The Enlightenment Foundation Libraries

Prise en main de Edje

Nicolas Aguirre
aguirre.nicolas@gmail.com

5 mai 2011

Table des matières

1 Avant Propos

Le but de ce tutorial est de survoler au travers d'un exemple pratique toutes les fonctionnalités de Edje.

J'espére qu'il vous permettra également de vous faire comprendre comment Edje peut vous aider dans le développement de vos interfaces graphiques.

De considerer cette technologie comme l'un des outils les plus puissant des EFL plutôt que comme votre plus grand cauchemar.

Comment, en séparant la logique et le code d'une part et l'interface d'une autre, vos interfaces graphiques peuvent gagner en flexibilité.

Comme exemple concret, permettant d'illustrer cette présentation, j'ai choisis le développement d'un interface (tactile) tres simple. Voici a quoi ressemblera l'interface a la fin de ce tutoriel :

2 Introduction

Edje est une des briques de base des EFL. Elle vous permet de décrire une interface graphique sans écrire une seule ligne de C. Ce qui permet,de facto, de réaliser une des choses les plus complexes lors du développement d'un programme avec une interface utilisateur : la séparation de l'interface et du code. Cette séparation est importante à plus d'un titre. Elle permet d'une part d'avoir la logique du programme et la gestion des données d'un côté et l'interface utilsateur de l'autre. Elle permet donc d'avoir deux équipes distincte qui travaille sur le projet, les graphistes, designer, ergonomes et les développeurs.

Parler de "Edje", c'est employer un therme générique pour 3 concepts différents :
– Le format de description, le format EDC, pour Edje Data Collection ;
– Le fichier binaire EDJ, résultante compilée de toutes les ressources décrites dans le fichier EDC ;
– La bibliotheque de fonctions libedje.so, permettant de manipuler les objets décrits dans le EDC au niveau de evas.
Le schémas ci dessus montre a quel moment ces trois concepts sont utilisés lors de la création d'une application utilisant Edje :

