COMPO : UN ENVIRONNEMENT DE DEVELOPPEMENT D'APPLICATIONS INTERACTIVES EMBARQUEES

Bérangère Collet, Joëlle Coutaz

Laboratoire de Génie Informatique, IMAG
B.P. 53 X, 38041 Grenoble Cedex
tel.: +33 76 51 44 40
E-mail: bcollet@imag.fr, joelle@imag.fr

Résumé
Compo est un environnement de développement d'interfaces adapté à la construction d'applications interactives embarquées dans des véhicules de tourisme. Il comprend : un éditeur graphique pour définir par manipulation directe l'interface, un éditeur logique pour lui associer un comportement à l'aide d'un langage de spécification événementiel et iconique, un interpréteur (lié à un simulateur des événements-véhicule) pour exécuter et tester immédiatement l'interface construite et enfin, un metteur au point. Au-delà des services usuels d'un environnement de développement, Compo sert de plate-forme d'expérimentation en vue de l'évaluation automatique d'interfaces.

1. Introduction

Les outils d'aide à la construction d'interfaces se répartissent en trois catégories :

- les boîtes à outils, comme Motif [OSF89] construit au-dessus des Intrinsics du serveur X, sont des bibliothèques de procédures dont les services couvrent les différents niveaux fonctionnels d'un logiciel interactif (fenêtrage, affichage, objets de présentation),
- les squelettes d'application, comme Serpent [Bass88] ou MacApp [Schmucker86], se servent des boîtes à outils pour regrouper les notions de noyau d'exécution et de services sous un logiciel réutilisable et extensible,
- les générateurs d'interfaces, comme InterfaceBuilder [Webster89], créent l'interface à partir d'une spécification. Un mécanisme de liaison assemble cette interface, le noyau fonctionnel et le noyau d'exécution en un système interactif exécutable. Le noyau d'exécution s'apparente à un squelette d'applications. La génération d'interfaces s'effectue selon trois axes complémentaires : (a) la **description de la présentation** (schémas d'écrans statiques, UIL [OSF89]), (b) l'**expression du contrôle du dialogue** (règles de comportement, HyperCard [Harvey88]) et (c) la **spécification de l'interface avec le noyau fonctionnel** (production automatique de présentation et du contrôle du dialogue, MacIda UIMS [Petoud89]).

Compo, plus qu'un générateur d'interfaces, est un environnement de développement d'interfaces adapté à la construction d'applications interactives embarquées. Il s'adresse à des concepteurs ergonomes. Il permet de définir l'interface (description de la présentation), de préciser son comportement (expression du contrôle du dialogue) et de l'exécuter avec ou sans simulation des événements-véhicules. Enfin, il sert de plate-forme d'expérimentation pour l'évaluation d'interfaces.
Dans cet article, nous parlerons dans un premier temps, de certains générateurs d'interfaces existants afin de situer Compo. Puis nous décrirons Compo à travers ses concepts et services. En conclusion, nous ferons la liste des perspectives.

2. Quelques générateurs d'interfaces

Les exemples de générateurs d'interfaces suivants, comme Compo, s'orientent vers les axes : expression du contrôle du dialogue et description de la présentation par manipulation directe. Le concept de manipulation directe, utilisé pour la première fois

dans SOS [Hullot86], permet au concepteur de visualiser immédiatement l'interface, ce qui diminue le temps de mise au point.

HyperCard, disponible sur Macintosh, concilie la construction interactive avec la programmation textuelle (HyperTalk) et par démonstration (instruction LinkTo). Une application est composée d'objets organisés selon la hiérarchie Pile-Fond-Carte-Objet. A chaque objet est associé un script composé de traitants d'événements. L'appel de fonctions externes en langage Pascal ou C est possible pour les traitements plus complexes. HyperCard facilite la mise au point et la réutilisation. Cependant il reste limité dans le choix des objets de présentation et nécessite des connaissances informatiques.
HyperLook [HyperLook92], suit une hiérarchie semblable à celle de HyperCard mais fonctionne sous OpenWindows (Unix). Le contrôle du dialogue s'exprime en langage usuel (PostScript) ou spécifique (PdB).

