Prima, nouvelle boîte à outils Perl pour les interfaces graphiques

Dmitry Karasik

Ecrit en Perl, Prima est censé être utilisé dans la plus pure tradition du codage en Perl, dont les maîtres mots sont fait ce que je pens ou il existe plus d'une façon de programmer.

La programmation des interfaces graphiques est souvent considérée comme une tâche quelque peu pénible, non sans raison. Savoir que vos boutons et vos barres de défilement fonctionneront exactement comme des millions d'autres boutons et barres de défilement n'est guère gratifiant. Ainsi, quelle que soit le choix de votre boîte à outils d'interface graphique, il s'agit d'un outil de piètre importance, et le moins gênant sera le mieux. Ceci étant dit, nous tenterons, dans le présent article, d'appliquer le fameux mantra de Perl, à savoir *faciliter les choses simples et rendre possible les choses difficiles*, sous forme de tutorial introductif visant à faciliter la programmation au moyen de Prima, nouvelle boîte à outils Perl pour les interfaces graphiques.

Cette boîte à outil est *officiellement* décrite comme *un outil pour interface graphique extensible et grand public doté d'un large choix de widgets standard et plus particulièrement spécialisé dans les tâches de traitement des images en 2D*. Toujours selon la description officielle, *l'apparence et la convivialité d'un programme Perl rédigé grâce à Prima seront strictement identiques sur X11, Win32 et OS/2*. Cette description ne parle pas, en revanche, de ce qui distingue de prime abord cette boîte à outils des autres outils Perl pour les interfaces graphiques, ni de Perl-Tk. Prima a débuté, en réalité sur OS/2, sur lequel il était impossible de lancer Perl-Tk. A cette époque, deux options étaient envisageables : soit porter Perl-Tk, soit écrire son propre programme, probablement meilleur que les outils existants. Nous avons opté pour cette dernière solution avec assez de réussite. Prima se distingue également des autres boîtes à outils Perl pour interfaces graphiques dans la mesure où notre solution n'a recours à aucune bibliothèque d'interface graphique tierce afin d'accéder aux ressources graphiques, comme les liens Perl vers Qt et Wx, ou les bibliothèques GTK.

Sur l'auteur :

Développeur de logiciels, Dmitry Karasik *dmitry@karasik.eu.org* est un des auteurs de la boîte à outils Prima.

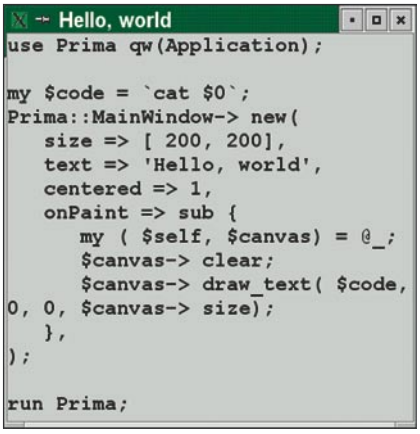

Figure 1. *Programme Hello world*

Application Hello world

Perl est le langage de programmation (sans conteste) le plus efficace sur les petits programmes, dans la mesure où le programmeur n'a nul besoin d'inclure des lignes et des lignes de code préalable avant même d'atteindre le problème en question. Bien évidemment, Prima ne saurait rivaliser avec une telle capacité, mais la charge de travail introductif est relativement minime. Ainsi, par exemple, une simple application Hello world nécessite trois lignes de code uniquement :

```
use Prima qw(Application);
Prima::MainWindow-> new( text => 'Hello
                      world!');
run Prima;
```

La première ligne représente l'invocation des modules `Prima` et `Prima::Application`. Ils peuvent être évoqués de manière explicite au moyen des déclarations `use Prima` et `use Prima::Application`, mais, dans la mesure où le module `Prima` n'exporte pas les noms de méthodes, une telle compression est particulièrement bien adaptée dans ce contexte.

