Un modèle de composants (inter)actifs centré sur les documents

Olivier Beaudoux

Un modèle de composants (inter)actifs centré sur les documents

Olivier Beaudoux

Ecole Supérieure d'Electronique de l'Ouest (ESEO) &
Laboratoire de Recherche en Informatique (LRI) / INRIA Futurs †
4 rue Merlet de la Boulaye, 49009 Angers
olivier.beaudoux@eseo.fr et
http://www.eseo.fr/~obeaudoux

Résumé

Le terme de document suggère le plus souvent les données que le document peut contenir et rarement les (inter)actions qu'il est possible de faire sur le document. Il en résulte une difficulté pour définir la manière dont les utilisateurs peuvent interagir avec les documents : cet aspect est généralement relégué à des applications dédiées avec lesquelles l'utilisateur est contraint de jongler. Nous proposons dans cet article un modèle de composants (inter)actifs mettant en jeu trois métaphores, le document, la présentation et l'instrument. Il vise à mettre le document et l'interaction au centre des espaces de travail de manière à s'affranchir de la notion même d'application.

Mots-clés *: Systèmes centrés sur les documents, modèle d'intéraction, composants actifs, boîtes à outil pour interfaces graphiques*

Abstract

The document concept commonly deals with data that documents contain, but rarely with (inter)actions that documents can handle. As a consequence, it is hard to specify how users can interact on documents : this is mainly done by dedicated applications from which users have to swap in order to achieve a single editing task. In this paper, we suggest the use of a model for interactive components based on three metaphors, the document, presentation and instrument. The proposed model aims at replacing the application concept by the document and interaction instrument concepts within interactive workspaces, thus resulting in a higher powerfulness / ease-of-use ratio.

Key-words*: Document centered systems, interaction model, active components, GUI toolkits*

†projet In Situ, Pôle Commun de Recherche en Informatique du plateau de Saclay, CNRS, Ecole Polytechnique, INRIA, Université Paris-Sud

1 CONTEXTE

1.1 Documents *versus* applications

Le fait que le document soit l'objet d'intérêt principal des environnements interactifs a déjà été identifié il y a plus de 20 ans : « le document est au centre du monde et l'unifie » [13]. Cependant, dans leur immense majorité, les environnements actuels sont fondés sur la notion d'application, chaque application étant dédiée à la manipulation de données typées. Un utilisateur est alors souvent contraint de jongler avec plusieurs applications de manière à créer un document. Les éditeurs de logiciels utilisent trois stratégies pour réduire la complexité qui en résulte :

1. Création de « mini-applications » à l'intérieur même d'applications plus larges – La suite logicielle Microsoft Office est constituée d'applications qui incluent chacune des fonctions de dessin vectoriel souvent semblables. Ces fonctions sont cependant moindres que celles fournies par une application spécialisée et sont, tout comme les formats de données, dupliquées.
2. Architecture ouverte basée sur les extensions (« plug-ins ») – Un grand nombre de plug-ins sont disponibles pour l'application de retouche d'image Adobe Photoshop. Cependant, leurs interfaces sont souvent mal intégrées dans les applications mères et restent accessibles au travers de boîtes de dialogues souvent complexes.
3. Suite d'applications à interfaces compatibles – Les applications Photoshop, Illustrator, GoLive et InDesign ont toutes la même présentation de l'interface graphique et des palettes d'outils similaires. L'interaction devient plus homogène et plus fluide, et le passage d'une application à l'autre est facilité. Cependant, cette approche impose de travailler avec plusieurs documents pour réaliser une tâche unique.

Le but de ces trois approches est de positionner le document, et non l'application, au centre de l'interaction. Cependant, elles ne parviennent pas à leurs fins car les utilisateurs ont toujours à jongler entre plusieurs applications et plusieurs documents afin de réaliser une tâche unique. Elles essaient de rendre les applications moins visibles en réduisant leurs écarts, mais la logique générale reste centrée sur l'application.

1.2 (Inter)actions et documents

Le terme de document suggère le plus souvent les données que le document peut contenir. Il est cependant pertinent de pouvoir spécifier, en plus de la sémantique des données, la sémantique des actions qu'il est possible de faire sur ces données [16]. L'élaboration de notre modèle s'inscrit dans ce cadre : son principe est de définir non seulement comment les documents

XML peuvent être perçus par les utilisateurs, mais aussi (et surtout) comment les utilisateurs peuvent (inter)agir sur ces documents. Nous adjoignons alors à un modèle de document un modèle d'interaction. Ce faisant, nous fournissons un mécanisme générique qui rend indépendantes les données, les actions qu'il est possible de faire sur ces données, et les interactions qui peuvent engendrer ces actions.

1.3 Plan de l'article

La section 2 positionne nos travaux par rapport aux systèmes centrés sur le document, aux modèles de composant ainsi qu'à la notion de document numérique. Les principes théoriques du modèle DPI (Document, Présentation, Instrument) sont synthétisés dans la section 3 en présentant les trois composants D, P et I et la manière dont ils dialoguent. La section 4 décrit le modèle de composant et la façon dont il permet de décrire nos trois composants. L'implantation du modèle dans la boîte à outil OpenDPI et les perspectives de nos travaux sont discutés en conclusion.