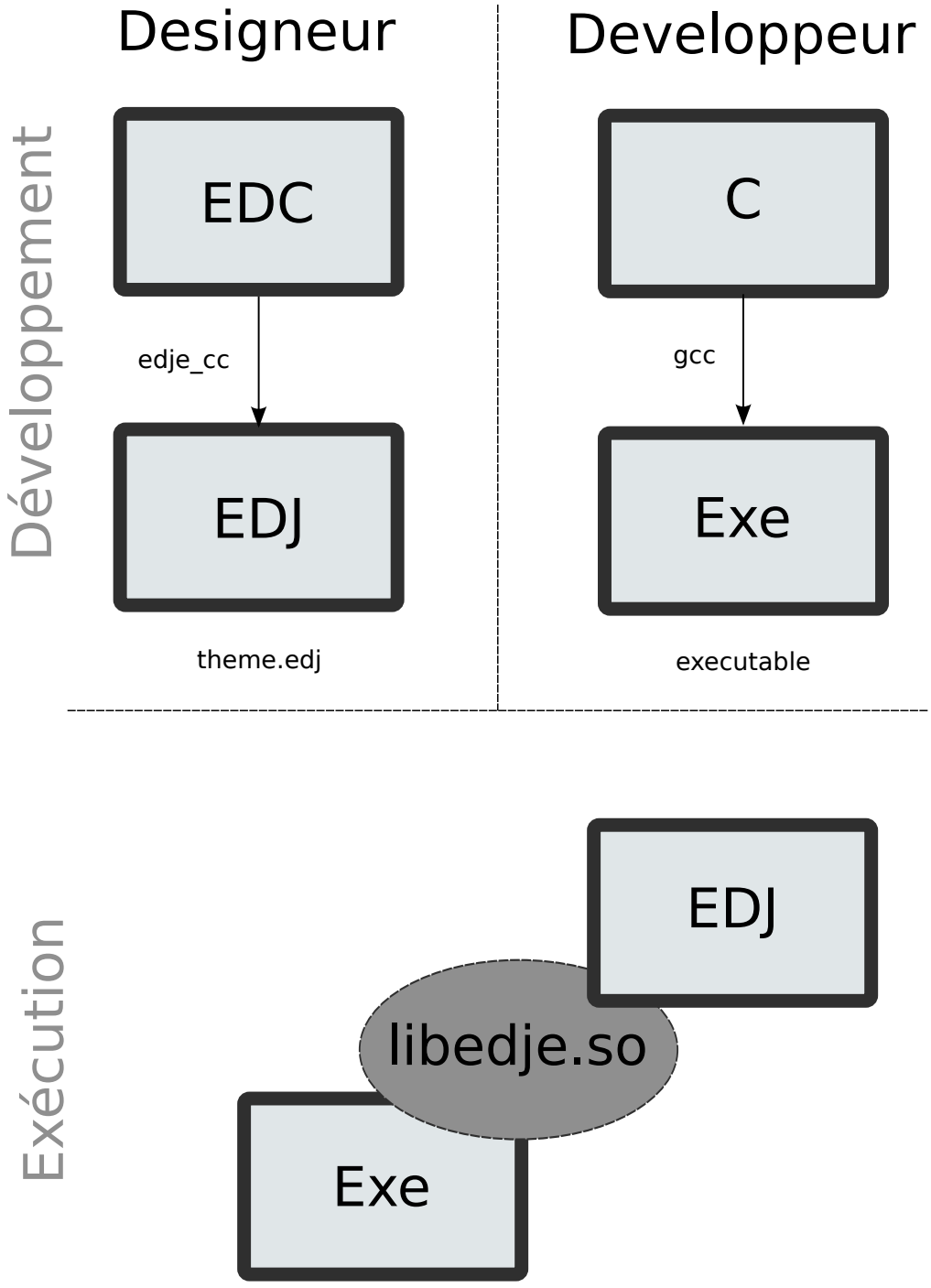

Figure 1: Edje Workflow

3 Les Bases

Dans ce chapitres, nous allons voir les bases du language de description EDC.

Une fichier EDC (Edje Data Collection) minimal ressemble a ceci : (fichier tut01/tut01.edc)

```
collections {
  group {
     name: "interface";
     parts {
        /* Rectangle Rouge */
        part {
           name: "Rectangle";
           type: RECT;
           description {
              state: "default" 0.0;
              color: 255 0 0 255;
           }
        }
     }
  }
}
```

Nous pouvons voir dans cette exemple le mot clef “collection” qui comme son nom l’indique est un ensemble de “groupes”. Dans cette exemple nous avons un seul groupe, nommée “interface”. Un groupe est lui meme un ensemble, et représente un objet, qui pourra être manipulé sur le canvas graphique plus tard dans notre programme ou réutilisé dans le fichier edc. Un Group contient des “parts” qui sont les primitives que sais manipuler Evas. Voici une liste exhaustive des “parts” que nous pouvons utiliser :
– Les rectangles : RECT ;
– Les images : IMAGE ;
– Les textes : TEXT ;
– Les blocs de texte : TEXTBLOCK ;
– Les containers : SWALLOW ;
– Les groupes : GROUP ;
– Les boites : BOX ;
– Les tables : TABLE ;
– Les objects externes : EXTERNAL ;
Chaque type fera l’objet d’une étude plus approfondie dans la suite de ce tutoriel.

Dans notre exemple nous décrivons donc un Rectangle rouge, rien de bien original. Nous allons maintenant compiler ce fichier edc en un fichier binaire EDJ. :

```
edje_cc tut01.edc
```

Si tout c’est bien passé, nous devrions trouver un fichier tut01.edj dans notre répertoire. Comme nous l’avons vu un peu plus haut. Ce fichier edj doit être chargé par notre programme pour pouvoir être affiché. Dans un premier temps nous allons donc utiliser un outil tres pratique

Figure 2: Interface edje à la fin de ce tutoriel

proposé par edje : edje_player.

```
edje_player tut01.edc
```

Et voici le résultat : Un Rectangle Rouge affiché a l'écran ! Emotionnelement intense. Que ceux qui n'ont pas la cher de poule a ce moment précis arrêtent tout de suite la lecture. Quand aux autres, vous pouvez trouver ci-dessous une capture d'écran de l'interface que nous allons développer dans la suite de ce tutoriel. j'ai choisis le développement d'un interface (tactile) simple, qui nous permettra d'apprehender les différents concept de Edje par la pratique.

Voici le résultat final :

4 Les Images

4.1 Les parts de type IMAGE

Les explication de cette section portent sur le fichier tut02/tut02.edc.

En partant du premier exemple, nous allons ajouter un fichier qui sera le fond de notre interface.

Edje supporte un large type d'images, celles supportées par Evas, PNG, JPEG, TIFF, BMP, ... Pour décrire une image, il faut créer un part de type "IMAGE". Et dire a edje quelle image insérer :

```
part {
  name: "Fond";
  type: IMAGE;
  description {
    state: "default" 0.0;
    image.normal: "bg.jpg";
  }
}
```

4.2 la directive images

Comme nous avons vu précédement, le fichier EDJ généré contient toutes les ressources de notre interface, images incluses. Si nous compilons avec edje_cc ce fichier, "bg.jpg" ne sera pas trouvé dans nos ressources. Il faut ajouter cette images dans la collection. Ceci est réalisé par l'ajout de cette directive :

```
  images {
     image: "bg.jpg" COMP;
  }
```

A quoi correspond "COMP". Edje ajoute les images dans le fichier binaire EDJ, et nous pouvons lui dire de compresser ou non cette image. Plusieurs type de compressions sont supportés :
– RAW : Uncompressed.
– COMP : compression sans perte, comparable au format PNG.
– LOSSY [0-100] : comression avec perte avec une qualité pouvant allé de 0 à 100, comparable au format JPEG
– USER : l'image n'est pas intégrée au fichier edje, mais est lue depuis le disque.
Attention cependant, si vous utilisez la balise RAW, a la taille finale de votre fichier binaire. Pour une image de 800x600 en 32bits de couleurs (RGBA), la taille embarqué dans le fichier binaire sera : 800x600x4 = 1.8MO !

