Réconcilier conception d'interfaces et conception logicielle : vers la "conception orientée-systèmes"

Stéphane Chatty

Réconcilier conception d'interfaces et conception logicielle : vers la « conception orientée-systèmes »

Stéphane Chatty

Université de Toulouse - ENAC
7 avenue Edouard Belin, 31055 Toulouse Cedex, France
chatty@enac.fr

RÉSUMÉ

Conception d'interfaces et conception logicielle ont des liens complexes, que la conception par objets a échoué à clarifier. Nous analysons ici ces rapports et proposons une voie pour les clarifier. L'échec de la conception par objets provient de son lien avec le modèle de programmation par objets, probablement très éloigné de ce qu'imaginait Alan Kay, dans lequel les échanges entre objets sont stéréotypés et décrivent mal les interactions observées entre objets physiques, entre humains, et entre humains et appareils numériques. Appliquer ce modèle à l'IHM restreint le pouvoir d'expression des concepteurs, et tend à opposer des activités de conception qui devraient n'en constituer qu'une. En s'appuyant sur le modèle de programmation par composants interactifs, appuyé sur un modèle théorique de processus en interaction, il devient envisageable de disposer d'un ensemble commun de concepts de base. Dans ce cadre, concevoir revient à analyser, modéliser ou définir des systèmes et leurs interactions, et spécifier revient à imposer des contraintes aux systèmes restant à concevoir. Conception logicielle et conception d'interfaces peuvent alors reposer sur un socle commun de méthodes et d'outils, tout en définissant des interactions entre systèmes de nature différentes : entre composants logiciels d'une part, entre logiciel et système cognitif et perceptif d'autre part. Cela ouvre aussi la voie à la conception de systèmes homme-machine plus complexes, intégrant procédures, formation et interaction entre humains, ainsi qu'à de nouvelles approches du partage de responsabilités entre humain et automatisme.

ABSTRACT

Object-oriented design has failed to avoid a rift between interface design and software design, because the messages used in object-oriented poorly describe the interactions between between users and their environment. By relying on process theories and interactive component programming, one can propose a common language between designers : that of system-oriented design, in which they all describe interfaces between systems and design new systems.

Author Keywords: design, specification, model, interface, object, process, component, system, interaction

ACM Classification Keywords: H5.2 [Information Interfaces and presentation] User Interfaces—GUI ; D2.11 [Software engineering] Software Architectures.

General Terms: Design ; Human factors ; Language

INTRODUCTION

Grâce à la popularisation du design d'interaction et du « UX design », le métier de concepteur d'interfaces homme-machine est désormais reconnu dans la profession informatique. Il n'en a pas toujours été ainsi, et la conception d'interfaces elle-même a parfois été mise en cause en tant qu'activité de conception. Historiquement, en matière de logiciel ce terme était réservé à la définition de la structure des programmes afin de répondre à une spécification [1], activité qui a donné naissance au génie logiciel. La conception d'interfaces est née des besoins que ce domaine ne couvre pas, d'abord sous forme de réglages finaux (couleurs et paramètres divers), puis progressivement comme une étape préalable à la spécification.

C'est ainsi qu'ont émergé deux tâches et deux communautés de conception, chacune protégée des problèmes et des méthodes de l'autre par un document de spécification. Hélas, comme le souligne la difficulté à définir un format de spécification qui convienne à tous, cette organisation du travail n'est pas optimale. Tant que l'espace de conception est étroit, il est possible de trouver une organisation et un langage de spécification qui fonctionnent. Mais dès qu'il s'agit de système complexe ou d'interaction riche, il est difficile de définir une organisation efficace, tant les choix effectués de part et d'autre deviennent interdépendants. La « conception du logiciel » que Winograd appelait de ses voeux [21] reste un objectif à atteindre en tant qu'activité intégrée et organisée.

Cet article esquisse un socle conceptuel unifié, afin de contribuer à l'harmonisation des processus d'ingénierie des systèmes homme-machine. J'y constate d'abord qu'il n'y a pas deux, mais plusieurs métiers de conception. Je distingue ensuite les aspects organisationnels, qui reposent sur des frontières entre rôles, et la nature des concepts manipulés, qui doivent reposer sur des bases communes. J'analyse alors en quoi les méthodes de conception par objets ont échoué à proposer des concepts utilisables par tous. Je propose, en m'appuyant sur des recherches en modélisation des systèmes in-

1. qui n'est pas une expression de besoins, voir ISO/IEC 2382-20

teractifs, d'utiliser le concept de système comme base commune à toutes les activités de conception. Cette proposition est illustrée par des exemples d'utilisation du concept de système comme support à la description de solutions, voire à la résolution de problèmes, dans diverses phases de conception.

CONCEPTION D'INTERFACES, CONCEPTION LOGICIELLE

Histoire d'un divorce à l'amiable

Quand a été la dernière fois qu'un spécialiste en génie logiciel s'est exclamé « mais enfin, la conception c'est ce qui suit la spécification, pas ce qui la précède ! » ? Car en effet, selon les canons du génie logiciel, la conception générale et la conception détaillée suivent la spécification. C'est sur ce malentendu, qui mène à croire que la spécification est systématiquement la première étape d'un projet informatique, que se sont construits de nombreux échecs industriels depuis la généralisation des interfaces graphiques dans les années 1990. Cela a débuté lorsque des ingénieurs n'ont pas vu que lister des fonctionnalités ne constituait plus une spécification suffisante pour que des informaticiens produisent le système espéré. Cela a continué lorsque certains ingénieurs, pour réagir aux échecs croissants, ont confié le travail de spécification à des ergonomes en espérant que ces derniers sauraient imposer les bonnes contraintes aux informaticiens. Las, les méthodes du génie logiciel ne produisent pas plus de programmes utilisables à partir d'exigences d'utilisabilité qu'à partir de listes de fonctionnalités. C'est ainsi que de nombreux ergonomes se sont retrouvés face au défi de la conception, alors qu'ils étaient plutôt préparés à l'analyse de besoin, la formulation de critères de qualité et l'évaluation. La suite est connue : la conception d'interfaces est devenue une activité voire un métier à part entière, où se retrouvent ergonomes, ingénieurs, designers, et parfois même informaticiens.

