Vers des outils pour les équipes de conception d'interfaces post-WIMP

Stéphane Chatty *Stéphane Sire* *Alexandre Lemort*
IntuiLab
Prologue 1, La Pyrénéenne
31372 LABEGE CEDEX, FRANCE
{chatty,sire,lemort}@intuilab.com

RÉSUMÉ
La programmation d'interfaces évoluées reste un travail complexe avec les outils disponibles. A partir d'entretiens avec environ 50 équipes de conception ou de production industrielle de systèmes interactifs, nous avons identifié l'importance du travail collectif pluridisciplinaire dans la réalisation d'un prototype ou d'un produit. Supporter un travail efficace de chacun impose des contraintes nouvelles sur l'architecture des systèmes et sur les outils à réaliser. Nous présentons l'architecture à base de modèles XML que nous avons choisie pour notre atelier IntuiKit, afin de combler le fossé entre conception d'interfaces et génie logiciel. Puis nous illustrons cette solution sur un cas concret de réalisation d'un produit.

MOTS CLÉS : conception participative, conception itérative, prototypage, designers, graphistes, ergonomes, programmeurs, toolkit, atelier, architecture

ABSTRACT
Programming highly interactive user interfaces remains a complex task with the available software tools. Interviews and observation of 50 design or production teams of industrial products shows the importance of group production between graphics designers, programmers, usability experts and the lead designer. Tools must allow each of them to efficiently produce their part. That imposes new constraints on the architecture of interactive software. We present the solutions used our development suite, IntuiKit, designed to bridge the gap between interface design and software engineering. We illustrate it on a concrete product development.

CATEGORIES : H5.2 [Information Interfaces and presentation] User Interfaces — GUI&UIMS ; D2.11 [Software engineering] Software Architectures.

GENERAL TERMS : design, human factors, languages

IHM 2004 Aug 30 – Sep 3, 2004, Namur, Belgium
Copyright 2004 ACM 1-58113-926-8 ……. $5.00

INTRODUCTION

Un article publié à IHM98 portait le titre "Conception des interfaces : et si nous analysions enfin la tâche des programmeurs ?" [1]. L'article soulignait la complexité de la programmation d'interfaces post-WIMP avec les outils disponibles. Il proposait de considérer les programmeurs comme des utilisateurs d'outils interactifs nommés langages et boîtes à outils, et de leur appliquer des méthodes de conception d'interfaces. Quelques années plus tard, les interfaces WIMP dominent encore le marché. Pourtant les avantages des systèmes hautement interactifs ou multimodaux sont reconnus, les méthodes de conception centrée sur l'utilisateur se sont popularisées, et les graphistes et designers ont pris un rôle central dans les interfaces du Web. Mais aucune évolution n'est perceptible dans les outils de programmation les plus répandus et l'industrie informatique peine à trouver des processus rentables pour concevoir et réaliser des interfaces évoluées.

Nous avons choisi d'étudier ce problème selon la ligne proposée par l'article d'IHM98, à la fois par la recherche et sur un terrain industriel. Nous avons découvert que cet article comportait deux erreurs. La première est qu'il existait depuis longtemps une communauté de recherche sur la psychologie et l'activité des programmeurs, autour par exemple de Green [8]. Cependant, ces travaux portent sur les aspects cognitifs de la programmation, et on trouve peu de travaux sur les problèmes spécifiques aux interfaces. La seconde erreur est le point de départ du présent article. On peut la résumer ainsi : *le problème n'est pas d'assister le travail des programmeurs, mais celui des équipes de conception d'interfaces dans leur ensemble, en incluant graphistes, ergonomes et chefs de projets.* Les outils de programmation d'interfaces doivent être conçus pour des équipes pluridisciplinaires, et les choix d'architecture doivent se plier à cette contrainte.

Dans cet article, nous analysons les besoins des équipes de conception tels que nous les avons perçus à travers plusieurs années de travail d'une équipe pluridisciplinaire en laboratoire, et deux ans d'activité commerciale de conception et prototypage d'interfaces hautement interactives ou multimodales. Nous proposons ensuite des directions à prendre en matière d'outils de programmation d'interfaces pour répondre à ces besoins. Nous décrivons enfin les solutions que nous mettons en oeuvre dans un environnement de prototypage d'interfaces nommé IntuiKit, et nous

illustrons ces solutions et leur effet sur un cas réel de réalisation de produits.

LA CONCEPTION DANS L'INDUSTRIE

Sur une période de deux ans, nous avons obtenu des entrevues avec environ 50 équipes chargées de la conception ou la réalisation de systèmes interactifs, dans les domaines de l'automobile, de l'aéronautique, de la défense, de l'espace, et des télécommunications. Nous avons eu l'opportunité d'observer de plus près le fonctionnement d'une dizaine d'équipes au cours de projets menés avec eux.