4.3 edje_cc et les images

Le compilateur edje cherche les images dans les repertoires relativement au repertoire ou il est executé. Nous avons la possibilité de donner l'emplacement relative dans nos fichiers edc. Par exemple :

```
   images {
      image: "images\bg.jpg" COMP;
   }
```

Ceci peut tres vite devenir rebarbatif et long a écrire, nous pouvons donc ajouter les répertoires qui contiennent nos images en utilisant l'argument "-id" (image directory) de edje_cc

```
edje_cc -id images -id images\icons file.edc
```

Pour faciliter la compilation, vous trouverez dans le répertoire de ce tutorial, un script shell qui permet de compiler tous les fichiers edc (et C dans la suite) nommé build.sh

```
.\build.sh #compile l'intégralite des exemples.
.\build.sh #tut02 compile l'exemple contenu dans le repertoire tut02
```

Les fichiers binaires sont quand a eux générés dans le répertoire build.

4.4 Les motifs

Edje permet également l'affichage de motifs a l'écran en répétant une images. Le fichier tut03/tut03.edc montre comment utiliser une image motif avec la balise fill

```
 size {
   relative: 0.0 0.0;
   offset: 20 20;
 }
```

Dans notre cas l'image a une taille de 20x20px nous voulons qu'elle soit répétée sur l'axe des X et des Y.

Il y a plusieurs autres options permettant de repeter les motifs, je vous laisse les découvrir par vous même, tout est décris dans la [documentation de edje]

5 Le Texte

5.1 description d'une icone

Regardons le code du fichier tut04/tut04.edc plus en détails. Un nouveau groupe a été ajouté nommée "icon". Il contient une image "icon.png" et un nouveau part de type "TEXT".

```
part {
  name: "text";
  type: TEXT;
  description {
    state: "default" 0.0;
    color: 0 0 0 255;
    text {
      font: "Sans";
      size: 12;
      text: "Description";
    }
  }
}
```

Les différentes options parlent d'elle méme :
– Couleur du texte : Noir ;
– Fonte utilisée "Sans" ;
– Taille de la fonte : 12 ;
– Texte a afficher "Description" ;
Regardons a quoi ressemble notre exemple avec edje_player. Attention dans cet exemple nous avons deux groupes. Nous devons donc spécifier a ejde_player quel groupe nous voulons visualiser, et nous allons également lui dire de changer la couleur de fond.

```
edje_player -c=255,255,255,255 -g icon tut04.edj
```

Nous sommes loins du résultat escompté ! Par défaut tous les parts sont centrés au centre de l'écran et occupe tout le taille du groupe. Nous devons décrire comment les objets sont placés les uns par rapport aux autres. C'est l'objet des tutoriaux 5 à 8.

6 Placement des objets

6.1 Proportions

L'icône se doit d'être carré, c'est le cas de toutes les icônes en informatique non ? Pour cela Edje propose la balise aspect et aspect preference. Ces deux balises sont liés. Regardons a quoi ca ressemble pour un définir un part carré :

```
  aspect: 1.0 1.0;
  aspect_preference: BOTH;
```

aspect prends deux flottants comme paramétres, min et max. Dans un cas normal, les dimensions de l'objet ne sont pas liés, en utilisant le paramétre aspect, on force edje a garder un ration entre la largeur et la hauteur de notre part. Dans le cas 1.0 1.0 l'icône aura donc meme hauteur et largeur. aspect_preference, lui donne la direction on veux que ce ration s'applique. Rien de mieux pour comprendre qu'un exemple. regardez les fichier tut05.1.edj, tut05.2.edj et tut05.3.edj pour visualiser les effets de aspect et aspect_preference

6.2 Positions relatives et offsets

Edje permet de positionner les objets les uns par rapport aux autres de deux façons. Positionnement relatifs et positionnement absolu.

Le positionnement relatif est en pourcent (valeur ramenée a un floatant entre 0.0 et 1.0) alors que les positions absolu sont en pixels.

Les balises rel1 et rel2 permettent de donner la position d'un objet par rapport à un autre. Par défaut la position est celle par rapport au groupe. Le positionnement relatif est donnée avec la balise rel1.relative et rel2.relative alors que le positionnement absolu est donnée par rel1.offset et rel2.offset.

Dans cet exemple le positionnement est relatif au groupes, mais nous pouvons spécifier un positonnement par rapport à un autre part en utilisant la balise :

```
rel1.to: "autre_part1";
rel1.to: "autre_part2";
```

Ou encore relativement à un autre part mais uniquement sur un l'axe X ou Y :

```
rel1.to_x: "autre_part1";
rel1.to_y: "autre_part2";
```

Pour l'exemple du fichier tut06.edc, j'ai choisis depositionner le texte en bas du groupe. Le texte pouvant être dynamique, comme nous allons voir plus loin, nous pouvons demander a Edje de calculer sa taille, c'est le role de