Interface Builder utilise la programmation par démonstration pour exprimer une partie du contrôle du dialogue (liens entre objets de présentation). Mais l'essentiel s'exprime en langage ObjectiveC. Dans la version 3.0 de NeXTStep, la gestion des fichiers liés au projet est géré par Project Builder et non plus par Interface Builder qui se cantonne au rôle de constructeur d'interfaces. Interface Builder permet l'extensibilité des classes de base par édition de lien dynamique de nouvelles classes (possible grâce au système d'exploitation Mach). Bien que très pratique pour la construction et les tests d'interface, Interface Builder impose des connaissances informatiques.

AuthorWare [AuthorWare90] permet l'expression du contrôle du dialogue par manipulation directe d'un langage de spécification iconique. L'appel de fonctions externes en langage C ou Pascal est aussi autorisé. Une application ayant pour but la formation peut guider l'apprenant et évaluer ses performances en utilisant des variables mises à jour par AuthorWare qui renseignent sur les actions de l'utilisateur. AuthorWare permet de se placer facilement dans les conditions d'utilisation. Il est abordable par des non-informaticiens dans la limite de complexité des traitements à effectuer.

En résumé, l'expression du contrôle du dialogue peut se faire à l'aide d'un langage de spécification (HyperLook) ou à l'aide d'un langage de spécification doublé de la programmation par démonstration (HyperCard ou Interface Builder) ou encore par manipulation directe (AuthorWare). De manière générale, les outils utilisant des langages spécifiques (PdB ou HyperTalk) autorisent l'appel de fonctions externes écrites dans un langage de programmation usuel.
L'extensibilité des classes de base ou des objets de présentation est un critère important pour élargir le domaine d'utilisation d'un générateur d'interfaces. Certains (Interface Builder) facilitent l'extensibilité mais pour d'autres (HyperCard) elle est impossible. Les critères de réutilisation et d'aide à la structuration sont appréciées des concepteurs nouveaux ou en début de développement. Ils sont parfois évidents (HyperCard) et quelquefois difficiles à manier (AuthorWare).

Pour Compo, notre objectif a été de réaliser un environnement de développement intégré complet allant de la conception à la mise au point et s'adressant à des concepteurs pas nécessairement informaticiens. Ces concepteurs doivent pouvoir construire des applications complexes réelles. La principale contrainte de Compo est donc la qualité ergonomique. Elle a motivé les choix suivants :

- l'expression du contrôle du dialogue se fait par manipulation directe d'un langage événementiel et iconique,
- aucune connaissance informatique n'étant requises, Compo n'a pas besoin de permettre la programmation, par appel de fonctions externes, dans un langage usuel,
- Compo permet la réutilisation et la structuration des applications,
- les objets de présentation sont en nombre limité (les menus déroulants n'existent pas) et ne sont pas extensibles. En effet le domaine des applications

embarquées impose des interfaces simples et stylées pour une compréhension immédiate, comportant un minimum de texte pour réduire le temps de lecture, et permettant une désignation rapide par l'utilisateur (ce dernier pouvant être le conducteur).
Le chapitre suivant décrit la mise en œuvre de ces critères.

3. Description de Compo

L'origine de Compo est la mise au point de postes interactifs embarqués. Ces postes doivent fournir et recevoir des informations en provenance du véhicule (variation du niveau d'essence) ou de l'utilisateur (action à l'écran pour régler la température).
Compo suit une architecture logicielle fondée sur le modèle PAC [Coutaz90]. Il est développé sur Sparc Station et utilise le serveur X, les boîtes à outils Motif et InterViews, et les langages C et C++.

3.1. Structuration des concepts et réutilisation

Pour faciliter et guider le travail du concepteur, les applications sont construites en suivant une structure hiérarchique portant sur les concepts du domaine : **application, scenario, groupement d'objets de présentation**. Ces derniers sont organisés selon trois bibliothèques, ce qui clarifie et simplifie le travail du concepteur car il peut construire incrémentalement une application et réutiliser des concepts. La figure1 donne l'exemple d'une application véhicule constituée des deux scénarios, réglages et niveaux. Le scénario réglages est lui-même formé de deux groupements d'objets de présentation pour le réglage du chauffage et le réglage du siège.