Listing 1. *Code source du programme chargé d'invoquer la boîte de dialogue standard pour les couleurs*

```
use Prima qw(StdDlg Application);
my $dialog = Prima::ColorDialog->
                      new();
printf "Color selected:%06x\n",
                      $dialog->
                      value
   if $dialog-> execute == mb::Ok;
```

La deuxième ligne permet de créer une fenêtre de la classe `Prima::MainWindow`, affichée sous forme de fenêtre d'écran intitulée *Hello world*. Cette classe termine l'application à la fermeture de la fenêtre ; il s'agit de l'unique différence des fenêtres de classe `Prima::Window`, censées ne rien faire une fois fermées.

La troisième ligne, enfin, permet d'introduire la boucle d'évènement Prima. Cette boucle se termine lorsque la seule instance de la classe `Prima::Application`, créée en invoquant `use Prima::Application` et stockée dans le scalaire `$::application`, est détruite.

Vous pouvez désormais modifier quelque peu ce programme afin de le rendre plus utile, en affichant ses propres sources, par exemple. Vous trouverez dans la Figure 1 le résultat et le code source.

Les lignes 4 à 13 permettent de créer une fenêtre légèrement différente. `new()` (alias `create()`) est la méthode constructeur de tous les objets Prima. Cette méthode prend une liste de noms de propriétés et des paires de valeurs correspondant à ces propriétés en paramètres. Dans notre exemple, quatre propriétés ont été paramétrées : `text` et `centered` du type Perl `SCALAR`, la propriété `size` de la référence `ARRAY`, et la propriété `onPaint` de la référence `CODE`.

Plus de 100 propriétés sont disponibles. Elles peuvent être paramétrées pour un widget générique, et chaque classe dérivée de `Prima::Widget` ajoutera ses propres propriétés. En règle générale, il est inutile de spécifier les valeurs pour la plupart des propriétés, dans la mesure où elles sont automatiquement initialisées avec des valeurs par défaut acceptables.

Fenêtres de dialogue

L'élément le plus important sur toute boîte à outils d'interface graphique utilisateur est le mécanisme facilitant la création d'utilitaires d'interfaces classiques comme les boîtes de message, l'ouverture et la sauvegarde de fichiers, le choix des couleurs et des polices, etc..., généralement appelés *boîtes de dialogue*. Prima propose un ensemble de boîtes de dialogue, dont l'arborescence complète est illustrée dans la Figure 6. Nous avons exposé dans la Figure 2 une capture d'écran de la boîte de dialogue standard intitulée *Choisir la couleur*. Le Listing 1 expose un programme chargé d'invoquer la boîte de dialogue.

Nous avons exposé dans la Figure 3 un programme très simple d'affichage d'images basiques.

Le programme exposé dans le Listing 2 permet à l'utilisateur de choisir un fichier image au moyen de la boîte de dialogue standard d'ouverture de fichiers, de charger puis d'afficher l'image grâce à la classe du widget `Prima::ImageViewer`

L'utilisation de `Prima::ImageOpenDialog` est transparente. Vous aurez toutefois noté le positionnement implicite du widget `ImageViewer` grâce à la directive `pack`. Ce positionnement est un concept puissant emprunté à Tk, et permet de créer des couches de widgets sophistiqués sans avoir à indiquer les coordonnées spécifiques de chaque widget. Le déréférencement du widget par son nom `IV` dans la construction `$self->IV->image`, fonctionnalité propre à Prima, ne vous aura certainement pas échappé non plus.

Afin de faciliter le développement des interfaces graphiques d'applications, la boîte à outils Prima propose un programme de construction visuelle. Comme il est impossible de vous fournir le code source ici, nous avons exposé dans la Figure 4 une capture d'écran à la place.

Ce programme n'est guère différent des nombreuses autres applications partageant le même objectif, si ce n'est qu'il est écrit en pur Perl.

