remplissage", "sortie" sont ordonnés dans le temps. Cela est décrit dans le nœud LUSTRE de la **Figure 44**.

```
node propriete3(e1 : bool^2 ; e2 : bool^3 ; e3 : bool^6 ; e4 : bool^2 ; e5 : bool^5 ; e6 : bool^1 ; s2 : bool^7 )returns( ok: bool);
  var Entree1, Entree2, Sortie : bool;
let
  Entree1 = LIN_OR(2,e1) or LIN_OR(2,e4) ;
  Entree2 = LIN_OR(3, e2) or LIN_OR(6, e3) or LIN_OR(5,e5) ;
  Sortie = LIN_OR(7,s2);
  p3 = once_from_to(Entree2,Entree1,Sortie);
tel
```

Figure 44 ***: nœud LUSTRE pour la troisième propriété de complémentarité***

Résultats des tests

Nous avons joué plusieurs séries de test, en activant et désactivant les 3 nœuds LUSTRE des propriétés. Cela nous a permis de montrer que la propriété 1 est violée sur 2 et 3 % cycles de chaque test en moyenne. Nous présentons à la **Figure 45** un tableau d'une partie d'un test ou la propriété 1 est violée. Ce tableau n'est pas représentatif de tous les résultats produits, puisqu'elle exhibe une erreur de l'assemblage de composants chargés d'assurer la complémentarité des entrées. Nous utilisons des préfixes pour exprimer les entrées - sorties dans le tableau : "V-" pour la voix, "CS-" pour clavier – souris, et "R-" pour les commandes résultantes.

Cycle	Entrées	Sorties	Oracle
76	V-LabelName	-	vrai
77	V-LabelName		vrai
78	CS-Cooper	R-LabelName, R-Cooper	vrai
79	V-LabelName	-	vrai
80	V-LabelAddress	-	vrai
81	V- B_Grenoble	R-LabelAddress	faux
82	V-LabelAddress	-	vrai
83	V-Laurence	R-LabelName, R-Laurence	vrai
84	V-Laurence	-	vrai

Figure 45 : ***tableau de synthèse des résultats partiels d'un test pour la complémentarité***

Au cycle 77, nous observons une entrée vocale "Name" pour sélectionner le champ nom, et aucune sortie. Au cycle 78, nous observons une entrée Clavier "Cooper", afin de spécifier la valeur du champ Nom. Pendant de même cycle, nous observons la sortie "Name + Cooper", ce qui montre un exemple de complémentarité réussi entre deux modalités. Respectivement au cycle 80 et 81, la trace d'exécution montre une entrée vocale "Address" et "B Grenoble". La sortie du cycle 81 n'a pas la forme "Address" + "B in Grenoble". Les deux entrées complémentaires ont provoqué une sortie qui n'est pas conforme à celle attendue. Connaissant la manière de construire une application interactive avec ICARE, nous découvrons rapidement l'origine de ce disfonctionnement. Il s'agit d'une erreur dans la description du fichier de configuration du composant ICARE "SimpleCommand9" de la **Figure 41**. En effet, la ligne devant contenir la valeur "B in Grenoble" ne contenait que "B Grenoble", le "in" étant manquant. Seule la valeur "B in Grenoble" est acceptée par le contrôleur de dialogue, c'est pourquoi notre instrumentation effectuait un test sur cette valeur et pas sur la valeur "B Grenoble". Il est toujours intéressant d'observer qu'une méthode de test permet de trouver des erreurs.

Bilan

Après nos premières expériences, il est possible de dresser un bilan de notre proposition de couplage ICARE – Lutess pour la connexion de Lutess à une application interactive multimodale. Notre couplage est intéressant à plusieurs titres :

1. Le couplage ICARE –Lutess est efficient du point de vue du mécanisme d'attente mutuelle entre Lutess et une application interactive multimodale. En effet, il a permi de réaliser de nombreux tests sur la première version de YellowPage.
2. Notre approche de vérification d'une application interactive multimodale par Lutess permet de vérifier une adaptation des propriétés CARE sur les assemblages de composants ICARE.
3. Notre couplage identifie clairement les points de connexion en entrée et en sortie d'ICARE, ce qui simplifie l'utilisation de la méthode.
4. Notre couplage ICARE – Lutess est systématique au niveau de son implémentation pour la partie ICARE. Le testeur peut, une fois l'ensemble de l'instrumentation produite, effectuer de nombreux tests en modifiant moins de 100 lignes de code (en JAVA : la boucle "while" de connexion ICARE – Lutess, et en LUSTRE : les fichiers environnement et oracle).

Néanmoins, notre approche montre des trois faiblesses :

1. Les points d'entrées – sortie de connexion au-delà d'ICARE ne sont pas précisés. L'habileté du testeur est donc mise à l'épreuve.
2. Le code du couplage ICARE – Lutess est systématique, mais il est très long à écrire, et peut être source de nombreuses erreurs.
3. Notre proposition de couplage ne tient pas compte de l'éditeur de schéma ICARE. Cet outil est pourtant très utile pour produire un moteur de fusion ICARE

Ce bilan intermédiaire, montre que notre méthode de test est efficace, mais qu'il faut traiter ses défauts pour la rendre utilisable. Notre objectif suivant a été d'effacer les points négatifs 2 et 3. Pour cela, nous proposons d'automatiser la production de l'instrumentation, entre ICARE et Lutess au sein de l'éditeur de schémas ICARE. Cette nouvelle proposition doit avoir pour objectif de corriger les inconvénients de la méthode tout en préservant les points que nous avons montrés comme positifs.

Automatisation du couplage Lutess – ICARE

Nous présentons dans cette partie notre proposition de couplage automatique entre ICARE et Lutess. Notre proposition tient compte des spécificités de la production des schémas ICARE au sein de l'éditeur, ainsi que de notre expérience de couplage ICARE – Lutess produite manuellement sur la version 1 de YellowPage. Nous montrons d'abord le couplage que nous proposons, puis, sa description à l'exécution, et son implémentation au sein de l'éditeur de schémas ICARE. Enfin, nous présentons la validation de notre outil par son utilisation sur la deuxième version de YellowPage.

Notre proposition de couplage automatique

Notre proposition pour le couplage automatique d'ICARE à Lutess reprend les grandes lignes de la structure de la proposition manuelle. Elle applique les principes de simulation de composants ICARE et d'observation des sorties des composants ICARE. La **Figure 46** détaille notre nouvelle proposition de couplage ICARE - Lutess à l'exécution. La connexion avec Lutess comprend trois classes : "Codeur", "Décodeur" et "LutessInterface". La classe "Décodeur" correspond aux entrées de l'application interactive YellowPage. Elle comprend un ensemble de méthodes "Generate* (booléen [] B)", chacune correspondant à une flèche du schéma ICARE. Une méthode "Generate* (booléen [ ] B)" crée une instance d'ICAREEvent, lui attribue une sémantique en fonction de B, et l'envoie au composant "*". La **Figure 46** montre la connexion en entrée du composant de langage d'interaction "SC2, Simple Command 2". Cette connexion permet de simuler le comportement du composant de type dispositif "Microphone".

La Classe "Codeur" implémente plusieurs méthodes, écouteurs d'ICAREEvents. Elle s'abonne aux différents composants ICARE, en sollicitant l'appel d'une méthode "From*" en cas d'événements. Ainsi chaque composant ICARE possédant une sortie est sollicité pour un abonnement supplémentaire. Il existe donc une méthode "From*" par composant ICARE, ce qui permet d'observer tous les fils d'interaction du schéma ICARE. Les méthodes "From*" traduisent l'ICAREEvent transmis en paramètre en un vecteur de booléens qui est ensuite stocké. La **Figure 46** illustre l'observation de la sortie du composant de type langage d'interaction "SC2, Simple Command 2". Cela revient à observer les événements ICARE en entrée du port droit du composant "Redondance/Équivalence".