Le schéma ci-dessous, imaginé par l'auteur en 1994 et utilisé avec succès dans des projets industriels [19], résume la situation résultante : une phase itérative pour la conception d'interfaces, suivie d'un cycle en V pour la conception logicielle. Il a le mérite de définir une frontière rassurante pour tous, tant en termes de compétences qu'en termes d'organisation et de responsabilités : les uns peuvent s'exonérer des risques liés au développement et des détails informatiques sans intérêt, les autres peuvent travailler dans un cadre industriel rassurant et éviter la fréquentation des utilisateurs, et tous peuvent se retrancher derrière les ambiguïtés de la spécification.

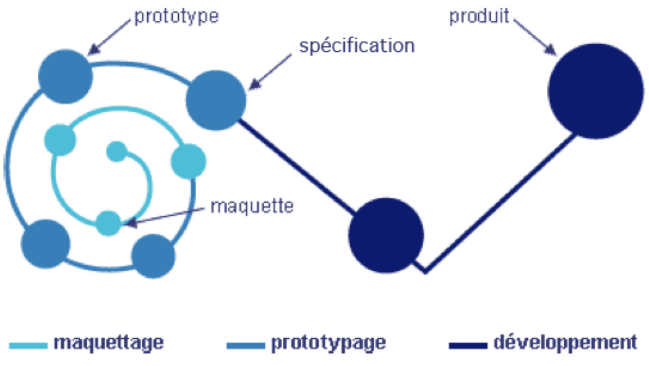

FIGURE 1 : Le cycle « en escargot », combinant spirale et V

Mais ce schéma peine à masquer les problèmes. C'est vrai en termes temporels d'abord ; il est souvent nécessaire d'itérer pendant le développement du logiciel, c'est d'ailleurs l'objet de RUP et des méthodes de développement agiles [7]. Mais alors, à quoi devrait ressembler le schéma et comment développeurs et concepteurs d'IHM doivent-ils travailler ensemble ? De même, pour les grands projets il est souvent impensable de concevoir totalement l'IHM avant de débuter le développement, et les deux activités doivent être parallélisables. C'est vrai en termes de contenu ensuite ; pourquoi tout ce qui relève de l'IHM devrait-il être itératif, et tout ce qui est logiciel traité par un cycle en V ? C'est vrai enfin en termes de risque ; la clé de voûte du schéma est la spécification, alors même que tous sont à la recherche de solutions complémentaires : storyboards ou maquettes dynamiques par exemple.

La situation n'est donc pas complètement satisfaisante, ce qui se traduit en coûts et en risques dans les projets d'ingénierie de systèmes interactifs. Le divorce est-il temporaire, ou tout au moins est-il possible de redéfinir les termes de la cohabitation ? C'est l'objet de la suite de cet article, en commençant par distinguer deux axes dans le découpage entre les activités.

Qui conçoit quoi ?

Depuis le début de cet article, plusieurs expressions ont été associées au mot « conception » : logiciel, produit, interface, interaction, IHM, système homme-machine, « user experience ». Toutes sont employées dans des contextes différents, mais il est difficile de clarifier leurs différences. A titre d'exemple, concevoir une interaction est-il totalement distinct de concevoir du code ? En apparence, sans doute ; mais dès qu'on produit une description concrète d'une interaction, sous forme d'états et de transitions par exemple, ce qu'on obtient est un programme. Ce dernier est exprimé dans un langage inhabituel pour un ordinateur, mais il suffit de le traduire pour le rendre exécutable ; c'est d'ailleurs l'une des bases de l'ingénierie dirigée par les modèles [4]. Prenons un second exemple : les cabinets de design conçoivent des véhicules, des meubles et maintenant des logiciels interactifs. Ils ont le même rôle dans tous les cas, et il est tentant de parler de conception de logiciel comme on parle de conception de mobilier. Mais alors on en est réduit aux subtilités grammaticales pour maintenir la distinction avec le génie logiciel : conception de logiciel d'un côté, conception logicielle de l'autre. Un troisième exemple est le travail des designers graphiques : de plus en plus, leur conception visuelle est faite en amont lors de la phase itérative, mais réutilisée directement dans le logiciel final. Ont-ils conçu du logiciel ou de l'interface ?

Cette confusion dans le vocabulaire est le reflet d'une réalité sous-jacente : s'ils interviennent sur des parties différentes, avec des compétences et des méthodes différentes, à des niveaux de granularité, de précision ou d'abstraction différents, c'est le même artefact que tous contribuent à définir. Il est donc logique que leurs productions interagissent, voire que les frontières varient selon les projets. Mais cette réalité est brouillée par la diversité des concepts utilisés : tâche, activité, processus cognitif pour les uns, interaction pour les autres, formes et contraste pour d'autres encore, fonctions, classes et patrons pour les derniers. Sans langage commun, il est difficile de discerner comment fonctionne la collaboration.

Qui conçoit, qui spécifie ?

Notons d'abord que si le vocabulaire du génie logiciel induit une distinction entre ceux qui rédigent une spécification et ceux qui doivent la respecter, les rôles et les métiers sont plus nombreux : architectes, programmeurs, ergonomes, designers, etc. Ces rôles sont rarement cantonnés à l'un des deux camps : les programmeurs peuvent être amenés à spécifier des choix techniques, les designers peuvent participer au développement du logiciel. La spécification est donc une frontière moins structurante qu'il n'y paraît au premier abord.