2 TRAVAUX CONNEXES

Notre approche se situe à la jonction de plusieurs domaines relativement indépendants : les systèmes centrés sur le document, les approches basées sur les composants logiciels et les modèles de documents structurés. L'ensemble des travaux connexes proposés n'est pas exhaustif mais correspond aux aspects les plus proches de notre modèle DPI.

2.1 Approches centrées sur les documents

Dans les approches centrées sur les documents, l'utilisateur n'a plus à manipuler des applications : il manipule directement le document.

HotDoc est une extension de MVC qui permet la mise en œuvre de documents composites [10]. Cette approche donne une lecture intéressante du motif MVC mais ne propose aucun modèle de document, d'affichage ni d'interaction. Le système OOE est une extension du système NextStep qui autorise une édition simplifiée des documents composites [2] en utilisant les capacités de l'affichage « display PostScript ». Cependant, ce système, en permettant de basculer très simplement d'une application à l'autre, reste centré sur les applications. Le système OLE permet l'édition de documents composites par le biais d'une communication inter-applications [9]. Cependant, il reste également centré sur les applications et impose un mécanisme d'interopérabilité complexe à mettre en œuvre.

L'approche la plus aboutie est probablement celle d'OpenDoc [1] dans laquelle les documents sont hiérarchisés en *parties* typées et hiérarchisées. Ces parties possèdent leur propre modèle de contenu et de comportement,

sont visualisées par des *visualisateurs* et sont éditées *via* des *éditeurs*. Le modèle proposé permet aux applications de disparaître réellement au profit des éditeurs et visualisateurs. Cependant, nous jugeons que :

- La granularité des éditeurs et visualisateurs reste encore élevée.
- Il n'y a pas d'interopérabilité entre les différents éditeurs.
- D'un éditeur à l'autre, l'interface utilisateur change.
- Chaque partie définit son propre format de fichier.
- La collaboration est abordée sous l'angle restrictif des « drafts ».
- Aucun modèle précis d'interaction n'est fourni.

Le modèle DPI a été bâti afin d'éliminer ces différents problèmes.

2.2 Modèles de composant

L'idée d'utiliser des composants logiciels au niveau même des espaces de travail est apparue il y un peu moins de dix ans. Nous pouvons citer, par exemple, les composants « JavaBeans » [19] dont l'implantation du modèle DPI s'est inspiré pour certains aspects (définition des propriétés et utilisation de l'introspection). Cependant, le modèle de Sun, comme ceux des autres éditeurs, reste très générique et vise de plus à construire des applications par assemblage de composant. Le modèle DPI propose son propre modèle de composant générique mais intègre également une instanciation de ce modèle en un modèle de document et un modèle d'interaction. De plus, les composants DPI sont avant tout *centrés sur l'utilisateur* : ce dernier choisit les composants de son espace de travail en fonction des tâches qu'il souhaite accomplir. Dans l'approche usuelle, les composants logiciels sont choisis par les concepteurs d'application.

2.3 Modèles de document

La spécification DOM [20] est devenu un standard de fait qui permet d'aborder le formalisme XML dans un contexte orienté objet. Dans notre approche, les documents étant *exclusivement* structurés en arbre, le modèle de document de DPI est assez semblable à DOM. La spécification DOM-Events adjoint au le modèle DOM la possibilité d'écoute des modifications réalisées sur un arbre XML. Nous retrouvons dans DPI une telle capacité d'écoute afin de permettre la synchronisation des données et de leur présentation.

Le deuxième aspect de DOM-Events concerne les événements utilisateurs. Cependant, il s'avère insuffisant pour définir un modèle d'interaction : les événements utilisateurs restent de bas niveau et ne permettent pas de considérer indépendamment les périphériques d'entrée, les interactions et les actions. De plus, le modèle DOM n'aborde pas la concurrence d'accès aux documents ni la problématique du partage des documents.

Nos travaux étant centrés sur la spécification d'un protocole d'interaction sur les documents XML, l'une de nos perspectives est de rendre possible la cohabitation des modèles DOM et DPI (section 5).

2.4 Le document comme forme

La réflexion du collectif « Roger T. Pédauque » [17] propose d'aborder la notion de document selon trois dimensions non exclusives : le document comme forme, signe et médium. Nous pensons que notre approche, de part le fait de (re)centrer l'interaction sur la notion même de document, permet de compléter la réflexion.

Le fait que nous considérions le document numérique comme l'objet d'intérêt principal pour l'utilisateur implique assez logiquement que notre approche aborde principalement le document en tant que *forme*. En particulier, puisque nous souhaitons qu'il masque le concept d'application à l'utilisateur, le document fixe lui-même le cadre de son utilisation, ce qui inclut sa représentation et (surtout) sa *manipulation*. Cet aspect rejoint la notion de « contrat de lecture » de Pédauque véhiculé par le document lui-même et non spécifiquement par des applications de lecture (ou lecteurs) dédiées. Cependant, dans l'approche du document de Pédauque, l'accent est mis sur les données et les structures portées par le document. Dans notre approche, nous nous focalisons davantage sur les actions et, par voie de conséquence, les interactions, qui peuvent être faites sur le document. Ceci nous suggère l'idée d'agrémenter l'équation ainsi : « Document XML = données structurées + mise en forme + *mise en interaction* ». Tout comme la mise en forme n'est pas spécifiée par le document lui-même mais par une feuille de styles associée, la mise en interaction n'est pas spécifiée par le document mais par les instruments d'interaction que le modèle DPI permet d'associer aux documents.

