Conception des interfaces : et si nous analysions enfin la tâche du programmeur ?

Johnny Accot *Stéphane Chatty* *Yannick Jestin* *Stéphane Sire*

Centres d'Études de la Navigation Aérienne
7, avenue Édouard Belin
31055 Toulouse Cedex

{accot, chatty,jestin, sire}@cena.dgac.fr

RÉSUMÉ

Le programmeur est un client d'une ou plusieurs toolkits au même titre que l'utilisateur du logiciel final est un client du programmeur. Il se pose les même questions en terme de puissance d'expression et de facilité d'utilisation. Le choix d'utiliser une toolkit particulière pour un projet pose des contraintes à différent niveaux de conception. Ces contraintes amenènent souvent le programmeur à appliquer des stratégies variées pour contourner les limites de la toolkit, là où son pouvoir d'expression ne correspond pas à ses attentes. A partir d'un série d'interviews, cette prise de position présente quelques points à prendre en considération pour simplifier la tâche de programmation d'interfaces.

Mots-clés

Boîtes à outils, événements, comportement, génie de l'interaction, langages, conception

INTRODUCTION

Pour programmer des interfaces aujourd'hui, un grand choix de langages et de toolkits s'offre au programmeur. En observant des programmeurs dans leur activité quotidienne, il nous est apparu que celle-ci pouvait être analysée au même titre que la tâche de tout utilisateur d'un système informatique. Nous avons donc entrepris une analyse de l'activité, sous la forme d'une étude ethnographique de programmeurs d'interfaces. Pour cela, nous avons interviewé 4 individus, de sexe mâle, âgés de 25 à 35 ans, pris au hasard au sein d'un centre d'études dont nous gardons l'anonymat. Ensuite, au cours d'une séance de confrontation avec nos sujets, méthode du *brainstorming*, les réponses données aux interviews nous ont servi à définir quelques points à améliorer pour augmenter l'utilisabilité des toolkits. Nous proposons d'utiliser les résultats de cette enquête comme point de départ à un atelier sur les toolkits et la conception des interfaces.

INTERVIEWS

Les interviews suivantes visent à bien mettre en évidence les différences de besoins des programmeurs d'interfaces.

Programmeur 1

Q : Quelle toolkit utilisez vous le plus souvent pour programmer vos interfaces ?

R : Après quelques essais non concluants, mes préférences vont sans hésiter au couple Perl/Tk. J'ai opté pour Perl tout d'abord pour sa flexibilité et sa forte adéquation au prototypage rapide. Personnellement des langages comme C++ ne me semblent guère adaptés aux besoins des développeurs d'interfaces — trop lourd, trop enclin aux erreurs. Ensuite, Tk me semble être la boîte à outils avec le meilleur compromis qualité/coût. La boîte à outils ne doit pas être un frein au développement des projets mais doit devenir à terme transparente, être un outil et non une tare. Avec une toolkit comme Tk, la créativité est moins brimée par la technique...

Q : Mais quelles sont donc les limites que vous voyez aux boîtes à outils actuelles ?

R : En fait, malgré l'immense ensemble de boîtes à outils et de langages existants, tous les choix possibles sont quasiment équivalents. Nous ne pourrons pas résoudre le problème des toolkits en réécrivant 100 variantes d'une même toolkit ou en interfaçant cette boîte à outils avec plusieurs langages différents. Mon point de vue est qu'il faut repenser entièrement le concept de boîte à outils, pour s'en faire un allié et non un ennemi.

Q : Mais que suggérez-vous ?

R : Intuitivement et intimement, je pense que la communauté en interaction homme-machine doit se pencher sur la définition d'un nouveau langage de description d'interface. Ne pas refaire l'erreur de faire une même couche en C++ au dessus d'X ou utiliser le dernier langage à la mode en espérant le miracle... Mon opinion est qu'il faut reprendre l'étude des toolkits à la base, et définir un nouveau langage de description des entrées, des comportements et du rendu. Ne pas se restreindre aux aspects graphiques également ; les interfaces homme-machine ne doivent pas se résumer à de l'image, mais inclure les recherches sur le multimodal, le collecticiel, l'infographie, l'interaction gestuelle, etc.

Programmeur 2

Q : Quelle toolkit utilisez vous le plus souvent pour programmer vos interfaces ?

