Projet VERBATIM

Sous-projet 3 – Lot 5

CASTING et test d'IHM multimodales

Version : 1.2

Date : 17/01/07

Auteur : Lionel VAN AERTRYCK (Silicomp-AQL)

LSR

Table des matières

1. Introduction

1.1 Rappel de l'objectif

Le présent document est effectué à l'occasion du projet RNRT VERBATIM, dont l'objet est la VERification Biformelle et Automatisation du Test des Interfaces Multimodales.

Note : Les documents sont rédigés généralement en Français mais peuvent comprendre des parties rédigées en Anglais car certaines travaux ont des connexions internationales pour des contextes de coopération ou de publication.

Ce document a pour objectif de présenter les travaux réalisés dans le cadre du projet VERBATIM à partir de la méthodologie de génération automatique de test CASTING.

1.2 Historique

Date	Version	Commentaire
03/06/06	1.0	Création du document.
15/01/07	1.1	Version finale

1.3 Référence

Abréviation	Référence
[Van98]	Lionel Van Aertryck. *Une méthode et un outil pour l'aide à la génération de jeux de tests de logiciels*. Thèse de doctorat de l'université de Rennes I, 1998. (ftp://ftp.irisa.fr/techreports/theses/1998/van_aertryck.ps.gz)
[VBL97a]	Lionel Van Aertryck, Marc Benveniste et Daniel Le Métayer. *CASTING : une méthode formelle pour la génération de tests*. Actes des journées AFADL (Approches Formelles dans l'Assistance au Développement de Logiciel), pp 99-112, 28-29 mai 1997, Toulouse, France.
[VBL97b]	Lionel Van Aertryck, Marc Benveniste et Daniel Le Métayer. *CASTING: A formally based software test generation method*. IEEE International Conference on Formal Engineering Methods (ICFEM'97), première édition. 12-14 novembre 1997, Hiroshima, Japon.
[INTERACT95]	J. Coutaz, L. Nigay, D. Salber, A. Blandford, J. May, and R. Young. *Four easy pieces for assessing the usability of multimodal interaction : The CARE properties*. In IFIP Int. Conf. on Human-Computer Interaction Interact'95, pages 115–120, London, 1995. Chapman & Hall.

2. Présentation de CASTING

2.1 Introduction

CASTING (**[VBL97a]**, **[VBL97b]** et **[Van98]**) est une méthode de génération de tests qui consiste à appliquer de manière systématique une stratégie de test à une spécification. Elle apporte une solution efficace en adoptant une approche fondée sur le traitement syntaxique de documents d'entrée structurés et sur l'utilisation et la résolution de contraintes.

CASTING est une méthode de génération de tests à partir de documents formalisés. Elle est générale dans le sens ou elle peut s'appliquer à différents formalismes. Deux phases principales composent le processus de génération de tests. La première consiste à extraire du document d'entrée les informations qui sont ensuite utilisées dans la seconde phase pour produire les scénarios de tests. Seule la phase d'extraction est dépendante du formalisme du document d'entrée et nécessite d'être adaptée pour pouvoir traiter un nouveau formalisme.

L'architecture de CASTING est conçue de façon à exploiter le fait qu'un certain nombre de techniques de test ne dépendent pas de la forme du document d'entrée. L'unique exigence imposées pour CASTING est que ce document soit exprimé dans un formalisme disposant d'une syntaxe définie formellement. Deux phases principales composent le processus de génération de tests. La première consiste à extraire du document d'entrée les informations qui sont ensuite utilisées dans la seconde phase pour produire les suites de tests. Seule la phase d'extraction est dépendante du formalisme du document d'entrée. Aussi, la spécialisation de la première phase à un formalisme donné est-elle obtenue par la définition de règles d'extraction spécifiques. Cette charge est dévolue à un utilisateur particulier de la méthode : l'administrateur. La seconde phase repose sur différentes techniques qui sont encapsulées dans des modules qui composent le noyau de CASTING. Par conséquent, l'essentiel de la méthode présentée ici n'est pas lié à un formalisme d'entrée particulier : en fait, CASTING permet d'exploiter aussi bien des spécifications formelles, du code source, des annotations de l'utilisateur qu'une combinaison de toutes ces informations. En d'autres termes, le cadre d'utilisation de CASTING couvre à la fois les tests fonctionnels et les tests structurels.

Le second objectif est d'offrir à l'utilisateur les moyens d'adapter la méthode à ses besoins. Pour y répondre, nous avons intégré différents moyens d'interaction, les plus significatifs étant les suivants : l'utilisateur peut choisir parmi plusieurs stratégies de test fournies par l'administrateur. Il dispose aussi des moyens pour définir lui-même des stratégies de test, adaptées aux formalismes des documents d'entrée. L'utilisateur peut fournir des cas de test supplémentaires de façon à compléter une stratégie de test donnée. Cette possibilité est nécessaire pour inclure des cas de test précis qui, par expérience, sont connus pour être utiles dans les suites de tests.

La méthode inclut une phase de résolution de contraintes qui peut nécessiter une aide extérieure lorsqu'elle échoue à produire des données de test qui satisfont une description de cas de test. Cette interaction avec l'utilisateur lors de la phase de résolution de contraintes est nécessaire de manière à pallier la non-décidabilité potentielle des problèmes traités.

De notre point de vue, l'automatisation de la production de jeux de tests ne doit pas s'arrêter à la génération de cas de test, car un cas de test pris isolément ne contient pas suffisamment d'indications sur la manière de le soumettre à une implantation. Dans le cadre des applications à états que nous considérons, ceci revient à positionner l'implantation dans un état qui satisfasse les conditions du cas de test. Aussi proposons-nous une technique intégrant la notion d'état et de changement d'état qui permet de produire des suites de tests incluant la mise en situation de l'implantation nécessaire au passage des cas de test.

Finalement, le système que nous proposons repose sur des techniques bien établies telles que les grammaires attribuées et les graphes. De plus, les hypothèses d'uniformité [Gau95] qui sont à l'origine de chaque cas de test engendré sont produites automatiquement. Ces hypothèses sont la justification formelle des cas de test. Il est important de préciser qu'elles sont exprimées dans un langage qui est indépendant du formalisme utilisé pour les documents d'entrée.

2.2 Présentation technique

2.2.1 Une génération en trois étapes

La génération des tests est décomposée en trois étapes dans CASTING : l'extraction de l'ensemble des spécifications de cas de test (*sct*), la génération d'un graphe d'états symboliques et la génération des suites de tests qui forment le jeu de tests J à partir de ce graphe.

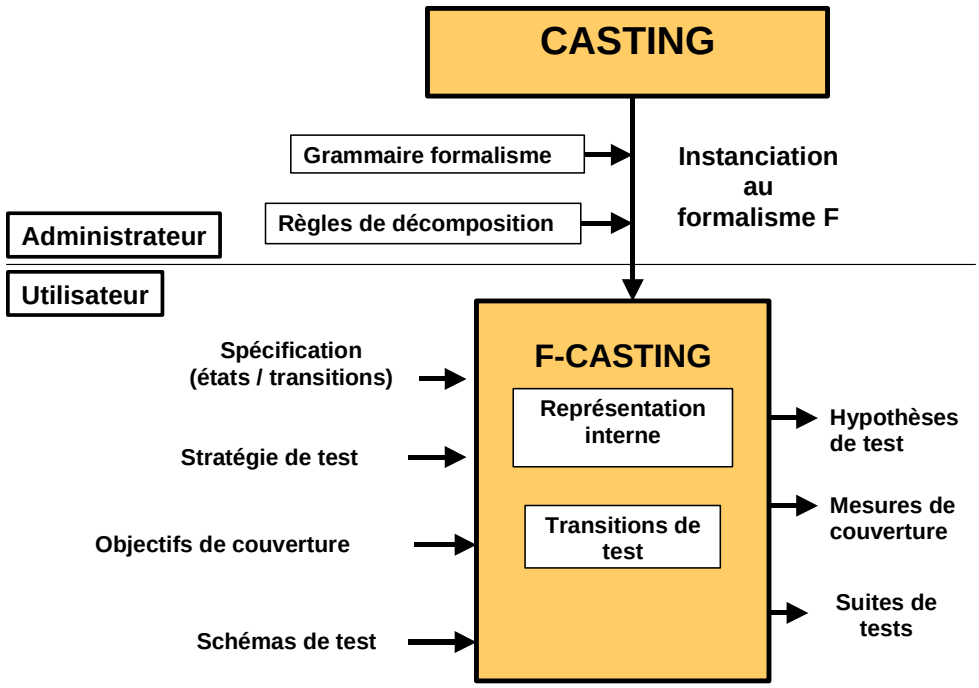

Figure 1 : Architecture de la méthodologie

Il appartient ainsi à un « administrateur » d'instancier la méthode à un formalisme donné en définissant les règles de décomposition associées ainsi que sa traduction dans le langage intermédiaire de l'outil.

L'utilisateur utilise alors la version instanciée de CASTING pour générer ses tests en appliquant la stratégie de test aux opérations présentes dans les spécifications fonctionnelles. Il obtient ensuite un ensemble de cas particuliers d'utilisation des fonctions susceptibles de donner lieu à des tests pertinents. La Figure 2 présente les étapes de décomposition des expressions de la spécification qui conduisent à la définition de cas de test précis.

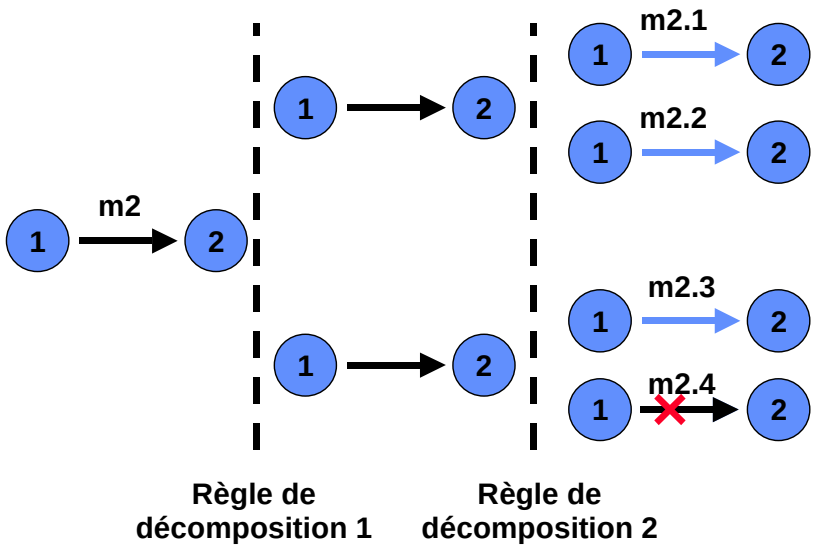

Figure 2 : Processus de génération des tests

Un filtre permet d'éliminer une partie des transitions de test produites de manière automatique mais pour lesquelles il n'existe pas de solution (exemple : m2.4).

Les transitions de test produites sont des « spécifications » de cas de test dans le sens où elles donnent une description symbolique, sous forme d'expressions logiques, d'un ensemble de cas de test.

Les transitions de test constituent le cœur de la représentation interne (un graphe) qui sera utilisée pour produire les suites de tests.

2.2.2 Le document d'entrée

La méthode CASTING repose sur le constat qu'une part très importante des cas de test correspondant à une stratégie de test donnée peuvent être obtenus automatiquement à partir du texte des spécifications fonctionnelles.

L'utilisation de CASTING repose sur la disponibilité d'une spécification écrite dans un langage disposant d'une syntaxe (langage de programmation, langage formel tel que celui de la méthode B par exemple). Cette restriction est indispensable dès lors que l'on souhaite construire une méthode automatique sur laquelle doit pouvoir s'appuyer un outil.

Le format < Variable d'états, Invariant, ensemble d'opérations (décrites avec des Préconditions et Posconditions) > choisi pour le document d'entrée est très peu contraignant. Tout en fournissant suffisamment d'informations pour la génération de jeux de tests, il reste tout à fait général et n'exclut aucun des formalismes auxquels nous avons pu confronter notre méthode jusqu'à présent. Par exemple, des langages de spécifications ensemblistes comme Z, VDM ou B s'inscrivent directement dans ce cadre (Pre et Post peuvent être extraites directement des schémas, fonctions ou machines abstraites). Les langages de spécifications algébriques sont des cas particuliers où Ve et I ne jouent aucun rôle puisque la notion d'état est absente de ce type de spécification. Les opérations sont des fonctions et les axiomes correspondent typiquement à des postconditions (puisqu'ils mettent en jeu les paramètres et les résultats). Pour ce qui est des langages de programmation traditionnels, Pre et Post sont les propriétés des opérations déduites de la sémantique du langage.

2.2.3 La stratégie de test

Une stratégie de test consiste ici en un ensemble de règles de décomposition qui sont appliquées aux expressions présentes dans les spécifications fonctionnelles afin de mettre en évidence et d'expliciter des cas de test potentiellement pertinents.

Une stratégie de test est l'expression formelle d'objectifs de test. Au niveau le plus fin de détail, une stratégie de test est définie par des règles de décomposition pour chacun des opérateurs du langage cible. La définition de ces règles est réalisée par l'administrateur. Certaines règles décomposent le domaine d'entrée d'opérateurs tandis que d'autre se chargent de synthétiser les partitionnements et identifier les cas de test pour la fonction testée. Ceci est réalisée en reportant les décompositions identifiées aux niveaux élémentaires vers le niveau global de la spécification.

2.2.4 Extraction des spécifications de cas de test

Le point de départ du processus d'extraction est l'objectif de test de l'utilisateur. Par exemple, l'utilisateur peut décider que seules les informations directement présentes dans la définition des opérations doivent être prises en compte pour mettre en évidence des spécifications de cas de test. Il doit donc établir une correspondance entre cet objectif de test et les éléments de la spécification qui constitue le document d'entrée de la méthode.

L'utilisateur choisit ensuite une stratégie de test (c'est-à-dire des valeurs de paramètres des règles d'extraction) exprimant son intention. Le processus d'extraction consiste à appliquer une stratégie donnée à une spécification écrite dans un formalisme d'entrée pour produire un ensemble de sct (spécification de cas de test).

Une fois la stratégie choisie, elle est appliquée de manière systématique à l'ensemble de la spécification. Un ensemble de cas exprimés de façon symbolique est ainsi associé à chaque fonction de l'application à tester. Chaque spécification de cas représente un sous-ensemble du domaine d'entrée de la fonction initiale, ce qu'en d'autres termes nous appelons une *spécification de cas de test*. Ces *spécifications de cas de test* sont des descriptions de tests fonctionnels car ils sont établis à partir de la spécification du logiciel à tester.

Toutes les opérations de la spécification doivent être représentées dans S. Les suites de tests sont construites à partir des spécifications de cas de test et non des opérations. Aussi, si le domaine d'application d'une opération n'est pas complètement couvert par les spécifications de cas de test, certains états peuvent ne pas être accessibles par application des sct alors qu'ils sont accessibles par application des opérations. Cette propriété est importante pour assurer que l'impossibilité d'appliquer une sct dans les suites de tests n'est pas due à une décomposition des opérations qui ne couvre pas le domaine d'entrée de chacune d'entre elles.

Il faut par ailleurs garder en tête le fait qu'un ensemble S peut contenir des spécifications de cas de test qui ne sont pas applicables parce que leur domaine d'application correspond à des états inaccessibles même s'ils respectent l'invariant.

2.2.5 Ajout de cas de test par l'utilisateur

A l'issue de la phase d'extraction des informations de test, l'utilisateur dispose d'un ensemble de sct qui décrivent précisément les cas de test qui seront pris en compte lors des étapes suivantes. L'utilisateur peut fournir des spécifications de cas de test supplémentaires qui sont ensuite traitées comme les sct produites dans la première phase.

2.2.6 Génération du graphe d'états symboliques

Cette étape consiste à engendrer les suites de tests qui permettront d'instancier les sct dans des séquences valides d'appels d'opérations de la spécification. On construit dans ce but un graphe d'états symboliques qui recense les successions de sct possibles. L'objectif est ensuite de calculer des jeux de tests couvrant chaque sct au moins une fois, c'est-à-dire sélectionnant suffisamment de chemins pour appliquer au moins une fois chaque sct.

L'étape suivante consiste à construire des *scénarios de tests* qui vont permettre d'enchaîner ces *spécifications de cas de test* tout en choisissant les données de test qui correspondent à ces enchaînements. Cette étape est dévolue à un solveur de contraintes qui détermine les

enchaînements possibles d'opérations, attribue des valeurs aux variables et calcule les résultats attendus.