3 Principes du modèle DPI

Nous synthétisons dans cette section les principes théoriques essentiels à la compréhension du modèle DPI présentés dans [7, 6].

3.1 Composants essentiels du modèle

Notre modèle part du constat que, dans la vie réelle, nous utilisons des instruments afin d'interagir avec les objets d'intérêt, et non des applications. Par exemple, nous utilisons un stylo pour écrire sur une feuille de papier et un lecteur de CD audio pour écouter notre musique. Un *instrument* est défini dans notre modèle par sa partie physique qui capte l'action de l'utilisateur et lui présente un retour d'information, ainsi que par sa partie logique qui transforme cette action en une action sur l'objet ciblé et fournit une représentation de l'instrument [3]. La facette physique fait que l'instrument existe en dehors du système, et la facette logique implique que l'instrument existe à l'intérieur du système tout en restant perceptible depuis l'extérieur.

Dans notre approche, nous considérons le *document* comme étant l'objet d'intérêt principal pour l'utilisateur, l'instrument représentant le moyen

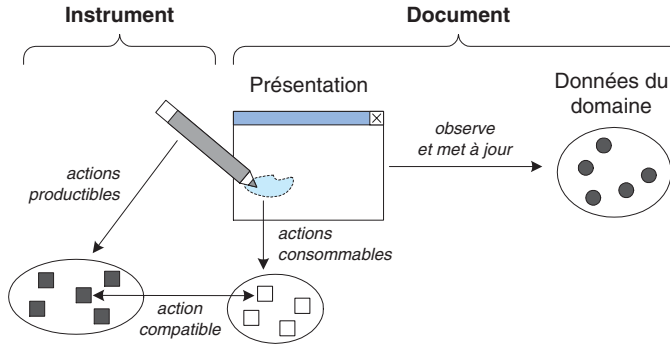

FIG. 1 – Principe du modèle DPI

d'(inter)agir sur cet objet. Nous considérons que tout document possède une facette perception et une facette persistance. La facette perception se décompose en une perception directe relative à la *présentation* (typiquement graphique) du document, et une perception sémantique relative aux *données du domaine* interprétées par l'utilisateur *via* la présentation. Ce découplage est désormais usuel dans la conception des interfaces (motif de conception MVC [14] et systèmes auteur multi-vues [15]). La facette persistance précise la capacité naturelle qu'a un document de conserver son état. Elle concerne les données du domaine *et* leurs présentations ce qui, cette fois, n'est pas l'usage habituel dans les interfaces. Cette approche du document conserve la métaphore du document réel, à ceci près que nous autorisons plusieurs présentations pour un même document. Par conséquent, le fait que le document contienne ses propres présentations implique qu'il n'est pas nécessaire de posséder une application dédiée pour pouvoir consulter le document, dès lors que les présentations utilisent un langage commun et standard (tel que PostScript ou SVG [21]).

3.2 Dialogue inter-composant

La communication entre un instrument et un document s'effectue *via* le vecteur appelé *action*. Une action est une grandeur qui peut être *produite* par (la partie logique de) l'instrument et *consommée* par un élément de l'une des présentations du document. La figure 1 illustre le chaînage qui s'opère entre la production d'une action et sa consommation :

1. Lorsqu'une interaction est effectuée par l'utilisateur sur l'instrument, elle correspond à une intention particulière de l'utilisateur et donc à une action (intentionnelle) appartenant à l'ensemble des actions que

l'instrument peut produire.

2. Par ailleurs, l'utilisateur précise *via* l'instrument (directement ou indirectement) quel élément est la cible de son action. Cet élément définit quant à lui les actions qu'il sait consommer.

3. Lorsque l'interaction a effectivement lieu, l'instrument effectue un test de compatibilité[1] entre l'action qu'il doit produire et l'ensemble des actions que l'élément ciblé est en mesure de consommer. Dès lors que le test retourne au moins une action consommable compatible, le chaînage de l'interaction peut avoir lieu : l'instrument transmet l'action productible (choisie par l'utilisateur si plusieurs actions sont compatibles) vers l'élément qui la consomme.

Bien que l'utilisateur interagisse sur l'une des présentations d'un document, son *intention* porte sur les données du domaine. Le couplage document - présentation assure alors la transmission des actions intentionnelles consommées par les présentations vers les données du domaine du document, ainsi que la synchronisation des présentations avec ces données. Il consiste à utiliser un mécanisme d'*observation* qui permet aux présentations d'observer tout changement de l'état du document (figure 1) :

1. Lorsque l'utilisateur effectue une action sur la présentation, cette dernière modifie son propre état ainsi que l'état du document.