R : J'utilise Power Plant sur Macintosh qui offre de nombreuses classes pour la création d'éléments standards tels que boutons, vues, fenêtres, etc. En particuliers il fournit un mécanisme intéressant de récupération des événements à l'aide des classes LBroadcaster et

LListener. Le broadcaster appelle la méthode ListenTo du listener lorsqu'il désire lui envoyer un événement.

Q : Avez-vous recours à d'autres mécanismes pour gérer les événements en provenance de l'utilisateur ?

R : Oui. Malgré la simplicité du schéma listener / broadcaster, celui-ci ne me suffit pas pour passer rapidement du comportement que je veux programmer au code. En effet ce mécanisme est assez lourd pour reconfigurer l'interface en fonction de l'évolution de son état. Par exemple il faut explicitement penser à retirer les listeners lorsqu'on ne veut plus écouter les messages, ce qui correspond à un changement d'état, chose très fréquente dans les interfaces.

Q : Vous voulez dire que vous avez mis au point votre propre mécanisme ?

R : En effet, d'autant plus que j'avais des besoins particuliers pour créer des applications distribuées. J'aime concevoir la partie décrivant le comportement de mon application sous forme de graphes d'états, sur le papier. Je traduis ces graphes en C++ à l'aide de quelques classes qui me permettent de représenter les états et leurs transitions directement par des fonctions C++. Mon mécanisme d'abonnement effectue le désabonnement automatique à la fin de l'exécution du code de la transition. L'avantage de cette méthode est qu'elle me permet d'utiliser des astuces dans le code représentant les états et les transitions, car c'est du C++, pour effectuer à la main quelques optimisations impossibles sur les graphes d'état. D'autre part pour distribuer les transitions, je n'ai pas de code à ajouter, c'est mon mécanisme d'abonnement qui est capable de répliquer les transitions qu'il franchit.

Q : Vous avez réécrit toute une partie de Power Plant ?

R : Non, en fait j'ai créé des classes d'adaptation pour transformer les broadcasters de Power Plant en source d'événements auxquels je peux m'abonner avec mes propres objets.

Programmeur 3

Q : Quelle toolkit utilisez-vous le plus souvent pour programmer vos interfaces ?

R : Tout d'abord, je tiens à préciser de quels types d'IHM il s'agit. J'essaie de rajouter des mécanismes d'enregistrement et de rejeu à des interfaces écrites par d'autres personnes. Dans cette optique, il me faut à la fois intercepter les échanges entre les applications et au sein d'elles, et proposer des mécanismes pour réinjecter ces événements au sein d'une application et entre les applications sans douleur. Pour cela, il me faut travailler avec des programmeurs le plus souvent sous X11, avec des toolkits au dessus comme tk, Xt/Motif, ou des surcouches comme l'AWT/JFC. J'ai programmé moi-même des agents de supervision et des petites applications dans les trois toolkits citées ci-dessus.

Q : Vos besoins sont-il satisfaits ?

R : Je suis extrêmement insatisfait de la puissance d'expression proposée par les toolkits. Il s'avère que mon travail porte sur les événements et leurs mécanismes de diffusion, et que chaque toolkit enferme le programmeur dans un schéma classique du « giant switch statement ». Ce schéma de X11, associé à la lourdeur du langage de programmation sous-jacent, aboutit à des IHMs maladroites et sur lesquelles il est impossible d'appliquer un contrôle extérieur : il y a des années qu'il est possible de « scripter » des applications sous MacOS ! Je pense donc qu'il faut repenser le clivage entre les langages et les toolkits pour donner au programmeur plus de contrôle dans l'adressage et le flux des événements dans ses applications.

Programmeur 4

Q: Quelle toolkit utilisez-vous le plus souvent pour programmer vos interfaces ?

R: Oh, je ne programme plus beaucoup. Ces dernières années, j'ai essayé Tcl/Tk, Open Inventor, Perl/Tk, la vieille toolkit Xtv du LRI, et même Motif. Mais plus rien ne me satisfait. Alors, je fais comme tous les vieux dans ces cas-là, je me fais du cinéma dans ma tête. J'imagine une application avec de belles formes et de jolies couleurs, qui ferait tout ce que j'attend d'elle. Je rêve tout dans les moindre détails. Et quand j'essaie de passer à l'action, ça ne dure jamais bien longtemps. Le langage et la toolkit me forcent à des contorsions qui ne sont plus de mon âge. Il me faudrait des heures pour exprimer des choses qui se disent en un dessin ou deux phrases sur un coin de tableau. D'abord, il y a les médias, et en particulier le graphisme. Quand on a fait de l'Illustrator, du 3D Studio ou du Photoshop, on est frustré par le pouvoir d'expression des toolkits. Mais le pire, c'est le comportement : je n'arrive pas du tout à exprimer ce que j'ai en tête. Le summum de la frustration, c'est quand il m'arrive de vouloir faire des programmes non graphiques selon le même schéma que des programmes interactifs classiques. Et là, c'est l'horreur : je me retrouve avec un langage tout nu, sans rien.