La classe "LutessInterface" contient plusieurs méthodes pour la lecture et l'écriture sur l'entrée-sortie standard, ainsi qu'une méthode "ConnectToLutess". Celle-ci implémente l'attente mutuelle entre le logiciel sous test et Lutess. Nous retrouvons les quatre étapes suivantes :

1. Lecture de vecteurs de booléens
2. Appels des méthodes de la classe Décodeur correspondant aux vecteurs lus
3. Attente de 1000 ms
4. Écriture des ICAREEvents observés

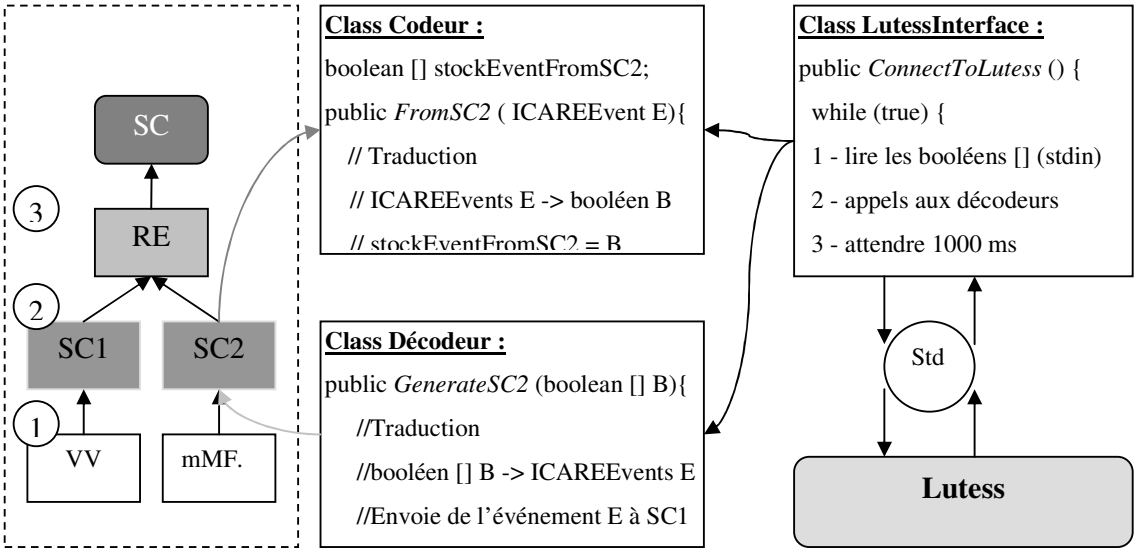

*Figure **46 : schéma détaillé de la connexion ICARE - Lutess.***
A gauche nous considérons une partie du schéma ICARE YellowPage de la Figure 22.

L'instrumentation proposée pour la connexion ICARE - Lutess consiste à simuler des événements ICARE et donc à simuler le comportement de composants ICARE. Nous distinguons trois niveaux d'abstraction de simulation notés 1, 2 et 3 à la **Figure 46**.

1. **Niveau dispositif :** il s'agit de simuler un composant ICARE de type Dispositif par la génération d'événements ICAREEvents à destination du composant de langage d'interaction associé. Par exemple à la **Figure 46**, nous simulons le composant de dispositif "VV, ViaVoiceInput" en émettant des événements vers le composant "SC2, Simple Commands 2".
2. **Niveau langage d'interaction :** il s'agit de simuler un composant ICARE de type Langage d'Interaction. La simulation est réalisée en émettant des événements ICAREEvents à destination des composants cibles de type Composition ou Langage d'Interaction ou à destination du reste du système en simulant des tâches élémentaires abstraites ou commandes. Par exemple à la **Figure 46**, pour simuler le composant "SC2, Simple Commands 2", des événements sont générés à destination du composant de composition "RE, Redondance/Équivalence".
3. **Niveau composition :** il s'agit de simuler un composant ICARE de Composition par l'émission d'événements à destination de ses cibles : des composants de type Composition ou Langage d'Interaction ou encore le reste du système si il s'agit de tâches élémentaires abstraites. Pour exemple à la **Figure 46**, pour simuler le composant "RE, Redondance/Équivalence", des événements sont générés à destination de la tâche abstraite "Simple Commands".

Un utilisateur de notre générateur automatique d'instrumentation du couplage ICARE – Lutess doit réaliser quelques opérations manuellement pour réaliser les tests. Le processus de vérification d'une application par le test avec Lutess est illustré à la **Figure 47**. L'utilisateur peut activer ou désactiver la génération du couplage par la boîte à cocher du menu de l'éditeur ICARE. Une grande partie du code de connexion est générée automatiquement (étape 2 de la **Figure 47**). Pour effectuer des tests, il convient néanmoins de décrire les fichiers environnement et oracle (étape 3 de la **Figure 47**), puis de compléter le contenu de la boucle "while" de la classe "LutessInterface" (étape 4 de la **Figure 47**). Ces activités manuelles de

programmation dépendent des vérifications à effectuer et correspondent à l'écriture d'un total de moins de 100 lignes de code (JAVA/LUTRE). Après compilation (étape 5 de la **Figure 47**), les tests peuvent alors être effectués (étape 6 de la **Figure 47**). Tandis que la vérification de nouvelles propriétés sur les mêmes entrées-sorties entraîne la modification de l'oracle seulement (étape 3 de la **Figure 47**), effectuer des tests avec de nouvelles entrées-sorties implique de refaire les étapes 3, 4 et 5 de la **Figure 47**.

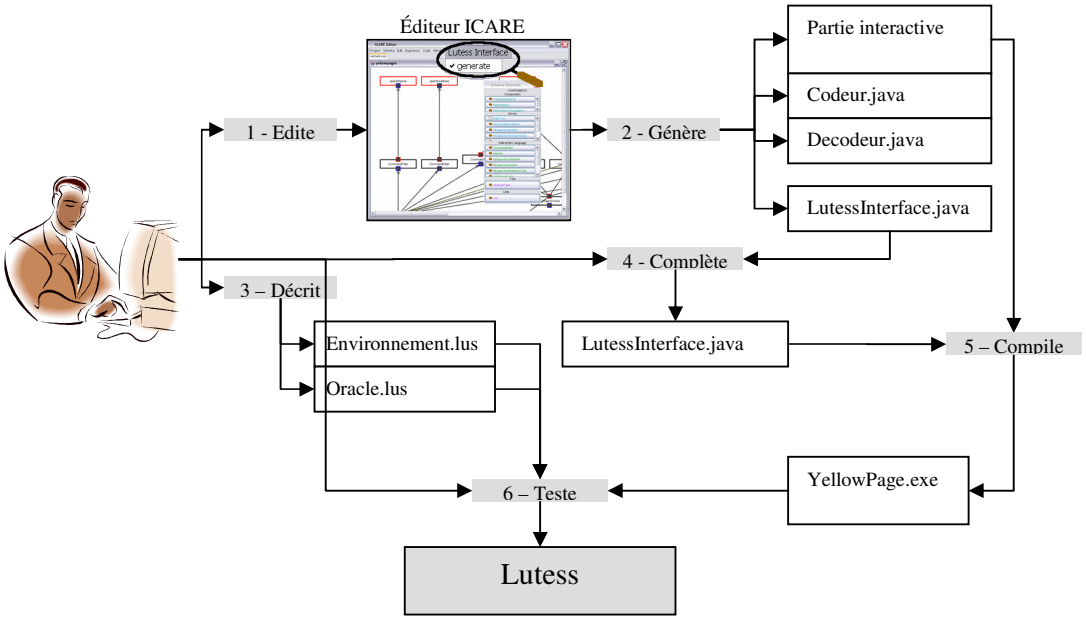

Figure 47 ***: processus de vérification avec l'outil Lutess d'un système interactif multimodal développé avec la plateforme ICARE.***

Conception et réalisation de l'extension de la plateforme ICARE

Nous montrons dans ce paragraphe comment nous avons conçu et réalisé notre générateur de couplage ICARE – Lutess au sein de l'éditeur ICARE. La description des besoins a été proposée à la section précédente. Nous précisions dans notre solution implique une quasi-totalité du code de couplage ICARE – Lutess généré automatiquement. Notons enfin que les parties qui ne sont pas générées automatiquement sont celles qui dépendent directement des tests à effectuer.

La communication entre les composants ICARE se fait au moyen d'événements ICARE. Nous souhaitons produire au sein du code de couplage, les méthodes de traduction contenue dans les classes "codeur" et "décodeur". La génération du traducteur nécessite du point de vue du générateur, la connaissance préalable de la sémantique des événements pouvant circuler dans le schéma ICARE. Pour plusieurs composants de type Langage d'Interaction, la sémantique des événements en entrée/sortie, est décrite au sein du fichier externe de configuration. Ce fichier est spécifié au sein de l'éditeur ICARE. C'est le cas notamment des composants de Langage d'Interaction "CommandFilter" qui ont été utilisés à de nombreuses reprises dans l'implémentation de YellowPage sous les noms de "Simple Command <numéro>". Néanmoins ce fichier de configuration n'est pas présent pour tous les composants ICARE, citons à titre illustratif le composant générique de composition

"Redondance/Équivalence" et les composants de type Dispositif (sémantique propre au code comme dans le dispositif "Souris"). Dans ce contexte, aucun des composants au sein de l'éditeur ne possède une description des événements qu'il peut recevoir et émettre. Il s'agit d'une première difficulté pour l'instrumentation automatique que nous résolvons par la modification de la classe "IcareContainer.java". La classe "IcareContainer.java" gère le schéma ICARE en cours d'édition en maintenant à jour une représentation interne de celui-ci. Nous ajoutons à cette classe le rôle de gestion de la sémantique des événements ICARE pouvant circuler entre les composants du schéma ICARE. Les composants ICARE actuellement construits émettent et reçoivent des événements dont nous pouvons décrire la sémantique :

- Soit en décrivant préalablement la sémantique, à condition qu'elle soit fixée avant l'utilisation du composant.
- Soit en décrivant dynamiquement la sémantique, lors de la construction du schéma. C'est le cas pour les composants de Langage d'Interaction "Command Filter" utilisés pour "Simple Command".
- Soit en décrivant le modèle de transformation que ces composants implémentent entre leurs entrées et sorties. C'est le cas des composants de composition.

