DoPIdom : Une boîte à outils pour la conception d'interfaces centrées sur les documents XML

Olivier Beaudoux

ESEO, Département Informatique &
Laboratoire de Recherche en Informatique / INRIA Futurs*
4 rue Merlet de la Boulaye
49009 Angers, France
olivier.beaudoux@eseo.fr

RESUME
La notion de document évoque le plus souvent les données que le document contient mais rarement les 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 décrivons dans cet article une boîte à outils *DoPIdom* dédiée à la réalisation d'environnement interactifs centrés sur les documents XML. Elle s'appuie sur le modèle DPI (Document, Présentation, Instrument) qui vise à masquer le concept d'application aux utilisateurs.

MOTS CLES : Modèle de documents, modèle d'intéraction, composants logiciels, boîtes à outil

ABSTRACT
The concept of document usually refers to data that documents contain, but rarely with actions that documents can handle. As a consequence, it is hard to specify how users can interact with documents: this is mainly done with dedicated applications that users have to juggle in order to achieve a single editing task. This paper presents *DoPIdom*, a user-interface toolkit designed to create interactive environments centered on XML documents. It is based on the DPI (Document, Presentation, Instrument) model, which aims at hidding the concept of application from the user point of view.

CATEGORIES AND SUBJECT DESCRIPTORS: H.1 [Models and Principles]: User/Machine Systems; H.4 [Information Systems Applications]: Office Automation - *Desktop publishing*

GENERAL TERMS: Interaction and document design

IHM 2004 Aug 30 – Sep 3, 2004, Namur, Belgium

KEYWORDS : Document model, interaction model, software components, GUI toolkits

INTRODUCTION
Les documents sont devenus omniprésents dans les environnements interactifs et, s'ils peuvent être lus indépendamment d'une application de par l'existence de formats standards, ils ne peuvent être édités sans avoir recours à des applications dédiées. L'utilisateur est ainsi contraint d'interagir avec le document en jonglant avec plusieurs applications, rendant l'interaction plus complexe. Trois stratégies sont actuellement mises en œuvre par les éditeurs de logiciels afin de réduire cette complexité [3, 4] : utilisation de « mini-applications » à l'intérieur d'applications plus larges, mise en œuvre d'une interface homogène pour les applications d'une suite logicielle, et extension possible d'une application par l'utilisation de « plug-ins ». Cependant, si elles améliorent la situation, ces stratégies restent finalement centrées sur les applications. Le système OpenDoc [1] a historiquement défini une architecture intégralement centrée sur les documents. Cependant, ses « composants » sont d'une granularité élevée, ne définissent pas de modèle d'interaction, et ne séparent pas les données de leurs représentations. Ces points constituent probablement les raisons pour lesquelles cette architecture est devenue obsolète.

Afin de pouvoir construire des environnements interactifs masquant les applications, nous avons conçu le modèle conceptuel DPI (Document, Présentation, Instrument) [4], puis nous en avons fourni une implantation expérimentale OpenDPI [3]. Nous avons alors identifié deux limitations majeures : OpenDPI ne permet pas de traiter un document XML ayant une grammaire quelconque, et elle impose une utilisation intensive de l'héritage entre classes de composant, là où il est préférable d'utiliser la composition [6]. Nous proposons dans cet article une implantation du modèle DPI, appelée *DoPIdom*[1], basée sur deux standards reconnus, DOM [7] pour les documents et SVG [8] (instance de DOM) pour

[1] http://www.eseo.fr/~obeaudoux/dopidom

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

leurs présentations. En adjoignant à ces standards un modèle de composants (inter)actifs DPI, DoPIdom permet de spécifier dans tout documents XML les actions qu'ils peuvent consommer. Cette implantation sépare clairement le comportement et la représentation des composants, dans une approche privilégiant la composition à l'héritage. Dans les deux sections suivantes, nous présentons le modèle des composants DPI puis un exemple de réalisation détaillé et en nous limitant à la présentation en SVG des composants DPI.

MODELE DE COMPOSANT

Documents et instruments

Le principe du modèle DPI est résumé sur la figure 1. Le document se décompose en séparant ses *présentations* et ses *données du domaine*, ces deux parties restant synchronisées. L'utilisateur, afin d'agir sur les données du domaine du document, interagit à l'aide d'un *instrument* [2] sur l'une de ses présentations. Une présentation est constituée de composants interactifs qui définissent chacun un ensemble d'actions *consommables*, tandis qu'un instrument est un composant interactif qui définit l'ensemble des actions *productibles*. La compatibilité action produite / action consommée permet alors aux instruments de produire les actions désirées sur les composants susceptibles de les consommer.