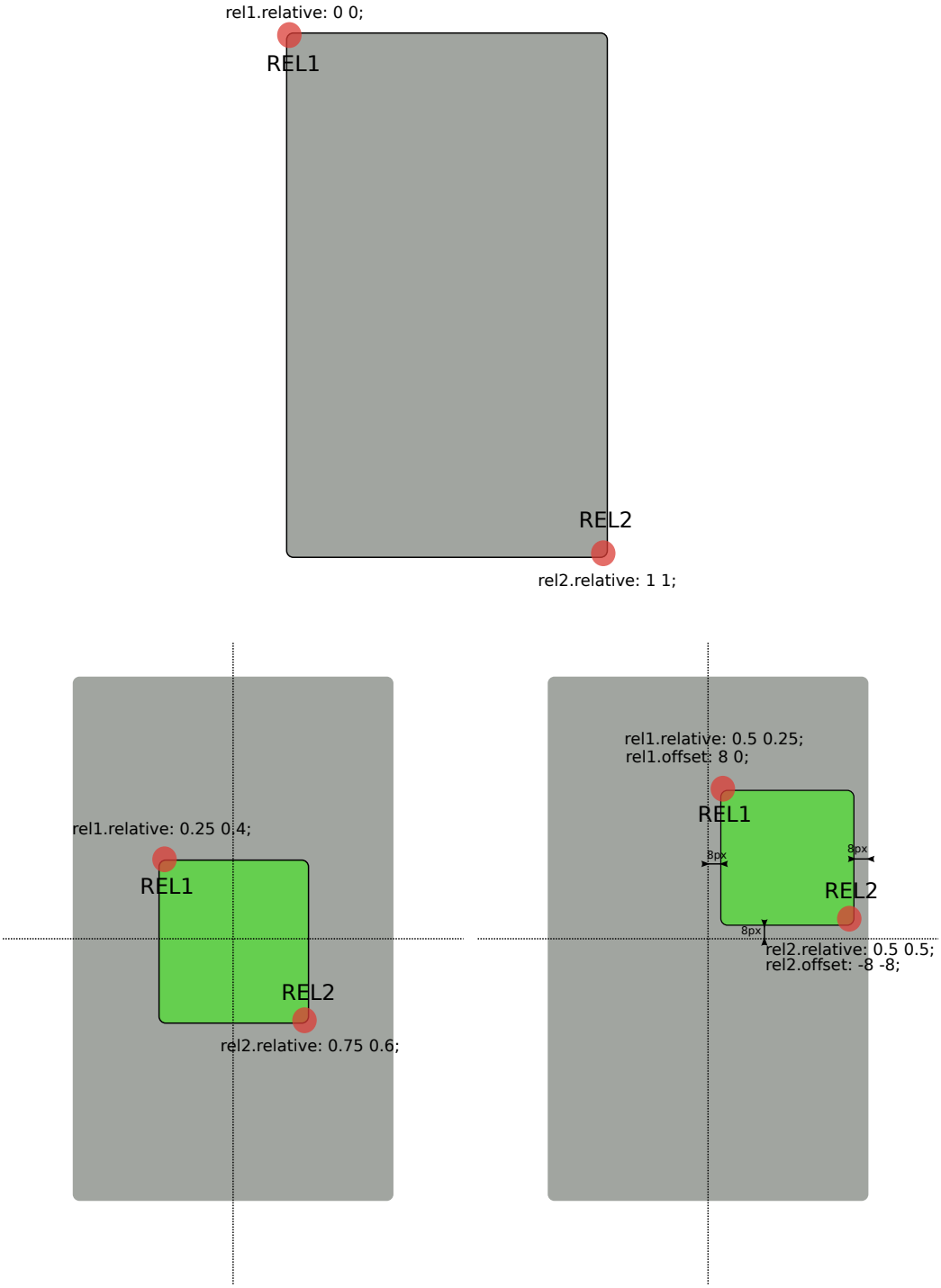

Figure 3: Positionnement relatif et absolu

```
min:: 1 1;
```

Edje vas calculer la hauteur et la largeur du texte en fonction de la police, et le part aura donc cette taile. Il nous reste plus qu'ensuite a positionner l'icone au dessus de ce texte (balise rel2.to : ïtexte") On ajoute également un bordure de 8 pixels autour de l'icone pour plus de lisibilité.

```
  rel1.relative: 0 0;
  rel1.offset: 8 8;
  rel2.relative: 1 0;
  rel2.offset: -7 -7;
  rel2.to: "text";
```

Vous verrez également une telle description sous cette forme :

```
  rel1 {
    relative: 0 0;
    offset: 8 8;
  }
  rel2 {
    relative: 1 0;
    offset: -7 -7;
    to: "text";
  }
```

Ces deux notations sont entiérement identiques.

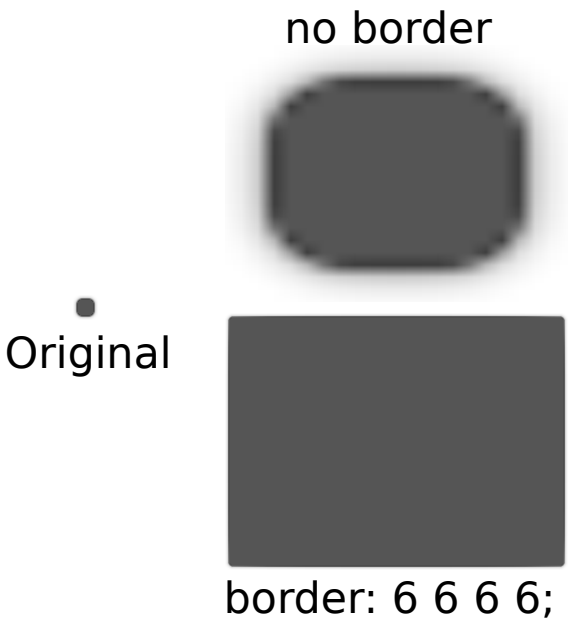

Figure 4: Avec et sans bordure

7 Images bordurées

Nous arrivons ici a une des fonctionalités que je préfére dans edje ! les bordures (borders en anglais). Ca tient en une ligne, mais ca rends de grands services.

```
image.border: 8 8 8 8;
```