Le codage de la sémantique est fait de plusieurs manières :

- Soit sous la forme d'une chaîne de caractères (String JAVA).
- Soit sous la forme d'un entier (Integer JAVA).
- Soit d'une composition ordonnée de chaînes de caractère et d'entiers.

Nous proposons donc une syntaxe simple pour décrire les événements émis et reçus par les composants à partir du constat précédent. Un événement est une composition non vide de chaînes de caractères et d'entier, qui peut être décrite par :

<EVENT> = <Element> <EVENT>* ;

<Element> = <Int> | <String> <chaine_identifiant> ;

<chaine_identifiant> = <command> <command>* ;

<command> = "une chaîne de caractères" ;

Afin que l'éditeur maintienne la description selon la syntaxe ci-dessus des événements circulant entre les composants d'un schéma, nous définissons que :

- Une description est attachée à chaque port de chaque composant utilisé dans un schéma, c'est-à-dire que l'on considère que ce sont les ports d'entrées et de sorties des composants qui reçoivent et émettent des événements ICARE.
- Nous décrivons dans des fichiers les composants pouvant l'être statiquement, tels certains composants de type Dispositif ou encore de type Langage d'Interaction. Pour ces composants, la sémantique des événements est cablée dans le code et n'est pas modifiée lors de l'assemblage au sein de l'éditeur.
- Nous ajoutons à l'éditeur de schémas ICARE, un éditeur des correspondances entre les entrées et les sorties pour les composants "Command Filter". L'interface de ce nouvel éditeur est présentée à la **Figure 48**. Celui-ci sert deux objectifs : d'une part, il

permet à l'utilisateur de spécifier simplement les associations de commandes au sein de l'éditeur ICARE, tâche qui auparavant se pratiquait dans des fichiers de configuration, après l'édition du schéma. D'autre part, elle permet au générateur d'obtenir un modèle des données en entrées et en sorties du composant. En effet, les éléments décrits en entrées de la **Figure 48** sont traduits en la description des événements du port d'entrée des composants "Command Filter". De la même manière, l'intersection de l'ensemble des éléments décrit dans la colonne sortie de la **Figure 48** est traduit en la description des événements du port de sortie du composant "Command Filter".

- Nous décrivons le fonctionnement des composants de composition, afin de pouvoir construire la description des événements émis par le port de sortie, en fonction des descriptions des événements reçus par les ports d'entrées.

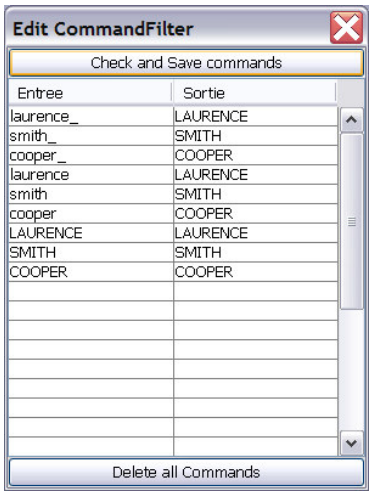

Figure 48 ***: éditeur de commandes pour les composants "commandFilter"***

La description attachée à chaque port n'est pas mise à jours lors de l'édition, afin d'éviter les problèmes liés aux suppressions de composant et de liaison. Nous réalisons un algorithme qui parcourt le schéma ICARE lors de la sauvegarde, juste avant la génération de code. Une tâche abstraite ICARE est effectuée à partir d'une composition d'événements de plus bas niveau. Il convient descendre en suivant les flèches du schéma ICARE pour obtenir les composants de type dispositif à l'origine de cette composition. Nous avons réalisé un algorithme qui parcourt en profondeur d'abord, l'arbre constitué des ports des composants du schéma ICARE. Cet algorithme complète en premier la description des ports de plus bas niveaux, puis remonte ensuite pour compléter les descriptions d'événements de plus haut niveaux. Cela permet de compléter de manière efficace l'ensemble des ports du schéma, et particulièrement les ports de sortie des composants de composition. La **Figure 49** montre la progression de notre algorithme pour le remplissage des ports d'un schéma en Redondance/Équivalence de YellowPage. Dans un premier temps, notre algorithme descend dans les ports du schéma depuis le composant tâche abstraite "Simple Commands", jusqu'au composant de type Dispositif "ViaVoiceInput". A partir de ce dernier, il remplit successivement les port numérotés de (1) à (4), en remontant dans le schéma. Une fois le port (4) rempli, l'algorithme considère alors que le composant de composition "Redondance/équivalence" possède deux ports d'entrée. Il ne peut donc pas remplir directement le port de sortie. L'algorithme parcourt alors le schéma jusqu'au composant "myMappyFrame". Il remplit ensuite les ports de (5) à

(8) en remontant dans le schéma. Une fois les ports (4) et (8) remplis, l'algorithme attribue au port de sortie du composant "Redondance/Equivalence" une description qui est l'union des descriptions des ports d'entrées.

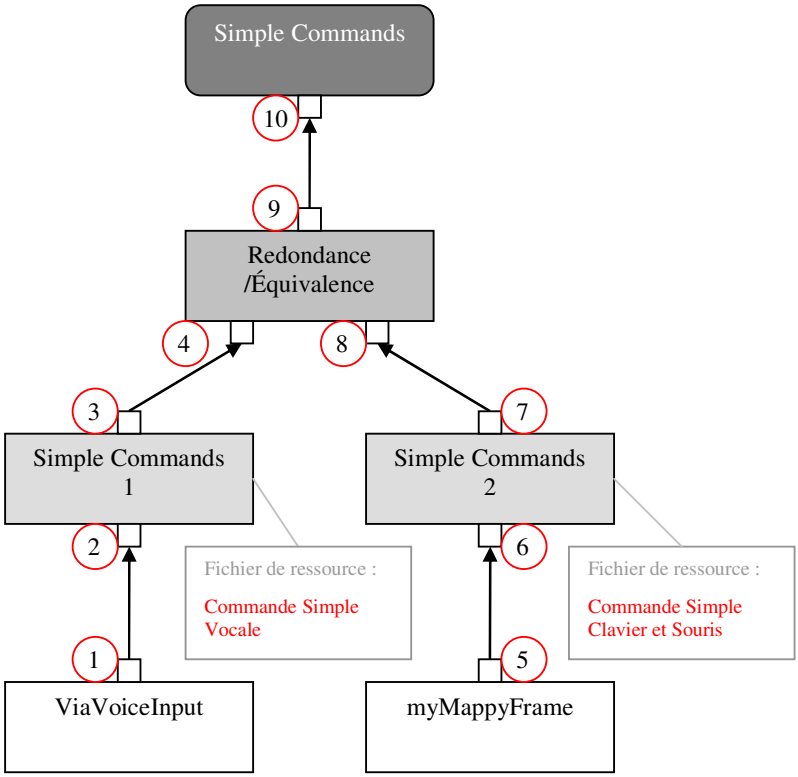

Figure 49 ***: parcours des ports d'un schéma ICARE de YellowPage en Redondance/Équivalence***

La description des ports du schéma donne un ensemble d'informations suffisant pour produire les traducteurs en entrée et en sortie. Il faut toutefois définir des règles pour la traduction vers des vecteurs de booléens. La génération du code de traduction est possible à partir de des règles suivantes que nous avons définies.

- Un entier est représenté par un vecteur de 32 booléens
- Un chaîne de caractères d'un événement pouvant prendre N valeurs distincts, est représenté par un vecteur de N bits. Chaque bit représente alors une chaîne de caractères dans un ordre défini automatiquement par le générateur.
- Si la description d'un événement comporte plusieurs éléments "<Element>", alors le vecteur de booléens correspondant est la concaténation ordonnée des vecteurs de booléens de chaque élément.

