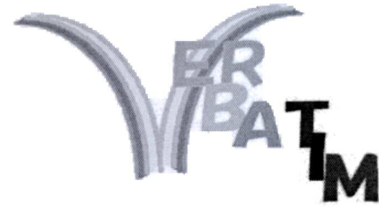
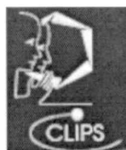
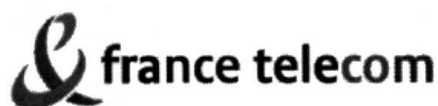


Démarche de Modélisation d'IHM3 avec B et CTT

SP 3
LOT 3
LISI/ENSMA

Auteurs : Y. Aït-Ameur, I. Aït-Sadoune, M. Baron et Jean-Marc Mota
Adresse : LISI / ENSMA - Téléport 2 - 1, avenue Clément Ader - B.P. 40109 - 86960 Futuroscope Cedex - France
Emails : {yamine, idir.aitsadoune, baron, mota}@ensma.fr





LSR

ONERA

Table des matières

Table des figures iv

1 Introduction 1
- 1.1 Introduction . . . 1
- 1.2 Démarche de modélisation . . . 1

2 Développement d'IHM3 fondé sur la preuve et le raffinement avec CTT 3
- 2.1 Introduction . . . 3
- 2.2 Utilisation des modèles de tâches . . . 3
  - 2.2.1 Modèle de tâches pour la spécification . . . 4
  - 2.2.2 Modèle de tâches pour la validation . . . 4
  - 2.2.3 Systèmes de transitions et tâches . . . 5
- 2.3 Application à l'étude de cas "Matis" . . . 5
  - 2.3.1 Modèle de tâches pour la spécification . . . 5
  - 2.3.2 Modèle de tâches pour la validation . . . 8
- 2.4 Application à l'étude de cas "Pages Jaunes" . . . 11
  - 2.4.1 Vérification de propriétés CARE par construction . . . 18
  - 2.4.2 Validation de tâches utilisateurs . . . 19
  - 2.4.3 Bilan : preuves de propriétés . . . 20
- 2.5 Conclusion . . . 20

3 Approche fondée sur les composants 21
- 3.1 Introduction . . . 21
- 3.2 Modèle de tâches, modèle de composant . . . 21
- 3.3 Les composants ICARE en B . . . 22
  - 3.3.1 Les composants ICARE . . . 22
  - 3.3.2 Un modèle B pour les composants ICARE . . . 24
  - 3.3.3 Exemple d'application à une tâche utilisateur . . . 29

3.4 Conclusion 29

4 Bilan **31**

4.1 Résultats obtenus 31

4.2 Insuffisances et perspectives 31

Bibliographie **33**

Table des figures

2.1 Machine abstraite de MATIS 6
2.2 Raffinement 1 de MATIS 7
2.3 Raffinement 2 de MATIS 8
2.4 Raffinement 3 de MATIS 9
2.5 Raffinement 1 de du modèle de tâche à valider sur MATIS 10

3.1 Le modèle d'architecture logiciel ARCH et Les composants ICARE dans ARCH 23

Chapitre 1

Introduction

1.1 Introduction

En pratique, les concepteurs d'IHM3 utilisent différents modèles et notations propres à ce types de logiciels. Nous avons déjà montré, dans le SP3-LOT 2, comment les architectures logicielles définies pour les IHM3 ainsi que les propriétés d'utilisabilité peuvent être prises en considération dans les développements par raffinement en B.

Une autre notion importante lors de la conception d'IHM3 est celle de tâche utilisateur. Ces tâches permettent de décrire les différents chemins d'interaction permettant d'atteindre un objectif particulier. Elles sont utilisées par les ergonomes pour mesurer l'utilisabilité d'une IHM3. Jusqu'à présent nous n'avons pas pris en compte cette notion dans les développements avec B. Nous nous proposons de traiter ce point dans ce livrable.

De plus, les développements fondés sur le raffinement souffrent des problèmes inhérents aux aux modèles descendants. En particulier la réutilisation est compromise alors qu'elle une pratique courante dans le développement d'IHM. Nous traitons ce problème en proposant une démarche méthodologique combinant modèle de tâches et composants réutilisables pré-définis de la plate-forme ICARE.

1.2 Démarche de modélisation

Les modèles de tâches que nous avons considérés sont décrits par la notation CTT sous-forme d'expressions d'une algèbre de processus. Nous représentons la décomposition en sous-taches utilisée par CTT en utilisant le raffinement. Cette démarche permet la prise en compte des modèles de tâches dans les développements

en B événementiel. De plus, nous montrons que les tâches utilisateur sont prises en considération aussi bien pour la conception que pour la validation. Ce travail sera présenté dans le chapitre 2.

Le développement descendant par décomposition de tâches en sous-tâches, suit une démarche descendante. A ce niveau, se pose le problème de determiner à quel niveau de décomposition le modèle de tâches s'arrête-t-il. Une réponse fondée sur l'utilisation de composants logiciels d'interaction pré-définis au niveau des feuilles de l'arbre de tâches est apportée. Ces composants représentent les différentes combinaisons de modalités respectant les critères imposés par les propriétés CARE. Ce point est traité dans le chapitre 3.

Chapitre 2

Développement d'IHM3 fondé sur la preuve et le raffinement avec CTT

2.1 Introduction

La conception d'une IHM en général et d'une IHM3 en particulier fait appel en particulier à des descriptions de tâches permettant de décrire les actions et ou tâches qu'un utilisateur est sensé réaliser en utilisant cette IHM. La conception prenant en compte ces tâches est dite conception centrée utilisateur.

Différentes notations et modèles permettant de représenter ces tâches ont été définis. Ils souffrent de l'absence de description du système d'une part et d'un manque de sémantique formelle qui en fait des notations ambiguës d'autre part. Un panorama de ces modèles de tâches a été établi dans le lot 1 du SP3.

Ce chapitre décrit une contribution à la prise en compte des notations de modélisation des tâches utilisateurs dans la conception d'une IHM3. Nous montrons comment une formalisation du modèle de tâches conduit à une conception formelle de l'IHM3 et également à la validation formelle de tâches utilisateurs sur un système déjà développé.

2.2 Utilisation des modèles de tâches

Nous avons exploité les modèles de tâches pour la conception d'une IHM3. Notre choix s'est porté sur CTT [Pat01] pour les raisons suivantes :

- CTT est un modèle de tâches fondé sur la décomposition par une approche descendante ;

- CTT permet d'exprimer précisément les différentes actions qui sont réalisées en utilisant des expressions d'une algèbre de processus ;
- nous déjà effectué des travaux sur la modélisation d'arbres de tâches CTT en B dans le cas des interfaces de type WIMP.

Les modèles de tâches sont utilisés dans le développement à des fins de conception et/ou de validation.

2.2.1 Modèle de tâches pour la spécification

Une IHM3 peut être spécifiée par la composition de tâches élémentaires grâce à l'opérateur de choix présent dans CTT. De cette façon, une interface peut être spécifiée complètement par la donnée de toutes les tâches possibles sous forme d'une expression CTT. Ainsi, une spécification se présentera sous la forme $T = T_1[]T_2 \ldots []T_n$ où les T_i sont les tâches élémentaires.

Le développement d'une interface se fera en codant l'expression de tâches par des raffinements B en utilisant les schémas d" raffinement des opérateurs présentés dans le SP3-LOT 1. Les événements apparaissant dans les feuilles de l'arbre de tâche constituent les événements de base utilisés dans la conception du contrôleur de dialogue.