2. Une seconde présentation (non visible sur la figure) du même document, observant également l'état du document, peut alors mettre à jour son propre état s'il y a lieu. Ceci garantit la synchronisation entre les différentes présentations d'un même document.

4 LE MODÈLE DE COMPOSANT

Les documents, les présentations et les instruments s'appuient sur un modèle de composant générique basé sur la notion d'état observable et de production d'actions. Le modèle proposé permet d'exprimer les facettes document / présentations (section 4.2) et interaction instrumentale (section 4.4).

Dans cette section, nous utilisons un exemple relativement simple mais néanmoins suffisant pour aborder les différents aspects du modèle (figure 2) : il s'agit de deux présentations d'un rectangle, l'une représentant le rectangle sous sa forme graphique et l'autre indiquant ses propriétés d'une manière textuelle. Le rectangle, au travers de sa présentation graphique, peut être dimensionné en utilisant un instrument constitué de quatre poignées de dimensionnement.

[1] Nous ne considérons dans cet article que la compatibilité triviale basée sur l'égalité du type de l'action productible et consommable.

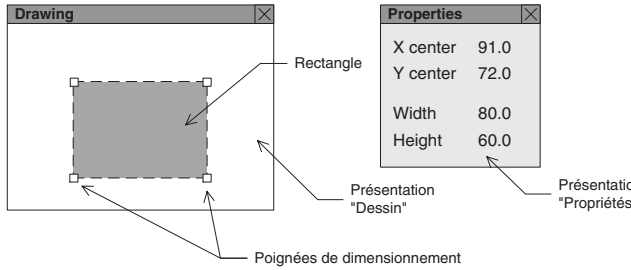

FIG. 2 – Deux présentations d'un rectangle

4.1 État observable

L'état d'un composant se décompose en un état propre et un état structurel. L'*état propre* caractérise les valeurs des *propriétés* [19] propres au composant. Par exemple, le rectangle composite définit les propriétés *x*, *y*, *width* et *height*. Par ailleurs, les composants peuvent être composés entre-eux de manière hiérarchique. L'*état structurel* qualifie la structure d'une telle composition en précisant quels sont les composants fils du composant considéré.

Selon les contextes d'utilisation, les changements d'état de composants peuvent intéresser d'autres composants. Afin de permettre la notification de tels changements d'état, nous utilisons le motif de conception « observable / observateur » [11], largement utilisé dans la conception des interfaces utilisateur. Nous considérons, d'une part, que tout composant *est* un *observable* et, d'autre part, que tout composant possède la capacité d'observer un composant et est donc un *observateur potentiel*.

Toute classe décrivant un composant DPI implante l'*interface observable* commune à toutes les classes de composant. Une classe de composant souhaitant observer l'état de composants doit implanter différentes *interfaces d'observation*. S'il s'agit d'une observation de l'état structurel d'un composant, l'interface à implanter est l'interface d'observation de la structure, commune à tous les composants. Cette interface définit les méthodes notifiant tout ajout et retrait d'un composant fils du composant observé. Dans le cas d'une observation de l'état propre d'un composant, l'interface à implanter est l'interface d'observation des propriétés qui est, quant à elle, définit conjointement à la classe du composant observé.

Nous étendons la notation UML afin de faire apparaître des différentes interfaces liées au mécanisme d'observation (figures 3) :

– Le disque noir représente l'interface observable. Cette notation rappelle que tout composant est un observable.
– Un disque blanc représente une interface d'observation implantée par le composant observateur :

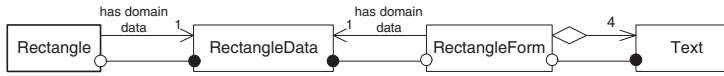

FIG. 3 – Exemple de synchronisation d'états

  - Lorsqu'il est complété du symbole de composition, il représente l'interface d'observation de la structure (non représenté sur la figure3).
  - Dans le cas contraire, il représente l'une des interfaces d'observation définit par la classe du composant observable.
- La connexion entre l'interface observable et une interface d'observation indique que l'observation est engagée.

4.2 Application aux documents

Dans notre approche, nous nous limitons volontairement à des documents structurés en arbre et dont chaque noeud est un composant DPI. Les données du domaine du document sont constituées de composants observés par les présentations associées. Les présentations, incluses dans le document, sont à leur tour constituées de composants (typiquement graphiques).

La synchronisation entre les présentations et les données du domaine d'un document est assurée par le mécanisme d'observation. La figure 3 précise comment l'exemple du rectangle avec ses deux présentations a été réalisé :

- Les données du rectangle sont définies par les quatre propriétés *xCenter*, *yCenter*, *width* et *height* du composant *RectangleData*.
- La présentation « dessin » contient le composant graphique instance de *Rectangle* qui représente graphiquement les données du domaine *RectangleData*. Ce composant observe les changements des données du domaine de manière à mettre à jour son état propre en conséquence.
- La présentation « propriétés » définit un formulaire (composant *RectangleForm*) qui permet d'éditer les quatre données du domaine par des cellules texte (composants *Text*). Le formulaire observe d'une part les données du domaine afin de rafraîchir ses cellules texte et, d'autre part, les états propres de ses cellules pour mettre à jour les données du domaine.