Ces équipes se répartissaient à part égales entre PMEs, SSIIs, grandes administrations et grandes sociétés industrielles. Deux tiers d'entre elles interviennent dans des projets de R&D en amont de la production, et un tiers dans des projets de production. Parmi les équipes contactées, une faible minorité (environ 5) ne considèrent pas la conception de systèmes interactifs comme un problème. A l'opposé, une très faible minorité (2 à 3) considère avoir traité le problème et avoir mis en place un fonctionnement satisfaisant. La grande majorité (environ 40) a identifié la conception de systèmes interactifs comme un problème, et en est à divers stades d'investissements sans avoir encore trouvé un fonctionnement satisfaisant. Nous avons identifié trois types de situations :

– les sociétés, en particulier les SSII, qui ont entendu parler de diverses méthodes de conception d'interfaces mais n'ont pas encore investi dans le domaine, ne sachant ni comment gérer des équipes pluridisciplinaires ni comment mettre en place des processus de développement maîtrisés ;
– les sociétés qui se sont dotées d'équipes pluridisciplinaires, mais qui sont à la recherche d'un mode de fonctionnement efficace, en particulier en liaison avec les équipes de production ;
– les sociétés qui ont mis en place un processus de conception en amont de la production, mais considèrent ce processus comme trop coûteux et sont à la recherche d'outils pour l'accélerer.

C'est sur cette majorité de groupes que nous allons nous concentrer, en analysant les enjeux pour ces entreprises des processus de conception et les attentes des différents acteurs.

PROCESSUS ET ACTEURS

Les enjeux industriels

Les industriels gèrent leurs processus de production de manière extrêmement stricte, en cherchant à minimiser les risques sur les délais. Les industriels de l'aéronautique, par exemple, considèrent à juste titre que leurs processus de production d'avions à partir de milliers d'exigences et de technologies sont une de leurs plus grandes richesses. Un industriel ne mettra pas ses processus en danger, fût-ce dans l'espoir d'améliorer l'utilisabilité de son produit. Il attendra d'être convaincu de disposer de processus sûrs et prédictibles avant d'intégrer de nouvelles méthodes de travail. Pour les logiciels, les processus sont des variantes du cycle en V.

Les industriels que nous avons rencontrés sont convaincus que la conception de systèmes interactifs gagne à être itérative. Ils adhèrent au principe illustré par la figure 1 :

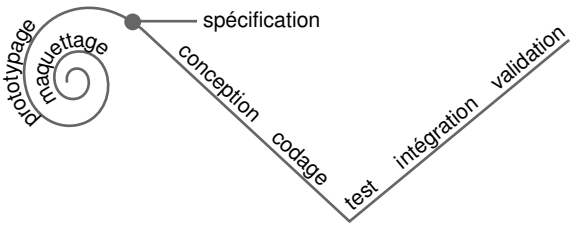

Figure 1: La production de systèmes interactifs : spirale puis cycle en V

faire déboucher un cycle de conception itérative sur l'entrée d'un cycle de développement en V. Pour eux, c'est la mise en œuvre optimale de ce cycle qui pose problème, avec par exemple les questions suivantes :

– comment produire à moindre coût des spécifications en sortie de la conception utilisables par le cycle en V ?
– comment assurer que les spécifications soient réalisables avec les outils utilisés en production ?
– comment éviter que le travail de conception ne soit dénaturé par les programmeurs du produit final (logiques de comportement, dessins, etc) ?
– comment réutiliser en production des ressources développées en conception ?

Toutes ces questions peuvent se synthétiser ainsi : *comment assurer la continuité du cycle ?* Les principaux points de rupture à ce jour sont le raccord entre le cycle itératif et le cycle en V, et dans une moindre mesure le passage des prototypes basse-fidélité à des prototypes fonctionnels au cours du cycle itératif.

Par ailleurs, les projets prévoient encore des périodes très courtes pour la conception. Plutôt que de prêcher pour un allongement des délais, il est plus efficace de chercher des solutions pour raccourcir les temps de prototypage, par exemple en parallélisant le travail.

Enfin, les industriels sont conscients de l'intérêt de disposer des acteurs compétents pour chaque facette du travail : programmeurs, ergonomes, ou graphistes quand l'interface est visuelle. Leur problème est plus de gérer efficacement la relation avec ces divers acteurs, voire de pouvoir sous-traiter certaines de ces activités à l'extérieur de l'entreprise. Il faut trouver des méthodes et des outils qui le permettent, et donc comprendre les besoins de chacun.

Les graphistes

Depuis le début des années 1990, certains chercheurs prédisaient que l'interface graphique ferait à terme plus appel aux designers ou aux graphistes qu'aux programmeurs. C'est chose faite pour le Web et les jeux. Avec des outils comme Flash et Director en particulier, on voit des graphistes réaliser des applications seuls, sans intervention d'informaticiens. Dans le monde de l'informatique plus traditionnelle, malgré des exemples comme Mac OSX ou plus modestement DigiStrips [13], le rôle du designer ou du graphiste est moins visible. Pourtant, quelques industriels ont déjà inclus des designers dans leurs équipes de conception (en particulier dans le domaine automobile), et

de nombreux designers affirment leur capacité à prendre en charge une partie importante de l'interface graphique. Ce qui les en empêche à ce jour semble essentiellement être une question d'outils et de processus. Deux processus sont courants :

– le graphiste travaille sur la conception visuelle globale du produit, livre son travail sous forme de papier ou d'images, qui servent aux programmeurs à réaliser le prototype ou le produit ;
– le graphiste sous-traite des parties de l'interface graphique finale, qu'il doit fournir sous forme d'images. L'exemple des boutons est assez courant.

