Spécification de dialogues et construction d'interfaces modulaires et réutilisables

Groupe COMBIEN
Duma J., Giroire H., Le Calvez F., Tisseau G., Urtasun M.

Résumé :

L'objectif du groupe Combien est de construire un EIAO. Nous avons été amenés à la contruction d'interfaces complexes et présentent des parties similaires pour contrôler l'activité de l'élève. Nous avons réfléchi plus particulièrement au problème de la réutilisation de "morceaux" d'interfaces. Nous décrivons ici les outils généraux permettant la *spécification*, la *réalisation* et la *réutilisation* d'interfaces, que nous avons conçus et réalisés. A partir d'une interface de saisie d'une fiche d'identité, nous introduisons le formalisme IREC de spécification du dialogue d'une interface. Puis nous montrons comment l'environnement de développement d'interfaces EDIREC permet d'éditer cette spécification. A ce propos nous exposons les fonctionnalités d'EDIREC. Celui-ci transforme en code exécutable les spécifications et fournit des aides à la mise au point et au test de composants de l'interface. EDIREC est intégré à VisualWorks et il a été produit à partir de lui-même par amorçage (bootstrap).

Mots-clés : environnement de développement d'IHM, spécification du dialogue, réutilisation de composants, EIAO.

Introduction

Le groupe Combien? travaille depuis plusieurs années sur les différents éléments d'un EIAO dans le domaine du dénombrement. Nous avons mené de front des réflexions théoriques et des expériences d'implémentation. A partir de notre expérience dans l'enseignement des dénombrements en classe, nous avons défini les fondements mathématiques d'une méthode de résolution : la méthode constructive adaptée aux conceptions usuelles des élèves et permettant d'accéder à la théorie mathématique du domaine [Le Calvez et al. 97] [Tisseau et al. 96]. Un des buts du système est d'enseigner cette méthode aux élèves.
Un autre but est de faire prendre conscience aux élèves de l'existence de classes de problèmes liées à la résolution. Pour cela nous proposons à l'élève une série d'interfaces, les machines à construire [Le Calvez et al. 97], spécifiques à chaque classe leur permettant de modéliser un problème de cette classe et de la résoudre suivant la méthode constructive que nous avons définie. Ces interfaces contrôlent l'activité de l'élève, elles sont donc complexes et présentent des parties similaires. Le nombre de classes de problèmes étant important, nous avons été amenés à réfléchir plus particulièrement au problème de la réutilisation de "morceaux" d'interfaces.
Nous avons alors réalisé des outils généraux permettant la *spécification*, la *réalisation* et la

réutilisation d'interfaces.
Ce papier présente l'utilisation de ces outils à travers des exemples. Il reprend le plan de la présentation faite au colloque métaconnaissances de Berder . Cette présentation comportait la création en direct d'interface au moyen de l'environnement de développement d'interfaces EDIREC qui ne peut pas être transcrite ici. Nous avons essayé grâce aux figures d'en donner une approche la plus fidèle possible.
A partir d'une interface de saisie d'une fiche d'identité, nous introduisons le formalisme IREC de spécification du dialogue d'une interface. Puis nous montrons comment EDIREC permet d'éditer cette spécification. A ce propos nous exposons les fonctionnalités d'EDIREC. Celui-ci transforme aussi en code exécutable les spécifications et fournit des aides à la mise au point et au test de composants de l'interface. EDIREC est intégré à VisualWorks et il a été produit à partir de lui-même par amorçage (bootstrap).

Les systèmes interactifs

L'importance croissante des interactions entre des utilisateurs non informaticiens et les ordinateurs a provoqué l'émergence d'une nouvelle branche de l'informatique : les IHM. De nombreuses architectures pour la conception et la réalisation d'interfaces ont été proposées. Elles prennent en compte le domaine, le dialogue et la présentation. Le *domaine* est le noyau fonctionnel du système, qui dans le projet Combien? est formé des objets et connaissances que nous avons définis pour le dénombrement. Le *dialogue* contrôle la dynamique des échanges entre l'utilisateur et le système. La *présentation* concerne la partie perceptible par l'utilisateur. Elle se compose des widgets qui permettent d'une part de présenter les données à l'utilisateur et d'autre part de donner à l'utilisateur les moyens d'agir sur le système.

De nombreux outils commerciaux, souvent désignés de manière générique sous le terme de Systèmes de Gestion d'Interfaces Utilisateurs (SGIU ou UIMS pour User Interface Management Systems), sont proposés actuellement pour faciliter la réalisation de systèmes interactifs [Myers 95]. La plupart d'entre eux concernent essentiellement la partie «présentation» du système : ils évitent au programmeur d'avoir à utiliser directement les primitives d'une boîte à outils ou d'un système de fenêtrage. Leur mise en œuvre est facilitée par l'existence d'éditeurs graphiques permettant de dessiner directement l'interface.
Ces outils rendent effectivement abordable la réalisation de la présentation d'un système interactif à des programmeurs non spécialistes des primitives graphiques et événementielles d'un environnement particulier, et il est avantageux de les utiliser. Cependant, ils laissent la gestion des autres aspects de l'interface à la charge du programmeur, en particulier la spécification et la réalisation du dialogue, comme par exemple la détermination des commandes qui doivent être rendues disponibles ou indisponibles suivant l'état du système. La plupart du temps, le programmeur doit incorporer ces aspects aux procédures de réaction aux événements et les mélanger ainsi à des appels au noyau fonctionnel du domaine. Or la gestion du dialogue est un aspect essentiel d'une interface pédagogique et elle doit pouvoir être traitée séparément, avec des outils appropriés permettant de spécifier le dialogue et d'engendrer une version exécutable de cette spécification. De nombreux travaux de recherche ont traité le problème du dialogue, en particulier pour définir des formalismes de spécification [Dix 91] [Palanque et al. 97]. La plupart de ces formalismes restent théoriques, mais certains d'entre eux sont exécutables [Bastide et al. 95] [Browne et al. 97]. Cependant, pour le moment, peu d'entre eux ont débouché sur des outils intégrés de génération d'interfaces

