Modélisation Objet -- MOBJ

Contents

TME 7 -- Gestion des Symboles/Icônes 1
Modifications du Code du TME6 2
Code de la Classe `Box` 2
Dans la Classe `Cell` 2
Dans la Classe `Node` 2
Nouvelle Classe `Line` 2
Dans la Classe `Net` 3
La Nouvelle Classe `Shape` 3
Nouveaux Fichiers XML des Cellules 6
Programme Principal 6
Travail à Réaliser 6
Question 1 6
Question 2 6
Question 3 6

TME 7 -- Gestion des Symboles/Icônes

L'objectif de ce TME est d'implanter les classes `Symbol`, `Shape` et ses classes dérivées. Les *shapes*, assemblées dans un symbole définissent l'icône qui sera associée dans la visualisation à une instance d'un modèle donné. L'ossature des classes `Shape` a été présentée en cours et illustre les mécanismes d'héritage et de fonctions virtuelles.

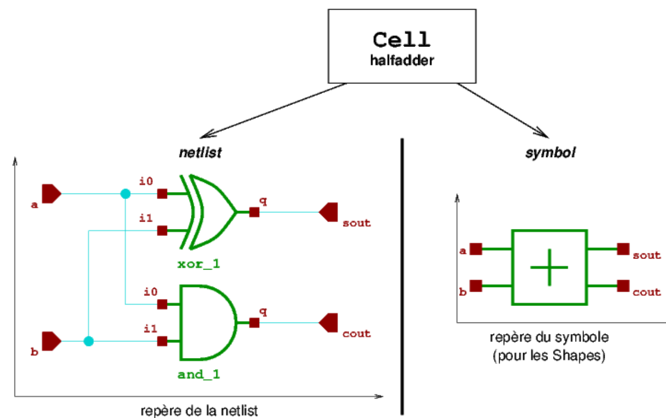

Figure 1: Figure 2 -- Netlist vs. Symbol

Une `Cell` aura donc deux repésentations, décrites dans le même fichier XML:

1. La *netlist*, qui décrit ce que contient la `Cell`, de quoi elle se compose.
2. Le *symbole* (ou l'icône) qui sera utilisée pour représenter cette `Cell` lorsqu'elle sera *instanciée*, pour construire des `Cell` de niveau hiérarchique supérieur, plus complexes.

Note
Chacune de ces deux représentations possède son propre système de coordonnées, on prendra un soin particulier à ne pas les confondre.

Modifications du Code du TME6

Code de la Classe `Box`

- Code de la classe `Box`, Box.h et Box.cpp.

Dans la Classe `Cell`

- Ajout d'un attribut `symbol_`, de type `Symbol` ainsi qu'un accesseur `Cell::getSymbol()`.
- Dans les méthodes `Cell::toXml()` et `Cell::fromXml()` la génération et la lecture du symbole en XML a été ajoutée.
- Code mis à jour de Cell.h et Cell.cpp.

Dans la Classe `Node`

- Deux types de noeuds peuvent maintenant être créés:
  1. Les `NodeTerm`, qui correspondent à ceux des TMEs précédents (associés à des `Term`).
  2. Les `NodePoint`, qui servent à créer des points purement géométriques sur lesquels ancrer les lignes (`Line`).
  3. Ces deux classes dérivent de la classe de base, virtuelle pure, `Node`.
- Les noeuds servent maintenant de point d'ancrage aux lignes et possèdent donc un tableau de (`vector<>`) des `Line` qui partent de ce point géométrique. Un nombre quelconque de lignes peut s'ancrer sur un même noeud.
- Code mis à jour de Node.h et Node.cpp.

Note
Dans *votre* implantation de la classe `Term`, remplacer le type de l'attribut `node_` par `NodeTerm`, et partout où ce type apparaît. Mais uniquement dans cette classe.
Ce changement n'affecte pas la classe `Net` dont les opérations s'effectuent sur la classe de base `Node`.

Nouvelle Classe `Line`

- Cette classe, qui représente une ligne entre deux `Node`, vous est fournie. Une ligne est un simple trait entre deux `Node` `source` et `target`. Dans le fichier XML elle repère ces deux noeuds par leurs index, et dans la représentation en mémoire, par les pointeurs sur ces mêmes noeuds.
- Code des fichiers Line.h et Line.cpp.