4.3 Production et consommation des actions

4.3.1 Caractérisation des actions

L'observabilité permet d'assurer la synchronisation des composants mais ne permet pas une communication inter-composants. Les actions permettent

la transmission, de proche en proche, des interactions de l'utilisateur aux différents composants concernés.

L'état d'une action est caractérisé par un ensemble de propriétés. L'action de translation définit par exemple les propriétés *deltaX* et *deltaY* représentant les déplacements en x et en y. Une classe d'actions définit uniquement son état et ne caractérise aucun comportement lié à la nature de l'action. Ceci est une conséquence de l'aspect polymorphe des actions [4] : l'action polymorphe a une forme imprécise définie par l'action elle-même, et des formes précises définies par les objets susceptibles de la consommer. Une classe d'actions ne précise ainsi pas les comportements de son exécution, de son annulation ni de sa ré-exécution : ils sont spécifiés par les *consommateurs* de l'action.

Le cycle de vie d'une action est immuable : l'action débute sur le composant cible, est produite dans l'intervalle de temps nécessaire à son exécution par l'utilisateur, puis se termine. Par exemple, la translation du rectangle de notre exemple s'effectue de proche en proche en modifiant successivement des propriétés *deltaX* et *deltaY* de l'action et en notifiant la cible de ces changement d'état.

4.3.2 Actions concurrentes et marquage des composants

Si plusieurs actions sont produites sur un même composant cible dans des intervalles de temps se *chevauchant*, il y a un risque de modification concurrente de l'état du composant. Il est important de remarquer qu'une telle modification concurrente n'induit jamais une incohérence de l'état du composant puisque cette dernière concerne uniquement les composants distribués sur un réseau. Par contre, une telle simultanéité résulte généralement en une *interaction incohérente*. Par exemple, si deux actions de translation sont consommées par le rectangle sans gestion des actions concurrentes[2], il en résulte une translation égale à la somme des deux translations concurrentes puisque l'action de la translation a la sémantique d'une translation relative (propriétés *deltaX* et *deltaY*) : la position du rectangle n'est alors plus conforme aux positions des curseurs représentant les instruments ayant produits les translations, d'où une interaction alors qualifiée d'incohérente.

Afin de garantir la cohérence de l'interaction et donc de prohiber la production des actions concurrentes, nous introduisons le marquage des composants qui se décompose en un marquage de propriétés et en un marquage de la structure. Le principe du marquage est le suivant : une action susceptible de modifier un ensemble de propriétés P ou/et la structure d'un composant peut être produite uniquement si aucune propriété de $p \in P$ n'est marquée ou/et si la structure n'est pas marquée. Dès lors que chaque action, dans l'intervalle de temps de sa consommation, marque les propriétés ou/et la structure qu'elle est susceptible de modifier, le principe précédent permet de prohiber

[2]Les deux translations peuvent être effectivement appliquée « simultanément » en utilisant un mécanisme de synchronisation des processus légers (mot clé *synchronized* en Java).

la production d'actions concurrentes. Ce faisant, l'instrument, en tant que producteur de l'action, *sait* (par le jeu des interfaces de consommation décrites dans la section suivante) s'il peut produire ou non l'action souhaitée par l'utilisateur : il peut alors fournir un feed-back informant l'utilisateur d'une éventuelle impossibilité.

Notons que le marquage est une technique assez similaire au verrouillage. Cependant, le verrouillage a pour but de garantir la cohérence des données et impose de ne pouvoir modifier de manière concurrente les données verrouillées. Le marquage n'interdit par contre pas explicitement la modification des données marquées : il permet aux actions d'*éviter* de telles modifications concurrentes.

4.3.3 Interface de production et de consommation

La production d'une action depuis un composant producteur vers un composant consommateur suit le cycle suivant :

1. L'action est initialement associée à son producteur.
2. Lorsqu'une interaction pouvant donner lieu à la production de l'action a lieu, un test de faisabilité est effectué sur le composant cible.
3. Lorsque ce test répond positivement (*i.e.* lorsque les propriétés ou/et la structure concernées ne sont pas marquées), l'action démarre en marquant les propriétés ou/et la structure concernées puis s'exécute.
4. A la fin de l'interaction, l'action se termine et ôte tout marquage préalablement effectué.

Ce cycle est mis en œuvre en définissant, pour chaque classe d'actions, une interface de production et une interface de consommation associées.

L'*interface de production* d'une classe d'actions A est implantée par toute classe de composants producteurs qui peuvent produire des instances de A. Elle permet de spécifier le *contrat* que le producteur doit satisfaire pour pouvoir produire l'action. La plupart des actions définissent une interface de production vide. Par exemple, l'interface de production de l'action de translation est vide car la translation ne nécessite par un contrat particulier pour pouvoir être effectuée. Son existence n'est ici utile que pour typer fortement la production d'une action. Par contre, l'action de clonage est un exemple d'action à interface de production non vide : elle définit la méthode *pushClone(Component clone)* qui doit être invoquée par le consommateur sur le producteur afin que le consommateur lui fournisse son clone.