Un problème important subsiste pour la connexion de la classe "codeur" ainsi généré. En effet, lors de l'édition du schéma ICARE, il faut abonner l'instance **future** de la classe "codeur", aux différents composants du schéma ICARE. Nous utilisons le terme futur, car l'instance de la classe codeur n'est pas créée dans l'éditeur, mais à l'exécution. Rappelons que le code de la classe codeur doit être compilée après l'édition du schéma ICARE au sein de l'éditeur. Nous contournons le problème par un mécanisme d'appel dynamique à une méthode sur un objet. Ce mécanisme ne nécessite pas de connaître statiquement l'instance

d'une classe pour l'écriture d'un appel de méthode à cette classe. Nous ajoutons une attribut "Object" JAVA à la classe "IcareContainer.java" par la déclaration présentée à la **Figure 50**. Celle-ci contient à l'exécution une instance de la classe "codeur".

```
private Object codeur ;
public void setCodeur( Object value ){ codeur= value;}
public Object getCodeur() { return codeur ; }
```

Figure 50 ***: déclaration de l'attribut "codeur" et de ses méthodes d'éditions***

Les méthodes de traduction de la classe codeur sont générées. Leur entêtes est la suivante :

"public void EventComeTo<nom composants>_<identifiant> (ICAREEvent event) ". Dans le code du chargement des liens entre composants du schéma ICARE, nous ajoutons quelque instructions ne s'exécutant que lorsque l'application est en mode "déployer" (hors de l'éditeur). Ces lignes, décrites à la **Figure 51**, ne sont pas triviales à comprendre en première lecture. Elles consistent en un abonnement à un composant ICARE vers une méthode donnée de la classe "codeur.java". La classe "Statement" (b) permet de créer une instruction qu'il est possible d'exécuter par la méthode "execute" (d). Nous créons une instruction "Statement" avec comme arguments :

- Le composant auquel nous souhaitons abonner la méthode de la classe codeur.
- La nom de la méthode qui permet l'abonnement (c). Le nom de cette méthode a été récupéré par introspection lors de l'instanciation du composant au sein de l'éditeur.
- Le tableau d'objet (a) contenant l'instance de la classe codeur et le nom de la méthode à appeler en cas d'événement.

Notons que l'instance de la classe codeur est considérée comme un "Object" JAVA, et, est récupérée au moyen de la méthode "getcodeur()" de la **Figure 50**. Afin que lors de exécutions de l'instruction "statlutess.execute()" (d), il n'y ait pas d'exception, il est nécessaire que l'attribut "codeur" contienne la référence de l'instance de la classe codeur à l'exécution. Le remplissage de cet attribut s'effectue lors du chargement du schéma à l'exécution, avant le chargement des différents liens.

```
int Discriminant = this.getLinksNum(idSource,idTarget);
Object [] objlutess = new Object[1];

// création d'un tableau de parametre pour la methode d'abonnement
objlutess[0] = EventHandler.create(                                          a
      (Class) linkVector.elementAt(1),
      this.getCodeur() ,
      new String("EventComeTo"+((Vector)(componentsList.get(new Integer(idTarget)))).elementAt(4)+"_"+Discriminant),
      linkVector.elementAt(3).toString(),
      linkVector.elementAt(4).toString());

// instanciation de la methode d'abonnement
Statement statlutess =  new Statement(((Vector)(                             b
      componentsList.get(new Integer(idSource)))).elementAt(0),
      linkVector.elementAt(0).toString(), c
      obj);

// appel de la methode d'abonnement
statlutess.execute();                                                        d
```

Figure 51 ***: création d'un abonnement à un composant ICARE***

Nous avons décrit le fonctionnement et l'implémentation de notre coupleur automatique ICARE – Lutess. La génération de code JAVA de cette connexion ne soulève pas de problème particulier. Nous utilisons des conventions pour déterminer de manière unique les noms des méthodes générées. Afin d'aider l'utilisateur de notre coupleur dans sa démarche de test, nous générons deux fichiers supplémentaires contenant l'aide pour la connexion de son application. Ces fichiers précisent les entêtes des méthodes de connexion en entrée "Generate*(booleen [] B)" et en sortie "getStockEvent*()". Ils précisent également la correspondance dans les traductions ICAREEvents vers booléens, et booléens vers ICAREEvents. Cela permet d'une part, de simplifier l'écriture des fichiers d'environnement et d'oracle, et d'autre part de simplifier l'écriture de la boucle "while" du fichier "LutessInterface.java".

Utilisation de l'extension de la plateforme ICARE sur YellowPage v2

Afin de tester notre générateur automatique de couplage ICARE - Lutess, nous l'utilisons sur la deuxième version de l'application interactive multimodale YellowPage. Tout d'abord, donnons quelques chiffres :

- La génération totale du code (YellowPage v2 + couplage) prend moins d'une seconde.
- 2000 lignes de code JAVA sont produites pour le couplage.
- 1200 lignes de documentation sont générées pour aider l'utilisateur.

Rappelons à titre de comparaison, que le couplage manuel (en considérant YellowPage v1) a demandé environ 3 jours pour fonctionner correctement. Notons que le générateur n'est pas encore parfait, et que nous avons du modifié le code produit par endroit, afin qu'il soit sémantiquement correct. Précisons que les codes présentés dans ce paragraphe peuvent contienne des fautes d'orthographe. Nous avons préférer donner ce code sans modification dans ce mémoire, afin que le lecteur puisse se faire une bonne idée de l'état actuel de l'extension de la plateforme ICARE.

La **Figure 52** montre une méthode de traduction en sortie produite par notre générateur automatique. Nous retrouvons la déclaration d'un tableau de quatre booléens pour stocker les événements observés (a). Une méthode de récupération de ce tableau de booléens est également générée (b). Enfin, le traducteur à proprement parler est produit, au travers de la méthode "EventComeToCommandFiterToTraduct_5" (c).

```
// variable de stokage du vecteur traduit
private boolean [] EventTraductionOfCommandFilter_5 = new boolean [4] ;   a

// getter pour cette variable
public boolean [] getEventTraductionOfCommandFilter_5 () {
    return  EventTraductionOfCommandFilter_5 ;                            b
}

// Traducteur pour cette variable
public void EventComeToCommandFilterToTraduct_5 ( ICAREEvent event ) {
    Vector v = event.getVector();
    boolean [] booltemp = new boolean [32] ;
    if ( (( String ) v.elementAt(0)).equals("name_") ){
        EventTraductionOfCommandFilter_5[0] = true ;
    } if ( (( String ) v.elementAt(0)).equals("address_") ) {             c
        EventTraductionOfCommandFilter_5[1] = true ;
    } if ( (( String ) v.elementAt(0)).equals("Name")  {
        EventTraductionOfCommandFilter_5[2] = true ;
    } if ( (( String ) v.elementAt(0)).equals("Address") )   {
        EventTraductionOfCommandFilter_5[3] = true ;
}}
```

Figure 52 ***: code produits pour la traduction en sortie du composant "Simple Command 5"***

La **Figure 53** est la partie de la documentation produite en sortie correspondant à l'instrumentation donnée à la **Figure 52**, pour le composant de type Langage d'Interaction "Simple Command 5". Nous notons que d'une part, la documentation correspond au code généré du couplage, et d'autre part, que cette documentation est lisible : elle présente l'entête et la correspondance des bits du tableau de booléens récupéré.

```
/*
* ICAREEvent intercepter en entrée de CommandFilter depuis le port 1
*/

public boolean [] getEventTraductionOfCommandFilter_5 () ;

 Le booleen renvoyer represente 1 Elements et est definit sur par [0 ..3]
  Element 0 == <String>   bool[0..3]
     bool[0] : "name_"
     bool[1] : "address_"
     bool[2] : "Name"
     bool[3] : "Address"
```

Figure 53 ***: documentation de connexion en sortie correspondant au composant "Simple Command 5"***

Le code produit pour la traduction en entrée du composant "Simple Command 5" est donné à la **Figure 54**. Le nom de la méthode générée (a) pour effectuer cette traduction, est légèrement différent de ce qui est annoncé précédemment : cette différence a pour but de mieux discriminer les entêtes de méthodes. Le corps de la méthode reprend la structure attendue : instanciation d'un événement ICARE (b), traduction du tableau de booléens en paramètre vers une sémantique pour cet événement (c), puis envoi de cet événement (e). Le partie (d) correspond au remplissage de champs supplémentaires de l'événement ICARE qu'il n'est pas utile de détailler dans ce mémoire.

```
/* Genere un ICAREEvent vers CommandFilter à partir de
*  myMappyFrameet de dicriminant 4
* et de la description booleenne de l'evenement
*/

public void FTD_GeneratemyMappyFrameTOCommandFilter_4                    (a)
( boolean[] bool , long initial, long time) {
   ICARE.ICAREEvent event = new ICARE.ICAREEvent(this);                  (b)
   Vector v = new Vector();
   boolean [] btemp = new boolean [32];
   if ( bool[0] == true ) v.addElement(new String("name_" )) ;
   if ( bool[1] == true ) v.addElement(new String("address_" )) ;        (c)
   if ( bool[2] == true ) v.addElement(new String("Name" )) ;
   if ( bool[3] == true ) v.addElement(new String("Address" )) ;

   event.setVector(v);
   event.setTime(time);                                                  (d)
   event.setInitialTime(initial);
   event.setConfidenceFactor(1);
   event.setNbOfValues(1);
Object [] obj = new Object[1];
obj[0] = event;
container.executeMethod(6 , new String("setData") , obj);               (e)
}
```