accessibles sur différentes plates-formes de développement. C'est pour cette raison que nous avons été amenés à réaliser un outil de développement d'interfaces adapté à notre environnement de travail et à nos objectifs, dans le cadre d'un projet de système interactif d'aide à l'apprentissage humain.
Dans cet article les différents éléments du formalisme seront introduits à partir d'un exemple, ce formalisme est plus précisément décrit dans [Tisseau et al. 99].

Formalisme de spécification du dialogue

Notre formalisme est inspiré des réseaux de Petri à objets. Ce type de représentation nous permet de prendre en compte les données et leur circulation au même titre que les actions et leur séquencement.

Interacteur

L'unité de structuration du formalisme est appelé un *interacteur*. Il modélise le comportement d'une fenêtre d'interface ou d'une sous-fenêtre. Pour décrire ce comportement il faut préciser à tout moment quelles actions sont applicables à quels objets et quel est le résultat de l'application de l'action. Dans le formalisme IREC les entités responsables de l'exécution des actions s'appellent des *commandes* ; les entités destinées à contenir des objets s'appellent des *variables*.

Les variables et les commandes d'un interacteur sont structurées en graphe biparti. Nous appelons ce graphe le réseau de contrôle de l'interacteur (cf. figure 1).

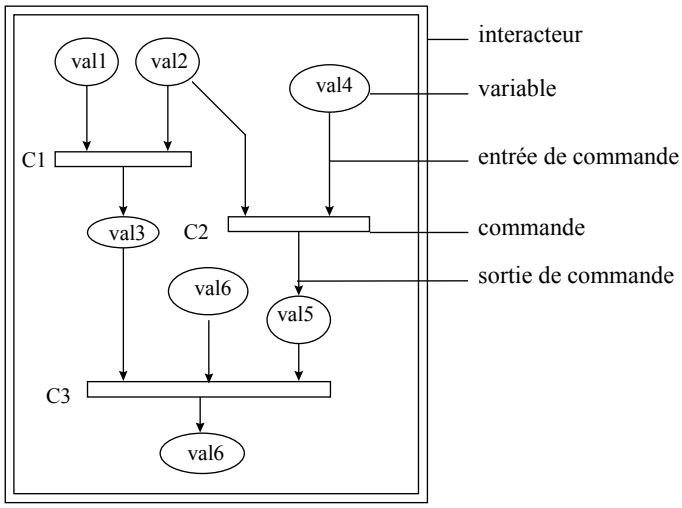


Figure 1

Dans un tel réseau les commandes seront déclenchables seulement lorsque leurs variables d'entrée acquerront une valeur. Dans l'exemple ci-dessus les commandes ont plusieurs variables d'entrée et une variable de sortie. C1 ne pourra être déclenchée que lorsque ses variables d'entrée auront des valeurs (ici val1 et val2) ; l'application de l'action de la commande C1 affecte à sa variable de sortie une valeur (ici val3). Cela se passe de la même façon pour les commandes C2 et C3.

Variable

Une variable peut contenir à tout moment une valeur, qui est un objet quelconque, ou aucune. C'est une restriction et une simplification par rapport aux réseaux de Petri à objets généraux, dont les places peuvent contenir plusieurs jetons étant eux-mêmes des n-uplets d'objets. Cette simplification permet de faire jouer un rôle important à deux sortes d'événements : la *validation* d'une variable (lorsqu'une variable sans valeur acquiert une valeur) et son *invalidation* (lorsqu'une variable qui a une valeur perd cette valeur). Pour une variable donnée d'un interacteur donné, ces événements peuvent avoir seulement trois causes : 1) l'utilisateur peut affecter une valeur à la variable par l'intermédiaire d'un widget, 2) l'interacteur lui-même peut affecter une valeur à la variable comme résultat de son fonctionnement interne et 3) un autre interacteur peut affecter une valeur à la variable (ce qui n'est possible que si celle-ci est déclarée être *partagée* par les deux interacteurs, ce qui sera expliqué plus loin).