L'*interface de consommation* d'une classe d'actions A est implantée par toute classe de composants consommateurs qui définissent la façon dont ils consomment les instances de A. Elle définit les méthodes suivantes (figure 5) :

– La *méthode de faisabilité* effectue le test de faisabilité de l'action relativement au contexte courant.

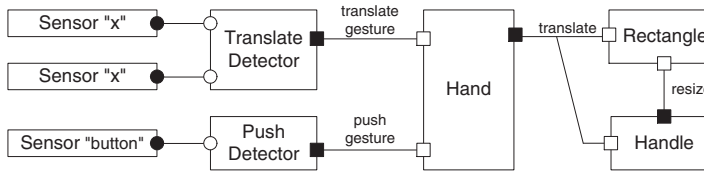

FIG. 4 – Exemple de l'instrument *main*

- La *méthode de début* est invoquée lorsque l'action démarre.
- La *méthode d'exécution* est régulièrement appelée pendant toute la durée de l'action.
- La *méthode de fin* est invoquée lorsque l'action se termine.
- La *méthode d'annulation* est appelée dès qu'une annulation de l'action est ultérieurement demandée par l'utilisateur (non représentée dans la figure 5).
- La *méthode de re-exécution* est appelée dès qu'une re-exécution d'une action préalablement annulée a été demandée (non représentée dans la figure 5).

Nous adoptons une syntaxe étendant UML afin de mettre en évidence les interfaces de production et de consommation (figure 4) :

- Le carré noir représente l'interface de production qui qualifie le composant de producteur de l'action.
- Le carré blanc représente l'interface de consommation qui qualifie le composant de consommateur potentiel de l'action.
- La *possibilité* de produire l'action depuis le producteur jusqu'au consommateur est symbolisée par le lien orienté de l'interface de production vers l'interface de consommation.

4.4 Application à l'interaction instrumentale

Le chaînage des actions de l'utilisateur vers le document passe immuablement par les instruments d'interaction. La détection des actions à caractère gestuel effectuées par l'utilisateur (ou *gestes*) est d'abord effectuée par la partie physique de l'instrument, sa transformation en actions intentionnelles (ou *actions*) étant ensuite prise en charge par sa partie logique.

La figure 4 illustre comment le modèle producteur - consommateur permet, conjointement au modèle observable - observateur, la mise en œuvre d'un tel chaînage en considérant l'exemple de l'action de translation productible par l'instrument *main*[3] et consommable par le rectangle ou l'une des poignées de

[3]L'instrument main représente le curseur classique des environnements graphiques et produit les actions génériques tel que le déplacement ou le glisser-déposer.

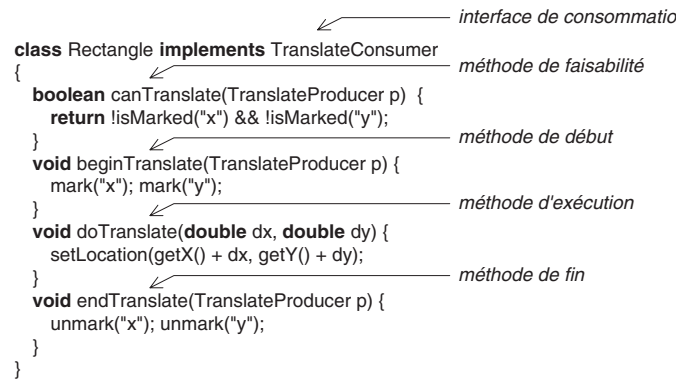

FIG. 5 – Interface de translation pour le rectangle

dimensionnement. La figure 5 précise conjointement la partie du code de la classe *Rectangle* qui implante l'interface de consommation de la translation :

1. L'instrument main est associé à la souris et observe les états propres de ses capteurs *x*, *y* et *bouton*. Cette observation est déléguée à deux composants détecteurs de geste : le détecteur de translation produit un geste de translation à chaque changement de l'état des capteurs de *x* ou *y*, et le détecteur d'appui produit un geste d'appui à chaque changement de l'état du capteur *bouton*. L'instrument consomme alors tout geste de translation et d'appui produit par les détecteurs.
2. Lorsque le geste d'appui démarre, l'instrument effectue un *piqué* qui consiste à déterminer quel composant graphique situé sous le curseur sera la cible de l'action de translation. Dans notre exemple, la cible est soit le rectangle, soit une poignée.
3. L'instrument entame la production de la translation vers la cible : la méthode de faisabilité indique que la translation peut être effectuée (nous supposons qu'il n'y a pas d'action concurrente) et la méthode de démarrage est invoquée sur la cible. La cible effectue alors un marquage des propriétés *x* et *y*.
4. Lorsque l'instrument consomme un geste de translation, il met à jour la position du curseur et invoque la méthode d'exécution de la translation sur la cible.
5. Lorsque le geste d'appui se termine, l'instrument invoque la méthode de fin sur la cible qui ôte alors le marquage précédent[4].