2.2.2 Modèle de tâches pour la validation

A l'image de la spécification, on peut utiliser le codage des modèles de tâches CTT pour effectuer de la validation. Le principe est le suivant.

Une IHM3 est décrite par un ensemble d'événements décrivant les différentes interactions possibles. Ils peuvent avoir été produits par un raffinement d'une tâche CTT (comme précédemment) ou bien par un développement ne prenant pas en compte de modèle de tâches.

La validation d'une tâche consiste à produire un développement de cette tâche par l'une des deux approches suivantes :

- la tâche est raffinée selon une décomposition CTT qui atteint les événements de base décrivant le système i.e. les événements des feuilles de l'arbre sont des événements décrivant le système. Dans ce cas, ce développement devra respecter les propriétés du système de l'invariant et des assertions ;
- la tâche peut être décrite comme un raffinement du système. Dans ce cas, la tâche est obtenue en pinçant les gardes des événements du systèmes pour qu'ils ne permettent les déclenchement des seuls événements décrivant cette tâche.

2.2.3 Systèmes de transitions et tâches

Du point de vue des systèmes de transitions, une tâche CTT représente un système de transitions codé en B événementiel. Cette tâche peut représenter soit le système dans le cas d'une spécification par une tâche, soit représenter une tâche à valider. Les modèles B événementiel associés à ces deux tâches décrivent des systèmes de transitions qui sont reliés par une relation de simulation. En effet, lorsque la tâche est validé, alors le système de transitions de l'une est contenu dans l'autre.

Dans la suite, nous présentons la mise en œuvre de cette approche sur les deux études de cas définies.

2.3 Application à l'étude de cas "Matis"

Dans un premier nous proposons un modèle de tâche pour la spécification de l'application MATIS. Par la suite, nous utilisons un modèle de tâche pour valider une tâche utilisateur sur le modèle de tâche de la spécification [AS05].

2.3.1 Modèle de tâches pour la spécification

Le modèle de tâche proposé pour l'application MATIS consiste à lancer une requête pour rechercher des vols qui correspondent aux paramètres qui sont : la ville de départ, la ville d'arrivée, l'heure de départ et l'heure d'arrivée de ces vols. Les paramètres peuvent être saisis dans un ordre quelconque et on offre la possibilité de modifier les différents paramètres avant de lancer la requête.

La tâche CTT qui décrit le modèle à construire est la suivante :

$(Evt_{InputDeparture}* \mid\mid Evt_{InputArrival}* \mid\mid Evt_{HourDeparture}* \mid\mid Evt_{HourArrival}*)$
$>> QueryReady >> QuerySending$

La machine abstraite

Le modèle abstrait se compose de deux événements principaux qui sont $QueryReady$ et $QuerySending$ dont le rôle et de préparer la requête puis de récupérer les résultats (figure 2.1). Les deux événements sont synchronisés par les deux variables $queryReady$ et $querySending$ qui ont un comportement booléen pour indiquer que la requête est prête à être lancée ($querySending := 1$) et que les résultats sont prêts ($queryReady := 1$).

```
VARIABLES
    queryReady, querySending, ...
INVARIANT
    queryReady ∈ {0,1} ∧ querySending ∈ {0,1} ∧ ...
    queryReady ≠ querySending
...
EVENTS
...
    QueryReady = SELECT
        queryReady = 1∧
        ...
        THEN
        queryReady := 0 ‖ ...
        querySending := 1 ‖ ...
        ...
        END ;

    QuerySending = SELECT
        querySending = 1
        THEN
        queryReady := 1 ‖
        querySending := 0 ‖
        ...
        END
```

FIG. 2.1 – Machine abstraite de MATIS

Raffinement 1

Dans le premier raffinement, l'événement *QueryReady* est raffiné en quatre événements décrivant la tâche de récupération des paramètres de la requête. Nous introduisons à ce niveau quatre nouveaux événements : $Evt_{InputDeparture}$, $Evt_{InputArrival}$, $Evt_{HourDeparture}$ et $Evt_{HourArrival}$ (figure 2.2). Ces quatre événements représentent les événements d'interaction avec l'utilisateur et peuvent être déclenchés dans un ordre quelconque. Les valeurs récupérées des paramètres sont sauvegardées dans les variables *PlaceDeparture* et *PlaceArrival* pour les noms de villes concernées par le vol, et les variables *HourDeparture* et *HourArrival* pour l'heure de départ et de l'arrivée du vol recherché. Les variables $Evt1, Evt2, Evt3etEvt4$ jouent le rôle de variant pour les événements d'interaction.

Raffinement 2

Ce nouveau raffinement détaille les événements de la récupération des paramètres $Evt_{InputDeparture}$, $Evt_{InputArrival}$, $Evt_{HourDeparture}$, $Evt_{HourArrival}$. La caractéristique à modéliser définit la possibilité de modifier les paramètres plusieurs fois avant de lancer la requête. Pour cela, nous utilisons l'opérateur de boucle ($*$) de

VARIABLES
$PlaceDeparture, PlaceArrival, HourDeparture, HourArrival,$
$Evt1, Evt2, Evt3, Evt4,$
$queryReady, querySending, ...$
INVARIANT
$PlaceDeparture \in 1..9 \wedge PlaceArrival \in 1..9 \wedge$
$HourDeparture \in 0..23 \wedge HourArrival \in 0..23 \wedge$
$Evt1 \in \{0,1\} \wedge Evt2 \in \{0,1\} \wedge Evt3 \in \{0,1\} \wedge Evt4 \in \{0,1\} \wedge$
...
EVENTS
$Evt_{InputDeparture} =$ **SELECT**
$Evt1 = 1$
THEN
$PlaceDeparture :\in 1..9 \parallel Evt1 := 0$
END ;
$Evt_{InputArrival} =$ **SELECT**
$Evt2 = 1$
THEN
$PlaceArrival :\in 1..9 \parallel Evt2 := 0$
END ;
$Evt_{HourDeparture} =$ **SELECT**
$Evt3 = 1$
THEN
$HourDeparture :\in 0..23 \parallel Evt3 := 0$
END ;
$Evt_{HourArrival} =$ **SELECT**
$Evt4 = 1$
THEN
$HourArrival :\in 0..23 \parallel Evt4 := 0$
END ;

Fig. 2.2 – Raffinement 1 de MATIS

CTT. Nous prenons l'exemple de l'événement $Evt_{InputDeparture}$. Ce dernier est raffiné en $InitLoop_{InputDeparture}$ dont le rôle est de préciser le nombre de modifications à faire sur la variable $PlaceDeparture_{ref}$ et $modify_{InputDeparture}$ qui se déclenche à chaque modification de ce paramètre. L'événement $Evt_{InputDeparture}$ de l'abstraction récupère la dernière valeur de $PlaceDeparture_{ref}$ (figure 2.3).

```
...
EVENTS
  InitLoop_InputDeparture = SELECT
    startLoop = 1
    THEN
    index1 :∈ NAT || startLoop := 0
    END ;
  modify_InputDeparture = SELECT
    Evt1 = 1 ∧ index1 > 0
    THEN
    index1 := index1 − 1 || PlaceDeparture_ref :∈ 1..9
    END ;
  Evt_InputDeparture = SELECT
    Evt1 = 1 ∧ index = 0
    THEN
    PlaceDeparture := PlaceDeparture_ref || Evt1 := 0
    END ;
```

FIG. 2.3 – Raffinement 2 de MATIS

Raffinement 3

Dans ce raffinement (figure 2.4), Les événements $modify_{InputDeparture}$, $modify_{InputArrival}$, $modify_{HourDeparture}$ et $modify_{HourArrival}$ sont raffinés pour faire apparaître les modalités d'interaction de l'application avec l'utilisateur. Nous considérons l'existence de deux modalités, la voix et la manipulation directe (clavier et souris). Pour exemple, $modify_{InputDeparture}$ est raffiné par deux nouveaux événements $modifyByVoice_{InputDeparture}$ (pour la voix) et $modifyByDirectManipulation_{InputDeparture}$ (pour la manipulation directe).