Les graphistes perçoivent ces processus comme contraignants et limitant la portée de leur intervention. Les limitations les plus fréquemment mentionnées sont :

– *les limites graphiques des outils de prototypage et de production* : il leur manque des notions comme la transparence, les dégradés, une gestion évoluée du texte, etc. Cela contraint le mode d'expression ;
– *les formats d'échange imposés* : certains graphistes travaillent en dessin vectoriel et doivent livrer des images en pixels ;
– *le prix de réplication du travail graphique par le programmeur* : il demande souvent beaucoup de temps ;
– *l'intervention du programmeur* : le programmeur ne prête pas toujours une attention suffisante aux détails et il dégrade l'intervention du graphiste ;
– *la difficulté à décrire le comportement dynamique de l'interface* : s'il doit intervenir sur la dynamique de l'interface, le graphiste ne peut le faire qu'en donnant des explications au programmeur, ou par des maquettes de type Flash, sans grande garantie de fidélité à la fin.

Enfin, notre expérience montre que l'activité créative des designers et des graphistes demande un temps de maturation sur un projet. La création d'une dépendance initiale entre le travail des informaticiens et les productions des graphistes n'est donc pas une excellente idée.

Les ergonomes

Les ergonomes sont des spécialistes de l'humain et de l'utilisabilité plutôt que de la réalisation du système. Ils interviennent donc surtout en amont de la conception par des recueils de besoins et une analyse de l'activité et de la tâche, pendant la conception en rappelant des règles de conception ou en choisissant une philosophie d'interaction pour le produit, et à la fin de chaque étape sous forme d'évaluation. Néanmoins en fonction des tâches qui leur sont confiées dans les projets, les ergonomes ont souvent besoin d'intervenir concrètement dans la réalisation :

– *pour la rédaction des spécifications*, en fin de processus itératif de conception. Leurs spécifications doivent conduire à la réalisation du logiciel, et donc doivent refléter sa structure ;
– *pour la définition des logiques de comportement des interacteurs*, par exemple dans l'industrie automobile. Ils se retrouvent alors dans la même situation que les graphistes par rapport aux programmeurs ;
– *pour paramétrer et tester des variantes des prototypes réalisés* : certains définissent en amont du prototypage l'ensemble des paramètres sur lesquels ils veulent intervenir, de manière à ce que le programmeur les définisse dans le prototype. Dans la même ligne, il semble qu'une tendance chez les jeunes diplômés soit de maîtriser le langage CSS de paramétrage des sites Web.

Les programmeurs

Les programmeurs sont aujourd'hui incontournables dans la plupart des projets de conception, et sont les principaux acteurs dans la phase de réalisation qui suit. Ils souffrent de l'écart entre la tâche qu'ils ont à réaliser et les outils à leur disposition. Comme cela est noté par de nombreux chercheurs [2, 11, 14], les abstractions fournies par les langages et les boîtes à outils du commerce ne correspondent pas à celles manipulées dans les projets. Mais cette analyse correspond à une minorité de programmeurs : *le programmeur expérimenté et spécialiste en interaction homme-machine*, qui rêve d'un langage adapté à son travail. En pratique il existe de nombreux autres types de programmeurs dans les équipes, parmi lesquels :

– *l'informaticien assembleur* : il traite l'interface comme un assemblage visuel de composants, en dehors de toute programmation (environnements type XFaceMaker ou JavaBeans) ;
– *le programmeur expert d'un langage* : il cherche des outils compatibles avec le modèle et la syntaxe de son langage ;
– *l'informaticien spécialiste d'un domaine* (le contrôle aérien, la capture de données, etc) : il utilise un outil d'IHM spécifique de son domaine, VAPS par exemple ;
– *le spécialiste du génie logiciel* : il structure son logiciel en composants ; il a du mal à gérer l'articulation entre la structuration en composants interactifs et la structuration en composants logiciels.

Il est donc difficile d'analyser *la* tâche du programmeur. De plus, la pratique de celui chargé du cycle en V est très différente de celle de l'informaticien chargé du prototypage : plate-formes, langages et méthodes de travail diffèrent. Fournir un outil commun est donc une gageure.

Variabilité des situations

Nous avons observé une grande variabilité des équipes et des situations de travail. On peut noter en particulier :

– *la composition de l'équipe*, qui varie du programmeur ou du graphiste seul jusqu'à l'équipe complète avec un spécialiste de chaque domaine et un chef de projet ;
– *le coordinateur*. Qui dit travail d'équipe dit coordination, en général par la personne chargée des choix de conception. Elle a la difficile tâche de trouver un cadre commun pour organiser la production du prototype. Ce rôle est souvent assuré par l'ergonome ou par un ingénieur chargé de la supervision de l'équipe, mais parfois encore par un programmeur ;
– *la complexité du produit*. Selon les domaines et les phases de la conception, les prototypes vont d'objets simples réalisables avec un outil de dessin ou une page Flash jusqu'à un logiciel de plusieurs milliers de lignes connecté à un simulateur. A chaque niveau de complexité doit correspondre une architecture appropriée.

Il est donc important que les outils soient très flexibles dans la répartition des tâches et dans les modes de fonctionnement qu'ils supportent.