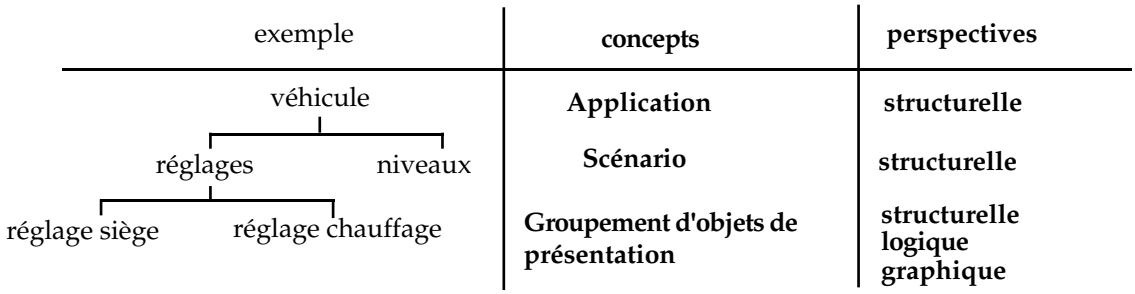


figure 1. Concepts de Compo et leurs persectives

Pour plus de souplesse, Compo offre différentes perspectives sur les concepts (voir la figure1). La **perspective structurelle** montre l'ensemble des sous-concepts d'un concept. La **perspective graphique** correspond à la description de la présentation. La **perspective logique** correspond à l'expression du contrôle du dialogue, possible par manipulation directe d'un langage événementiel et iconique.
Dans Compo, le mécanisme de liaison (assemblage de l'interface, le noyau d'exécution et le noyau fonctionnel en un système interactif) est basé sur les **variables**. Le noyau d'exécution résulte de l'expression du contrôle du dialogue et, particularité de Compo, le noyau fonctionnel correspond à l'ensemble des événements en provenance du véhicule.
Une variable a un rôle interne, variable de calcul, ou un rôle externe, variable de liaison. Elle fait le lien entre des objets de présentation et des événements-véhicule ou utilisateur. Les valeurs actives de Graffiti [Karsenty87] jouent sensiblement le même rôle. Un exemple d'utilisation de variables : A chaque variation du niveau d'essence, le capteur Essence fait varier l'objet de présentation représentant une jauge ObjEss, ceci par l'intermédiaire de la variable VarEss liée au capteur et à l'objet. Inversement, si l'utilisateur fait bouger, par désignation à l'écran, un objet de présentation représentant la température, la variable liée à cet objet est mise à jour afin de refléter la nouvelle valeur. Un exemple de noyau fonctionnel : un système embarqué de traitement de signaux RDS (Radio Data System) annonçant les embouteillages, est le noyau

fonctionnel pour l'interface construite avec Compo représentant le réseau routier. Les informations échangées sont "RN7" pour le capteur route et "87" pour le capteur kilomètre. Les variables liées déclenchent alors à l'écran l'affichage en rouge du kilomètre 87 de la route nationale 7.

3.2. Services intégrés

Compo comprend un ensemble d'outils intégrés ; trois éditeurs (un par perspectives), un simulateur des événements-véhicule, un interpréteur et un metteur au point. La figure 2 montre ces outils et leurs interactions (flèches normales) ainsi que le rôle des variables (flèches en gras).