Figure 54 ***: code générer pour la traduction en entrée du composant "Simple Command 5"***

De manière symétrique à la documentation produite pour la sortie, celle produite pour l'entrée du composant "Simple Command 5" est présenté à la **Figure 55**. Nous retrouvons le nom de la méthode a appeler dans la boucle "while" de la classe "LutessInterface.java", ainsi que la description de la traduction du vecteur de booléen vers une sémantique pour l'événement simulé.

```
/*Liaison entre myMappyFrame et CommandFilter : Generation FTD (From Target Descriptor), à partir de la
*descripotion du port du composants cible (ce qu'accepte le composants cible )
*/
public void FTD_GeneratemyMappyFrameTOCommandFilter_4( boolean[] bool , long initial, long time);

  bool represente 1 Elements, et est definit sur [0 .. 3] par :
  Element 0 == <String>   bool[0..3]
    bool[0] : "name_"
    bool[1] : "address_"
    bool[2] : "Name"
    bool[3] : "Address"
```

Figure 55 : ***documentation de connexion en entrée pour le composant "Simple Command 5"***

Le générateur de couplage Lutess – ICARE a produit des méthodes de connexion en entrée – sortie pour chaque flèche du schéma ICARE. Il est donc possible de simuler et d'observer l'ensemble des événements circulant dans le schéma. Parallèlement, les documentations de la connexion ont été produites. Avec la description d'un fichier d'oracle et d'environnement et, la complétion de la boucle "while" YellowPage v2 est prêt pour une série de tests.

Un test d'un assemblage en complémentarité

Nous présentons dans ce paragraphe un test produit grâce à Lutess sur la deuxième version YellowPage. Nous avons comme objectifs d'une part de montrer que notre générateur produit une instrumentation fonctionnelle, et d'autre part de montrer l'efficience de notre instrumentation automatique pour le test d'une propriété CARE [Coutaz 95, Nigay 97] pour la multimodalité. Nous proposons ici de vérifier le comportement de l'assemblage de composant ICARE en complémentarité pour la spécification d'un nom ou d'une adresse. La **Figure 56** illustre le positionnement de la boîte noire sur le schéma ICARE complet de l'application interactive multimodale YellowPage v2. Le schéma ICARE présenté à la **Figure 56** reprend une partie du schéma complet de la **Figure 29.** Nous simulons en entrée le comportement des composants "ViaVoiceInput" et "myMappyFrame" et nous observons en sortie les événements arrivant au niveau du composant Abstrait "Specify Name or Address".

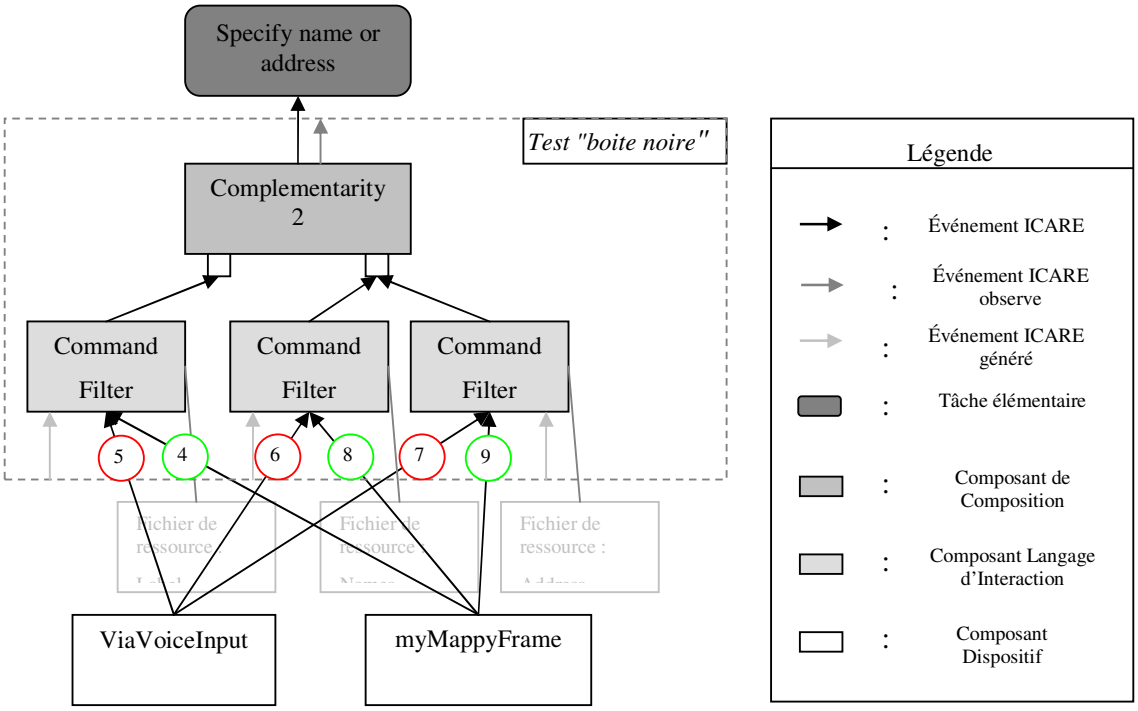

Figure 56 : ***Connexion en entrée - sortie pour le test de l'assemblage en complémentarité***

Fichier d'environnement

YellowPage permet à l'utilisateur de spécifier un nom et une adresse avec une combinaison des trois modalités voix clavier et souris. Rappelons que le composant "ViaVoiceInput" reconnaît un ensemble de commandes vocales, et que le composant "myMappyFrame" regroupe l'utilisation du clavier et de la souris. Ces deux composants peuvent émettrent un ensemble important d'événements donc seulement certains nous intéressent ici. La spécification d'un nom ou d'une adresse s'effectue en deux temps : d'abord, l'utilisateur sélectionne un champ avec la souris ou oralement (commande "name", ou "address"), puis l'utilisateur spécifie la valeur qu'il souhaite attribuer à ce champ oralement et en utilisant le clavier. Le schéma ICARE de la **Figure 56** illustre la connexion par six flèches des composants "ViaVoiceInput" et "myMappyFrame" aux trois composants de type Langage d'Interaction "CommandFilter". La simulation en entrée s'effectue selon les flèches. Ainsi, pour chacune des six flèches, nous devons utiliser une des méthodes de simulation proposées par notre instrumentation. Nous donnons d'abord la sémantique des événements pouvant circuler sur les flèches du schéma numérotées de 4 à 9 (numéro unique des flèches dans le schéma ICARE). Nous préfixons ces numéros par "m" pour "myMappyFrame", et par "v" pour "ViaVoiceInput" pour une plus grande lisibilité. Nous présentons également la sémantique des événements en entrée de la tâche abstraite " Specify Name or Address", sous le code "sortie".

- m4 : bool [2] // valeurs : "name", "address".
- v5 : bool[2] // valeurs : "name_", "address_".
- v6 : bool[3] // valeurs : "laurence_", "smith_", "cooper_".
- v7 : bool[6] // valeurs : "building_B_in_grenoble_", "B_in_grenoble_", "B_grenoble_", "building_C_in_grenoble_", "C_in_grenoble_", "C_grenoble_".
- m8 : bool[3] // valeurs : "laurence", "smith", "cooper".

- m9 : bool[6] // valeurs : "building_B_in_grenoble", "B_in_grenoble", "B_grenoble", "building_C_in_grenoble", "C_in_grenoble", "C_grenoble".
- Sortie : bool[11] // la sortie peut avoir plusieurs valeurs parmi : "Name", "Address", "Laurence", "Smith", "Cooper", "building_B_in_grenoble", "B_in_grenoble", "B_grenoble", "building_C_in_grenoble", "C_in_grenoble", "C_grenoble".

Nous souhaitons ici tester la complémentarité : ainsi, nous proposons un environnement qui génère un remplissage des champs Nom et Adresse. Nous prenons comme contrainte générale le fait qu'un seul événement puisse être généré pour un cycle de test donné, quel que soit le dispositif. Ceci met en relief la complémentarité dans les résultats produits. Nous écrivons cette contrainte avec une expression de la forme :

"environnement(#(m4[0], m4[1],v5[0], …, m9[6]);".

Nous avons décrit au paragraphe **0** le test de l'assemblage de composant en complémentarité de la première version de YellowPage pour la spécification d'un nom et d'une adresse. Nous adaptons le fichier environnement produit pour ce test. Afin de représenter un comportement réaliste de l'utilisateur, nous définissons un scénario de test, qui vise à reproduire des séquences de la forme : <sélection, remplissage>. Nous écrivons le scénario à l'aide de probabilités conditionnelles en quatre étapes : la sélection du champ nom, puis son remplissage, la sélection du champ d'adresse puis son remplissage. Précisons que le test démarre dans l'état initial de l'application, c'est-à-dire où aucune adresse ni nom n'a été spécifié. Nous ne présentons pas ici le fichier d'environnement, qui est semblable à celui produit au paragraphe **0**.