Images

Prima possède un large choix d'opérations et de transformations applicables aux mémoires tampons à deux dimensions des pixels (images), parmi lesquelles, la conversion entre différents formats d'images, le redimensionnement des images, et la combinaison de trames. Outre les formats de pixels à 1, 4, 8, et 24 bits, Prima supporte également les types de pixels `byte`, `short`, `long`, `float`, `double`, et `complex`. Lorsque le traitement d'une conversion d'image implique un sous-échantillonage des données pixels, un des quatre algorithmes de distribution d'erreur peut être sélectionné soit de manière automatique, soit selon les instructions formulées par le programmeur.

Prima peut charger et sauvegarder des images dans de nombreux formats de fichiers différents. La liste exacte que

Listing 2. *Code source du programme d'affichage d'images simples*

```
use Prima qw(StdDlg ImageViewer Application);
my $w = Prima::MainWindow-> new(
   menuItems => [
      [ file => '~File' => [
         [ '~Open' =>  'F3'     => kb::F3, \&load_image],
         [],
         [ 'E~xit' => 'Alt+X' => '@X' => sub { $::application-> close }],
      ]],
   ],
);

$w-> insert( ImageViewer =>
   pack => { expand => 1, fill => 'both'},
   name => 'IV',
);

sub load_image
{
   my $self = $_[0];
   my $dlg  = Prima::
         ImageOpenDialog-> new();
   my $i  = $dlg-> load;
   $self-> IV-> image( $i) if $i;
   $self-> text( $dlg-> fileName);
   $dlg-> destroy;
}
run Prima;
```

le noyau de Prima peut supporter est définie au moment de la compilation, en examinant les bibliothèques d'images (telles que libpng, libjpeg, ou libtiff) installées dans le système. Les distributions binaires de Prima peuvent charger et sauvegarder les formats d'images les plus répandus, comme GIF,PNG,JPG,TIFF, etc.

Trois classes permettant de manipuler les mémoires d'images sont disponibles. `Prima::Image` permet de charger, de sauvegarder, de convertir et d'afficher des mémoires d'images au format de pixel arbitraire. `Prima::Icon` est identique à la classe `Prima::Image` si ce n'est que cette classe a également recours à un masque

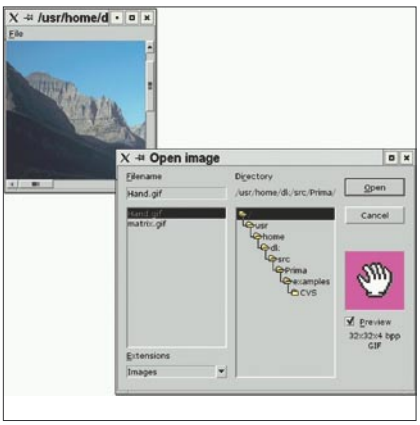

Figure 3. *Exemple de programme d'affichage d'images simples*

de transparence. Les objets de ces deux classes contiennent des tableaux de données sur les images directement accessibles par le code Perl. La classe `Prima::DeviceBitmap` représente la ressource d'une mémoire système d'images, dont le tableau de données des images est stocké dans la mémoire système, dans un format dépendant du système, que le programmeur ne peut accéder directement. Les objets de ces trois classes peuvent toutefois être convertis entre eux selon vos désirs.

Unicode

Prima supporte l'Unicode grâce à la bibliothèque Perl, avec laquelle l'outil est relié. Perl supporte de manière native l'Unicode depuis la version 5.6.0 en équipant chaque scalaire de chaîne d'un drapeau booléen chargé d'indiquer si les données d'une chaîne doivent être traitées en code ASCII ou UTF-8. Sur les systèmes le permettant, Prima propose un accès à la fonctionnalité Unicode propre au système, même avec les anciennes version de Perl. Toutefois, si Perl et le système d'exploitation supportent tout deux l'Unicode, Prima peut utiliser à la fois des chaînes codées en ASCII et en UTF-8, de manière transparente. Malgré certains problèmes d'utilisation simultanée des deux encoda-