[4]La sauvegarde de données pour l'annulation de la translation n'est pas précisée ici mais est intégrée à ce cycle.

```
class Handle extends Rectangle
{
  void beginTranslate(TranslateProducer producer) {
    super.beginTranslate(producer, user);
    resize.beginAction(editedGraphic);
  }
  void endTranslate(TranslateProducer producer) {
    super.endTranslate(producer);
    resize.endAction();
  }
  void doTranslate(double dx, double dy) {
    super.doTranslate(dx, dy);
    // ...
    resize.doAction();
  }
}
```

châinage de la production transalte->resize

calcul de resize en fonction de (dx,dy)

FIG. 6 – Interface de translation pour la poignée

FIG. 7 – Adaptateur translation →rotation

Comme l'illustre cet exemple, les actions permettent de considérer indépendamment les instruments des éléments sur lesquels ils interagissent. Cet aspect a un impact non négligeable pour l'utilisateur : il peut utiliser un même instrument dans des contextes variés. Par exemple, l'instrument main produit de la *même* manière l'action de translation sur des éléments de nature différente. Par contre, la translation du rectangle n'engendre que son déplacement à l'écran (figure 5), alors que la translation d'une poignée engendre de plus une action de dimensionnement appliquée sur le rectangle (figure 6). L'action de translation est ainsi considérée comme une action *générique*, au sens défini dans [4].

4.5 Application à l'adaptation d'action

Il peut se produire des situations où une action, consommable par une présentation non créée par l'utilisateur, ne peut être produite par aucun de ses instruments. Plutôt que de se procurer un nouvel instrument, l'utilisateur peut préférer positionner un *adaptateur* sur l'instrument existant. La figure 7 illustre le principe de l'adaptateur translation →rotation :

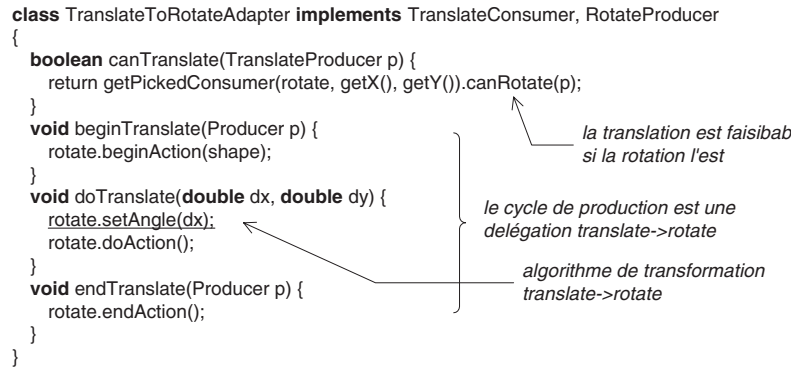

FIG. 8 – Adaptation d'une translation en une rotation

1. L'adaptateur est positionné sur l'instrument en connectant son interface de production de l'action de translation à l'interface de consommation de l'adaptateur.
2. Lorsque la main produit une translation, l'adaptateur transforme la propriété *deltaX* de la translation en une rotation *deltaAngle* qu'il peut alors produire sur le composant graphique ciblé.

L'implantation d'un adaptateur reste triviale (figure 8). Il est possible d'envisager que l'utilisateur définisse lui-même ses propres adaptateurs en fournissant des règles de transformation telle que celle soulignée dans la figure 8. Conjointement au polymorphisme des actions, l'adaptation accroît les possibilités d'utilisation des instruments.

4.6 Application au partage de documents

En adjoignant à notre modèle de document le concept de point de partage, le cycle de production et de consommation devient adapté au partage de composants et, par conséquent, au partage de documents.

Un *point de partage* est un objet invisible, identifié de manière unique par une adresse IP de groupe, et qu'un utilisateur peut attacher à un composant DPI graphique jouant alors le rôle de *container de partage* :

1. L'utilisateur qui crée le premier un container de partage donne accès, pour des utilisateurs distants, au *contenu* de ce container.
2. Les utilisateurs distants peuvent à leur tour associer leur propre container de partage au même point de partage. L'ensemble des utilisateurs possèdent ainsi des containers individuels qui ont tous le même contenu, car ils sont associés à un même point de partage.

3. Dès qu'un utilisateur effectue une action à l'intérieur de son container de partage, cette action est produite localement ainsi que sur l'ensemble des autres containers distants.

Le point 3 illustre que le cycle de production des actions doit prendre en compte l'existence des points de partage. En utilisant un tel concept, il devient possible de partager des présentations et des documents[5] dans des modes de collaboration variés [5].

5 Conclusion et perspectives

Nous avons proposé dans cet article un modèle de composants centré sur les documents et les instruments. Il vise à substituer la notion d'application à celle de composant logiciel au niveau des espaces de travail afin de s'affranchir des inconvénients que l'utilisation d'applications induit dans les environnements actuels.