Pour tenter de comprendre cette frontière, revenons à l'origine du génie logiciel. Pourquoi a-t-il été possible d'affirmer que la spécification était la première étape du processus de conception d'un logiciel ? La spécification étant de toute évidence le résultat d'un travail intellectuel, il est cohérent de définir la conception comme un processus d'analyse et de choix et la spécification comme le résultat d'une conception initiale. La question devient alors : pourquoi a-t-on considéré comme triviale cette conception initiale ? Je propose de l'expliquer par la complexité et la nature des logiciels concernés. Pour un logiciel de calcul, on peut définir ce que l'on attend sans recourir à une analyse du besoin : c'est le résultat du calcul. Cela explique pourquoi il faut aujourd'hui enrichir les spécifications par des maquettes dynamiques : les informations à transmettre sont plus nombreuses et complexes dans le cas d'un système interactif, et même la notation UML et ses dérivées [3] ne suffisent pas à les capturer de manière humainement acceptable. En dehors du cas trivial du calcul, la spécification constitue une étape entre activités de conception.

Il y a plusieurs frontières candidates entre ces activités :

– entre le quoi et le comment ;
– entre l'intérieur (le système à produire) et l'extérieur (son environnement), l'interface homme-machine étant un cas particulier de cette frontière ;
– entre ce qui a déjà été décidé, et ce qui reste à décider.

La première frontière, bien que souvent utilisée pour enseigner l'analyse des besoins et la spécification, n'est guère utile dans le cas des logiciels interactifs. Tout d'abord, elle peut conduire à ignorer des résultats déjà obtenus dans la phase amont, par exemple les éléments visuels ; les maquettes dynamiques servent entre autres à compenser l'étanchéité de cette frontière. De plus, une interaction gestuelle peut relever du quoi ou du comment selon les projets. Cette ambiguïté est un facteur d'échec dans les grands projets, où les rôles du quoi et du comment peuvent s'intervertir en fin de projet quand les utilisateurs font valoir des exigences. La seconde frontière est plus naturelle, et elle s'applique tant au calcul qu'à l'IHM. Cependant l'interface homme-machine n'est pas toujours totalement définie dans la spécification, en particulier dans les grands projets. Qui plus est, plus un logiciel est interactif et plus la définition de l'extérieur contraint la conception de l'intérieur ; ce partage réduirait donc la marge de manœuvre des architectes logiciels. C'est donc la troisième frontière qui semble la plus pertinente. Dans ce cadre, la conception consiste à accumuler les choix jusqu'à ce que le produit final soit complètement défini, et la spécification est la matérialisation d'une frontière entre la conception déjà effectuée et celle qui reste à faire, sans préjuger de leur nature.

Cette définition rend compte de la variabilité des situations. La spécification est un contrat et selon les équipes, les enjeux et les relations contractuelles [17], la spécification sera macroscopique ou fine, abstraite ou concrète, exprimée sous forme de fonctions ou d'« ilities », fixant des choix d'interaction et de couleur ou d'architecture logicielle et d'idiomes de programmation. Dans tous les cas il s'agira compléter la conception déjà capturée dans cette spécification. Le cas du calcul (conception initiale triviale) ou du « cycle en escargot » (conception de l'IHM en amont, de l'architecture en aval) en sont des cas particuliers simples, et il n'existe à ce jour pas de schéma général.

Vers un langage pour l'interaction entre concepts

Derrière la complexité des notions de conception et la spécification se cachent donc deux réalités :

– divers métiers produisent des éléments de conception du système interactif, différents dans leur nature et leur niveau d'abstraction, mais ayant vocation à interagir entre eux ;
– les schémas d'organisation entre ces tâches de conception sont fluctuants, un métier devant s'adapter aux choix des autres ou l'inverse selon les projets.

Par analogie avec les approches d'ingénierie dirigée par les modèles, on peut ainsi analyser le travail collectif comme une série d'enrichissements successifs des choix de conception, entrecoupés de traductions de ces choix vers un langage compréhensible par les autres métiers. Plus les langages sont compatibles et moindres sont les pertes lors de ces traductions, par exemple dans la rédaction et la lecture de la spécification. Il ne s'agit pas de produire un langage unique utilisé par tous, car chaque métier a ses propres concepts à manipuler. Mais disposer d'une base de concepts communs dont dérivent les concepts propres à chacun permettrait de diminuer les pertes lors des traductions, voire d'établir directement des interactions entre les productions, par exemple :

– la combinaison d'une interaction tactile avec un style visuel, de même qu'on peut combiner un algorithme de tri avec une structure de données ;
– la vérification, qu'il s'agisse de compatibilité entre éléments d'interaction [1] ou de conformité d'un logiciel avec la tâche analysée [18] ;
– le raffinement d'un choix de conception. Cela peut être la définition du contenu de chaque zone d'une interface graphique dont la structure a été définie dans la spécification, ou la mise en œuvre de principes de conception définis auparavant (exemple : « toute opération de destruction sera précédée de deux demandes de confirmation »).

Un tel ensemble de concepts de base a déjà été proposé : celui des langages à objets, popularisé à la même époque que les interfaces graphiques et qui a donné naissance à la conception par objets ou « conception orientée objets ». Il remplit les exigences que nous venons de décrire et certains auteurs ont proposé de l'appliquer à la conception de systèmes interactifs [8], mais avec un succès mitigé.

ATOUTS ET LIMITES DES OBJETS

La structuration des logiciels en collections récursives d'objets qui communiquent par échange de messages a été proposée par Alan Kay [14]. La communauté du génie logiciel s'en est rapidement emparée comme base d'une méthode de conception logicielle [2], celle de la programmation d'IHM

comme base pour ses travaux d'architecture avec notablement MVC [12] et les « design patterns » [11], et cette influence s'est étendue jusqu'à la conception d'interfaces. Ce succès n'est pas dû au hasard. D'une part, l'objectif de Kay était explicitement la conception de systèmes interactifs. D'autre part le modèle est délibérément construit sur des théories cognitives et des concepts philosophiques de base. Il vise à permettre à chacun, du concepteur à l'utilisateur final, de manipuler des concepts qui lui sont naturels pour décrire les entités qu'il manipule, et inclut des mécanismes d'abstraction pour décrire tant des idées que des phénomènes concrets.