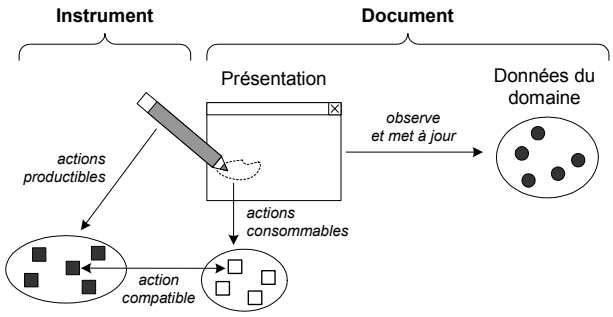

Figure 1: Principe du modèle DPI

Dans les document DoPIdom, les données du domaine sont décrites par des composants actifs associés à des nœuds DOM, et les présentations par des composants interactifs associés à des nœuds SVG. L'ensemble des nœuds SVG affichés à l'écran forme la *scène*. Dans cet article, nous présentons uniquement les composants interactifs associés à la scène SVG.

Composants interactifs

Un *composant interactif* est un objet attaché à un nœud SVG de la scène et auquel est adjoint un comportement définissant la production et la consommation d'actions et de requêtes. Les actions se distinguent des requêtes par le fait qu'elles modifient l'état du composant consommateur et ne renvoient pas de réponse au producteur, alors que les requêtes ne modifient pas l'état du consommateur mais renvoient un résultat. Par exemple, le fait de peindre définit l'action *paint* alors que le fait de récupérer une couleur définit la requête *pick-color*.

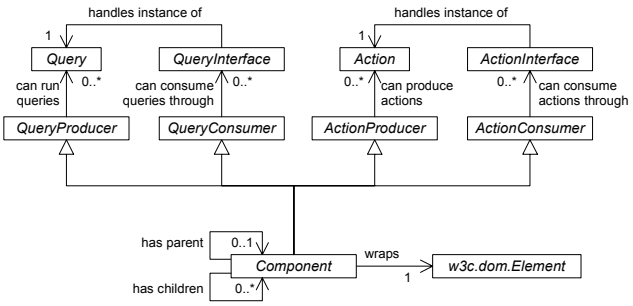

Figure 2: Modèle UML des composants interactifs

Le schéma de production / consommation est semblable pour les actions et les requêtes (figure 2) : chaque consommateur définit des *interfaces de consommation* qui permettent aux producteurs de produire les actions ou requêtes selon le modèle client / serveur.

SCENARIO ILLUSTRATIF

Nous illustrons plus précisément les principes précédents en considérant la réalisation avec DoPIdom du scénario d'application d'une couleur à des formes géométriques.

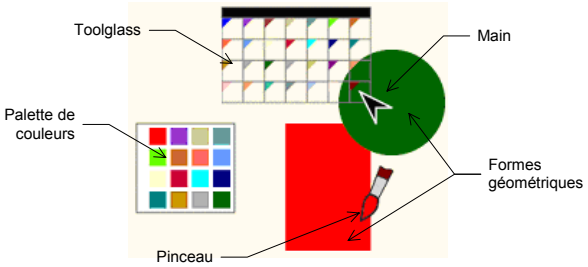

Figure 3: Application d'une couleur à des formes géométriques

La figure 3 représente les différents composants interactifs utilisés dans ce scénario. Le cercle et le rectangle peuvent être peints en utilisant un « toolglass » [5] ou un pinceau. Le toolglass est manipulé en utilisant l'outil main symbolisé par le curseur triangulaire. Le pinceau peut sélectionner une couleur sur la palette de couleurs.

Définition des composants

La première phase de conception du scénario consiste à spécifier les composants interactifs. La figure 4 montre les interfaces de consommation des six composants du scénario. Les deux actions principales du scénario y apparaissent : l'action *paint* permet l'application d'une couleur et la requête *pick-color* permet de récupérer la couleur d'un composant.