2.3.2 Modèle de tâches pour la validation

Le tâche utilisateur à valider consiste à exécuter une requête dont les paramètres sont introduits dans un ordre précis. Ce scénario est valide car le modèle du système réalisé correspond à une tâche utilisateur globale où les paramètres peuvent être introduits dans un ordre quelconque.

La tâche CTT à valider est :

...
EVENTS
$modifyByVoice_{InputDeparture}$ = **SELECT**
$Evt1 = 1 \wedge index1 > 0$
THEN
$PlaceDeparture_{ref2} :\in 1..9$
END ;
$modifyByDirectManipulation_{InputDeparture}$ = **SELECT**
$Evt1 = 1 \wedge index1 > 0$
THEN
$PlaceDeparture_{ref2} :\in 1..9$
END ;
$modify_{InputDeparture}$ = **SELECT**
$Evt1 = 1 \wedge index1 > 0$
THEN
$index1 := index1 - 1 \parallel PlaceDeparture_{ref} := PlaceDeparture_{ref2}$
END ;

FIG. 2.4 – Raffinement 3 de MATIS

$Evt_{InputDeparture} >> Evt_{InputArrival} >> Evt_{HourDeparture} >> Evt_{HourArrival}$
$>> QueryReady >> QuerySending$

Le modèle obtenu après traduction des notations CTT vers un modèle B se compose de trois niveaux : une machine abstraite et deux raffinements. La machine abstraite reste la même que celle obtenue dans le modèle développé pour le système, par contre, le premier raffinement (figure 2.5) contient quelques différences.

La variable *state*, définie dans l'invariant comme un entier dont la valeur varie entre 0 et 3, garantit que les événements $Evt_{InputDeparture}$, $Evt_{InputArrival}$, $Evt_{HourDeparture}$ et $Evt_{HourArrival}$ se déroulent en séquence dans l'ordre définie dans la tâche CTT.

A l'issue du dernier raffinement, nous retrouvons tous les événements et ils respectent tous les invariants définis dans le modèle initial, ce qui nous permet de conclure que cette cette tâche est supportée par le modèle du système.

Bilan : preuves de propriétés

Le Tableau 2.1 résume les résultats obtenus en termes de nombre d'obligations de preuve par chaque modèle. La nature des modèles qui se composent essentiellement de structures simple en terme de type (typage simple des variables utilisées), ont fait que toutes les obligations de preuves générées ont été prouvées automatiquement sans aucune intervention du concepteur.

```
...
INVARIANT
...
    state ∈ {0, 1, 2, 3} ∧
...
EVENTS
    Evt_InputDeparture = SELECT
        Evt1 = 1 ∧ state = 3
        THEN
        PlaceDeparture :∈ 1..9 || Evt1 := 0
        || state := state − 1
        END ;
    Evt_InputArrival = SELECT
        Evt2 = 1 ∧ state = 2
        THEN
        PlaceArrival :∈ 1..9 || Evt2 := 0
        || state := state − 1
        END ;
    Evt_HourDeparture = SELECT
        Evt3 = 1 ∧ state = 1
        THEN
        HourDeparture :∈ 0..23 || Evt3 := 0
        || state := state − 1
        END ;
    Evt_HourArrival = SELECT
        Evt4 = 1 ∧ state = 0
        THEN
        HourArrival :∈ 0..23 || Evt4 := 0
        || state := state − 1
        END ;
```

Fig. 2.5 – Raffinement 1 de du modèle de tâche à valider sur MATIS

Modèle	Obv	nOP	Auto	Interactif	%Pr
$MATIS_CTT$ (Tâche de spécification)	4406	117	117	0	100
$MATIS_CTT_User$ (Tâche de validation)	561	114	114	0	100

Tab. 2.1 – Résultat du nombre de preuves générées

2.4 Application à l'étude de cas "Pages Jaunes"

Quatre modèles ont été définis suivant le scénario 4 [AAASBM06, AAASB06] et chacun raffine le précédent. Le premier introduit les événements de haut niveau. Le deuxième présente les événements de la présentation ainsi que leurs synchronisations avec le *CD*. Le troisième introduit les interactions multi-modales avec les différentes possibilités d'interaction. Enfin, le quatrième introduit concrètement les modalités et décompose les interactions multi-modales en événements d'interaction atomiques de base. Ces modèles font tous abstraction du noyau fonctionnel et ne font que rendre compte du fait que des événements abstraits du noyau fonctionnel ont été déclenchés.

Dans la suite, nous décrivons une représentation simplifiée du développement de l'IHM3 associée à l'étude de cas "Pages Jaunes CLIPS". Nous avons volontairement réduit les modèles B événementiel.

Le modèle racine

Le modèle racine est un modèle abstrait représentant la synchronisation entre les événements de saisie et de déclenchement du noyau fonctionnel.

```
MODEL IMAPPY
VARIABLES
  Query_sending, Query_ready,
INVARIANT
  Query_sending ∈ 0..1 ∧ Query_ready ∈ 0..1 ∧
  Query_sending ≠ Query_ready
ASSERTIONS
  (Query_ready = 1) ∨ (Query_sending = 1)
INITIALISATION
  Query_sending, Query_ready := 0, 1
EVENTS
Query = SELECT
    Query_ready = 1
  THEN
    Query_sending := 1 || Query_ready := 0
  END;

Result_query = SELECT
    Query_sending = 1
  THEN
    Query_sending := 0 || Query_ready := 1
  END
END
```

Deux événements abstraits (sans modalités) *Query* et *Result_query* décrivent ce modèle. Le premier indique ($Query_sending := 1$) que la requête est prête à être

envoyée au noyau fonctionnel alors que le second indique que le résultat de la requête est prêt ($Query_ready := 1$). Les deux variables $Query_sending$ et $Query_ready$ décrivent la synchronisation de ces deux événements.

La suite s'intéresse au raffinement de l'événement $Query$ qui décrit la saisie du nom et de l'adresse selon différentes modalités et qui déclenchent les événements du noyau fonctionnel.

Identification des interactions abstraites

L'événement $Query$ est décomposé de sorte à prendre en compte les différentes possibilités de saisie du nom et de l'adresse de la personne à rechercher. Ainsi, on pourra :

- saisir le nom un nombre arbitraire de fois $Input_Name^*$;
- saisir l'adresse un nombre arbitraire de fois $Input_Address^*$;
- lancer ensuite la recherche $Search$.