Fichier d'oracle

Nous souhaitons vérifier la complémentarité des entrées de l'application interactive YellowPage, pour les tâches de spécification du nom et de l'adresse. Pour ce test, nous avons adapté les propriétés utilisées sur l'assemblage en Complémentarité de la première version de YellowPage. Les trois propriétés proposées pour le test découlent de la propriété de complémentarité : la première illustrée à la **Figure 57** permet de vérifier qu'une sortie de l'assemblage en complémentarité est bien la combinaison de deux entrées : la sélection d'un champ, et le remplissage d'un champ.

```
node propriete1( sortie : bool^11 ) returns( ok: bool);
var labelname, nom, labeladdress, adresse : bool;
let
  labelname = sortie[0] ;
  labeladdress = sortie[1] ;
  nom = sortie[2] or sortie[3] or sortie[4] ;
  adresse = LIN_OR(6, sortie[5..10]) ;

  ok  = (not (labelname or labeladdress)
        or (nom or adresse)) and not( lname and laddress) ;
tel
```

Figure 57 ***: code de la propriété 1***

D'après la description du composant complémentarité, une sortie est émise seulement lorsque le deuxième événement est reçu par le composant. Rappelons que le composant complémentarité 2 utilisée ici effectue une fusion des événements en entrée seulement si ceux-ci arrivent de manière ordonnée (port gauche, puis port droit). Le port droit correspond à

l'arrivée des événements de remplissage d'un champ de texte. La propriété 2 présentée à la **Figure 58**, permet de vérifier qu'une sortie est définie que si un événement de remplissage d'un champ à été généré durant un cycle donné.

```
node propriete2( v6 : bool^9 ; v7 : bool^12 ; m8 : bool^9 ; m9 : bool^12 ; sortie : bool^11 ) returns( ok: bool);
var sortiename, name, sortieaddress , address: bool;
let
  sname = sortie[0] ;
  name = LIN_OR(9, v6)  or LIN_OR(9, m8);
  saddress = sortie[1] ;
  address = LIN_OR(12, v7) or LIN_OR(12, m9);

  ok = ((not sortiename) or name) and ((not sortieaddress) or address)  ;
tel
```

Figure 58 ***: code de la propriété 2***

La troisième propriété illustrée à la **Figure 59,** exploite l'ordonnancement des commandes afin de vérifier que celle-ci respecte bien la séquences "sélection", "remplissage" puis "sortie".

```
nodepropriete3 ( v5 : bool^4 ; v6 : bool^9 ; v7 : bool^12  ; m4 : bool^4 ; m8 : bool^9 ; m9 : bool^12 ;
                 sortie : bool^11 ) returns (ok : bool);
var label, valeur, specify  : bool;
let
  label = LIN_OR(4,v5) or LIN_OR(4,m4) ;
  valeur = LIN_OR(9, v6) or LIN_OR(12, v7) or LIN_OR(9, m8) or LIN_OR(12, m9) ;
  specify = LIN_OR(11,sortie);

  ok = once_from_to(label,valeur,specify);

tel
```

Figure 59 ***: code de la propriété 3***

Résultats des tests

Nous avons produit plusieurs tests à partir de l'environnement et de l'oracle décrits dans les paragraphes précédents. Ces tests révèlent un dysfonctionnement de notre assemblage. En effet, l'oracle constitué d'un ET logique des trois propriétés est violé en moyenne dans 8% des cycles de test. Nous avons évalué, en activant et en désactivant les propriétés une à une, que seule la propriété 2 était violée au cours du temps. Le tableau de la **Figure 60** montre une partie d'un de nos fichiers résultats. Nous utilisons les préfixes suivant pour exprimer les entrées - sorties dans le tableau : "V<numero>-" pour la voix, "M<numero>-" pour "myMappyFrame", et "C-" pour les commandes résultantes. Les <numeros> correspondent aux identifiants uniques des flèches du schéma ICARE.

Cycle	Entrées	Sorties	Oracle
39	V5-labelname	-	vrai
40	M8-laurence	C-<Name, Laurence>	vrai
41	M4-labeladdress	-	vrai
42	M4-labeladdress	-	vrai
43	V7-B_grenoble_	C-<Address, B Grenoble>	vrai
44	-	-	vrai
45	V5-labelname	-	vrai
46	V5-labelname	-	vrai
47	M8-cooper	C-<Name, Cooper>	vrai
48	M4-labeladdress	-	vrai
49	V6-cooper	C-<Address, Cooper>	faux

Figure 60 : ***tableau de synthèse des résultats partiels d'un test pour la complémentarité***

Au cycle 39, nous observons une entrée vocale "Name" (V5-labelname) pour sélectionner le champ nom, et aucune sortie. Au cycle 40, nous observons une entrée clavier "laurence" (m8-laurence) qui spécifie la valeur du champ Nom. Durant ce même cycle, nous observons la sortie "<Name + Laurence>". Cette sortie montre que les deux entrées complémentaires ont été fusionnées. De la même manière, dans les cycles 42 et 43, nous observons respectivement une entrée souris "Address"(M4-labeladdress) et une entrée voix "B_grenoble" (V7-B_grenoble_), suivies au cycle 43 d'une sortie "<Address + B Grenoble>". Les deux entrées complémentaires des cycles 48, 49 montrent une erreur manifeste de notre implémentation de YellowPage v2 au cycle 49 avec un oracle faux. En effet, au cycle 48, nous observons une entrée "Address" (M4-labelAddress), suivie au cycle 49 d'une entrée vocale "cooper" (V6-cooper). Ces deux entrées sont fusionnées comme le montre la sortie "<Address + Cooper>". La propriété de l'oracle violée par cette séquence est la propriété 2. En effet, du point de vue de l'application, l'entrée "Cooper" pour le remplissage du champ "Address" ne fait pas partie de celle prévue dans la spécification de l'application. Cette erreur dans la conception du schéma ICARE provient du fait que nous avons construit le schéma de YellowPage v2, en factorisant les composants de type Langage d'Interaction pour la voix et pour le couple clavier – souris (myMappyFrame). Afin d'éliminer cette erreur, il convient de découper clairement le schéma ICARE pour la voix et pour le couple clavier – souris. Notre approche est donc mise en valeur, car elle montre notre propre erreur, commise lors de la conception de YellowPage. Cette erreur a été conservée dans le code final, puisque celui-ci est généré automatiquement avec l'éditeur de schéma ICARE. Notons que dans d'autres sections des résultats des tests, nous observons une erreur inverse, c'est-à-dire qu'un couple d'entrées champ Nom, spécification d'une adresse ont été fusionnées.

Proposition d'amélioration de YellowPage v2

A partir des erreurs constatées lors du test de l'assemblage en complémentarité du paragraphe précédent, nous modifions l'application YellowPage v2. Cette modification consiste à produire un assemblage en complémentarité pour chaque champ de texte : un pour le champ Nom, et un pour le champ Adresse. Cette proposition d'assemblage est présentée dans la

Figure 61. Dans cette figure, nous retrouvons deux composants de complémentarité, un pour la spécification du champ Nom (à gauche), et un pour la spécification du champ Adresse (à droite). Les composants "ViaVoiceInput" et "myMappyFrame" émettent des événements à l'ensemble des quatre composants de type Langage d'Interaction "CommandFilter". Ces derniers permettent ainsi de filtrer certaines commandes non reconnues. Dans le cas du composant "CommandFilter" à gauche pour la sélection du champ Nom, ce composant ignore les événements en entrée ne possédant pas une sens correspondant à la sélection du champ Nom. Les tâches abstraites "Specify Name" et Specify Address" ne sont pas de nouvelles entrées du contrôleur de dialogue : il s'agit en fait des entrées existantes qui avait été réservées à l'usage non complémentaire de la voix pour la spécification d'un nom et d'une adresse. Nous pouvons observer ces deux composants de types "tâche abstraite" sur la gauche de la Figure 29**.** Notons que nous n'avons pas encore testé ce nouvel assemblage.