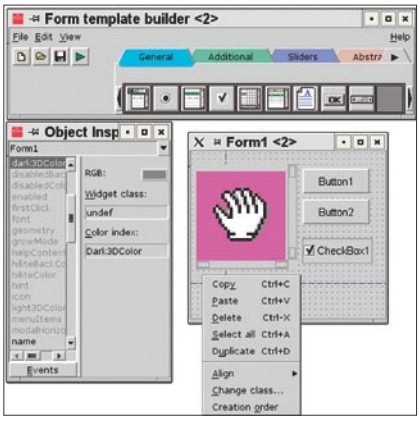

Figure 4. *Capture d'écran du constructeur visuel de Prima*

ges sous Perl 5, le niveau de l'interface de programmation XS n'en souffre pas, et donc Prima non plus. L'utilisation des chaînes codées en UTF-8 est direct sous Prima :

```
use charnames 'greek';
$window-> text("\N{Sigma} and \x{3C3} are
                        Greek sigmas");
```

Hiérarchie des classes

Le principal objectif de Prima consiste à proposer des services de fenêtrage et de graphismes élémentaires dans un cadre de classes Perl. Bien évidemment, certaines parties de Prima ont été rédigées en Perl, et d'autres en C. Dans la mesure où Prima est multi-plateforme, le code C est divisé en deux parties : une dépendant du système et l'autre complètement autonome.

La hiérarchie des classes centrales de Prima est exposée dans la Figure 5.

La classe racine `Prima::Object` fournit certaines méthodes fondamentales de construction, d'initialisation, de nettoyage, et de destruction pour toutes les classes de Prima. Elle ne comporte aucune autre fonction. La classe `Prima::Component` se charge d'ajouter des évènements et des processus de notification, et peut également posséder d'autres objets ou être possédée par eux. La classe abstraite `Prima::Drawable` est dotée d'une large gamme de méthodes de dessin. Les classes `Prima::Image` et `Prima::Widget` sont toutes deux lisibles directement après écriture.

La classe `Prima::Widget` est la plus importante dans la hiérarchie de Prima, puisque tous les widgets standard en sont des descendants. Les seuls descendants centraux de `Prima::Widget` sont `Prima::`