Commande

A une commande sont associées des variables d'entrée, éventuellement un déclencheur, deux préconditions d'autorisation et d'acceptation, une action et la plupart du temps des variables de sortie.
La commande devient disponible si toutes ses variables d'entrée sont valides et si la *précondition d'autorisation* (par exemple : la valeur d'une variable d'entrée est du bon type) est vraie. Lorsqu'une commande est disponible, si un widget (exemple champ de texte ou bouton) lui est associé, il sera disponible pour l'utilisateur. Une commande associée à un widget aura une variable d'entrée particulière appelée *déclencheur*. Ce déclencheur est validé lorsque l'utilisateur agit sur le widget associé.
Les commandes sans déclencheur sont déclenchées lorsqu'elles sont disponibles, celles avec déclencheur sont déclenchées lorsqu'elles sont disponibles et que le déclencheur devient valide.
Nous avons introduit la notion de *précondition d'acceptation*, spécifique des interfaces pédagogiques. Elle permet au concepteur de l'interface de contrôler certaines erreurs de l'élève. L'action associée à la commande sera exécutée si la commande est déclenchée et si la précondition d'acceptation est vérifiée. On laisse ainsi à l'utilisateur la possibilité de faire certaines erreurs (ce qui, dans un contexte pédagogique peut être une bonne chose) mais sans conséquence néfaste (l'action n'est pas exécutée et l'erreur est expliquée).

Un exemple

Pour présenter notre travail nous allons nous appuyer sur un exemple très simple. Il se fait en deux étapes. Dans la première partie le domaine est un monde d'"individus". Un individu est décrit par son nom, son prénom, son genre. L'interface permettant à l'utilisateur de rentrer les caractéristiques d'un individu se présente ainsi :

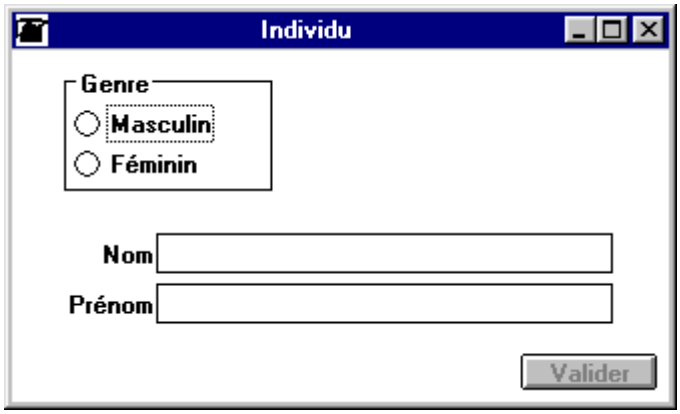


Figure 2

Lorsque l'utilisateur a validé sa saisie, l'interface devient :

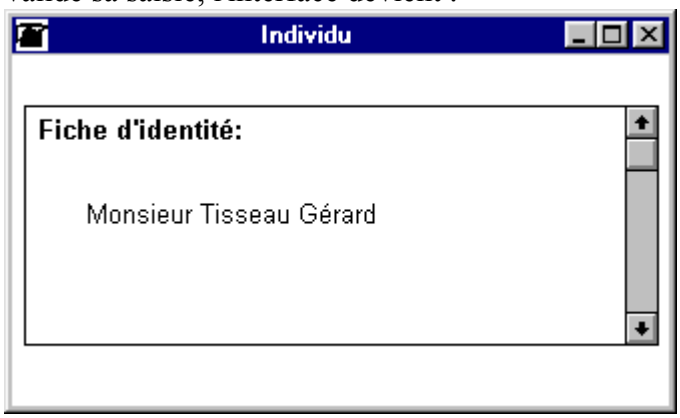


Figure 3

Interacteur

L'interacteur doit fournir un individu, ceci nous conduit à introduire une *variable de sortie* "individu" qui contiendra la valeur de l'individu créé. L'interacteur doit permettre la saisie des trois attributs, nom, prénom, genre. Nous aurons donc trois commandes de saisie associées à trois widgets, qui permettront d'affecter leur variable de sortie (ici nom, prénom, genre). Comme dans cet interacteur, nous n'imposons pas d'ordre dans la saisie des valeurs, elles doivent être disponibles en même temps, il suffit donc d'une seule variable d'entrée pour les trois commandes. Ici la précondition d'autorisation est par défaut à vrai. La commande valider ne doit être disponible que lorsque les trois variables nom, prénom et genre ont des valeurs. Ces trois variables sont donc les entrées de la commande valider. La précondition d'autorisation est aussi par défaut à vrai.

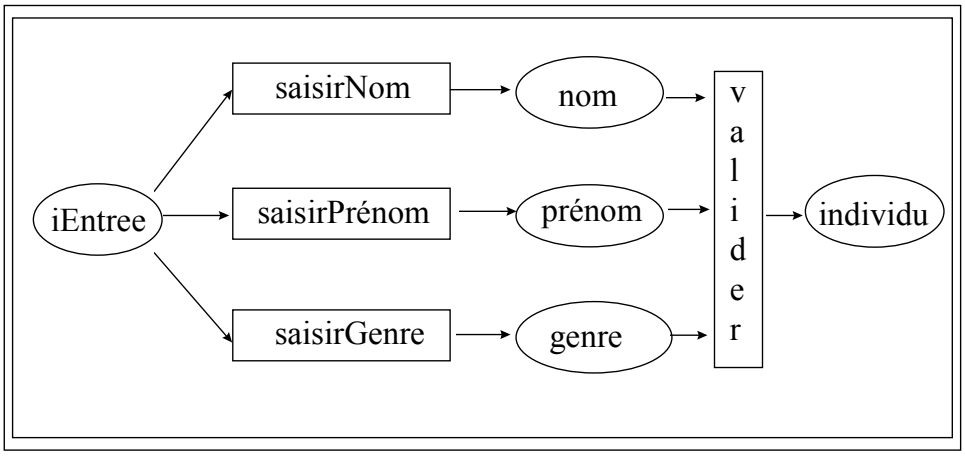
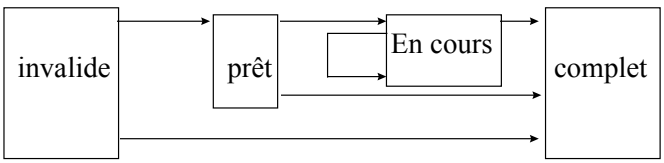