Dans la Classe Net

- En plus des `Node`, un `Net` doit connaître et gérer des `Line`.
- Il possède donc un nouvel attribut: `vector<Line*> lines_`.
- Et les méthodes associées:
  - Méthode `getLines()`:

    ```
    inline const std::vector<Line*>& Net::getLines () const
    { return lines_; }
    ```
  - Méthode `add()`:

    ```
    void  Net::add ( Line* line )
    { if (line) lines_.push_back( line ); }
    ```
  - Méthodes `remove()`:

    ```
    bool  Net::remove ( Line* line )
    {
      if (line) {
        for ( vector<Line*>::iterator il = lines_.begin()
            ; il != lines_.end() ; ++il ) {
          if (*il == line) {
            lines_.erase( il );
            return true;
          }
        }
      }
      return false;
    }
    ```
- Il vous reste à modifier les méthodes `Net::toXml()` et `Net::fromXml()`.

La Nouvelle Classe Shape

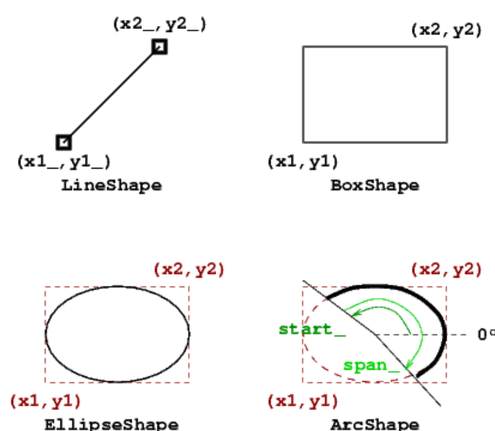

Figure 2: Figure 1 -- Tracé des classes `Shape`.

- La classe `LineShape`:
  1. Attributs: `x1_`, `y1_`, `x2_`, `y2_`.
  2. Balise (*tag*) XML:

     ```
     <line x1="-20" y1="10" x2="15" y2="10"/>
     ```

- La classe `BoxShape`:
  1. Attributs: `box_`.
  2. Balise (*tag*) XML:

     ```
     <box x1="20" y1="0" x2="100" y2="80"/>
     ```

- La classe `EllipseShape`: trace une ellipse inscrite dans la boite `box_`.
  1. Attributs: `box_`.
  2. Balise (*tag*) XML:

     ```
     <ellipse x1="10" y1="35" x2="20" y2="45"/>
     ```

- La classe `ArcShape`: trace une portion de l'ellipse inscrite dans la boite `box_`. Cette portion (arc) est définie par l'angle de départ (`start_`) et l'angle convert (`span_`). Les angles sont exprimés en degrés, l'angle *zéro* est à trois heures, les angles positifs sont dans le sens des aiguilles d'une montre.
  1. Attributs: `box_`, `start_` et `span_`.
  2. Balise (*tag*) XML:

     ```
     <arc x1="-80" y1="-20" x2="20" y2="80" start="-45" span="90"/>
     ```

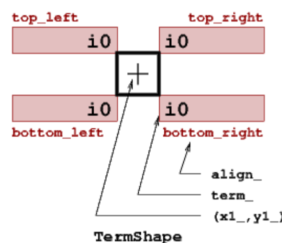

Figure 3: Figure 3 -- La classe `TermShape`.

- La classe `TermShape`: position d'un terminal du symbole. Dans le fichier XML, on repère le terminal par son nom. Ce nom doit bien sûr correspondre au nom d'un terminal de la `Cell`, défini dans la section `<terms>`. Dans l'objet, on ne stockera pas le nom mais directement un pointeur vers le `Term` associé (la méthode `::fromXml()` devra faire la conversion). Pour l'affichage du nom du terminal, on définit la position de la chaîne de caractères *par rapport* au point où il est placé avec l'alignement (quatre positions sont possibles).