Le processus de génération du graphe est itératif. Il débute par l'application des sct qui correspondent à l'opération d'initialisation (dont l'existence est un pré-requis à l'application de la méthode). Le premier état atteint est défini par la postcondition de l'opération d'initialisation. Ensuite, un algorithme de recherche identifie toutes les sct applicables. Dans ce but, les contraintes qui apparaissent dans les préconditions de chaque spécification de cas de test sont ajoutées successivement à l'état courant et le processus de résolution de contraintes est appliqué à chacun d'entre eux.

Pour qu'une spécifications de cas de test apparaisse dans le graphe il faut que le solveur parvienne à trouver une solution ou que l'utilisateur lui en fournisse une directement (vérifiée par le solveur). Dans les deux cas, il existe une solution et la sct est donc correctement appliquée.

L'ensemble des spécifications de cas de test obtenues lors de la phase d'extraction peut éventuellement contenir des sct invalides (dont la définition comporte des contradictions) ou des sct valides mais qui ne peuvent pas être atteintes par applications successives des opérations autorisées par la spécification. Dans les deux cas, le processus de génération du graphe peut ne pas se terminer car il tente d'appliquer chaque spécification de cas de test au moins une fois. L'utilisateur doit donc imposer une limite (en terme de taille du graphe ou de durée maximale de résolution) afin d'obtenir un résultat en un temps fini. Ceci constitue une nouvelle hypothèse de test qui doit être associée aux suites de tests lorsqu'elle intervient dans le processus de construction du graphe.

2.2.7 Génération des suites de tests à partir du graphe

Cette étape produit un ensemble de suites de tests (ou jeu de tests) J à partir d'une sélection des chemins du graphe. Une suite de tests est une séquence autorisée (par rapport au document d'entrée) d'appels des opérations de l'application testée auxquels sont associées les propriétés que les sorties correspondantes doivent satisfaire.

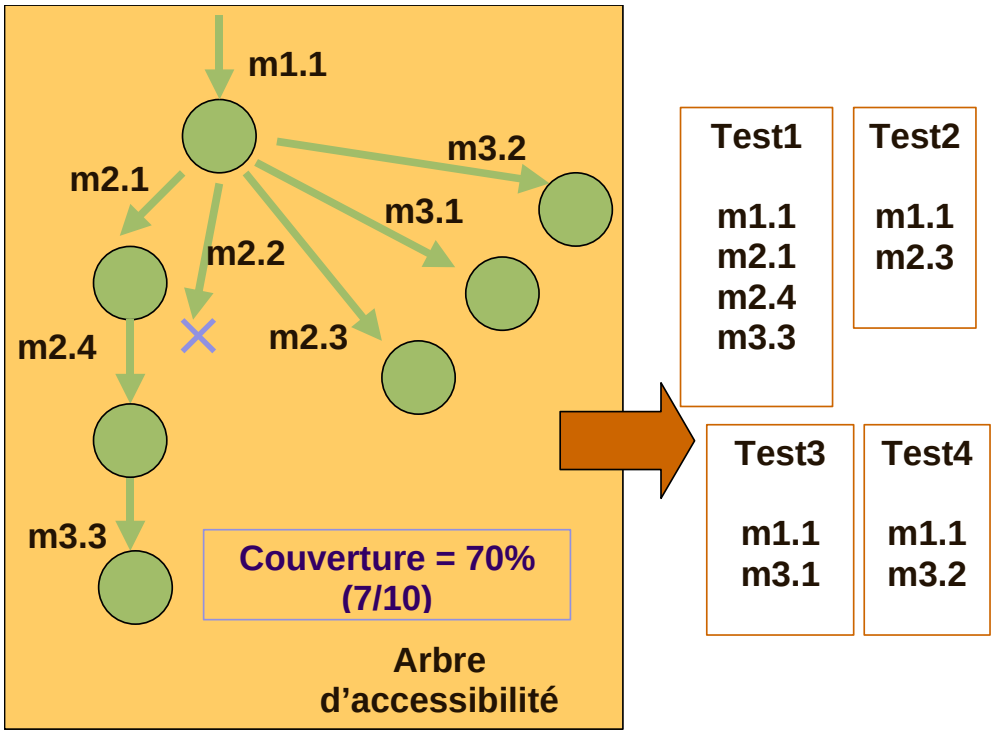

Figure 3: génération des suites de test

Cette étape passe par l'utilisation d'un solveur de contraintes avec une stratégie de résolution appropriée. L'intervention de l'utilisateur peut être sollicitée lorsque la procédure de résolution échoue.

Notons que le graphe est construit de telle façon que toutes les opérations sont appliquées sous leur précondition. Ces préconditions sont la traduction fidèle de la spécification initiale. Par conséquent, les suites de tests qui en sont extraites sont conformes à la spécification par construction.

La sélection de chemins se fait suivant un algorithme qui permet de produire les suites de tests les plus courtes possibles couvrant toutes les sct. Cette stratégie de construction des suites de tests, couramment adoptée dans la pratique, permet d'appliquer au plus tôt le plus de sct tout en limitant les répétitions d'application de sct dans les suites de tests de J.

Pour conclure ce paragraphe, nous dirons que la stratégie de génération des suites de tests se compose d'une stratégie de génération du graphe et d'une stratégie de parcours de celui-ci, les deux stratégies étant intimement liées.

L'outil applique ainsi des techniques de parcours de graphes et de résolution de contraintes (recherche des données de test) pour produire des suites de test permettant d'atteindre le taux de couverture annoncé.

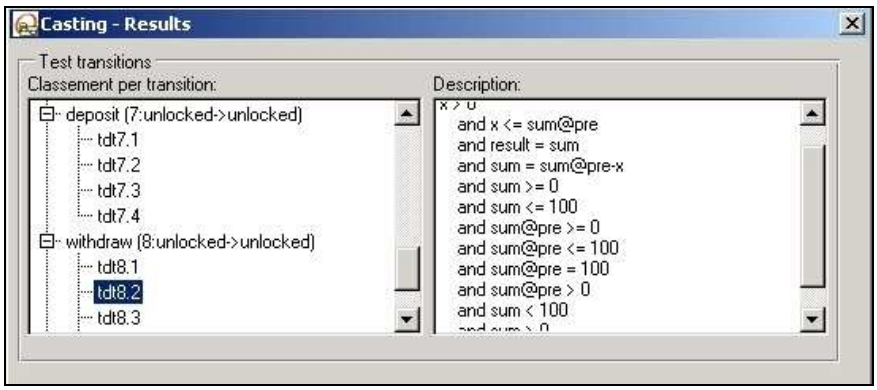

Figure 4 : Spécification des tests symboliques

Les mesures de couverture sont données par opération et par test.

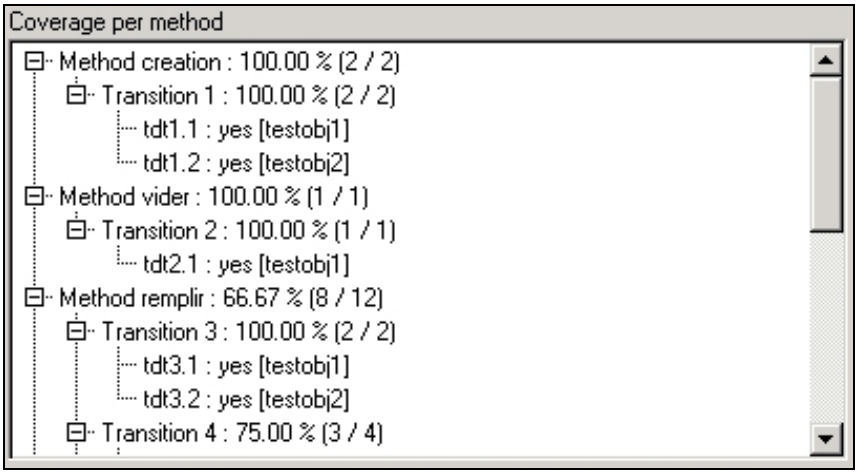

Figure 5 : Taux de couverture obtenus

2.3 CASTING et UML

Le projet COTE (RNTL 2001-2002) a permis d'étudier l'application de la méthode CASTING à UML. Au terme de cette étude nous disposions d'un outil capable d'analyser un modèle UML simple et de produire des cas de test ainsi que des mesures de couverture des méthodes de la spécification UML par les tests produits.

Cet outil présente les caractéristiques suivantes :

- Analyse de modèles UML et extraction automatique d'information, en exploitant en particulier les annotations OCL (Object Constraint Language) présentes dans le modèle.
- Production automatique de cas de test et des suites de test (complètement instanciées) : une valeur est choisie pour tous les paramètres des fonctions).
- Confrontation entre des tests spécifiés « manuellement » et la spécification, et possibilité de recherche de valeurs d'entrée permettant de compléter des tests « incomplets ».
- Interface intégrée dans l'atelier de modélisation Objecteering de Softeam.

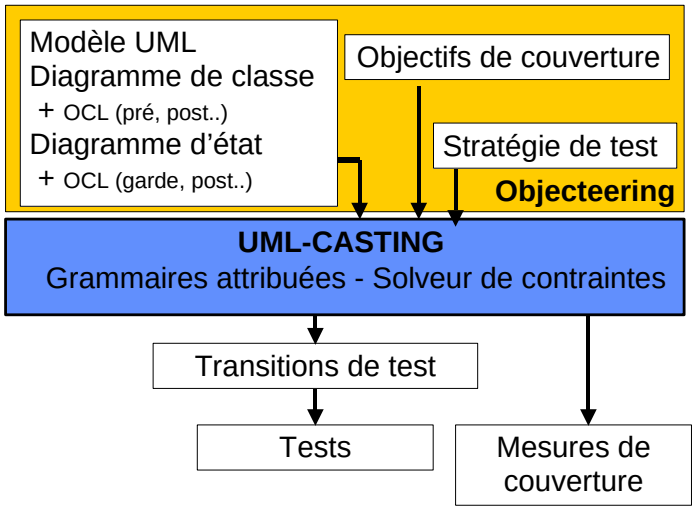

Figure 6 : UML-CASTING

2.4 Conclusion

Les avantages de cette solution proviennent directement de la première étape, à savoir la formalisation des spécifications fonctionnelles. En effet, il est alors possible de produire des tests conformes par construction aux spécifications et présentant des caractéristiques de couverture des spécifications directement mesurables. De plus, toute modification des spécifications fonctionnelles peut donner lieu à la production d'un nouveau jeu de test, à moindre coût, du fait de l'automatisation de cette tâche.

Sur le plan industriel, cette méthode à fait l'objet de plusieurs expériences et d'autres sont en projet. Lors des premières phases de conception, la méthode a été appliquée avec succès à une application industrielle de gestion de base de données dans le domaine des télécommunications. Une spécification formelle en Z a été réalisée et des scénarios de test ont été produits par application manuelle de la méthode. Les personnes en charge de la maintenance de l'application ont passé les tests et les ont jugés pertinents. Une étude à été menée sur le langage COBOL qui à conduit à un projet d'outil de génération automatique de tests pour ce langage en collaboration avec la société Sodifrance. Un projet européen (le projet T.W.O : Test and Warning Office) impliquant l'un des leaders du marché des outils de

test en Europe (Attol TestWare) bénéficie du soutien de la commission européenne. Ce projet repose en partie sur CASTING en ce qui concerne la production de données de test. L'objectif était alors de construire un outil de génération de tests pour des programmes C et C++.

En parallèle du projet VERBATIM, Silicomp-AQL est aussi impliquée dans un projet RNTL qui s'appuie en partie sur la méthodologie CASTING : le projet CASTLES (2003-2006). L'objectif principal du projet est de définir un environnement pour l'automatisation de la certification de la plate-forme et des applications Javacard, et d'évaluer cet environnement dans le contexte de la certification industrielle selon les Critères Communs.

3. Test d'IHM multimodales avec CASTING

3.1 Introduction

L'objectif est d'étendre notre méthode de génération de test à de nouveaux domaines, et en ce qui concerne le projet VERBATIM au test des IHM et à la vérification de propriétés d'ergonomie, tout en étudiant les complémentarités avec les approches de vérification de programmes développées par les autres partenaires du projet : analyse de code (CERT), preuve de programmes (ClearSy), génération de tests à partir de spécification écrites en B-événementiel (LISI).

Dans le cadre du projet VERBATIM, nous avons travaillé selon deux axes. Le premier axe concerne l'adaptation de CASTING au langage d'entrée de spécification B événementiel afin de produire des tests de respect de propriétés d'ergonomie pouvant être exprimées sous la forme de séquences d'appel de méthodes B. Le second axe concerne la connexion entre notre outil et un outil d'animation de spécification développé par le partenaire LISI.

Pour rappel, avec CASTING nous nous plaçons dans le cadre de la vérification de propriétés par le test et non par la preuve. Il est entendu que ces deux approches sont complémentaires, et que cette complémentarité devra être étudiée par ailleurs. Les tests ne permettent pas de prouver formellement que ces propriétés sont respectées mais apportent des éléments qui renforcent la confiance qu'elles le sont, ou qu'elles ne le sont pas en apportant un contre exemple violant ces propriétés.

Le test d'une propriété consiste à identifier des séquences d'appels de fonctions d'interaction de l'application qui mettent en évidence le respect d'un aspect de la propriété d'ergonomie. Transposé dans CASTING, un aspect est le résultat de l'application des règles de décomposition au texte des propriétés et de la spécification de l'application.

Considérons par exemple la recommandation suivante « *toute action doit être accessible en moins de N interactions avec l'application* ». Cette recommandation est exprimée de manière générale et s'applique à toutes les actions offertes par l'application. En première analyse, les tests qui vont permettre de vérifier cette propriété d'ergonomie vont consister à vérifier pour un certain nombre d'actions, que ces dernières sont effectivement toujours accessibles en moins de N interactions entre l'utilisateur et l'application.

Tout d'abord, cette propriété ainsi que les éléments qu'elle implique nécessitent d'être formalisés avant de pouvoir être testés. La formalisation permettra de répondre à un certain nombre de questions :

- à partir de quel état de l'application commence-t-on à compter les actions (est-ce que c'est après avoir complété une autre action ?)
- Est-ce que l'expression "moins que" signifie toujours "strictement inférieur à" ?
- Quelles sont les actions qui sont considérées comme des interactions ?

De plus, plusieurs paramètres de la génération de tests apparaissent immédiatement :

- Quelles sont les actions qu'il faut couvrir par un test ? toutes, uniquement celles qui répondent à un critère donné, un ensemble représentatif à déterminer ?
- Quel niveau de détail doit-on atteindre ? En effet, chaque action peut dans la majorité des cas être accessible en moins de N interactions mais ne pas l'être pour un cas particulier. Il est généralement impossible de tester tous les cas (du fait de la taille des domaines des variables considérées) et il est donc nécessaire d'identifier, pour chaque action, des sous-ensembles de cas d'application et de ne tester qu'un représentant pour chacun d'entre eux. Ce niveau de détail constitue l'un des paramètres de la génération de test. En fonction de la valeur qui lui sera attribué, chaque action offerte par

l'application sera décomposée en un certain nombre de sous-ensembles de cas, pour lesquels il faudra considérer que chaque élément est représentatif de l'ensemble. Le résultat de la vérification n'aura de valeur que sous l'hypothèse que les choix qui auront été faits pour fixer les valeurs de ces paramètres sont valides.

Un autre point à traiter provient du caractère général de cette propriété d'ergonomie. En effet, cette propriété d'ergonomie n'est pas explicitée dans l'application mais doit être satisfaite par cette dernière. La génération des tests ne peut donc se faire uniquement depuis l'analyse du texte de la spécification mais doit être guidée par le texte de la propriété.

3.2 Les grandes lignes de notre approche

Apres analyse de la spécification fournie en entrée, CASTING élabore une représentation interne des interactions possibles avec l'application à tester. Cette représentation est le résultat de l'application d'une stratégie de test qui vise à identifier et expliciter des cas particuliers d'applications des opérations puis à fournir pour chacune d'entre elles un scénario d'interaction la mettant en oeuvre. La représentation interne est complétée par l'ajout de variables internes destinées à controller les interactions de manière à pouvoir vérifier des propriétés d'ergonomies prédéfinies. Cette vérification nécessite une « instrumentalisation » de la spécification. Cette instrumentalisation peut consister en une intégration de variables et d'opérations directement dans la spécification mais peut aussi reposer sur un ensemble de variables internes à CASTING. Dans l'état actuel de nos travaux, c'est la première approche qui a été choisie.