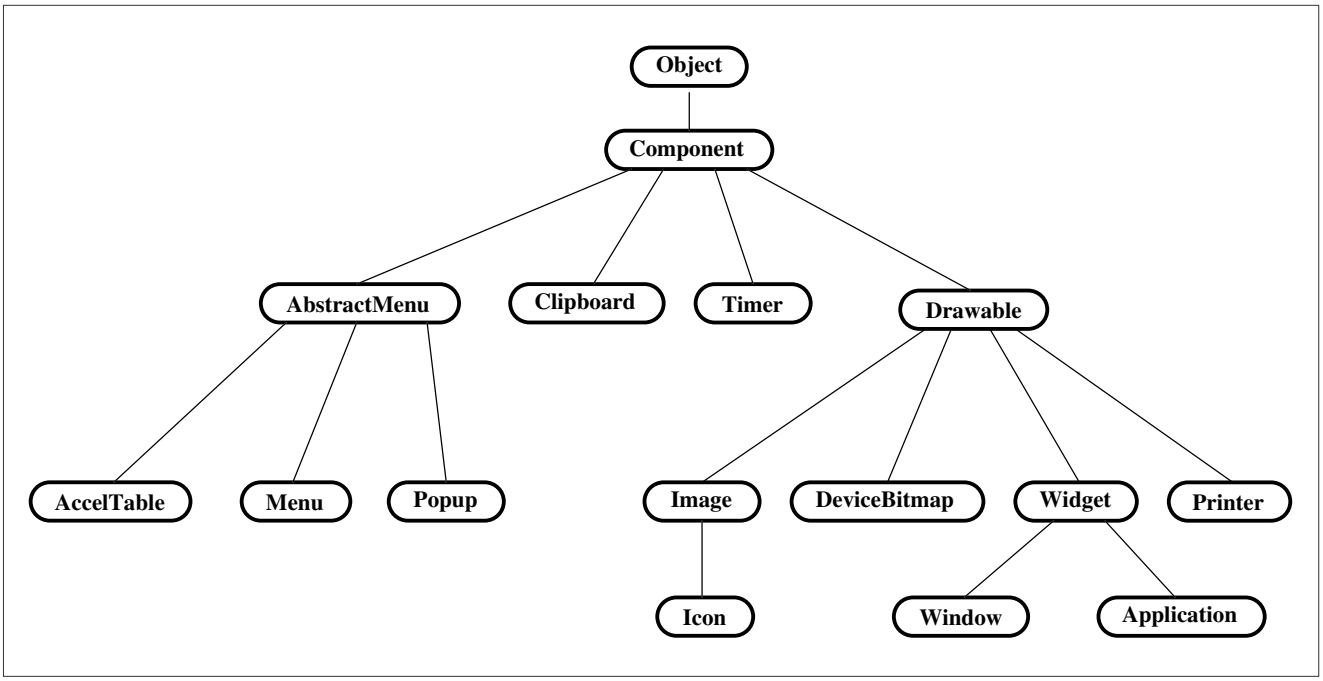

Figure 5. *Hiérarchie des classes centrales de Prima*

`Application` et `Prima::Window`, ce dernier étant chargé d'implémenter des fenêtres de niveau supérieur.

La distribution standard de Prima comprend de nombreuses classes Perl, chargées d'implémenter un ensemble de widgets généralement présents dans toutes boîtes à outils de fenêtrage : boutons, boîtes de contrôle et cases d'option, champs d'entrée, boîtes de liste, etc. La liste complète des widgets réellement installés, exposée dans la Figure 6, est trop importante pour pouvoir l'évoquer en détail dans le présent article. Prima est un projet libre dynamique, et de nouveaux widgets y sont régulièrement ajoutés.

Propriétés des objets

L'utilisation des propriétés des objets sous Prima est similaire à Perl-Tk. Est désigné par propriétés des objets, un ensemble de valeurs scalaires définies, dotées d'une signification et d'une fonction propres à chaque classe. Ces propriétés, en terme de programmation orientée objet, représentent des méthodes de classes pouvant être héritées. Un ensemble de fonctionnalités a été implémenté dans les propriétés des objets afin d'améliorer la flexibilité de la boîte à outils. Ces méthodes, déclarées sous forme de propriétés d'objet, se chargent de traiter le nombre de paramètres passés sous forme d'indications, afin de déterminer si la valeur de la propriété doit être rétablie (`my $name = $self->name`) ou paramétrée (`$self->name( $name)`).

En dépit du grand nombre de propriétés disponibles, il est inutile de spécifier les valeurs pour la plupart d'entre elles. Si un appel vers `new` ne contient pas la valeur d'une propriété, une valeur par défaut lui est substituée. La méthode `profile_default` retourne une table de hachage des valeurs par défaut de chaque propriété, utilisée dans ce but précis. Cette méthode permet également de modifier les valeurs par défaut de la classe parent en surchargeant la méthode standard.

Par souci de programmation pratique, les fonctions de certaines propriétés se chevauchent. Par exemple, la taille et la position d'un widget peuvent être indiquées de plusieurs manières :

```
rect => [ 10, 10, 100, 100] or
size => [ 90, 90], pos => [ 10, 10]
   or even
left => 10, right => 100, bottom =>
   100, top => 100
```

La propriété `font` fournit un autre exemple de type de données plus complexe traité par les classes Prima. Cette propriété se présente sous forme de référence à la table de hachage avec plusieurs noms clés autorisés, comme `name`, `size`, et `height`. Les propriétés relatives à la police de caractère peuvent être paramétrées de manière individuelle :

```
$widget-> font-> size( 18 )
```

ou regroupées :

```
$widget-> font-> set( size => 18, name =>
   'Helvetica', style => fs::Italic);
```