Q: Donc, vous voulez faire une toolkit radicalement nouvelle ?

R: Non, même pas. On retomberait dans les mêmes problèmes : il y a de bons outils pour faire du graphisme ou du son, on a tous des moyens plus ou moins précis d'exprimer les comportements, et dès qu'on essaye de faire une toolkit on mélange tout ça, on ajoute le langage, et le résultat est médiocre. Pour moi, le problème principal c'est de savoir ce qu'on a envie d'exprimer : "ça fait ça, puis ça attend 30 secondes, et ça fait autre chose ; mais si entretemps il se passe quelque chose, alors...", et ça c'est en général le rôle du langage. Les drivers qui gèrent des dessins, des fenêtres, de la voix, une souris ou une télécommande à infra-rouge, c'est presque secondaire. Ça devrait s'insérer de manière uniforme, comme un driver dans le noyau ou des procédures dans un langage.

DISCUSSION

Après discussion avec ces programmeurs, il nous semble que la cause commune de leurs frustrations provienne de la rigidité des compromis réalisés par le triplet toolkit, langage de programmation et système

d'exploitation. En effet, une fois choisie une triade, le programmeur est condamné à subir un modèle de programmation unique (appel de fonction et diffusion d'événements, architecture, etc.) qu'il ne peut pas faire évoluer dynamiquement ou adapter à des besoins variés pour des parties différentes de l'interface. De plus, dans les toolkits, ces modèles de programmation sont bien souvent reconstruits autour des drivers (fenêtres, graphisme, son, réseau, etc.) utilisés par la toolkit. Au final, les mêmes mécanismes se trouvent souvent réécrits à des niveaux différents, ce qui ne fait que prêter à confusion.

A l'opposé, l'activité de programmation d'une interface peut-être considérée comme une véritable tâche de programmation, et non pas uniquement comme la spécialisation de morceaux de codes déjà écrits par d'autres. L'un n'empêche pas l'autre bien sûr, mais il nous semble qu'aujourd'hui l'aspect programmation soit bien souvent négligé au profit de solutions toutes faites. L'absence d'un véritable langage adapté à la programmation des interfaces est l'une des causes de cette situation. Pour détailler cette position, nous avons fait exprimer aux quatres programmeurs quelques unes des propriétés pour lesquelles ils n'ont pas trouvé de support dans leurs outils. Bien qu'incomplète, cette liste de propriétés définit un espace de conception dont nous nous inspirons pour esquisser des pistes d'étude.

PROPRIÉTÉS DÉSIRÉES

Les propriétés suivantes donnent quelques indications sur les limites connues des boîtes à outils.

Modèle graphique efficace et expressif

Aujourd'hui, développer une application en deux dimensions ou trois, nécessite des boîtes à outils tout à fait différentes : le modèle graphique, la gestion de l'interaction, l'esthétique du rendu sont fondamentalement différents. Les moteurs graphiques 3D du type OpenGL sont très efficaces et puissants. Est-il possible d'utiliser ces apports récents dans un monde en 2D ? Est-il acceptable de voir que la gestion de la transparence, de calques ou du plaquage de textures est réservée à certaines architectures ?

Possibilités d'extension, dynamicité

Les boîtes à outils classiques fournissent un ensemble de widgets bien standardisés (radio buttons, menus, scroll-bars, check buttons, etc). Ces widgets constituent des bases puissantes pour le développement d'interfaces classiques, car elles permettent en quelques lignes de créer des menus, des boutons, etc. Pourtant, tout développeur qui a voulu explorer un jour un type d'interaction non standard s'est heurté à la barrière « nouveau-widget ». Il n'y a alors pas de secret : on recopie chez soi le code du widget ayant le comportement le plus proche de ce que l'on désire, et on modifie ce code jusqu'à en être satisfait. Serait-il possible d'inclure dans une boîte à outils une notion d'héritage de comportement, de spécialisation, etc... des widgets ? Est-ce possible de façon dynamique ?