CONSEQUENCES SUR L'ARCHITECTURE

Les usages que nous venons de décrire et que nous cherchons à supporter posent des problèmes nouveaux en termes d'architecture des logiciels. En effet, jusqu'à présent on considérait que la production de logiciel est affaire de programmeurs, et les outils s'adressaient à telle ou telle catégorie de programmeurs. La seule exception à cette règle était représentée par les outils d'assemblage de composants tels JBuilder ou VisualStudio, ce qui explique probablement le succès des interfaces WIMP. La situation que nous décrivons amène selon nous les choix d'architecture suivants :

Architecture à base de modèles

Permettre la production d'éléments de logiciel par des non-programmeurs est un vieux rêve des chercheurs en IHM. Il implique d'une manière ou d'une autre la manipulation de données modélisées en lieu et place de la création de code, dans la mesure où les outils mis à la disposition des non-programmeurs doivent pouvoir écrire mais aussi relire les éléments produits. La version moderne de cette approche est l'architecture à base de modèles, où le programme est défini par un assemblage d'instances de modèles : objets graphiques, ou machines à états par exemple. Elle revient à considérer que la programmation, tout comme la manipulation d'outils graphiques, consiste à instancier et paramétrer des objets fournis par un modèle. Dans le cas de la programmation, cette réalité est masquée par l'utilisation d'une syntaxe "naturelle pour un programmeur", mais il suffit de considérer l'arbre abstrait d'un programme pour y retrouver les concepts de base présents dans les autres types de modèles.

Ici, le principe consiste à faire manipuler les mêmes modèles par les différents acteurs, à travers des interfaces spécialisées : outils graphiques pour les uns, APIs ou langages pour les autres. Les modèles en jeu sont tous ceux mis en oeuvre par les systèmes interactifs : objets graphiques, comportements, contraintes géométriques, etc. Une telle approche ne peut être que progressive, dans la mesure où beaucoup d'aspects de la programmation d'interfaces ne sont pas encore modélisés.

Utilisation de XML

Certains acteurs du processus disposent déjà d'outils adaptés pour faire leur travail. C'est le cas des graphistes avec Adobe Illustrator ou Corel Draw par exemple. A l'avenir, il est probable que des outils équivalents seront disponibles pour les spécialistes d'autres modalités (son, voix, etc). Il est important d'utiliser des standards pour échanger des modèles, pour permettre à chaque métier de conserver ses meilleurs outils tout en collaborant avec les autres. Le format XML s'impose aujourd'hui, en particulier grâce à l'existence du format SVG (Scalable Vector Graphics), disponible en sortie de la plupart des outils de dessin professionnels. Il n'existe hélas pas de format XML pour les machines à états, les StateCharts ou les réseaux de Petri utilisés dans la description de comportements discrets, mais notre opinion est qu'il faudrait les créer.

Unification de l'architecture

L'architecture d'un logiciel reflète la nature des tâches assurées par le logiciel, mais surtout le cycle de vie de ce logiciel : quelles parties sont destinées à être réutilisées ou modifiées, par exemple. Une architecture est un outil de travail pour une communauté, par exemple pour des programmeurs qui veulent se partager le travail. Or les programmeurs d'interfaces sont aujourd'hui confrontés à deux types d'architectures : la gestion de projets logiciels d'une part, la conception de systèmes interactifs d'autre part. Les cycles de vie étant différents, il n'est pas surprenant que les architectures le soient. Pour raccorder les cycles, il nous semble important de proposer des architectures unifiées tenant compte des besoins des deux phases du cycle unifié. Or les langages de programmation eux-mêmes reflètent des choix d'architecture. Le rôle central donné aux appels de fonctions dans les langages classiques, y compris pour la compilation séparée, en est un exemple. Pour obtenir des architectures unifiées, il faudra peut-être repenser certains aspects des langages, à l'image de C# qui introduit la notion d'abonnement à un événement.

INTUIKIT

Pour mettre en oeuvre ces principes, nous développons un environnement de programmation nouveau appelé IntuiKit et destiné à chacun des acteurs d'un projet. Le noyau d'IntuiKit est bâti sur un ensemble restreint de concepts pour créer des composants à partir de la définition de structures arborescentes. Ce type de structure est adapté à la création de nombreuses architectures logicielles, comme PAC ou MVC, il est facilement représentable dans un langage XML, et il est facile à dessiner schématiquement. La définition de l'architecture du composant sert de squelette pour intégrer les différentes facettes représentées par les modèles additionnels. Chaque modèle additionnel correspond à une modalité ou à une facette de la description des composants. lls sont si possible construits à partir de standards du domaine de chacun des acteurs, comme SVG pour le graphiste, ou des fichiers de configuration CSS.

Le noyau d'IntuiKit et les modèles additionnels sont accessibles uniformément par des langages de programmation ou par des formats XML, de façon à ce que chaque acteur puisse utiliser ses outils favoris : code pour les uns, outils de dessin (Illustrator par exemple) pour les autres, ou encore futurs éditeurs graphiques de modèles.

La philosophie générale d'IntuiKit est résumée par la figure 2 : au cours d'un processus de conception itératif, les modèles produits par les acteurs d'un projet sont échangés et intégrés sur le modèle de l'architecture programmé avec le noyau d'IntuiKit. Suivant les compétences des acteurs, l'application est écrite directement avec un langage de programmation, avec des modèles XML, dans le futur avec un environnement d'édition graphique pour rendre les modèles plus accessibles, ou bien avec un mélange des trois. Les fichiers de paramètres servent à tester de petits ajustements de l'interface, tandis que des changements plus profonds dans l'apparence ou dans le comportement nécessitent une nouvelle itération.