La génération de tests, dits ergonomiques, vise soit à produire des contre exemples montrant que la propriété d'ergonomie visée est violée, soit à exhiber un état de l'application qui satisfasse une propriété. Les propriétés d'ergonomie sont en partie intégrées dans la spécification à tester de manière à contrôler la génération des suites de tests. Les objectifs de test sont construits à partir de la négation des propriétés d'ergonomie dans le cas de la génération de contre exemple, ou de contraintes sur un ensemble de variables observables.

La décomposition des opérations de la spécification permet d'identifier les cas particuliers des opérations qui ont conduit à la violation d'une propriété d'ergonomie dans les suites de test produites, ceci afin d'obtenir des informations plus détaillées sur les conditions qui ont conduit à la violation de la propriété. Il est à noter qu'il est possible d'appliquer cette approche sans décomposer les opérations de la spécification et en se limitant à sa traduction dans le langage interne de CASTING. Cependant, ceci reviendrait à sous-utiliser la méthodologie en se privant d'une part importante de ce qu'elle peut produire : les suites de tests instanciées.

Etant donné que ce n'est pas parce que la spécification satisfait les propriétés d'ergonomie que l'implémentation les satisfera aussi, il est nécessaire de réaliser des tests pour s'en assurer ensuite.

La méthodologie que nous proposons consiste donc dans un premier temps à vérifier le respect des propriétés d'ergonomie par la spécification, puis à s'assurer par le test que l'implémentation est conforme à la spécification. Il y a donc deux phases de génération : recherche de contre exemples de propriété d'ergonomie puis production de jeux de tests fonctionnels.

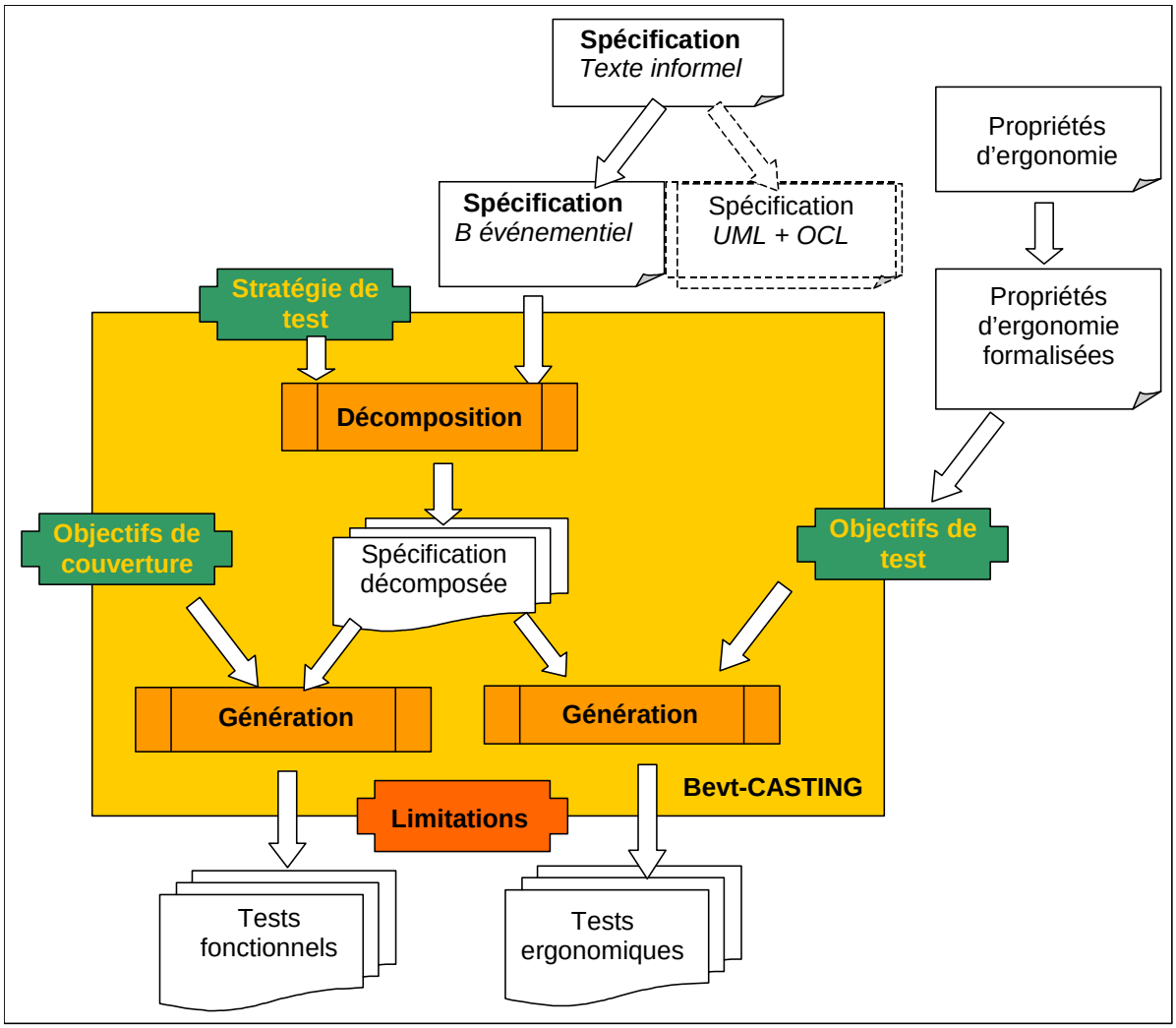

Figure 7 : Présentation générale des approches envisagées

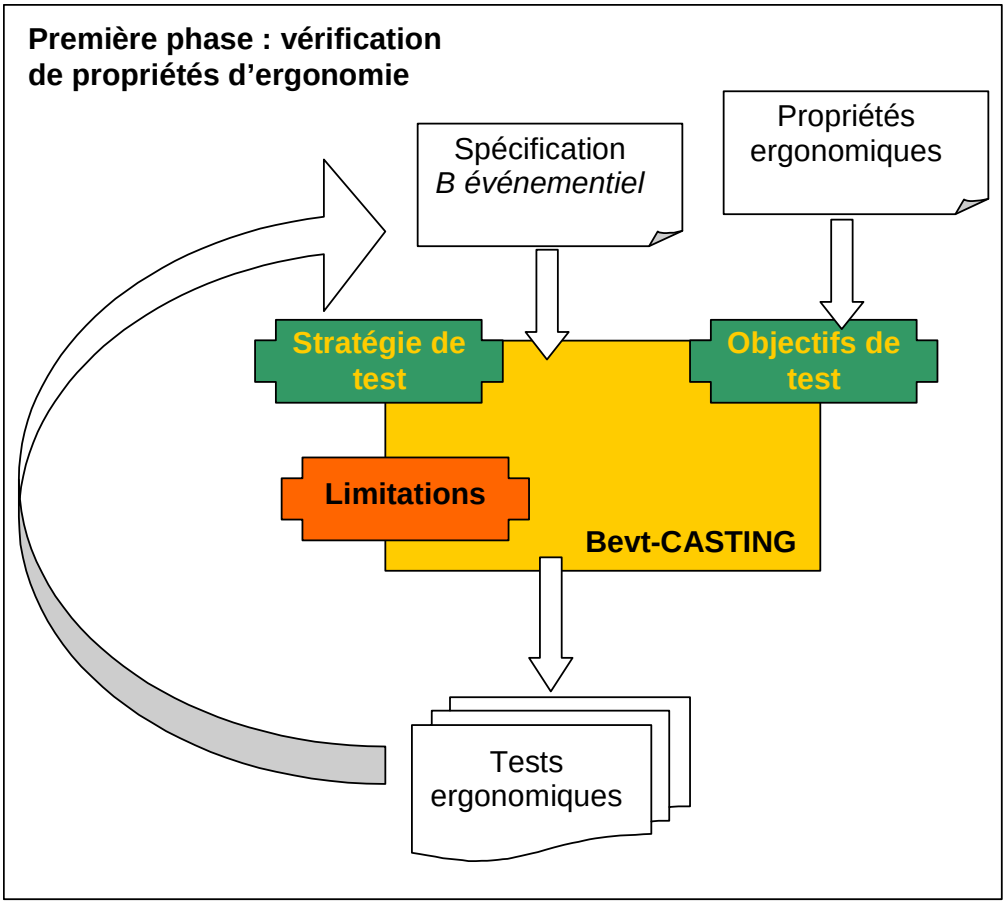

Figure 8 : Première Phase - vérification de propriétés d'ergonomie

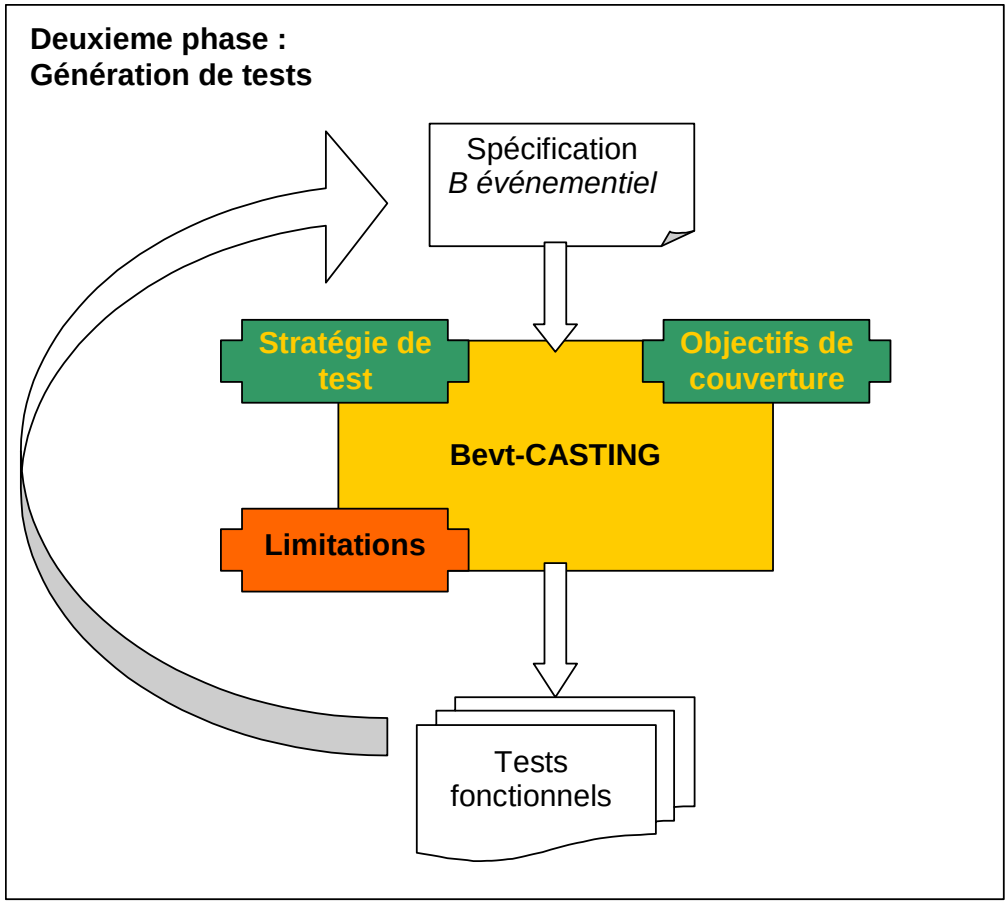

Figure 9 : Deuxième phase - Génération des tests

3.3 Adaptation de CASTING : questions/réponses

Cette section présente l'adaptation de CASTING sous la forme d'une liste de question/réponse.

Que contient la spécification de l'IHM fournie en entrée de CASTING ?

Un modèle des interactions possibles avec l'utilisateur (ordonnancement, type d'interaction, pré-requis). Ce modèle est spécifié en B-événementiel (objet de l'adaptation réalisée dans le cadre du projet VERBATIM).

Comment sont prises en compte les propriétés d'ergonomie dans l'analyse ?

Les propriétés sont prises en compte en les intégrant dans la spécification de l'application IHM à tester. Cette solution permet d'exprimer directement et facilement les propriétés d'ergonomie mais à néanmoins le désavantage de ne pas être indépendante de la spécification. Cette approche nous a permis de nous affranchir d'une partie du probléme (l'expression générale de propriétés d'ergonomies), mais constitue une limitation qui devra être levées dans des travaux futurs, en intégrant ces propriétés dans la stratégie de test de manière à produire systématiquement les cas qui correspondent aux propriétés d'ergonomie visées.

En pratique, les propriétés sont des opérations intégrées dans la spécification (qui sont ensuite traitées comme les opérations de la spécification) exprimés à l'aide de contraintes (pré et post condition sur les variables de la spécification ainsi que sur un ensemble de

variables supplémentaires d'observation qui permettent de spécifier les propriété d'ergonomie).

Comment la stratégie de test est-elle définie ?

Dans la version actuelle de CASTING, la stratégie de test est une sélection de règles de décomposition syntaxique qui sont appliquées aux expressions de la spécification fournie en entrée pour identifier des spécifications de cas de test. Il y a deux catégories de règles de décomposition : celles qui portent sur la structure de la spécification et celles qui portent sur les opérateurs utilisés.

Comment sont définis les taux de couvertures visés ?

Dans la version actuelle de CASTING, les taux de couverture correspondent aux transitions de test couvertes par les données de test obtenues à l'issue de la phase de résolution de contraintes.

L'objectif, dans le cadre du test d'applications multimodales, est de produire des séquences de test en rapport avec les propriétés d'ergonomies. La couverture doit donc être exprimée par rapport aux propriétés d'ergonomie, ce qui revient dans notre approche à couvrir les cas particuliers d'opérations de vérification de respect des propriétés.

Quels sont les résultats de la génération de tests ergonomiques ?

Une fois les propriétés exprimées en tant qu'objectifs de test, CASTING essaye de produire des séquences d'appels d'opérations mettant en évidence ces propriétés. La recherche de solution repose sur des techniques de résolution de contraintes. Du fait des mises en oeuvres de ces techniques (propagation de contraintes pour définir les domaines possibles pour chaque variable, puis choix d'une valeur par variable, jusqu'à obtenir une solution globale au système de contraintes, ou bien avoir essayé toutes les valeurs possibles et conclure à la non satisfaction du système), il est indispensable de limiter le processus de recherche afin d'obtenir une réponse dans un temps donné.

Les résultats possibles appartiennent à l'une des trois catégories suivantes :

- CASTING parvient à produire une séquence de test (un contre exemple) et il est alors avéré que la spécification ne permet pas de satisfaire la propriété d'ergonomie. De plus, la séquence de test met en évidence les conditions qui conduisent à la violation de la propriété (le niveau de détail obtenu dépend du niveau de décomposition impliqué par la stratégie de test choisie).
- CASTING ne parvient pas à produire une séquence de test, avant d'avoir atteint les contraintes de limitation de la recherche de solution. Dans ce cas, la propriété peut raisonnablement être considérée comme étant satisfaite par la spécification.
- CASTING ne parvient pas à produire une séquence de test, en ayant atteint l'une des limites de la recherche de solution. Dans ce cas, il appartient au testeur de décider s'il considère que les limites fixées sont suffisantes pour montrer que la propriété n'est « sans doute » pas violée, ou au contraire qu'il doit modifier un élément (la spécification, l'objectif de test, les limites de la recherche, etc.) de manière à se retrouver dans l'un des deux cas précédents.

3.4 Vérification de propriétés CARE avec Casting

3.4.1 Les propriétés CARE

Les propriétés CARE (Complémentarité, Assignation, Redondance et Equivalence) concernent les modalités qui interviennent dans la réalisation d'une tâche, ou qui sont impliquées dans une trajectoire d'interaction avec l'application testée.

Une trajectoire est la succession d'interactions entre l'application et l'utilisateur nécessaires pour passer d'un état d'entrée à un état résultat. Les états sont définis à partir d'éléments observables de l'application sous test.

Les définitions des propriétés CARE sont données dans [INTERACT95] et sont rappelées ici :

Complémentarité : deux modalités sont dites complémentaires sur une succession d'interactions avec l'application testée si elles sont nécessaires pour l'accomplissement de la tâche.

Assignation : une modalité est assignée à une trajectoire si elle est la seule modalité utilisée pour la réaliser. Il y a deux types d'assignation : l'assignation qui est le fait de l'application (l'application ne propose pas d'autre modalité pour réaliser une action) et celle qui est le fait de l'utilisateur qui, alors que l'application lui propose un choix de modalités pour réaliser une action, n'en utilise qu'une seule. Cette seconde version de la propriété concerne donc l'analyse de scénarios d'interaction enregistrés et n'est donc pas spécifique à l'application. Elle sort du périmètre de notre approche.