Cette généralité revendiquée donne au modèle à objets des atouts pour servir de base à la communication entre acteurs de la conception, et donc pour placer les frontières, les interfaces, entre éléments de conception là où le projet le requiert :

– Il intègre les données et le calcul. C'est très favorable pour les programmeurs, mais aussi pour tous ceux qui ont à dialoguer avec eux. C'est un bon exemple de concepts de base qui facilitent les traductions entre acteurs de la conception.
– Il atténue la frontière entre objets physiques et informatiques. Cette frontière est indésirable en conception d'interfaces, car d'une part de nombreux paradigmes d'interaction reposent sur la reproduction informatique de caractères attribués à des objets physiques, et d'autre part certains logiciels doivent reproduire des objets physiques ; c'est le cas des claviers virtuels par exemple. Cela permet de fixer les frontières entre concepteurs là où le projet le requiert et non à l'interface entre le monde physique et le monde virtuel.
– Il s'agit d'un modèle hiérarchique, qui rend compte des cas multiples de découpage des tâches et des niveaux d'analyse. On peut par exemple se mettre d'accord sur la structure d'un objet sans détailler complètement ses composantes. On peut aussi utiliser le modèle pour décrire des parties de système dont la conception interne est inconnue mais dont il faut étudier les interactions avec les autres parties. C'est le cas, dans les systèmes homme-machine, de l'humain et de ses processus cognitifs et physiques.
– Il offre des mécanismes pour définir des frontières entre ce qui est déjà décidé et ce qui ne l'est pas encore, à commencer par les concepts de classes et de dérivation. Toutefois, il faut reconnaître que ces mécanismes sont surtout utilisés par les informaticiens.

Depuis sa proposition dans les années 1990, la conception par objets appliquée à l'IHM n'a néanmoins pas rencontré de succès durable. Plusieurs raisons sont proposées ci-dessous pour expliquer cet échec. On peut les interpréter comme des difficultés qui perturbent la communication à certaines frontières et donc compliquentle partage des tâches de conception.

– Tout d'abord certains mécanismes de description semblent appropriés aux informaticiens mais moins aux autres métiers. C'est le cas pour les mécanismes de classe et de dérivation, donc pour la relation entre cas général et cas particulier. Ça l'est encore plus pour le mécanisme d'abstraction proposé : les classes abstraites, qui spécifient des principes sans les mettre en œuvre. Il semble que ce schéma soit spécifique aux raisonnements des informaticiens, et on observe d'ailleurs que le principe des classes abstraites est plus facile à expliquer en décrivant comment il est mis en œuvre dans la mémoire de l'ordinateur.
– De même, quoi qu'en aient dit les spécialistes du modèle à objets, le mécanisme de réification par lequel on peut transformer une idée, un processus, ou tout autre concept abstrait en objet ne semble naturel que pour les informaticiens. Ces derniers savent d'ailleurs que tous les objets sont représentés en mémoire de manière abstraite quoi qu'il arrive. Pour les autres le poids du mot « objet », très ancré dans la réalité du monde physique, est plus fort que le raisonnement proposé. De plus, le mot a déjà un sens particulier pour les designers. Cela introduit donc une contrainte très forte : le concept d'objet n'est utilisable en dehors des tâches informatiques que pour des objets suffisamment concrets. Dès qu'il s'agit de parler de couleur, de comportement, d'étape de dialogue, de tâche, de principe de conception, ou de processus mentaux, il faut en revenir à la traduction sans concepts de base partagés entre les métiers.
– Enfin, les langages à objets sont une libre application du concept d'objet imaginé par Alan Kay. À de rares exceptions près, ils s'inscrivent dans le cadre du calcul procédural, qui ramène toute exécution de programme à une suite d'instructions et d'appels de fonctions. Tout échange de messages est un appel de fonction, c'est-à-dire un fragment de dialogue : un appelant soumet des données (le contenu du message) à un appelé, lequel traite les données, produit un résultat puis permet à l'appelant de poursuivre son exécution. Dans cet échange, un objet prend l'initiative de s'adresser à un autre objet qu'il connaît, puis attend sa réponse. Cela éloigne le modèle à objets du concept intuitif d'objet, pour deux raisons. D'une part les concepteurs d'interfaces savent bien que le dialogue n'est que l'un des paradigmes d'interaction possibles. D'autre part, l'entité qui déclenche l'interaction n'est pas toujours celle qui « connaît » le destinataire des messages, car dans les ordinateurs les sources d'événements préexistent souvent aux comportements qu'elles déclenchent. En conséquence l'interaction entre entités, qui est l'un des éléments centraux des échanges entre concepteurs, requiert des traductions complexes pour s'exprimer dans le modèle à objets. Au sein même de l'informatique, ce problème a engendré des dizaines d'années de recherche sur des principes alternatifs d'architecture et d'exécution des IHMs, à commencer par le modèle à événements. Il est donc logique que le modèle à objets ne puisse être la lingua franca de la conception.

Le modèle à objets a donc des défauts importants en tant qu'ensemble de concepts de base à partager entre les acteurs de la conception de système homme-machine. Cela explique pourquoi la conception par objets d'IHM n'a pas eu de succès généralisé. Mais ce modèle pose des problèmes aux programmeurs d'IHM eux-mêmes, et cela donne une piste de solution explorée dans la suite de cet article : s'appuyer sur des modèles alternatifs conçus pour l'informatique interactive afin d'imaginer une base commune aux divers concepts manipulés par les concepteurs.

DE L'OBJET AU COMPOSANT INTERACTIF

De nombreuses propositions alternatives ou complémentaires au modèle à objets ont été faites par les chercheurs en architecture logicielle pour l'IHM. L'alternative décrite ici, celle des composants interactifs, est focalisée sur les faiblesses décrites plus haut, à commencer par le problème du modèle pro-

cédural. Elle s'appuie sur des travaux théoriques en informatique répartie, dont il faut d'abord introduire quelques bases.