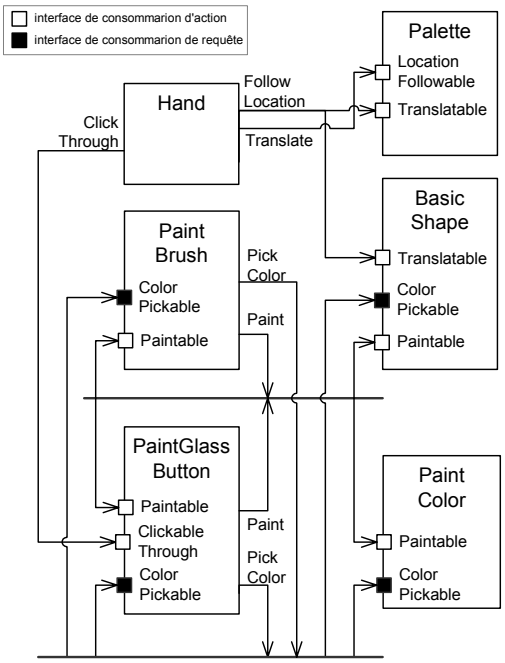

Figure 4: Les composants interactifs du scénario

Le *paint-brush*, tout comme les boutons du toolglass (*paint-glass-button*), peuvent produire l'action *paint* et la requête *pick-color*. Les interfaces de consommation *paintable* et *color-pickable* permettent leur consommation respective. Elles sont assemblées par composition (et non par héritage) dans les composants *basic-shape*, *paint-color*, *paint-brush* et *paint-glass-button*, autorisant alors des combinaisons interactives intéressantes : par exemple, le pinceau peut sélectionner sa couleur courante classiquement sur la palette de couleur, mais aussi sur une forme élémentaire, un toolglass et même un autre pinceau. Les *paint-color* et les *paint-glass-button* sont placés dans des palettes qui peuvent être translatées par l'outil *hand*, ainsi qu'être associées à cette main de manière à suivre en permanence sa position[2].

Définition de la scène

La deuxième phase de réalisation du scénario consiste à préciser comment les composants sont visualisés dans le graphe de scène SVG. L'association composant DPI / élément SVG se fait par l'ajout de l'attribut *dpi:component* dans l'élément considéré, comme l'illustre la scène (résumée) suivante:

```
<svg ...>
  <!-- Basic shapes -->
  <g>
    <rect dpi:component="BasicShape" fill="red" …/>
    <circle dpi:component="BasicShape" fill="darkgreen"…/>
  </g>
  <!-- Color palette -->
  <g dpi:component="Palette" tranform="translate(10,120)">
    <rect fill="white" stroke="black" stroke-width="1" …/>
    <g transform="translate(12,4)">
      <use href="#PaintColor" x=" 0" y=" 0" fill="red" />
      <use href="#PaintColor" x=" 20" y=" 0" fill="blueviolet" />
      …
    </g>
  </g>
  <!-- Color toolglass -->
  <g dpi:component="Palette" transform="translate(90,10)">
    <rect width="140" height="10" fill="black" …/>
    <g transform="translate(0,10)">
      <use href="#PaintGlassButton" x="0" y="0" fill="blue" />
      <use href="#PaintGlassButton" x="20" y="0" fill="violet"/>
    </g>
  </g>
  <!-- Tools -->
  <polygon dpi:component="Hand" fill="black" points="…" … />
  <use href="#PaintBrush" … fill="red"/>
  <!-- Symbols -->
  <defs>
    <symbol id="PaintColor" dpi:component="PaintColor" >
      <rect width="16" height="16" stroke-width="1" />
    </symbol>
    <symbol id="PaintGlassButton" dpi:component="PaintGlassButton"
>
      <polygon points="0,0 10,0 0,10" />
      <rect width="20" height="20" opacity="0" fill="white"/>
      ...
    </symbol>
    <symbol id="PaintBrush" dpi:component="PaintBrush" …>
      <path d="M 17 30 C 17 30 9 27 9 27 … " />
      ...
    </symbol>
  </defs>
</svg>
```

Le cercle et le rectangle sont deux éléments associés à deux composants instances de *basic-shape*. La palette de couleur est un groupe <*g*> constituée de couleurs (*paint-color*) définies par un symbole <*symbol*> associé à un composant DPI. Le toolglass est réalisé selon le même principe. Les deux outils *hand* et *paint-brush* sont deux composants DPI. Cet exemple illustre l'intérêt de séparer la représentation du comportement : l'approche tire bénéfice du standard SVG (instanciation de symboles, utilisation de styles CSS ou de XSL, utilisation d'applications telle que Illustrator) en y adjoignant un comportement décrit en langage objet (Java).

Définition des comportements

La troisième et dernière phase consiste à définir les comportements des composants au travers de classes Java spécifiées dans les attributs *dpi:component*. Les composants sont construits par assemblage d'interfaces de consommation. Par exemple, le composant *basic-shape* assemble les interfaces *translatable*, *paintable* et *color-pickable* comme suit (le lien composant-interface est réalisé grâce au passage du paramètre *this*):

```
public class BasicShape extends SVGComponent {
  public BasicShape() {
    new Translatable(this);
    new Paintable(this);
    new ColorPickable(this);
  }
```

[2] Nous n'abordons pas dans cet article la possibilité d'utiliser plusieurs périphériques de pointage avec DoPIdom.