Redondance : deux modalités sont dites redondantes lorsqu'elles permettent indépendamment l'une de l'autre d'obtenir le même résultat (i.e. d'atteindre le même état observable) dans le même espace temporel. On distingue deux types de redondance : la redondance séquentielle et la redondance parallèle.

Equivalence : un ensemble de modalités (au moins deux) est dit équivalent lorsque l'utilisation d'une seule d'entre elles est nécessaire et suffisante pour atteindre le même résultat depuis un état d'entrée donné.

3.4.2 Vérification à l'aide de CASTING

La technique proposée repose sur l'intégration d'opérations dédiées à la vérification de propriétés de type CARE directement dans la spécification fournie en entrée de CASTING. Ces opérations sont définies à l'aide de contraintes sur les variables qui constituent l'état observable de l'application. Elles sont prises en compte lors du processus de génération de test au même titre que les opérations d'interaction de l'application. Leur vérification consiste à demander à CASTING de produire un scénario partant d'un état défini à un autre état qui satisfasse la propriété visée.

Contrôle de la redondance de modalités

Cette vérification est réalisée en utilisant les variables de contrôle de l'application destinées à faire des vérifications lors de la génération des tests. Par exemple, la définition d'une opération qui comporte une précondition sur les modalités utilisées dans une trajectoire d'interaction permet de s'assurer que deux modalités sont redondantes. Pour cela, il faut définir une variable d'observation (`modalites_utilisées`) qui est mise à jour lors de l'utilisation des modalités, ainsi qu'une opération par modalité qui vise à vérifier que seule la modalité en question a été utilisée. Si la génération de test permet de produire deux scénarios qui comporte ces opérations, alors la propriété de redondance de ces opérations pour un chemin particulier est effectuée.

Contrôle de l'assignation d'une modalité

L'assignation d'une modalité pour une tâche qui est due à l'utilisateur n'est pas couverte par la génération de test. Par contre, le fait qu'une modalité soit assignée par l'application à une tâche donnée peut être vérifié.

Pour ce faire, il faut définir une opération qui vise à produire un contre exemple d'enchainement d'opérations utilisant au moins une autre modalité que la modalité assignée pour parcourir la trajectoire d'interaction considérée.

Contrôle de l'équivalence de deux modalités

Le contrôle de la propriété d'équivalence sur deux modalités est obtenue en demandant à CASTING de produire autant de suites d'opérations que de modalités concernées sur un chemin d'interaction donné.

Contrôle de la complémentarité d'un ensemble de modalités pour une tâche donnée

Le contrôle de la propriété de complémentarité sur un ensemble de modalités est obtenue en demandant à l'outil de produire des contre-exemples mettant en évidence que le chemin d'interaction concerné par la propriété peut être réalisé en n'utilisant pas toutes les modalités concernées.

3.5 Application au cas d'étude « Pages jaunes - Imappy »

3.5.1 Introduction

Le cas d'étude retenu dans le cadre du projet est une application qui permet à un utilisateur d'interroger le service « pages jaunes » en interagissant avec une IHM acceptant plusieurs modalités pour soumettre sa requête de recherche de plan d'adresses. L'utilisateur obtient en réponse une carte sur laquelle il peut effectuer quelques opérations de navigation (déplacement du centre de la carte, zoom avant ou arrière).

Nous présentons en première partie de cette section la spécification de l'étude de cas sans « instrumentalisation ». Les ajouts nécessaires pour utiliser l'approche seront présentés dans un second temps.

3.5.2 La spécification

La spécification utilisée ici est exprimée en B événementiel. Néanmoins, le langage de spécification n'est utilisé que pour décrire les interfaces du cas d'étude, et aucune preuve ni obligation de preuve n'a été démontrée.

De plus, l'objectif n'est pas de spécifier en détail le comportement de l'application mais bien de se focaliser sur les interactions entre les modalités offertes, c'est pourquoi cette spécification est une spécification des modalités, plus qu'une spécification de l'application. Néanmoins, toutes les interactions avec l'application sont prises en compte.

```
Classe Imappy
```

La clause SETS permet de définir les ensembles énumérés qui seront utilisés dans le reste de la spécification. La définition en extension des ensembles est indispensable dans notre approche car la phase de génération de données de test fonctionne sur des ensembles finis.

```
SETS
string_name : {Lionel, Yamine, Laurence};      /* Ensemble des valeurs
possibles pour le nom */
string_address : {Rennes, Poitier, Grenoble};  /* Ensemble des valeurs
possibles pour l'adresse */

/* HCI : ensemble de toutes les modalités/commandes possibles */
HCI : {
   NONE,
   SAY_NAME,
   SAY_ADDRESS,
   SAY_INPUT,
   SAY_SEARCH,
   ZOOM_IN,
   ZOOM_OUT,
   HERE,
   CLICK_ON_NAME,
   CLICK_ON_ADDRESS,
   CLICK_ON_SEARCH,
   CLICK_ON_MAP,
   INPUT_TEXT,
   ENTER_KEY,
   LEFT_ARROW,
   RIGHT_ARROW,
   UP_ARROW,
   DOWN_ARROW };

/* input_statut : ensemble des valeurs possibles pour le statut  */
input_statut : {unset, set};
```

La clause ATTRIBUTS contient les définitions (type) des variables d'état de l'application.

```
ATTRIBUTS
/* Attributs de l'IHM */
Query_sending  : boolean;     /* Statut d'envoi de la requête */
Query_ready    : boolean;     /* Statut de préparation de la requête */
map            : boolean;     /* Statut d'affichage de la carte, résultat de
la requête */
lock_name      : boolean;     /* Etat de saisi du nom */
name           : string_name;       /* Variable de type chaine de caractères
pour le nom */
name_statut    : input_statut;      /* Activation en cours ou pas de la
commande de saisie du nom */
lock_address   : boolean;     /* Activation en cours ou pas de la commande
de saisie de l'adresse */
address        : string_address; /* Variable de type chaine de caractères
pour l'adresse */
address_statut : input_statut;  /* Etat de saisi de l'adresse */
EV1            : integer;     /* Entier pour implémenter les ordonnancement
de tâches */
interaction   : HCI; /* modalité courante, i.e de la dernière tâche */
```

La clause « Initialisation » permet de fixer les valeurs initiales de tous les attributs.

```
INITIALISATION
  name := Lionel ||          /* valeur par défaut */
  address := Rennes ||       /* valeur par défaut */

  Query_sending := FALSE ||  /* La requête n'a pas été envoyée */
  Query_ready := TRUE    ||  /* la requête n'est pas prête à partir */
  map := FALSE           ||  /* TRUE quand la carte est affichée */

  EV1 := 3 ||                /* la variable est à 3 lorsqu'aucune
information n'est saisie */

  lock_name := FALSE ||           /* TRUE quand la commande de saisie du
nom a été activée */

  lock_address := FALSE ||   /* TRUE quand la commande de saisie de
l'adresse a été activée */
  name_statut := unset  ||         /* la valeur set indique que le nom a
été saisi */
  address_statut := unset ||     /* la valeur set indique que l'adresse a
été saisie */
```

La clause EVENTS contient les opérations de la spécification. Chaque opération d'interaction a au moins deux parametres qui sont : le mode d'interaction (p_interact dont les valeurs sont dans l'ensemble HCI préalablement défini) et la modalité utilisée (p_mode dont les valeurs sont dans l'ensemble HCI_modalites préalablement défini). L'opération Name_Address n'est pas une opération d'interaction mais constitue le point de départ de la saisie de nom et de l'adresse.

```
EVENTS

/* Démarrage d'une saisie */
Name_Address =
  SELECT
    EV1@pre = 3
  THEN
    EV1 := EV1@pre - 1
  END;
```

```
/* Selection de la zone de saisie du nom */
Input_Name_field(In p_interact : HCI, In p_mode : HCI_modalites) =
  SELECT
    lock_name@pre = FALSE
    and EV1@pre = 2 /* Phase "saisie" */
    and (
      (p_interact = CLICK_ON_NAME and p_mode = souris)
      or
      (p_interact = SAY_NAME and p_mode = microphone)
       )
  THEN
    lock_name := TRUE and
    name_statut := unset and
    /* verification */
    interaction := p_interact
  END ;
```

```
/* Saisie du nom */
Input_Name_value(In p_new_name : string_name, In p_interact : HCI, In
p_mode : HCI_modalites) =
   SELECT
      lock_name@pre = TRUE
      and EV1@pre = 2
      and ((p_interact = SAY_INPUT  and p_mode = microphone)
            or
           (p_interact = INPUT_TEXT and p_mode = clavier))
   THEN
      name := p_new_name and
      lock_name := FALSE and
      name_statut := set and
      interaction := p_interact
   END ;
```

```
/* Selection de la zone de saisie de l'adresse */
Input_Address_field(In p_interact : HCI, In p_mode : HCI_modalites) =
   SELECT
      lock_address@pre = FALSE and
      EV1@pre = 2
      and ((p_interact = CLICK_ON_ADDRESS  and p_mode = souris)
            or
           (p_interact = SAY_ADDRESS and p_mode = microphone))
   THEN
      lock_address := TRUE and
      address_statut := unset and
      interaction := p_interact
   END ;
```

```
/* Saisie de l'adresse */
Input_Address_value(In p_interact : HCI, In p_mode : HCI_modalites, In
p_new_address : string_address) =
   SELECT
      lock_address@pre = TRUE and
      EV1@pre = 2 and
      ((p_interact = SAY_INPUT and p_mode = microphone)
        or
        (p_interact = INPUT_TEXT and p_mode = clavier))
      and
      (interaction@pre = CLICK_ON_ADDRESS or interaction@pre = SAY_ADDRESS)
   THEN
      address := p_new_address and
      address_statut := set and
      lock_address :=  FALSE and
      interaction := p_interact
   END ;
```

```
/* Commande de zoom sur la carte */
Map_zoom(In p_interact : HCI, In p_mode : HCI_modalites) =
   SELECT
      map@pre = TRUE and
      (p_interact = ZOOM_OUT or p_interact = ZOOM_IN) and p_mode = souris
   THEN
      interaction := p_interact
   END ;
```

```
/* Commande de déplacement sur la carte */
Map_move(In p_interact : HCI, In p_mode : HCI_modalites) =
  SELECT
    map@pre = TRUE and
    (
      ((p_interact= LEFT_ARROW or p_interact = RIGHT_ARROW or p_interact =
UP_ARROW or p_interact = DOWN_ARROW)
        and p_mode = clavier)
     or
     (p_interact = CLICK_ON_MAP and p_mode = souris))
  THEN
    interaction := p_interact
  END ;
```

```
/* Lancement de la commande de recherche */
Search(In p_interact : HCI, In p_mode : HCI_modalites) =
  SELECT
    EV1@pre = 2 and
    ((p_interact = CLICK_ON_SEARCH and p_mode  = souris)
     or
      ((p_interact = SAY_SEARCH and p_mode  = microphone)
     or
       (p_interact = ENTER_KEY and p_mode  = clavier)))
    and name_status@pre = set and address_statut@pre = set
  THEN
    EV1 := 1 and
    interaction := p_interact
  END ;
```

```
/* Traitement de la commande : envoi */
Query =
  SELECT
    EV1@pre = 1
  THEN
    Query_sending := TRUE and EV1 := 0
  END;
```

```
/* Traitement de la commande : résultat */
Result_Query =
  SELECT
    Query_sending@pre = TRUE
  THEN
    Query_sending := FALSE and
    Query_ready := TRUE and
    EV1 := 3 and
    map := TRUE
  END;
```

3.5.3 Instrumentalisation de la spécification pour le test

Afin de réaliser des vérifications de propriétés la méthode que nous proposons consiste à instrumentaliser la spécification pour y intégrer des éléments observables ainsi que des variables de contrôle. Ces éléments sont utilisés dans les opérations de vérification qui sont elles aussi ajoutées à la spécification initiale.

Cette instrumentalisation est réalisée manuellement dans l'état actuel de la méthode, mais devrait à terme pouvoir être réalisée de manière automatique.

Les opérations de vérification se décomposent en deux familles : les opérations dites « positives », qui concernent les séquences de test qui doivent être produites, et les opérations dites « négatives » qui sont des contre-exemples de la propriété à vérifier. Ces

dernières sont aussi définies par des contraintes sur les variables observables de l'application ainsi que sur celles ajoutées spécifiquement pour le test.

Plusieurs attributs de vérification sont ajoutés à la spécification. Ils permettent d'ajouter des observations sur l'évolution de l'état de l'application et plus particulièrement ce qui concerne les modalités utilisées ainsi que les tâches réalisées, lorsque la modélisation de l'application ne les intègre pas déjà. Ces attributs sont ensuite utilisés dans le cadre des opérations de vérification de propriétés CARE.

```
/* Attributs de vérification */
cov_modalites : POW(HCI_modalites); /* ensembles des modalités utilisées
sur le chemin */
cov_commandes : POW(commandes); /* ensembles des commandes utilisées sur le
chemin */
```

La clause d'initialisation est donc modifiée afin d'affecter des valeurs initiales à ces deux nouveaux attributs :

```
   cov_commandes := {} ||
   cov_modalites := {} ||           /* valeur par défaut */
```

Chaque opération de la clause EVENT est complétée de manière à faire évoluer ces attributs. Par exemple, l'opération Input_Name_field est complétée de la manière suivante (l'ajout est en **gras**) :

```
/* Selection de la zone de saisie du nom */
Input_Name_field(In p_interact : HCI, In p_mode : HCI_modalites) =
   SELECT
      lock_name@pre = FALSE
      and EV1@pre = 2 /* Phase "saisie" */
      and p_interact = SAY_NAME
      and p_mode = microphone
   THEN
      lock_name := TRUE and
      name_statut := unset and
      /* verification */
      interaction := p_interact and
      cov_commandes := cov_commandes@pre \/ {Input_Name_field} and
      cov_modalites := cov_modalites@pre \/ {p_mode}
   END ;
```

3.5.4 Vérification de la complémentarité

La vérification de la complémentarité des modalités « voix » et « clavier » pour la saisie du nom et de l'adresse de la recherche passe par la définition des 3 opérations suivantes :

- `verif_complementarite_1_ok` : cette opération vérifie que l'application est capable d'atteindre un état dans lequel le nom et l'adresse ont pu etre saisis et que les deux modalités ont été utilisées pour atteindre cet état.
- `verif_complementarite_1_ko_clavier` : cette opération vérifie que l'application est capable d'atteindre un état dans lequel le nom et l'adresse ont pu etre saisie sans que la modalité « microphone » est été utilisée.
- `verif_complementarite_1_ko_micro` : cette opération vérifie que l'application est capable d'atteindre un état dans lequel le nom et l'adresse ont pu etre saisie sans que la modalité « clavier » est été utilisée.

Les contraintes sur les variables `cov_modalités` et `cov_commandes` permettent de s'assurer que seules les opération concernées ont été activées. En effet, sans cette contrainte une autre opération mettant en oeuvre une modalité supplémentaire aurait pu être activée.

```
/* Complémentarité de la voix et du clavier pour la saisie */
verif_complementarite_1_ok  =
  BEGIN
    name_statut@pre = set and
/* la valeur set indique que le nom a été saisi */
    address_statut@pre = set and
/* la valeur set indique que l'adresse a été saisie */
    cov_modalites@pre = {microphone, clavier} and
    cov_commandes@pre = {Input_Name_field, Input_Name_value,
Input_Address_field, Input_Address_value}
  END;
```

```
verif_complementarite_1_ko_clavier =
  BEGIN
    name_statut@pre = set and
/* la valeur set indique que le nom a été saisi */
    address_statut@pre = set and
/* la valeur set indique que l'adresse a été saisie */
    cov_modalites@pre = {clavier} and    /*  */
    cov_commandes@pre = {Input_Name_field, Input_Name_value,
Input_Address_field, Input_Address_value}
  END;
```

```
verif_complementarite_1_ko_micro =
  BEGIN
    name_statut@pre = set and      /* la valeur set indique que le nom a
été saisi */
    address_statut@pre = set and    /* la valeur set indique que l'adresse
a été saisie */
    cov_modalites@pre = {microphone} and /*  */
    cov_commandes@pre = {Input_Name_field, Input_Name_value,
Input_Address_field, Input_Address_value}
  END;
```

3.5.5 Objectifs de tests et vérifications de propriétés

La méthode (CASTING) initiale de génération de test est adaptée en guidant la génération de test de manière à ce que les opérations de vérifications soient atteintes.