Informatique répartie et théorie des processus

Il n'y a pas qu'en conception et développement d'IHM que le modèle procédural convient mal. Alors qu'en étudiant les racines des langages de programmation courants on observe à quel point ils sont influencés par le calcul [5], il existe depuis l'origine de l'informatique une approche alternative de modélisation et de programmation, pour laquelle l'architecture de van Neumann est un choix parmi d'autres et non un présupposé. Cette approche, qui se focalise sur la description du monde comme un ensemble de systèmes qui interagissent entre eux [20] et non comme un ensemble de calculs à effectuer, est à la base des résultats sur les systèmes répartis. Ses applications pratiques en termes de génie logiciel ont été pour l'essentiel limitées à quelques langages pour ordinateurs parallèles, et des intergiciels spécialisés dans la gestion de systèmes répartis. Mais ses résultats théoriques sont au coeur de la modélisation et la preuve de systèmes critiques, et ont une influence tant sur l'ingénierie des réseaux que sur la plupart des approches formelles en IHM. Par exemple, les ICOs [17] et les ConcurTaskTrees [16] s'appuient sur des notions issues de l'informatique répartie. Il existe plusieurs modèles théoriques des systèmes répartis. Outre les acteurs, qui correspondent à la définition qu'A.Kay donnait à l'origine pour les objets [2], les plus répandus sont les réseaux de Petri, utilisés dans les ICOs, et les algèbres de processus [13], qui sont à la base de la programmation réactive.

Les processus sont parmi les concepts les plus fondamentaux pour décrire l'interaction : un processus peut se définir par la suite de ses interactions avec d'autres processus, et l'on peut construire une théorie où les seuls concepts sont ceux de processus et d'interaction [15, 10]. L'exécution d'un programme se décrit alors comme la suite des interactions entre les processus qui le constituent. De même tout processus peut être décrit comme un ensemble de processus plus élémentaires interagissant entre eux. Par exemple une animation résultera de l'interaction entre une horloge, des objets graphiques et des trajectoires. Cette régression vers des processus de plus en plus élémentaires aboutit aux instructions de base de l'ordinateur, décrites elles aussi comme des processus.

En appliquant la même technique de régression, on peut définir l'état d'un processus. Les interactions provoquent des transitions entre états, ce qui permet d'analyser le comportement d'un processus sous forme de machine à états. Si l'on définit la valeur d'un processus comme la représentation de son état dans un ensemble que l'on choisit à volonté, par exemple l'ensemble des couleurs, on peut alors construire des processus abstraits qui sont « exécutés » par les processus plus concrets. Par exemple, si un processus prend ses valeurs dans l'espace des couleurs et reste toujours dans les nuances de vert ou de rouge, on peut construire sur cette base un processus à deux valeurs, vert et rouge. On a ainsi défini une relation de compatibilité, de raffinement, entre un processus abstrait et un processus concret, entre un processus et un processeur.

2. voir http ://www.mail-archive.com/fonc@vpri.org/msg03244.html

Les composants interactifs

En substituant le modèle des processus à celui des instructions et des appels de fonctions, on transforme le modèle à objets en un modèle de composants interactifs [6] : tout comme un objet est constitué de sous-objets qui échangent des messages, un composant interactif est constitué de sous-composants qui interagissent. La différence majeure est dans la richesse des interactions que l'on peut décrire. Alors que dans le modèle procédural on est contraint à l'appel de fonction, ici quatre schémas sont possibles :

- Appel de fonction ; étant donné un composant, on crée un second composant qui déclenche à volonté l'activation du premier. C'est ce qui se passe quand on imprime un texte dans une console et, de manière plus complexe, quand on maintient un objet graphique à l'écran.
- Événement ; étant donné un composant, on en crée un second qui est activé quand le premier l'est. C'est ce qui se passe quand on s'abonne aux changements d'état d'une souris, ou à la réception de messages sur le réseau.
- Aspect ; étant donnés deux composants, on en crée un troisième qui assure que le second est activé quand le premier l'est. C'est qui se passe quand on connecte des composants logiciels à la fois aux périphériques d'entrée et aux dispositifs d'affichage, ou quand on interconnecte des composants qui n'étaient pas destinés à fonctionner ensemble.
- Routeur : deux composants sont créés indépendamment l'un de l'autre, en utilisant la connaissance d'un troisième composant qui assure que l'un est activé quand l'autre l'est. C'est ce qui se passe dans les systèmes d'exploitation entre les pilotes de périphériques et les applications.

Ces quatre styles d'interaction entre composants permettent des organisations différentes du travail, en fonction de quel programmeur réutilise quels composants existants. Mais surtout, ils correspondent à la variété des situations que les programmeurs d'interfaces ont à gérer : respectivement les sorties, les entrées, leur orchestration, et les systèmes d'exploitation. Qui plus est, la définition des composants interactifs peut être étendue aux périphériques d'interaction voire aux autres objets du monde physique, en conservant les mêmes quatre types de connexions. En transformant ainsi le modèle à objets, on en conserve les atouts tout en palliant ses faiblesses :

- Le modèle intègre les données et le calcul, autant que le modèle à objets. Mais surtout, il permet de décrire d'autres notions essentielles en IHM : comportements, horloges, sources d'événements, filtres de flots de données, etc.
- Il atténue d'autant plus la frontière entre objets physiques et informatiques qu'il décrit des schémas d'interaction plus naturels : autant un objet physique ne dialogue pas, autant il interagit. Les périphériques d'interaction et d'affichage, tout comme les objets physiques de l'interaction tangible, se décrivent facilement comme des composants interactifs.
- Il est hiérarchique par construction.
- Il offre, si nécessaire, les mêmes mécanismes de structuration que les langages à objets puisqu'il en est un surensemble. Mais surtout, grâce au mécanisme de processus abstrait décrit plus haut, il permet de rendre compte de ce qu'un composant est bien la mise en œuvre, le raffinement, d'un processus abstrait défini au préalable. Il suggère ainsi de nouvelles manières de décrire des frontières entre éléments de conception, des contrats entre concepteurs.