Aide à la conception

Pour programmer une interface, il existe très peu d'outils spécialisés. Cela va à l'encontre de l'idée commune qui est qu'il existe de nombreux *interface builders*... Mais ceux-ci, à bien y regarder, permettent généralement de placer des boutons, menus et autres gadgets décoratifs. Pour faire le travail utile, les choses se gâtent : on ouvre son éditeur de texte préféré, on attaque le code à la main. On crée un canevas ou une vue pour y dessiner, déplacer, modifier des objets. Que l'on fasse du collecticiel, du rejeu, du multimodal ou du graphisme, le seul outil dont on dispose est l'éditeur syntaxique qui colore en bleu ou orange les mots clés du langage. Par contre, nul outil ne vient aider à appréhender le comportement final de l'interface. Par dépit, on en revient souvent à essayer de tout exprimer par des menus déroulants, se déroulant à perte de vue. L'interview du programmeur 4 a bien résumé le problème : il semble plus facile de spécifier un comportement sans utiliser de langage de programmation strict ; un dessin suffit bien souvent. N'y a-t-il pas donc une inadéquation entre besoins et outils ? N'existe-t'il pas de moyens pour assister le programmeur dans la spécification du comportement de son interface ? Et si ce support se faisait au niveau de la boîte à outils elle-même ?

Gestion de la multimodalité

Les boîtes à outils ne sont pas prévues pour supporter plusieurs moyens d'entrée : concurrence des périphériques pour l'accès aux objets, gestion parallèle et synergique des entrées ad hoc, faible degré de contrôle du flux de données, etc. Toutes des raisons font que l'on est souvent incapables de gérer de nombreuses entrées et sorties de façon multimodale. L'argument de l'inutilisabilité de l'interface résultante apparaît plutôt comme un prétexte.

Temps réel et synchronisme

Pour inclure du son dans une interface, il n'y a pour ainsi dire aucun moyen de l'associer avec l' image dans les boîtes à outils courantes. Le programmeur doit redescendre au plus bas niveau d'abstraction, gérer lui-même les priorités entre tâches, synchroniser les flots de données sonores et visuels, etc. Des tentatives ont été faites pour offrir un support pour le son, nécessaire pour la téléphonie par exemple, mais tous ces supports sont greffés au dessus de la boîte à outils. Pourtant, le son est une donnée de même niveau que l'image. Il nous semble alors impératif d'étendre le champ d'application des toolkits à des moyens d'entrées ou de sorties non standards, et d'offrir tous les moyens pour les synchroniser, les diffuser, les gérer dans le temps comme cela est fait de mieux en mieux pour les interfaces visuelles.

Intégration du réseau

Intégrer le réseau dans une application est une tâche ardûe, en particulier lorsque le but de cette intégration est de distribuer une interface, et donc d'ajouter des utilisateurs à l'application. Le problème pourraît venir du codage de l'information transmise par le réseau, trop distinct des événements en provenance de l'utilisateur. Pourtant de par sa nature : variabilité des délais de

transmissions, asynchronisme, le comportement du réseau est du même type que celui d'un utilisateur. En fait on peut se demander si légitimement on n'aurait pas avantage à considérer le réseau comme un périphérique supplémentaire, et donc à considérer celui-ci comme transportant des événements et non plus des messages, avec les mêmes mécanismes de diffusion et de traitement de ces événements que ceux en provenance de l'utilisateur.

Paramétrage dynamique

Cette notion est liée à l'existence d'un contexte d'exécution variable dans les interfaces, qui peut dépendre par exemple des temps de réaction des utilisateurs, mais aussi de l'état des périphériques. En effet, pour maintenir à jour un état observable de l'interface, les mêmes portions de code peuvent s'exécuter sous des contraintes variables. Un exemple de contrainte évident est le temps. Celui-ci est pris en compte pour effectuer du rendu en 3D, lorque l'on veut garantir un nombre d'images secondes fixe. Dans ce cas, les routines d'affichage sont paramétrées par le temps qui leur est alloué. En fonction de ce temps, elles peuvent dégrader leur rendu pour s'exécuter plus vite. Une autre contrainte, dans le collecticiel, est le débit du réseau. Par exemple si l'on veut garantir une synchronisation parfait entre deux applications, on peut être ammené à exiger un échange de messages (ou d'événements) à intervalles réguliers. Si le débit du réseau diminue, il peut être utile d'en informer les routines qui émettent des événements vers le réseau. En fonction du débit, elle peuvent diminuer le nombre ou la taille des événements qu'elles vont émettre.