Dans le cas des opérations de vérifications dites « négatives », le fait que le générateur de test réussisse à produire un scénario de test qui l'inclut permet de vérifier qu'une propriété n'est pas vérifiée.

Dans le cadre de l'exemple choisi pour illustrer cette section, si lors de la génération, un scénario de test mettant en oeuvre l'une des deux opérations d'échec est produit, alors il sera possible de conclure que la propriété de complémentarité n'est pas respectée pour la saisie du nom et de l'adresse. Par contre, pour que la propriété soit considérée comme satisfaite, il faut qu'un scénario de test inclut la première opération de vérification et qu'il ne soit pas possible de générer des scénarios qui permette d'atteindre l'un des deux autres.

3.5.6 Production des tests

Après avoir généré les transitions de test pour chaque opération de la spécification, l'outil produit les séquences de tests qui permettent de couvrir chacune d'entre elles, dans la limites des contraintes qui lui sont imposées (quelles transitions, temps de recherche bornée, longueur des suites de test, etc.) toutes les transitions de tests. Nous ne donnons ici que le scénario qui permet de couvrir l'opération de vérification **verif_complementarite_1_ok** (les autres scénarios sont fournis en annexe) :

```
[Test_4:initialisation();
Name_Address();
Input_Name_field(SAY_NAME,microphone);
Input_Name_value(Lionel,INPUT_TEXT,clavier);
Input_Address_field(SAY_ADDRESS,microphone);
Input_Address_value(INPUT_TEXT,clavier,Rennes);
verif_complementarite_1_ok()
]
```

Etape n°0	**initialisation (*tdt0.1*)**
Paramètres	-
Etat	Query_sending = FALSE Query_ready = TRUE map = FALSE lock_name = FALSE name = Lionel name_statut = unset lock_address = FALSE address = Rennes address_statut = unset EV1 = 3 interaction = NONE cov_modalites= {} cov_commandes= {}

Etape n°1	**Name_Address (*tdt1.1*)**
Paramètres	-
Etat	Query_sending = FALSE Query_ready = TRUE map = FALSE lock_name = FALSE name = Lionel name_statut = unset lock_address = FALSE address = Rennes address_statut = unset EV1 = 2 interaction = NONE cov_modalites= {} cov_commandes= {}

Etape n°2	**Input_Name_field (*tdt2.1*)**
Paramètres	p_interact = SAY_NAME p_mode = microphone
Etat	Query_sending = FALSE Query_ready = TRUE map = FALSE lock_name = TRUE name = Lionel name_statut = unset lock address = FALSE

	address = Rennes address_statut = unset EV1 = 2 interaction = SAY_NAME cov_modalites= {microphone} cov_commandes= {Input_Name_field}

Etape n°3	**Input_Name_value (*tdt3.1*)**
Paramètres	p_new_name = Lionel p_interact = INPUT_TEXT p_mode = clavier
Etat	Query_sending = FALSE Query_ready = TRUE map = FALSE lock_name = FALSE name = Lionel name_statut = set lock_address = FALSE address = Rennes address_statut = unset EV1 = 2 interaction = INPUT_TEXT cov_modalites= {clavier, microphone} cov_commandes= {Input_Name_field, Input_Name_value}

Etape n°4	**Input_Address_field (*tdt4.1*)**
Paramètres	p_interact = SAY_ADDRESS p_mode = microphone
Etat	Query_sending = FALSE Query_ready = TRUE map = FALSE lock_name = FALSE name = Lionel name_statut = set lock_address = TRUE address = Rennes address_statut = unset EV1 = 2 interaction = SAY_ADDRESS cov_modalites= {clavier, microphone} cov_commandes= {Input_Name_field, Input_Name_value, Input_Address_field}

Etape n°5	**Input_Address_value (*tdt5.1*)**
Paramètres	p_interact = INPUT_TEXT p_mode = clavier p_new_address = Rennes
Etat	Query_sending = FALSE

	Query_ready = TRUE map = FALSE lock_name = FALSE name = Lionel name_statut = set lock_address = FALSE address = Rennes address_statut = set EV1 = 2 interaction = INPUT_TEXT cov_modalites= {clavier, microphone} cov_commandes= {Input_Name_field, Input_Name_value, Input_Address_field, Input_Address_value}

Etape n°6	**verif_complementarite_1_ok (*tdt11.1*)**
Paramètres	-
Etat	Query_sending = FALSE Query_ready = TRUE map = FALSE lock_name = FALSE name = Lionel name_statut = set lock_address = FALSE address = Rennes address_statut = set EV1 = 2 interaction = INPUT_TEXT cov_modalites= {clavier, microphone} cov_commandes= {Input_Name_field, Input_Name_value, Input_Address_field, Input_Address_value}

Quelque soit les contraintes que l'on relache ensuite, l'outil ne parvient pas à couvrir les deux autres opérations de vérification. Nous pouvons donc estimer que la propriété de complémentarité de la voix et du clavier pour la saisie des informations de la requête est bien satisfaite par cette spécification.

Afin d'illustrer le cas de la production d'un contre-exemple, nous modifions la spécification en ajoutant une modalité supplémentaire sur la saisie des valeurs (la voix) ce qui a pour effet de violer la propriété de complémentarité considérée dans cet exemple.

La spécification est modifiée de la manière suivante :

```
/* Saisie du nom */
Input_Name_value(In p_new_name : string_name, In p_interact : HCI, In
p_mode : HCI_modalites) =
  SELECT
     lock_name@pre = TRUE
     and EV1@pre = 2
     and ((p_interact = SAY_INPUT  and p_mode = microphone) /* Modification
*/
           or
          (p_interact = INPUT_TEXT and p_mode = clavier))
  THEN
     name := p_new_name and
     lock_name := FALSE and
     name_statut := set and
     /* verification */
     interaction   := p_interact and
     cov_commandes := cov_commandes@pre \/ {Input_Name_value} and
     cov_modalites := cov_modalites@pre \/ {p_mode}
  END ;

...

/* Saisie de l'adresse */
Input_Address_value(In p_interact : HCI, In p_mode : HCI_modalites, In
p_new_address : string_address) =
  SELECT
     lock_address@pre = TRUE and
     EV1@pre = 2 and
     ((p_interact = SAY_INPUT and p_mode = microphone)
       or
       (p_interact = INPUT_TEXT and p_mode = clavier))
  THEN
     address := p_new_address and
     address_statut := set and
     lock_address :=  FALSE and
     /* verification */
     interaction := p_interact and
     cov_commandes := cov_commandes@pre \/ {Input_Address_value} and
     cov_modalites := cov_modalites@pre \/ {p_mode}
  END ;
```

L'outil permet de focaliser la recherche de scénarios de test en fonction d'une sélection de transitions de test à couvrir. Dans le cas présent nous sélection les opérations de vérifications `verif_complementarite_1_ko_clavier` et `verif_complementarite_1_ko_micro`.

L'opération est couverte et le scénario de test qui permet de l'appliquer constitue un contre exemple attestant de la violation de la propriété de complémentarité :

```
[Test_1:initialisation();
Name_Address();
Input_Name_field(SAY_NAME,microphone);
Input_Name_value(Lionel,SAY_INPUT,microphone);
Input_Address_field(SAY_ADDRESS,microphone);
Input_Address_value(SAY_INPUT,microphone,Rennes);
verif_complementarite_1_ko_micro()
]
```

Le détail du scénario est le suivant :

Etape n°0	**initialisation (*tdt0.1*)**
Paramètres	-
Etat	Query_sending = FALSE Query_ready = TRUE

	map = FALSE lock_name = FALSE name = Lionel name_statut = unset lock_address = FALSE address = Rennes address_statut = unset EV1 = 3 interaction = NONE cov_modalites= {} cov_commandes= {}

Etape n°1	**Name_Address (*tdt1.1*)**
Paramètres	-
Etat	Query_sending = FALSE Query_ready = TRUE map = FALSE lock_name = FALSE name = Lionel name_statut = unset lock_address = FALSE address = Rennes address_statut = unset EV1 = 2 interaction = NONE cov_modalites= {} cov_commandes= {}

Etape n°2	**Input_Name_field (*tdt2.1*)**
Paramètres	p_interact = SAY_NAME p_mode = microphone
Etat	Query_sending = FALSE Query_ready = TRUE map = FALSE lock_name = TRUE name = Lionel name_statut = unset lock_address = FALSE address = Rennes address_statut = unset EV1 = 2 interaction = SAY_NAME cov_modalites= {microphone} cov_commandes= {Input_Name_field}

Etape n°3	**Input_Name_value (*tdt3.1*)**
Paramètres	p_new_name = Lionel p_interact = SAY_INPUT p_mode = microphone

Etat	Query_sending = FALSE Query_ready = TRUE map = FALSE lock_name = FALSE name = Lionel name_statut = set lock_address = FALSE address = Rennes address_statut = unset EV1 = 2 interaction = SAY_INPUT cov_modalites= {microphone} cov_commandes= {Input_Name_field, Input_Name_value}

Etape n°4	**Input_Address_field (*tdt4.1*)**
Paramètres	p_interact = SAY_ADDRESS p_mode = microphone
Etat	Query_sending = FALSE Query_ready = TRUE map = FALSE lock_name = FALSE name = Lionel name_statut = set lock_address = TRUE address = Rennes address_statut = unset EV1 = 2 interaction = SAY_ADDRESS cov_modalites= {microphone} cov_commandes= {Input_Name_field, Input_Name_value, Input_Address_field}

Etape n°5	**Input_Address_value (*tdt5.1*)**
Paramètres	p_interact = SAY_INPUT p_mode = microphone p_new_address = Rennes
Etat	Query_sending = FALSE Query_ready = TRUE map = FALSE lock_name = FALSE name = Lionel name_statut = set lock_address = FALSE address = Rennes address_statut = set EV1 = 2 interaction = SAY_INPUT cov_modalites= {microphone} cov_commandes= {Input_Name_field, Input_Name_value, Input_Address_field, Input_Address_value}

Etape n°6	verif_complementarite_1_ko_micro (*tdt13.1*)
Paramètres	-
Etat	Query_sending = FALSE Query_ready = TRUE map = FALSE lock_name = FALSE name = Lionel name_statut = set lock_address = FALSE address = Rennes address_statut = set EV1 = 2 interaction = SAY_INPUT cov_modalites= {microphone} cov_commandes= {Input_Name_field, Input_Name_value, Input_Address_field, Input_Address_value}

3.6 Conclusion

3.6.1 Résultats

Les résultats obtenus à l'issue de ce projet concernent à la fois la méthodologie de génération de tests et le développement de l'outil CASTING.

D'un point de vue méthodologique, nous avons exploré la production de contre-exemples dans le but de vérifier des propriétés en nous appuyant sur une instrumentalisation de la spécification. Jusqu'à présent, la méthode que nous proposions reposait uniquement sur la production de cas de test « positifs », autrement dit de cas particulier de tests destinés à vérifier qu'une implémentation satisfait bien à sa spécification.

Nous avions (avant le projet VERBATIM) déjà abordé le problème de la validation de la spécification par l'exploitation des tests produits en proposant de les examiner et de les considérer comme des scénarios possibles d'exécution de l'application spécifiée. Avec la prise en compte des propriétés et l'ajout de variables d'observation dans la spécification nous abordons d'une manière différente et sensiblement plus efficace la vérification de la spécification par rapport à un ensembles d'exigences généralement exprimées en dehors de la spécification (ici, les propriétés d'ergonomies).

Cette modification de la méthode a nécessité des adaptations de l'outil, afin de pouvoir prendre en compte le langage B événementiel utilisé pour spécificier les IHM multimodales ainsi que les propriétés d'ergonomies à vérifier. La prise en compte des ensembles et des opérations sur les ensembles constitue la majeure amélioration de l'outil.

Des modifications ont aussi été apportées aux formats de sortie de l'outil de manière à permettre l'interfaçage avec l'outil développé par le LISI pour animer des spécifications à l'aide de scénarios produits par CASTING. Cette interface est réalisée en produisant les suites de tests dans le langage EXPRESS, qui constitue le point d'entrée de l'outil B2EXPRESS.

3.6.2 Travaux futurs

Les opérations de vérification doivent être définies directement dans la spécification et ce pour chaque propriété à vérifier. Cette phase d'instrumentalisation de la spécification est actuellement une des principales limitations de notre approche. La définition et l'utilisation d'un langage intermédiaire de propriétés d'ergonomie devrait permettre de palier ce problème en permettant de déclarer des propriétés sur des opérations, et pour des modalités précises. Le but étant de produire automatiquement les opérations de vérification, de même que les attributs d'observations et leurs contraintes associées.