C'est spécifique aux parts de type "IMAGES". Cette ligne impose a Edje de ne redimensionner toutes les parties de l'image, sauf les bordures. Le cas se présente souvent lorsque on utilise une image avec des coins arrondis par exemples, comme c'est le cas avec le contour du texte.

les paramétres de cette balise sont les suivants left, right, top, width et les parmétres sont donnés en pixels.

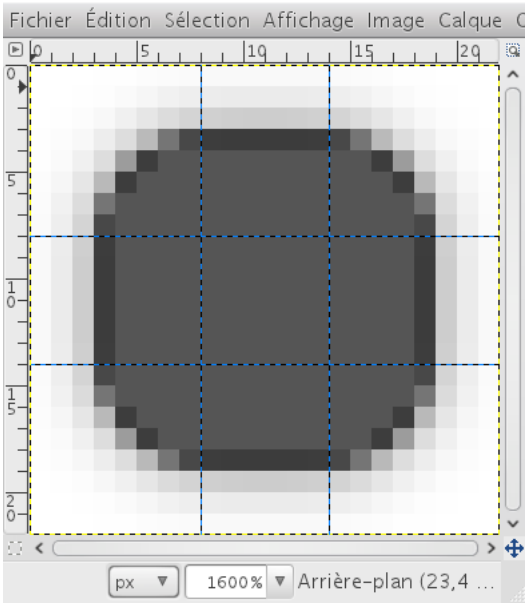

Figure 5: Gimp et bordures

8 Les programmes

8.1 Evenements souris

Pour nous simplifier la vie par la suite, j'ai ajouté dans le fichier tut09 un part que j'ai nommé "events", qui se place au dessus de tous les autres parts. L'ordre des parts est fonction de la position de la description dans le groupe. C'est a dire que le part le plus bas dans le fichier est le plus haut dans l'interface.

Ce part "events" semble ne servir a rien, puisqu'il a une transparence à 0, mais il va nous rendre de grand service. Dans la suite il va nous permettre de capturer tous les évenements souris de notre icone.

La balise visible indique a Edje si le part doit être affiché ou pas à l'écran quelque soit la position, la taille ou toute autre propriétée du part. Par défaut les parts sont visibles, mais parfois, nous pouvons faire en sorte qu'un part devienne invisible. Nous voudrions par exemple a moment donné que les évenements souris ne soient pas capturés, Il nous suffirait dont de mettre cette propriété a 0 pour que plus aucun evenements ne soient traités pour ce part.

8.2 Les Programmes

Un programme est une action que Edje va réaliser. Cela peut être de différents types, un événement souris sur notre groupe, une signal provenant du programme qui manipule notre groupe, un signal envoyé par un autre programme, un signal envoyé par edje.

Le fichier tut10.edc présente un exemple de programme qui réagit a un événement bouton 1 de la souris enfoncé :

```
program {
  name: "mouse_down";
  signal: "mouse,down,1";
  source: "events";
  action: STATE_SET "down" 0.0;
  target: "icon";
}
```

A quoi correspondent les différentes balises :

- name : C'est le nom du programme
- signal : c'est la chaine de caractére qui défiini le signal reçu, dans ce cas c'est un signal envoyé par edje lorsque le bouton 1 de la souris est enfoncé sur le part défini dans la balise "source".
- source le part qui est responsable du signal. La souris doit être enfoncé sur ce part. On voit ici que ce part nous sert a quelque chose ! Si il n'existait pas nous aurions du dupliquer ce programme pour réaliser l'action sur le part icon, texte et texte_bg !
- action : L'action a réalisée lorsque le signal est reçu. Ici nous demandons a Edje de changer

l'état du part défini dans la balise “target” à ”down” 0.0

Un part avoir plusieurs états, si il existe plusieurs description de celui-ci C'est le role de la balise “description” que nous avons rencontrée mais pas encore expliquée.

Tous les paramétres d'un part que nous avons vu jusqu'à présente : couleur, alignement, la fonte ou la taille pour un texte, ... peuvent avoir différentes valeurs d'une description a une autre.

```
state: "default" 0.0;
```

Ce balise permet de donner un nom a notre état, ainsi qu'une flotant. Revenons a notre exemple :

```
  part {
    name: "icon";
    type: IMAGE;
    mouse_events: 0;
    description {
      state: "default" 0.0;
      aspect: 1.0 1.0;
      aspect_preference: BOTH;
      image.normal: "icon.png";
      rel1.relative: 0 0;
      rel1.offset: 8 8;
      rel2.relative: 1 0;
      rel2.offset: -7 -7;
      rel2.to_y: "text";
      align: 0.5 0.5;
    }
    description {
      state: "down" 0.0;
      inherit: "default" 0.0;
      color: 255 255 255 128;
    }
  }
```

Ici nous avons deux états : ðefault0.0 et ðown0.0. L'état down, hérite de toutes les propriétés de default sauf pour la couleur, ou nous changeons l'alpha.