Figure 4

Etat de l'interacteur

Tous les interacteurs dont nous avons eu besoin présentent des comportements communs. Ils nous permettent de construire un objet en saisissant les réponses de l'élève et la fenêtre associée se présente sous diverses formes au début, en cours et à la fin de la saisie. Ceci nous a amené à introduire la notion d'état de l'interacteur. Nous avons défini quatre états : invalide, prêt, en cours, complet. Nous avons ainsi pu associer à ces états la visibilité ou non des widgets.
L'état invalide correspond à un interacteur qui ne peut pas être utilisé (il n'y a pas de fenêtre).
L'état prêt est l'état initial de l'interacteur (cf. figure 2). Dès que l'utilisateur agit sur l'interface, l'interacteur passe à l'état en cours ou à l'état complet (final) si il a terminé.
Nous avons associé un automate de transition à chaque interacteur qui prend en compte ces quatre états et les transitions entre ces quatre états. Ces transitions se produisent lors des événements de validation ou d'invalidation des variables. L'automate étant défini à part une fois pour toutes, chaque interacteur n'a plus à redéfinir cette structure commune, mais seulement à préciser les préconditions et les actions des transitions qui y figurent.

a) sur validation d'une variable



b) sur invalidation d'une variable

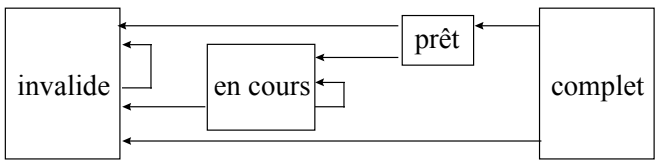


Figure 5

Cet automate complète le réseau de contrôle et sert en fait d'abréviation. Il pourrait être traduit directement dans le réseau, mais cela surchargerait tous les automates. C'est une composante commune à tous les interacteurs.
La détermination de l'état courant et des transitions à exécuter fait jouer un rôle particulier à certaines variables de l'interacteur, qui doivent être distinguées dans le réseau de contrôle. Pour distinguer ces variables, on a ajouté au formalisme la notion de *statut* d'une variable. Les statuts possibles sont : *variable d'entrée*, *variable interne*, *variable de sortie* de l'interacteur. Dans notre exemple, iEntrée est la variable d'entrée, nom, prénom et genre sont des variables internes et individu est la variable de sortie.
Nous avons spécifié l'interacteur, nous allons maintenant utiliser EDIREC (Environnement Interactif de Développement d'Interfaces à Réseaux de Contrôle) pour implémenter cet interacteur.

EDIREC

Le logiciel EDIREC que nous avons développé permet d'éditer une spécification d'interfaces dans le formalisme IREC, il engendre la quasi-totalité du code exécutable de l'interface avec ses widgets dans l'architecture AGIREC, il assiste le programmeur d'interfaces pour compléter ce code et le mettre au point. AGIREC pour Architecture Générique pour les Interacteurs à Réseau de Contrôle, est une architecture logicielle à base d'objets qui permet de mettre en oeuvre IREC. EDIREC a été développé en Smalltalk, dans l'environnement VisualWorks.
La fenêtre de l'assistant proposé au concepteur de l'interacteur comporte trois grandes zones, deux concernant la spécification (la liste des composants et la zone d'édition d'un composant) et la troisième les fonctions de génération et de mise au point du code Smalltalk de l'interface. La figure 6 montre cette structure.

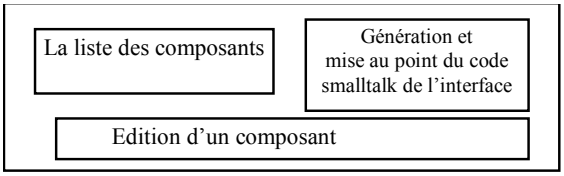


Figure6

La figure 7 ci-dessous montre l'assistant de création d'un l'interacteur au cours de la création de l'interface de saisie d'un individu décrit ci-dessus.

Zone Edition d'un composant

Les panneaux associés à cette zone permettent d'éditer une commande, une variable, un fils (notion que nous présenterons ultérieurement).