Notons que l'on peut entrelacer la saisie du nom et de l'adresse (saisir l'un puis l'autre dans n'importe quel ordre et retour). Ainsi la tâche CTT (ConcurTaskTrees) [Pat01] qui décrit les différentes possibilités de saisie est définie par :

$$Query = (Input_Name^* || Input_Address^*) >> Search >> Result_query$$

où * désigne l'itération et $>>$ désigne l'opérateur de séquence. La tâche $Query$ est implantée dans le raffinement $IMAPPY_Ref_1$ décrit ci-dessous. Nous nous concentrons sur les événements introduits pour raffiner les deux événements principaux du niveau abstrait.

REFINEMENT $IMAPPY_Ref_1$
REFINES $IMAPPY$
SETS
  $string$
VARIABLES
  $Query_sending, EV1, Nn, Na, map, name, address$
INVARIANT
  $EV1 \in 0..3 \wedge Nn \in \mathbb{N} \wedge Na \in \mathbb{N} \wedge$
  $map \in BOOL \wedge name \subseteq string \wedge address \subseteq string \wedge$
  $(Query_ready = 1 \Leftrightarrow EV1 \neq 0)$
ASSERTIONS ...
INITIALISATION
  $Query_sending := 0 \parallel map, EV1 := FALSE, 3 \parallel$
  $name, address, Nn, Na := \emptyset, \emptyset, 1, 1$

Nous avons introduit de nouvelles variables pour la description de l'interaction en entrée.

Tout d'abord, nous décrivons les variables *name* et *address* de type *string* qui récupérent respectivement le nom et l'adresse. Les variables entières Nn et Na désignent les nombres arbitraires d'itérations * de CTT. Elles sont initialisées, dans l'événement $Name_Address$, grâce à l'opérateur $:\in$ qui considère un élément quelconque dans un ensemble et indiquent le nombre de fois que cet événement est déclenché. Enfin, la variable $EV1$ (variable entière décroissante) initialisée à 3 (nombre d'événement à déclencher) est un variant qui décrit l'ordre de déclenchement de chaque événement de l'abstraction.

```
EVENTS
Name_Address = SELECT
    EV1 = 3
  THEN
    EV1 := EV1 − 1 || Nn :∈ ℕ || Na :∈ ℕ
  END ;

Input_Name = SELECT
    EV1 = 2 ∧ Nn ≠ 0
  THEN
    Nn := Nn − 1 || name :∈ ℙ(string)
  END ;

Input_Address = SELECT
    EV1 = 2 ∧ Na ≠ 0
  THEN
    Na := Na − 1 || address :∈ ℙ(string)
  END ;

Search = SELECT
    EV1 = 2 ∧ Nn = 0 ∧ Na = 0
  THEN
    EV1 := 1
  END ;

Query = SELECT
    EV1 = 1
  THEN
    Query_sending, EV1 := 1, 0
  END ;

Result_query = ...

END
```

Les événements ci-dessus décrivent la tâche CTT précédente. L'événement $Name_Address$ initialise itérateurs Nn et Na alors que les trois événements $Input_Name$, $Input_Address$ et $Search$ décomposent (raffinent) l'événement abstrait $Query$. Lorsque ces trois évé-

nements ont été déclenchés, ils rendent le contrôle à l'événement abstrait *Query* qui ne fait que le transmettre aux autres événements.

Notons qu'à ce niveau, nous avons pris en compte les déclenchements des interactions sans entrer dans les détails de leurs définitions abordées dans les prochains raffinements.

Comme nous l'avions déclaré précédemment, nous ne nous intéressons pas à la multi-modalité en sortie ni au noyau fonctionnel. C'est pour cette raison que les événements associés (*Result_Query*) n'ont pas été raffinés. Cette démarche montre qu'il est possible de développer les différents aspects d'une IHM3 de façon modulaire afin de simplifier le processus de preuve. En effet, les obligations de preuve engendrées sont plus simples du fait que certains événements restent abstraits.

Identification des modalités

L'étape suivante consiste à identifier et à introduire les différentes interactions multi-modales en entrée mises en jeu. Une nouvelle variable booléenne *Lock* indique que le focus est sur la saisie du nom.

REFINEMENT $IMAPPY_ref2$
REFINES $IMAPPY_ref1$
VARIABLES
  $lock, \ldots$
INVARIANT
  $lock \in BOOL \wedge \ldots$
ASSERTIONS
  $\ldots$
INITIALISATION
  $lock := TRUE \parallel \ldots$

Nous procédons à la décomposition (raffinement) des événements $Input_Name$ et $Input_Address$ faisant intervenir les modalités voix (V) et clavier/souris (CS) sans introduire explicitement les modalités, ainsi que leurs caractéristiques. Ceci est un autre aspect de la modularité puisqu'il est possible de valider l'entrelacement des événements sans s'intéresser à ce qu'ils font explicitement. Pour cela, il est nécessaire d'enrichir le modèle B par la déclaration de nouveaux événements.

Dans la suite, seuls les événements raffinant l'événement $Input_Name$ sont présentés selon la tâche de décomposition suivante :

$$Input_name^* = (Input_Name_V_V [] Input_Name_V_CS [] \\ Input_Name_CS_V [] Input_Name_CS_CS)^*$$

où [] désigne l'opérateur de choix. Le nom est entré en deux étapes : saisie du champ à saisir (le nom) par la voix ou le clavier/souris puis entrée du nom (valeur

du nom) par la voix ou bien par clavier/souris. Cela donne quatre possibilités. Ici on peut saisir le nom par la V puis V, ou bien par V puis CS ou bien par CS puis V ou bien par CS puis CS. Tous ces événements peuvent être itérés un nombre arbitraire de fois.

```
EVENTS
Name_Address = ...

Input_Name_V_V = SELECT
    EV1 = 2 ∧ Nn ≠ 0 ∧ lock = TRUE
  THEN
    Nn := Nn − 1 || name :∈ ℙ(string) || lock := FALSE
  END;

Input_Name_V_CS = SELECT
    EV1 = 2 ∧ Nn ≠ 0 ∧ lock = TRUE
  THEN
    Nn := Nn − 1 || name :∈ ℙ(string) ||
    lock := FALSE
  END;

Input_Name_CS_CS = SELECT
    EV1 = 2 ∧ Nn ≠ 0 ∧ lock = TRUE
  THEN
    Nn := Nn − 1 || name :∈ ℙ(string) ||
    lock := FALSE
  END;

Input_Name_CS_V = SELECT
    EV1 = 2 ∧ Nn ≠ 0 ∧ lock = TRUE
  THEN
    Nn := Nn − 1 || name :∈ ℙ(string) ||
    lock := FALSE
  END;

Input_Name = SELECT
    EV1 = 2 ∧ Nn ≠ 0 ∧ lock = FALSE
  THEN
    lock := TRUE
  END;

Input_Address = ...

Search = ...

Query = ...

Result_query = ...
END
```

Nous avons introduit de nouveaux événements, utilisant la multi-modalité, sans la décrire concrètement, qui représentent les différentes combinaisons permettant de saisir le nom.

À ce niveau, il faut observer que tous les événements peuvent être déclenchés. C'est-à-dire, qu'il est possible de saisir le nom par la voix ou par le clavier/souris uniquement, ou bien par une combinaison des deux. On peut restreindre ces choix en fonction des propriétés CARE que l'on souhaite établir.