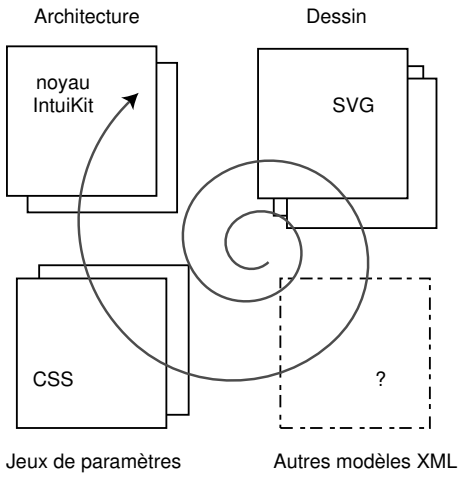

Figure 2: Conception itérative avec IntuiKit

Pour le programmeur, IntuiKit se présente comme une boîte à outil de programmation d'interfaces. Dans la suite de cette section nous illustrons quelques principes de mise en œuvre de cet outil sur un micro-projet de réalisation d'un bouton à deux états, en utilisant l'API Perl d'IntuiKit. Afin d'illustrer plusieurs types de cycles de réalisation, nous proposons trois scénarios différents.

Scénario 1 : programmation traditionnelle

Dans le premier scénario, Etienne, un ergonome, a demandé à Paul, un programmeur, de lui réaliser le bouton en suivant des spécifications relativement précises. Etienne souhaite un bouton rectangulaire avec un bord en relief, du texte centré et un fond de couleur uniforme. Il souhaite aussi tester différentes couleurs de fond pour la lisibilité du texte, et il souhaite paramétrer le texte du bouton affiché dans chaque état ainsi que les dimensions du bouton et la police de caractère. Les spécifications d'Etienne sont donc une liste de constituants du bouton, qu'il appelle respectivement bouton, fond, texte1, texte2 et une liste de paramètres pour ces constituants, respectivement police, hauteur, largeur pour le bouton, couleur pour le fond et texte pour texte1 et texte2. Ces spécifications sont directement transposables en IntuiKit sous la forme d'une architecture pour le bouton et d'un fichier de ressource qui contient les paramètres.

Paul choisit de réaliser le bouton de Etienne avec une API Perl. En partant de la spécification de Etienne, il commence par créer l'architecture du bouton à partir de la liste de ses constituants :

```
$b = new Component (—id => bouton);
$f = new GUI::Rectangle (—id => fond, —parent => $b);
$t1 = new GUI::Text (—id => texte1, —parent => $b);
$t2 = new GUI::Text (—id => texte2, —parent => $b);
$fsm = new FSM (—parent => $b, ... );
```

Le composant bouton est un arbre, l'option "-parent" définit les liens de parenté. Le nœud racine est un nœud abstrait (Component) ; celui-ci contient des nœuds qui représentent les facettes graphiques du composant (GUI : :) et des nœuds qui représentent le comportement (FSM). Nous ne développons pas ici les facettes comportementales. Ce type de nœud sert à lier les états d'un automate avec l'affichage d'un ou plusieurs nœuds graphiques associés à chaque état.

Dans un deuxième temps, Paul déclare les paramètres de son composant bouton de façon, entre autres, à ce que IntuiKit sache lire un fichier de configuration :

```
$b—>define (—name => 'police');
$b—>define (—name => 'largeur);
$b—>define (—name => 'hauteur);
$f—>define (—name => 'couleur');
$t1—>define (—name => 'texte1');
$t2—>define (—name => 'texte2');
```

Finalement, Paul doit initialiser les composants avec les valeurs à l'exécution de leurs paramètres. C'est à ce moment qu'il configure les objets graphiques avec leurs polices, couleurs, dimensions, relief et autres caractéristiques. Dans notre exemple, il le fait à l'aide d'un fichier de configuration :

```
$rsc = load Resource ('bouton.css');
$bouton—>apply ($rsc);
run ();
```

Le format de configuration suit la syntaxe des feuilles de style CSS étendue avec l'utilisation d'expressions inspirées des chemins XPath dans les sélecteurs des règles. Ces sélecteurs servent à désigner des nœuds de l'arbre du composant :

```
bouton {
  police: 'helvetica—12';
  largeur: 150;
  hauteur: 40;
}
bouton/fond {
  couleur: yellow;
}
bouton/texte1 {
  texte: 'ON';
}
bouton/texte2 {
  texte: 'OFF';
}
```

Le bouton réalisé est un bouton à l'apparence plutôt rudimentaire présenté dans ses deux états sur la figure 3.

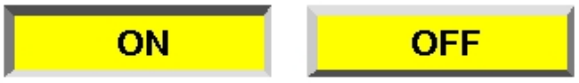

Figure 3: Boutons réalisés par Paul

Scénario 2 : conception avec intégration

Dans le second scénario, Etienne préfère laisser le soin de la conception du bouton à Gaëlle, la graphiste. Il lui indique seulement quels doivent être les textes et la taille du bouton. Il lui demande également de produire des variantes du bouton à l'esthétique différente. Dans ce cas, le

travail de Paul va consister à intégrer le graphisme dans l'architecture du bouton. Cette opération est simplifiée par le fait que les dessins en SVG sont directement intégrés dans les modèles du noyau d'IntuiKit. Il suffit donc à Paul et à Gaëlle de s'entendre au préalable sur la structuration du fichier SVG, pour que Paul puisse récupérer les éléments dont il a besoin pour contrôler l'aspect graphique du bouton avec un automate. En pratique, la structure du fichier reflète celle des composants, et il leur suffit de se mettre d'accord sur les noms donnés par Gaëlle aux calques et groupes : 'fond', 'on', 'off', 'masque".

```
$b = new Component( —id => bouton );
$look = load Component ( —id => 'dessin',
        —parent => $b, —file => 'bouton.svg' );
$f = $look—>find ('.//fond');
$on = $look—>find ('.//on');
$off = $look—>find ('.//off');
$m = $look—>find ('.//masque');
new FSM (—parent => $b, ... );
```