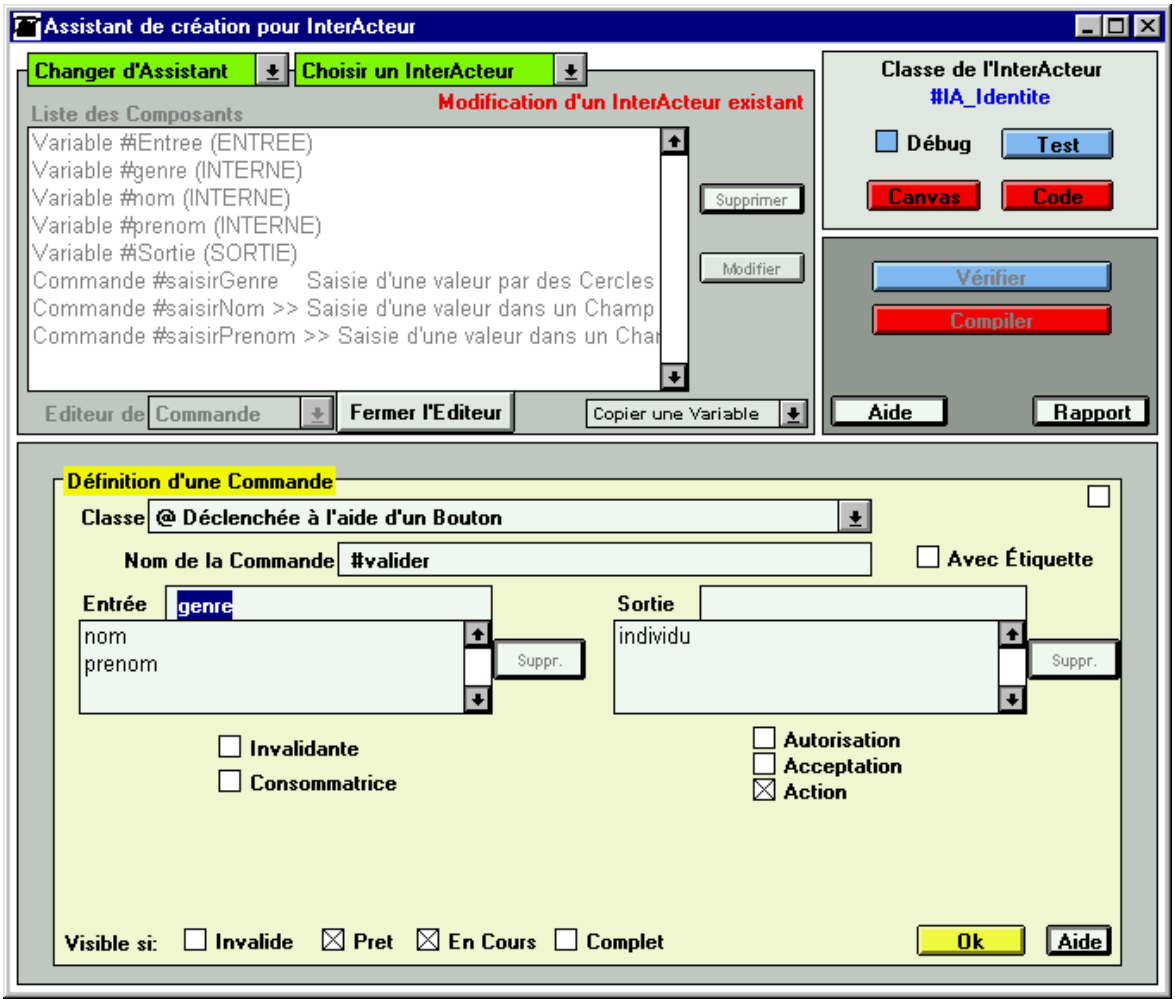


Figure 7

Edition d'une commande

L'éditeur propose une vingtaine de classes de commandes prédéfinies, elles permettent de gérer facilement les éléments graphiques de base (widgets) que l'interface contrôle. En Smalltalk ces commandes sont des classes d'objets, sous-classes de la classe Commande. L'utilisateur d'EDIREC doit indiquer les variables d'entrée et de sortie de la commande qu'il aura définies ou définira au moyen du panneau (figure 8). Les cases à cocher, autorisation, acceptation, action permettent d'indiquer s'il y a une condition d'autorisation et/ou d'acceptation autre que celle par défaut et une action associée à la commande. Dans ce cas, EDIREC va générer une partie du code correspondant. L'utilisateur n'aura plus qu'à programmer l'action ou donner les conditions. Dans le bas du panneau, des cases à cocher permettent de préciser pour chaque état de l'interacteur la visibilité du widget associé à la commande. Le bouton valider associé à la commande sera visible mais grisé lorsque la saisie des différents attributs de l'individu sera possible (états de l'interacteur : prêt et en cours). Ce bouton ne sera dégrisé que lorsque la commande associée sera disponible, c'est à dire lorsque ses trois variables d'entrée seront valides. Le bouton sera invisible lorsque l'état de l'interacteur sera invalide (il n'existe pas) ou complet, l'objet individu a été créé. L'interface se présente alors comme indiqué dans la figure 3 .

Edition d'une variable

Le panneau correspondant à la création de variable permet d'indiquer le type de la variable : individu est une variable de sortie de l'interacteur.