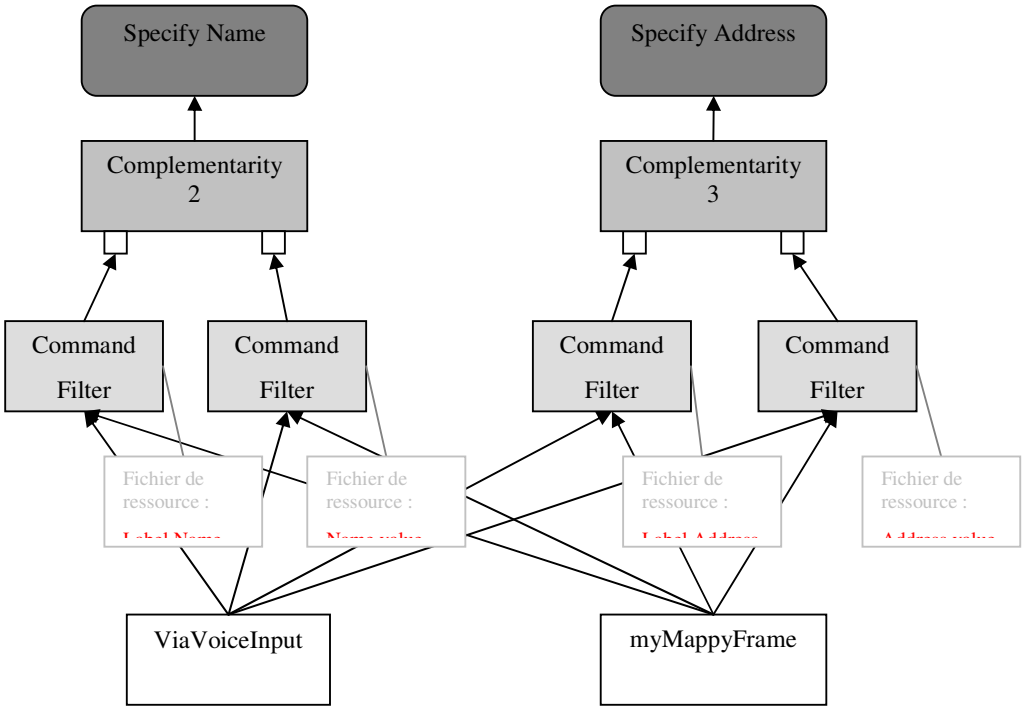

Figure 61 ***: proposition d'assemblages en complémentarité pour corriger l'erreur de fusion***

Bilan

Nous avons présenté notre proposition de générateur automatique de couplage : son implémentation et le fonctionnement du couplage produit. Nous pouvons dresser un bilan quant à l'efficacité de celui-ci. Nous identifions plusieurs points positifs :

- Notre couplage est opérationnel, notre exemple de test de YellowPage v2 le confirme.
- Notre couplage permet d'identifier clairement les entrées de l'application sous test, et une partie de ses sorties.
- Le code de couplage est produit automatique lors de l'édition du schéma ICARE. L'utilisateur (testeur) a peu de modifications à produire afin de réaliser un test.
- Notre proposition de couplage permet de tester les propriétés CARE [Coutaz 95] sur des applications multimodales dont la partie interactive est produite avec l'éditeur de

schéma ICARE. Nous avons d'ailleurs présenté un test qui montrait une erreur dans la conception de YellowPage pour l'usage complémentaire de modalités.

Notre proposition et sa réalisation ont néanmoins les défauts suivant :

- Notre solution ne permet pas d'identifier des points de connexion en sortie pour la partie de l'application autre que celle réalisée en ICARE.
- Notre générateur de code produit du code exécutable. Néanmoins, la sémantique du code produit est mise en cause et nécessite une intervention manuelle.
- Les fichiers d'aide aux tests se sont révélés peu lisibles. Ils ont néanmoins fortement contribué à la production du test sur l'assemblage de composants en complémentarité, mais ils peuvent être améliorés.

En synthèse, nous concluons que le générateur automatique de code de couplage ICARE – Lutess est un outil pertinent et opérationnel pour produire ce couplage, mais qu'il contient encore quelques défauts. Ces derniers constituent des perspectives à court terme à notre étude, comme exposées dans le chapitre suivant.

Conclusion

En conclusion à notre étude, nous proposons un résumé de nos contributions selon deux facettes, les résultats conceptuels et ceux pratiques. Une prise de recul par une analyse critique des résultats permet d'envisager pour nos travaux de multiples perspectives que nous organisons en deux parties : les extensions et les prolongements à plus long terme.

Contributions

Nos travaux contribuent au domaine de l'ingénierie logicielle de l'interaction multimodale en traitant le problème de la complexité de réalisation logicielle des systèmes interactifs multimodaux. La complexification de tels systèmes interactifs rend nécessaire leur développement et leur validation rigoureux. Pour cela, nous avons étudié l'application de techniques de test de logiciels réactifs synchrones à la validation de systèmes interactifs multimodaux, en s'appuyant sur l'hypothèse suivante : un système interactif peut, sous certaines conditions, être considéré comme un système réactif synchrone. S'il est vrai que la rapidité de réaction de systèmes interactifs à un événement externe est moins cruciale que dans le cas des systèmes traditionnellement qualifiés de synchrones, on peut toutefois constater que leurs comportements sont constitués de cycles "action-réaction". Des interactions multimodales correspondant à des comportements des utilisateurs ainsi que certaines propriétés du système et de son contexte peuvent s'exprimer dans un formalisme synchrone à des fins de tests. Cette hypothèse sous-jacente à notre étude est confirmée par les résultats obtenus.

Dans notre étude, nous motivons d'abord nos travaux en les situant dans le domaine de l'IHM. Notre objectif de vérification formelle par le test constitue une méthode d'évaluation formelle et prédictive (chapitres 2 et 3). Nous focalisons ensuite sur l'interaction multimodale (chapitres 4 et 5). En particulier la démarche adoptée pour mener ces travaux s'appuie sur l'identification de propriétés caractéristiques de la multimodalité. Au chapitre 4, nous cernons la conception de l'interaction multimodale au choix motivé de propriétés et en particulier les propriétés CARE (Complémentarité, Assignation, Redondance et Equivalence).

Notre contribution se présente alors sous deux formes complémentaires :

- une méthode de vérification d'applications interactives multimodales au moyen d'un outil de vérification de systèmes réactifs synchrones de tests (chapitres 8 et 9), et
- une plateforme logicielle de tests (chapitre 10).

La méthode de vérification repose sur l'expression de propriétés logiques et temporelles à vérifier dans un fichier d'oracle et qui correspondent à des propriétés ergonomiques de la multimodalité. En effet, l'originalité de notre approche réside dans la vérification formelle de propriétés ergonomiques propres à la multimodalité. La méthode de vérification préconise aussi différents niveaux d'abstraction pour les tests qui sont identifiés à partir de l'architecture logicielle de l'application sous test que nous considérons selon le modèle à composants ICARE (chapitre 8).

La plateforme logicielle de tests que nous avons développée constitue une instrumentation de notre méthode. Elle repose sur le couplage de la plateforme ICARE avec celle Lutess. Notre plateforme de tests consiste donc en un générateur du code de connexion des composants ICARE avec Lutess. Nous l'avons conçue et développée selon une approche incrémentale, en menant d'abord une étude de faisabilité en développant manuelle la connexion puis nous avons conçu et développé le code qui génère automatiquement la connexion ICARE-Lutess. Les résultats de tests obtenus pour l'application multimodale YellowPage (chapitre 7)

soulignent d'une part le caractère fonctionnel de l'instrumentation automatique, et d'autre part, son adéquation pour la vérification des propriétés ergonomiques CARE propres à la multimodalité.

En synthèse, les résultats de notre étude sont à la fois conceptuels et pratiques.

- Les résultats conceptuels incluent :
  - La vérification de notre hypothèse de travail : un système interactif peut, sous certaines conditions, être considéré comme un système réactif synchrone.
  - Une méthode de vérification d'applications interactives multimodales au moyen d'un outil de vérification de systèmes réactifs synchrones de tests.
  - L'expression formelle de propriétés ergonomiques et la façon de les vérifier avec notre méthode.
  - Des niveaux d'abstraction différents de tests d'utilisabilité basés sur l'architecture logicielle à composants ICARE.
- Les résultats pratiques (réalisations logicielles et tests) comprennent :
  - un générateur du code de connexion des composants ICARE et Lutess. Celui-ci est complètement intégré dans la plateforme ICARE et prend en compte l'assemblage dynamique des composants dans l'éditeur graphique ICARE. Nous avons ainsi défini une plateforme de conception-développement rapide d'interfaces multimodales et de vérification ergonomique prédictive formelle.
  - Le développement de la connexion manuelle des composants ICARE et Lutess dans l'application multimodale existante YellowPage-V1 pour effectuer un ensemble de tests formels sur l'application.
  - La conception et le développement d'une nouvelle version de l'application multimodale YellowPage-V2 pour tester notre plateforme de vérification basée sur le couplage automatique ICARE-Lutess.

L'approche de vérification basée sur le couplage d'ICARE avec Lutess est opérationnelle et efficace. L'originalité de nos travaux réside dans sa pluridisciplinarité par la vérification formelle de propriétés ergonomiques. Il convient néanmoins de souligner que nos travaux ne concernent que la vérification de la partie du système interactif dédiée à l'interaction concrète entre l'utilisateur et le système. Le reste du système comme le contrôleur de dialogue qui gère l'enchaînement des tâches n'est pas traité dans nos travaux, ceci limitant les propriétés ergonomiques à vérifier.