La fonction "find" sert à récupérer un nœud nommé d'un arbre à partir de son chemin d'accès, avec une expression XPath étendue. Le code pour afficher le bouton précédent est similaire à celui du scénario précédent. Un exemple de bouton réalisé par Gaêlle est présenté sur la figure4

Figure 4: Boutons réalisés par Gaëlle

Scénario 3 : conception pour la réutilisation
Finalement, dans un troisième scénario, Etienne souhaite faire appel aux services de Gaëlle, mais en plus il souhaite également définir des paramètres pour modifier le rendu du dessin. Par exemple Etienne souhaite comme dans le premier scénario pouvoir modifier le texte affiché dans chaque état. Pour cela, Gaëlle et Paul doivent s'entendre sur les identificateurs à donner aux éléments de texte dans le SVG, par exemple "texte1" et "texte2". Paul peut alors déclarer deux paramètres de même nom pour le composant bouton, de manière similaire au scénario 1. Une fois que Paul a récupéré des références aux nœuds de texte du SVG via leur identificateur, comme dans le scénario 2, un mécanisme d'IntuiKit lui permet de fusionner les valeurs des attributs du SVG avec les paramètres déclarés pour le bouton.

Le scénario 3 permet d'obtenir une interface configurable à la fois par un jeu de paramètres et par un thème graphique pris dans un fichier SVG. IntuiKit possède également des mécanismes, sortant du cadre de cet article, pour transformer le bouton obtenu en un modèle de classe de composant bouton pouvant être instancié dans d'autres interfaces.

EXEMPLE D'APPLICATION

Nous avons pu tester les principes décrits plus haut sur des cas concrets, avec différentes configurations d'équipes de conception. De plus, deux projets d'ampleur et de complexité similaire menés à un an d'intervalle nous ont permis de mesurer les apports de l'architecture que nous avons choisie et du processus associé, par rapport à une approche plus traditionnelle de la programmation.

Le premier projet était la réalisation d'une interface de type "groupware" dans lequel le design graphique jouait un rôle central : il s'agissait d'afficher une très grande quantité d'information sur un écran destiné à être regardé depuis une distance de 4 à 5 mètres. Le second projet était un ensemble de deux interfaces de type collecticiel, destinées à la gestion du trafic aérien au sol et la coordination entre les contrôleurs de sol et les contrôleurs de tour d'un aéroport. Là aussi, le graphisme jouait un rôle important, cette fois-ci pour des raisons commerciales et d'acceptabilité par les contrôleurs.

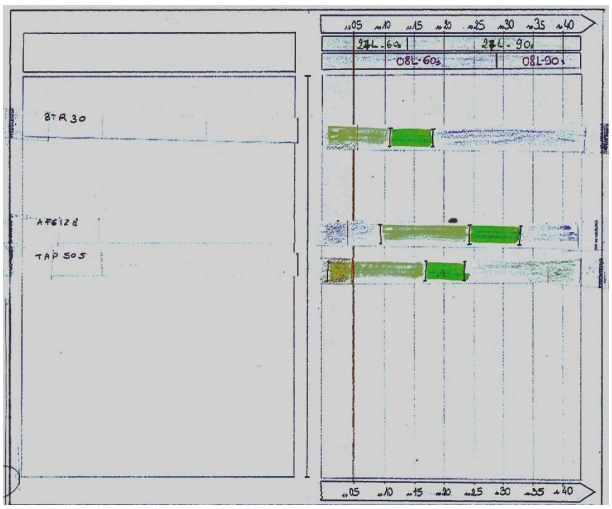

Figure 5: L'esquisse utilisée pour structurer le travail en composants

Les deux projets ont représenté un effort de travail comparable pour les graphistes, et un effort de programmation d'interface similaire (hors graphisme). Le premier projet a été réalisé selon un processus traditionnel : production des visuels par le graphiste, puis recodage par le programmeur avec la bibliothèque TkZinc du CENA. Il était facilité par la participation du graphiste à la tâche de recodage. Le second projet a été réalisé comme suit. Tout d'abord, le travail de conception préliminaire a été réalisé. Il a donné lieu pour chaque interface à la production d'une esquisse comme celle de la figure 5. Cette esquisse a permis à la responsable de la conception de définir un ensemble de composants à réaliser, et de définir entre les acteurs une structure et un ensemble de noms.

A partir de cette étape, le travail a été parallélisé entre le designer graphique et les programmeurs d'interfaces. Le premier s'est consacré au choix d'une ambiance, puis à la réalisation des composants. Les seconds se sont consacrés à la réalisation de la structure et des comportements, en se basant sur des dessins primitifs réalisés par eux en code. Le résultat de leur travail sur l'un des deux programmes est illustré par la figure 6. A cette étape, les programmes

étaient totalement fonctionnels, aux mises au point près.

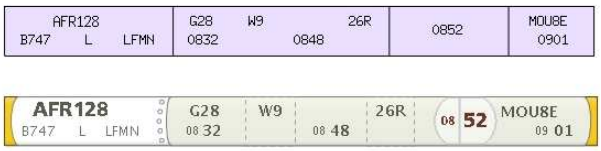

Figure 6: Les strips avant et après l'intégration du travail graphique

Enfin, environ deux semaines avant la livraison finale du produit, le graphiste a livré les visuels au format SVG en respectant le schéma de nommage défini. Les visuels ont été incorporés par les informaticiens en lieu et place des leurs, sans autre modification du logiciel (à l'exception des objets graphiques répétitifs de la grille de droite, la notion de disposition géométrique n'étant pas encore gérée par les modèles de l'atelier). Le résultat final sur l'un des programmes est représenté à la figure 7.