Couplé au programme précédent, cela aura pour effet de changer la transparence lors d'un clic sur l'icone.

Nous pouvons vérifier ceci avec edje_player :

```
edje\_player tut10.edj -g icon
```

Effectivement la couleur change, mais lorsque nous relachons le clic, notre icone reste dans cet état. Nous devons ajouter le programme équivalent pour le mouse up. Le fichier tut11.edc montre ceci.

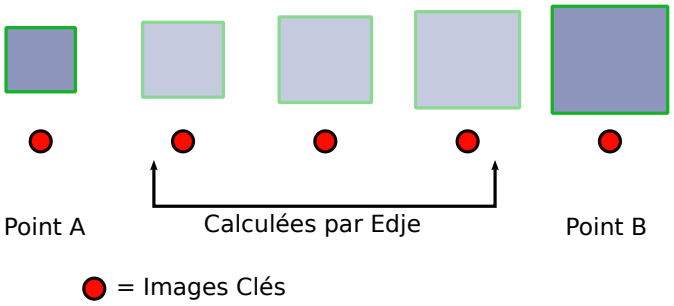

Figure 6: Création des images clés

8.3 Les animations

Comme nous pouvons voir, la transition est abrupte. Pour réaliser un effet plus relaxant, nous allons ajouter une animation entre deux états de nos parts.

```
transition: LINEAR 1.0;
```

La balise transition, permet comme son nom l'indique d'indiquer a Edje qu'il doit changer l'état, que ce changement doit durer 1 seconde. Entre nos deux états Edje va interpoler la position et la couleur de notre objet et cet interpolation sera de type linéaire. Le framerate par défaut de Edje et de 60 images par secondes, il va donc calculer 60 image clés différentes pour notre part.

9 Intégrer un objet Edje dans un programme C

Nous avons décris des objets, nous avons généré le binaire et nous avons testé le tout avec edje_player. Mais notre but est de creer un interface, un creer un executable.

Dans ce chapitre, nous allons voir comment intégrer les groupes que nous avons créés dans une application Elementary.

Pourquoi avec ELementary et pas avec Edje ? Tout simplement parce que c'est plus simple, et concis avec elementary, que Elementary utilise Edje pour nous et que je suis plutot de nature fénéante !

Tout d'abord créons un programme tres simple qui cree une fenêtre avec un titre, et qui quitte lorsque on clique sur la croix. (fichier tut13.c)

```
/* gcc -o test tut13.c `pkg-config elementary --cflags --libs` */

#include <Elementary.h>

static void
_win_del(void *data, Evas_Object *obj, void *event_info)
{
  elm_exit();
}

int main(int argc, char **argv)
{
  Evas_Object *win;

  elm_init(argc, argv);

  win = elm_win_add(NULL, "tuto", ELM_WIN_BASIC);
  elm_win_title_set(win, "Edje Tutorial");
  evas_object_smart_callback_add(win, "delete,request", _win_del, NULL);

  evas_object_resize(win, 800, 480);

  evas_object_show(win);

  elm_run();
  elm_shutdown();
}
```

Maintenant ajoutons notre groupe “interface” que nous avons creer au tout début de ce tutorial dans notre fenêtre :

```
layout = elm_layout_add(win);
elm_layout_file_set(layout, "tut14.edj", "interface");
evas_object_show(layout);
```

Et faisons en sorte que celui-ci soit redimensionné lorsqu'on redimensionne la fenêtre :

```
   elm_win_resize_object_add(win, layout);
```

Nous rencontrons ici un des concepts les plus importants de Evas. Les Smart Objects. elm_layout_add utilise edje en interne, et cree un Smart Objet qui pourra ensuite être manipulé par les primitives de evas, move, resize, ... Nous démandons ensuite a ce que cet objet utilise le groupe ïnterface# du fichier tut14.edj. Et nous affichons cet objet a l'écran avec evas_object_show. elm_win_resize_object_add va faire en sorte que evas redimensionne notre objet a la taille de la fenêtre.

Recompilons notre programme et testons :

```
./tut14
```

Nous avons écris notre premier programme intégrant une objet edje !

Rajoutons maintenant le groupe icon : (fichier tut15.c)

```
   icon = elm_layout_add(win);
   elm_layout_file_set(icon, "tut14.edj", "icon");
   evas_object_resize(icon, 256, 256);
   evas_object_move(icon, 64, 64);
   evas_object_show(icon);
```

Notre Smart Object est cree comme précédement, et cette fois-ci nous allons le manipuler nous meme, en luis donnant une taille de 256x256px et en le placant en haut à gauche x=64px et y=64px.

Notre icone posséde un texte (part ïext¨, que nous allons changer grâce à :

```
   elm_layout_text_set(icon, "text", "Vers l'infini et au dela!");
```

Elementary my dear Watson !