Identification des interactions concrètes

Le dernier raffinement présenté introduit explicitement les modalités utilisées ainsi que leurs valeurs (au sens de la modélisation B). Ainsi, on décrit les ensembles *SPEECH*, *KEYBOARD* et *MOUSE* indiquant et typant les différentes interactions multi-modales possibles. Notons que ces ensembles peuvent être enrichis ou réduits sans remettre en cause la modélisation effectuée, ni le processus de preuve.

```
REFINEMENT IMAPPY_ref3
REFINES IMAPPY_ref2
SETS
  HCI = {NONE, NAME, INPUT, SEARCH,
    ZOOM_IN, INPUT_TEXT, ENTER_KEY,
    LEFT_ARROW, CLICK_NAME,
    CLICK_SEARCH, ...}
CONSTANTS
  SPEECH, KEYBOARD, MOUSE
PROPERTIES
  SPEECH ∪ KEYBOARD ∪ MOUSE ⊆ HCI ∧
  SPEECH = {NAME, INPUT, SEARCH, ...}
  MOUSE = {CLICK_SEARCH, CLICK_NAME, ...}
  KEYBOARD = {INPUT_TEXT, ENTER_KEY, ...}
VARIABLES
  Query_sending, map, EV1,
  name, adress, Nn, Na,
  modality, lock,
  V_speech_speech, V_speech_keyboard,
  V_mouse_speech, V_mouse_keyboard
```

```
INVARIANT
  name ⊆ string ∧
  V_speech_speech ∈ 0..3 ∧
  V_mouse_speech ∈ 0..3 ∧
  V_mouse_keyboard ∈ 0..3 ∧
  V_speech_keyboard ∈ 0..3 ∧
  modality ∈ HCI ∧ ...
ASSERTIONS
  ...
INITIALISATION
  Query_sending := 0 || map := FALSE || EV1 := 3 ||
  name, adress := ∅, ∅ || Nn, Na := 1, 1 ||
  modality := NONE || lock := TRUE ||
  V_speech_speech, V_speech_keyboard := 2, 2 ||
  V_mouse_speech, V_mouse_keyboard := 2, 2
```

Les variables V_x_y, où x et y sont des modalités, définissent le type d'interaction en cours de déclenchement dans le modèle. La variable *modality* décrit la modalité activée à un instant donné. Elle est initialisée à $NONE$.

Les événements du raffinement précédent sont décomposés (par raffinement) en :

$$Input_Name_V_V = Input_Name_V_V_field >> Input_Name_V_V_value$$

$$Input_Name_V_CS = Input_Name_V_CS_field >> Input_Name_V_CS_value$$

$$Input_Name_CS_V = Input_Name_CS_V_field >> Input_Name_CS_V_value$$

$$Input_Name_CS_CS = Input_Name_CS_CS_field >> Input_Name_CS_CS_value$$

Les suffixes $field$ et $value$ indiquent respectivement les événements associés au choix du champ à remplir et à la valeur entrée pour ce champ. On obtient :

```
EVENTS
Name_Address = SELECT
    EV1 = 3
  THEN
    ...
  END ;

Input_Name_V_V_field = SELECT
    EV1 = 2 ∧ Nn ≠ 0 ∧ lock = TRUE ∧
    V_speech_speech = 2
  THEN
    V_speech_speech := V_speech_speech − 1 ||
    modality := NAME
  END ;

Input_Name_V_V_value = SELECT
    EV1 = 2 ∧ Nn ≠ 0 ∧ lock = TRUE ∧
    V_speech_speech = 1
  THEN
    V_speech_speech := V_speech_speech − 1 ||
    name :∈ ℙ(string) ||
    modality := INPUT
  END ;

Input_Name_V_V = SELECT
    EV1 = 2 ∧ Nn ≠ 0 ∧ lock = TRUE ∧
    V_speech_speech = 0
  THEN
    Nn := Nn − 1 || lock := FALSE
  END ;

Input_Name_V_CS_field = SELECT
    EV1 = 2 ∧ Nn ≠ 0 ∧ lock = TRUE ∧
    V_speech_keyboard = 2
  THEN
    V_speech_keyboard := V_speech_keyboard − 1 ||
    modality := NAME
  END ;
```

$Input_Name_V_CS_value =$ **SELECT**
$EV1 = 2 \wedge Nn \neq 0 \wedge lock = TRUE \wedge$
$V_speech_keyboard = 1$
THEN
$V_speech_keyboard := V_speech_keyboard - 1 \parallel$
$name :\in \mathbb{P}(string) \parallel$
$modality := INPUT_TEXT$
END ;

$Input_Name_V_CS =$ **SELECT**
$EV1 = 2 \wedge Nn \neq 0 \wedge lock = TRUE \wedge$
$V_speech_keyboard = 0$
THEN
$Nn := Nn - 1 \parallel lock := FALSE$
END ;

$Input_Name_CS_V_field = \ldots$

$Input_Name_CS_V_value = \ldots$

$Input_Name_CS_V = \ldots$

$Input_Name_CS_CS_field = \ldots$

$Input_Name_CS_CS_value = \ldots$

$Input_Name_CS_CS = \ldots$

$Input_Name =$ **SELECT**
$EV1 = 2 \wedge Nn \neq 0 \wedge lock = FALSE$
THEN
$lock = TRUE \parallel modality = NONE$
END ;

$Input_Adress = \ldots$

$Search =$ **SELECT**
$EV1 = 2 \wedge Nn = 0 \wedge Na = 0$
THEN
$EV1 := 1 \parallel$
$modality :\in \{CLICK_SEARCH,$
$ENTER_KEY, SEARCH\}$
END ;

$Query = \ldots$

$Result_query = \ldots$
END

2.4.1 Vérification de propriétés CARE par construction

Le modèle B événementiel $IMAPPY_ref2$ offrait toutes les combinaisons d'interactions possibles. Il permettait ainsi le codage de l' équivalence de modalités avec la voix exclusivement (événement $Input_Name_V_V$) ou bien avec le cla-

vier/souris exclusivement (événement $Input_Name_CS_CS$). Mais, ce modèle permettait également le codage de la complémentarité de la voix et du clavier/souris avec les événements ($Input_Name_V_CS$ et $Input_Name_CS_V$).

La représentation d'une interaction multi-modale satisfaisant d'autres types de propriétés CARE est possible par le codage d'une autre tâche CTT. Cette tâche doit décrire le type de propriété que l'on souhaite représenter. Par exemple,

- une équivalence de modalités entre clavier/souris et voix impliquerait l'implantation de la tâche :

$$(Input_Name_CS_CS [] \ Input_Name_V_V)^* || \ldots$$

- la redondance impliquerait l'implantation de la tâche :

$$(Input_Name_V_V \ || \ \ Input_Name_CS_CS)^* || \ldots$$

- la complémentarité impliquerait l'implantation de la tâche :

$$Input_Name_V_CS \ [] \ Input_Name_CS_V$$

Le raisonnement effectué ici s'applique également au dernier raffinement $IMAPPY_ref3$. En effet, nous avons choisi d'identifier le champ puis de saisir sa valeur selon une tâche de la forme :

$$Input_Name_x_y = Input_Name_x_y_field >> Input_Name_x_y_value$$

D'autres choix auraient pu être décrits, comme :

$$Input_Name_x_y = Input_Name_x_y_field \ || \ Input_Name_x_y_value$$

pour décrire entre le choix du champ et la saisie.

2.4.2 Validation de tâches utilisateurs

La démarche développée est fondée sur la notion de tâche. Nous avons représenté la totalité de l'IHM3 par des systèmes de transitions. Nous avons veillé à ne pas représenter le système de transitions explicitement, mais nous nous sommes appuyés sur la définition de tâches utilisateurs en prenant pour langage de tâches, la notation CTT. Nous avons montré que le modèle de tâches peut être représenté par un modèle B et que le modèle de tâches dirige la définition des modèles B. Enfin, il faut noter que les tâches implantées en B événementiel permettent de valider des propriétés CARE grâce à la construction des modèles B événementiel.