– Le concept de composant interactif semble plus approprié que celui d'objet pour décrire des entités abstraites, à la fois parce que le mot lui-même évoque la notion abstraite d'assemblage plus que celle de réalité physique, et parce que l'interaction est un concept plus souple que celui d'échange de messages. Il n'est donc pas choquant d'envisager un comportement ou une animation comme un composant interactif. Mais surtout, grâce à son articulation avec le modèle de processus (un composant interactif peut être décrit par un processus), le modèle est compatible avec la description de concepts véritablement abstraits : une activité, une tâche, une procédure et un processus cognitif ne sont sans doute pas des composants interactifs, mais ce sont des processus dont on peut facilement décrire l'interaction avec des composants interactifs.
– Et pour finir, il propose par construction une définition beaucoup plus riche de l'interaction entre composants puisqu'il est appuyé sur le modèle des processus interagissants.

Somme toute, ce modèle de composants interactifs correspond à la notion intuitive d'objet telle qu'introduite par Kay, tout en étant enraciné dans un modèle de processus qui rend compte de phénomènes plus abstraits. Qui plus est, le modèle de processus sous-jacent permet aussi de passer à volonté à des modélisations par machines à états, réseaux de Petri ou flots de données, donc d'appliquer les résultats déjà disponibles sur ces modèles. Il semble donc naturel de tenter la généralisation à la conception de systèmes homme-machine.

VERS LA CONCEPTION ORIENTÉE SYSTÈMES

Revenons donc vers notre problème de départ, qui est l'identification de concepts de base pour dialoguer entre métiers de conception, ou pour servir de cadre d'interprétation des concepts de chaque métier. Le modèle des composants interactifs compense certaines lacunes du modèle à objets pour décrire les logiciels interactifs. Peut-on le généraliser afin de le rendre pertinent pour tous les acteurs de la conception, et réussir là où le modèle à objets a échoué ? Cela permettrait de clarifier les relations entre les différents métiers, de rendre les spécifications plus utilisables, et surtout d'adapter les processus d'ingénierie aux besoins de chaque projet plutôt qu'aux contraintes imposées par les frontières entre langages. J'esquisse ci-dessous un cadre conceptuel qui vise à remplir ce rôle et permettre la traduction des concepts propres à chaque métier vers un langage commun.

Processus, composant, système

Les noms ont un poids, nous l'avons vu à propos de la conception par objets : pour un designer, une interaction n'est pas un objet. Si un cadre doit servir à tous, il est important de choisir un nom neutre, qui corresponde le mieux possible à la nature de ce que l'on étudie ou l'on conçoit. Ainsi, même si le modèle décrit plus haut est proche du concept originel d'objet, il est impensable de réutiliser le même nom, tant pour les raisons déjà évoquées que parce que son sens est figé pour de nombreuses années. De même, le mot « composant » possède un sens particulier pour les concepteurs logiciels, proche du modèle à objets ; de plus il est probable qu'il poserait des problèmes similaires pour traiter de concepts très abstraits. Convenons donc de réserver le mot « composant interactif » à ce qui est manipulé par les informaticiens, quitte à ce que cela inclue le matériel, voire les humains pour ceux qui en acceptent une vision mécaniste.

À l'inverse, le mot « processus » est sans doute trop abstrait pour que chacun s'imagine concevoir un processus. Même s'il a l'avantage d'être applicable quasi-universellement, il convient mieux à la description et à l'étude des interactions. Le mot qui semble représenter le meilleur compromis est « système » : il est habituellement associé au concept d'interaction [3], il est d'ores et déjà utilisé par chaque métier pour décrire des notions soit concrètes soit abstraites, et ces usages sont compatibles avec la définition que nous souhaitons donner au mot, à commencer par le système informatique, le système homme-machine, et le système cognitif. Par ailleurs, il est cohérent de modéliser le comportement des systèmes par des processus, comme c'est le cas pour les composants interactifs. Ces derniers deviennent alors des cas particuliers de systèmes, produits ou manipulés par des informaticiens.

La proposition avancée ici repose donc sur trois notions : *on conçoit ou on met en **interaction** des **systèmes**, que l'on modélise au besoin par des **processus***, ou le cas échéant par des modèles dérivés (états, flots de données, etc). C'est donc une conception à base de systèmes qui est proposée ou, en acceptant la loi de l'anglicisme, une conception orientée systèmes. Les systèmes étudiés sont en interaction à des niveaux très variés, mais d'une manière qui peut être décrite avec les mêmes concepts théoriques : interaction entre système humain et système informatique, entre composants interactifs que l'on assemble, entre système perceptif et affichage, entre humain, automatisme et procédure, voire entre systèmes homme-machine (le contrôle aérien et l'avion, par exemple).

À ce stade il n'est pas question de proposer des méthodes de conception orientée-système : cela nécessiterait des études pluridisciplinaires qui dépassent le cadre de cet article. Il s'agit plutôt de stimuler l'intérêt des diverses communautés concernées et de se convaincre qu'il est possible d'y travailler en traduisant sous forme de systèmes tout ce qui est conçu ou analysé lors de la conception d'un système homme-machine. C'est l'objet des exemples ci-dessous, destinés chacun à un ou des métiers différents.

Situation 1 : conception logicielle

Considérons un objet simple qui représente la console d'un ordinateur dans les langages à objets usuels ; il a deux méthodes `print` et `beep`. On peut voir la console comme un composant interactif, donc un système, dont `print` et `beep` sont deux sous-composants. Pour les activer, il faut les mettre en interaction avec d'autres systèmes. Une souris est un système physique relié à un composant logiciel qui la représente, et possède des sous-composants nommés boutons, eux-mêmes contenant `press` et `release`. Idem pour le clavier. En utilisant un composant spécial nommé `binding`, on peut fabriquer un composant qui met en interaction les trois systèmes et constitue un programme simple.

```
<component id="console-logger">
<binding source="mouse.right.press" action="console.beep"/>
<binding source="keyboard.release" action="console.print"/>
</component>
```

3. définition : un système est un complexe d'éléments en interaction

Situation 2 : analyse d'une technique d'interaction
La plupart des acteurs de la conception parlent d'« événement clic » pour désigner ce qui se passe lorsqu'on sélectionne une icône sur un écran. Mais combien d'entre eux sont conscients qu'il s'agit en réalité d'un enchaînement d'actions, différent entre un écran tactile et une souris, et que cela a des conséquences tant en matière de programmation que de performance humaine ? Avec la multiplication des moyens d'interaction, cette complexité sous-jacente peut de moins en moins être ignorée. Par exemple, quels sont les conséquences du passage d'un écran tactile à une caméra 3D et que faut-il adapter ? Pour l'étudier il faut un langage commun entre informaticien, designer d'interaction et ergonome.