Autre propriété pratique, `name` de la classe `Prima::Component`, à double caractéristique. Cette propriété fournit à l'utilisateur une ancre à appliquer sur les paramètres personnels via des mécanismes dépendant du système comme la base de données des ressources X11. Elle facilite, d'autre part, au programmeur la référence aux composants fils : si une méthode inconnue d'un composant est appelée, `Prima::Component::AUTOLOAD` cherche un composant fils doté du nom de la méthode. Grâce à la propriété `name`, il est alors possible d'élaborer des constructions telles que :

```
$window-> insert( Button =>
   name => 'OKButton',
   ...
);
$window-> OKButton-> enabled( 0 );
```

Gestion des évènements

La boîte à outils Prima déclare la boucle d'évènements sous forme d'une seule fonction uniquement terminée lorsque la seule instance de la classe `Prima::Application` est détruite, de manière explicite en appelant `$::application->close`, ou de manière implicite lorsque l'utilisateur ferme la fenêtre `Prima::MainWindow`. Prima

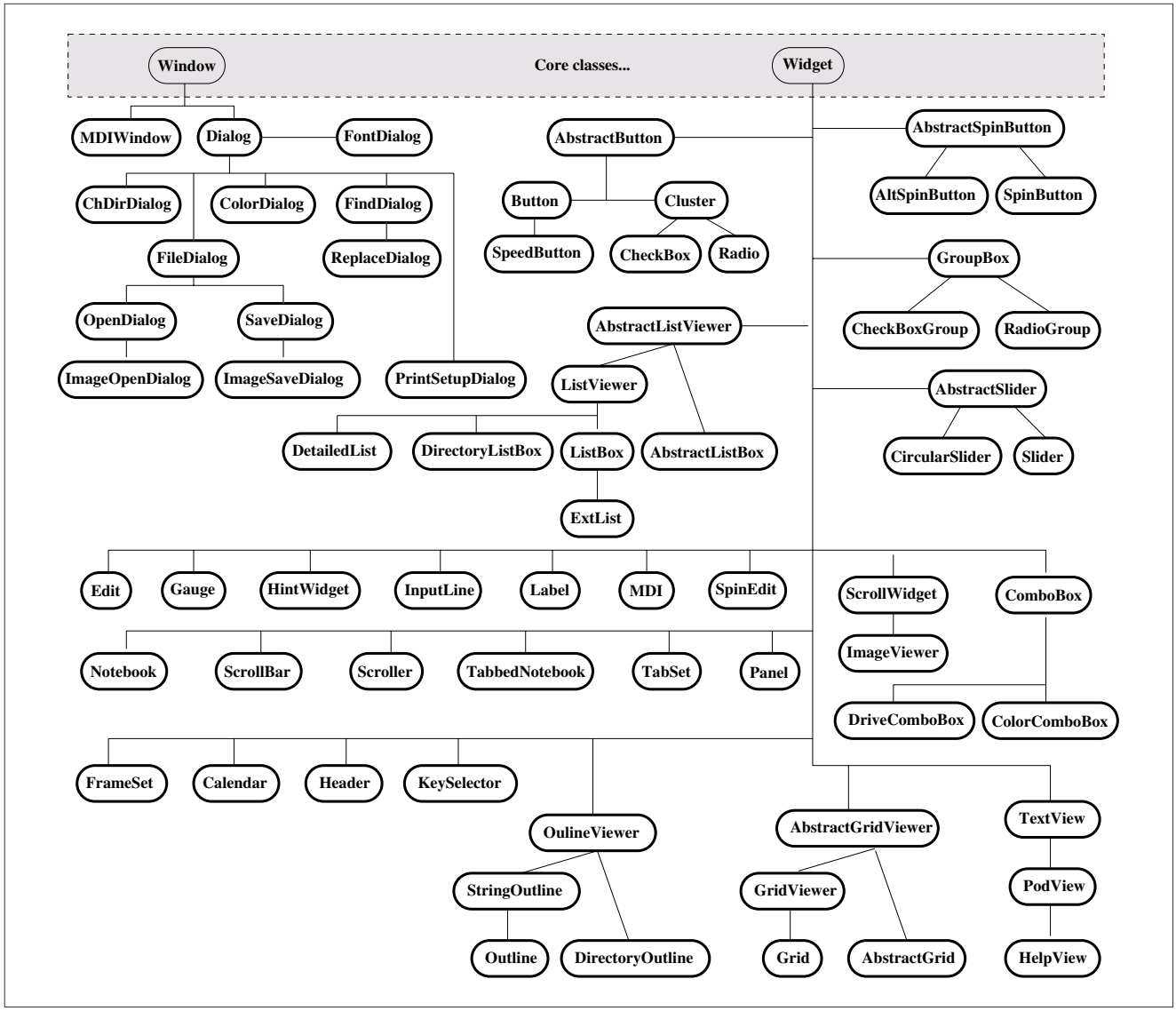

Figure 6. *Hiérarchie des widgets de Prima*

se charge de définir les sous-programmes de rappel des évènements sous forme d'un ensemble de méthodes virtuelles d'une classe, dans lequel la surcharge de méthode standard est pratiquée. Alors que presque toutes les bibliothèques d'interfaces graphiques ont recours de manière extensive à cette approche, nous avons exposé dans la Figure 1 un autre type de surcharge de sous-programmes de rappel. Ici, le rappel `onPaint` est invoqué à chaque retraçage nécessaire des fenêtres. Ce procédé d'invocation est similaire à d'autres