Le paramètres peuvent jouer sur les effets de bords demandés à certaines routines, mais on peut imaginer de paramétrer également les mécanismes même d'exécution de la toolkit, comme son modèle de mémoire par exemple. D'autre part les paramètres peuvent être physiques comme le temps ou un débit, mais également de plus haut niveau comme, pourquoi pas, le fait qu'un utilisateur soit présent ou non devant l'écran.

VERS UN LANGAGE DE L'INTERACTION

La prise en compte des besoins cités ci-dessus ne doit pas être prétexte à la création d'une n+1ème boîte à outils. Des solutions existent à la fois dans les toolkits, les langages et les systèmes d'exploitation. Pourtant, lee schéma fonctionnel ou procédural des langages n'est pas adapté à la programmation événementielle, c'est pourquoi nous pensons qu'il faut commencer par recentrer les outils de développement sur la notion d'événement.

Quel que soit son mode de développement d'IHM, le programmeur a besoin de connaître quels sont les événements qui transitent dans son application. Pour pouvoir implémenter des comportements différents de ceux des widgets standards, il faut soit créer de nouveaux types d'événements, soit modifier des widgets existant. La difficulté résulte de la pauvreté de la notion d'événement dans les boîtes à outils actuelles et du contrôle que l'on peut exercer sur leur propagation.

Il nous paraît utile d'améliorer ce modèle sous trois aspects :

- La notion d'événement doit être enrichie, hiérarchisée indépendamment du langage de programmation. Plutôt qu'enrichir une collection de champs, il faut rajouter à l'événement des informations à chaque transition pertinente qu'il franchit dans l'automate de l'IHM. De la sorte, il sera possible d'analyser les causalités de façon hiérarchique, dans un but d'historique, d'undo hiérarchique, de rejeu ou de programmation par démonstration.

- Le programmeur devrait pouvoir contrôler les flux de distribution d'événements plus finement. En effet, les mécanismes de callback ou d'event handler rendent très complexe, voire impossible, l'insertion de mécanismes de traduction, d'abonnement dynamique ou de distribution asynchrone. Pour cela, il faut un mécanisme plus souple qu'une file d'attente unique, et plus utilisable qu'une file d'attente par destinataire. On peut aller plus loin en proposant même au programmeur de modifier la stratégie de diffusion d'événement au cours de l'exécution du programme. Pour cela, il faut peut être descendre au niveau du langage de programmation, qui est déjà astreint à un modèle d'exécution.

- De même que dans les langages à objets, on devrait pouvoir désigner des familles d'événements de manière unifiée, et au contraire spécialiser leur utilisation en fonction de leur source réelle. Par exemple, un programme utilisant le pointage devrait pouvoir être écrit indépendamment du périphérique utilisé, souris, crayon optique ou écran tactile. Cependant, on peut spécialiser cette application pour tenir compte de la pression exercée, ou de toute forme d'information supplémentaire fournie par un périphérique particulier. Pour cela, il est indispensable de pouvoir s'abonner à un click souris quelconque, au click réalisé sur ce bouton, à tous les événements ayant trait à la souris, qu'ils soient primaires, comme le click, ou étendus, comme le double click, ou à tous les événements ayant trait à une sélection à travers un dispositif de pointage quelconque.

CONCLUSION

Bien sûr, la pseudo-étude ethnographique que nous avons faite plus haut n'est qu'un moyen plaisant d'aborder le problème. Mais il n'en reste pas moins qu'en tentant de prendre un regard extérieur, les outils dont disposent les programmeurs et surtout les concepteurs semblent singulièrement fragmentaires et désorganisés par rapport à leur tâche : exprimer sous forme exécutable des comportements conçus par ailleurs. Il est connu qu'on ne peut pas bien analyser sa propre activité. Alors, en attendant que quelqu'un s'en charge pour nous, nous proposons au moins de faire le tri dans ce que nous arrivons à identifier : les fonctionnalités offertes par les toolkits, les langages et les systèmes d'exploitation. Nous proposons donc un atelier qui prendrait une approche méthodique, en se concentrant sur les constructions sémantiques de base utilisées par les programmeurs ou les concepteurs et la manière de les leur rendre accessibles.