Ces séquences d'interaction peuvent s'exprimer utilement en termes de systèmes en interaction : le cerveau connecté à la vue, au système moteur et à des capteurs tactiles, la souris constituée de systèmes mécaniques, d'un système d'échantillonnage et d'un système USB, l'ordinateur, son propre système USB et son système d'exploitation, et une série de composants logiciels qui filtrent les signaux. Chaque système de cette chaîne apporte son propre comportement, ses contraintes de fonctionnement, et ses pannes, dans ce qui est perçu comme un événement ponctuel.

Analyser ces systèmes et leurs interactions permet de mieux maîtriser la conception. Par exemple, dans le cas d'une souris, l'interaction continue entre le système moteur et le système « bouton » provoque dans ce dernier un soudain changement d'état qui est transmis à la fois par les capteurs tactiles du doigt et par le système électronique, créant ainsi deux processus de rétroaction. Cette redondance joue un rôle dans la robustesse de l'interaction, comme en témoignent les performances dégradées en cas de défaillance des pièces qui transmettent le changement d'état au doigt (on peut considérer alors que c'est le système homme-machine qui est en panne). Donc transposer le « clic » en utilisant une caméra 3D est délicat, car cette redondance est alors absente. Il faut donc soit proposer un nouveau système pour supporter le processus manquant, soit accepter des performances dégradées.

Situation 3 : automatisation et partage de responsabilités
Une question fréquente de conception est celle de l'automatisation totale ou partielle d'une tâche, dont les conséquences sont souvent mal anticipées. Analyser le système homme-machine comme une collection de sous-systèmes mettant en œuvre des processus permet de structurer la conception : la liste des processus requis doit être identifiée (exigences fonctionnelles) et tracée (spécification), puis les concepteurs définissent les sous-systèmes qui les mettent en œuvre et vérifient que les interactions entre sous-systèmes sont efficaces. Chaque migration de processus d'un sous-système à l'autre doit s'accompagner de nouvelles vérifications. Le partage de responsabilités est ainsi traité comme l'architecture en sous-systèmes du système homme-machine.

Les détecteurs d'armes dans les aéroports affichent des images aux opérateurs chargés de les interpréter et donc de mettre en œuvre le véritable processus de détection. Certains envisagent de remplacer une partie du processus mental de détection par des algorithmes d'analyse d'image. Cette automatisation revient donc à déplacer un processus du cerveau humain vers l'ordinateur. Mais, si l'humain reste une partie du système d'analyse, quelle sera la nouvelle interaction avec lui ? Autrement dit, quelle sera l'IHM du nouvel algorithme ? S'il s'agit d'une interface visuelle, et sachant que le processus de lecture mis en œuvre ne sera plus le même, comment vérifier si le système global peut être plus efficace que l'ancien ? Des modèles théoriques comme ScanVis [9] peuvent contribuer à cette vérification en explicitant les processus mis en œuvre lors de la lecture. On voit à cette occasion comment les modèles de performance de l'interaction peuvent s'inscrire dans la vérification de l'architecture du système homme-machine.

Situation 4 : tâches et processus mentaux
L'usage des processus et processeurs n'est pas nouveau en sciences cognitives, le modèle ICS en est un exemple. Sans rentrer dans le débat entre connexionnisme et cognitivisme, il est intéressant de remarquer que le modèle des processus est un modèle de type « boîte noire » qui décrit ce qui est observé sans porter de jugement sur le fonctionnement interne. Cela se prête bien à décrire comment des systèmes peuvent interagir avec le système cognitif, et l'aspect hiérarchique du modèle permet de transformer progressivement la boîte noire en boîte transparente au fur et à mesure que des modèles plus fins de la cognition sont proposés. Ainsi, on peut décrire la lecture d'une image comme une interaction entre l'écran et le système de balayage et de détection de contours, puis éventuellement raffiner le modèle du système visuel pour décrire quels sous-systèmes sont à l'œuvre.

Quant aux tâches et activités, qui jouent un rôle important en ergonomie, il peut être exagéré de les présenter comme des systèmes car ce sont des constructions très abstraites. Mais cela ne fait que souligner leur rôle descriptif, par opposition à des objets que l'on construit. Ainsi, les activités sont des processus résultat du fonctionnement de systèmes, par exemple le système homme-machine ou le système homme-environnement. Quant aux tâches, ce sont des processus abstraits, dont il faut vérifier que l'activité constitue bien un raffinement. On retrouve ainsi l'approche décrite dans [18] dès que l'on confronte les tâches aux modèles de systèmes homme-machine sous forme de processus.

PERSPECTIVES

L'analyse proposée dans cet article est un point de départ plus qu'un résultat. Des environnements de développement à base de composants interactifs vont bientôt être disponibles [6], ce qui permettra l'expérimentation de la conception orientées systèmes appliquée au logiciel. La possible généralisation à la conception d'interfaces ouvre de nombreuses perspectives de recherche. Tout d'abord, les définitions esquissées ci-dessus pour les concepts de chaque métier auront besoin d'être approfondies, et leur utilité et leur utilisabilité validées avec des professionnels de chaque métier. L'intérêt de disposer de concepts de base communs devra aussi être évalué : quels dialogues, quelles validations, quels contrats deviennent possibles ? En cas de succès, on pourrait alors envisager de définir sur cette base des schémas répétables d'organisation dans les projets, au-delà du schéma global conception-spécification-conception dont les limites sont connues. On peut aussi envisager à terme des techniques réutilisables de spécification (au sens « document contractuel »).