9.1 La multiplication des icones

Notre exemple final contient plus qu'une seule icone. Machinalement, nous pourrions dupliquer a la fois dans Edje et dans le code C les blocs permettant de changer le nom de l'icône et le texte. Mais cela serait d'une part tres long, source d'erreur mais surtout tres rébarbatif. Et comme je le disais précédement je suis fénéant. Nous allons donc utiliser ce que Edje nous propose pour nous faciliter la vie.

Les descriptions Edje sont préprocéssées. Comme les fichier .c et .h par le programme cpp. Le rôle de celui-ci est de remplacer les toute occurence a une macro dans le texte par le morceau de code défini dans la macro. Lorsqu'il recontre un #include il copie l'intégratlite de ce fichier a cette position.

Comme dans un programme C, nous allons donc creer une macro ICON pour nous faicilité la vie.

Notre macro prendra comme paramétre d'entrée le nom de l'icone et le nom du fichier icon à utiliser.

Nous pouvons ensuite ajouter nos différents groupes comme ceci :

```
   ICON("video.png", "video");
   ICON("meteo.png", "meteo");
   ICON("calc.png", "calc");
   ICON("music.png", "music");
   ICON("calendar.png", "calendar");
   ICON("mail.png", "mail");
   ICON("rss.png", "rss");
```

10 les tables

Maintenant que nos groupes ont étés ajoutés a notre fichier edje, nous pourrions afficher chaque icones à l'écran, les positionner les unes par rapport aux autres avec evas_object_move. Cela serait fastidieux. D'autre part si nous voulons changer le layout, il faudrait fournir une nouvelle version du programme et l'avantage de edje c'est de pouvoir faire ca pour nous.

Nous allons donc utiliser un nouveau type de parts, apres les TEXT, les RECT et les IMAGES : les TABLE. Voici comment positionner les icones que nous avons créés précédement :

```
part {
  name : "table_description";
  type : TABLE;
  description {
    state : "default" 0.0;
    fixed: 1 1;
  }
  table {
    items {
      item {
        type: GROUP;
        source: "video";
        align: -1 -1;
        position: 0 0;
      }
      item {
        type: GROUP;
        source: "meteo";
        align: -1 -1;
        position: 0 1;
      }
      item {
        type: GROUP;
        source: "calc";
        position: 0 2;
        align: -1 -1;
      }
      item {
        type: GROUP;
        source: "music";
        position: 1 0;
        align: -1 -1;
      }
      item {
        type: GROUP;
        source: "calendar";
        position: 1 1;
        align: -1 -1;
      }
      item {
```

```
          type: GROUP;
          source: "mail";
          position: 1 2;
          align: -1 -1;
        }
        item {
          type: GROUP;
          source: "rss";
          position: 2 0;
          align: -1 -1;
        }
      }
    }
  }
```

Chaque items de la table a une balise position, qui donne sa postion dans la table. La petite subtilité ici est l'alignement de chaque items. On demande à Edje de desactiver l'alignement (valeur -1) et donc de s'adapter a la taille de la celulle.

11 Icônes et multi-résolutions

Nous allons étudier ici une autre fonctionalités interessante de edje, les “set d’icones”. Nous voulons que notre applications s’affiche quelque soit sa taille et nous somme aidé par l’utilisation de la balise border dans les images. Imaginons que nous voulons intégrer des icônes qui s’affichent aussi bien en 512x512 que en 16x16. Nous avons plusieurs possibilité. La plus simple est d’utiliser une icone source de 512x512. Dans le cas d’un affichage en 16x16, celle-ci est réduite par Evas et Edje avec la perte d’informations que cela améne. Une autre solution serait d’utiliser une image de type SVG, qui s’adaptera en fonction de la résolution. Mais le plus souvent ce type de fichier est tres lourd a gérer, en tout cas bien plus qu’un fichier bitmap. Pour résoudre ce probléme, Edje intégre les set d’icones. Nous pouvons lui indiquer quelle image utiliser en fonction de la taille. Un autre type d’utilisation de cette fonctionalité et d’afficher une image complétement différente en fonction de la taille de la zone d’affichage.

Le fichier tut18 présente cette fonctionalité. J’ai ajouté un script dans le répertoire images, qui permet a partir d’un fichier svg source de généré les fichier png dans les différentes résolutions voulues.

Lorsque nous testons avec edje_player -g calc tut18.edj, et que nous redimensionnons la fenêtre, on peut voir que Edje adapte la résolution de l’icône à la taille.

12 Les Containeurs

Nous Avons vu comment creer des groupes, et comment ajouter ces groupes via notre programme. Dans le cas du fond d'écran, c'est un layout Evas qui affiche un Objet Edje Dans le cas de l'icone, c'est un layout Edje avec pilotage Edje, puisque c'est la table qui fait l'affichage et le layout. La fonctionalité que nous allons voir, c'est la troisiéme possibilité : Un layout Edje avec pilotage par Evas : les Swallows ou containeurs. C'est un part de type SWALLOW. C'est une zone, qui comme les autres parts peut avoir différentes position, taille, ... en fonction de la description et qui peut contenir un Smart Object. C'est notre programme qui décidera quoi mettre à l'intérieur.