Extensions et prolongements

Le développement d'une plateforme de tests formels est certes une tâche ambitieuse dans le temps imparti d'un Master Recherche. Nos extensions à court terme visent donc à améliorer la plateforme :

- Corriger le générateur de code du couplage ICARE - Lutess, afin qu'il produise un code parfaitement lisible et simple pour sa modification manuelle lors de tests d'une application donnée. En l'état, obtenir un code correct par modifications du code généré nécessite environ 4 heures de travail.
- Améliorer la génération des fichiers d'aide à l'utilisateur, le testeur. Ceux-ci sont volumineux et pas suffisamment clairs, lorsque le nombre de composants ICARE et donc de liens augmente. Le passage à grande échelle comme le test d'une application multimodale complexe soulève donc des problèmes de lisibilité de la documentation.
- Revoir et modifier les assemblages des composants ICARE développés de la deuxième version de YellowPage en scindant lorsque c'est nécessaire les assemblages de composants, puis effectuer à nouveau des tests pour valider la plateforme.

A plus long terme, nous envisageons une intégration encore plus forte de la plateforme ICARE et de l'outil Lutess pour définir un environnement de développement et de test dédié à l'interaction multimodale qui soit complètement intégré, permettant ainsi de tester les assemblages ICARE *au cours de leur production*.

Comme autres prolongements à nos travaux, nous souhaitons étudier d'autres propriétés ergonomiques que CARE. Ce travail se basera sur une revue des propriétés et règles ergonomiques et sans nul doute une classification en vue de leur formalisation. De plus leurs vérifications formelles impliqueront de nouveaux points de contact au sein de l'architecture logicielle de l'application interactive sous test. Nous avons juste commencé cette étude et avons constaté que pour certaines propriétés comme l'observabilité, des points de contact sont nécessaires avec d'autres composants que ceux d'ICARE. Ce constat ouvre un vaste programme de recherche.

Enfin, pour le guidage de la génération des tests, nous souhaitons approfondir le lien existant entre les scénarios d'utilisabilité qui peuvent être issus de l'analyse de la tâche dans le cadre d'une conception itérative centrée sur l'utilisateur et les scénarios de guidage des tests utilisés par Lutess. Ce travail peut aboutir à intégrer un outil de spécification des tâches comme CTTE (Concurrent Task Tree Environment) ou MAD à notre plateforme ICARE-Lutess. Cette intégration rejoint notre objectif de vérifier d'autres propriétés ergonomiques que CARE et en particulier les propriétés ergonomiques qui concernent le contrôleur de dialogue comme celle de "dialogue à plusieurs fils d'activité".

Bibliographie

[Abowd 92] G. Abowd, J. Coutaz et L. Nigay, *Structuring the Space of Interactive System Properties*, Proceedings of the IFIP TC2/WG2.7 on Engineering for Human- Computer Interaction, Ellivuori, Finlande, 10-14 august 1992, pp. 113-128.

[Aït-Ameur 03] Y. Aït-Ameur, M. Baron, P. Girard, *Formal verification of HCI user tasks*, HESSD, In Proceedings of The 2003 International Conference on Software Engineering Research and Practice - SERP 2003 (June 23 - 26, 2003, Las Vegas, Nevada USA), CSREA Press, 2003, pp. 732-738.

[Ausbourg 98] *B.* Ausbourg, *Using Model Checking for the Automatic Validation of User Interface Systems*, International Eurographics Workshop on Design, Specification and Verification of Interactive Systems*(*DSVIS), Abingdon, United Kingdom, June 3-5, 1998, pp. 242-260.

[Balbo 95] S. BALBO, *Automatic evaluation of user interface usability: Dream or reality*. In S. Balbo, Ed., Proceedings of the Queensland Computer-Human Interaction Symposium*,* Queensland, Australia, August, Bond University, 1995.

[Berstel 05] J. Berstel, S. Crespi Reghizzi, G. Roussel, P. San Pietro, *A scalable formal method for Design and checking of user interfaces*, ACM Transactions on software Engineering and Methodology, Vol. 14, No. 2, April 2005, pp. 124-167.

[Bouchet 04] J. Bouchet, L. Nigay, T. Ganille, *ICARE Software for Rapidly Developing Multimodal Interface*, Proceedings of the 6th international conference on Multimodal interfaces (ICMI), new York, 2004, pp. 251-258.

[Bourguet 02] M. L. Bourguet, *Outil de Prototypage pour la Conception et l'Évaluation d'interfaces Utilisateur Multimodales*, in 14th French-speaking conference on human-computer interaction (IHM02), Poitiers, 26-29 November 2002, pp. 239-242.

[Bourguet 03] M. L. Bourguet, *How Finite State Machines can be used to build error free multimodal interaction systems*, in The 17th British HCI Group Annual Conference vol. 2, pp. 81-84, September 2003.

[Bourguet 04] M. L Bourguet, *Software design and development of multimodal interaction*, In the IFIP 18th World Computer Congress Topical Days,

chapter. Multimodal Interaction, Toulouse, august 2004.

[Carr 97] D. A. Carr, *Interaction Object Graphs: An Executable Graphical Notation for Specifying User Interfaces*, Formal Methods for Computer-Human Interaction, P. Palanque and F. Paterno', editors, ISBN 3-540-76158-6, Springer-Verlag, 141-156, Nov. 1997

[Coutaz 95] J. Coutaz, L. Nigay, D. Salber, A. Blandford, J. May, R. Young, *Four Easy Pieces for Assessing the Usability of Multimodal Interaction : The CARE properties*, Proceedings of the INTERACT'95 conference, S. A. Arnesen, D. Gilmore Eds., Chapman & Hall, Lillehammer, Norway, pp. 115-120, 1995

[Ivory 01 M. Y. Ivory, M. A. Hearst, *The state of the Art in Automating Usability Evaluation of User Interfaces*, ACM Computing Surveys, Vol. 33, No. 4, December 2001, pp 470-516.

[Jambon 02] F. Jambon, *From formal specifications to secure Implementation*, Computer-Aided Design of User Interfaces (CADUI'2002), edited by Kolski, Christophe and Vanderdonckt, Jean, Valenciennes, France, Kluwer Academics, 2002, pp. 43-54.

[Madani 05] L. Madani, C. Oriat, I. Parissis, J. Bouchet, L. Nigay, *Synchronous Testing of Multimodal Systems: an Operational Profile-Based Approach*, Proceedings of the International Symposium on Software Reliability Engineering (ISSRE), Chicago, Illinois, USA, november 8-11, 2005

[MacCall 77] J. McCall., *Factors in software quality*, general electric edition 1977. extract présenté dans le cours IHM de Master 2 R par J. Coutaz.

[Nielsen 94] J. Nielsen, *Heuristic evaluation*. In Nielsen, J., and Mack, R.L. (Eds.), Usability Inspection Methods, John Wiley & Sons, New York, NY. 1994.

[Nigay 95] L. Nigay, J. Coutaz, *A Generic Platform for Addressing the Multimodal Challenge*. Proc. Of CHI'95, 1995, 98-105.

[Nigay 97] L. Nigay, J. Coutaz, *Multifeature Systems : The CARE Properties and Their Impact on Software Design*, in Intelligence and Multimodality in Multimedia Interfaces, AAAI Press, 1997

[Palanque 03] P. Palanque, A. Schyn*, A Model-Based Approach for Engineering Multimodal Interactive Systems,* Human – Computer – Interaction – INTERACT'03, Editor : M. Rauterberg et al., Published by IOS Press, IFIP, 2003, pp.543-550.

[Palen 02] L. Palen, *Beyond the Handset: Designing for Wireless Communications usability*. ACM Transactions on Computer-Human Interaction, 9(2), june 2002, p. 125-151.

[Parissis 96] I. Parissis, F. Ouabdesselam, *Specification-based Testing of Synchronous Software*, Proceeding of ACM SIGSOFT Fourth Symposium on the Foundations of Software Engineering, ACM Press, pp. 127-134, 1996.

[Scapin 90] D. L. Scapin, *Des critères ergonomiques pour l'évaluation et la conception d'interfaces utilisateurs*, Actes du XXVIème congrès de la SELF, Montréal, Canada,1990.

[Schyn 03] A. Schyn, D. Navarre, P. Palanque*, Description formelle d'une technique d'interaction multimodale dans une application de réalité virtuelle immersive*, IHM 2003, november, pp.150 – 157.

[Shneiderman 98] B. Shneiderman, *Designing the User Interface*. Addison-Wesley Longman Inc., Reading Massachusetts, 1998

[Vernier 00] F. Vernier, L. Nigay, *A Framework for the Combination and Characterization of Output Modalities*, Actes de la conférence DSVIS'2000, Limerick, Irelande, 5-6 juin 2000, pp. 32-48.