Il sera par ailleurs intéressant d'étudier les limites en largeur du concept de système. Le système cognitif et le processeur humain ont déjà été évoquées, mais la dimension collective mérite aussi d'être étudiée. On sait par exemple que pour construire un système socio-technique, il faut des outils, mais aussi des méthodes, des procédures et des formations. Or qu'est-ce qu'une procédure sinon un processus ? Peut-on la concevoir de la même manière qu'un système physique ou informatique ? Peut-on migrer ce système d'un processeur humain vers un processeur informatique, réalisant ainsi de manière simple une délégation de tâche de l'humain vers l'automatisme ? De même, on peut voir les méthodes comme des processus, qui doivent être conçus mais aussi installés dans les habitudes des opérateurs ; ainsi, la formation peut être vue comme la création de nouveaux processus mentaux, et l'on peut peut-être raisonner sur le succès d'une formation en utilisant les outils de l'analyse de la fiabilité des systèmes. Idéalement, les règles, lois et décrets qui contraignent les concepteurs et opérateurs de systèmes critiques pourraient aussi être modélisés selon les mêmes principes et pris en compte dans la conception.

Enfin, alors même que le présent travail met la conception au centre du processus d'ingénierie et est donc en fort contraste avec la dérivation automatique de modèles, il peut servir de base à une convergence entre les deux approches. En effet, si l'on considère la conception comme une série d'enrichissements et de raffinements d'un ensemble de choix de conception, on se rapproche du processus de base de l'ingénierie dirigée par les modèles. Améliorer la compatibilité entre modèles et diminuer les coûts de traduction devient alors un objectif commun aux deux approches, et peut permettre à terme une convergence. La dérivation de modèles deviendrait alors l'un des outils utilisables pour la conception orientée système.

CONCLUSION

Dans l'optique d'une réflexion sur les méthodes d'ingénierie système pour l'interaction homme-machine, nous avons examiné les difficultés rencontrées par les acteurs de la conception de logiciels interactifs à avoir une vision claire et partagée de leurs processus d'ingénierie. En analysant le statut respectif de la spécification et des activités usuelles de conception, on observe qu'un schéma où des métiers différents font chacun des choix de conception et définissent des contrats aux frontières entre ces choix rendrait mieux compte de la réalité, pour autant qu'un langage ou un ensemble de concepts communs leur permettent d'interconnecter leurs productions. Analyser en quoi le modèle à objets a échoué à devenir cette base commune permet d'envisager une nouvelle solution où les concepts d'interaction et de système jouent un rôle central. Cette approche, la conception orientée systèmes, est ambitieuse au premier abord mais n'a d'autre objectif que de revenir à des évidences parfois oubliées : d'une part, les concepts d'architecture logicielle doivent être au service de la conception de la totalité du logiciel et il est donc logique de les remettre en question tant que la convergence n'est pas atteinte ; et d'autre part, alors même que l'interaction est au coeur des modèles usuels des systèmes, il serait paradoxal qu'une discipline centrée sur l'interaction et la conception de systèmes ne cherche pas à disposer de méthodes et de théories construites sur ces deux concepts.

REMERCIEMENTS

Ce travail a été financé en partie par l'ANR (projet Istar) et le FUI (projet Medusa du pôle Mer). Il prend ses racines dans les projets de conception ou recherches sur les outils et méthodes menés avec mes collègues d'IntuiLab et d'Intactile Design ; ils se reconnaîtront. Merci à H. Gaspard-Boulinc et M. Magnaudet pour leurs relectures et leurs suggestions.

REFERENCES

1. J. Accot et al. Formal transducers : models of devices and building bricks for the design of highly interactive systems. In *Proc. DSVIS'97*. Springer-Verlag.
2. G. Booch. *Object-oriented analysis and design with applications*. Benjamin/Cummings, 1994.
3. G. Booch et al. *The Unified Modeling Language User Guide*. Pearson Education, 1999.
4. G. Calvary et al. A la croisée de l'ingénierie de l'interaction homme-machine et de l'ingénierie dirigée par les modèles. *Technique et science informatiques*, 29, 2010.
5. S. Chatty. Programs = data + algorithms + architecture. In *Proc. Engineering Interactive Systems*, Lecture Notes in Computer Science. Springer-Verlag, 2007.
6. S. Chatty. Supporting multidisciplinary software composition for interactive applications. In *Proc. Software Composition*, LNCS 4954. Springer-Verlag, 2008.
7. A. Cockburn. *Agile Software Development*. Addison-Wesley Professional, 2001.
8. D. Collins. *Designing object-oriented user interfaces*. Benjamin/Cummings Publishing Company, 1995.
9. S. Conversy et al. Visual scanning as a reference framework for interactive representation design. *Information Visualization*, 10 :196–211, Sage, 2011.
10. A. Dittmar et P. Forbrig. A unified description formalism for complex HCI-systems. In *Proc. Software Engineering and Formal Methods*, IEEE, 2005.
11. E. Gamma et al. *Design Patterns : Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1995.
12. A. Goldberg et D. Robson. *Smalltalk-80 : the language and its implementation*. Addison-Wesley, 1983.
13. C. Hoare. Communicating sequential processes. *Communications of the ACM*, 21(8) :666–677, 1978.
14. A. C. Kay. The early history of Smalltalk. *ACM SIGPLAN*, 28(3) :69–75, Mar. 1993.
15. E. A. Lee et A. Sangiovanni-Vincentelli. Comparing models of computation. In *Proc. ICCAD 1996*.
16. G. Mori et al. CTTE : Support for developing and analysing task models for interactive system design. *IEEE Trans. on Software Engineering*, 28(2) :797–813, 2002.
17. P. Palanque et R. Bastide. Petri net based design of user-driven interfaces using the interactive cooperative object formalism. In *Proc. DSV-IS'94*, 1994.
18. P. Palanque et al. Validating interactive system design through the verification of formal task and system models. In *Proc. EHCI'95*. Chapman & Hall, 1995.
19. C. Schlienger et al. Une expérience de conception et de prototypage d'interfaces évoluées en milieu industriel. In *Actes d'IHM 2004*, ACM Press, 2004.
20. P. Wegner. Why interaction is more powerful than algorithms. *Communications of the ACM*, 40(5), May 1997.
21. T. Winograd et al. *Bringing design to software*. ACM Press, 1996.