2.4.3 Bilan : preuves de propriétés

Le tableau ci-dessous illustre les résultats obtenus sur l'étude de cas. 219 obligations de preuve sont automatiquement générées. Seules deux d'entre elles ne sont pas prouvées automatiquement. La première dans le raffinement $IMAPPY_ref2$ consiste à prouver un lemme. La seconde dans $IMAPPY_ref3$, est une preuve par cas liée à la disjonction des gardes des événements (pour prouver l'absence de blocage). Elle consiste à prouver les différents cas obtenus dans cette disjonction en appelant le prouveur qui les établit.

Modèle	Obv	nOP	Auto	Interactif	%Pr
$IMAPPY$	75	10	10	0	100
$IMAPPY_ref1$	360	22	22	0	100
$IMAPPY_ref2$	465	28	27	1	100
$IMAPPY_ref3$	557	159	158	1	100
Total	145	219	217	2	100

2.5 Conclusion

Le développements présentés dans ce chapitre nous permettent d'affirmer qu'il est possible de concevoir une IHM à partir d'un modèle de tâches avec B et de valider des tâches utilisateur sur un modèle de système en utilisant le raffinement de B. Ces travaux nous ont permis de mettre en pratique les schémas de représentation des opérateurs de CTT présentés dans le SP3-LOT 1 et surtout de donner une sémantique d'entrelacement (celle de B événementiel) aux opérateurs de CTT.

Par contre, les modèles de tâches sont valides par rapport au système conçu. On ne sait pas si ce que fait ce système est conforme à un besoin utilisateur. Il faudrait pouvoir tester ou animer les modèles B en vue d'en assurer la validité vis-à-vis des besoins utilisateurs. C'est ce que nous présenterons dans le lot 4.

Chapitre 3

Approche fondée sur les composants

3.1 Introduction

Les développements par raffinements successifs suivent une approche descendante. Cette approche permet de définir le système résultant dans le dernier raffinement par un ensemble d'événements. Il est difficile dans ce cas de cibler des événements déjà existants dans d'autres applications, en particulier dans le contexte d'une réutilisation qui est une pratique courante dans les IHM (avec les boîtes à outils).

Pour traiter ce problème, nous avons défini une approche mixte fondée sur l'utilisation de modèles de composants d'interaction pré-définis de la plate-forme ICARE définie au CLIPS/IMAG et des modèles de tâches.

Ce chapitre présente une démarche de développement descendante fondée sur la décomposition d'une tâche jusqu'à atteindre les événements associés à des composants pré-définis.

3.2 Modèle de tâches, modèle de composant

Le principe de décomposition d'une tâche est le même que celui qui a été décrit dans le chapitre précédent. Seulement, nous traitons en plus dans ce travail le problème de l'arrêt de la décomposition.

En d'autres termes, nous nous posons la question suivante : comment déterminer le dernier niveau de décomposition dans les arbres de tâches ?

En fait, la réponse à cette question consiste à utiliser la décomposition de tâche jusqu'à atteindre les événements de consommation des modalités. Ces événements

sont ceux des composants ICARE. Chaque composant ICARE (composants de Complémentarité, Assignation, Redondance, Equivalence) traite de la consommmation de modalités à partir de la définition de langage d'entrée et de langage de sortie. Ils sont codées en B de façon générique. L'intérêt de cette approche fondée sur les composants est la validation des propriétés CARE par construction.

Notons que pour ces développements nous avons été influencés par l'approche de développement utilisée au CLIPS-IMAG.

La suite de ce chapitre décrit la modélisation en B des composants ICARE et son utilisation dans les arbres de tâches.

3.3 Les composants ICARE en B

La plate-forme ICARE, pour Interaction-CARE (Complémentarité, Assignation, Redondance, Equivalence), permet aux concepteurs de manipuler graphiquement et d'assembler des composants logiciels [BN04b, BN04a] afin de spécifier l'interaction multi-modale pour une tâche donnée du système interactif. De cette spécification, le code de l'assemblage est automatiquement généré.

Comme le montre la figure (3.1), les composants ICARE s'intègrent au sein du modèle d'architecture ARCH et couvrent à la fois l'interaction logique et l'interaction physique.

3.3.1 Les composants ICARE

Nous identifions deux types de composants ICARE :

1. les composants élémentaires qui permettent au concepteur de définir les modalités ;
2. les composants de composition (ou de combinaison) qui permettent au concepteur de spécifier l'usage combiné de modalités. Les composants de composition sont indépendants des modalités à composer.

Le premier est basé sur la définition d'une modalité d'interaction comme étant une association d'un dispositif physique d avec un langage d'interaction L. Un composant dispositif représente le niveau physique d'une modalité (exemple de dispositif physique : le GPS). Un composant langage d'interaction représente le niveau logique d'une modalité d'interaction (Latitude, longitude et altitude). Un composant Dispositif peut être lié à un composant logique pour former une modalité pure (Latitude, longitude et altitude sont les données abstraites du composant GPS).

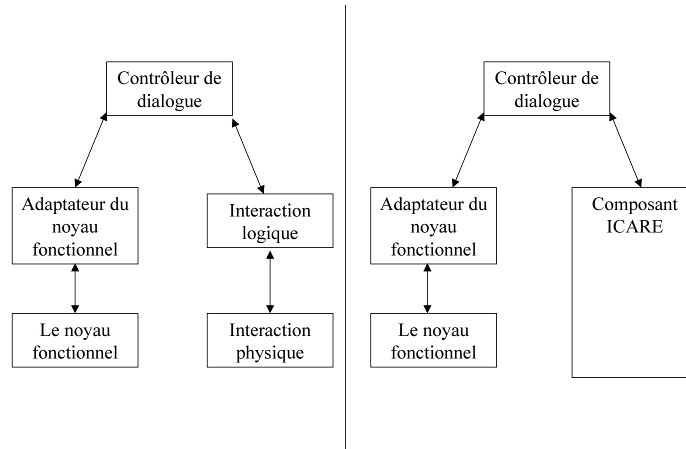


Fig. 3.1 – Le modèle d'architecture logiciel ARCH et Les composants ICARE dans ARCH

Le second permet de combiner les données obtenues des composants élémentaires ou des compositions entre eux sur la base des propriétés CARE. Il existe quatre composants : un pour la complémentarité, un pour la redondance, un pour l'équivalence et un pour la redondance/équivalence.

Leur comportement suit les règles suivantes :

1. à la réception d'une nouvelle donnée sur une entrée, un objet est créé avec la nouvelle donnée réceptionnée et une date de création. A ce moment, l'objet créé est considéré comme étant incomplet ;
2. s'il y a des objets incomplets et qui peuvent être fusionnés avec le dernier objet créé, et qui sont dans la même fenêtre temporelle, on duplique ce dernier en autant d'objets avec qui la fusion est possible et on les fusionne (la fusion revient à construire un objet qui a comme données celles récupérées des deux objets fusionnés et comme date de création celle de l'objet source la plus ancienne) ;
3. s'il y a un seul objet complet, il est envoyé en sortie du composant ICARE. S'il y a plusieurs objets, on renvoie en sortie celui qui a la date de création la plus récente ;
4. s'il reste des objets fusionnés, on supprime la dernière donnée réceptionnée au niveau de chaque objet et on applique à nouveau la règle 2 ;

5. les objets dont la durée de vie dépasse la longueur de la fenêtre temporelle sont détruits.