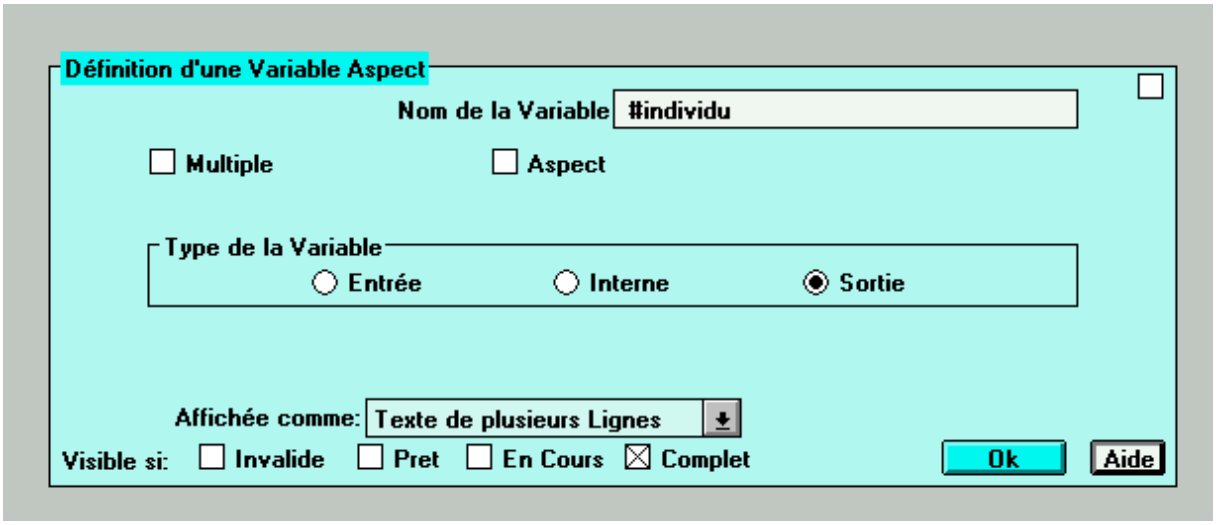


Figure 8

Elle ne sera visible que dans l'état complet et affichée comme "**un texte de plusieurs lignes**" (cf figure 3).

Zone génération et mise au point du code Smalltalk de l'interface

Le bouton **vérifier** permet de vérifier la cohérence des informations qui ont été fournies à l'éditeur. Lorsqu'elles sont cohérentes, le bouton **compiler** permet de faire engendrer par EDIREC le code Smalltalk de l'interface et de préparer les widgets de la fenêtre à afficher. Il reste alors à l'utilisateur d'une part la programmation éventuelle des actions, conditions d'autorisation et d'acceptation d'autre part la mise en forme de la fenêtre (positionnement et taille des widgets, choix des étiquettes). Cette mise en forme se fait au moyen de la palette Visual Works accessible par le bouton **canvas**. Il faudra aussi compléter éventuellement le code correspondant aux transitions entre les états de l'interacteur.
Le bouton **test** permet de tester l'interacteur indépendamment des autres interacteurs de l'application.

Enrichissement de l'exemple

Dans notre interacteur, une fois que l'individu est créé, on ne peut le modifier. On se propose de modifier cet interacteur pour que l'élève puisse recommencer la saisie d'un individu. La fenêtre du nouvel interacteur dans l'état complet se présente ainsi :

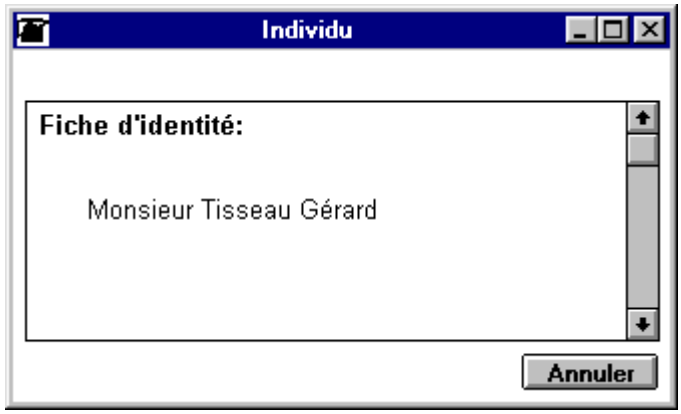


Figure 9

La commande annuler a pour effet de supprimer l'individu créé. L'élève se retrouve devant l'interface de la figure 10.

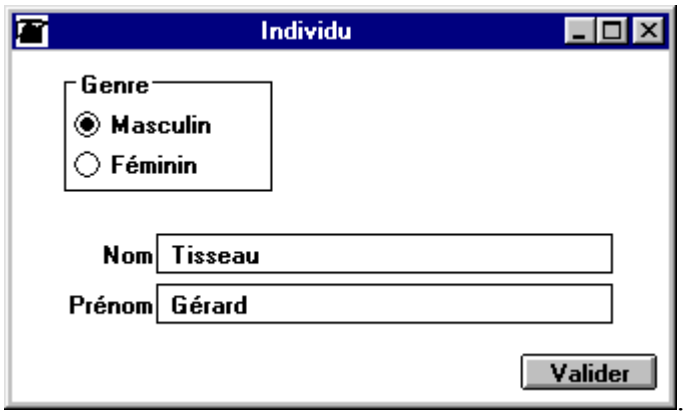


Figure 10

Les commandes de type annuler se sont avérées fréquentes. Pour ne pas laisser à la charge de l'utilisateur la programmation de cet effet, nous avons introduit la notion de *consommation* pour une commande. Une commande consommatrice invalide ses variables d'entrée. De la même façon une commande peut être *invalidante*, dans ce cas elle invalide sa variable de sortie en réaction à l'invalidation d'une de ses variables d'entrée.
La spécification du nouvel interacteur est donnée ci-dessous