4. Annexes

4.1 Spécification complète de l'exemple IMAPPY

```
CLASSE Imappy

/*
    Démonstration de la vérification de complémentarité : cas ok.

    - micro et clavier complémentaires : Saisie du nom et de l'adresse
uniquement en utilisant le micro pour selectionner la zone à saisir et le
clavier pour entrer les valeurs.

*/

SETS
string_name : {Lionel, Yamine, Laurence};      /* Ensemble des valeurs
possibles pour le nom */
string_address : {Rennes, Poitier, Grenoble};  /* Ensemble des valeurs
possibles pour l'adresse */

/* HCI : ensemble de toutes les modalités/commandes possibles */
HCI : {
  NONE,
  SAY_NAME,
  SAY_ADDRESS,
  SAY_INPUT,
  SAY_SEARCH,
  ZOOM_IN,
  ZOOM_OUT,
  HERE,
  CLICK_ON_NAME,
  CLICK_ON_ADDRESS,
  CLICK_ON_SEARCH,
  CLICK_ON_MAP,
  INPUT_TEXT,
  ENTER_KEY,
  LEFT_ARROW,
  RIGHT_ARROW,
  UP_ARROW,
  DOWN_ARROW };

HCI_modalites : {clavier, microphone, souris};

/* input_statut : ensemble des valeurs possibles pour le statut  */
input_statut : {unset, set};

commandes : {Input_Name_field, Input_Name_value, Input_Address_field,
Input_Address_value, Search, Map_move, Map_zoom};

ATTRIBUTS
/* Attributs de l'IHM */
Query_sending  : boolean;     /* Statut d'envoi de la requête */
Query_ready    : boolean;     /* Statut de préparation de la requête */
map            : boolean;                /* Statut d'affichage de la carte,
résultat de la requête */
lock_name      : boolean;          /* Etat de saisi du nom */
name           : string_name;            /* Variable de type chaine de
caractères pour le nom */
name_statut    : input_statut;     /* Activation en cours ou pas de la
commande de saisie du nom */
lock_address   : boolean;     /* Activation en cours ou pas de la commande
de saisie de l'adresse */
address        : string_address;    /* Variable de type chaine de caractères
pour l'adresse */
address_statut : input_statut;     /* Etat de saisi de l'adresse */
```

```
EV1            : integer;                /* Entier pour implémenter les
ordonnancement de tâches */

/* Attributs de vérification */
interaction   : HCI;                 /* modalité courante, i.e de la
dernière tâche */
cov_modalites : POW(HCI_modalites); /* ensembles des modalités utilisées
sur le chemin */
cov_commandes : POW(commandes);     /* ensembles des commandes utilisées
sur le chemin */


INITIALISATION
  name := Lionel ||          /* valeur par défaut */
  address := Rennes ||       /* valeur par défaut */

  Query_sending := FALSE ||  /* La requête n'a pas été envoyée */
  Query_ready := TRUE    ||  /* la requête n'est pas prête à partir */
  map := FALSE           ||  /* TRUE quand la carte est affichée */

  EV1 := 3 ||                /* la variable est à 3 lorsqu'aucune
information n'est saisie */

  lock_name := FALSE ||            /* TRUE quand la commande de saisie du
nom a été activée */

  lock_address := FALSE ||   /* TRUE quand la commande de saisie de
l'adresse a été activée */
  name_statut := unset  ||         /* la valeur set indique que le nom a
été saisi */
  address_statut := unset ||     /* la valeur set indique que l'adresse a
été saisie */

  cov_commandes := {} ||
  cov_modalites := {} ||           /* valeur par défaut */
  interaction := NONE


EVENTS

/* Démarrage d'une saisie */
Name_Address =
  SELECT
    EV1@pre = 3
  THEN
    EV1 := EV1@pre - 1
  END;

/* Selection de la zone de saisie du nom */
Input_Name_field(In p_interact : HCI, In p_mode : HCI_modalites) =
  SELECT
    lock_name@pre = FALSE
    and EV1@pre = 2 /* Phase "saisie" */
    and p_interact = SAY_NAME
    and p_mode = microphone
  THEN
    lock_name := TRUE and
    name_statut := unset and
    /* verification */
    interaction := p_interact and
    cov_commandes := cov_commandes@pre \/ {Input_Name_field} and
    cov_modalites := cov_modalites@pre \/ {p_mode}
  END ;

/* Saisie du nom */
Input_Name_value(In p_new_name : string_name, In p_interact : HCI, In
p_mode : HCI_modalites) =
  SELECT
    lock_name@pre = TRUE
    and EV1@pre = 2
```

```
     and p_interact = INPUT_TEXT
     and p_mode = clavier
  THEN
     name := p_new_name and
     lock_name := FALSE and
     name_statut := set and
     /* verification */
     interaction   := p_interact and
     cov_commandes := cov_commandes@pre \/ {Input_Name_value} and
     cov_modalites := cov_modalites@pre \/ {p_mode}
  END ;


/* Selection de la zone de saisie de l'adresse */
Input_Address_field(In p_interact : HCI, In p_mode : HCI_modalites) =
  SELECT
     lock_address@pre = FALSE and
     EV1@pre = 2
     and p_interact = SAY_ADDRESS and p_mode = microphone
  THEN
     lock_address := TRUE and
     address_statut := unset and
     /* verification */
     interaction := p_interact and
     cov_commandes := cov_commandes@pre \/ {Input_Address_field} and
     cov_modalites := cov_modalites@pre \/ {p_mode}
  END ;

/* Saisie de l'adresse */
Input_Address_value(In p_interact : HCI, In p_mode : HCI_modalites, In
p_new_address : string_address) =
  SELECT
     lock_address@pre = TRUE and
     EV1@pre = 2 and
     p_interact = INPUT_TEXT and p_mode = clavier
  THEN
     address := p_new_address and
     address_statut := set and
     lock_address :=  FALSE and
     /* verification */
     interaction := p_interact and
     cov_commandes := cov_commandes@pre \/ {Input_Address_value} and
     cov_modalites := cov_modalites@pre \/ {p_mode}
  END ;


/* Commande de zoom sur la carte */
Map_zoom(In p_interact : HCI, In p_mode : HCI_modalites) =
  SELECT
     map@pre = TRUE and
     (p_interact = ZOOM_OUT or p_interact = ZOOM_IN) and p_mode = souris
  THEN
     interaction := p_interact and
     cov_commandes := cov_commandes@pre \/ {Map_zoom} and
     cov_modalites := cov_modalites@pre \/ {p_mode}
  END ;

/* Commande de déplacement sur la carte */
Map_move(In p_interact : HCI, In p_mode : HCI_modalites) =
  SELECT
     map@pre = TRUE and
     (
       ((p_interact= LEFT_ARROW or p_interact = RIGHT_ARROW or p_interact =
UP_ARROW or p_interact = DOWN_ARROW)
          and p_mode = clavier)
      or
      (p_interact = CLICK_ON_MAP and p_mode = souris))
  THEN
     /* verification */
     interaction := p_interact and
```

```
     cov_commandes := cov_commandes@pre \/ {Map_move} and
     cov_modalites := cov_modalites@pre \/ {p_mode}
   END ;



/* Lancement de la commande de recherche */
Search(In p_interact : HCI, In p_mode : HCI_modalites) =
   SELECT
     EV1@pre = 2 and
     ((p_interact = CLICK_ON_SEARCH and p_mode  = souris)
      or
       ((p_interact = SAY_SEARCH and p_mode  = microphone)
      or
        (p_interact = ENTER_KEY and p_mode  = clavier)))
     and name_status@pre = set and address_statut@pre = set
   THEN
     EV1 := 1 and
     /* verification */
     interaction := p_interact and
     cov_commandes := cov_commandes@pre \/ {Search} and
     cov_modalites := cov_modalites@pre \/ {p_mode}
   END ;

/* Traitement de la commande : envoi */
Query =
   SELECT
     EV1@pre = 1
   THEN
     Query_sending := TRUE and EV1 := 0
   END;

/* Traitement de la commande : résultat */
Result_Query =
   SELECT
     Query_sending@pre = TRUE
   THEN
     Query_sending := FALSE and
     Query_ready := TRUE and
     EV1 := 3 and
     map := TRUE and
     cov_modalites := {} and /* on vide la liste des modalités utilisées */
     cov_commandes := {}
   END;


/********************

Vérification de la complémentarité

********************/

/* Complémentarité de la voix et du clavier pour la saisie */

verif_complementarite_1_ok  =
   BEGIN
     name_statut@pre = set and      /* la valeur set indique que le nom a
été saisi */
     address_statut@pre = set and    /* la valeur set indique que l'adresse
a été saisie */
     cov_modalites@pre = {microphone, clavier} and   /*  */
     cov_commandes@pre = {Input_Name_field, Input_Name_value,
Input_Address_field, Input_Address_value}
   END;

verif_complementarite_1_ko_clavier =
   BEGIN
     name_statut@pre = set and      /* la valeur set indique que le nom a
été saisi */
```

```
     address_statut@pre = set and    /* la valeur set indique que l'adresse
a été saisie */
     cov_modalites@pre = {clavier} and /*  */
     cov_commandes@pre = {Input_Name_field, Input_Name_value,
Input_Address_field, Input_Address_value}
   END;

verif_complementarite_1_ko_micro =
   BEGIN
     name_statut@pre = set and      /* la valeur set indique que le nom a
été saisi */
     address_statut@pre = set and    /* la valeur set indique que l'adresse
a été saisie */
     cov_modalites@pre = {microphone}  and
     cov_commandes@pre = {Input_Name_field, Input_Name_value,
Input_Address_field, Input_Address_value}/*  */
   END;


END
```

4.2 Transitions de tests produites

initialisation	
tdt0.1	name = 0 and (address = 0 and (Query_sending = 0 and (Query_ready = 1 and (map = 0 and (EV1 = 3 and (lock_name = 0 and (lock_address = 0 and (name_statut = 0 and (address_statut = 0 and (cov_commandes={} and (cov_modalites={} and interaction = 0))))))))))

Name_Address	
tdt1.1	EV1@pre = 3 and EV1 = EV1@pre-1

Input_Name_field	
tdt2.1	p_interact : 0..17 and (p_mode : 0..2 and (lock_name@pre = 0 and (EV1@pre = 2 and (p_interact = 1 and p_mode = 1)) and (lock_name = 1 and (name_statut = 0 and (interaction = p_interact and (cov_commandes=cov_commandes@pre \/ V706797 and (V705626 < 7 and 0 <= V705626 and (V705615= set(V705626) and cov_commandes@pre<:V705615)) and (cov_modalites=cov_modalites@pre \/ V718204 and (V717033 < 3 and 0 <= V717033 and (V717022= set(V717033) and cov_modalites@pre<:V717022))))))))

Input_Name_value	
tdt3.1	p_new_name : 0..2 and (p_interact : 0..17 and (p_mode : 0..2 and (lock_name@pre = 1 and (EV1@pre = 2 and (p_interact = 12 and p_mode = 0)) and (name = p_new_name and (lock_name = 0 and (name_statut = 1 and (interaction = p_interact and (cov_commandes=cov_commandes@pre

	\/ V782533 and (V781362 < 7 and 0 <= V781362 and (V781351= set(V781362) and cov_commandes@pre<:V781351)) and (cov_modalites=cov_modalites@pre \/ V793940 and (V792769 < 3 and 0 <= V792769 and (V792758= set(V792769) and cov_modalites@pre<:V792758)))))))))))

Input_Address_field	
tdt4.1	p_interact : 0..17 and (p_mode : 0..2 and (lock_address@pre = 0 and (EV1@pre = 2 and (p_interact = 2 and p_mode = 1)) and (lock_address = 1 and (address_statut = 0 and (interaction = p_interact and (cov_commandes=cov_commandes@pre \/ V337200 and (V336029 < 7 and 0 <= V336029 and (V336018= set(V336029) and cov_commandes@pre<:V336018)) and (cov_modalites=cov_modalites@pre \/ V348607 and (V347436 < 3 and 0 <= V347436 and (V347425= set(V347436) and cov_modalites@pre<:V347425))))))))

Input_Address_value	
tdt5.1	p_interact : 0..17 and (p_mode : 0..2 and (p_new_address : 0..2 and (lock_address@pre = 1 and (EV1@pre = 2 and (p_interact = 12 and p_mode = 0)) and (address = p_new_address and (address_statut = 1 and (lock_address = 0 and (interaction = p_interact and (cov_commandes=cov_commandes@pre \/ V413024 and (V411853 < 7 and 0 <= V411853 and (V411842= set(V411853) and cov_commandes@pre<:V411842)) and (cov_modalites=cov_modalites@pre \/ V424431 and (V423260 < 3 and 0 <= V423260 and (V423249= set(V423260) and cov_modalites@pre<:V423249)))))))))))

Map_zoom	
tdt6.1	p_interact : 0..17 and (p_mode : 0..2 and (p_interact = 6 and (map@pre = 1 and ((p_interact = 6 or p_interact = 5) and p_mode = 2) and (interaction = p_interact and (cov_commandes=cov_commandes@pre \/ V477651 and (V476480 < 7 and 0 <= V476480 and (V476469= set(V476480) and cov_commandes@pre<:V476469)) and (cov_modalites=cov_modalites@pre \/ V489058 and (V487887 < 3 and 0 <= V487887 and (V487876= set(V487887) and cov_modalites@pre<:V487876))))))))
tdt6.2	p_interact : 0..17 and (p_mode : 0..2 and (p_interact = 5 and (map@pre = 1 and ((p_interact = 6 or p_interact = 5) and p_mode = 2) and (interaction = p_interact and (cov_commandes=cov_commandes@pre \/ V477651 and (V476480 < 7 and 0 <= V476480 and (V476469= set(V476480) and cov_commandes@pre<:V476469)) and (cov_modalites=cov_modalites@pre \/ V489058 and (V487887 < 3 and 0 <= V487887 and (V487876= set(V487887) and cov_modalites@pre<:V487876))))))))

Map_move

tdt7.1	p_interact : 0..17 and (p_mode : 0..2 and (p_interact = 14 and (p_mode = 0 and (map@pre = 1 and ((p_interact = 14 or (p_interact = 15 or (p_interact = 16 or p_interact = 17))) and p_mode = 0 or p_interact = 11 and p_mode = 2) and (interaction = p_interact and (cov_commandes=cov_commandes@pre \/ V561417 and (V560246 < 7 and 0 <= V560246 and (V560235= set(V560246) and cov_commandes@pre<:V560235)) and (cov_modalites=cov_modalites@pre \/ V572824 and (V571653 < 3 and 0 <= V571653 and (V571642= set(V571653) and cov_modalites@pre<:V571642))))))))
tdt7.2	p_interact : 0..17 and (p_mode : 0..2 and (p_interact = 15 and (p_mode = 0 and (map@pre = 1 and ((p_interact = 14 or (p_interact = 15 or (p_interact = 16 or p_interact = 17))) and p_mode = 0 or p_interact = 11 and p_mode = 2) and (interaction = p_interact and (cov_commandes=cov_commandes@pre \/ V561417 and (V560246 < 7 and 0 <= V560246 and (V560235= set(V560246) and cov_commandes@pre<:V560235)) and (cov_modalites=cov_modalites@pre \/ V572824 and (V571653 < 3 and 0 <= V571653 and (V571642= set(V571653) and cov_modalites@pre<:V571642))))))))
tdt7.3	p_interact : 0..17 and (p_mode : 0..2 and (p_interact = 16 and (p_mode = 0 and (map@pre = 1 and ((p_interact = 14 or (p_interact = 15 or (p_interact = 16 or p_interact = 17))) and p_mode = 0 or p_interact = 11 and p_mode = 2) and (interaction = p_interact and (cov_commandes=cov_commandes@pre \/ V561417 and (V560246 < 7 and 0 <= V560246 and (V560235= set(V560246) and cov_commandes@pre<:V560235)) and (cov_modalites=cov_modalites@pre \/ V572824 and (V571653 < 3 and 0 <= V571653 and (V571642= set(V571653) and cov_modalites@pre<:V571642))))))))
tdt7.4	p_interact : 0..17 and (p_mode : 0..2 and (p_interact = 17 and (p_mode = 0 and (map@pre = 1 and ((p_interact = 14 or (p_interact = 15 or (p_interact = 16 or p_interact = 17))) and p_mode = 0 or p_interact = 11 and p_mode = 2) and (interaction = p_interact and (cov_commandes=cov_commandes@pre \/ V561417 and (V560246 < 7 and 0 <= V560246 and (V560235= set(V560246) and cov_commandes@pre<:V560235)) and (cov_modalites=cov_modalites@pre \/ V572824 and (V571653 < 3 and 0 <= V571653 and (V571642= set(V571653) and cov_modalites@pre<:V571642))))))))
tdt7.5	p_interact : 0..17 and (p_mode : 0..2 and (p_interact = 11 and (p_mode = 2 and (map@pre = 1 and ((p_interact = 14 or (p_interact = 15 or (p_interact = 16 or p_interact = 17))) and p_mode = 0 or p_interact = 11 and p_mode = 2) and (interaction = p_interact and (cov_commandes=cov_commandes@pre \/ V561417 and (V560246 < 7 and 0 <= V560246 and (V560235= set(V560246) and cov_commandes@pre<:V560235)) and (cov_modalites=cov_modalites@pre \/ V572824 and (V571653 < 3 and 0 <= V571653 and (V571642= set(V571653) and cov_modalites@pre<:V571642))))))))

Search	
tdt8.1	p_interact : 0..17 and (p_mode : 0..2 and (p_interact = 10 and (p_mode = 2 and (EV1@pre = 2 and ((p_interact = 10 and p_mode = 2 or (p_interact = 4

	and p_mode = 1 or p_interact = 13 and p_mode = 0)) and (name_status@pre = set and address_statut@pre = 1)) and (EV1 = 1 and (interaction = p_interact and (cov_commandes=cov_commandes@pre \/ V652711 and (V651540 < 7 and 0 <= V651540 and (V651529= set(V651540) and cov_commandes@pre<:V651529)) and (cov_modalites=cov_modalites@pre \/ V664118 and (V662947 < 3 and 0 <= V662947 and (V662936= set(V662947) and cov_modalites@pre<:V662936)))))))))
tdt8.2	p_interact : 0..17 and (p_mode : 0..2 and (p_interact = 4 and (p_mode = 1 and (EV1@pre = 2 and ((p_interact = 10 and p_mode = 2 or (p_interact = 4 and p_mode = 1 or p_interact = 13 and p_mode = 0)) and (name_status@pre = set and address_statut@pre = 1)) and (EV1 = 1 and (interaction = p_interact and (cov_commandes=cov_commandes@pre \/ V652711 and (V651540 < 7 and 0 <= V651540 and (V651529= set(V651540) and cov_commandes@pre<:V651529)) and (cov_modalites=cov_modalites@pre \/ V664118 and (V662947 < 3 and 0 <= V662947 and (V662936= set(V662947) and cov_modalites@pre<:V662936)))))))))
tdt8.3	p_interact : 0..17 and (p_mode : 0..2 and (p_interact = 13 and (p_mode = 0 and (EV1@pre = 2 and ((p_interact = 10 and p_mode = 2 or (p_interact = 4 and p_mode = 1 or p_interact = 13 and p_mode = 0)) and (name_status@pre = set and address_statut@pre = 1)) and (EV1 = 1 and (interaction = p_interact and (cov_commandes=cov_commandes@pre \/ V652711 and (V651540 < 7 and 0 <= V651540 and (V651529= set(V651540) and cov_commandes@pre<:V651529)) and (cov_modalites=cov_modalites@pre \/ V664118 and (V662947 < 3 and 0 <= V662947 and (V662936= set(V662947) and cov_modalites@pre<:V662936)))))))))

Query	
tdt9.1	EV1@pre = 1 and (Query_sending = 1 and EV1 = 0)

Result_Query	
tdt10.1	Query_sending@pre = 1 and (Query_sending = 0 and (Query_ready = 1 and (EV1 = 3 and (map = 1 and (cov_modalites={} and (cov_commandes={})))))))

verif_complementarite_1_ok	
tdt11.1	name_statut@pre = 1 and (address_statut@pre = 1 and (cov_modalites@pre={1,0} and (cov_commandes@pre={0,1,2,3})))

verif_complementarite_1_ko_clavier	
tdt12.1	name_statut@pre = 1 and (address_statut@pre = 1 and (cov_modalites@pre={0} and (cov_commandes@pre={0,1,2,3})))

verif_complementarite_1_ko_micro	
tdt13.1	name_statut@pre = 1 and (address_statut@pre = 1 and (cov_modalites@pre={1} and (cov_commandes@pre={0,1,2,3})))