Les données manipulées par les composants ICARE sont des mots de langages de haut niveau (données logiques).

3.3.2 Un modèle B pour les composants ICARE

Dans le but d'intégrer les composants ICARE dans le processus de validation du contrôleur de dialogue des systèmes interactifs multi-modaux, nous développons un modèle B événementiel générique pour les composants ICARE avec une spécialisation vers un composant précis (*Complémentarité, Redondance, Redondance/Equivalence*) avec, soit un raffinement supplémentaire, soit par enrichissement de la clause **PROPERTIES**.

Le modèle B événementiel développé se compose d'une machine abstraite et de 3 raffinements :

- La machine abstraite : décrit le rôle d'un composant ICARE ;
- Le 1er raffinement : décrit le comportement d'un composant ICARE ;
- Le 2eme raffinement : décrit les contraintes de temps dans un composant ICARE ;
- Le 3eme raffinement : décrit la spécialisation d'un composant ICARE. Nous donnons l'exemple de la redondance et de complémentarité.

La machine abstraite

Dans le modèle abstrait d'un composant ICARE, nous définissons ce dernier comme étant une boite noire avec une sortie ($outPutICARE$) dont le type est abstrait nommé $LANGUAGE$. Ce dernier est un ensemble qui regroupe toutes les constructions des langages manipulés par les composants ICARE. L'événement $OutPutEvent$ définit la sortie d'un composant ICARE. Le code ci-dessous résume toutes les constructions définies plus haut.

```
...
SETS
    LANGUAGE
...
VARIABLES
    outPutICARE, ...
...
INVARIANT
    outPutICARE ∈ LANGUAGE
...
EVENTS
...
    OutPutEvent = ANY
        XX
    WHERE
        XX ∈ LANGUAGE
    THEN
        outPutICARE := XX
    END ;
```

Le 1er raffinement

Nous raffinons le modèle de la machine abstraite pour expliquer le comportement d'un composant ICARE. La boite noire décrite dans l'abstraction est une fonction totale F qui a comme entrées deux attributs $inPutICARE1$ et $inPutICARE2$ de types $LANGUAGE1$ et $LANGUAGE2$ qui sont des sous ensembles de $LANGUAGE$.

```
...
CONSTANTS
    LANGUAGE1, LANGUAGE2,
    F, ...
PROPERTIES
    LANGUAGE1 ⊆ LANGUAGE ∧ LANGUAGE1 ≠ ∅∧
    LANGUAGE2 ⊆ LANGUAGE ∧ LANGUAGE2 ≠ ∅∧
    F ∈ (LANGUAGE1 * LANGUAGE2) → LANGUAGE∧
...
VARIABLES
    inPutICARE1, inPutICARE2, ...
INVARIANT
    inPutICARE1 ⊆ LANGUAGE1∧
    inPutICARE2 ⊆ LANGUAGE2∧
    outPutICARE ∈ ran(F)
...
```

L'événement $InPutPort1Event$ se déclenche dès qu'une donnée est présente sur l'entrée 1 du composant ICARE, cette donnée est sauvegardée dans l'ensemble $inPutICARE1$. L'événement $InPutPort2Event$ accomplit la même mission au niveau de la deuxième entrée. Ainsi la sortie du composant qui est la variable $outPutICARE$, devient le résultat de la fonction F appliquée aux entrées du composant.

EVENTS
$InPutPort1Event =$ **ANY**
$x1$
WHERE
$x1 \in LANGUAGE1 \wedge ...$
THEN
$inPutICARE1 := inPutICARE1 \cup \{x1\} \parallel ...$
END ;

$InPutPort2Event =$ **ANY**
$x2$
WHERE
$x2 \in LANGUAGE2 \wedge ...$
THEN
$inPutICARE2 := inPutICARE2 \cup \{x2\} \parallel ...$
END ;

$OutPutEvent =$ **ANY**
$xx, x1, x2$
WHERE
$x1 \in inPutICARE1 \wedge$
$x2 \in inPutICARE2 \wedge$
$xx \in LANGUAGE \wedge xx = F(x1, x2) \wedge ...$ **THEN**
$outPutICARE := xx \parallel ...$
END ;

Le 2eme raffinement

Pour compléter le fonctionnement des composants ICARE, nous enrichissons le modèle précédent par des contraintes temporelles dans un raffinement supplémentaire. Il va nous servir à coller le modèle avec les règles de fonctionnement définies dans la section 3.3. Dans un premier temps, nous introduisons une constante qui représentera la valeur de la fenêtre temporelle ($temporalWindow$). Pour chaque donnée qui arrive à l'une des entrées du composant, nous lui associons une date d'arrivée que nous mémorisons dans l'ensemble $dateEvent$. La dernière date est sauvegardée dans la variable $lastDate$.

...
CONSTANTS
$temporalWindow$
PROPERTIES
$temporalWindiw \in NATURAL1$
...
VARIABLES
$lastDate, dateEvent, ...$
INVARIANT
$lastDate \in NATURAL1 \wedge$
$dateEvent \in LANGUAGE * NATURAL1 \wedge ...$
...

Les événements du modèle deviennent plus explicites. Les événements $InPutPort1Event$

et $InPutPort1Event$ reprennent les règles 1 et 5 de la section 3.3.1. Une date est associée à la nouvelle information qui arrive (visible sur la substitution appliquée à la variable $dateEvent$). Et les données dont la durée de vie est supérieure à la fenêtre temporelle sont supprimées (visible sur la substitution appliquée à les variables $inPutICARE1$ et $inPutICARE1$). L'événement $OutPutEvent$ applique les règles 2 et 3. La règle 3 est exprimée dans la garde de l'événement, par contre la règle 2 est dans la substitution appliquée à la variable $outPutICARE$.

La règle 4 est implicite, du fait que la règle 2 n'est appliquée que partiellement, c'est-à-dire nous ne fusionnons que les données qui vont construire l'objet à renvoyer en sortie du composant ICARE. Donc il y a ni duplication (règle 2), ni suppression de la donnée dupliquée (règle 4).

$InPutPort1Event =$ **ANY**
  $inPort1, date$
WHERE
  $inPort1 \in LANGUAGE1 \wedge$
  $date \in NATURAL \wedge date > lastDate$
THEN
  $inPutICARE1 := inPutICARE1 \cup \{inPort1\} -$
  $\{xx | xx \in inPutICARE1 \wedge$
  $\exists dd.(dd \in NATURAL1 \wedge (xx, dd) \in dateEvent \wedge date - dd > temporalWindow\} \parallel ...$
...
$OutPutEvent =$ **ANY**
  $xx, x1, x2$
WHERE
  $x1 \in inPutICARE1 \wedge$
  $x2 \in inPutICARE2 \wedge$
  $\forall y1.(y1 \in inPutICARE1 - \{x1\} \Rightarrow$
  $\exists d1, d2.(d1 \in NATURAL \wedge d2 \in NATURAL \wedge$
  $(x1, d1) \in dateEvent \wedge (y1, d2) \in dateEvent \wedge d1 > d2)) \wedge$
  ...
  $xx \in LANGUAGE \wedge xx = F(x1, x2) \wedge ...$
THEN
  $outPutICARE := xx \parallel ...$
END ;

Le 3eme raffinement

Ce raffinement sert à spécialiser le composant ICARE par rapport à l'une des propriétés CARE que nous voudrions obtenir. Cette étape peut être intégré au niveau du deuxième raffinement, puisqu'il n'y a ni nouvelle variable, ni nouvel événement. C'est juste des propriétés à ajouter dans les clauses $PROPERTIES$ et $INVARIANT$. Nous avons préféré le faire dans un nouveau raffinement pour avoir dans un premier temps un modèle générique pour les composants ICARE, à partir duquel nous pourrons avoir tous les composants pour chaque propriété CARE.