1. Attributs:
   1. `x1_` et `y1_`.
   2. `term_`.
   3. `align_`. On définira pour celui-ci, l'`enum` suivant:

      ```
      enum NameAlign { TopLeft=1, TopRight, BottomLeft, BottomRight };
      ```
2. Balise (*tag*) XML:

   ```
   <term name="i0" x1="-20" y1="10" align="top_right"/>
   ```

- Méthode `Shape::fromXml()`: elle est un peu particulière, en terme d'ingénierie logicielle on parle de *Factory* (ou usine, dans les *Design Patterns*).

  Cette méthode, la seule qui devra être appelée depuis l'extérieur de la classe `Shape` et ses dérivées, se comporte comme une sorte d'aiguillage. Elle va déterminer à quelle `Shape` nous avons à faire, puis déléguer la conversion à la bonne classe dérivée. Son code vous est fourni:

```
Shape* Shape::fromXml ( Symbol* owner, xmlTextReaderPtr reader )
{
// Factory-like method.
  const xmlChar* boxTag
    = xmlTextReaderConstString( reader, (const xmlChar*)"box" );
  const xmlChar* ellipseTag
    = xmlTextReaderConstString( reader, (const xmlChar*)"ellipse" );
  const xmlChar* arcTag
    = xmlTextReaderConstString( reader, (const xmlChar*)"arc" );
  const xmlChar* lineTag
    = xmlTextReaderConstString( reader, (const xmlChar*)"line" );
  const xmlChar* termTag
    = xmlTextReaderConstString( reader, (const xmlChar*)"term" );
  const xmlChar* nodeName
      = xmlTextReaderConstLocalName( reader );

  Shape* shape = NULL;
  if (boxTag == nodeName)
    shape = BoxShape::fromXml( owner, reader );
  if (ellipseTag == nodeName)
    shape = EllipseShape::fromXml( owner, reader );
  if (arcTag == nodeName)
    shape = ArcShape::fromXml( owner, reader );
  if (lineTag == nodeName)
    shape = LineShape::fromXml( owner, reader );
  if (termTag == nodeName)
    shape = TermShape::fromXml( owner, reader );

  if (shape == NULL)
    cerr << "[ERROR] Unknown or misplaced tag <" << nodeName << "> (line:"
         << xmlTextReaderGetParserLineNumber(reader) << ")." << endl;

  return shape;
}
```

Nouveaux Fichiers XML des Cellules

Ces fichiers remplacent les précédents, ils contiennent une description des symboles des portes ainsi que le dessin complet des fils du `halfadder`. Les `Net` du `halfadder` contiennent donc des nodes pour les `Term`, des nodes pour les points simples et des `Line` à tirer entre ces points.

Archive: cells.tar.gz

Décompactage de l'archive:

```
etudiant@pc:~> cd MON_TME7/work
etudiant@pc:work> tar zxvf cells.tar.gz
```

Programme Principal

Le programme de test: Main.cpp.

Travail à Réaliser

En prélude à la compilation, modifier le fichier `CMakeLists.txt` pour inclure les fichiers ajoutés au projet pour ce TME.

Note

Rappel la méthode pour organiser votre code et compiler est décrite dans procédure de compilation.

Question 1

Implanter les classes `BoxShape`, `LineShape` et `TermShape`, y compris leurs méthodes `::toXml()` et `::fromXml()`.

Question 2

Implanter la classe `Symbol`. La déclaration de cette classe vous est gracieusement fournie: Symbol.h

Question 3

Maintenant que la classe `Cell` dispose d'un `Symbol` pour la représenter (i.e. une icône), nous pouvons implanter complètement la méthode `Instance::setPosition()`.

Cette méthode affecte, bien sûr, l'attribut `position_` d'une `Instance`, mais elle va aussi positionner ses connecteurs (`Term`). La position du connecteur d'une `Instance` est la position de ce connecteur (en tant que `TermShape`) *dans le symbole* associé au modèle (`Cell`) de l'instance, translaté de la position de l'instance.

Note

Tout cela revient à dire que l'on trace le symbole à la position de l'`Instance`. L'opération mathématique simple qui est effectuée est un changement de repère.

Note

Un *terminal* a deux positions:

1. Sa position dans le symbole/icône associée au modèle (`Cell`), qui est donnée par une `Shape`, la `TermShape`.
2. Sa position dans l'`Instance` (du modèle), placé à un certain endroit. Et ici, c'est la position du `Term` appartenant à l'`Instance`.