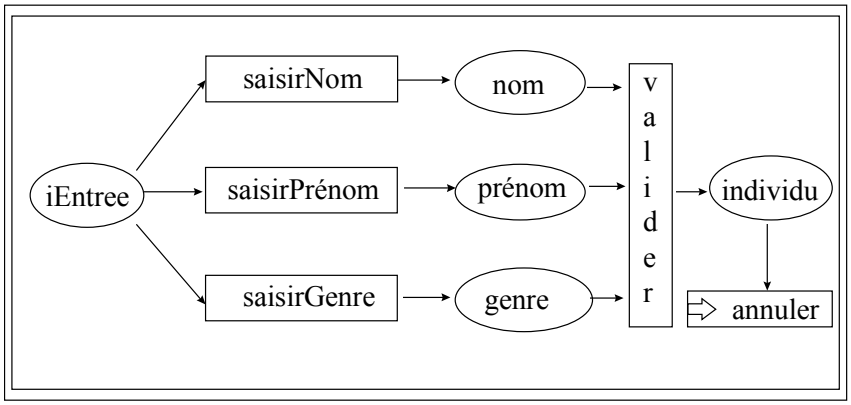


Figure 11

Réutilisation d'interacteurs

On désire permettre la conception modulaire des interacteurs et leur réutilisation. Dans le formalisme IREC, tout interacteur peut en contenir d'autres qui eux-mêmes peuvent en contenir d'autres. La communication entre interacteurs s'effectue par l'intermédiaire de variables partagées.

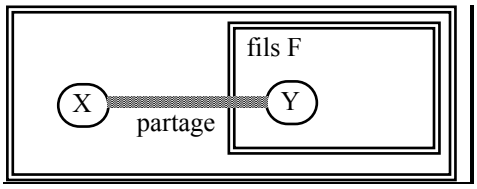


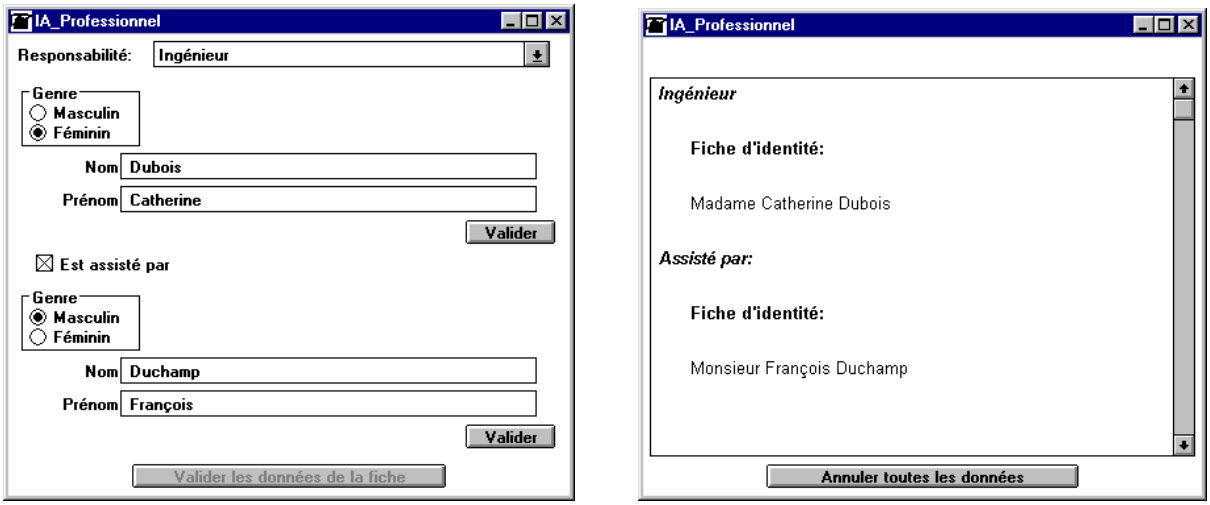
Figure 12 : Inclusion d'un fils et partage de variables

L'interacteur père connaît la variable sous le nom X et le fils sous le nom Y.

Exemple

Le domaine des "individus" s'enrichit de la notion de "professionnel". Le professionnel est un individu qui a une profession, et qui peut avoir ou non un associé (un individu). On retrouve donc les interfaces correspondant à individu aussi bien pour la saisie du professionnel que pour son associé.

état prêt → état en cours



état en cours → état complet

Figure 13 : Différents aspects de l'interface au cours du temps.

L'interacteur saisie d'un professionnel décrit figure 14 contient deux interacteurs de la classe IA-Individu, l'un pour saisir l'identité du chef, l'autre pour saisir l'identité de l'associé.