Voici un exemple de part de type swallow :

```
part {
  name: "table_swallow";
  type: SWALLOW;
  description {
    state: "default" 0.0;
    rel1.to: "table_bg";
    rel2.to: "table_bg";
  }
}
```

et coté programme, on cree un rectangle avec Evas et on l'ajoute dans notre part

```
rect = evas_object_rectangle_add(evas_object_evas_get(win));
evas_object_color_set(rect, 0, 255, 0, 128);
evas_object_resize(rect, 10000, 10000);
evas_object_move(rect, -5000, -5000);
evas_object_show(rect);
elm_layout_content_set(layout, "table_swallow", rect);
```

Comme on peut le voir ici, j'ai changé la position de l'objet ainsi que sa taille en donnant des valeurs absurdes, on voit bien en éxécutant tut19 que Edje drive l'affichage de notre objet.

Nous avons ajouté un rectangle mais nous pouvons ajouter n'importe que tupe de Smart Objects. Dans l'example suivant nous allons ajouter un panel a notre interface, qui affichera sous forme de texte des infos a l'écran. Et lorsque nous cliquerons sur l'icone info nous afficherons ce nouveau groupe a l'écran depuis notre programme.

13 les TEXTBLOCKS

Notre pannaux d'information affichera du texte. Tout à l'heure nous avons ajouter un nom a notre icône. Mais celui-ci avait une couleur unique. Si nous voulons ajouter des styles de texte différents, afficher le texte sur plusieurs ligne, Edje propose pour cela les TEXTBLOCKS.

Edje utilise un systeme de marqueurs pour différencier les différentes styles. Si vous connaissez les balises html, vous ne serer pas perdu ici. Un style ressemble a ceci :

```
styles {
  style { name: "textblock_style";
    base: "font=Sans font_size=16 color=#EEE wrap=word";
    tag:  "br" "\n";
    tag:  "ps" "ps";
    tag:  "hilight" "+ font=Sans:style=Bold color=#3dadff";
    tag:  "b" "+ font=Sans:style=Bold colo=#2d3e46";
    tag:  "tab" "\t";
    tag:  "h1" "+ font_size=40 color=#b5de29";
    tag:  "h2" "+ font_size=30";
    tag:  "h3" "+ font_size=30";
    tag:  "h4" "+ font_size=18";
    tag:  "rhinoceros" "+ color=#F0F";
    tag:  "link" "+ color=#00000080";
  }
}
```

Comme toujours un nom, unique auquel on pourra faire référence. Une base, qui est la valeur par défaut de notre texte, c'est a dire quand nous n'utilisons pas de balise. Et enfin les tags, qui définissent chacune des balises que nous pouvons utiliser pour ce style. J'ai défini des balises que nous retrouvons pour la plus part dans le HTML. Mais nous pouvons mettre dans le nom des tags ce que nous voulons. C'est le cas par exemple de la balise "rhinoceros". Elle change uniquement la couleur du texte, mais comme on peut le voir pour les autres balises, on peut changer la fonte du texte, sa couleur, sa taille.....

Une fois un style défini, nous pouvons déclarer notre nouveau part de type TEXTBLOCK :

```
part {
  name: "textblock";
  type: TEXTBLOCK;
  entry_mode: EDITABLE;
  source5: "anchor";
  description {
    state: "default" 0.0;

    text {
      style: "textblock_style";
    }
  }
}
```

Dans notre programme nous pouvons spécifier le texte comme tout à l'heure, mais et enrobé certaines partie avec les balises que nous avons définis dans le style.

```
elm_layout_text_set(textblock,"textblock", "<h1>What is Enlightenment?</h1><br>");
```

14 Capturer les signaux edje dans le programme

Pour avoir une interraction entre edje et notre programme, nous pouvons utiliser des programmes contenant l'action SIGNAL_EMIT. Lorsque ce programme est exécuté, Edje envoie alors ce signal. A nous de le capter et de réagir en fonction. Voici le code d'exemple qui permet de capture absolument tous les signaux emis par le groupe layout :

```
  edje = elm_layout_edje_get(table);
  edje_object_signal_callback_add(edje, "*", "*", _edje_signal_cb, NULL);
```

et le code du callback executé lorsque un signal est reçu :

```
static void
_edje_signal_cb(void *data, Evas_Object *obj, const char *emission, const char *source)
{
   printf("Emission : %s - Source : %s\n", emission, source);
}
```

Les * utilisé dans l'ajout du callback permettent de filtrer les signaux, ici nous affichons tout.

Pour le besoin de notre exemple, nous allons envoyer, lors d'un clic sur une icone, le nom de l'icone.

```
   program {                                                    \
     name: "mouse_click";
     signal: "mouse,clicked,1";
     source: "events";
     action: SIGNAL_EMIT "info,clicked" "";
   }
```

Attention ici, nous utilisons le signal mouse,clicked, qui est différent de mouse down et mouse up. Celui-ci est envoyé uniquement si on relache la souris au dessus de la source.

Dans le programme nous allons recevoir les données : emission == "info_clicked" et source == "".

Lorsque info_clicked sera reçu, nous allons alors creer notre panel, et l'afficher à la place de la table.

14.1 Les Ancres(anchors)

Les textblock peuvent en plus du texte de différents style être clickables. C'est le rôle des encres ..... blablablalbla

14.2 Transformations 3D

blablablalbalbalbal....