architectures d'interfaces graphiques, à quelques différences près : tout d'abord, la fonction de rappel peut être un sous-programme anonyme ou une méthode surchargée, et deuxièmement, il peut y avoir plus d'une fonction de rappel anonyme pour chaque type de notification, ce qui se révèle plus utile.

Lorsqu'un ensemble de paramètres nommés est passé dans une fonction sous forme de table de hachage, la technique standard Perl est alors utilisée dans les fonctions à plusieurs paramètres `new` et `set`, capables de configurer une instance d'un objet au moyen d'un seul appel. Cette technique est également exposée dans la Figure 1 : les paramètres passés dans le constructeur de la classe contiennent toutes les informations requises pour l'initialisation d'une instance d'un objet.

Résumé

Le présent article vise à présenter les fonctionnalités de la boîte à outils Prima, ainsi que ses techniques de programmation parmi les plus fondamentales. Une documentation plus détaillée est fournie avec la boîte à outils. La liste des fonctionnalités propres à Prima, que nous n'avons

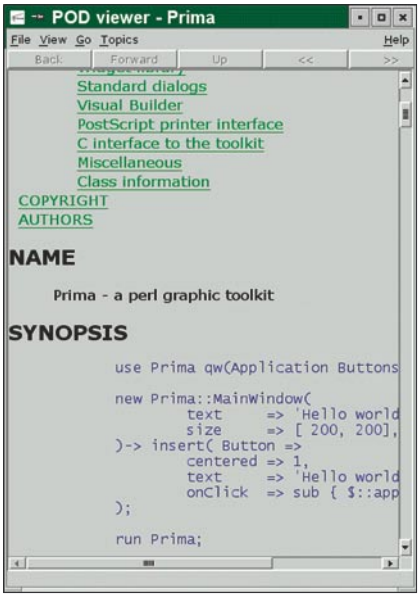

Figure 7. *Capture d'écran du navigateur POD de Prima*

pas pu aborder dans le présent article, comprend, entre autres, tous les éléments d'une interface implémentée en pur Perl, un sous-système de conversion d'images, un constructeur visuel, un navigateur POD (voir la Figure 7), et une interface d'impression PostScript(tm).

La boîte à outils est téléchargeable à partir de sa page d'accueil : *http://www.prima.eu.org/*. Vous pourrez également y consulter la liste de diffusion ainsi que de plus amples informations sur cet outil. ■