Les instruments jouent le rôle de producteur et précisent la manière dont ils produisent les actions et requêtes sur les consommateurs. Par exemple, le composant *paint-brush* capture les clics souris et produit l'action *paint* ou la requête *pick-color* sur tout composant situé sous l'outil et pouvant consommer cette action ou cette requête, comme suit :

```
public class PaintBrush extends SVGToolInteractor {
  public PaintBrush() {
    new Paintable(this);
    new ColorPickable(this);
  }

  public void mousePressed(MouseEvent e) {
    int x = e.getX(), y= e.getY(), b= e.getButton() ;
    // picking sur la scène à la position du pinceau
    SVGPicking p = SVGScene.getPickedElements(x,y);
    // l'action paint est produite à l'issue d'un clic gauche
    // et si un consomateur compatible a été piqué
    if ((b==1) && p.hasActionConsumer(Paint.class))
      doActionOn(
        new Paint(getWrappedElement().getAttribute("fill"))),
        p.getActionConsumer(Paint.class));
    // la requête pick-color est exécutée à l'issue d'un clic
    // droit et si un consomateur compatible a été piqué
    else if ((b==3) && p.hasQueryConsumer(PickColor.class)) {
        PickColor pickColor = new PickColor();
        doQueryOn(pickColor,
            p.getQueryConsumer(PickColor.class));
        // la couleur retournée sert à  peindre le pinceau
        doActionOn(new Paint(pickColor.getColor()), this);
    }
  }
}
```

Enfin, les interfaces de consommation définissent comment actions et requêtes peuvent être consommées. Par exemple, l'interface *paintable* est implantée comme suit :

```
public class Paintable extends SVGActionInterface {
  public Class getAction() {  return Paint.class;  }

  public Class[] getScope() {
    return new Class[]{SVGStylable.class};
  }

  public void doAction(Action action) {
    SVGStylable stylable = (SVGStylable) getWrappedElement();
    CSSValue fill = stylable.getPresentationAttribute("fill");
    fill.setCssText(((Paint) action).getColorName());
  }
}
```

Les méthodes *getAction* et *getScope* précisent l'action gérée par l'interface et les classes interfaces que l'élément SVG lié au composant DPI doit implanter. Elles forment le *contrat d'assemblage* de l'interface dans un composant. La méthode *doAction*, par des forçages de type respectant le contrat précédent, met à jour la valeur de l'attribut *fill* . Cette méthode est invoquée, par exemple, *via* la méthode *doActionOn* du *paint-brush*.

CONCLUSION

Nous avons présenté dans cet article la manière dont la boîte à outils DoPIdom permet de spécifier les actions sur les documents XML. Nous avons présenté le principe des composants interactifs DPI liés à la scène SVG. DoPIdom gère également les composants actifs DOM liés aux composants interactifs SVG par un mécanisme de synchronisation non abordé ici. Par ailleurs, DoPIdom propose un mécanisme de production / consommation d'actions complet qui permet la gestion de l'annulation, la gestion des actions locales concurrentes (interaction bimanuelle et « single display groupware ») ainsi que la réplication d'actions pour la collaboration à distance. Elle inclut également la notion de points d'événement (*event-points*) qui permet d'aborder différents aspects, tels que le rendu altéré de la scène, la collaboration dans une architecture répliquée et la gestion d'historiques des actions, d'une manière unifiée. La prochaine étape de nos travaux consiste à utiliser DoPIdom afin de bâtir différents composants DPI d'un environnement interactif. Un tel environnement permettra de mesurer la pertinence de l'approche ainsi que de découvrir les nouveaux problèmes qu'elle pourrait engendrer.

BIBLIOGRAPHIE

1. Apple. *OpenDoc technical summary*. Apple Computer Inc., 1994.
2. Beaudouin-Lafon, M. Instrumental interaction : An interaction model for designing post-wimp interfaces. In *Proc. CHI'00*, pp. 446-453. ACM Press, 2000.
3. Beaudoux, O. *Un modèle de composants (inter)actifs centré sur les documents*. Revue I3, 4(1), 2004.
4. Beaudoux, O. et Beaudouin-Lafon M. DPI : A conceptual model based on documents and interaction instruments. In *Proc. IHM-HCI'01*, pp. 247-263, Springer Verlag, 2001.
5. Bier, E. A., Stone, M. C., Pier, K., Buxton, W. and DeRose, T. D. Toolglass and magic lenses : the see-through interface. In *Proc. of SIGGRAPH'93*, pp. 73–80. ACM Press, 1993.
6. Szyperski, C. *Component software : Beyond object-oriented programming*, 2nd edition. Addison-Wesley, 2002.
7. W3C. *Document object model (DOM) level 2 specification*. W3C recommendation, 2002.
8. W3C. *Scalable vector graphics (SVG) 1.1 specification*. W3C recommendation, 2003.