4.3 Jeux de tests produits

4.3.1 test_1.html

Etape n°0	**initialisation (*tdt0.1*)**
Paramètres	-
Etat	Query_sending = FALSE Query_ready = TRUE map = FALSE lock_name = FALSE name = Lionel name_statut = unset lock_address = FALSE address = Rennes address_statut = unset EV1 = 3 interaction = NONE cov_modalites= {} cov_commandes= {}

Etape n°1	**Name_Address (*tdt1.1*)**
Paramètres	-
Etat	Query_sending = FALSE Query_ready = TRUE map = FALSE lock_name = FALSE name = Lionel name_statut = unset lock_address = FALSE address = Rennes address_statut = unset EV1 = 2 interaction = NONE cov_modalites= {} cov_commandes= {}

Etape n°2	**Input_Name_field (*tdt2.1*)**
Paramètres	p_interact = SAY_NAME p_mode = microphone
Etat	Query_sending = FALSE

	Query_ready = TRUE map = FALSE lock_name = TRUE name = Lionel name_statut = unset lock_address = FALSE address = Rennes address_statut = unset EV1 = 2 interaction = SAY_NAME cov_modalites= {microphone} cov_commandes= {Input_Name_field}

Etape n°3	**Input_Name_value (*tdt3.1*)**
Paramètres	p_new_name = Lionel p_interact = INPUT_TEXT p_mode = clavier
Etat	Query_sending = FALSE Query_ready = TRUE map = FALSE lock_name = FALSE name = Lionel name_statut = set lock_address = FALSE address = Rennes address_statut = unset EV1 = 2 interaction = INPUT_TEXT cov_modalites= {clavier, microphone} cov_commandes= {Input_Name_field, Input_Name_value}

Etape n°4	**Input_Address_field (*tdt4.1*)**
Paramètres	p_interact = SAY_ADDRESS p_mode = microphone
Etat	Query_sending = FALSE Query_ready = TRUE map = FALSE lock_name = FALSE name = Lionel name_statut = set lock_address = TRUE address = Rennes address_statut = unset EV1 = 2 interaction = SAY_ADDRESS cov_modalites= {clavier, microphone} cov_commandes= {Input_Name_field, Input_Name_value, Input_Address_field}

Etape n°5	**Input_Address_value (*tdt5.1*)**

Paramètres	p_interact = INPUT_TEXT p_mode = clavier p_new_address = Rennes
Etat	Query_sending = FALSE Query_ready = TRUE map = FALSE lock_name = FALSE name = Lionel name_statut = set lock_address = FALSE address = Rennes address_statut = set EV1 = 2 interaction = INPUT_TEXT cov_modalites= {clavier, microphone} cov_commandes= {Input_Name_field, Input_Name_value, Input_Address_field, Input_Address_value}

Etape n°6	**Search (*tdt8.1*)**
Paramètres	p_interact = CLICK_ON_SEARCH p_mode = souris
Etat	Query_sending = FALSE Query_ready = TRUE map = FALSE lock_name = FALSE name = Lionel name_statut = set lock_address = FALSE address = Rennes address_statut = set EV1 = 1 interaction = CLICK_ON_SEARCH cov_modalites= {clavier, microphone, souris} cov_commandes= {Input_Name_field, Input_Name_value, Input_Address_field, Input_Address_value, Search}

Etape n°7	**Query (*tdt9.1*)**
Paramètres	-
Etat	Query_sending = TRUE Query_ready = TRUE map = FALSE lock_name = FALSE name = Lionel name_statut = set lock_address = FALSE address = Rennes address_statut = set EV1 = 0 interaction = CLICK_ON_SEARCH cov_modalites= {clavier, microphone, souris}

	cov_commandes= {Input_Name_field, Input_Name_value, Input_Address_field, Input_Address_value, Search}

Etape n°8	**Result_Query (*tdt10.1*)**
Paramètres	-
Etat	Query_sending = FALSE Query_ready = TRUE map = TRUE lock_name = FALSE name = Lionel name_statut = set lock_address = FALSE address = Rennes address_statut = set EV1 = 3 interaction = CLICK_ON_SEARCH cov_modalites= {} cov_commandes= {}

Etape n°9	**Map_zoom (*tdt6.1*)**
Paramètres	p_interact = ZOOM_OUT p_mode = souris
Etat	Query_sending = FALSE Query_ready = TRUE map = TRUE lock_name = FALSE name = Lionel name_statut = set lock_address = FALSE address = Rennes address_statut = set EV1 = 3 interaction = ZOOM_OUT cov_modalites= {souris} cov_commandes= {Map_zoom}

Etape n°10	**Map_zoom (*tdt6.2*)**
Paramètres	p_interact = ZOOM_IN p_mode = souris
Etat	Query_sending = FALSE Query_ready = TRUE map = TRUE lock_name = FALSE name = Lionel name_statut = set lock_address = FALSE address = Rennes address statut = set

	EV1 = 3 interaction = ZOOM_IN cov_modalites= {souris} cov_commandes= {Map_zoom}

Etape n°11	**Map_move (*tdt7.1*)**
Paramètres	p_interact = LEFT_ARROW p_mode = clavier
Etat	Query_sending = FALSE Query_ready = TRUE map = TRUE lock_name = FALSE name = Lionel name_statut = set lock_address = FALSE address = Rennes address_statut = set EV1 = 3 interaction = LEFT_ARROW cov_modalites= {clavier, souris} cov_commandes= {Map_move, Map_zoom}

Etape n°12	**Map_move (*tdt7.2*)**
Paramètres	p_interact = RIGHT_ARROW p_mode = clavier
Etat	Query_sending = FALSE Query_ready = TRUE map = TRUE lock_name = FALSE name = Lionel name_statut = set lock_address = FALSE address = Rennes address_statut = set EV1 = 3 interaction = RIGHT_ARROW cov_modalites= {clavier, souris} cov_commandes= {Map_move, Map_zoom}

Etape n°13	**Map_move (*tdt7.3*)**
Paramètres	p_interact = UP_ARROW p_mode = clavier
Etat	Query_sending = FALSE Query_ready = TRUE map = TRUE lock_name = FALSE name = Lionel name_statut = set lock_address = FALSE

	address = Rennes address_statut = set EV1 = 3 interaction = UP_ARROW cov_modalites= {clavier, souris} cov_commandes= {Map_move, Map_zoom}

Etape n°14	**Map_move (*tdt7.4*)**
Paramètres	p_interact = DOWN_ARROW p_mode = clavier
Etat	Query_sending = FALSE Query_ready = TRUE map = TRUE lock_name = FALSE name = Lionel name_statut = set lock_address = FALSE address = Rennes address_statut = set EV1 = 3 interaction = DOWN_ARROW cov_modalites= {clavier, souris} cov_commandes= {Map_move, Map_zoom}

Etape n°15	**Map_move (*tdt7.5*)**
Paramètres	p_interact = CLICK_ON_MAP p_mode = souris
Etat	Query_sending = FALSE Query_ready = TRUE map = TRUE lock_name = FALSE name = Lionel name_statut = set lock_address = FALSE address = Rennes address_statut = set EV1 = 3 interaction = CLICK_ON_MAP cov_modalites= {clavier, souris} cov_commandes= {Map_move, Map_zoom}

4.3.2 test_2.html

Etape n°0	**initialisation (*tdt0.1*)**
Paramètres	-
Etat	Query_sending = FALSE Query_ready = TRUE

	map = FALSE lock_name = FALSE name = Lionel name_statut = unset lock_address = FALSE address = Rennes address_statut = unset EV1 = 3 interaction = NONE cov_modalites= {} cov_commandes= {}

Etape n°1	**Name_Address (*tdt1.1*)**
Paramètres	-
Etat	Query_sending = FALSE Query_ready = TRUE map = FALSE lock_name = FALSE name = Lionel name_statut = unset lock_address = FALSE address = Rennes address_statut = unset EV1 = 2 interaction = NONE cov_modalites= {} cov_commandes= {}

Etape n°2	**Input_Name_field (*tdt2.1*)**
Paramètres	p_interact = SAY_NAME p_mode = microphone
Etat	Query_sending = FALSE Query_ready = TRUE map = FALSE lock_name = TRUE name = Lionel name_statut = unset lock_address = FALSE address = Rennes address_statut = unset EV1 = 2 interaction = SAY_NAME cov_modalites= {microphone} cov_commandes= {Input_Name_field}

Etape n°3	**Input_Name_value (*tdt3.1*)**
Paramètres	p_new_name = Lionel p_interact = INPUT_TEXT p_mode = clavier

Etat	Query_sending = FALSE Query_ready = TRUE map = FALSE lock_name = FALSE name = Lionel name_statut = set lock_address = FALSE address = Rennes address_statut = unset EV1 = 2 interaction = INPUT_TEXT cov_modalites= {clavier, microphone} cov_commandes= {Input_Name_field, Input_Name_value}

Etape n°4	**Input_Address_field (*tdt4.1*)**
Paramètres	p_interact = SAY_ADDRESS p_mode = microphone
Etat	Query_sending = FALSE Query_ready = TRUE map = FALSE lock_name = FALSE name = Lionel name_statut = set lock_address = TRUE address = Rennes address_statut = unset EV1 = 2 interaction = SAY_ADDRESS cov_modalites= {clavier, microphone} cov_commandes= {Input_Name_field, Input_Name_value, Input_Address_field}

Etape n°5	**Input_Address_value (*tdt5.1*)**
Paramètres	p_interact = INPUT_TEXT p_mode = clavier p_new_address = Rennes
Etat	Query_sending = FALSE Query_ready = TRUE map = FALSE lock_name = FALSE name = Lionel name_statut = set lock_address = FALSE address = Rennes address_statut = set EV1 = 2 interaction = INPUT_TEXT cov_modalites= {clavier, microphone} cov_commandes= {Input_Name_field, Input_Name_value, Input_Address_field, Input_Address_value}

Etape n°6	**Search (*tdt8.2*)**
Paramètres	p_interact = SAY_SEARCH p_mode = microphone
Etat	Query_sending = FALSE Query_ready = TRUE map = FALSE lock_name = FALSE name = Lionel name_statut = set lock_address = FALSE address = Rennes address_statut = set EV1 = 1 interaction = SAY_SEARCH cov_modalites= {clavier, microphone} cov_commandes= {Input_Name_field, Input_Name_value, Input_Address_field, Input_Address_value, Search}

Etape n°7	**Query (*tdt9.1*)**
Paramètres	-
Etat	Query_sending = TRUE Query_ready = TRUE map = FALSE lock_name = FALSE name = Lionel name_statut = set lock_address = FALSE address = Rennes address_statut = set EV1 = 0 interaction = SAY_SEARCH cov_modalites= {clavier, microphone} cov_commandes= {Input_Name_field, Input_Name_value, Input_Address_field, Input_Address_value, Search}

Etape n°8	**Result_Query (*tdt10.1*)**
Paramètres	-
Etat	Query_sending = FALSE Query_ready = TRUE map = TRUE lock_name = FALSE name = Lionel name_statut = set lock_address = FALSE address = Rennes address_statut = set EV1 = 3

	interaction = SAY_SEARCH cov_modalites= {} cov_commandes= {}

Etape n°9	**Map_zoom (*tdt6.1*)**
Paramètres	p_interact = ZOOM_OUT p_mode = souris
Etat	Query_sending = FALSE Query_ready = TRUE map = TRUE lock_name = FALSE name = Lionel name_statut = set lock_address = FALSE address = Rennes address_statut = set EV1 = 3 interaction = ZOOM_OUT cov_modalites= {souris} cov_commandes= {Map_zoom}

Etape n°10	**Map_zoom (*tdt6.2*)**
Paramètres	p_interact = ZOOM_IN p_mode = souris
Etat	Query_sending = FALSE Query_ready = TRUE map = TRUE lock_name = FALSE name = Lionel name_statut = set lock_address = FALSE address = Rennes address_statut = set EV1 = 3 interaction = ZOOM_IN cov_modalites= {souris} cov_commandes= {Map_zoom}

Etape n°11	**Map_move (*tdt7.1*)**
Paramètres	p_interact = LEFT_ARROW p_mode = clavier
Etat	Query_sending = FALSE Query_ready = TRUE map = TRUE lock_name = FALSE name = Lionel name_statut = set lock_address = FALSE address = Rennes

	address_statut = set EV1 = 3 interaction = LEFT_ARROW cov_modalites= {clavier, souris} cov_commandes= {Map_move, Map_zoom}

Etape n°12	**Map_move (*tdt7.2*)**
Paramètres	p_interact = RIGHT_ARROW p_mode = clavier
Etat	Query_sending = FALSE Query_ready = TRUE map = TRUE lock_name = FALSE name = Lionel name_statut = set lock_address = FALSE address = Rennes address_statut = set EV1 = 3 interaction = RIGHT_ARROW cov_modalites= {clavier, souris} cov_commandes= {Map_move, Map_zoom}

Etape n°13	**Map_move (*tdt7.3*)**
Paramètres	p_interact = UP_ARROW p_mode = clavier
Etat	Query_sending = FALSE Query_ready = TRUE map = TRUE lock_name = FALSE name = Lionel name_statut = set lock_address = FALSE address = Rennes address_statut = set EV1 = 3 interaction = UP_ARROW cov_modalites= {clavier, souris} cov_commandes= {Map_move, Map_zoom}

Etape n°14	**Map_move (*tdt7.4*)**
Paramètres	p_interact = DOWN_ARROW p_mode = clavier
Etat	Query_sending = FALSE Query_ready = TRUE map = TRUE lock_name = FALSE name = Lionel name statut = set

	lock_address = FALSE address = Rennes address_statut = set EV1 = 3 interaction = DOWN_ARROW cov_modalites= {clavier, souris} cov_commandes= {Map_move, Map_zoom}

Etape n°15	**Map_move (*tdt7.5*)**
Paramètres	p_interact = CLICK_ON_MAP p_mode = souris
Etat	Query_sending = FALSE Query_ready = TRUE map = TRUE lock_name = FALSE name = Lionel name_statut = set lock_address = FALSE address = Rennes address_statut = set EV1 = 3 interaction = CLICK_ON_MAP cov_modalites= {clavier, souris} cov_commandes= {Map_move, Map_zoom}

4.3.3 test_3.html

Etape n°0	**initialisation (*tdt0.1*)**
Paramètres	-
Etat	Query_sending = FALSE Query_ready = TRUE map = FALSE lock_name = FALSE name = Lionel name_statut = unset lock_address = FALSE address = Rennes address_statut = unset EV1 = 3 interaction = NONE cov_modalites= {} cov_commandes= {}

Etape n°1	**Name_Address (*tdt1.1*)**
Paramètres	-
Etat	Query_sending = FALSE Query_ready = TRUE map = FALSE lock name = FALSE

	name = Lionel name_statut = unset lock_address = FALSE address = Rennes address_statut = unset EV1 = 2 interaction = NONE cov_modalites= {} cov_commandes= {}

Etape n°2	**Input_Name_field (*tdt2.1*)**
Paramètres	p_interact = SAY_NAME p_mode = microphone
Etat	Query_sending = FALSE Query_ready = TRUE map = FALSE lock_name = TRUE name = Lionel name_statut = unset lock_address = FALSE address = Rennes address_statut = unset EV1 = 2 interaction = SAY_NAME cov_modalites= {microphone} cov_commandes= {Input_Name_field}

Etape n°3	**Input_Name_value (*tdt3.1*)**
Paramètres	p_new_name = Lionel p_interact = INPUT_TEXT p_mode = clavier
Etat	Query_sending = FALSE Query_ready = TRUE map = FALSE lock_name = FALSE name = Lionel name_statut = set lock_address = FALSE address = Rennes address_statut = unset EV1 = 2 interaction = INPUT_TEXT cov_modalites= {clavier, microphone} cov_commandes= {Input_Name_field, Input_Name_value}

Etape n°4	**Input_Address_field (*tdt4.1*)**
Paramètres	p_interact = SAY_ADDRESS p_mode = microphone

Etat	Query_sending = FALSE Query_ready = TRUE map = FALSE lock_name = FALSE name = Lionel name_statut = set lock_address = TRUE address = Rennes address_statut = unset EV1 = 2 interaction = SAY_ADDRESS cov_modalites= {clavier, microphone} cov_commandes= {Input_Name_field, Input_Name_value, Input_Address_field}