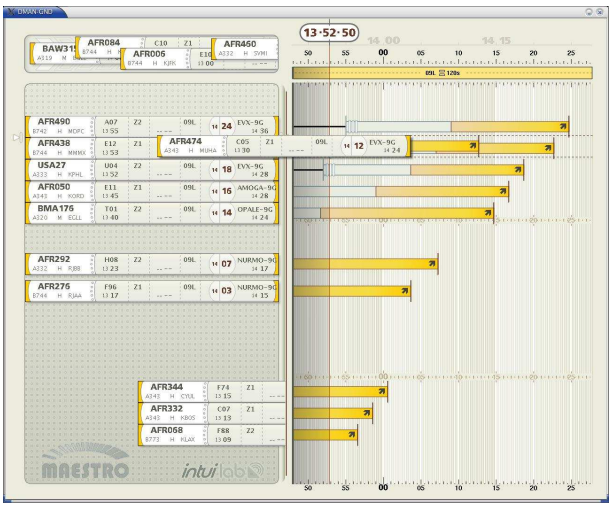

Figure 7: L'interface finale

Les deux projets étaient de taille et de complexité similaire. Le premier projet a été réalisé sur une période de trois à quatre mois, et le second sur une période de deux mois par une équipe de taille similaire. L'architecture choisie a non seulement permis de réduire au minimum l'effort d'intégration graphique, mais elle a aussi permis de retarder au maximum la date de livraison du travail graphique. Enfin, elle permet au client du logiciel de demander des évolutions visuelles sans faire intervenir de programmeur. Les personnes en charge du projet considèrent que cette architecture a été déterminante dans le succès du projet, dont les délais étaient extrêmement stricts et réduits.

TRAVAUX SIMILAIRES

Cette problématique d'ingénierie des systèmes interactifs d'ingénierie des systèmes interactifs a été abordée de multiples manières depuis le milieu des années 1980 par l'ingénierie des systèmes interactifs. Notre approche possède des similitudes avec de nombreux travaux dans ce domaine, ainsi qu'avec des travaux en génie logiciel pur ou dans la communauté du Web et du XML. Notre originalité réside dans la manière de combiner les notions utilisées autour d'un noyau d'atelier logiciel. On peut regrouper les travaux similaires selon les thèmes suivants :

Les processus. Le problème de la compatibilité entre les processus de conception de systèmes interactifs et de génie logiciel a été clairement identifié depuis quelques années et fait l'objet de travaux spécifiques. Des solutions ont par exemple été proposées pour étendre le Rational Unified Process [16]. Ces études traitent rarement de la phase de conception et de prototypage.

Les approches à base de modŁles. La réalisation d'interfaces à base de modèles est un domaine très actif depuis quelques années. Souchon et al [15] et da Silva [4] font un bilan des recherches récentes dans le domaine. Ces travaux s'inscrivent le plus souvent dans un objectif différent du nôtre : produire automatiquement des interfaces à partir de modèles de la tâche [5] ou produire des interfaces plastiques [17], capables de s'adapter à de multiples plate-formes. A ce jour, ils se limitent le plus souvent à des interfaces WIMP ou plus simples. Plusieurs tentatives sont en cours pour standardiser les modèles par des langages XML comme UIML ou XIML.

L'ingénierie des systŁmes interactifs. Les considérations de génie logiciel sont au coeur des recherches en ingénierie de l'interaction depuis près de vingt ans. La problématique initiale était de rendre la partie interactive indépendante de la partie fonctionnelle, car cela correspondait à des scénarios de réingénierie des systèmes. Les scénarios que nous traitons sont en partie différents, mais il existe une nette filiation avec les architectures MVC [6] et PAC [3]. Notre approche, bien que très concrète, est conçue pour intégrer progressivement les travaux issus de modélisation de diverses facettes des systèmes interactifs : flots de données [10], comportements discrets [9], géométrie, animation, etc.

La conception des langages. La conception des langages de programmation est une activité de plus en plus rationnalisée, et certains concepteurs justifient désormais leurs choix par des critères d'usage voire d'utilisabilité plus que par des propriétés formelles [20, 18]. Néanmoins, les langages restent encore fortement influencés par la partie non interactive des logiciels, et concentrés sur l'activité des seuls programmeurs.

Programmation par les non-informaticiens. La programmation visuelle et la programmation par les utilisateurs font l'objet d'une conférence annuelle (HCC), et des outils comme Hypercard [7] ou Director [12] sont des précurseurs célèbres en matière de programmation d'interfaces. Cependant, ces outils sont en général focalisés sur une tâche donnée et s'intègrent difficilement dans des processus de génie logiciel.

Les outils du Web. Autour du W3C, de nombreux travaux portent sur la modélisation systématique de données ou de contenus en XML pour les échanger sur un réseau. Le standard SVG [19] est issu de cette approche, et de multiples extensions y sont proposées pour décrire le comportement ou des données. Par ailleurs, divers lan-

gages XML de description d'interfaces (souvent WIMP) sont proposés : XUL, UIML, MXML, XDP, XAML par exemple. Ils permettent d'intégrer au sein d'un document XML des composants dont le code est réalisé par ailleurs. Notre approche est similaire, mais plus ambitieuse : nous proposons de construire les objets interactifs et pas seulement de les réutiliser.