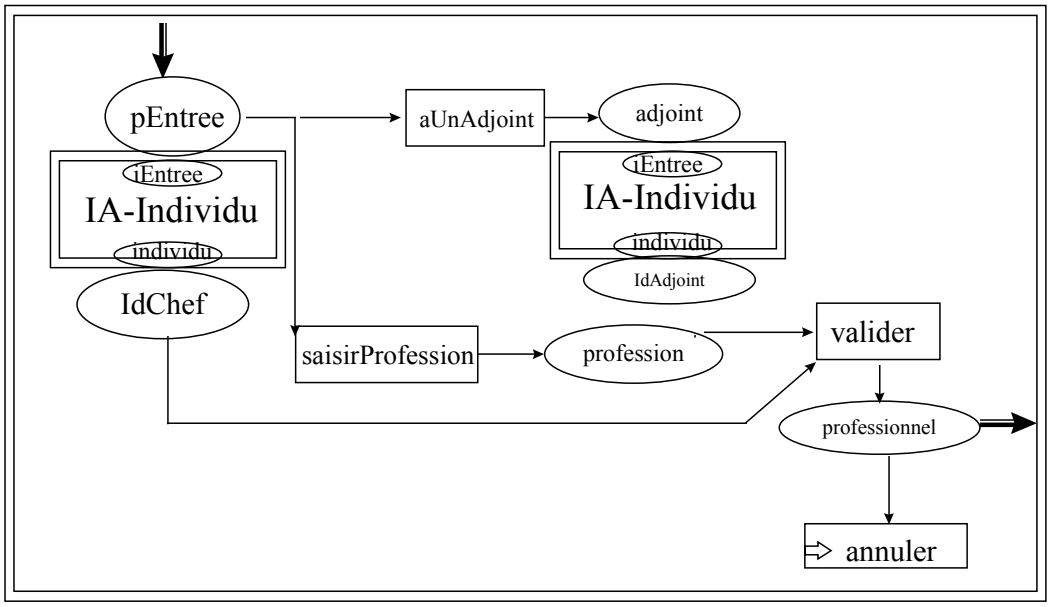


Figure 14

Leurs variables iEntrée et individu sont partagées avec les variables respectives de l'interacteur saisie d'un professionnel pEntrée, idChef d'une part et adjoint et idAdjoint d'autre part. On remarque que dans cet interacteur la commande valider-fiche a deux variables d'entrée idChef et profession. idAdjoint n'est pas variable d'entrée de cette commande car un professionnel n'a pas obligatoirement un associé. Nous avons associé à la commande valider-fiche une condition d'acceptation : si « est assisté par » alors un individu associé doit avoir été créé, qui conditionne l'exécution de l'action associée à la commande. Ceci permet d'être sûr que lorsque la case à cocher « est assisté par » est cochée un individu associé a bien été créé. L'appui sur le bouton commande valider-fiche fera apparaître la fenêtre modale ci-dessous.

Figure 15

EDIREC nous permet d'entrer les spécifications concernant l'utilisation d'un interacteur comme fils d'un autre interacteur comme le montre le panneau suivant. Il suffit de lui indiquer les couples de noms des variables partagées.

LE PANNEAU

Figure16

Application fondamentale d'EDIREC

L'une des premières applications importantes créées avec EDIREC a été … EDIREC lui-même ! En effet, il s'agit d'une application interactive où l'interface joue un rôle important pour aider l'utilisateur dans sa tâche. A ce titre, sa réalisation est susceptible d'être facilitée par un environnement de développement comme EDIREC. Il a fallu bien sûr amorcer le processus : nous avons d'abord programmé une version EDIREC 0 à la main, en utilisant les possibilités offertes par l'architecture AGIREC, puis nous avons utilisé EDIREC 0 pour spécifier et développer une application équivalente EDIREC 1. La comparaison entre les efforts de développement de la version 0 et de la version 1 a suffi pour nous convaincre de l'utilité de l'outil EDIREC. Désormais, chaque nouvelle version n + 1 de EDIREC est produite avec la version n.

Conclusion

Nous venons de présenter EDIREC qui est maintenant totalement opérationnel. Il est actuellement disponible dans les environnements Mac et PC. Nous nous en servons pour réaliser les différentes machines à construire [Tisseau et al. 99] du projet Combien ?. Notre prochaine étape est la réalisation de tests de ces machines auprès des élèves.

Bibliographie

[Bastide et al. 95] Bastide R. & Palanque Ph.(1995) A Petri Net Based Environment for the Design of Event-Driven Interfaces, ATPN'95, Torino, Italy *LNCS n°935*, pp. 66-83, Springer-Verlag.

[Browne et al. 97] Browne T., Davilla D., Rugaber S. & Stirewalt K. Using Declarative descriptions to Model User Interfaces with MASTERMIND. In *Formal Methods in Human-Computer Interaction* Palanque Ph. & Paternò F.(eds) Springer Verlag 1997.

[Dix 91] Dix A. *Formal Methods for Interactive Systems*, Academic Press 1991.

[Le Calvez et al. 97] Le Calvez F., Urtasun M., Tisseau G., Giroire H.& Duma J. "Les machines à construire : des modèles d'interaction pour apprendre une méthode constructive de dénombrement" *EIAO'97*, Cachan, M. Baron, P. Mendelsohn, J.F. Nicaud (eds), Hermès, 1997, pp.49-60.

[Myers 95] Myers Brad A. User Interface Software Tools, *ACM Transactions on Computer Human Interaction*. 1995. **2** (1) pp. 64-103.

[Palanque et al. 97] Palanque Ph. & Paternò F.(eds*).Formal Methods in Human-Computer Interaction* Springer Verlag 1997.

[Tisseau et al. 96] Tisseau G., Giroire H., Le Calvez F., Urtasun M., Duma J., "Une méthode « constructive » de résolution de problèmes de dénombrement et sa mise en oeuvre." Rapport interne Laforia 96/11, mai 1996.

[Tisseau et al. 99] Tisseau G., Giroire H., Duma J., Le Calvez F., Urtasun M., "Spécification du dialogue et génération d'interfaces à l'aide d'interacteurs à réseau de contrôle." IHM 99, Montpellier, 23-26 novembre 1999