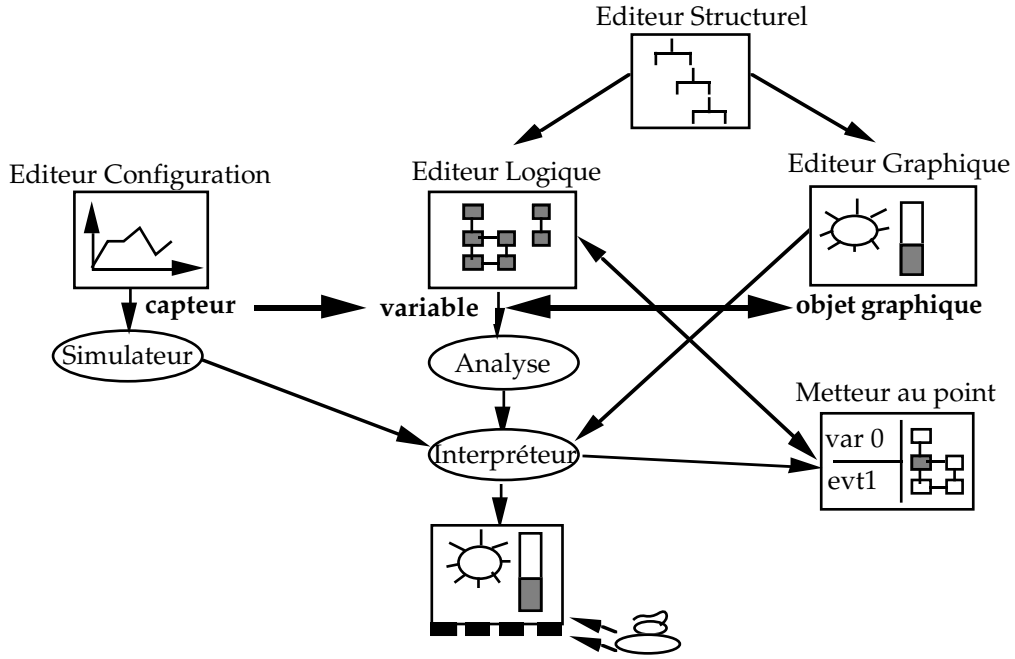


Figure 2. Synoptique des services de Compo.

L'éditeur structurel permet de naviguer parmi les concepts [figure1]. Il suit le modèle d'interaction iconique [Beaudouin91] en utilisant des analogies avec le monde réel (paradigmes). Par exemple, un scénario sera représenté par une bobine de film par analogie avec le monde du cinéma.

L'éditeur graphique permet de décrire la présentation (vue statique du futur écran) par manipulation directe d'une bibliothèque d'objets de présentation. Pour réaliser cet éditeur, nous avons utilisé l'éditeur Idraw [Idraw89] en le complétant par des spécificités du domaine.

L'éditeur logique permet d'exprimer le contrôle du dialogue (comportement dynamique des objets de présentation). Le concepteur définit des scripts par manipulation directe d'un langage iconique et événementiel [Chabert91]. Le formalisme à événements a été choisi pour améliorer la qualité des échanges en tenant compte du parallélisme [Hill87]. Un script est un ensemble de traitants d'événements représentés par une succession verticale d'icônes. A la plupart des icônes est associé à un formulaire permettant de préciser les opérandes de l'instruction représentée. Comme le montre la figure 3, le langage utilise des analogies avec le monde réel ou avec le domaine informatique pour faciliter l'apprentissage et l'utilisation.

Le but des applications construites étant entre autre de recevoir des informations du véhicule, l'interprétation n'a de sens qu'accompagnée d'un simulateur des événements-véhicules. Une simulation est une suite de triplet <capteur,temps,valeur> que le concepteur crée grâce à un éditeur de configuration qui permet de manipuler directement une courbe représentant les valeurs d'un capteur en fonction du temps.

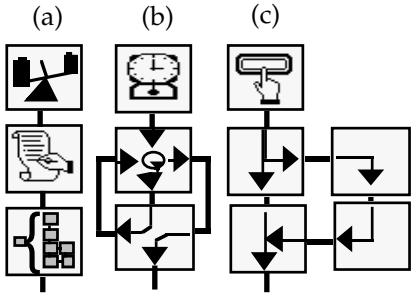
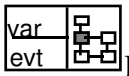


figure 3. Exemple de script : (a) instruction de surveillance de variables suivie d'une instruction de commentaires et d'une instruction zoom (regroupement d'instructions), (b) instruction de surveillance de réveil suivie d'une instruction d'itération, (c) instruction de surveillance de désignation (action de l'utilisateur) suivie d'une instruction conditionnelle.

L'interpréteur utilise les variables pour faire l'association entre les scripts qu'il exécute, les événements-véhicules et utilisateurs qu'il reçoit et les objets de présentation dont il gère l'animation à l'écran.



Le metteur au point est en cours de réalisation. Il permet, grâce à la fenêtre de suivi d'exécution, de visualiser l'état courant du programme, l'évolution des données et les événements surveillés. Il permet également de modifier la valeurs des variables, d'exécuter pas à pas ou encore de marquer des points d'arrêt.

3.3. Plate-forme d'expérimentation

Au-delà des services usuels d'un environnement de développement, Compo sert de plate-forme d'expérimentation dans le cadre d'une étude sur l'évaluation des interfaces Homme-Machine [Balbo92]. L'expérimentation a consistée en :

- la définition d'un modèle de la tâche orienté flot de données (arbre de séquences),
- la mise en œuvre d'un mécanisme de captures des actions de l'utilisateur. chaque action est matérialisée, dans la machine, par un événement auquel est associé un traitement. Chaque traitement stocke donc dans un fichier des informations sur l'action survenue.
- l'analyse manuelle du résultat de la capture pour identifier une base de règles de comportement protant sur les actions de nature sémantique et syntaxique.
- et enfin la liaison entre ces règles et des principes ergonomiques tels que la compatibilité, l'homogénéité, la concision, la flexibilité, le feed-back et le guidage.