CONCLUSION ET PERSPECTIVES

Notre étude et notre collaboration avec des équipes industrielles nous ont permis de vérifier l'existence d'un fossé entre les méthodes de conception de systèmes interactifs et les processus de génie logiciel, en particulier en termes d'outils. Pour combler ce fossé, il faut fournir des outils et des méthodes qui permettent de gérer de manière continue le passage du cycle itératif au cycle en V, et de gérer efficacement le travail des équipes pluridisciplinaires de conception et de prototypage. Une architecture à base de modèles nous semble permettre de mettre en place de tels processus, tout en laissant à chaque acteur du développement des outils adaptés à son travail et à ses compétences.

Nous avons réalisé l'atelier IntuiKit selon ce choix d'architecture. A ce jour, IntuiKit se présente sous forme de bibliothèques Perl avec une API adaptée à ce langage, et permet de décrire des interfaces à base d'objets graphiques et de comportements discrets. Nous travaillons actuellement à poursuivre les recherches dans les directions suivantes :

– conception d'API adaptées à d'autres langages et aux styles de programmation associés ;
– incorporation de modèles issus de la recherche en ingénierie des interfaces, afin de les rendre manipulables par les outils d'IntuiKit ;
– étude des concepts des langages de programmation mis en oeuvre dans la programmation des interfaces, en vue de les ajouter aux modèles gérés dans IntuiKit.

REMERCIEMENTS

Les techniques d'interaction et le rendu SVG évoqués dans cet article sont issues du CENA, sans qui IntuiLab n'existerait pas. Merci en particulier à P.Lecoanet et C.Mertz. Les analyses et réalisations décrites doivent beaucoup à toute l'équipe d'IntuiLab, en particulier D.Figarol, L.Karsenty et C.Schlienger sur les observations des équipes de conception, et C.Schlienger et S.Valès pour leur travail pratique de conception et de réalisation. Merci à Sofréavia pour le projet DMAN, à Robert Parise et Yves Rinato pour leur participation à notre travail d'analyse de besoins.

BIBLIOGRAPHIE

1. J. Accot, S. Chatty, Y. Jestin, and S. Sire. Conception des interfaces : et si nous analysions enfin la tâche du programmeur ? In *Actes d'IHM98*, 1998.
2. M. Beaudouin-Lafon. Designing interaction, not interfaces. In *Proceedings of AVI2004*, 2004.
3. J. Coutaz. *Interfaces homme-ordinateur, conception et réalisation*. Informatique, Dunod, 1990.
4. P. P. da Silva. User interface declarative models and development environments : A survey. In *DSV-IS 2000*, pages 207–226, 2000.
5. J. Eisenstein, J. Vanderdonckt, and A. Puerta. Adapting to mobile contexts with user-interface modeling. In *Third IEEE Workshop on Mobile Computing Systems and Applications*, pages 83–92, 2000.
6. A. Goldberg and D. Robson. *Smalltalk-80 : the language and its implementation*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, 1983.
7. D. Goodman. *The Complete HyperCard Handbook*. Bantam Books, 1987.
8. T. Green. The nature of programming. In J. M. Hoc et al, editor, *Psychology of Programming*. 1991.
9. D. Harel. Statecharts : A visual formalism for complex systems. *Science of Computer Programming*, 8(3) :231–274, June 1987.
10. R. Jacob, L. Deligiannidis, and S. Morrison. A software model and specification language for non-WIMP user interfaces. *ACM Transactions on Computer-Human Interaction*, 6(1) :1–46, 1999.
11. E. Lecolinet. XXL : A dual approach for building user interfaces. In *Proceedings of the ACM UIST*, pages 99–108, 1996.
12. MacroMedia Press. *MacroMedia Director 6 and lingo authorized*. Addison-Wesley, 1997.
13. C. Mertz and J.-L. Vinot. Conception par maquettage rapide : application à des écrans tactiles pour le contrôle aérien. In *Actes de la conférence Ergo-IA'98*, Nov. 1998.
14. B. A. Myers. Authoring interactive behaviors for multimedia. In T. Ishiguro, editor, *Proceedings of the 9th NEC Research Symposium : The Human-Centric Multimedia Community*, 1998.
15. N. Souchon and J. Vanderdonckt. A review of xml-compliant user interface description languages. In *Proceedings of DSV-IS 2003*, pages 377–391. Springer-Verlag, 2003.
16. K. S. Sousa and E. Furtado. An approach to integrate HCI and SE in requirements engineering. In M. Borup Harning and J. Vanderdonckt, editors, *Proceedings of the IFIP TC13 workshop on Closing the gaps : Software engineeering and Human-Computer Interaction.*, 2003.
17. D. Thévenin and J. Coutaz. Plasticity of user interfaces : framework and research agenda. In *Proceedings of INTERACT'99*. IOS Press.
18. B. Venners and B. Eckel. The C# design process. a conversation with Anders Hejlsberg. http ://www.artima.com/intv/csdes.html.
19. W3C Recommendation 14 January 2003. *Scalable Vector Graphics (SVG) 1.1 Specification*. http ://www.w3.org/TR/SVG11/, 2003.
20. L. Wall. The culture of Perl. Transcription of the keynote address at the Perl Conference in 1997, http ://www.perl.com/pub/a/1997/wall/keynote.html.