L'ensemble du modèle a été implanté dans la boîte à outil expérimentale OpenDPI constitué de 250 classes Java. Elle utilise Piccolo[6] pour la gestion de l'affichage et propose une gestion des périphériques d'entrée multiples. Nous avons implanté un ensemble de composants interactifs habituellement complexes à réaliser[7]. En utilisant l'approche unifiée du modèle DPI, leur réalisation est homogène et se simplifie. Nous avons pu constater au travers de projets étudiant que, une fois les fondements de DPI assimilés, la conception de nouveaux composants s'avère aisée. Les composants DPI peuvent fonctionner dans des contextes variés : interaction bimanuelle [12], collaboration locale (ou « single display groupware » [18]) et collaboration distante. L'altération du rendu graphique est également intégrée à notre modèle. Elle permet notamment de réaliser des lentilles magiques [8] en garantissant la cohérence de l'interaction .

La prochaine étape de nos travaux consiste à valider la pertinence du modèle DPI en construisant une suite logicielle définissant un espace de travail interactif et collaboratif dédié à des tâches spécifiques. Cet outil permettra de qualifier davantage l'approche masquant les applications ainsi que les problèmes (nouveaux) qui en découleraient. Parallèlement à cette étape, nous affinerons le modèle DPI en le basant sur des standards reconnus. Le modèle de document s'appuiera sur DOM et le modèle de graphe de scène sur le standard SVG [21], lui-même instance de DOM. Nous compléterons ainsi l'approche DOM en spécifiant comment les facettes (inter)action et collaboration présentes dans DPI peuvent y figurer.

---

[5] Les différents aspects spécifiques aux points de partage, tels que le choix d'une architecture répliquée et la gestion de la cohérence, sortent du cadre de cet article.

[6] http ://www.cs.umd.edu/hcil/piccolo

[7] Nous invitons le lecteur à tester les 20 scénarios proposés dans OpenDPI (http ://www.eseo.fr/ ~obeaudoux/ opendpi).

RÉFÉRENCES

[1] Apple. Opendoc technical summary. Technical documentation, Apple Computer Inc., 1994.

[2] Björn E. Backlund. OOE : A compound document framework. *ACM SIGCHI Bulletin*, 29(1) :68–75, Jan. 1997.

[3] M. Beaudouin-Lafon. Instrumental interaction : An interaction model for designing post-wimp interfaces. In *Proc. CHI'00*, pages 446–453. ACM Press, 2000.

[4] M. Beaudouin-Lafon et W.E. Mackay. Reification, polymorphism and reuse : Three principles for designing visual interfaces. In *Proc. AVI'00*, pages 102–109. ACM Press, may 2000.

[5] O. Beaudoux. Un pivot pour la collaboration : les places publiques. In *Annexes des actes IHM'02*, 2002.

[6] O. Beaudoux. *Espaces de travail interactifs et collaboratifs : vers un modèle centré sur les documents et les instruments d'interaction*. PhD thesis, LRI, Université d'Orsay - Paris XI, 2004.

[7] O. Beaudoux et M. Beaudouin-Lafon. DPI : A conceptual model based on documents and interaction instruments. In *Proc. IHM-HCI'01*, pages 247–263. Springer Verlag, 2001.

[8] Eric A. Bier, Maureen C. Stone, Ken Pier, William Buxton et Tony D. DeRose. Toolglass and magic lenses : the see-through interface. In *Proc. of SIGGRAPH'93*, pages 73–80. ACM Press, 1993.

[9] Kraig Brockschmidt. *Inside OLE, Second Edition*. Microsoft Press, 1995.

[10] J. Buchner. HotDoc : a framework for compound documents. *ACM Computing Surveys*, 32(1) :33–38, 2000.

[11] E. Gamma, R. Helm, R. Johnson et J. Vlissides. *Design Patterns : Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1994.

[12] Y. Guiard. Asymmetric division of labor in human skilled bimanual action : The kinematic chain as a model. *Journal of Motor Behavior*, 4 :486–517, 1987.

[13] J. Johnson, T. L. Roberts, W. Verplank, D. C. Smith, C. Irby, M. Beard et K. Mackey. The Xerox Star : A retrospective. *IEEE Computer*, 22(9) :11–29, 1989.

[14] G. E. Krasner et S. T. Pope. A cookbook for using the Model-View-Controller user interface paradigm in Smalltalk-80. *Journal of Objet Oriented Programming*, pages 26–49, 1988.

[15] C. Marlin. Multiple views based on unparsing canonical representations - the multiview architecture. In *Proc. of SIGSOFT '96 workshops*, pages 222–226. ACM Press, 1996.

[16] J.P. Sansonnet N. Sabouret. Un langage de description de composants actifs pour le Web sémantique. *Revue Information - Interaction - Intelligence (RI3)*, 3(2) :9–36, 2003.

[17] R. T. P(édauque). Document : forme, signe et médium, les reformulations du numérique. Document de travail collectif, STIC-CNRS, 2003.

[18] Jason Stewart, Benjamin B. Bederson et Allison Druin. Single display groupware : a model for co-present collaboration. In *Proc. CHI'99*, pages 286–293. ACM Press, 1999.

[19] Sun. JavaBeans API specification. Specification document, 1997.

[20] W3C. Document object model (DOM) level 2 specification. W3c recommendation, Consortium W3C, 2002.

[21] W3C. Scalable vector graphics (SVG) 1.1 specification. W3c recommendation, Consortium W3C, 2003.