La base de règles ainsi esquissée, lié au modèle de la tâche et au mécanisme de capture, constitue un premier pas vers l'évaluation automatique de prototypes exécutables.

4. Conclusion

Les services actuellement disponibles font de Compo un outil de prototypage pour des applications embarquées ayant des interfaces interactives et informatives de l'état du véhicule mais également de l'environnement extérieur (par exemple, représentation de la

route et des véhicules pour informer sur le dépassement). Pour faire de Compo un outil complet, dépassant le stade du prototypage, les perspectives suivantes sont envisagées :

- le metteur au point ; détection d'erreurs et test.
- le portage dans le véhicule des applications construites ; portage du module interprétation et, après transfert des fichiers de données, exécution en situations réelles des prototypes créés sur table.
- l'amélioration de la fonction d'annulation actuellement trop limitée, un des principes de la manipulation directe étant d'offrir des actions réversibles pour encourager l'exploration de l'outil par le nouvel utilisateur [Beaudouin91],
- l'utilisation des résultats du mécanisme de capture des actions de l'utilisateur pour (a) aider, comme le permet AuthorWare, le nouvel utilisateur, (b) améliorer la fonction d'aide en ligne qui pourrait se rapprocher de la tâche de l'utilisateur et (c) déterminer l'expérience de l'utilisateur pour rendre Compo adaptatif.

Compo est bien adapté à son domaine d'exploitation et à ses utilisateurs. La construction d'interface par manipulation directe est la plus utilisée, il est naturel de construire et de spécifier interactivement et graphiquement des interfaces pour des applications interactives. De plus, pouvoir simuler l'application permet des tests de prototypage dont la nécessité n'est plus à prouver pour obtenir des interfaces adaptées aux besoins. Enfin, Compo est un véritable outil pour de réelles applications !

Références

[AuthorWare90]
Intellia, *AuthorWare Professional*.

[Balbo92]
S.Balbo, J.Coutaz, *Un pas vers l'évaluation automatique des interfaces Homme-Machine*, congrès Ergo-IA'92.

[Bass88]
L.Bass, E.Hardy, K.Hoyt, R.Little, R.Searcord, *The Serpent run time architecture and dialogue model*, CMU Technical Report.

[Bass91]
L.Bass et J. Coutaz, *Developping Software for the User Interface*, SEI Series in Software Engineering.

[Beaudouin91]
M.Beaudouin-Lafon, *Interface Homme-Machine : vue d'ensemble et perspectives*, Genie Logiciel&Systèmes Experts (Sept 1991).

[Chabert91]
A.Chabert, *La programmation visuelle*, Rapport de DEA, INPG (Juin 1991).

[Coutaz90
J.Coutaz, *Interfaces homme-machine, conception et réalisation*, Dunod-informatique.

[Harvey88]
G.Harvey, *Understanding HyperCard for Version 1.1*, Sybex Book Publish.

[Hill87]
R.D.Hill, *Event Response Systems-A Technique for Specifying Mutli-thread Dialogues*, Conferences CHI+GI'87.

[Hullot86]
J.M.Hullot, *SOS Interface, un Générateur d'interfaces Homme-Machine*, Actes des journées Afcet-Informatique sur les langages Orientés Objet (Janvier 1986).

[HyperLook92]
Turing Institute Limitded (Juin 1992).

[Idraw89]
Standford University.

[Karsenty87]
S.Karsenty, *Graffiti, un outil interactif et graphique pour la construction d'interfaces homme-machine adaptables*, thèse de doctorat (Décembre 1987).

[OSF89]

OSF/Motif, *Programmer's Reference Manual*, OpenSoftware Foundation.

[Petoud89]
I.Petoud, Y.Pigneur, *An Automatic and Visual Approach for User Interface Design*, conférence IFIP WG2.7 (Août 1989).

[Schmucker86]
K.Schmucker, *MacApp, an application framework*, Byte.

[Webster89]
B.F.Webster, *The NeXT Book*, Addison Wesley (Août 1989).