Etape n°5	**Input_Address_value (*tdt5.1*)**
Paramètres	p_interact = INPUT_TEXT p_mode = clavier p_new_address = Rennes
Etat	Query_sending = FALSE Query_ready = TRUE map = FALSE lock_name = FALSE name = Lionel name_statut = set lock_address = FALSE address = Rennes address_statut = set EV1 = 2 interaction = INPUT_TEXT cov_modalites= {clavier, microphone} cov_commandes= {Input_Name_field, Input_Name_value, Input_Address_field, Input_Address_value}

Etape n°6	**Search (*tdt8.3*)**
Paramètres	p_interact = ENTER_KEY p_mode = clavier
Etat	Query_sending = FALSE Query_ready = TRUE map = FALSE lock_name = FALSE name = Lionel name_statut = set lock_address = FALSE address = Rennes address_statut = set EV1 = 1 interaction = ENTER_KEY cov_modalites= {clavier, microphone} cov_commandes= {Input_Name_field, Input_Name_value, Input_Address_field,

	Input_Address_value, Search}

Etape n°7	**Query (*tdt9.1*)**
Paramètres	-
Etat	Query_sending = TRUE Query_ready = TRUE map = FALSE lock_name = FALSE name = Lionel name_statut = set lock_address = FALSE address = Rennes address_statut = set EV1 = 0 interaction = ENTER_KEY cov_modalites= {clavier, microphone} cov_commandes= {Input_Name_field, Input_Name_value, Input_Address_field, Input_Address_value, Search}

Etape n°8	**Result_Query (*tdt10.1*)**
Paramètres	-
Etat	Query_sending = FALSE Query_ready = TRUE map = TRUE lock_name = FALSE name = Lionel name_statut = set lock_address = FALSE address = Rennes address_statut = set EV1 = 3 interaction = ENTER_KEY cov_modalites= {} cov_commandes= {}

Etape n°9	**Map_zoom (*tdt6.1*)**
Paramètres	p_interact = ZOOM_OUT p_mode = souris
Etat	Query_sending = FALSE Query_ready = TRUE map = TRUE lock_name = FALSE name = Lionel name_statut = set lock_address = FALSE address = Rennes address_statut = set EV1 = 3

	interaction = ZOOM_OUT cov_modalites= {souris} cov_commandes= {Map_zoom}

Etape n°10	**Map_zoom (*tdt6.2*)**
Paramètres	p_interact = ZOOM_IN p_mode = souris
Etat	Query_sending = FALSE Query_ready = TRUE map = TRUE lock_name = FALSE name = Lionel name_statut = set lock_address = FALSE address = Rennes address_statut = set EV1 = 3 interaction = ZOOM_IN cov_modalites= {souris} cov_commandes= {Map_zoom}

Etape n°11	**Map_move (*tdt7.1*)**
Paramètres	p_interact = LEFT_ARROW p_mode = clavier
Etat	Query_sending = FALSE Query_ready = TRUE map = TRUE lock_name = FALSE name = Lionel name_statut = set lock_address = FALSE address = Rennes address_statut = set EV1 = 3 interaction = LEFT_ARROW cov_modalites= {clavier, souris} cov_commandes= {Map_move, Map_zoom}

Etape n°12	**Map_move (*tdt7.2*)**
Paramètres	p_interact = RIGHT_ARROW p_mode = clavier
Etat	Query_sending = FALSE Query_ready = TRUE map = TRUE lock_name = FALSE name = Lionel name_statut = set lock_address = FALSE address = Rennes

	address_statut = set EV1 = 3 interaction = RIGHT_ARROW cov_modalites= {clavier, souris} cov_commandes= {Map_move, Map_zoom}

Etape n°13	**Map_move (*tdt7.3*)**
Paramètres	p_interact = UP_ARROW p_mode = clavier
Etat	Query_sending = FALSE Query_ready = TRUE map = TRUE lock_name = FALSE name = Lionel name_statut = set lock_address = FALSE address = Rennes address_statut = set EV1 = 3 interaction = UP_ARROW cov_modalites= {clavier, souris} cov_commandes= {Map_move, Map_zoom}

Etape n°14	**Map_move (*tdt7.4*)**
Paramètres	p_interact = DOWN_ARROW p_mode = clavier
Etat	Query_sending = FALSE Query_ready = TRUE map = TRUE lock_name = FALSE name = Lionel name_statut = set lock_address = FALSE address = Rennes address_statut = set EV1 = 3 interaction = DOWN_ARROW cov_modalites= {clavier, souris} cov_commandes= {Map_move, Map_zoom}

Etape n°15	**Map_move (*tdt7.5*)**
Paramètres	p_interact = CLICK_ON_MAP p_mode = souris
Etat	Query_sending = FALSE Query_ready = TRUE map = TRUE lock_name = FALSE name = Lionel name statut = set

	lock_address = FALSE address = Rennes address_statut = set EV1 = 3 interaction = CLICK_ON_MAP cov_modalites= {clavier, souris} cov_commandes= {Map_move, Map_zoom}

4.3.4 test_4.html

Etape n°0	**initialisation (*tdt0.1*)**
Paramètres	-
Etat	Query_sending = FALSE Query_ready = TRUE map = FALSE lock_name = FALSE name = Lionel name_statut = unset lock_address = FALSE address = Rennes address_statut = unset EV1 = 3 interaction = NONE cov_modalites= {} cov_commandes= {}

Etape n°1	**Name_Address (*tdt1.1*)**
Paramètres	-
Etat	Query_sending = FALSE Query_ready = TRUE map = FALSE lock_name = FALSE name = Lionel name_statut = unset lock_address = FALSE address = Rennes address_statut = unset EV1 = 2 interaction = NONE cov_modalites= {} cov_commandes= {}

Etape n°2	**Input_Name_field (*tdt2.1*)**
Paramètres	p_interact = SAY_NAME p_mode = microphone
Etat	Query_sending = FALSE Query_ready = TRUE map = FALSE lock name = TRUE

	name = Lionel name_statut = unset lock_address = FALSE address = Rennes address_statut = unset EV1 = 2 interaction = SAY_NAME cov_modalites= {microphone} cov_commandes= {Input_Name_field}

Etape n°3	**Input_Name_value (*tdt3.1*)**
Paramètres	p_new_name = Lionel p_interact = INPUT_TEXT p_mode = clavier
Etat	Query_sending = FALSE Query_ready = TRUE map = FALSE lock_name = FALSE name = Lionel name_statut = set lock_address = FALSE address = Rennes address_statut = unset EV1 = 2 interaction = INPUT_TEXT cov_modalites= {clavier, microphone} cov_commandes= {Input_Name_field, Input_Name_value}

Etape n°4	**Input_Address_field (*tdt4.1*)**
Paramètres	p_interact = SAY_ADDRESS p_mode = microphone
Etat	Query_sending = FALSE Query_ready = TRUE map = FALSE lock_name = FALSE name = Lionel name_statut = set lock_address = TRUE address = Rennes address_statut = unset EV1 = 2 interaction = SAY_ADDRESS cov_modalites= {clavier, microphone} cov_commandes= {Input_Name_field, Input_Name_value, Input_Address_field}

Etape n°5	**Input_Address_value (*tdt5.1*)**
Paramètres	p_interact = INPUT_TEXT p_ mode = clavier

	p_new_address = Rennes
Etat	Query_sending = FALSE Query_ready = TRUE map = FALSE lock_name = FALSE name = Lionel name_statut = set lock_address = FALSE address = Rennes address_statut = set EV1 = 2 interaction = INPUT_TEXT cov_modalites= {clavier, microphone} cov_commandes= {Input_Name_field, Input_Name_value, Input_Address_field, Input_Address_value}

Etape n°6	**verif_complementarite_1_ok (*tdt11.1*)**
Paramètres	-
Etat	Query_sending = FALSE Query_ready = TRUE map = FALSE lock_name = FALSE name = Lionel name_statut = set lock_address = FALSE address = Rennes address_statut = set EV1 = 2 interaction = INPUT_TEXT cov_modalites= {clavier, microphone} cov_commandes= {Input_Name_field, Input_Name_value, Input_Address_field, Input_Address_value}

4.4 Mesures de couverture

Specification	***..\bevt_imappy_demo_compl_ok.html***

Global coverage measure	**90.48 % (19 / 21)**

Selection coverage measure	**90.48 % (19 / 21)**

Operation	**Coverage**
initialisation	100.00 % (1 / 1)

Name_Address	100.00 % (1 / 1)
Input_Name_field	100.00 % (1 / 1)
Input_Name_value	100.00 % (1 / 1)
Input_Address_field	100.00 % (1 / 1)
Input_Address_value	100.00 % (1 / 1)
Map_zoom	100.00 % (2 / 2)
Map_move	100.00 % (5 / 5)
Search	100.00 % (3 / 3)
Query	100.00 % (1 / 1)
Result_Query	100.00 % (1 / 1)
verif_complementarite_1_ok	100.00 % (1 / 1)
verif_complementarite_1_ko_clavier	0.00 % (0 / 1)
verif_complementarite_1_ko_micro	0.00 % (0 / 1)

4.4.1 Coverage measure per method

initialisation	**100.00 % (1 / 1)**
Transition n°0	100.00 % (1 / 1)
tdt0.1	YES [*Test_1*, *Test_2*, *Test_3*, *Test_4*]

Name_Address	**100.00 % (1 / 1)**
Transition n°1	100.00 % (1 / 1)
tdt1.1	YES [*Test_1*, *Test_2*, *Test_3*, *Test_4*]

Input_Name_field	**100.00 % (1 / 1)**
Transition n°2	100.00 % (1 / 1)
tdt2.1	YES [*Test_1*, *Test_2*, *Test_3*, *Test_4*]

Input_Name_value	**100.00 % (1 / 1)**
Transition n°3	100.00 % (1 / 1)
tdt3.1	YES [*Test_1*, *Test_2*,

	Test_3, *Test_4*]

Input_Address_field	**100.00 % (1 / 1)**
Transition n°4	100.00 % (1 / 1)
tdt4.1	YES [*Test_1*, *Test_2*, *Test_3*, *Test_4*]

Input_Address_value	**100.00 % (1 / 1)**
Transition n°5	100.00 % (1 / 1)
tdt5.1	YES [*Test_1*, *Test_2*, *Test_3*, *Test_4*]

Map_zoom	**100.00 % (2 / 2)**
Transition n°6	100.00 % (2 / 2)
tdt6.1	YES [*Test_1*, *Test_2*, *Test_3*]
tdt6.2	YES [*Test_1*, *Test_2*, *Test_3*]

Map_move	**100.00 % (5 / 5)**
Transition n°7	100.00 % (5 / 5)
tdt7.1	YES [*Test_1*, *Test_2*, *Test_3*]
tdt7.2	YES [*Test_1*, *Test_2*, *Test_3*]
tdt7.3	YES [*Test_1*, *Test_2*, *Test_3*]
tdt7.4	YES [*Test_1*, *Test_2*, *Test_3*]
tdt7.5	YES [*Test_1*, *Test_2*, *Test_3*]

Search	**100.00 % (3 / 3)**
Transition n°8	100.00 % (3 / 3)
tdt8.1	YES [*Test_1*]
tdt8.2	YES [*Test_2*]

tdt8.3	YES [*Test_3*]

Query	**100.00 % (1 / 1)**
Transition n°9	100.00 % (1 / 1)
tdt9.1	YES [*Test_1*, *Test_2*, *Test_3*]

Result_Query	**100.00 % (1 / 1)**
Transition n°10	100.00 % (1 / 1)
tdt10.1	YES [*Test_1*, *Test_2*, *Test_3*]

verif_complementarite_1_ok	**100.00 % (1 / 1)**
Transition n°11	100.00 % (1 / 1)
tdt11.1	YES [*Test_4*]

verif_complementarite_1_ko_clavier	0.00 %
Transition n°12	0.00 %
tdt12.1	NO

verif_complementarite_1_ko_micro	0.00 %
Transition n°13	0.00 %
tdt13.1	NO

4.4.2 Coverage measure per test

Test_1		
initialisation		100.00 % (1 / 1)
	Transition n°0	100.00 % (1 / 1)
Name_Address		100.00 % (1 / 1)
	Transition n°1	100.00 % (1 / 1)
Input_Name_field		100.00 % (1 / 1)
	Transition n°2	100.00 % (1 / 1)

Input_Name_value		100.00 % (1 / 1)
	Transition n°3	100.00 % (1 / 1)
Input_Address_field		100.00 % (1 / 1)
	Transition n°4	100.00 % (1 / 1)
Input_Address_value		100.00 % (1 / 1)
	Transition n°5	100.00 % (1 / 1)
Map_zoom		100.00 % (2 / 2)
	Transition n°6	100.00 % (2 / 2)
Map_move		100.00 % (5 / 5)
	Transition n°7	100.00 % (5 / 5)
Search		33.33 % (1 / 3)
	Transition n°8	33.33 % (1 / 3 including 1 only covered by this test)
Query		100.00 % (1 / 1)
	Transition n°9	100.00 % (1 / 1)
Result_Query		100.00 % (1 / 1)
	Transition n°10	100.00 % (1 / 1)
verif_complementarite_1_ok		0.00 %
verif_complementarite_1_ko_clavier		0.00 %
verif_complementarite_1_ko_micro		0.00 %

Test_2		
initialisation		100.00 % (1 / 1)
	Transition n°0	100.00 % (1 / 1)
Name_Address		100.00 % (1 / 1)
	Transition n°1	100.00 % (1 / 1)
Input_Name_field		100.00 % (1 / 1)
	Transition n°2	100.00 % (1 / 1)
Input_Name_value		100.00 % (1 / 1)

	Transition n°3	100.00 % (1 / 1)
Input_Address_field		100.00 % (1 / 1)
	Transition n°4	100.00 % (1 / 1)
Input_Address_value		100.00 % (1 / 1)
	Transition n°5	100.00 % (1 / 1)
Map_zoom		100.00 % (2 / 2)
	Transition n°6	100.00 % (2 / 2)
Map_move		100.00 % (5 / 5)
	Transition n°7	100.00 % (5 / 5)
Search		33.33 % (1 / 3)
	Transition n°8	33.33 % (1 / 3 including 1 only covered by this test)
Query		100.00 % (1 / 1)
	Transition n°9	100.00 % (1 / 1)
Result_Query		100.00 % (1 / 1)
	Transition n°10	100.00 % (1 / 1)
verif_complementarite_1_ok		0.00 %
verif_complementarite_1_ko_clavier		0.00 %
verif_complementarite_1_ko_micro		0.00 %

Test_3		
initialisation		100.00 % (1 / 1)
	Transition n°0	100.00 % (1 / 1)
Name_Address		100.00 % (1 / 1)
	Transition n°1	100.00 % (1 / 1)
Input_Name_field		100.00 % (1 / 1)
	Transition n°2	100.00 % (1 / 1)
Input_Name_value		100.00 % (1 / 1)
	Transition n°3	100.00 % (1 / 1)

Input_Address_field		100.00 % (1 / 1)
	Transition n°4	100.00 % (1 / 1)
Input_Address_value		100.00 % (1 / 1)
	Transition n°5	100.00 % (1 / 1)
Map_zoom		100.00 % (2 / 2)
	Transition n°6	100.00 % (2 / 2)
Map_move		100.00 % (5 / 5)
	Transition n°7	100.00 % (5 / 5)
Search		33.33 % (1 / 3)
	Transition n°8	33.33 % (1 / 3 including 1 only covered by this test)
Query		100.00 % (1 / 1)
	Transition n°9	100.00 % (1 / 1)
Result_Query		100.00 % (1 / 1)
	Transition n°10	100.00 % (1 / 1)
verif_complementarite_1_ok		0.00 %
verif_complementarite_1_ko_clavier		0.00 %
verif_complementarite_1_ko_micro		0.00 %

Test_4		
initialisation		100.00 % (1 / 1)
	Transition n°0	100.00 % (1 / 1)
Name_Address		100.00 % (1 / 1)
	Transition n°1	100.00 % (1 / 1)
Input_Name_field		100.00 % (1 / 1)
	Transition n°2	100.00 % (1 / 1)
Input_Name_value		100.00 % (1 / 1)
	Transition n°3	100.00 % (1 / 1)
Input_Address_field		100.00 % (1 / 1)

	Transition n°4	100.00 % (1 / 1)
Input_Address_value		100.00 % (1 / 1)
	Transition n°5	100.00 % (1 / 1)
Map_zoom		0.00 %
Map_move		0.00 %
Search		0.00 %
Query		0.00 %
Result_Query		0.00 %
verif_complementarite_1_ok		100.00 % (1 / 1)
	Transition n°11	100.00 % (1 / 1 including 1 only covered by this test)
verif_complementarite_1_ko_clavier		0.00 %
verif_complementarite_1_ko_micro		0.00 %