La fonction du composant ICARE dans notre modèle, dépend du résultat renvoyé par la fonction F. Nous présentons dans ce qui suit les propriétés du composant ICARE pour la redondance et les propriétés du composant ICARE pour la complémentarité.

- La propriété de redondance exprime le fait d'avoir en entrée deux données qui portent la même information (garde de l'événement $OutPutEvent$) et de renvoyer l'une des deux données en sortie (clause $PROPERTIES$).

```
...
PROPERTIES
    ran(F) = LANGUAGE1 ∩ LANGUAGE2
...
EVENTS
...
    OutPutEvent = ANY
        xx, x1, x2
    WHERE
        x1 ∈ inPutICARE1∧
        x2 ∈ inPutICARE2∧
        x1 = x2 ∧
        ...
        xx ∈ LANGUAGE∧ xx = x1 ∧
        xx = F(x1, x2) ∧ ...
    THEN
        outPutICARE := xx || ...
    END ;
```

- la propriété de complémentarité exprime le fait d'avoir en entrée du composant ICARE deux données avec des informations qui peuvent être différentes (aucune condition) et renvoie en sortie une donnée avec une information construite à partir des entrées du composant (résultat de la fonction F au niveau de l'événement $OutPutEvent$) et qui est aussi différente des deux entrées (clause $PROPERTIES$).

```
...
PROPERTIES
    ran(F) ⊆ LANGUAGE − (LANGUAGE1 ∪ LANGUAGE2)
...
EVENTS
...
    OutPutEvent = ANY
        xx, x1, x2
    WHERE
        x1 ∈ inPutICARE1∧
        x2 ∈ inPutICARE2∧
        ...
        xx ∈ LANGUAGE∧
        xx = F(x1,x2) ∧...
    THEN
        outPutICARE := xx || ...
    END ;
```

3.3.3 Exemple d'application à une tâche utilisateur

En plus d'être générique, le modèle que nous avons développé pour les composants offre une facilité d'utilisation dans le cas d'application à des études de cas. Aucune modification majeure n'est nécessaire pour intégrer le modèle dans une spécification d'un système interactif multi-modal. Nous donnons ici l'exemple de la tâche $InputName$ de l'étude de cas $PagesJaunes$. Cette tâche est réalisé par complémentarité entre deux langages $LANGUAGE1$ et $LANGUAGE2$ et la fusion des données des deux langages est définie dans la fonction F.

Il suffit de synchroniser les événement $InPutICARE$ et $InputName$ avec un variant var pour permettre la sauvegarde de l'information $outPutICARE$ dans la variable du système $NameValue$.

```
...
PROPERTIES
    LANGUAGE = {NAME, JEAN, ..., NAME_JEAN, ...}∧
    LANGUAGE1 = {NAME, ..., }∧
    LANGUAGE2 = {JEAN, ..., }∧
    F = {((NAME, JEAN), NAME_JEAN), ..., }∧
...
EVENTS
...
    OutPutEvent = ANY
        xx, x1, x2
    WHERE
        x1 ∈ inPutICARE1∧
        x2 ∈ inPutICARE2∧
        ...
        xx ∈ LANGUAGE∧
        xx = F(x1, x2) ∧ ...
        var = 1
    THEN
        outPutICARE := xx || ...
        var := 0
    END;

    InputName = SELECT
        ...
        var = 0
    THEN
        NameValue := outPutICARE ||
        ...
    END;
```

3.4 Conclusion

L'approche fondée sur les composants permet d'avoir un développement mixte : descendant pour les tâches et ascendant pour la réutilisation de composants pré-

définis de la plate-forme ICARE.

On obtient ainsi deux approches de développement l'une qui utilise le modèle de tâches pour la représentation de la totalité du développement et la seconde est l'approche mixte présentée dans ce chapitre.

Chapitre 4

Bilan

4.1 Résultats obtenus

Dans cette partie du projet Verbatim, nous avons poursuivi les travaux présentés dans la première partie du projet décrite dans le SP3-LOT 2. Nous avons :

- pris en compte les modèles de tâches utilisés pour la spécification et/ou validation d'IHM3. Nous avons considéré les modèles de tâches décrits avec CTT et avons montré comment le raffinement permet de coder la décomposition de tâches en sous-tâches. L'approche proposée est complètement codée en B et plusieurs exemples ont permis de valider cette approche ;
- traité le problème de la réutilisation dû à l'utilisation d'une approche descendante aussi bien dans la modélisation des tâches (par la décomposition) que dans l'utilisation de la méthode B (avec le raffinement). La modélisation en B des composants ICARE a permis de constituer une bibliothèque générique de modèles B événementiel correspondants à chaque type de composant ICARE. Cette approche a permis d'une part d'introduire une méthodologie de conception d'IHM3 en indiquant à quel niveau la décomposition de tâches se termine pour passer au code, et d'autre part à produire des tâches qui satisfont les propriétés CARE par construction.

4.2 Insuffisances et perspectives

Les tâches définies sont formellement vérifiées lorsqu'elles sont formalisées en B. Cependant, il n'est pas possible de savoir si celles-ci correspondent effectivement à des tâches désirées.

Pour traiter ce problème, nous avons poursuivi notre travail de modélisation de tâche en proposant d'animer, a priori, les modèles B représentant les modèles de tâches. Ce travail sera présenté dans le SP3-LOT 4.

Bibliographie

[AAASB06] Y. Ait-Ameur, I. Ait-Sadoune, and M. Baron. Etude et comparaison de scénarios de développements formels d'interfaces multi-modales fondés sur la preuve et le raffinement. In *MOSIM 2006 - 6ème Conférence Francophone de Modélisation et Simulation. Modélisation, Optimisation et Simulation des Systèmes : Défis et Opportunités*, Rabat, 2006.

[AAASBM06] Y. Ait-Ameur, I. Ait-Sadoune, M. Baron, and JM. Mota. Validation et vérification formelles de systèmes interactifs multi-modaux fondées sur la preuve. In *18° Conférence Francophone sur l'Interaction Homme-Machine (IHM)*, pages 123–130, Montréal, 2006.

[AS05] I. Ait-Sadoune. *Vérification et validation formelle d'IHM Multimodales fondées sur la preuve. Utilisation de la Méthode B.* Mémoire d'ingénieur d'état en informatique, INI, Alger, juin 2005.

[BN04a] J. Bouchet and L. Nigay. Approche à composants pour l'interaction multimodale. In *Actes des Premières Journées Francophones : Mobilité et Ubiquité 2004*, pages 36–43, Nice Sophia-Antipolis, France, 2004.

[BN04b] J. Bouchet and L. Nigay. ICARE : A component-based approach for the design and development of multimodal interfaces. In *Extended Abstracts of CHI'04*, pages 1325–1328, Vienna, Austria, 2004.

[Pat01] Fabio Paternò. *Model-Based Design and Evaluation of Interactive Applications.* Springer, 2001.